POLITECNICO DI TORINO

*Collegio di Ingegneria Meccanica, Aerospaziale, dell'Autoveicolo e della Produzione*

Department of Automotive Engineering

Corso di Laurea Magistrale in Ingegneria Dell'Autoveicolo

Master Thesis

# Development of tool chain for vehicle electrification

| | |
|---|---|
| Supervisor: | Candidate: |
| Dr. Stefano Carabelli | Muaaz Tariq S250960 |

Academic Year 19/20

## i. Acknowledgment

First, I thank God for bestowing upon me his countless blessings.

Thanks to my family who supported me all through these years so that I can achieve more and live up to my true potential. Although at this moment my family is not with me, but I know they will always support me and prove to be my safety net. My dad is not here to see what I have achieved but I hope if he had been alive, he would have been proud of my accomplishments.

My prof. Stefano Carabelli taught me a lot during my thesis, and I am thankful to him. His method of teaching is innovative and the interactive session I had with him during this thesis always left me with countless innovative way to find solution of the same problem.

I am thankful to Brain tech. and especially Ing. Giovani Guida for giving me the opportunity to do my internship with them. Mr. Guida is a talented person and I learned many things from him. This knowledge was not restricted to books, but it was also related to dealing with a real-world problem and how to make the best of it.

Although, I was alone here but my friends never made me feel that I am alone. They have always been there for me in times when I needed them the most. Every one of them taught me something valuable and the experiences I had with them will remain in my memory for ever and I will cherish these moments throughout my life.

Last but not the least, in this moment of happiness we can’t forget the people who have lost their family members in this pandemic. I can feel their pain and assure them that we are in this together and we will fight it together.

Thanks

## ii. 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 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.

In the second chapter we 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. "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" (Brain Technologies). 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.

# Table of Contents

# List of Figures:

# List of tables

## iii. Introduction

Automotive sector is very competitive and challenging nowadays. There are many companies which are trying hard to increase there 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 (Jabil, 2017) shows that in 2017, 68% of the automotive manufactures told that there 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:

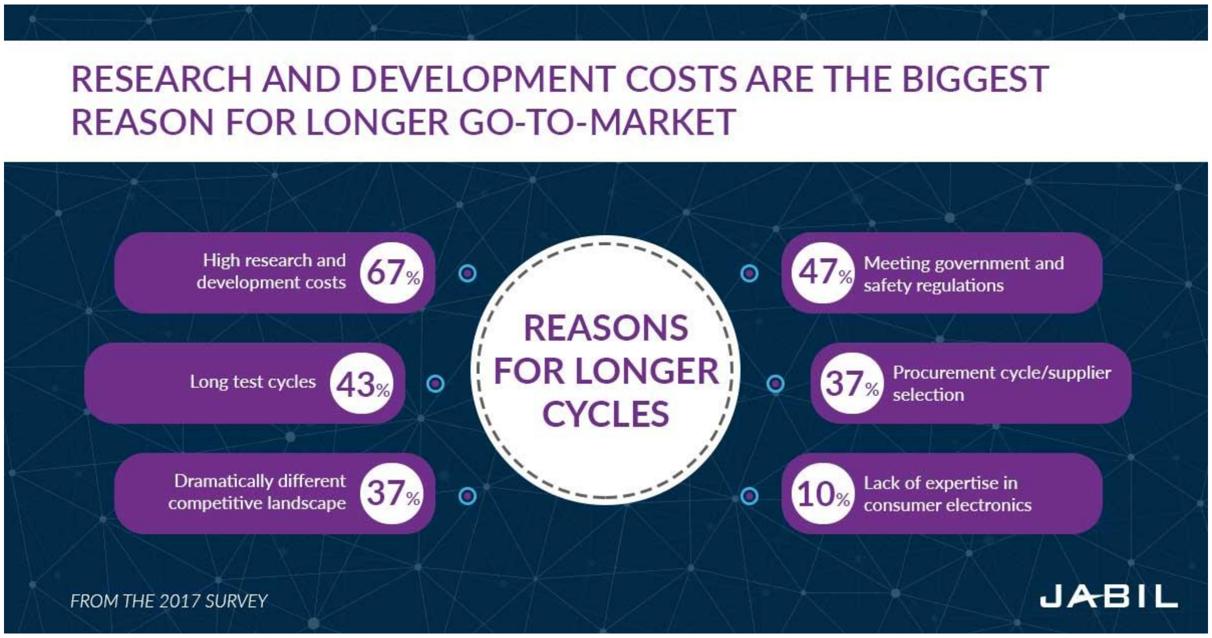


*Figure 1. Go to market statistics*

*(Jabil, 2017)*

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 manufactures 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 streamline the process 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 project.

# *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.

(Iso.org, 2018)

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).

(Instruments, 2019)

## 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”.

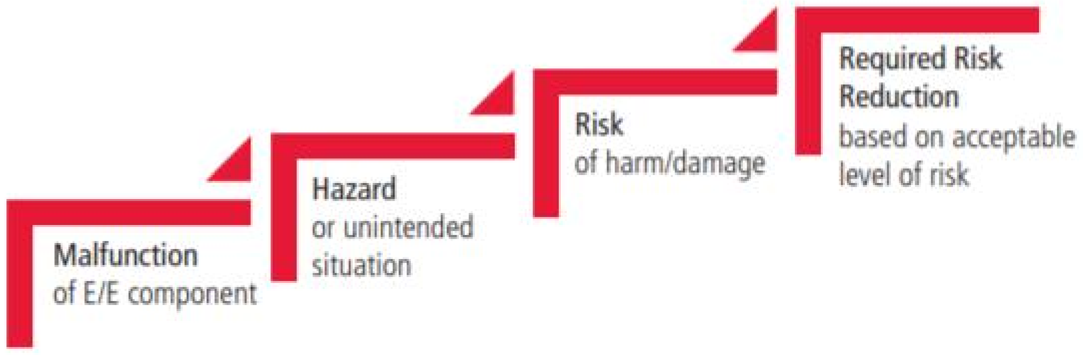


*Figure 2. Functional safety cascade (cadence, 2019)*

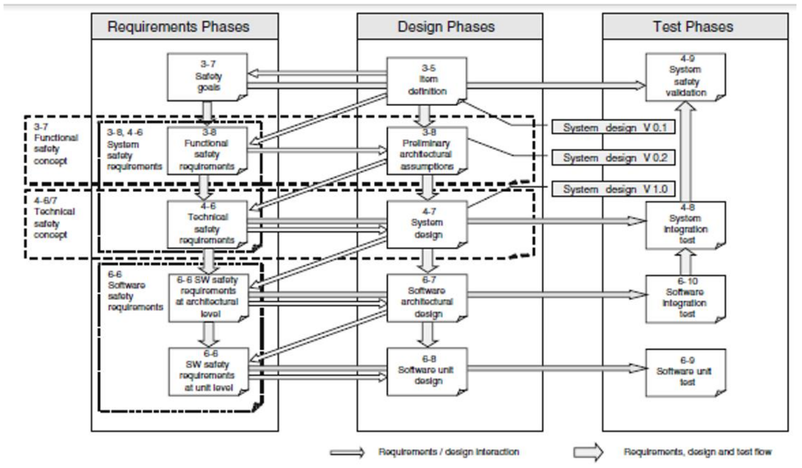

Requirements Phases
Design Phases
Test Phases
3-7 Safety goals
3-5 Item definition
4-9 System safety validation
3-7 Functional safety concept
3-8, 4-6 System safety requirements
3-8 Functional safety requirements
3-8 Preliminary architectural assumptions
System design V 0.1
System design V 0.2
System design V 1.0
4-6/7 Technical safety concept
4-6 Technical safety requirements
4-7 System design
4-8 System integration test
6-6 Software safety requirements
6-6 SW safety requirements at architectural level
6-7 Software architectural design
6-10 Software integration test
6-6 SW safety requirements at unit level
6-8 Software unit design
6-9 Software unit test
Requirements / design interaction
Requirements, design and test flow

*Figure 3. Safety lifecycle for software product development (Iso.org, 2018)*

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 must 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 must 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

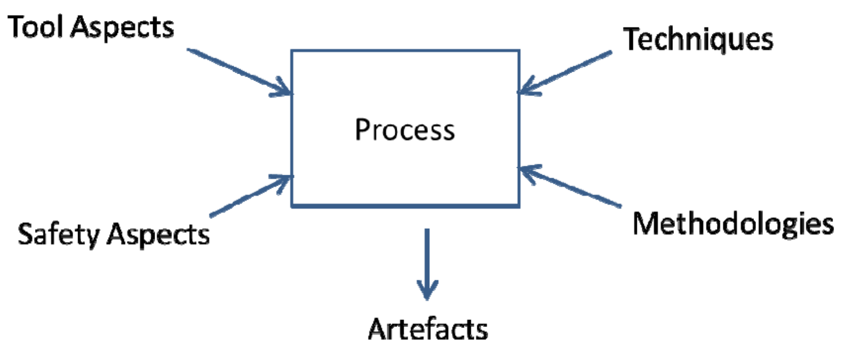


*Figure 4. Artefacts*

***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

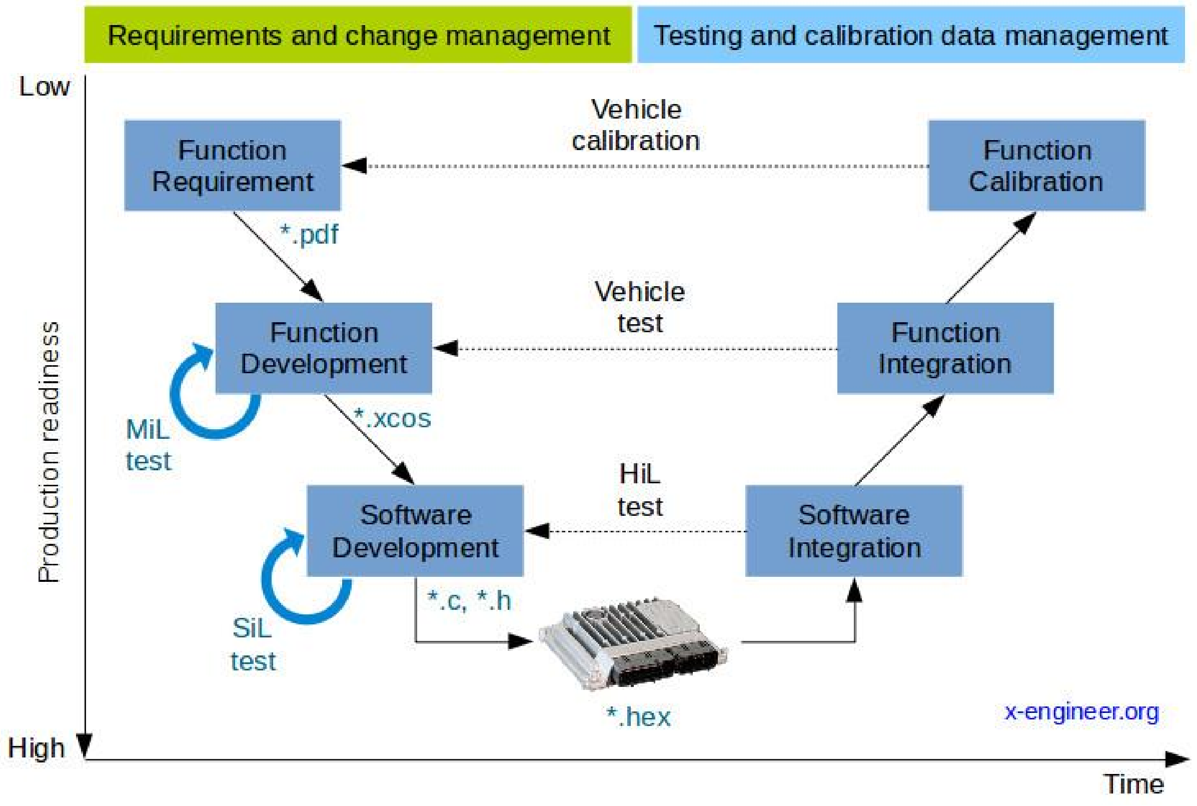


*Figure 5. V-cycle for software*

(x-engineer.org, 2019)

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 (2020) 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:

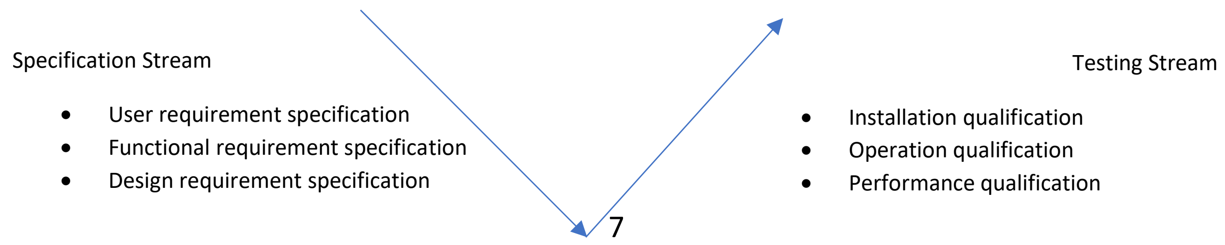
- Minimization of project risks
- Improvement and guarantee of quality
- Reduction of total cost over project and system life cycle
- Improvement of communication between stake holders

## 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.


Specification Stream
• User requirement specification
• Functional requirement specification
• Design requirement specification
Testing Stream
• Installation qualification
• Operation qualification
• Performance qualification

## 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:

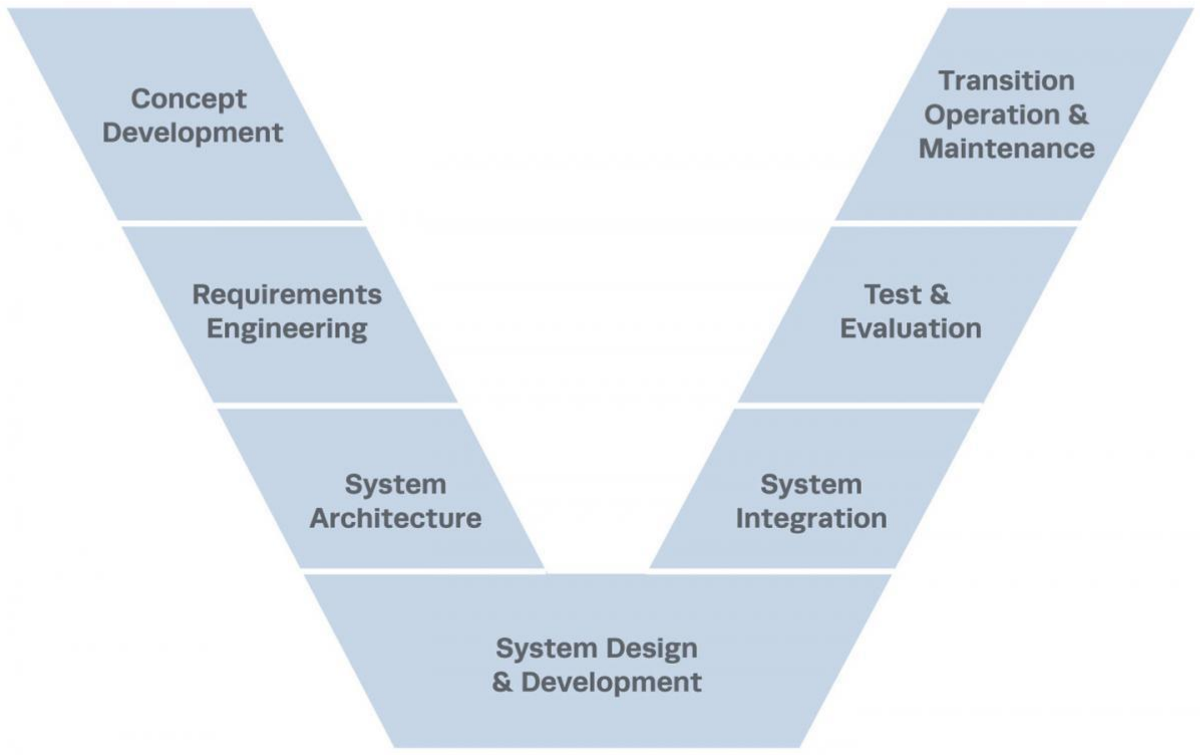


*Figure 6. Systems engineering*

(mitre.org, 2014)

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

(Jackson, 2019)

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.

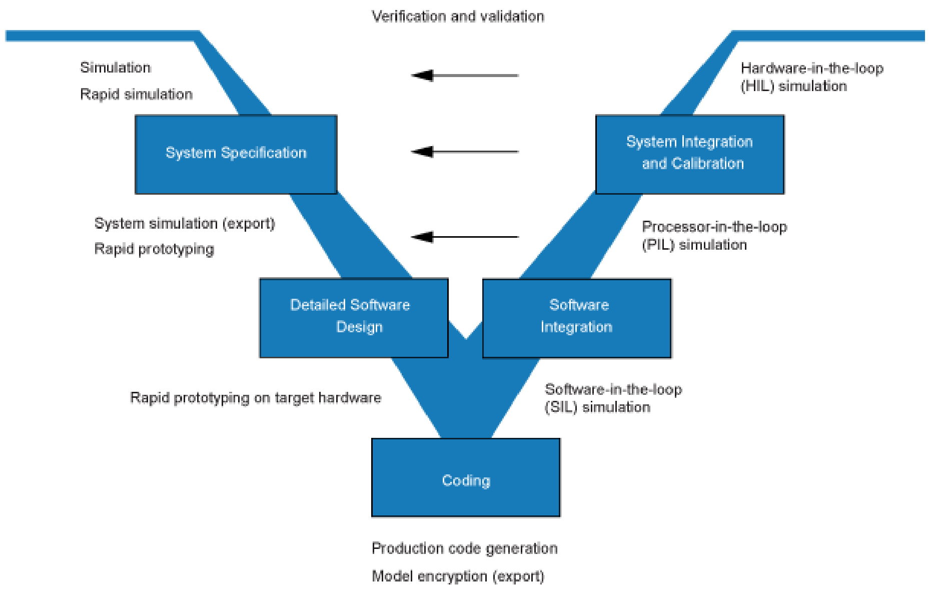

Verification and validation
Simulation
Rapid simulation
System Specification
System simulation (export)
Rapid prototyping
Detailed Software Design
Rapid prototyping on target hardware
Coding
Production code generation
Model encryption (export)
Hardware-in-the-loop (HIL) simulation
System Integration and Calibration
Processor-in-the-loop (PIL) simulation
Software Integration
Software-in-the-loop (SIL) simulation

*Figure 7. V-Model for System Development and types of Simulation (MathWorks, n.d.)*

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*** – *s*hould 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 block sets provided by the VMU and/or RCP manufacturers in order to generate and download code for their hardware.

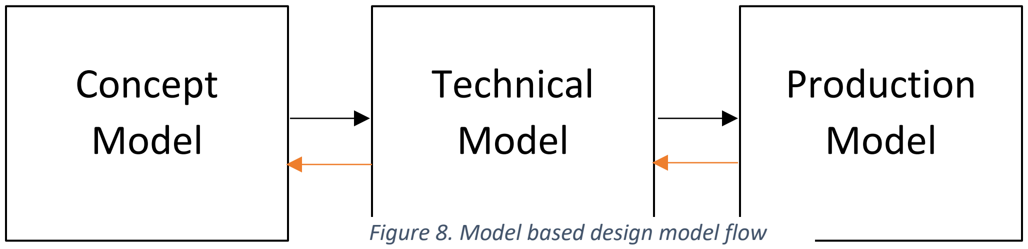


*Figure 8. Model based design model flow*

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 Guidelines[1]

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 essential.

[1] Mathworks.com

***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:

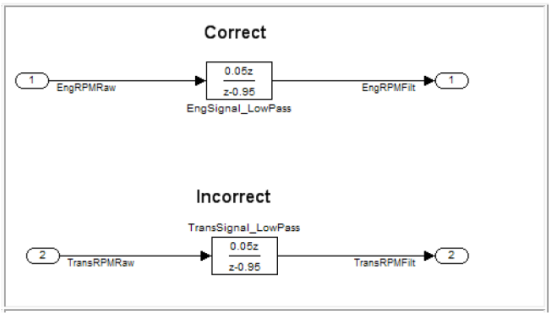
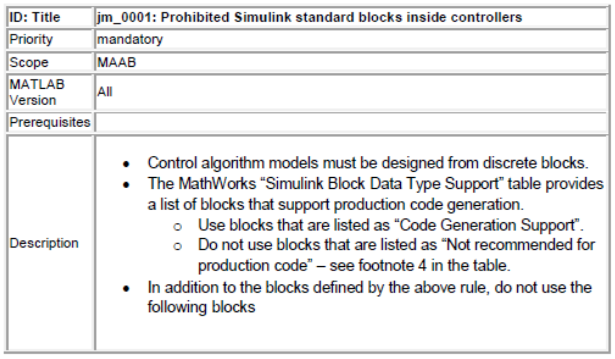


*Figure 9. MAAB guideline for Simulink modelling example (MathWorks, n.d.)*

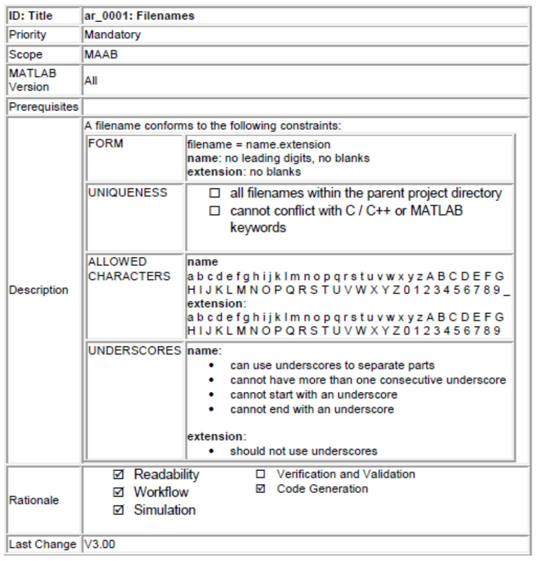
A *Mandatory* requirement that we shall apply is the block that are not allowed to be inside controllers as the figure below shows:



| ID: Title | jm_0001: Prohibited Simulink standard blocks inside controllers |
|---|---|
| Priority | mandatory |
| Scope | MAAB |
| MATLAB Version | All |
| Prerequisites | |
| Description | • Control algorithm models must be designed from discrete blocks.<br>• The MathWorks "Simulink Block Data Type Support" table provides a list of blocks that support production code generation.<br>  ○ Use blocks that are listed as "Code Generation Support".<br>  ○ Do not use blocks that are listed as "Not recommended for production code" – see footnote 4 in the table.<br>• In addition to the blocks defined by the above rule, do not use the following blocks |

*Figure 10. Prohibited blocks inside controllers (MathWorks, n.d.)*

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 | A filename conforms to the following constraints: | |
| | FORM | filename = name.extension **name**: no leading digits, no blanks **extension**: no blanks |
| | UNIQUENESS | ☐ all filenames within the parent project directory ☐ 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**: • can use underscores to separate parts • cannot have more than one consecutive underscore • cannot start with an underscore • cannot end with an underscore **extension**: • should not use underscores |
| Rationale | ☑ Readability ☑ Workflow ☑ Simulation | ☐ Verification and Validation ☑ Code Generation |
| Last Change | V3.00 | |

*Figure 11. MAAB for filenames (MathWorks, n.d.)*

For signal naming with a priority *Strongly recommended,* a signal name (MathWorks, n.d.):

- 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.

### 4.1. 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,

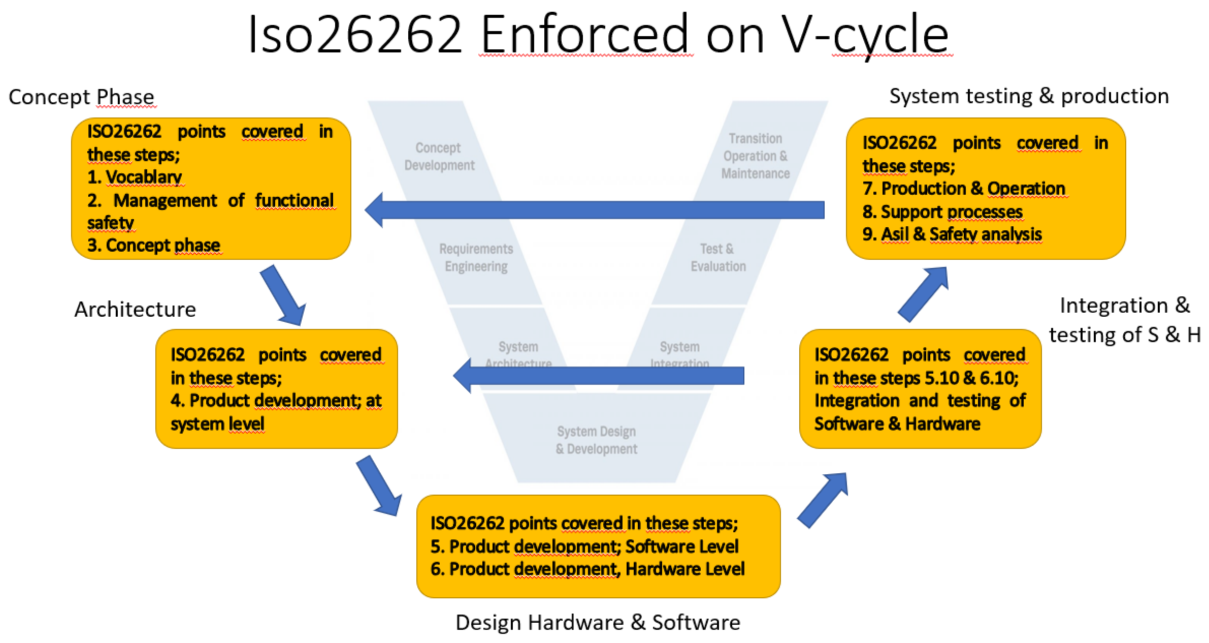

Iso26262 Enforced on V-cycle
Concept Phase
ISO26262 points covered in these steps;
1. Vocablary
2. Management of functional safety
3. Concept phase
Architecture
ISO26262 points covered in these steps;
4. Product development; at system level
ISO26262 points covered in these steps;
5. Product development; Software Level
6. Product development, Hardware Level
Design Hardware & Software
Integration & testing of S & H
ISO26262 points covered in these steps 5.10 & 6.10;
Integration and testing of Software & Hardware
System testing & production
ISO26262 points covered in these steps;
7. Production & Operation
8. Support processes
9. Asil & Safety analysis
Concept Development
Requirements Engineering
System Architecture
System Design & Development
System Integration
Test & Evaluation
Transition Operation & Maintenance

*Figure 12. Iso26262 Enforced on V-cycle*

## 4.2. 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.

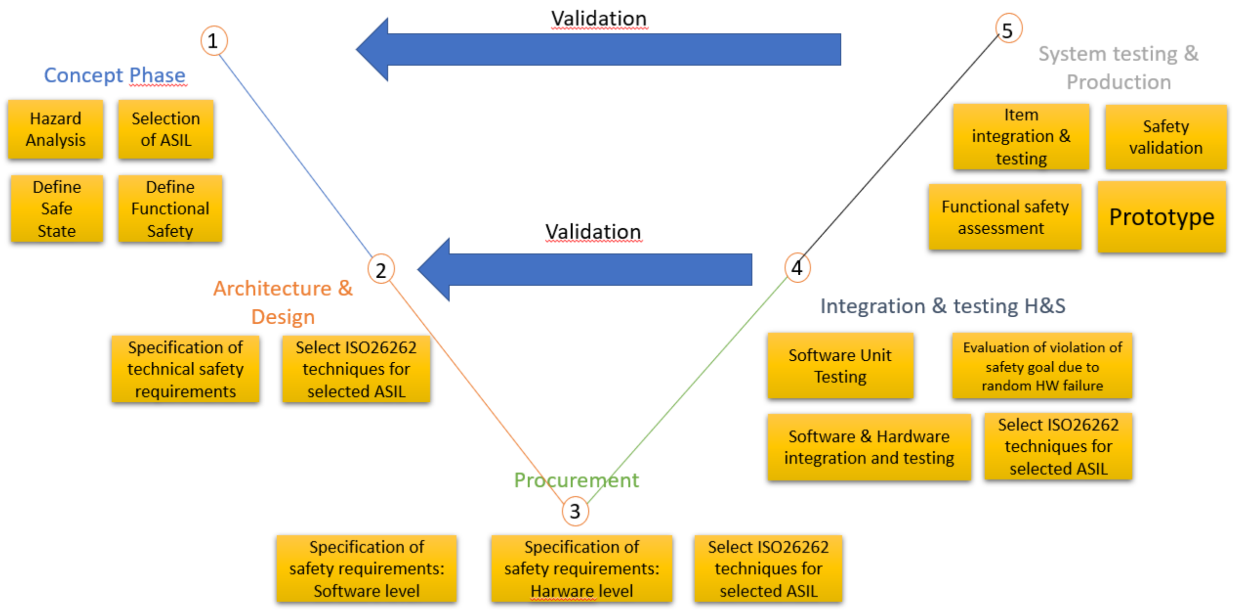

Validation
1
5
Concept Phase
System testing & Production
Hazard Analysis
Selection of ASIL
Item integration & testing
Safety validation
Define Safe State
Define Functional Safety
Functional safety assessment
Prototype
Validation
2
4
Architecture & Design
Integration & testing H&S
Specification of technical safety requirements
Select ISO26262 techniques for selected ASIL
Software Unit Testing
Evaluation of violation of safety goal due to random HW failure
Software & Hardware integration and testing
Select ISO26262 techniques for selected ASIL
Procurement
3
Specification of safety requirements: Software level
Specification of safety requirements: Harware level
Select ISO26262 techniques for selected ASIL

*Figure 13. Hybrid V-functional safety concept*

## 4.3. Steps for the Hybrid V-Functional safety cycle

The figure 13 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,

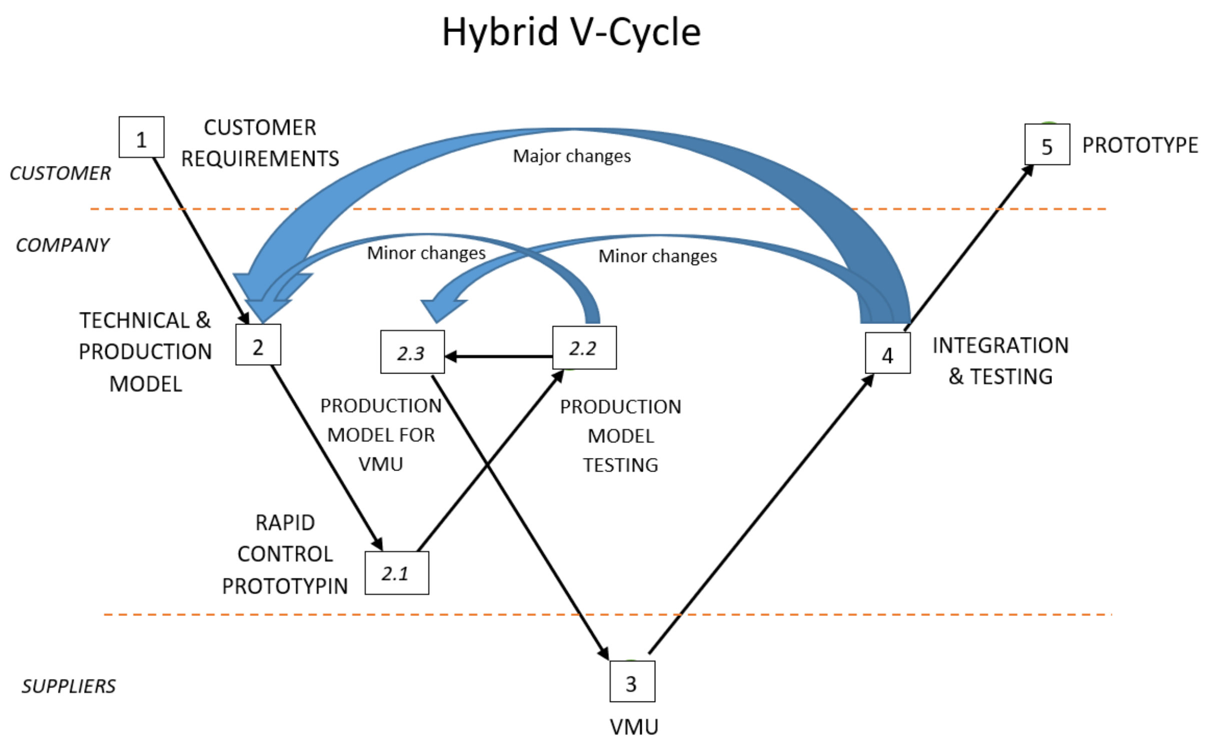


*Figure 14. Steps for hybrid V-cycle*

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 13. 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.

## 4.4. Explanation for the steps of Hybrid V- cycle

*Table 1 Steps for 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 |

Now, we will explain the steps we mentioned in table 1.

### 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.

### 2.1 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 (dspace, 2019).

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.

## 2.2 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 2.2 step to 2, 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.

## 2.3 Production model for VMU (Vehicle Management Unit)

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.

## 3. 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.

## 4. 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 must go back to fourth step or all the way back to second step.

**Minor changes**

At the fourth 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 fourth 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 2.2 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 cannot skip any step in between, and we must follow the procedure again.

## 5. 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:

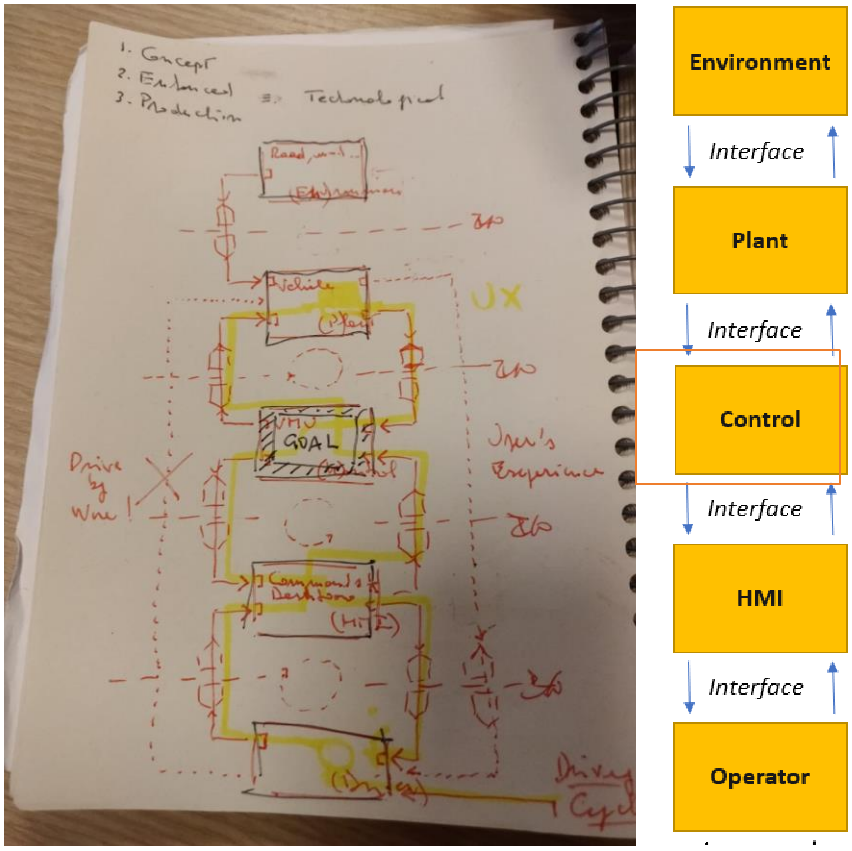


Figure 15. Reusability and modularity concept

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.

### 5.1. 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.

## 5.2. Plant

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

## 5.3. 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.

## 5.4. 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.

## 5.5. 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.

## 5.6. 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.

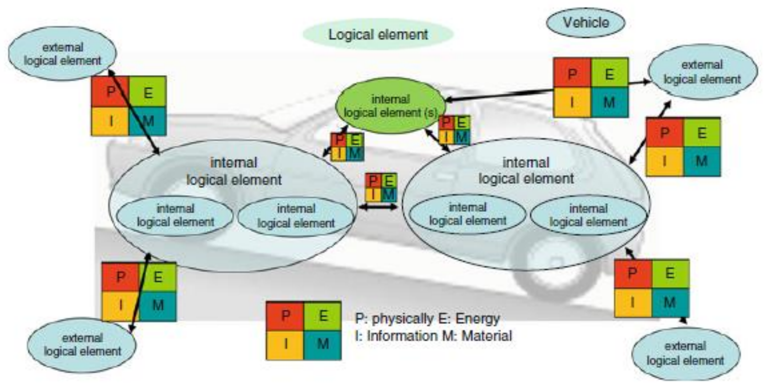


**Fig. 3.9** Multi-dimensional boundary and interface analysis (*Source* Derived from Ford-FMEA-Handbook)

*Figure 16. Interface analysis (Ross, 2016)*

According to the Ford-FMEA-handbook there are four kinds of interfaces. (Ross, 2016)

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

## 6. 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.

## 7. 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

## 7.1. Software in the loop

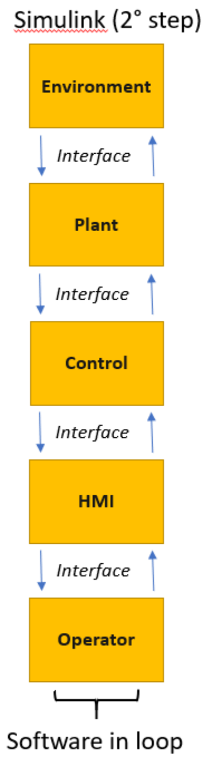


*Figure 21. Software in the loop*

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

## 7.2. Hardware in the loop containing Rapid control prototype

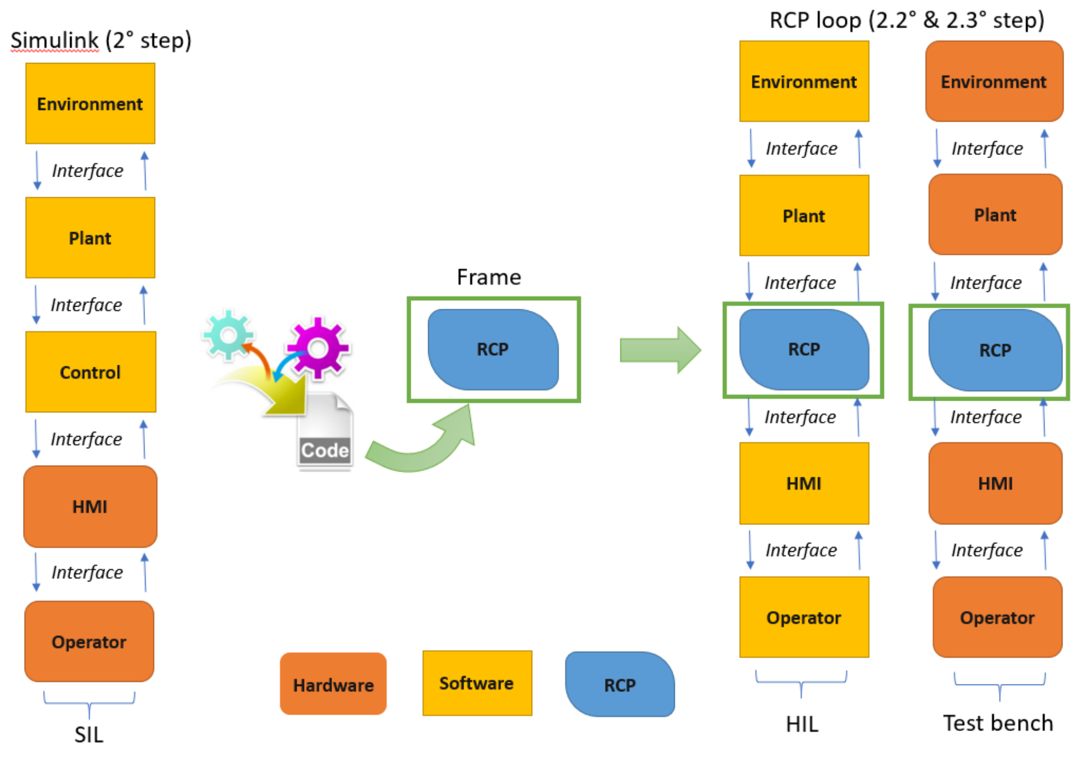


*Figure 22. Hardware in the loop containg rapid control prototype*

This step is covered by 2.2 and 2.3 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.

## 7.3. 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 generatioin*

### 7.4. 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.

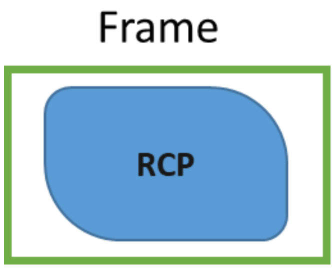


*Figure 24. Defining frame*

## 7.5. Rapid Control Prototype Loop (2.2 & 2.3 step)

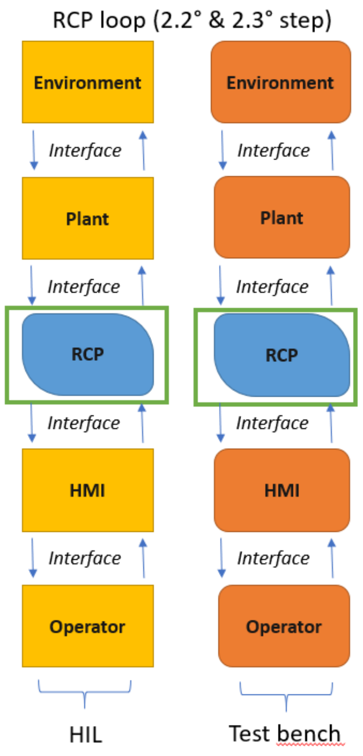


*Figure 25. RCP loop*

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.

## 7.6. Hardware in the loop for vehicle management unit

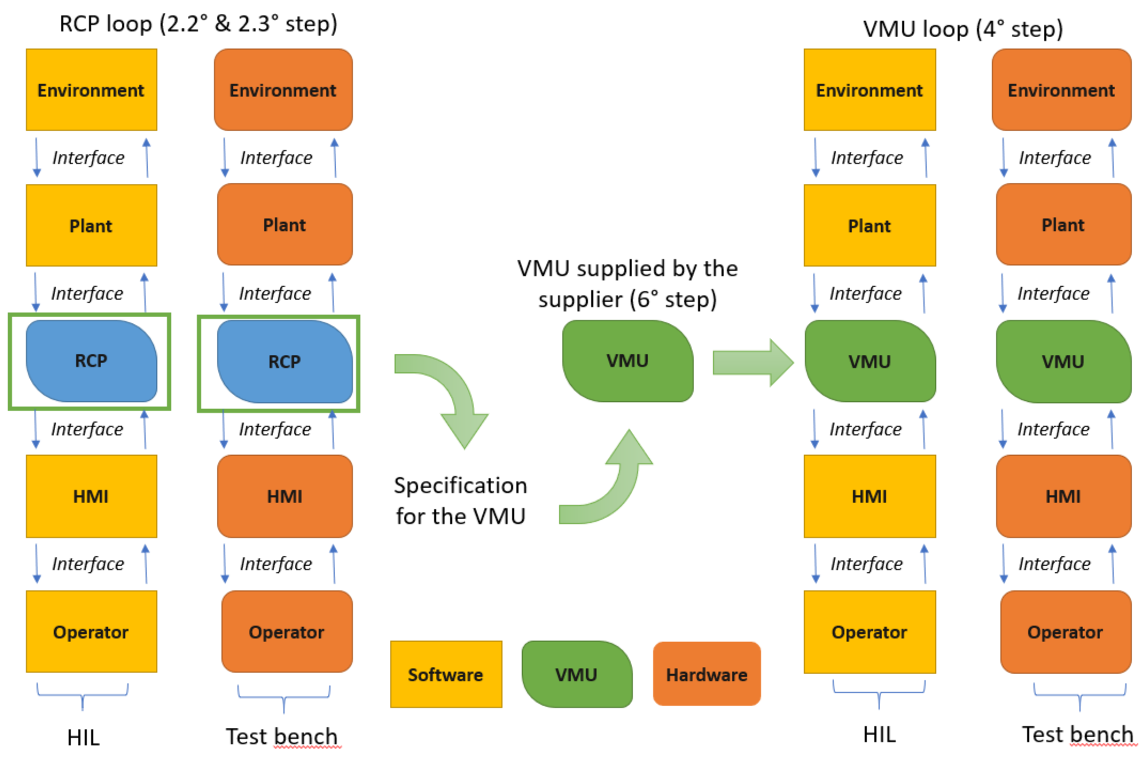

RCP loop (2.2° & 2.3° step)
VMU loop (4° step)
Environment
Interface
Plant
RCP
HMI
Operator
HIL
Test bench
VMU supplied by the supplier (6° step)
VMU
Specification for the VMU
Software
Hardware

*Figure 26. Hardware in the loop for VMU*

This step is covered by 4th 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 4th in our Hybrid V-functional safety cycle.

## 7.7. VMU loop (4th step)

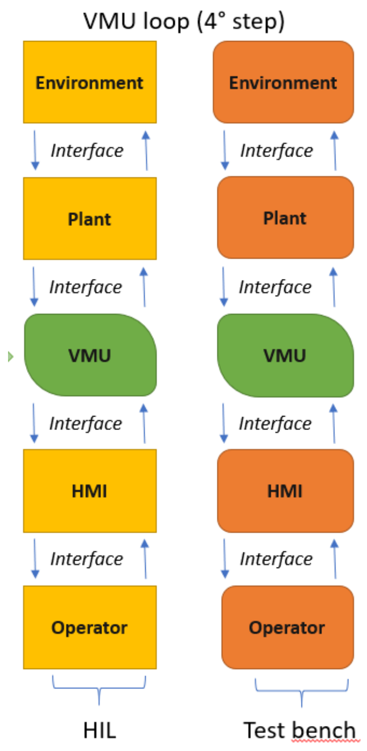


*Figure 27. VMU loop*

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 4th 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.

## 7.8. 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.

## 7.9. 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.

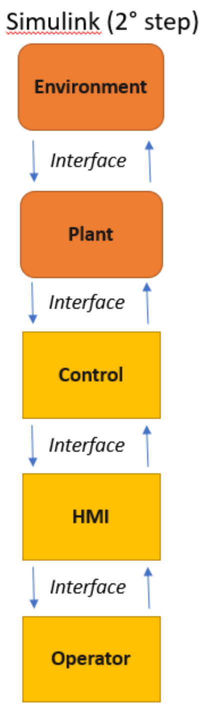


*Figure 28. Combinations of software in the loop*

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.

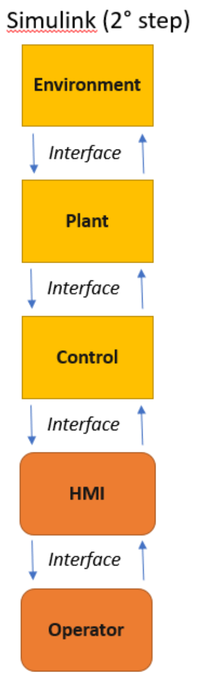


*Figure 29. Combinations of software in the loop*

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

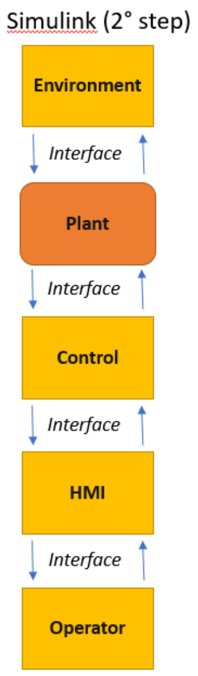


*Figure 30. Combinations of software in the loop*

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:

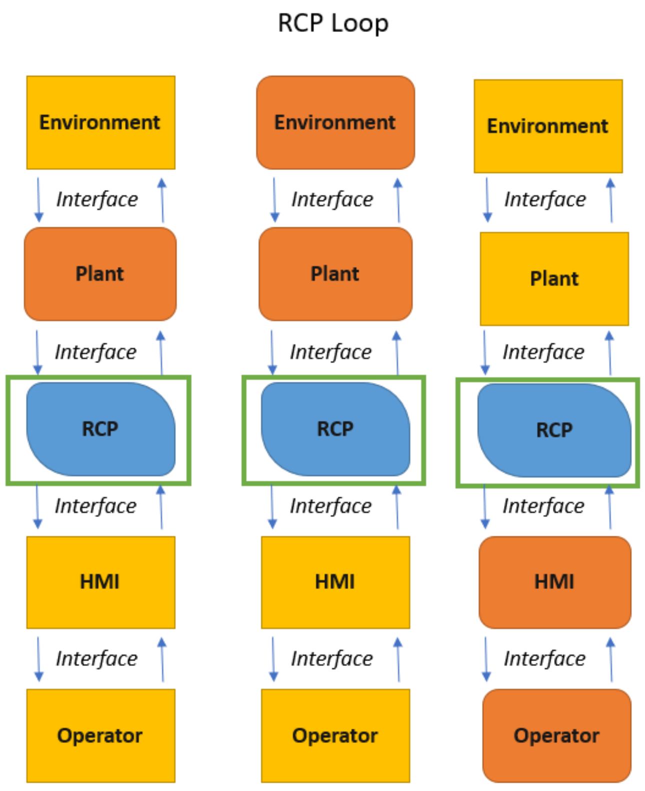


*Figure 31. Proposed combinations of hardware in loop for rcp*

Comparing figure 17 with 31 shows that even in the 2.3 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 31, 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 defined 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 think of other combinations also and make a hybrid model to satisfy our requirements. We will see an example to understand how it might work.

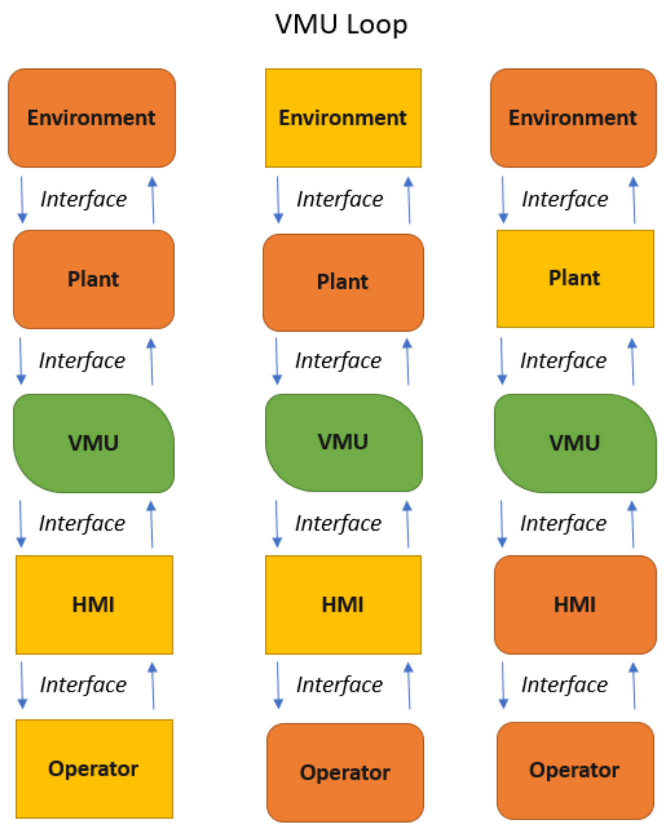


*Figure 32. Proposed combination of hardware in the loop for VMU*

## 8. Updated Hybrid V- Cycle

The following is our proposal for the updates in the V-cycle. There is small description of the whole cycle at the end. For complete consideration look at the previous headings.

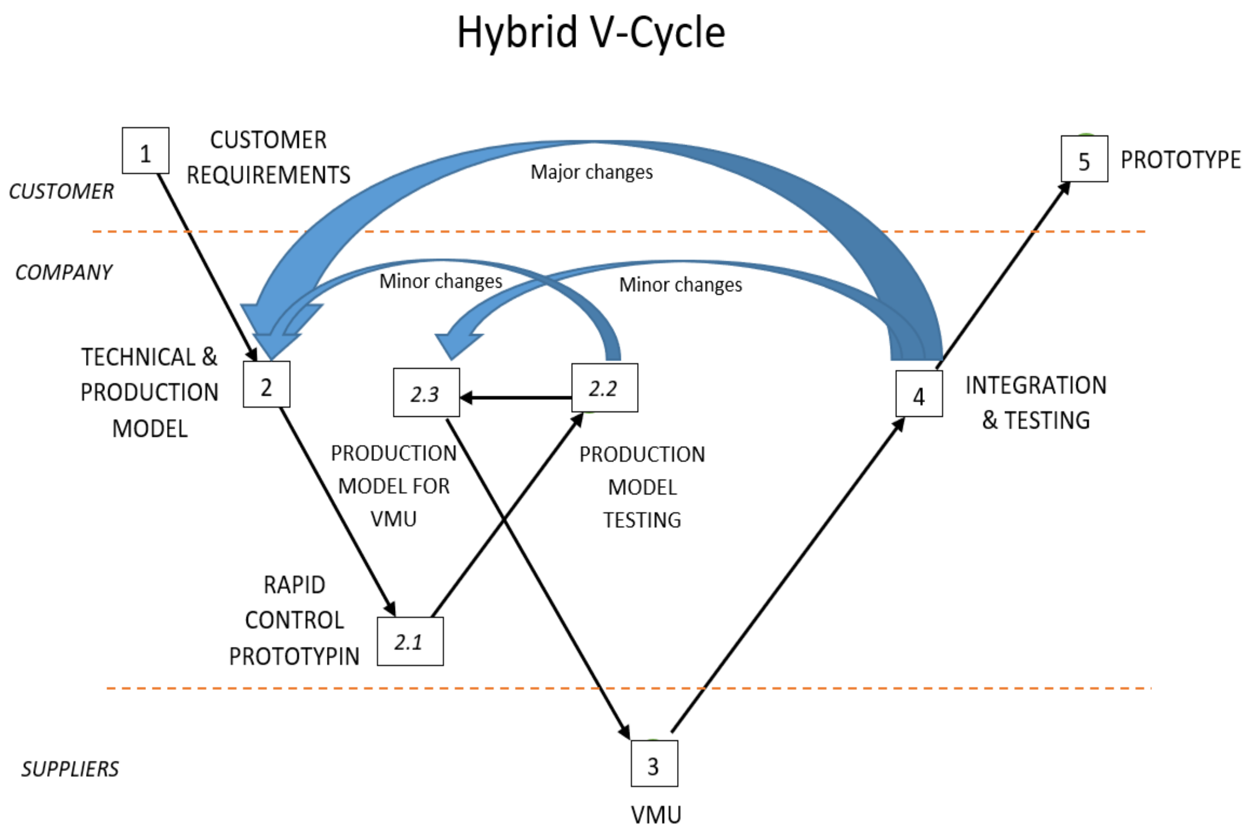


*Figure 33. Updated hybrid V-cycle*

This updated V-cycle is slightly different from the previous one. In this cycle, we have further explained what we do in each step while developing our model. 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 30-32, we can adopt different combinations of the software and hardware parts for making our system.

## 2.1 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 (dspace, 2019). 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.

## 2.1 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 2.2 step, we can have different combinations of software and hardware blocks. We can adopt the suitable combination of both.

2.2 Production model for VMU (Vehicle Management Unit)

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

3 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.

4 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 2.2 or step 2 depending upon the changes we must make. If we have to make minor changes we have to move from step 4th to step 2.3 while if we have to make a major change we have to move straight from 4th step to 2nd one and again we have to follow all the points leading up to 4th step again.

5 Final product requirements

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

## 9. 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: (MathWorks, n.d.)

- 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

## 10. 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 and 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:

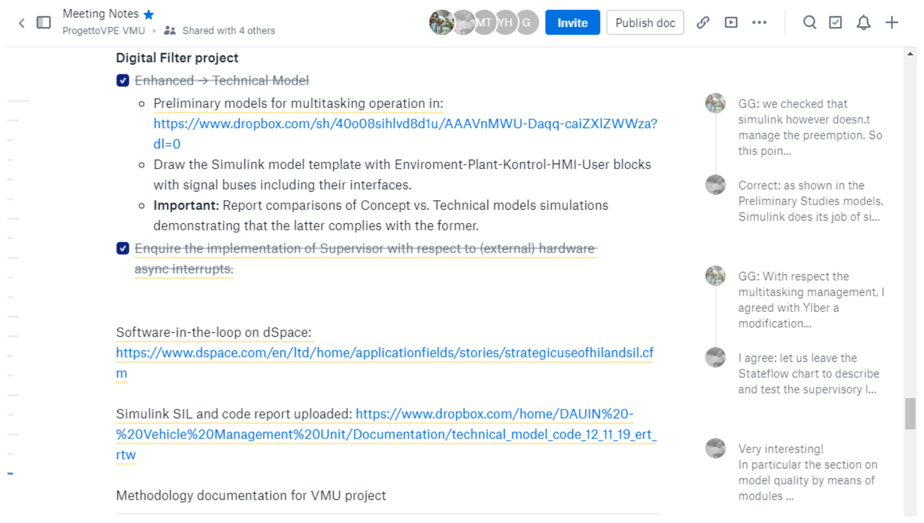
- Concept model
- Technical model
- Production model

***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:



Meeting Notes

ProgettoVPE VMU • Shared with 4 others

**Digital Filter project**

- [x] ~~Enhanced → Technical Model~~
  - Preliminary models for multitasking operation in: https://www.dropbox.com/sh/40o08sihlvd8d1u/AAAVnMWU-Daqq-caiZXlZWWza?dl=0
  - Draw the Simulink model template with Enviroment-Plant-Kontrol-HMI-User blocks with signal buses including their interfaces.
  - **Important:** Report comparisons of Concept vs. Technical models simulations demonstrating that the latter complies with the former.
- [x] ~~Enquire the implementation of Supervisor with respect to (external) hardware async interrupts.~~

Software-in-the-loop on dSpace: https://www.dspace.com/en/ltd/home/applicationfields/stories/strategicuseofhilandsil.cfm

Simulink SIL and code report uploaded: https://www.dropbox.com/home/DAUIN%20-%20Vehicle%20Management%20Unit/Documentation/technical_model_code_12_11_19_ert_rtw

Methodology documentation for VMU project

GG: we checked that simulink however doesn.t manage the preemption. So this poin...

Correct: as shown in the Preliminary Studies models, Simulink does its job of si...

GG: With respect the multitasking management, I agreed with Ylber a modification...

I agree: let us leave the Stateflow chart to describe and test the supervisory l...

Very interesting!
In particular the section on model quality by means of modules ...

*Figure 34. Meeting Notes snapshot*

The illustration below sums up the architecture of Dropbox:

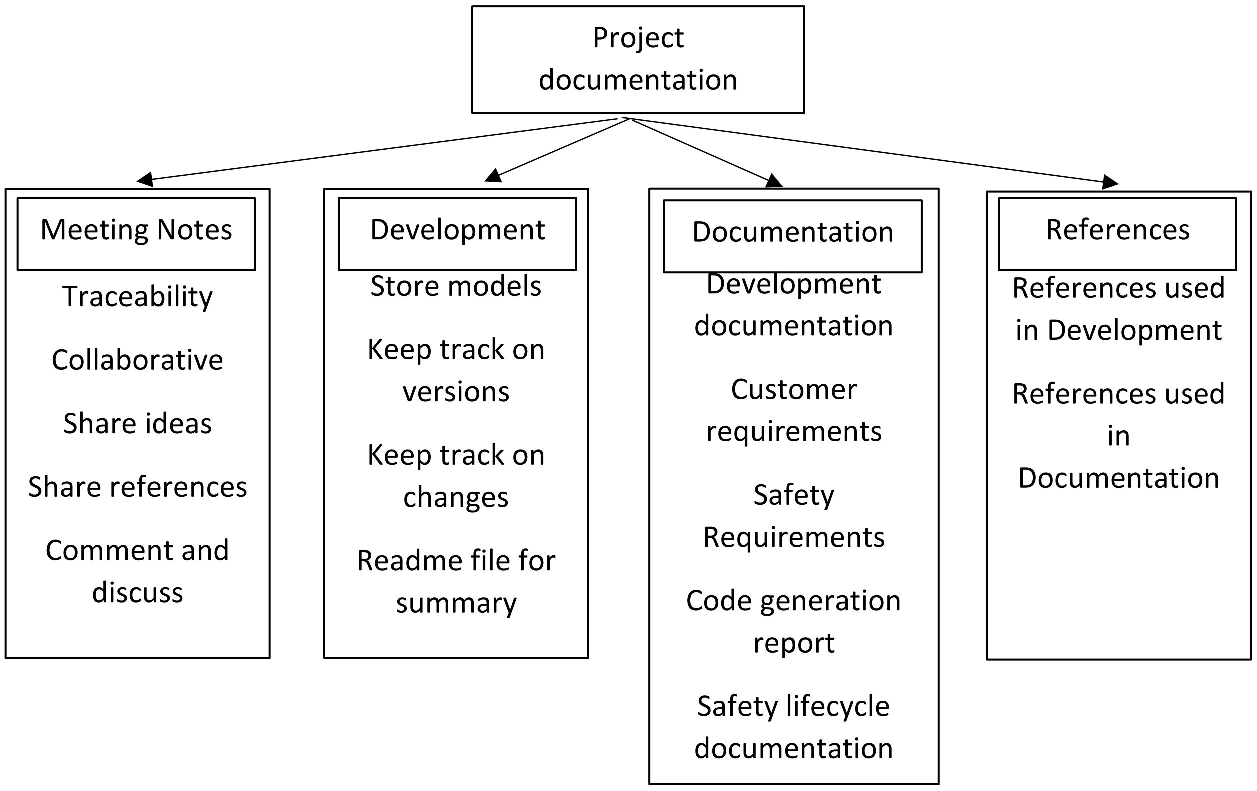


*Figure 35. Documentation architecture*

# Chapter 1 Part 2

## 1. Customer Requirements

Artefacts of this process are:

- Item definition
- Requirement specifications
- Hazard analysis and risk assessment
- Functional safety concept
- Concept model

### 1.1. 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.

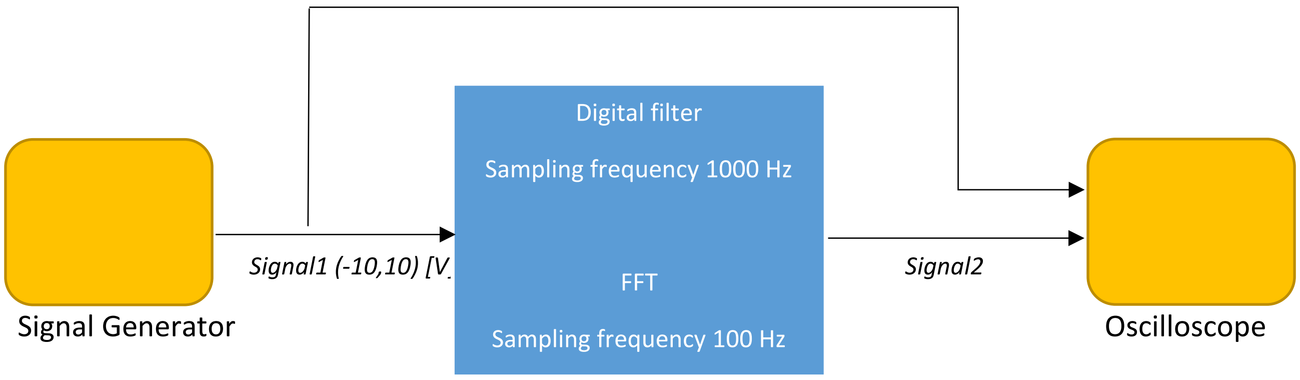


*Figure 36. Digital filter and FFT block scheme*

***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:

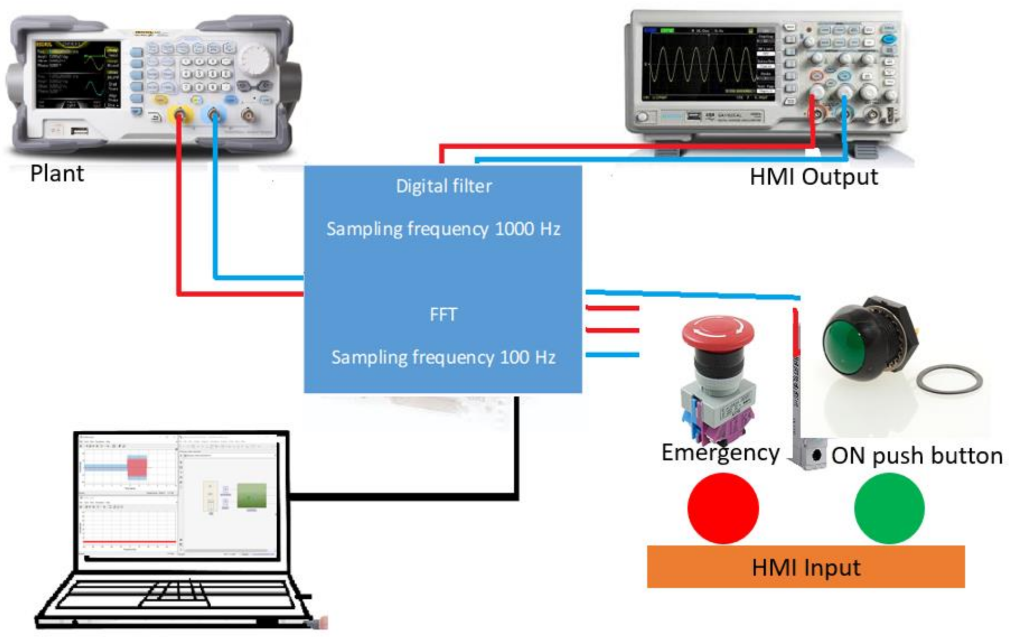
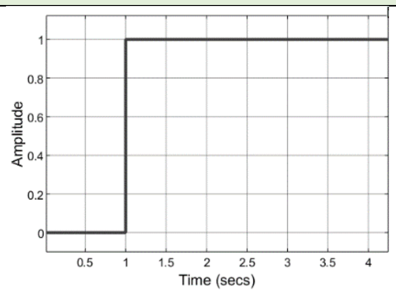
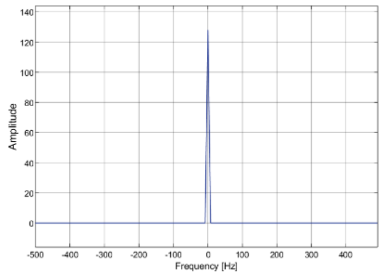
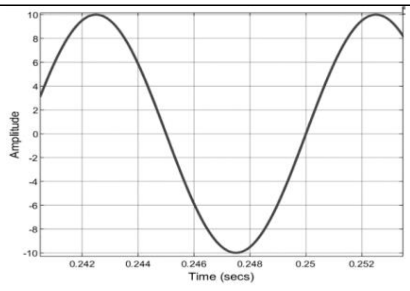
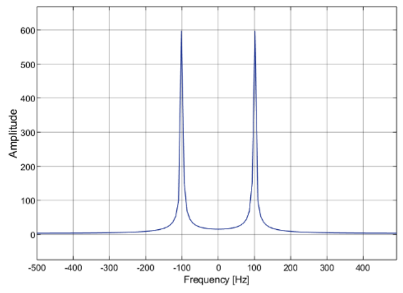


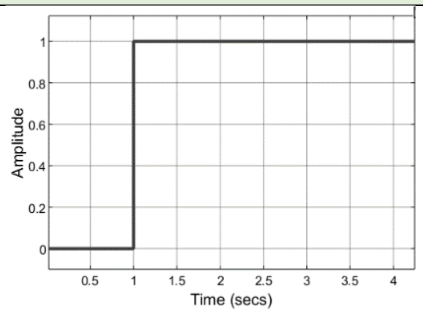
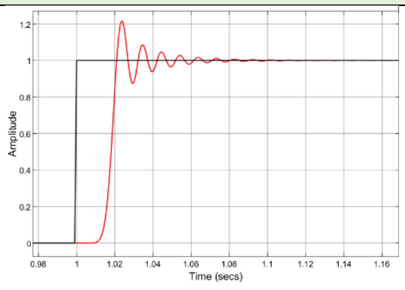
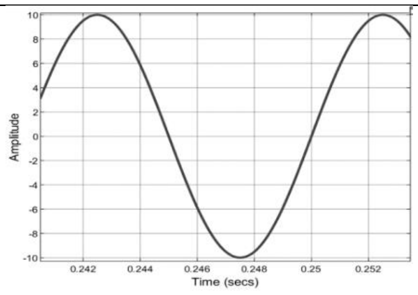
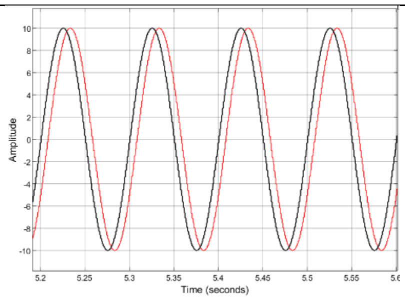
*Figure 37. Layout*

*Table 2 Signal identification*

| **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 |

Expected results:

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

| | Input | Expected Output for Filter |
|---|---|---|
| |  |  |
| |  |  |

## Requirement Specifications

*Table 3 Requirement specification*

| ***Function 1*** | ***Digital filter*** |
|---|---|
| *Requirement 1-1* | Sampling frequency must be $f_s$ = 1000 Hz |
| *Requirement 1-2* | Input signal voltage shall be in the range of $\pm 10\ [V]$ |
| *Requirement 1-3* | Output signal voltage shall be in the range of $\pm 10\ [V]$ |
| ***Function 2*** | ***Fast Fourier Transform*** |
| *Requirement 2-1* | Sampling frequency must be $f_s$ = 100 Hz |
| ***Function 3*** | ***Supervisory Control*** |
| *Requirement 3-1* | External Start/Stop push button must be added |
| *Requirement 3-2* | System is turned by the Start/Stop push button |
| *Requirement 3-3* | System is stopped by the Start/Stop push button |
| *Requirement 3-4* | An external emergency switch must be added |
| *Requirement 3-5* | When emergency switch is turned on, output must go to 0 |

## 1.2. Hazard analysis and risk assessments

Hazard identification:

| Component | Failure Mode | Effect on the item |
|---|---|---|
| Signal generator | FM1: Wrong input signal<br>FM2: Not grounded<br>FM3: Signal voltage beyond limits | Hazard 1: Item gives wrong output<br>Hazard 2: Possibility of damage<br>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<br>FM6: Switch does not turn off | Hazard 5: Item does not turn on<br>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<br>FM9: Not grounded<br>FM10: Oscilloscope shows wrong output | Hazard 8: Item does not give any output<br>Hazard 9: Possibility of damage<br>Hazard 10: Item gives a wrong output |

Table 4. Hazard analysis

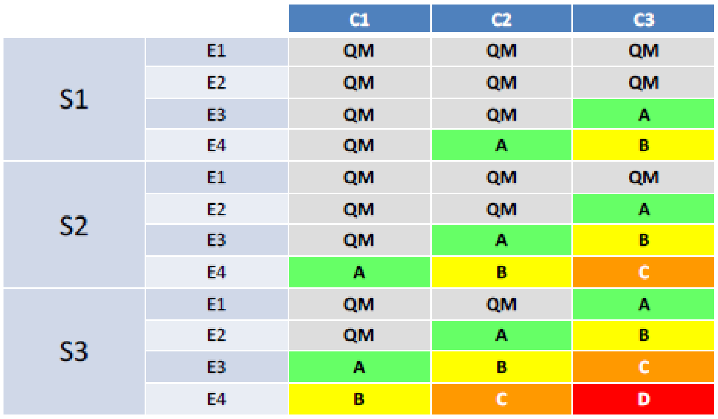
Hazard analysis and assessment:

| Hazard | Effect | Comment |
|---|---|---|
| Hazard 1: | Severity: 1<br>Exposure: 4<br>Controllability: 1 | Wrong output can lead on wrong conclusions<br>Signal generator is always on when item is on<br>Simply controllable |
| Hazard 2: | Severity: 1<br>Exposure: 4<br>Controllability: 1 | Possible damage<br>Signal generator is always on when item is on<br>Simply controllable |
| Hazard 3: | Severity: 1<br>Exposure: 4 | High voltage can cause electrical damage, causing damage to operator<br>Signal generator is always on when item is on |

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

*Table 5. 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 38. 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.

### 1.3. 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

## 1.4. 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.

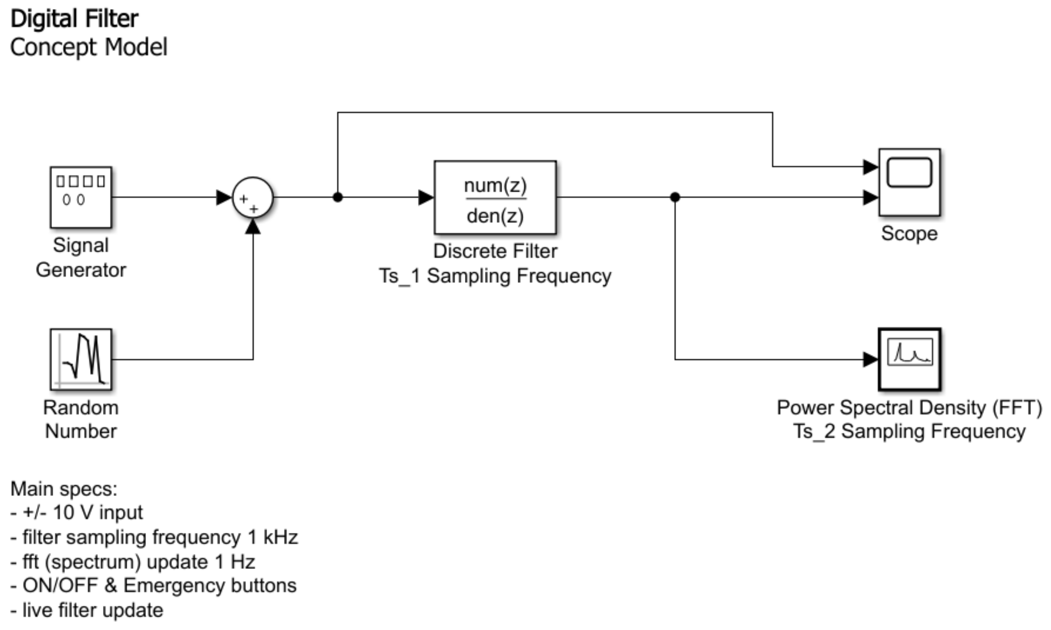
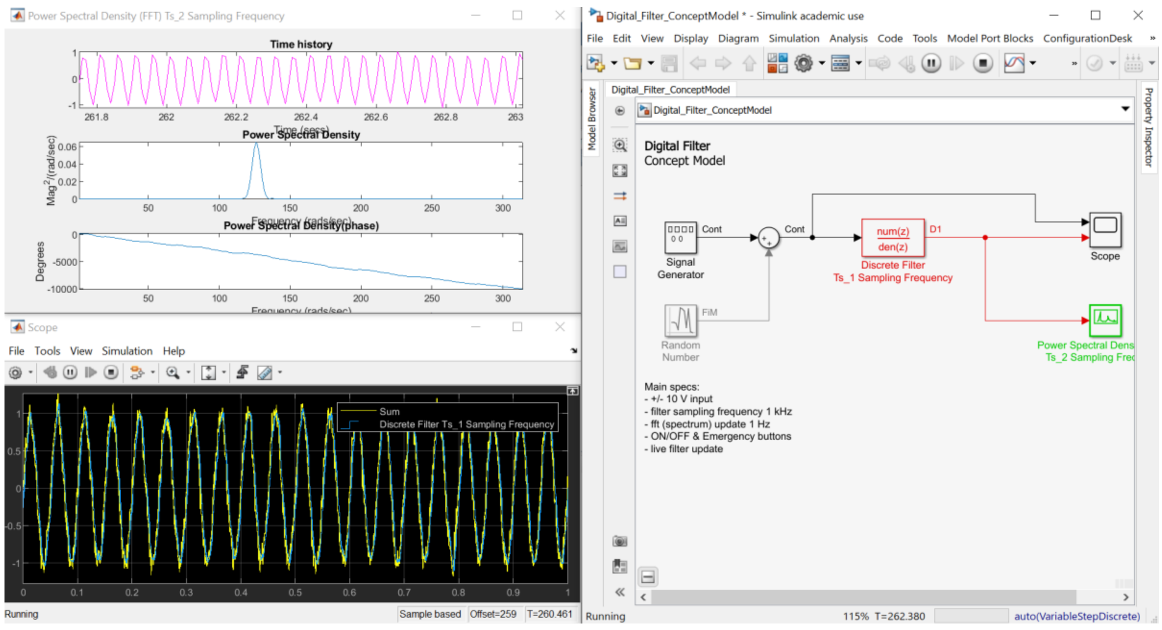


*Figure 39. Concept model in Simulink*



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

*Figure 40. Results of Concept Model*

## 2. Technical and production model

Modular architecture and the components with interfaces must be defined:

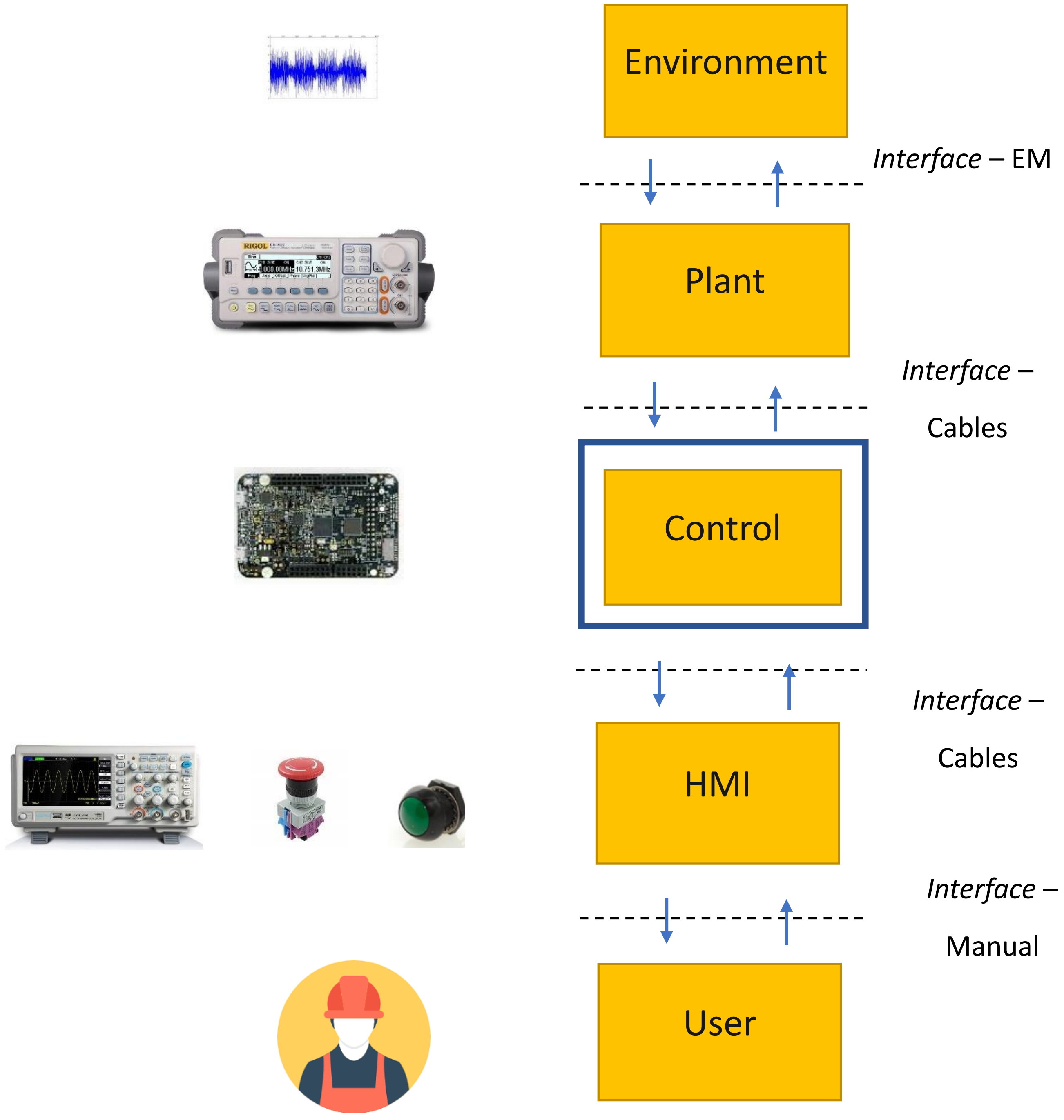


*Figure 41. Item Architecture*

The following characteristic or features specified should be seen as requirements: (Ross, 2016)

- 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 subsytems
- Input signal is analog with range ±10 V, with a certain high frequency noise
- Output signal is analog with range ±10 V

### 2.0.1. 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

### 2.0.2. Production development: software level – 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

## 2.0.3. Initiation

Guidelines to perform modelling that are required by ISO26262:

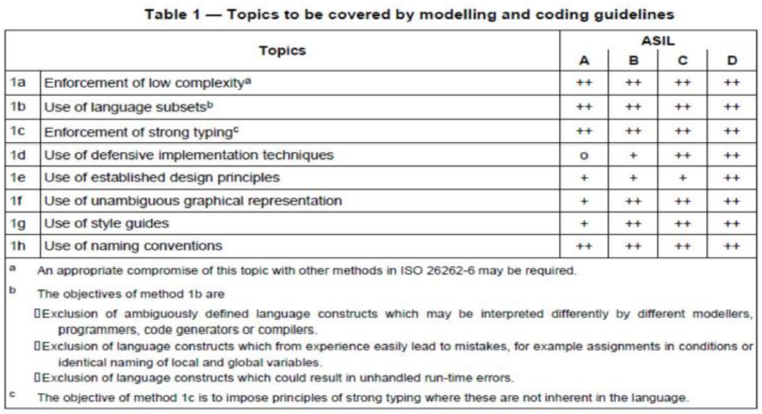


Table 1 — Topics to be covered by modelling and coding guidelines

| 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 | o | + | ++ | ++ |
| 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 42. 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***: A redundant switch must be added, thus in case of failure of one of the switches, the other switch realizes Safety Goal 1.

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

## 2.0.4. 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.

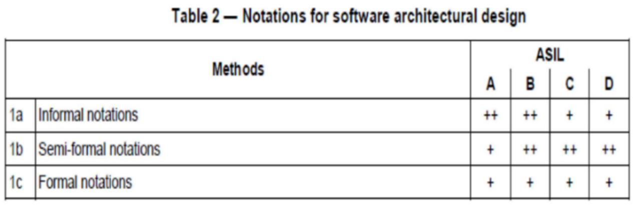


Table 2 — Notations for software architectural design

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

*Figure 43. 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.

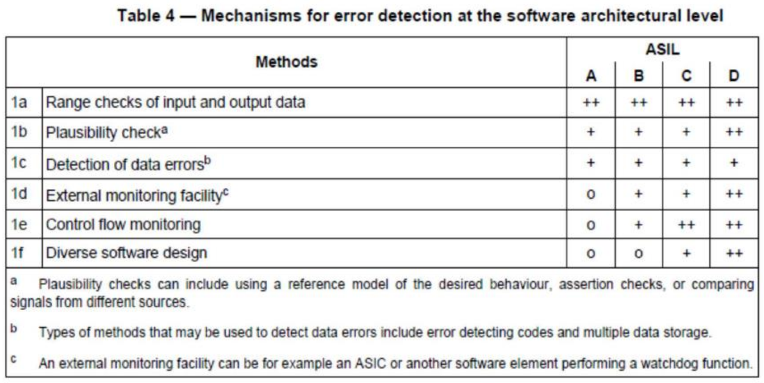


Table 4 — Mechanisms for error detection at the software architectural level

| 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] | o | + | + | ++ |
| 1e | Control flow monitoring | o | + | ++ | ++ |
| 1f | Diverse software design | o | o | + | ++ |

[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 44. 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*

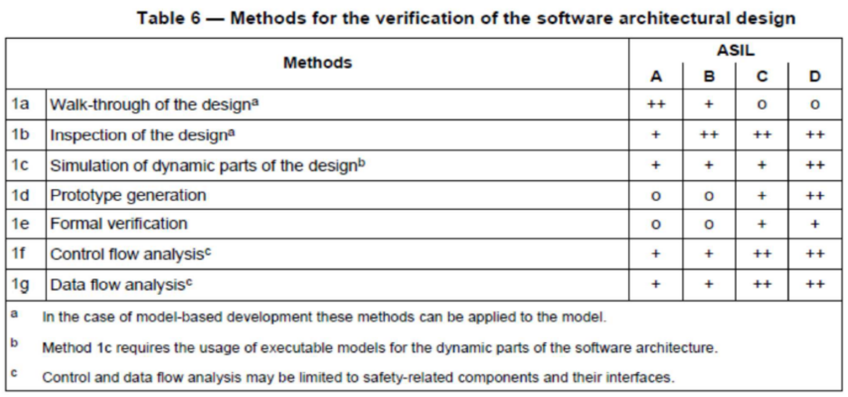


**Table 6 — Methods for the verification of the software architectural design**

| | Methods | ASIL | | | |
|---|---|---|---|---|---|
| | | A | B | C | D |
| 1a | Walk-through of the design[a] | ++ | + | o | o |
| 1b | Inspection of the design[a] | + | ++ | ++ | ++ |
| 1c | Simulation of dynamic parts of the design[b] | + | + | + | ++ |
| 1d | Prototype generation | o | o | + | ++ |
| 1e | Formal verification | o | o | + | + |
| 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 45. 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:

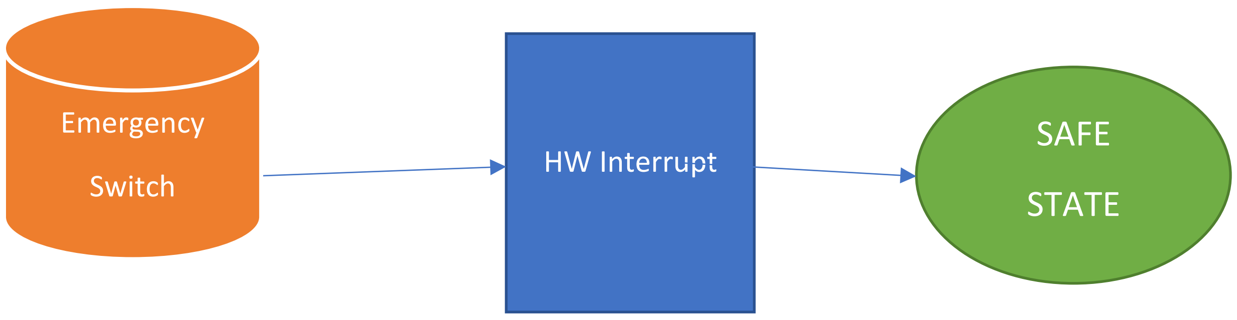


*Figure 46. Illustration of HW Interrupt*

We have three states:

- OFF State – which is set by default

- ON State – where the actions required are performed and infrom the User via HMI
- Emergency State – Where we set the output to zero and inform the User via HMI

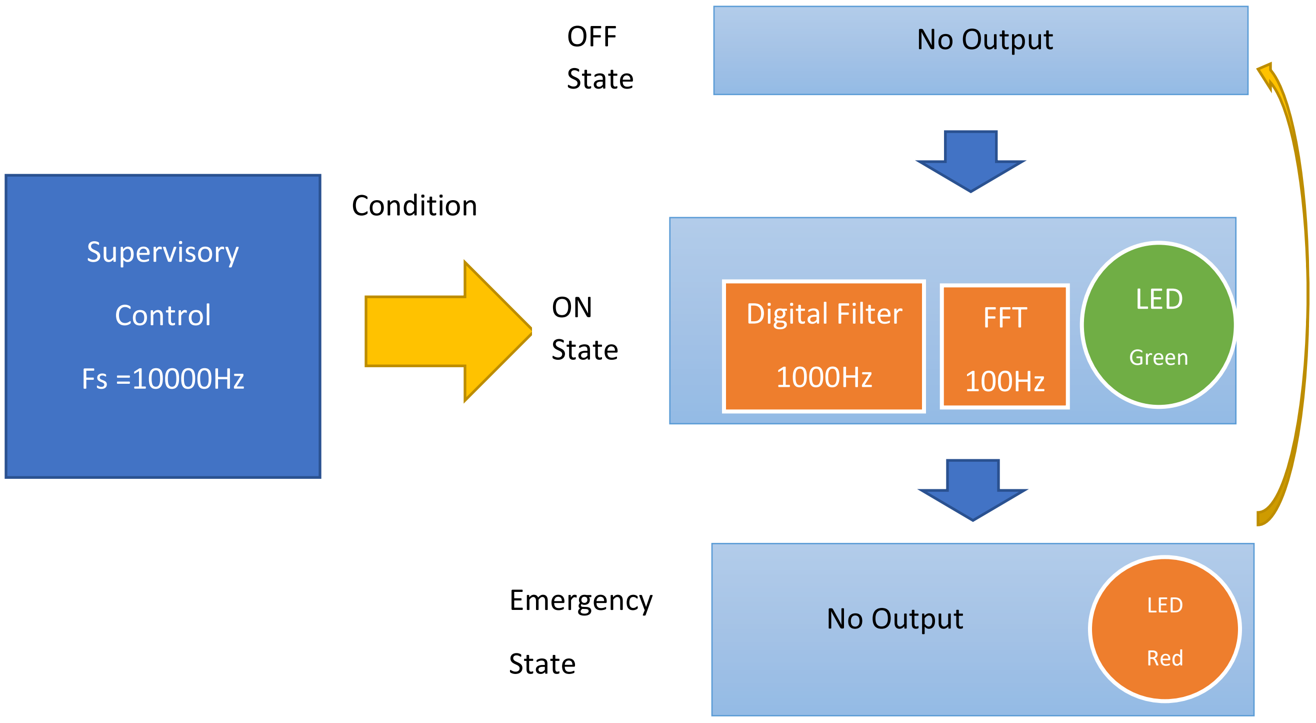


*Figure 47. Illustration of model hierarchy*

### 2.0.5. Model-based Design

Notations to be followed for ASIL A:

- *Natural language*
- *Informal notations*

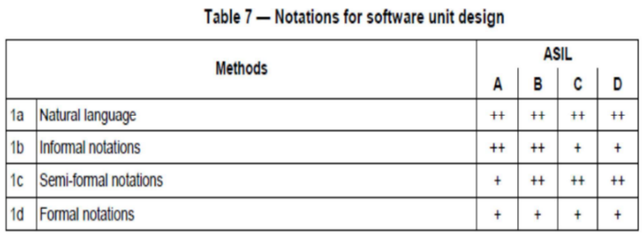


**Table 7 — Notations for software unit design**

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

*Figure 48. 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*

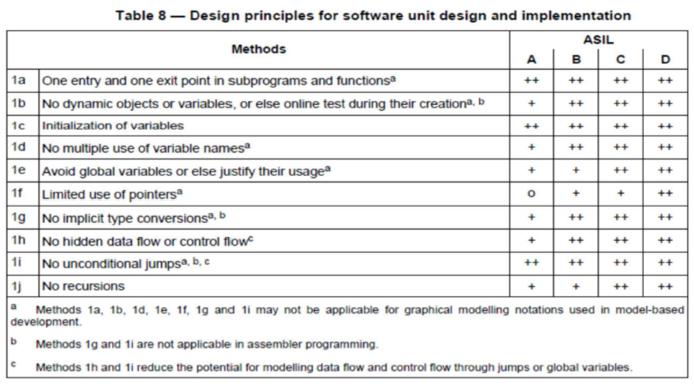


**Table 8 — Design principles for software unit design and implementation**

| | 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] | o | + | + | ++ |
| 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 49. Design principles for sw unit design and implementation*

Methods for software unit testing:

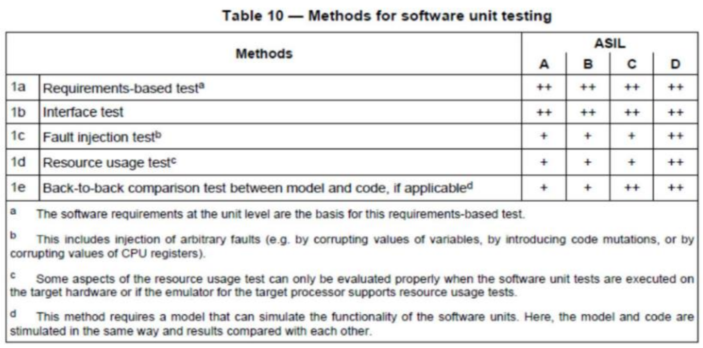


**Table 10 — Methods for software unit testing**

| | 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 50. Methods for software unit testing*

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

### 2.0.6. Supervisory control/Stateflow

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.

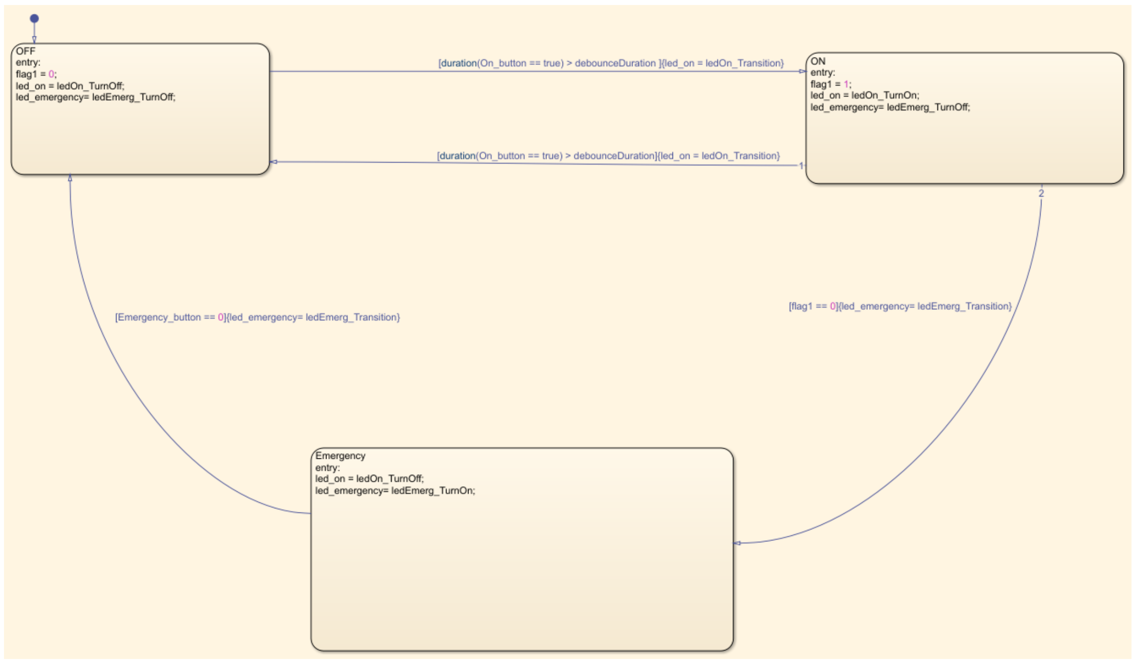

OFF
entry:
flag1 = 0;
led_on = ledOn_TurnOff;
led_emergency= ledEmerg_TurnOff;
[duration(On_button == true) > debounceDuration ]{led_on = ledOn_Transition}
ON
entry:
flag1 = 1;
led_on = ledOn_TurnOn;
led_emergency= ledEmerg_TurnOff;
[duration(On_button == true) > debounceDuration]{led_on = ledOn_Transition}
1
2
[Emergency_button == 0]{led_emergency= ledEmerg_Transition}
[flag1 == 0]{led_emergency= ledEmerg_Transition}
Emergency
entry:
led_on = ledOn_TurnOff;
led_emergency= ledEmerg_TurnOn;

*Figure 51. Stateflow chart*

Emergency function-call task Stateflow:

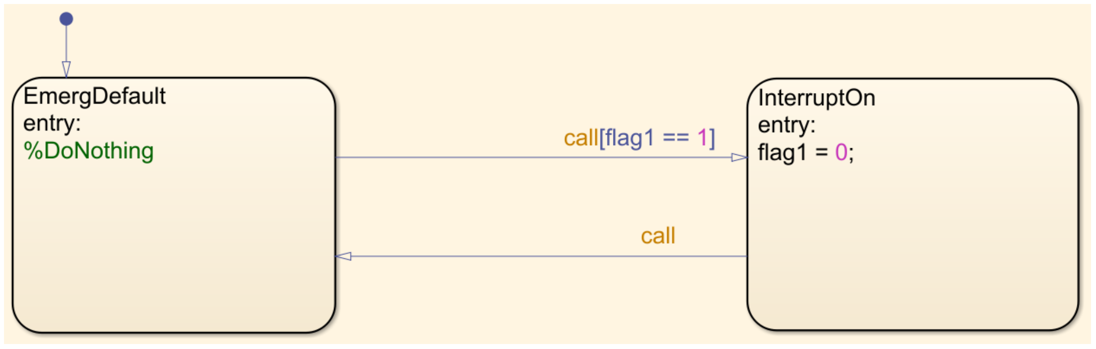


*Figure 52. Interrupt handling*

## Fast Fourier Transform

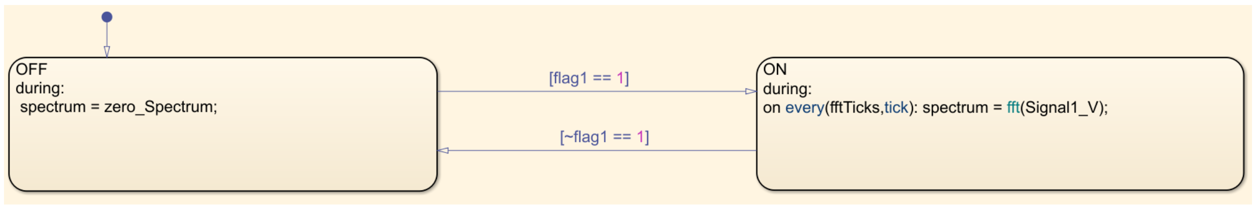

OFF
during:
spectrum = zero_Spectrum;
[flag1 == 1]
[~flag1 == 1]
ON
during:
on every(fftTicks,tick): spectrum = fft(Signal1_V);

*Figure 53. 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:***

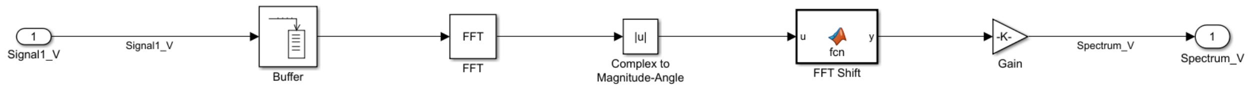


*Figure 54. FFT modelling*

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

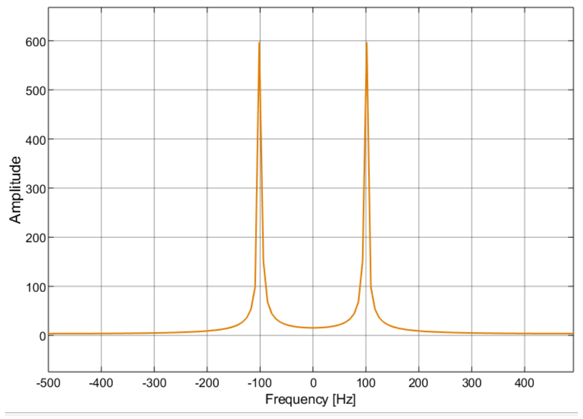


*Figure 55. FFT result*

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

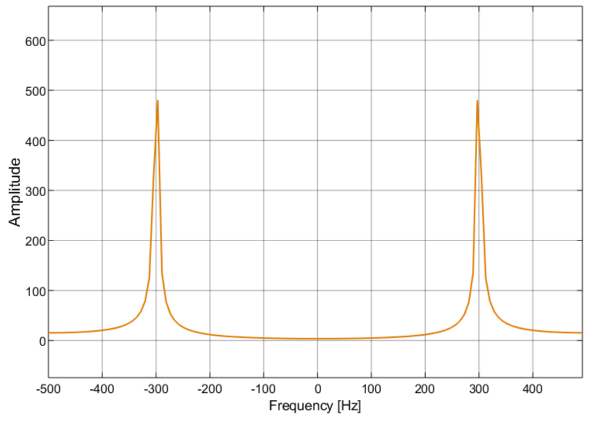
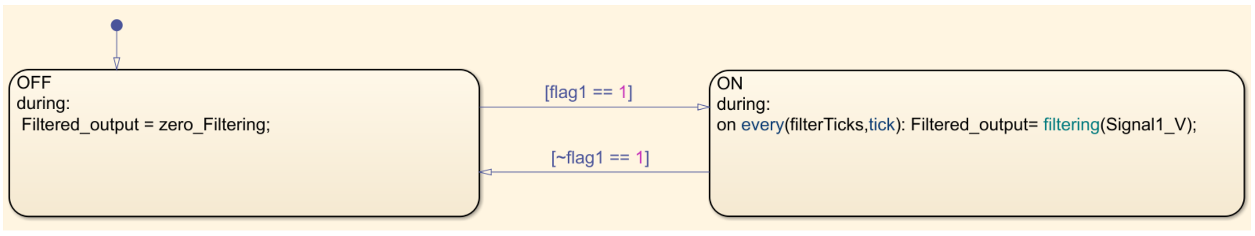


*Figure 56. FFT Result*

## Digital Filter



```
OFF
during:
 Filtered_output = zero_Filtering;
```

```
ON
during:
on every(filterTicks,tick): Filtered_output= filtering(Signal1_V);
```

*Figure 57. Task control*

Design characteristics:

- *Frequency response*: Lowpass
- *Sampling frequency*: $f_s = 1000\ Hz$
- *Cutting frequency:* $f_c = 100\ Hz$
- *Simulink block used*: FIR Filter
- *Filter order*: 2

Simulink block used:

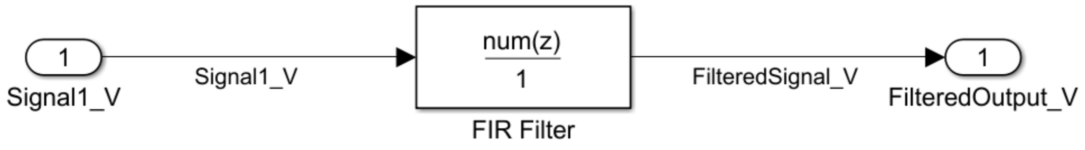


*Figure 58. Digital Filter modelling*

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

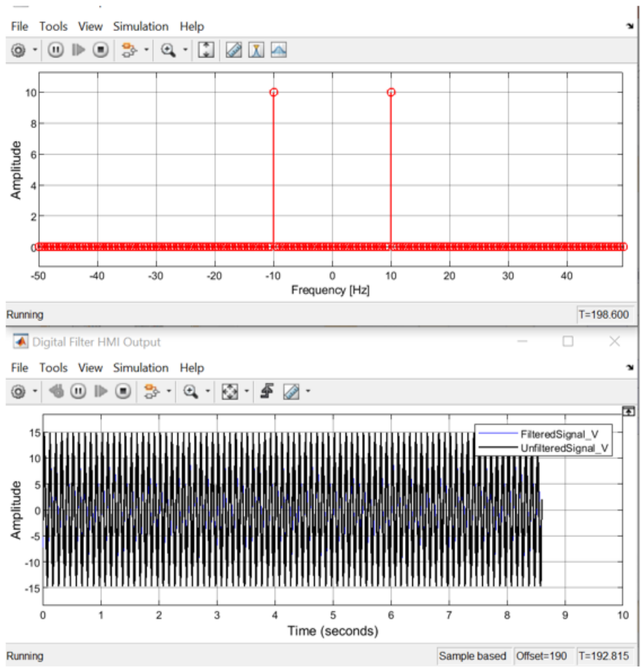


*Figure 59. Results 1 of Digital Filter and FFT*

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

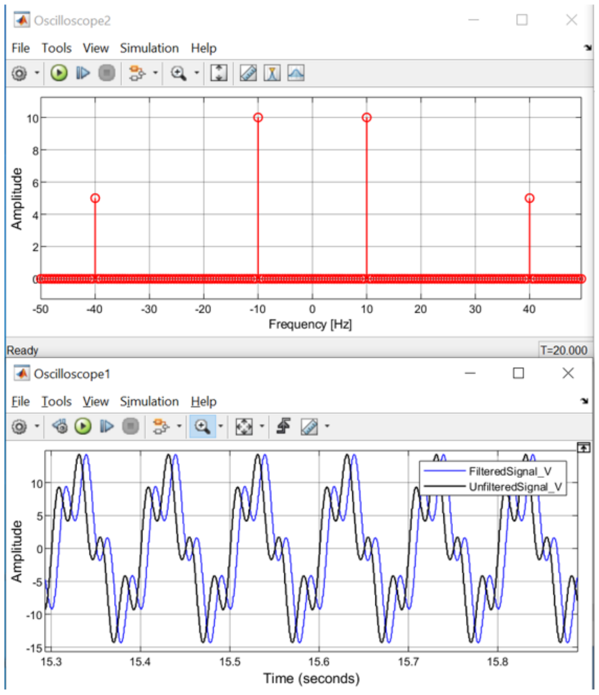


*Figure 60. Results 2 of Digital Filter and FFT*

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

## 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 must 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:

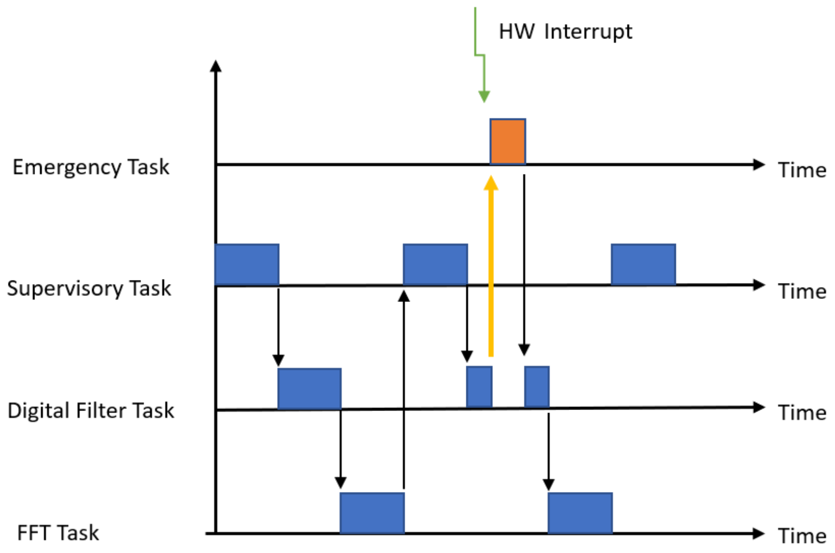


*Figure 61. Scheduling*

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*)

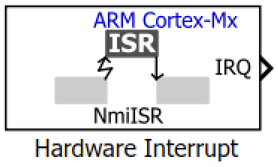
The example is given for the operator ‘every’ but it is the same for every operator. The important thing is that the variable ‘tick’ has to be linked in a Timer, Clock or to the base rate. Here, we decide to not put anything more and complicate the model and use the base rate. In this case, the base-rate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

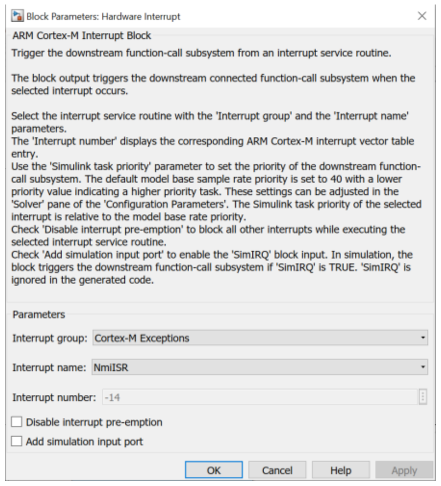
- Execute Supervisory Control Task in a base rate that is 10Khz
- Execute Digital Filter Task every 10 base rates, thus frequency is 1KHz
- Execute FFT Task every 100 base rates, thus frequency 100Hz.

## 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





ARM Cortex-M Interrupt Block

Trigger the downstream function-call subsystem from an interrupt service routine.

The block output triggers the downstream connected function-call subsystem when the selected interrupt occurs.

Select the interrupt service routine with the 'Interrupt group' and the 'Interrupt name' parameters.
The 'Interrupt number' displays the corresponding ARM Cortex-M interrupt vector table entry.
Use the 'Simulink task priority' parameter to set the priority of the downstream function-call subsystem. The default model base sample rate priority is set to 40 with a lower priority value indicating a higher priority task. These settings can be adjusted in the 'Solver' pane of the 'Configuration Parameters'. The Simulink task priority of the selected interrupt is relative to the model base rate priority.
Check 'Disable interrupt pre-emption' to block all other interrupts while executing the selected interrupt service routine.
Check 'Add simulation input port' to enable the 'SimIRQ' block input. In simulation, the block triggers the downstream function-call subsystem if 'SimIRQ' is TRUE. 'SimIRQ' is ignored in the generated code.

Parameters

Interrupt group: Cortex-M Exceptions

Interrupt name: NmiISR

Interrupt number: -14

☐ Disable interrupt pre-emption

☐ Add simulation input port

*Figure 62. 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.

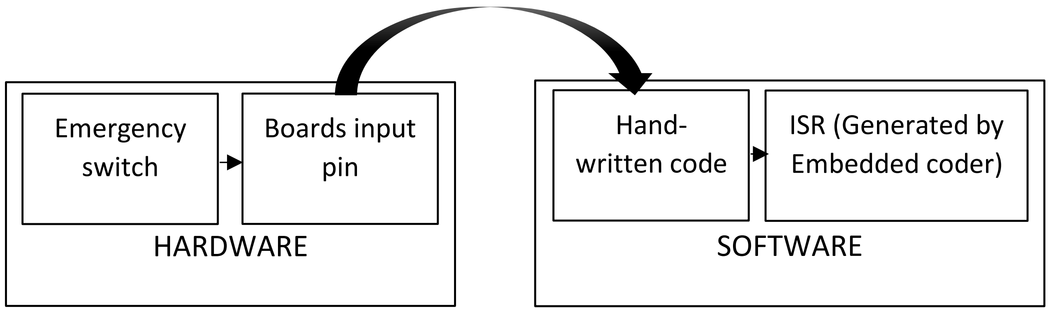


*Figure 63. Hardware interrupt flowchart*

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.

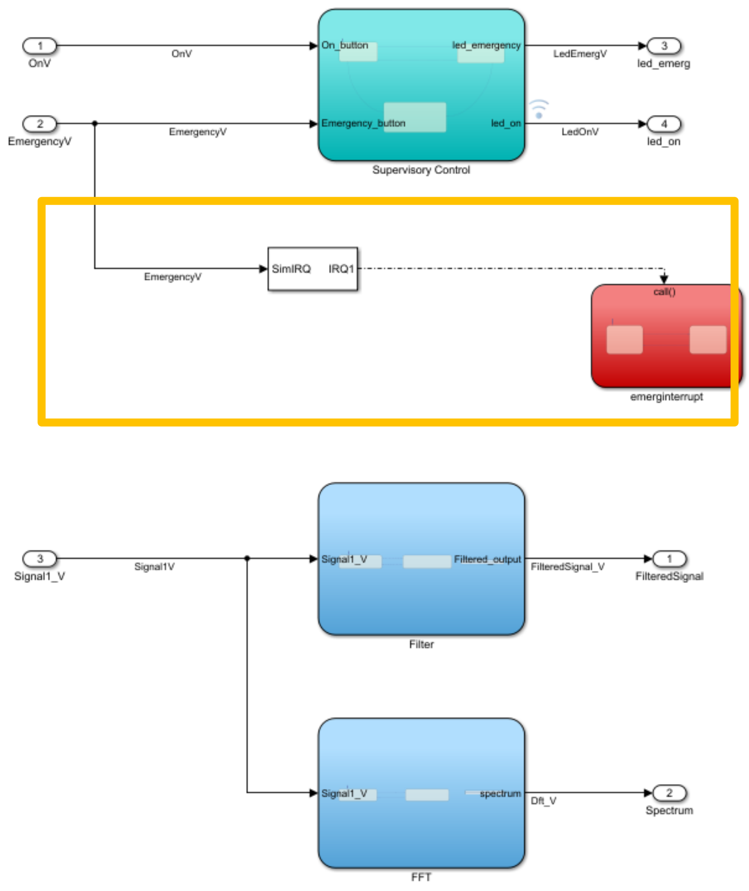


*Figure 64. Control content with Interrupt simulation*

Monitoring of the Supervisory Control:

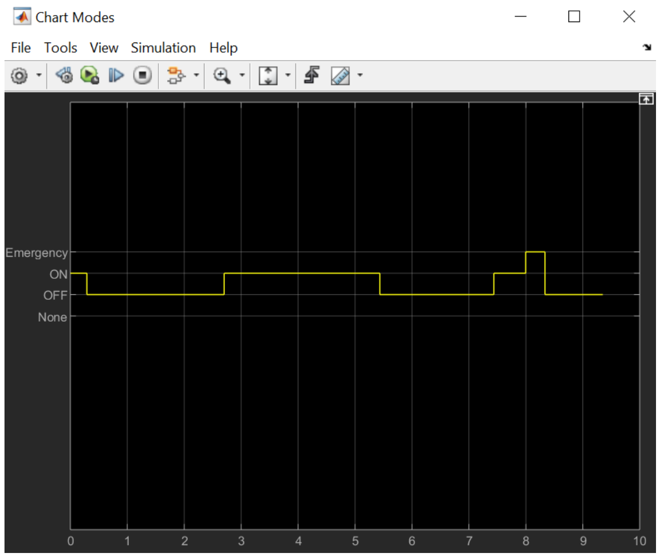


*Figure 65. Chart mode*

## 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.

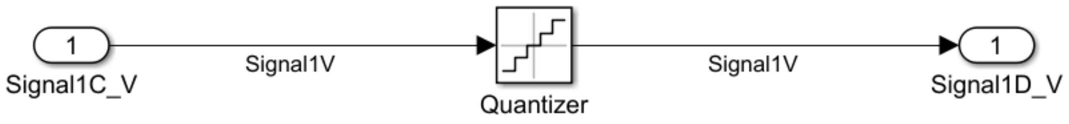


*Figure 66. Quantizer*

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:

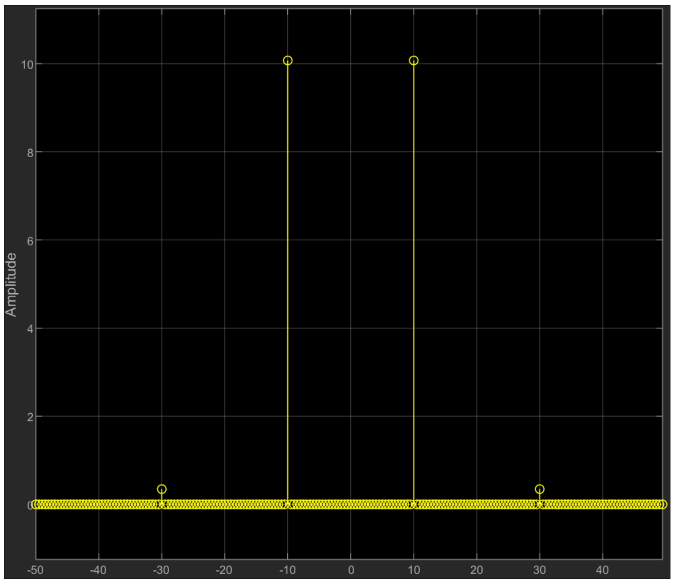


*Figure 67. FFT Output with 4-bit quantization*

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

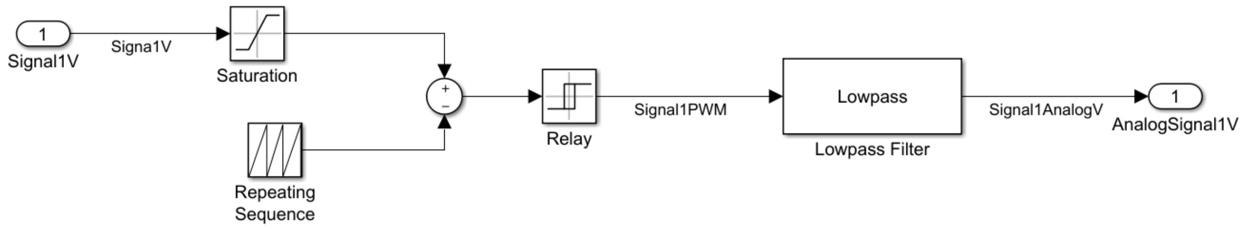


*Figure 68. PWM Design*

Results after reconstruction of the Signal1:

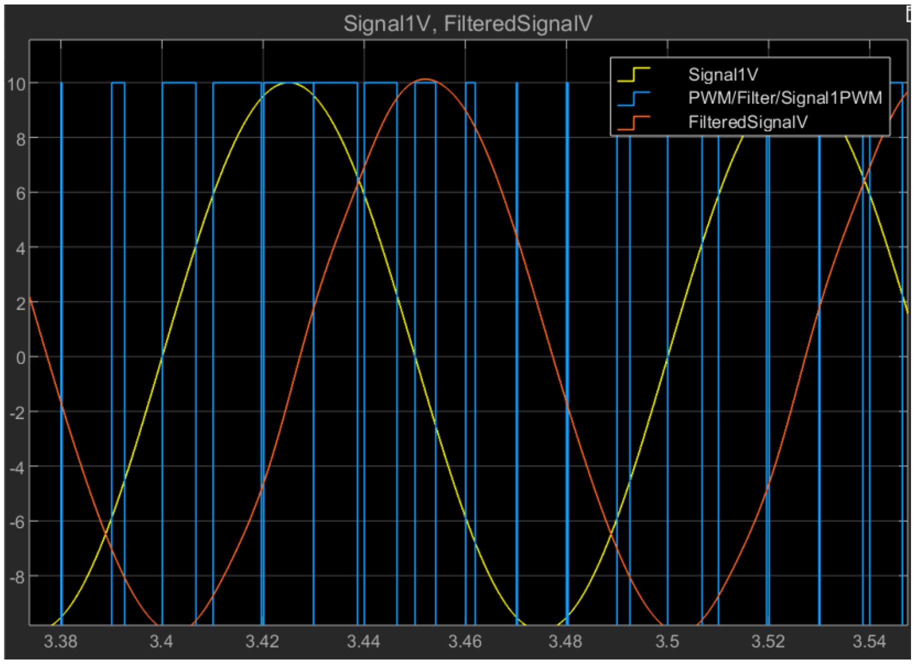


*Figure 69. PWM Results*

### 2.0.7. HMI

Components:

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

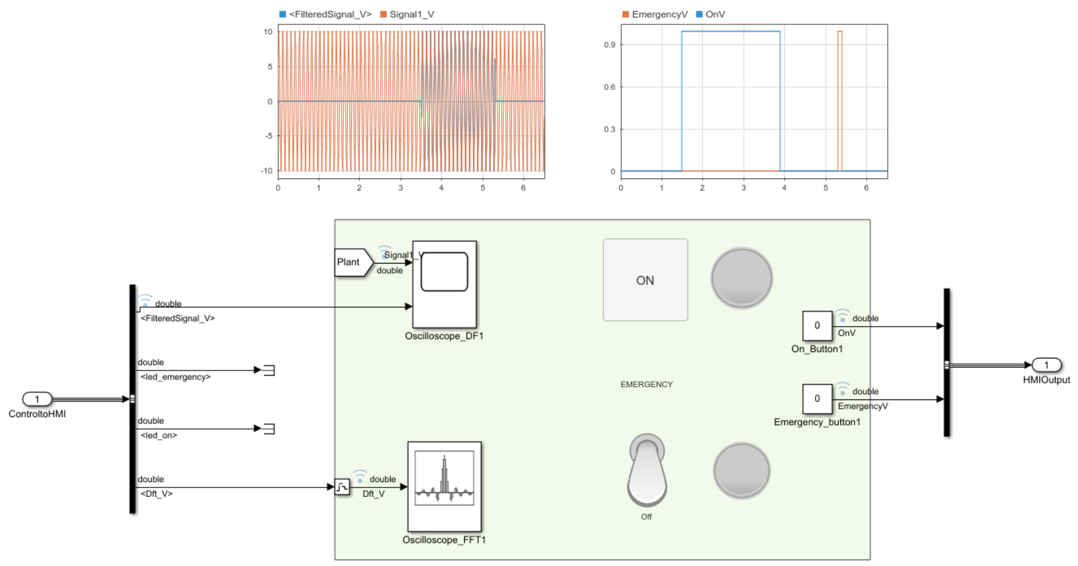


*Figure 70. HMI Design*

HMI feedback:

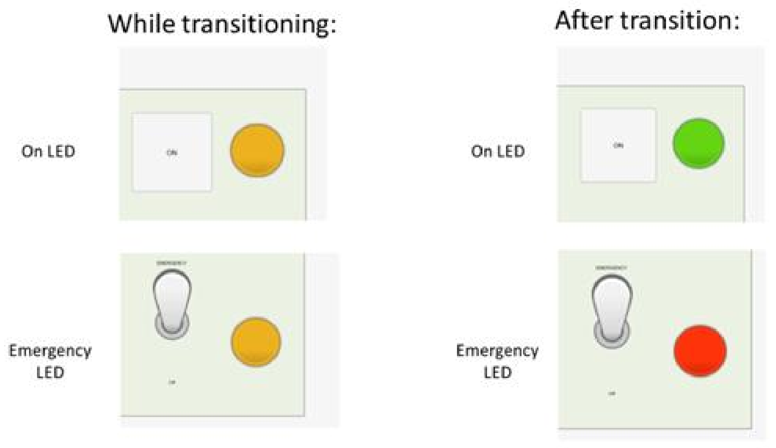
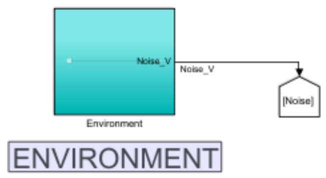
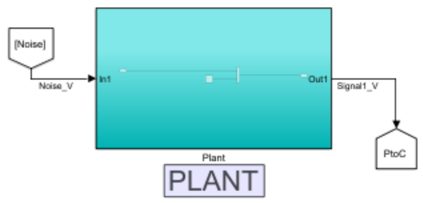
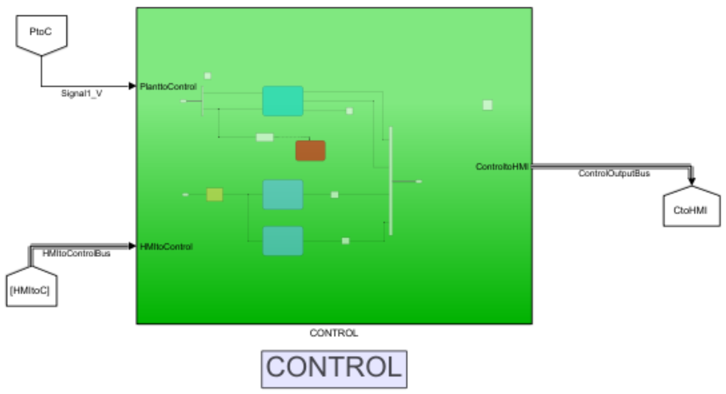


*Figure 71. HMI Feedback*


Noise_V
Noise_V
[Noise]
Environment
ENVIRONMENT


[Noise]
Noise_V
In1
Out1
Signal1_V
PtoC
Plant
PLANT


PtoC
Signal1_V
PlanttoControl
HMItoControl
HMItoControlBus
[HMItoC]
ControltoHMI
ControlOutputBus
CtoHMI
CONTROL
CONTROL

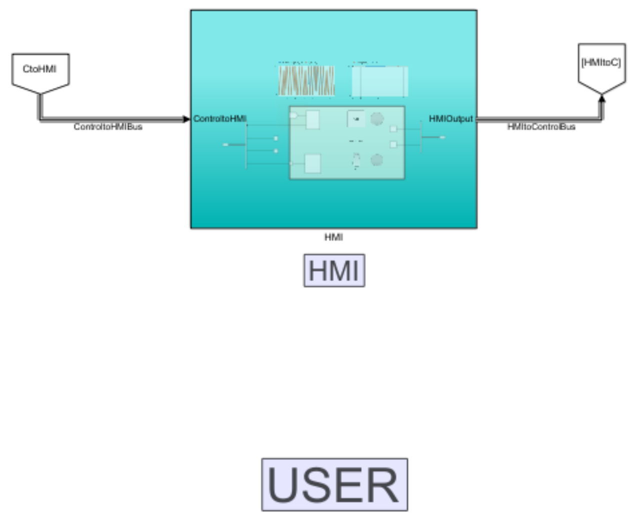


*Figure 72. Simulink model architecture view*

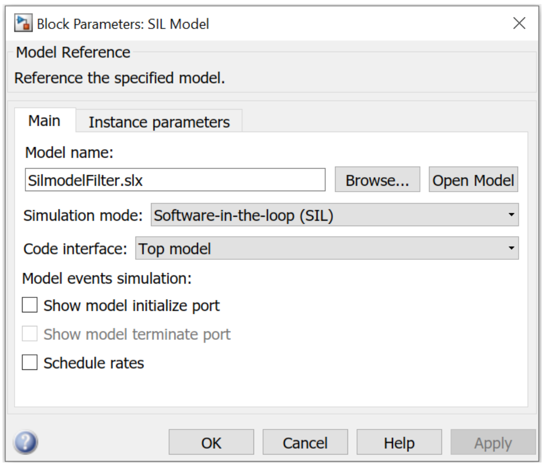
## 2.1. Rapid control prototyping

Since, the rapid control prototype was not essential for the use of this simple example so instead of testing it on a real prototype we resolved to using software only approach.

## 2.2. Production Model Testing

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. (Adelinde M. Uhrmacher, 2009)

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


Block Parameters: SIL Model

Model Reference

Reference the specified model.

Main | Instance parameters

Model name: SilmodelFilter.slx

Simulation mode: Software-in-the-loop (SIL)

Code interface: Top model

Model events simulation:

- [ ] Show model initialize port
- [ ] Show model terminate port
- [ ] Schedule rates

*Figure 73. SIL Settings*

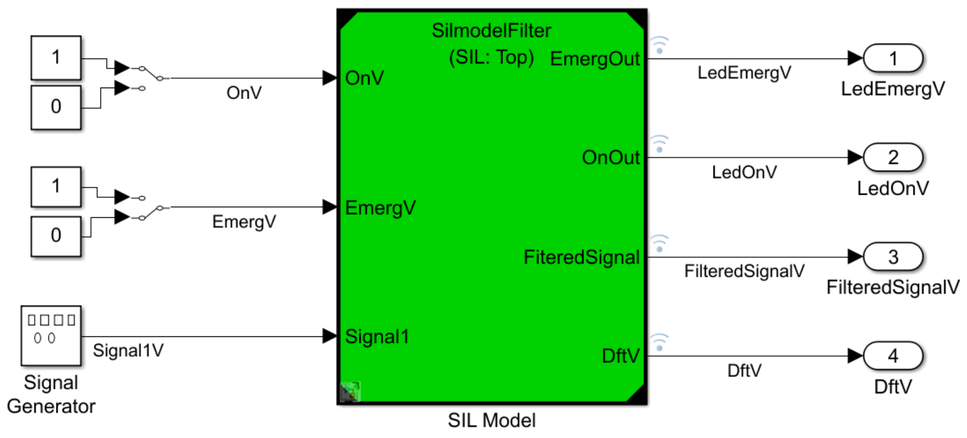


*Figure 74. Simulink model to run SIL*

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.

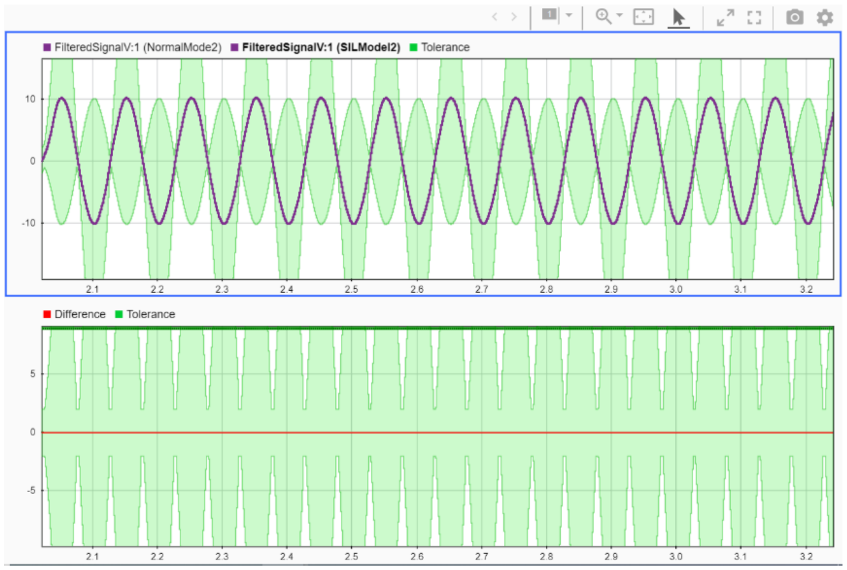
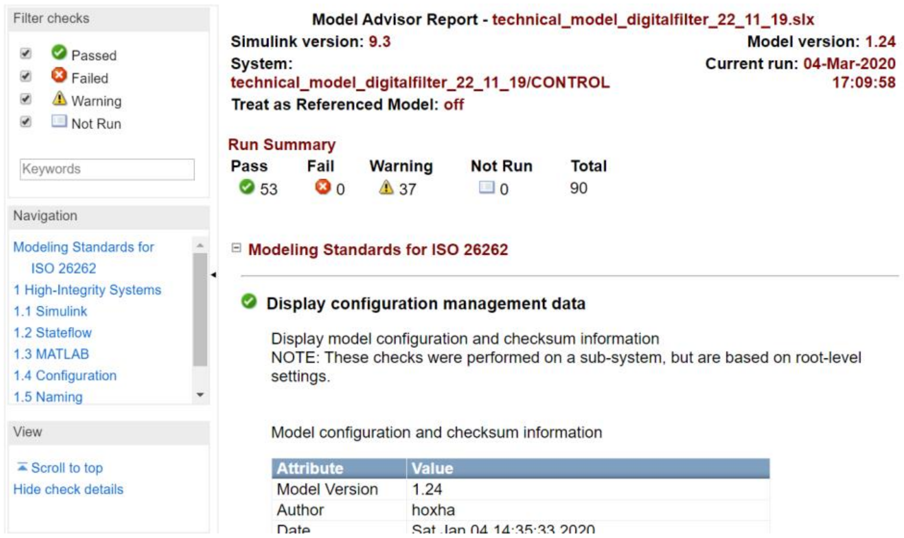


*Figure 75. SIL results*

## Standards check

The results of modelling were satisfactory and failed none of the standards or guidelines. But there is room to improve the *Warning*s 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:



**Model Advisor Report - technical_model_digitalfilter_22_11_19.slx**

**Simulink version:** 9.3
**Model version:** 1.24
**System:** technical_model_digitalfilter_22_11_19/CONTROL
**Current run:** 04-Mar-2020 17:09:58
**Treat as Referenced Model:** off

**Run Summary**

| Pass | Fail | Warning | Not Run | Total |
|---|---|---|---|---|
| 53 | 0 | 37 | 0 | 90 |

**Modeling Standards for ISO 26262**

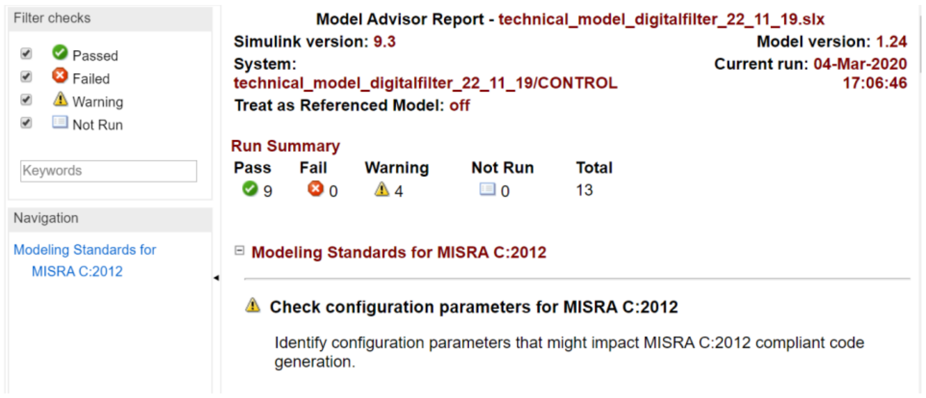
**Display configuration management data**

Display model configuration and checksum information
NOTE: These checks were performed on a sub-system, but are based on root-level settings.

Model configuration and checksum information

| Attribute | Value |
|---|---|
| Model Version | 1.24 |
| Author | hoxha |
| Date | Sat Jan 04 14:35:33 2020 |

*Figure 76. ISO 26262 check results*


Filter checks

✓ Passed
✓ Failed
✓ Warning
✓ Not Run

Keywords

Navigation

Modeling Standards for MISRA C:2012

**Model Advisor Report - technical_model_digitalfilter_22_11_19.slx**

**Simulink version:** 9.3 **Model version:** 1.24

**System:** technical_model_digitalfilter_22_11_19/CONTROL **Current run:** 04-Mar-2020 17:06:46

**Treat as Referenced Model:** off

**Run Summary**

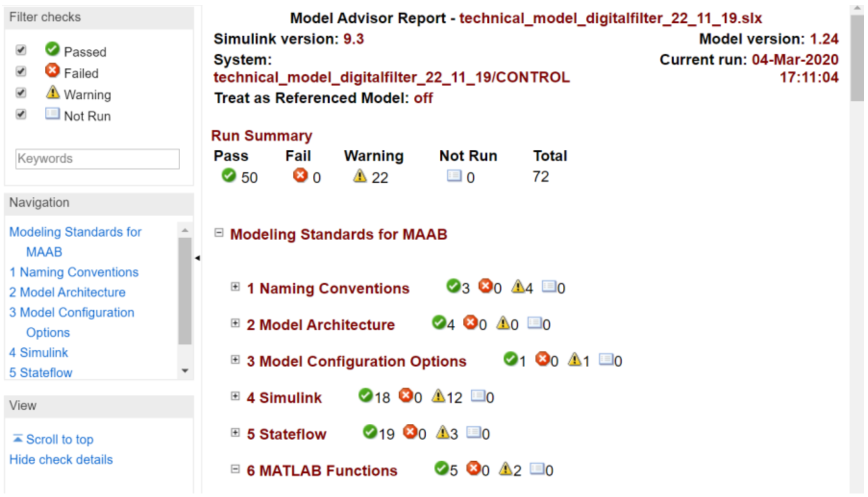
| Pass | Fail | Warning | Not Run | Total |
|---|---|---|---|---|
| 9 | 0 | 4 | 0 | 13 |

**Modeling Standards for MISRA C:2012**

**Check configuration parameters for MISRA C:2012**

Identify configuration parameters that might impact MISRA C:2012 compliant code generation.

*Figure 77. MISRA C:2012 check results*


Filter checks

✓ Passed
✓ Failed
✓ Warning
✓ Not Run

Keywords

Navigation

Modeling Standards for MAAB
1 Naming Conventions
2 Model Architecture
3 Model Configuration Options
4 Simulink
5 Stateflow

View

Scroll to top
Hide check details

**Model Advisor Report - technical_model_digitalfilter_22_11_19.slx**

**Simulink version:** 9.3 **Model version:** 1.24

**System:** technical_model_digitalfilter_22_11_19/CONTROL **Current run:** 04-Mar-2020 17:11:04

**Treat as Referenced Model:** off

**Run Summary**

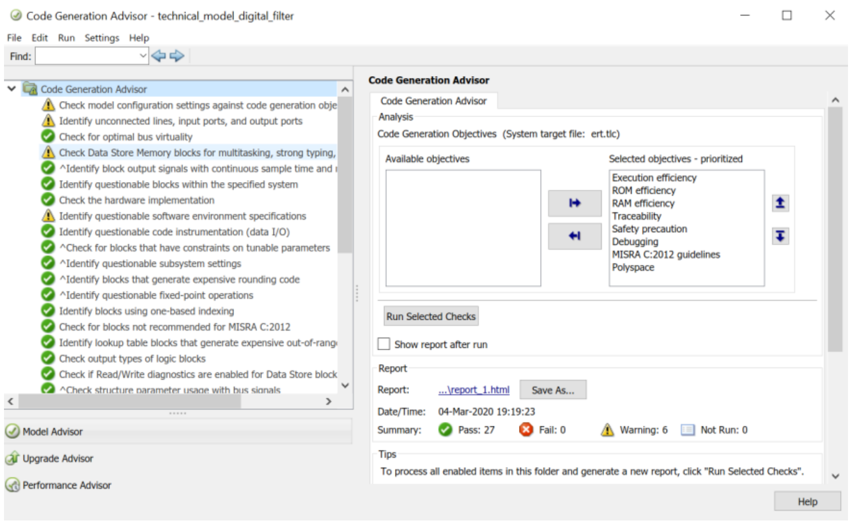
| Pass | Fail | Warning | Not Run | Total |
|---|---|---|---|---|
| 50 | 0 | 22 | 0 | 72 |

**Modeling Standards for MAAB**

| | Pass | Fail | Warning | Not Run |
|---|---|---|---|---|
| 1 Naming Conventions | 3 | 0 | 4 | 0 |
| 2 Model Architecture | 4 | 0 | 0 | 0 |
| 3 Model Configuration Options | 1 | 0 | 1 | 0 |
| 4 Simulink | 18 | 0 | 12 | 0 |
| 5 Stateflow | 19 | 0 | 3 | 0 |
| 6 MATLAB Functions | 5 | 0 | 2 | 0 |

*Figure 78. MAAB guidelines check results*

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



Code Generation Advisor - technical_model_digital_filter

Code Generation Advisor
- ⚠ Check model configuration settings against code generation obje
- ⚠ Identify unconnected lines, input ports, and output ports
- ✓ Check for optimal bus virtuality
- ⚠ Check Data Store Memory blocks for multitasking, strong typing,
- ✓ ^Identify block output signals with continuous sample time and r
- ✓ Identify questionable blocks within the specified system
- ✓ Check the hardware implementation
- ⚠ Identify questionable software environment specifications
- ✓ Identify questionable code instrumentation (data I/O)
- ✓ ^Check for blocks that have constraints on tunable parameters
- ✓ ^Identify questionable subsystem settings
- ✓ ^Identify blocks that generate expensive rounding code
- ✓ ^Identify questionable fixed-point operations
- ✓ Identify blocks using one-based indexing
- ✓ Check for blocks not recommended for MISRA C:2012
- ✓ Identify lookup table blocks that generate expensive out-of-rang
- ✓ Check output types of logic blocks
- ✓ Check if Read/Write diagnostics are enabled for Data Store block
- ✓ ^Check structure parameter usage with bus signals

**Code Generation Advisor**

Code Generation Advisor

Analysis

Code Generation Objectives (System target file: ert.tlc)

| Available objectives | Selected objectives - prioritized |
|---|---|
| | Execution efficiency<br>ROM efficiency<br>RAM efficiency<br>Traceability<br>Safety precaution<br>Debugging<br>MISRA C:2012 guidelines<br>Polyspace |

☐ Show report after run

Report

Report: ...\report_1.html

Date/Time: 04-Mar-2020 19:19:23

Summary: ✓ Pass: 27 ✗ Fail: 0 ⚠ Warning: 6 Not Run: 0

Tips

To process all enabled items in this folder and generate a new report, click "Run Selected Checks".

*Figure 79. Code generation advisor check results*

# Chapter 2

## 1.0. Customer Requirements

Develop a portable product to attach to the vehicle's battery to predict state of health of the battery.

Note: the solution of this problem will be found by applying hybrid V-cycle strategy to determine the efficiency of the proposed procedure.

## Background

The evolution of automotive industries has become apple of eye nowadays, the eyes of world is directed towards power production and power storage. Batteries are the most common source of power storage. There are many types of batteries, small cells to large battery packs used to run whole offices. These batteries are charged and recharged many times during their lifetime.

Here we are not going to discuss about how battery works and what are the factors involved in working of the batteries. But we are more interested in predicting the state of health and state of charge of the battery. Since, these two factors are important in predicting the usage of battery, so we should understand them first in order to get an idea how important is our product for present and in the future.

At this very moment, there is no such product in the market that has the same functionalities. This product took an inner view into the study of batteries. Also, it processes data and informs the user through a simple interface such as an application on a smart phone. The outcome of the project will be directly applied for the development and marketing of a wide range of aftermarket products for the monitoring of Lead-Acid batteries. Moreover, the know-how and the technologies developed can be transferred to other types of batteries, such as Lithium Ion (LION), and therefore applicable to the technological development agenda foreseen for the next few years, regarding hybrid/fully electric vehicles (HBEV and BEV) with optimal management of co-generation micro-networks (photovoltaic, wind… etcetera).

The aim of the model discussed below is to fully develop a battery State of Health (SOH) detection algorithm and implementing it in real time, afterwards, connect it to the device under management in order to observe and compile the data, and finally inform the user with various information about the battery under inspection specifically the battery State of Charge and State of Health. Before the discussion of model which helps us to predict the state of health of the battery, we should understand what State of health and other important factors about it is.

## State of health of battery

The state of health (SOH) of a battery is a ‘figure of merit’ to describe the battery condition, and how much of useful lifetime is remaining from the battery, thus, if you know state of health of your battery you would have ability to know whether your battery needs repairs or even replacement, or is it functioning with the optimum performance. It is calculated in percentage points ‘%’.

A 100% State of health of battery means that this battery is brand new (matching the specifications of the battery) and was not used before. Accordingly, a used battery has a decreasing state of health over time due to the battery decaying. The battery decaying happens due to chemical decomposition inside the battery due to cyclic charging and discharging of battery. A healthy battery behaves in a way that the ions flow freely from the anode to the cathode, and when the battery is put to a load (withdrawing electric current), the ion flows in the reverse direction.

This process tears down the cathode and wears it by time, thus it means reduced battery capacity because less ions are now flowing from the anode to the cathode and vice versa. Nevertheless, battery state of health (SOH) is not provided by battery manufacturers because in its current condition it is know that it has a (100%) state of health, so this concept applies only when a battery is put into use and start withdrawing electric current. Moreover, the state of health of the battery can be applied to each cell of the battery in which in the end gives the overall value. And since every battery has different specifications and behaves differently, a battery state of health is measured differently amongst different batteries because it is a reference to the battery at its optimum performance which are different. So, the factors must contain previous knowledge of the battery states or performance in order to be able to determine the new state of health of a battery.

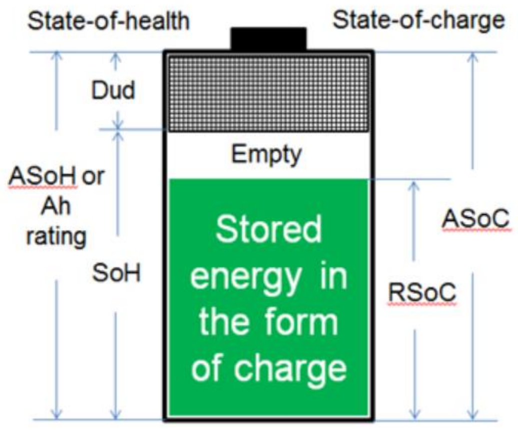


*Figure 80. Battery state*

As explained in the figure the state of health rated is different from the state of health at the current time, because of the degradation of the battery.

## State of health importance

As for a normal battery, a low state of charge is of no concerns in the terms of maintenance and replacement, as the battery will be subjected to the charging process, thus it will be able to provide the energy needed. But, a battery with a low State of health would require repairs or even replacements. As for any battery, there are 3 phases of working, which are:

**a. Service zone**

Which is the zone where the battery provides its rated capacity, thus providing fully the energy required.

**b. Replacement zone**

The zone where the battery starts degrading and the cathode is wearing out, which means that the battery is not providing its rated capacity. So, a battery in replacement zone will require either maintenance or replacement of the cathode or to be subjected to the necessary type of maintenance in order to retain it back in the Service zone.

**c. Failure zone**

Once you enter the failure zone, the battery is known as 'failed'. Which means that the battery behavior cannot be predicted, and the battery is either not functioning or requires replacement. This phase is very dangerous for many types of applications as they required fixed voltage and current input. So, this phase must be avoided, and the battery should be subjected to direct maintenance or repairs.

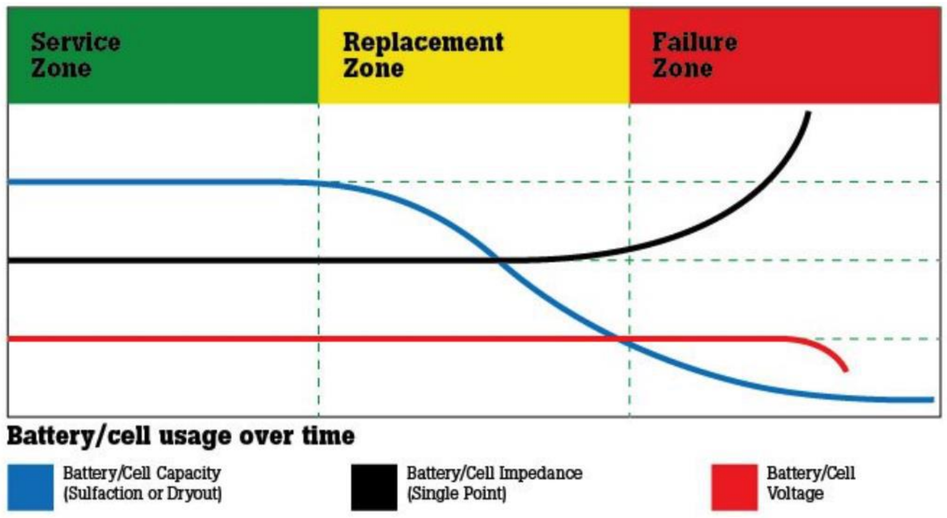


*Figure 81 Battery cell usage*

Hybrid V-cycle

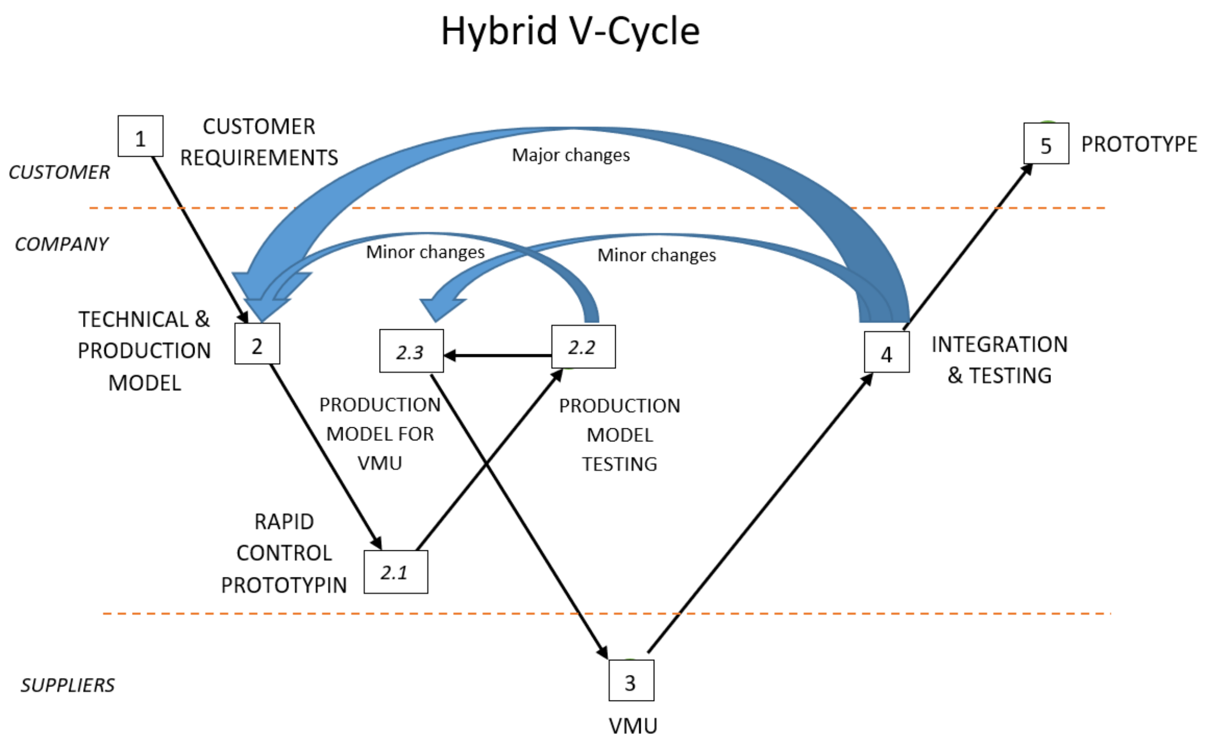


*Figure 82 Hybrid V-cycle*

### 1.1. Requirements

The customer provided us with Simulink model which predicts SOH of the battery with a certain precision for only two windows (the word window here means a certain time length in which signal has been divided, by this we can compare two adjacent windows to predict). Following are the customer requirements:

*Table 6 Customer Requirements*

| **Function 1** | **BATMAN Application** |
|---|---|
| Requirement 1-1 | Must calculate SoH |
| Requirement 1-2 | Predict whether Soh greater or less than 40% |
| Requirement 1-3 | Reduce buffer size |
| Requirement 1-4 | Reduce code memory size |
| **Function 2** | **Supervisory Control** |
| Requirement 2-1 | Start/Stop push button must be added (Mobile App) |

| | |
|---|---|
| Requirement 2-2 | Should print text for change in battery |
| **Function 3** | **Bluetooth transmission** |
| Requirement 3-1 | Outputs must be transmitted every time we have a result |

## 1.2. Simulink model provided by the customer

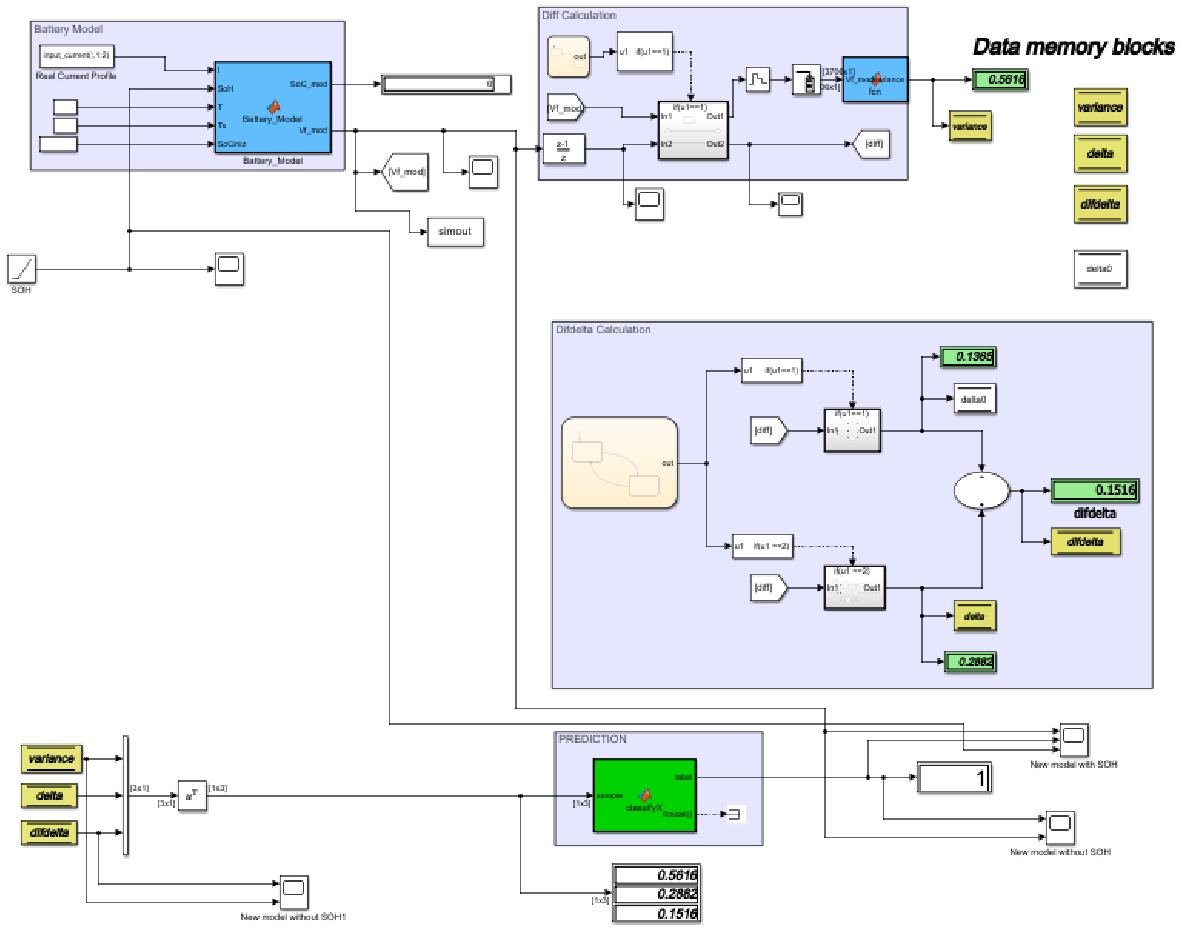


*Figure 83 Simulink model*

Above figure shows the Simulink model provided by the customer. We run this model as it is to see the results and then try to improve it as required by the customer.

## Results

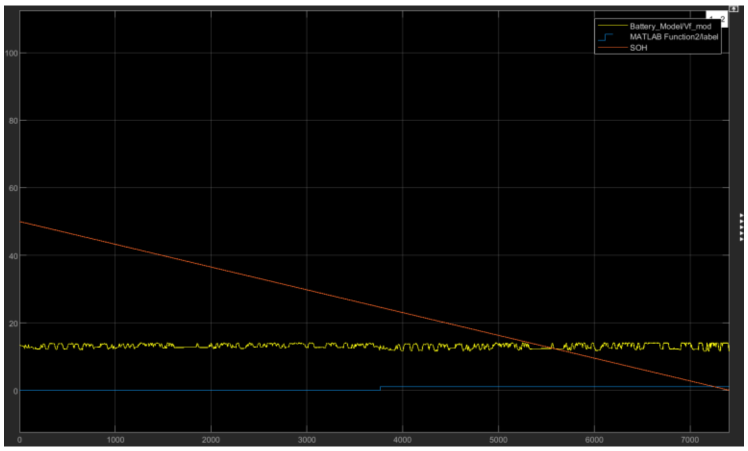


*Figure 84 Simulink Model Results*

The above figure shows the results of the model provided by the customer.

In the following description, we will discuss the model, in order to understand it better.

### *1.2.1 Model description*

We can divide the model in the following parts in order to understand it better. We are not going to change some parts of the model since they are the basic components. After changing the model, we will compare the results in order to see whether it is performing according to requirements set by the customer.

The model can be divided into mainly three parts.

- Battery model
- Support Vector Machine (SVM)
- Algorithm for SOH prediction

Out of the above-mentioned things we will change Algorithm for the prediction of the SOH. The battery model mimics the real battery conditions while the support vector machine predicts when the battery is degraded to such an extent that we need replacement.

### *1.2.2. Algorithm for SOH prediction*

The battery model mentioned above was used to predict the real battery voltage and the previous work done by my colleagues proved that battery model is a good predictor of battery in real world conditions. From the analysis done on voltage we observe 3 parameters which are defined below:

- Parameter 1 – Delta (voltage difference of battery)

- Parameter 2 – Difference delta (rate of change of voltage of batteries)
- Parameter 3 – Variance of maximum/minimum peaks of voltage

**Parameter 1- Delta**

The analysis done by company shows that the change in peak to peak voltage is nearly constant above 40% when the state of health of battery is above 40% while below it we can detect a slope which shows that the difference between peak to peak voltage is increasing.

**Parameter 2 Difference delta**

The other interesting feature of voltage of battery found by the company was that if we study the rate of change of voltage of battery when it is below 40% then we can see that it starts to increase. Which means that as SOH of the battery decreases the rate at which the voltage changes increases. It serves as the second indicator of the deteriorating state of health of the battery. But this parameter only gives the desired results when we have high initial state of charge of battery.

**Parameter 3 Variance of maximum/minimum peaks of voltage**

Relation between variance with respect to the SOH, this parameter shows an increasing trend with the decrease of the battery state of health throughout whole experiment. This trend was interesting to us as a factor that could be fed to the estimating algorithm, in the sense that:

$$Vpeaktopeak \propto 1/Variance$$

Hence, knowing the Variance over a window of time will help us understand the SOH of the battery, as the following cases applies:

- Variance is constant over time, which is a representation that battery SOH is maintained at same level, not increasing or decreasing but maybe there is a slight change which is undetectable
- Variance is decreasing over a window of time, which means that the battery life is degrading rapidly, and actions should be taken
- Third case which the variance will be decreasing over time is NOT applicable. Because it means that the battery health is automatically regaining over time which is practically impossible without any intervention.

Note that this parameter like the previous rate of change of voltage gives good results only when the initial state of charge of the battery is high.

The above three parameters are used in the Simulink model in order to predict the state of health of the battery. The above discussion gives us a brief overview of the algorithm the results of these parameters are fed into the SVM to predict SOH of the battery. In the following topics we will discuss how we are applying our Hybrid V-cycle for developing this model.

## 2.0. Technical and production model

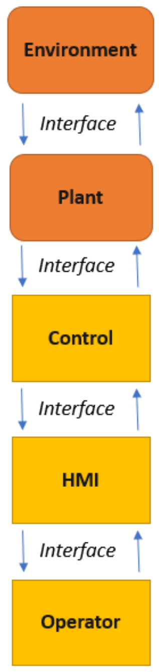
The second step in our hybrid V-cycle is to develop a technical model considering the requirements of our customer and then transform it into production model. In the first step we took the model from the customer and note down the requirements. The first step is important to understand what customer wants. This step is the base of our technical model.

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.

### 2.0.1. Technical model

The Simulink model for the prediction of state of health of the battery is as follows. It is divided in following parts according to the ''Hybrid V-cycle Methodology'' developed in our company in order to develop every project following some standardized procedure which makes it streamlined and easy for other persons who will work on this project to understand.

Simulink (2° step)



In our model, since we are using the battery model in Simulink, so we have all the blocks in software. In real world environment for this project, the environment software block will be replaced by real environment while the software block for the plant will be replaced by battery. According to the above figure we can arrange our Simulink model in the following given way:

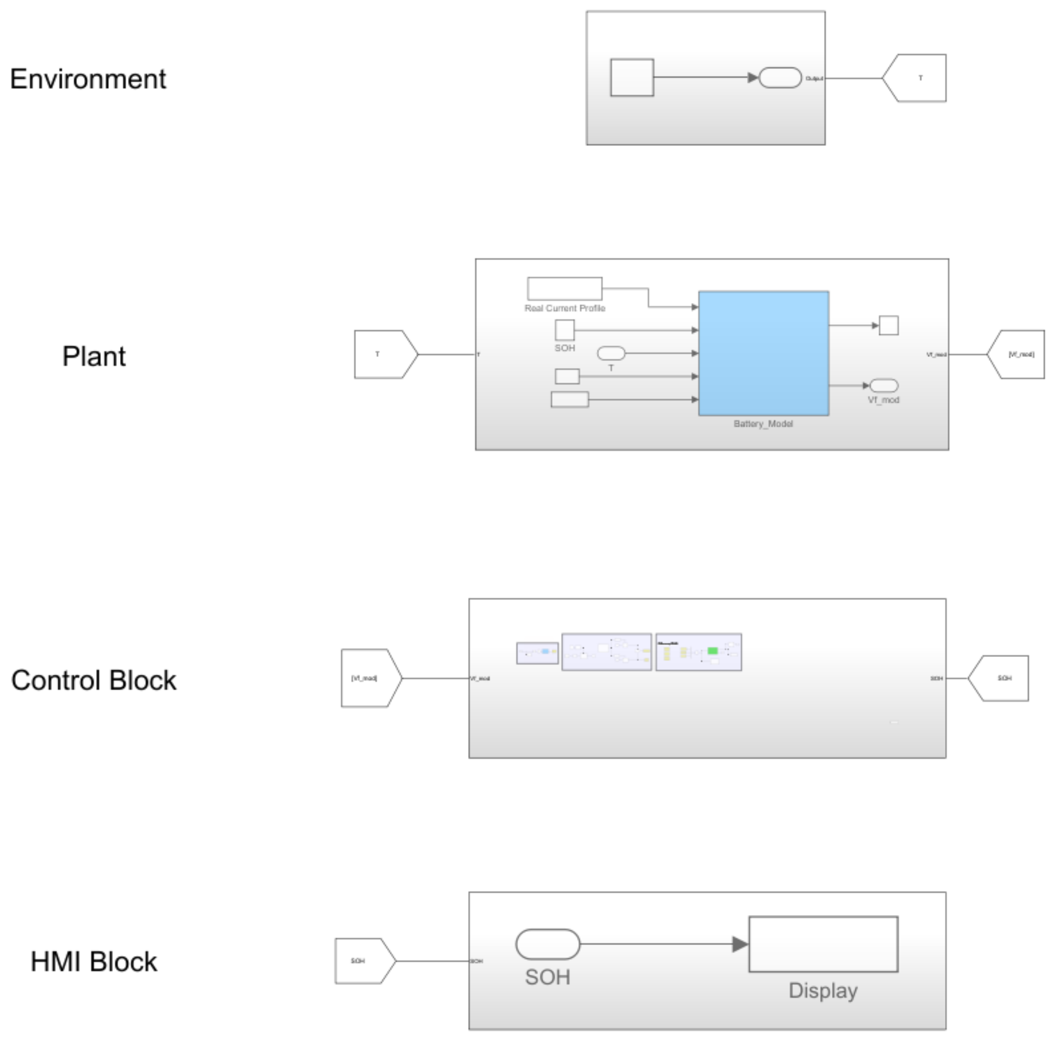


*Figure 85. Simulink Model in the wake of Hybrid V-cycle methodology*

Note that in this Hybrid V-cycle methodology user is not shown here which is the owner of this device. In the following passages we will take a brief overlook of the blocks present.

### 2.0.1.1. Environment

This block represents the things which are affecting our electronic circuits due to which they are unable to predict the value accurately. The major factor is temperature which is affecting the performance of electric circuits. If the change in temperature is within certain limits than, as communicated by the manufacturer of the IC's, our results are accurate. In real circuit we will have the voltage and current from the battery but in this case since we are working on the Simulink so we have to use a battery model which accurately predicts this. For this battery model we are placing value of the temperature.

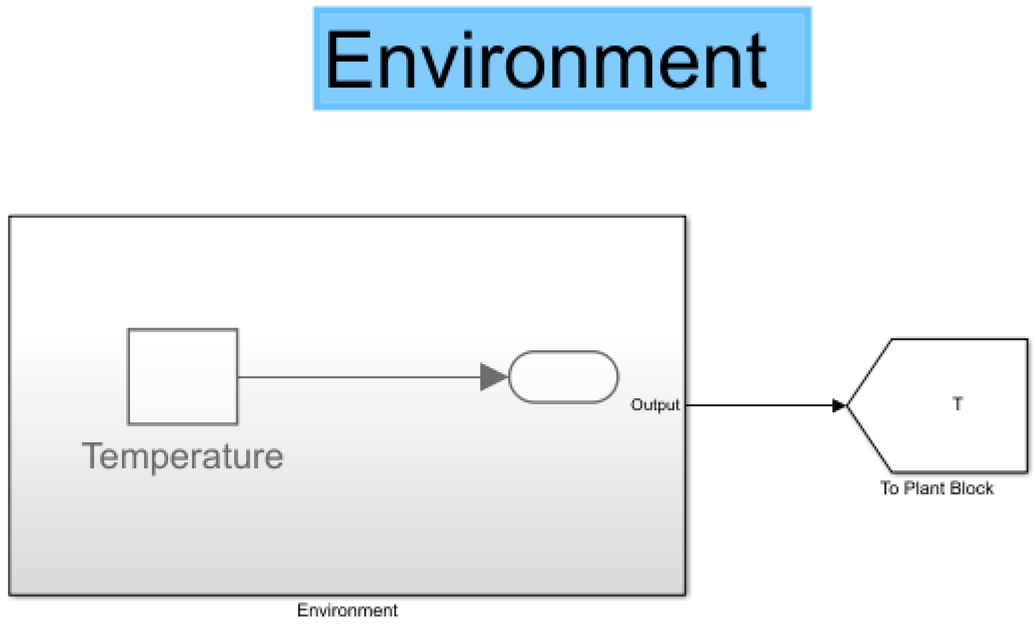


*Figure 86. Environment block*

### 2.0.1.2. *Plant*

This block takes input as temperature from the environment. This block contains the Battery model which mimics the real battery conditions and gives out the voltage of the battery as it's state of health is decreasing. In our control block we are only using the voltage coming from battery model.

The inputs to the battery model are:

- Temperature (from environment)
- State of health (produced by signal generator varying from 100% to 0%)
- Ts (simulation time step)
- SOCinz (initial state of charge)

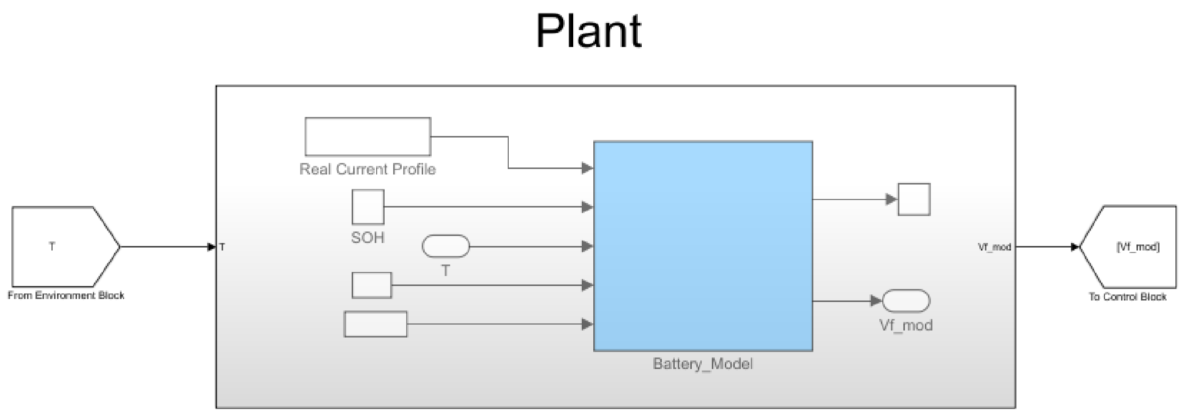


*Figure 87. Plant block*

### 2.0.1.3. Control Block

The input to the control block is voltage coming from the plant. In the control block we are calculating three types of values from that voltage coming from the battery model and giving it to SVM in order to predict the result.

- Variance
- Delta
- Difference between delta
- SVM

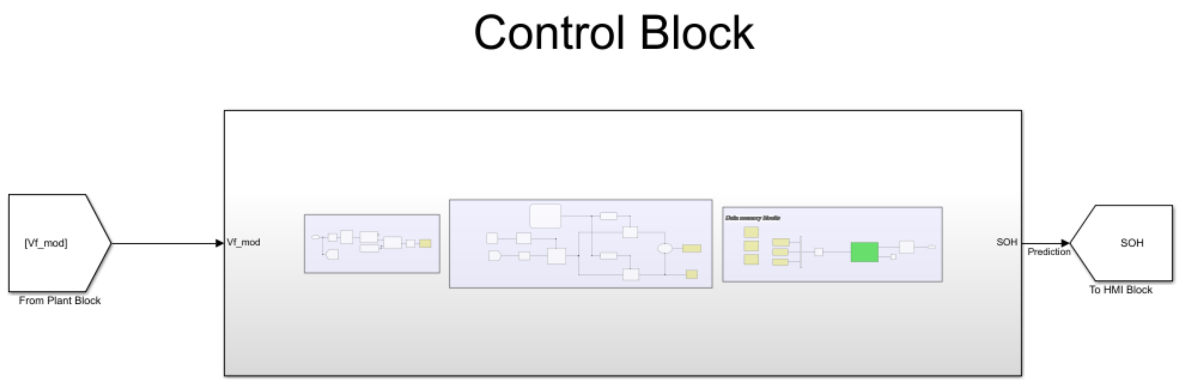


*Figure 88. Control Block*

We can see that this control block has one input which is voltage coming from the battery model while it has one output which is state of health for the human machine interface.

**Variance**

In order to predict the deterioration of state of health for the battery we calculate the variance of peaks of voltage profile. We see that when the state of health of the battery deteriorates the variance of the peaks of voltage profile starts to vary drastically.

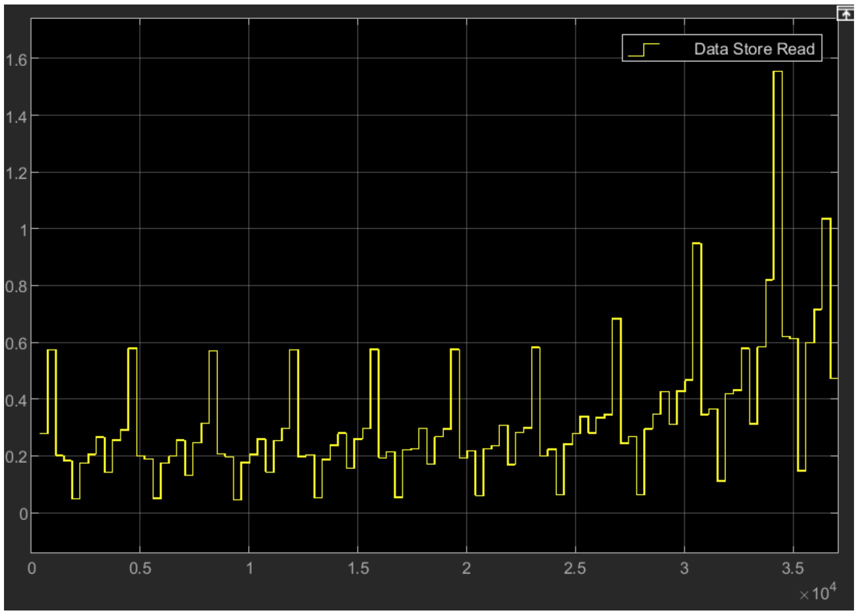


*Figure 89. SoH Deterioration effect on Variance*

As seen in the above figure, the variance starts to vary drastically as we reach state of health of battery below 40%. The calculation of variance is done by following scheme.

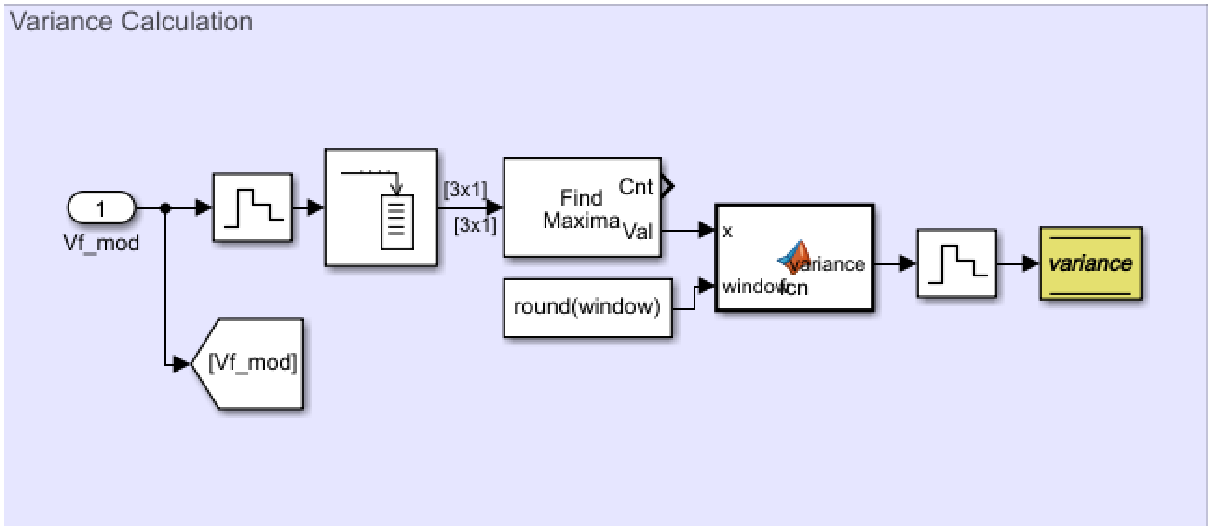


*Figure 90. Simulink Model for Variance*

## Delta

It is calculated by the difference between the maximum and minimum values of the difference between two consecutive values of the battery voltage. First, we find the difference between two values of the voltage input. This is done by using following blocks.

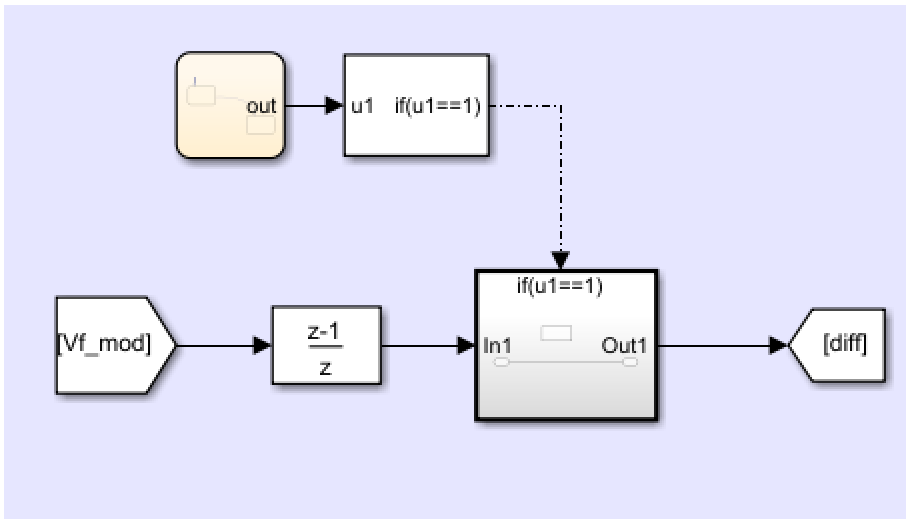


*Figure 91. Simulink Model for difference calculation*

After calculating the difference, we calculate the difference between these values. As, state of health of battery deteriorates the difference between them starts to vary drastically as can be seen in figure 10.

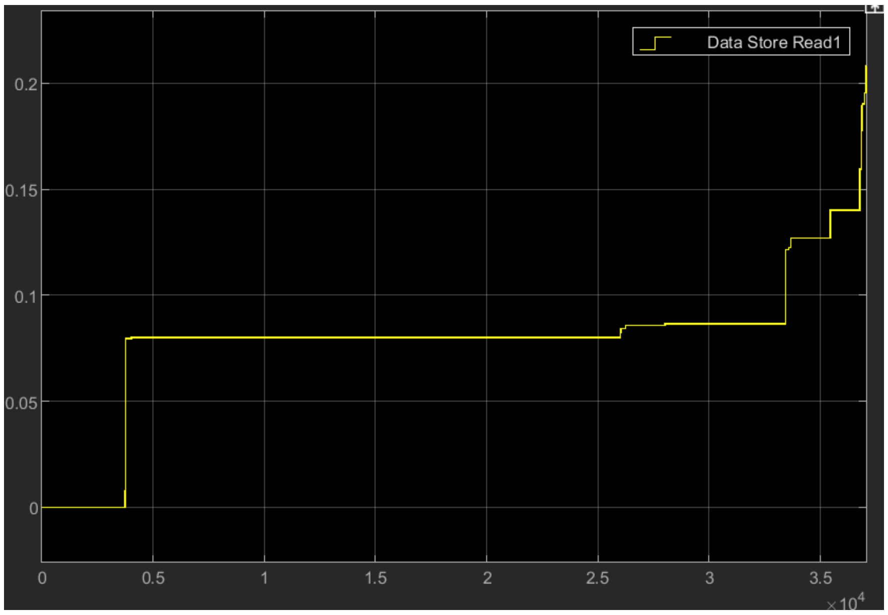


*Figure 92. SoH detrioration affect on Delta*

## Difdelta

The third and last variable is difference between the delta. It gives the same result as both above variables. As the state of health of battery deteriorates, we see that the value of difference delta changes as shown in figure 11.

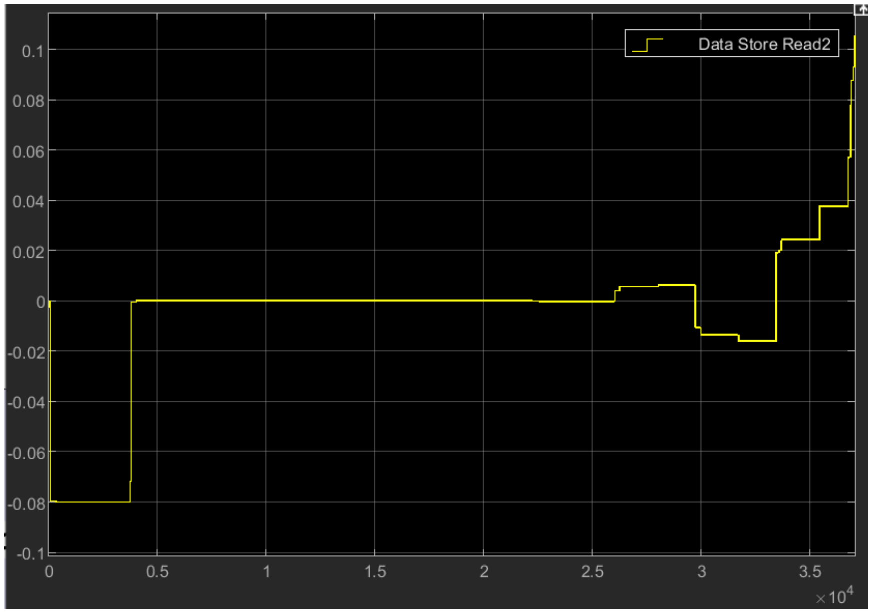


*Figure 93. SoH detrioration effect on Difdelta*

The Simulink blocks used to calculate this are shown in the figure 12.

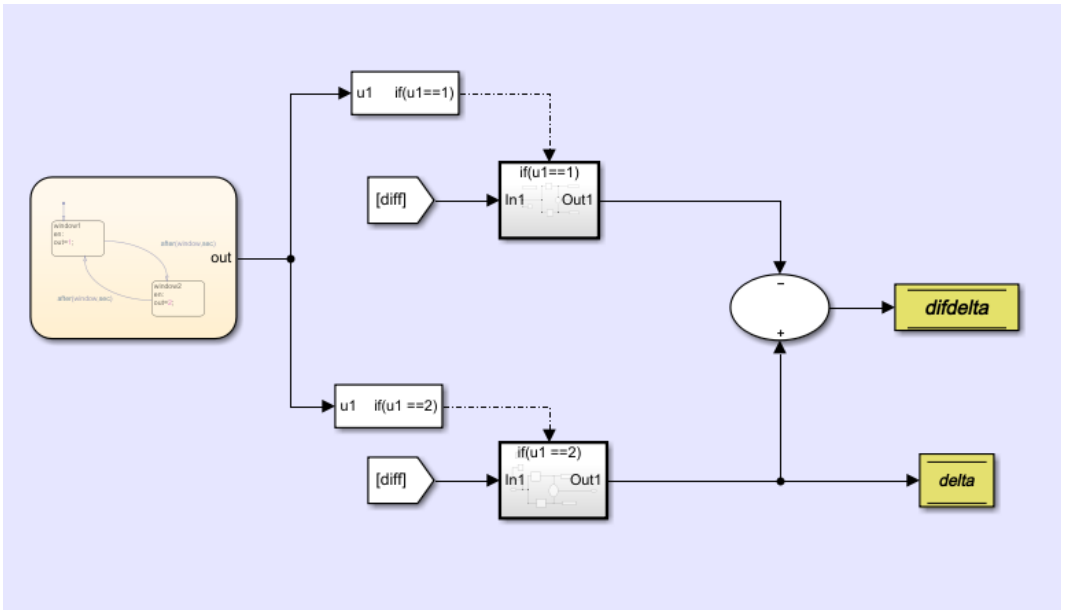


*Figure 94. Simulink model for difdelta and delta calculation*

## Support Vector Machine

We saw how variables namely variance, delta and difference delta vary with change in state of health of the battery. Using this information, we can train our support vector machine to predict for all other data available. The support vector machine gets better in predicting the state of health as more data is fed to it and with the passage of time it's accuracy will increase. The Simulink blocks used for predicting the state of health of  battery is shown in figure 13.

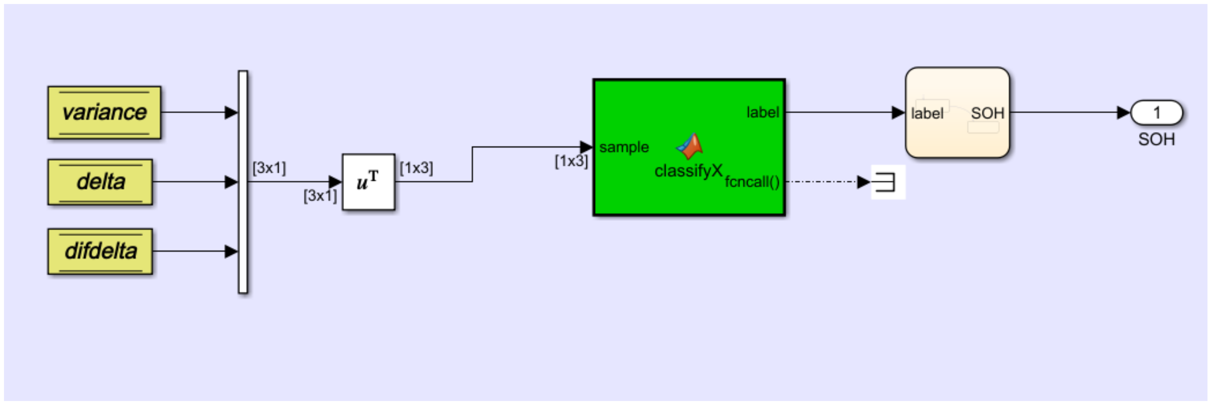


*Figure 95. Support Vector Machine*

### 2.0.1.4. Human Machine Interface Block

This block is used to display the result from the support vector machine. We keep in mind that the algorithm predicts whether state of health of battery is above a certain percentage or not. It doesn’t tell current value rather it tells that the battery can be used further, or we should replace it since, it’s crucial for the car. We have also used an app to communicate with customer to inform him when he should replace the battery. The app tells the state of health in percentage and tells whether the SoH of the battery is greater or less than 40%.

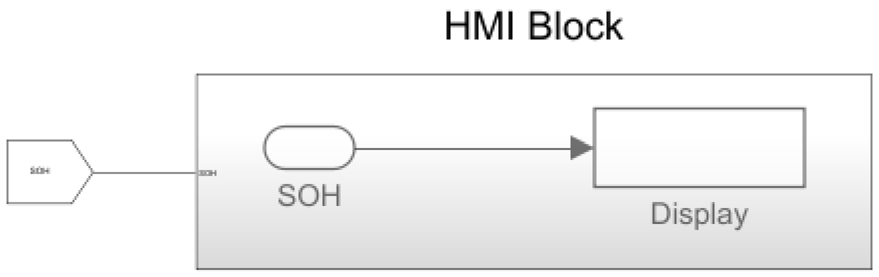


Figure 96. Human Machine Interface Block

### 2.0.2. Comparing technical model results with Customer model

We used 4 different types of real data in order to reproduce the real voltage values for our control block. We used this data to train our support vector machine also and then we used other 4 data to check the efficiency of our algorithm.

### Real_data1

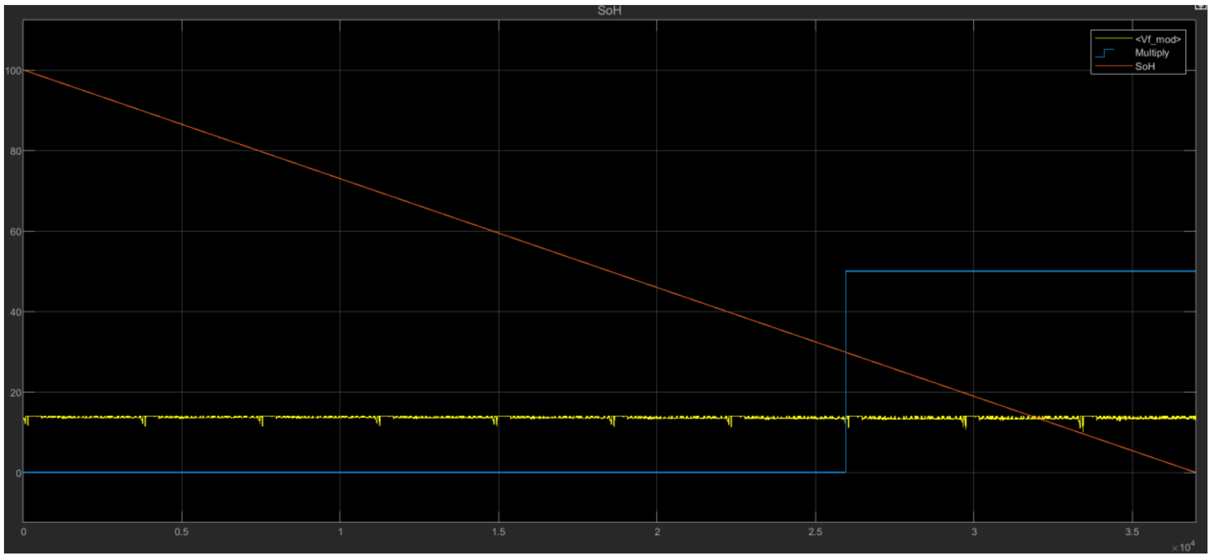
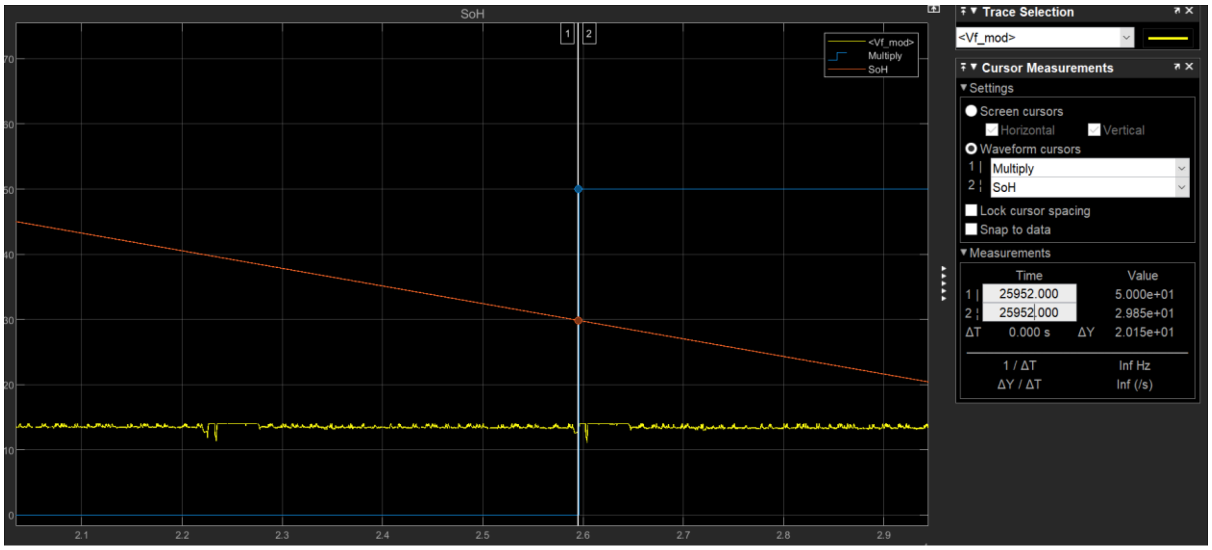


*Figure 97. Real_data1 Results*

The figure 15 shows three values which is State of health (red), voltage from battery model (yellow) and State of health prediction (blue). It can be seen from the figure that when the state of health of the battery decreases below 40% the algorithm perfectly detects the and gives an output in the form of 1 which is multiplied to show it properly on the graph above. The simulation was run for more than 10 hours in the Simulink.



1 | Multiply
2 | SoH

| | Time | | Value |
|---|---|---|---|
| 1 | 25952.000 | | 5.000e+01 |
| 2 | 25952.000 | | 2.985e+01 |
| ΔT | 0.000 s | ΔY | 2.015e+01 |
| 1 / ΔT | | | Inf Hz |
| ΔY / ΔT | | | Inf (/s) |

*Figure 98. Real_data1 SOH*

We can see from Figure 16 that value of SoH is 30% when our algorithm predicts the deterioration of state of health of battery.

## Real_data2

Now, we will reproduce the results by placing second set of data.

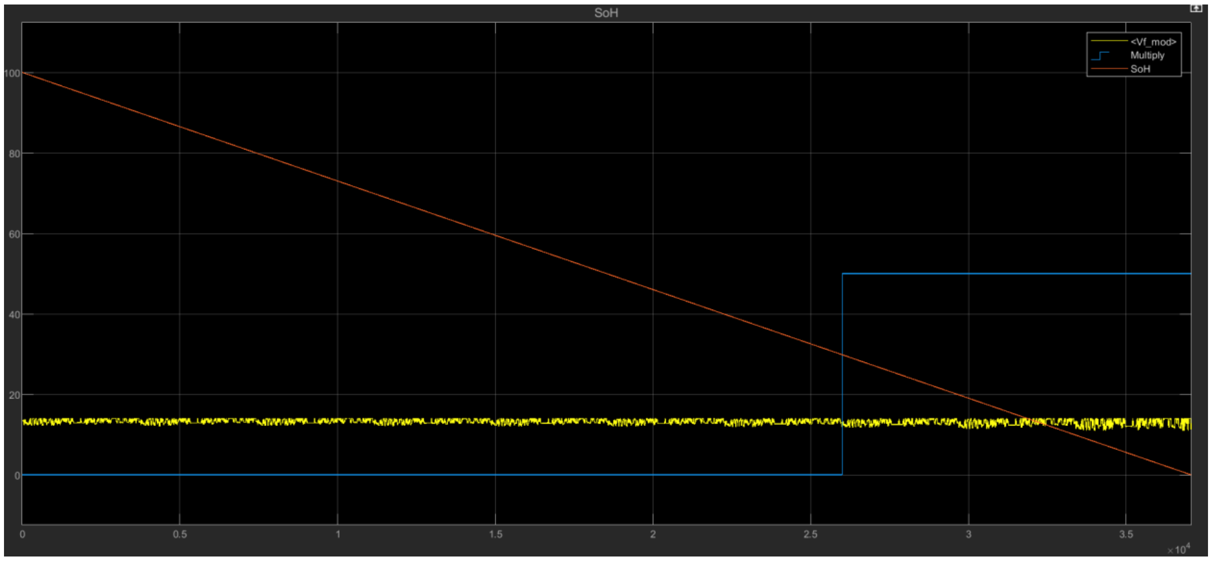
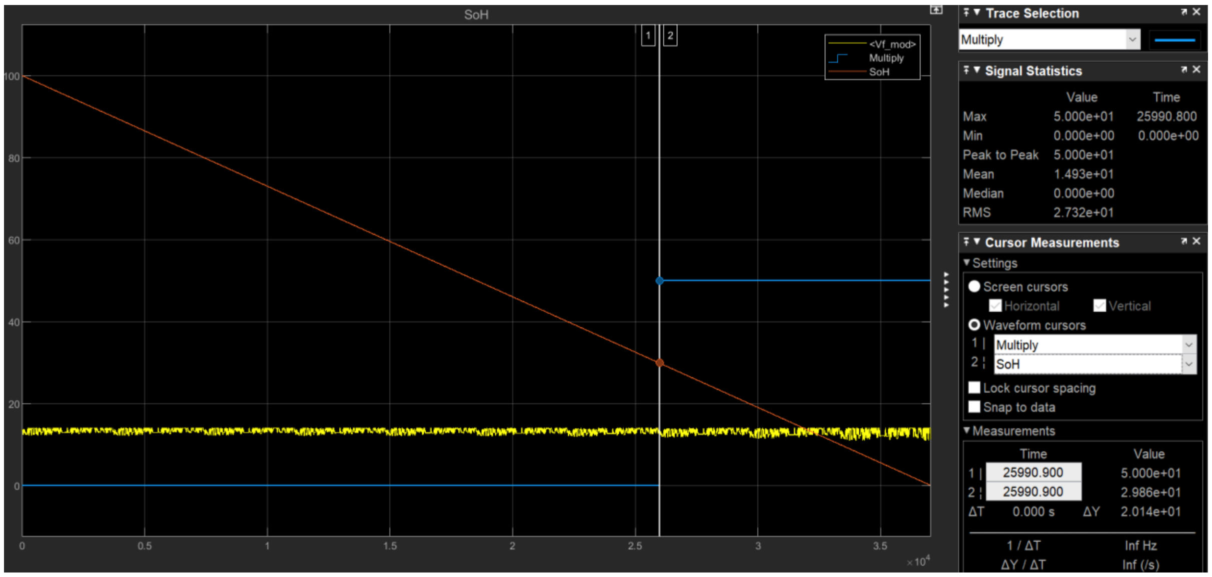


*Figure 99. Real_data2 Results*

As shown in Figure 17, our algorithm predicts the deterioration of state of health of battery when it goes below 40%. The figure below shows that our algorithm predicts when the SoH is at 30% for real data 2.

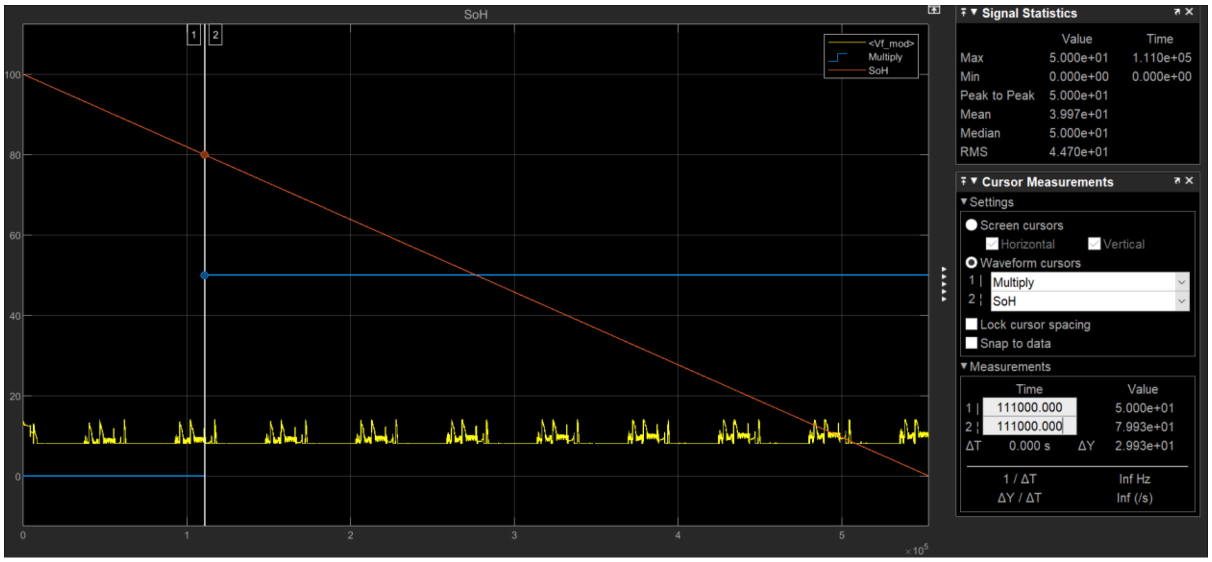


```
Trace Selection
Multiply

Signal Statistics
              Value        Time
Max           5.000e+01    25990.800
Min           0.000e+00    0.000e+00
Peak to Peak  5.000e+01
Mean          1.493e+01
Median        0.000e+00
RMS           2.732e+01

Cursor Measurements
Settings
  Screen cursors
    Horizontal   Vertical
  Waveform cursors
  1  Multiply
  2  SoH
  Lock cursor spacing
  Snap to data
Measurements
        Time              Value
  1     25990.900         5.000e+01
  2     25990.900         2.986e+01
  ΔT    0.000 s     ΔY    2.014e+01

  1 / ΔT                  Inf Hz
  ΔY / ΔT                 Inf (/s)
```

*Figure 100. Real_data2 SoH*

## Real_data3

We are placing real_data3 in our algorithm. As can be seen from the statistics our algorithm predicts that the state of health is below 40% but, it's 80%.

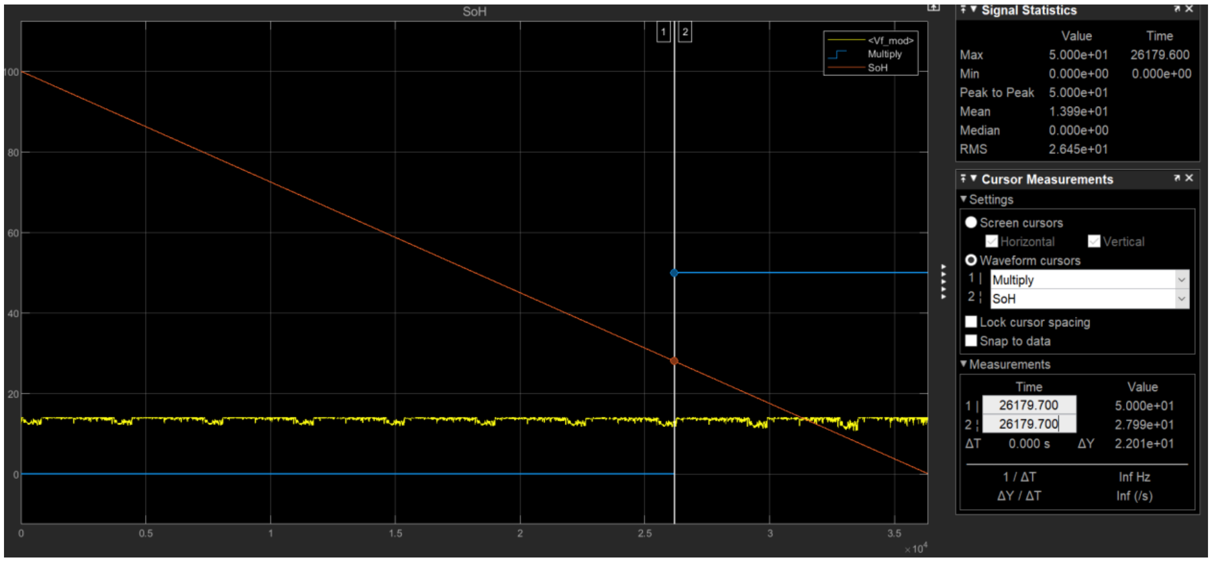


```
Signal Statistics
              Value        Time
Max           5.000e+01    1.110e+05
Min           0.000e+00    0.000e+00
Peak to Peak  5.000e+01
Mean          3.997e+01
Median        5.000e+01
RMS           4.470e+01

Cursor Measurements
Settings
  Screen cursors
    Horizontal   Vertical
  Waveform cursors
  1  Multiply
  2  SoH
  Lock cursor spacing
  Snap to data
Measurements
        Time              Value
  1     111000.000        5.000e+01
  2     111000.000        7.993e+01
  ΔT    0.000 s     ΔY    2.993e+01

  1 / ΔT                  Inf Hz
  ΔY / ΔT                 Inf (/s)
```

*Figure 101. Real_data3 Results*

## Real_data4

As shown by the figure 20, the algorithm predicts accurately state of health of the battery. The state of health of battery is 30% when our algorithm predicts the deterioration of SoH below our marked threshold.



Signal Statistics

| | Value | Time |
|---|---|---|
| Max | 5.000e+01 | 26179.600 |
| Min | 0.000e+00 | 0.000e+00 |
| Peak to Peak | 5.000e+01 | |
| Mean | 1.399e+01 | |
| Median | 0.000e+00 | |
| RMS | 2.645e+01 | |

Cursor Measurements

Settings

Screen cursors
Horizontal ✓ Vertical ✓
[✓ Waveform cursors]
1 Multiply
2 SoH
Lock cursor spacing
Snap to data

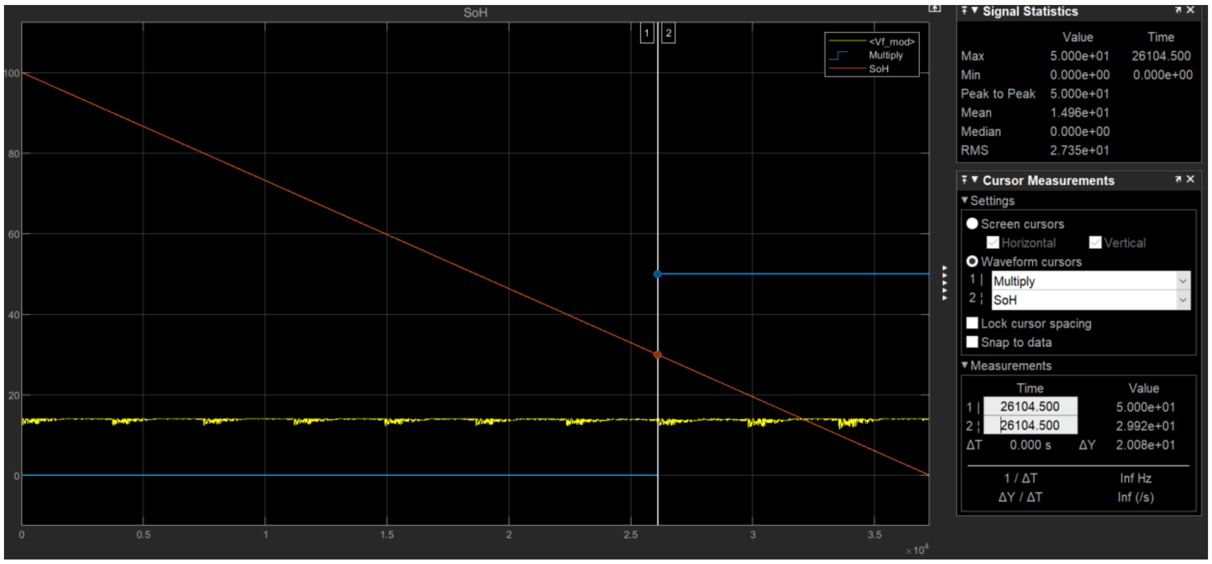
Measurements

| | Time | | Value |
|---|---|---|---|
| 1 | 26179.700 | | 5.000e+01 |
| 2 | 26179.700 | | 2.799e+01 |
| ΔT | 0.000 s | ΔY | 2.201e+01 |
| | 1 / ΔT | | Inf Hz |
| | ΔY / ΔT | | Inf (/s) |

*Figure 102. Real_data4 Results*

## Real_data5

When we run our program with input as Real_data5, our algorithm predicts accurately the deterioration of the battery. Algorithm gives 1 (which is multiplied by 50 for reading data easily) when the SoH of battery is 30%.



Signal Statistics

| | Value | Time |
|---|---|---|
| Max | 5.000e+01 | 26104.500 |
| Min | 0.000e+00 | 0.000e+00 |
| Peak to Peak | 5.000e+01 | |
| Mean | 1.496e+01 | |
| Median | 0.000e+00 | |
| RMS | 2.735e+01 | |

Cursor Measurements

Settings

Screen cursors

Horizontal ✓ Vertical

● Waveform cursors

1 Multiply

2 SoH

Lock cursor spacing

Snap to data

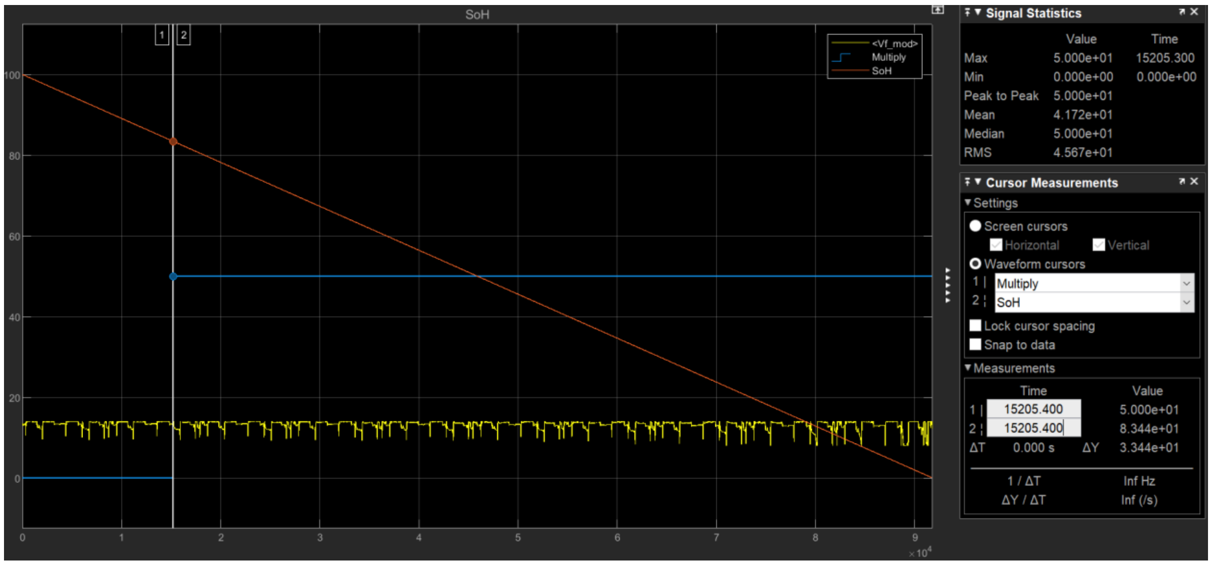
Measurements

| | Time | | Value |
|---|---|---|---|
| 1 | 26104.500 | | 5.000e+01 |
| 2 | 26104.500 | | 2.992e+01 |
| ΔT | 0.000 s | ΔY | 2.008e+01 |
| 1 / ΔT | | | Inf Hz |
| ΔY / ΔT | | | Inf (/s) |

*Figure 103. Real_data5 Results*

## Real_data6

After running the Simulink model with input of Real_data6 we see that our algorithm predicts that the state of health of battery is less than 40% but in fact the actual state of health of battery is 83%.



Signal Statistics

| | Value | Time |
|---|---|---|
| Max | 5.000e+01 | 15205.300 |
| Min | 0.000e+00 | 0.000e+00 |
| Peak to Peak | 5.000e+01 | |
| Mean | 4.172e+01 | |
| Median | 5.000e+01 | |
| RMS | 4.567e+01 | |

Cursor Measurements

Settings

Screen cursors

✓ Horizontal ✓ Vertical

● Waveform cursors

1 Multiply

2 SoH

Lock cursor spacing

Snap to data

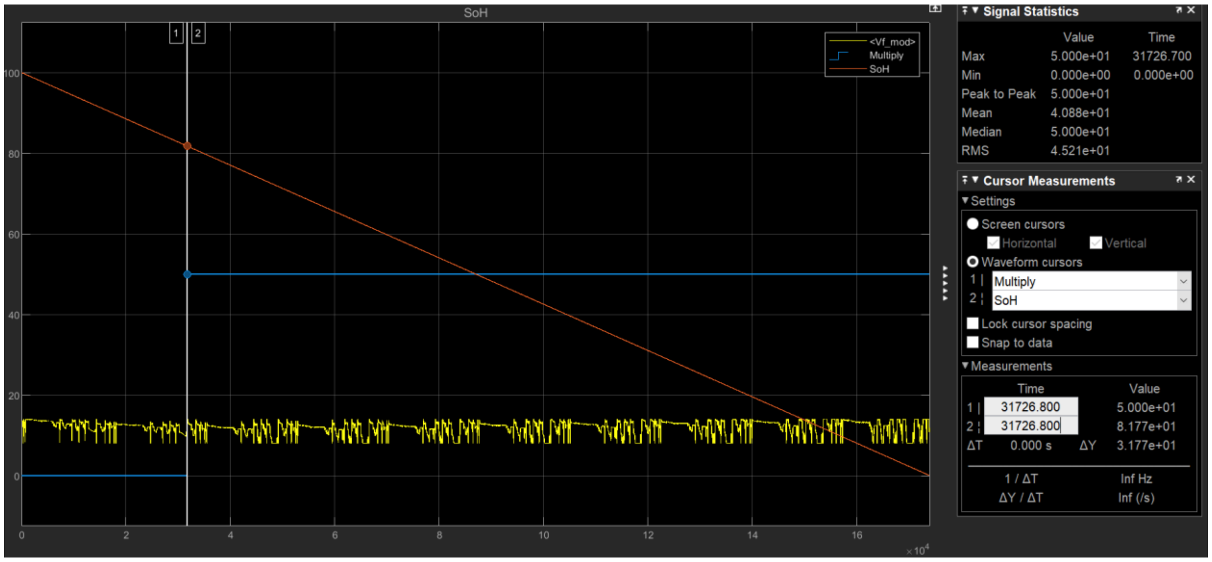
Measurements

| | Time | | Value |
|---|---|---|---|
| 1 | 15205.400 | | 5.000e+01 |
| 2 | 15205.400 | | 8.344e+01 |
| ΔT | 0.000 s | ΔY | 3.344e+01 |
| 1 / ΔT | | | Inf Hz |
| ΔY / ΔT | | | Inf (/s) |

*Figure 104. Real_data6 Results*

## Real_data7

The figure 23 shows that our algorithm predicted that the state of health of battery is below 40% but, it is 82%.



Signal Statistics

| | Value | Time |
|---|---|---|
| Max | 5.000e+01 | 31726.700 |
| Min | 0.000e+00 | 0.000e+00 |
| Peak to Peak | 5.000e+01 | |
| Mean | 4.088e+01 | |
| Median | 5.000e+01 | |
| RMS | 4.521e+01 | |

Cursor Measurements

Settings

Screen cursors

Horizontal Vertical

Waveform cursors

1 Multiply

2 SoH

Lock cursor spacing

Snap to data

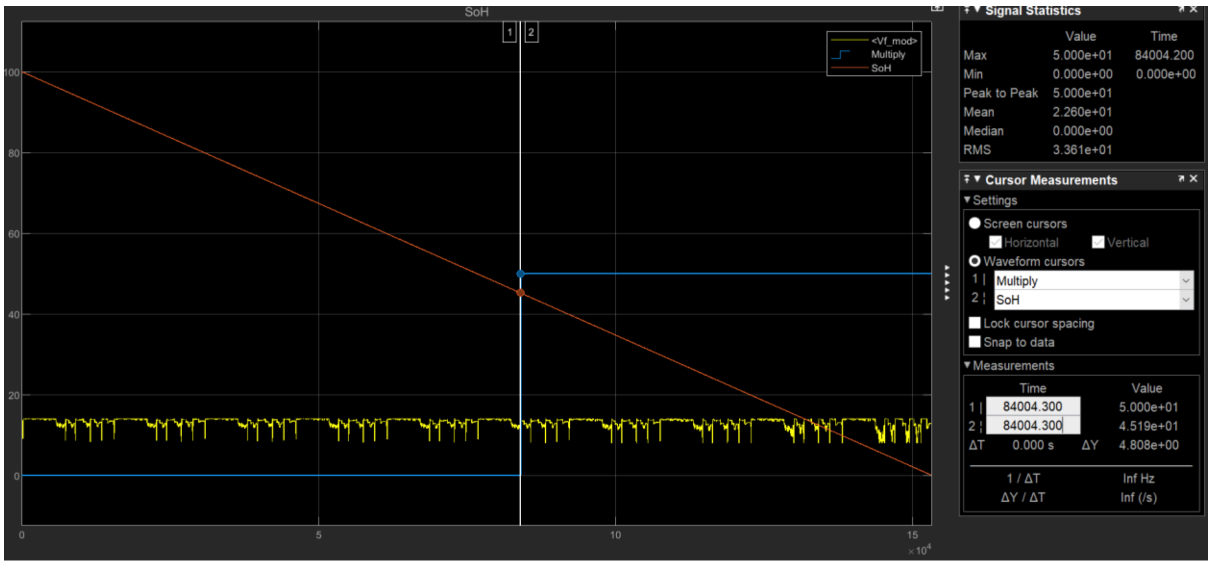
Measurements

| | Time | | Value |
|---|---|---|---|
| 1 | 31726.800 | | 5.000e+01 |
| 2 | 31726.800 | | 8.177e+01 |
| ΔT | 0.000 s | ΔY | 3.177e+01 |
| 1 / ΔT | | | Inf Hz |
| ΔY / ΔT | | | Inf (/s) |

*Figure 105. Real_data7 Results*

## Real_data8

After passing Real_data8 from our algorithm we can see that our algorithm predicts that the state of health of the battery is below 40% but it is 45%.



Signal Statistics

| | Value | Time |
|---|---|---|
| Max | 5.000e+01 | 84004.200 |
| Min | 0.000e+00 | 0.000e+00 |
| Peak to Peak | 5.000e+01 | |
| Mean | 2.260e+01 | |
| Median | 0.000e+00 | |
| RMS | 3.361e+01 | |

Cursor Measurements

Settings

Screen cursors

Horizontal Vertical

Waveform cursors

1 Multiply

2 SoH

Lock cursor spacing

Snap to data

Measurements

| | Time | | Value |
|---|---|---|---|
| 1 | 84004.300 | | 5.000e+01 |
| 2 | 84004.300 | | 4.519e+01 |
| ΔT | 0.000 s | ΔY | 4.808e+00 |
| 1 / ΔT | | | Inf Hz |
| ΔY / ΔT | | | Inf (/s) |

*Figure 106. Real_data8 Results*

### 2.0.3. Analysis of results

The following table shows the results:

*Table 7 Analysis of results*

| Data | Actual SoH while our algorithm predicts it less than 40% | RESULTS |
|---|---|---|
| **Real_data1** | 30% | PASS |
| **Real_data2** | 30% | PASS |
| **Real_data3** | 80% | FAIL |
| **Real_data4** | 30% | PASS |
| **Real_data5** | 30% | PASS |
| **Real_data6** | 83% | FAIL |
| **Real_data7** | 82% | FAIL |
| **Real_data8** | 45% | PASS (Conditionally) |

The Table 1 shows the summarized results. We can consider an error of 5% in prediction which will be gradually reduced as we get more data to train our support vector machine.

Hence, by calculating the average of pass in our tests conducted we get the following results. Our algorithm can predict the value with success ratio of **62.5%.**

## Conclusion of the comparison

As of right now our algorithm has a success ratio of **62.5%.** As we are using support vector machine to predict the results. The SVM can be improved with the passage of time as we pass different types of data through it. The more data we pass through it, we are training it to increase the efficiency of prediction. We have used 8 different types of real data in order to train the SVM.

As mentioned in the beginning, we can see the effect on variance of peaks, delta and difference delta as the state of health of the battery decreases. Hence, in conclusion we can predict if SoH is below certain percentage whose success ratio can be increased with better training of SVM in the future.

With respect to the customer requirements we can satisfy customer requirements by developing algorithm which predicts SOH with an efficiency of 62.5%.

### 2.0.4. Production model

For the production model we must reduce the size of the buffer and make certain changes in the way we calculate variance. We develop a formula so that instead of storing maximum/minimum values for the whole window time and then sending it to a function to calculate variance. We are calculating the running variance using method given below.

Running variance formula

The running variance can be calculated by using the following formulae.

Initialize $M_1 = x_1$ and $S_1 = 0$.

For subsequent $x$'s, use the recurrence formulas

$$M_k = M_{k-1} + (x_k - M_{k-1})/k$$

$$S_k = S_{k-1} + (x_k - M_{k-1})*(x_k - M_k).$$

For $2 \le k \le n$, the kth estimate of the variance is $s^2 = S_k/(k-1)$.

The above formula is used to calculate the running variance. This formula is incorporated in the Matlab function as follows:

```
function variance = fcn(x,window)
persistent i
persistent m
persistent S
persistent v

if isempty(i)
    m=x;
    S=0;
    variance=0;
    v=0;
    i=2;

else
    M=m+(x-m)/(i+1);
    S=S+(x-m)*(x-M);
    m=M;
    i=i+1;
    variance=v/(S/i)-1;
    v=S/i;
end
 if i>window
     v=0;
     S=0;
     i=2;
     m=x;

end
```

With the help of this, we can reduce the buffer size which will reduce the memory size of the code. This is shown in the following figure:

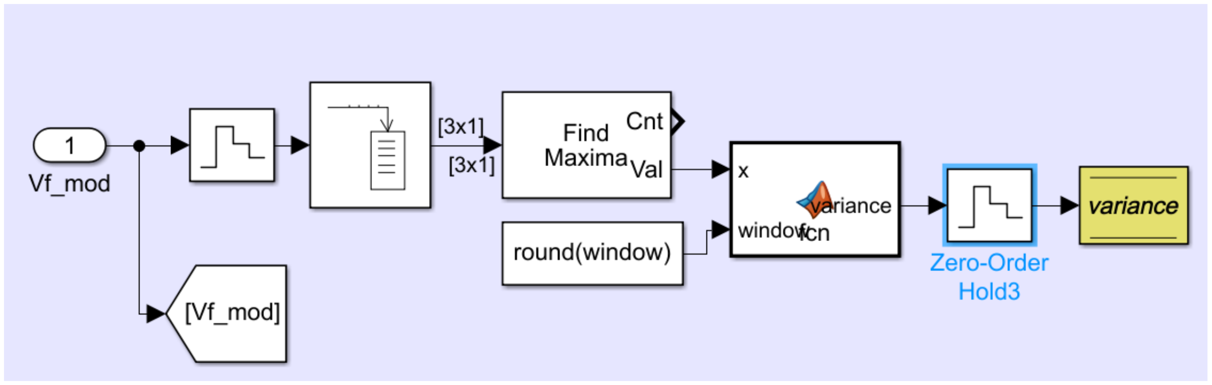


*Figure 107 Running variance*

The next step in order to move from step 2 of our hybrid V-cycle to next step we must generate the code and place it in our rapid control prototype. This can be shown by the following step:

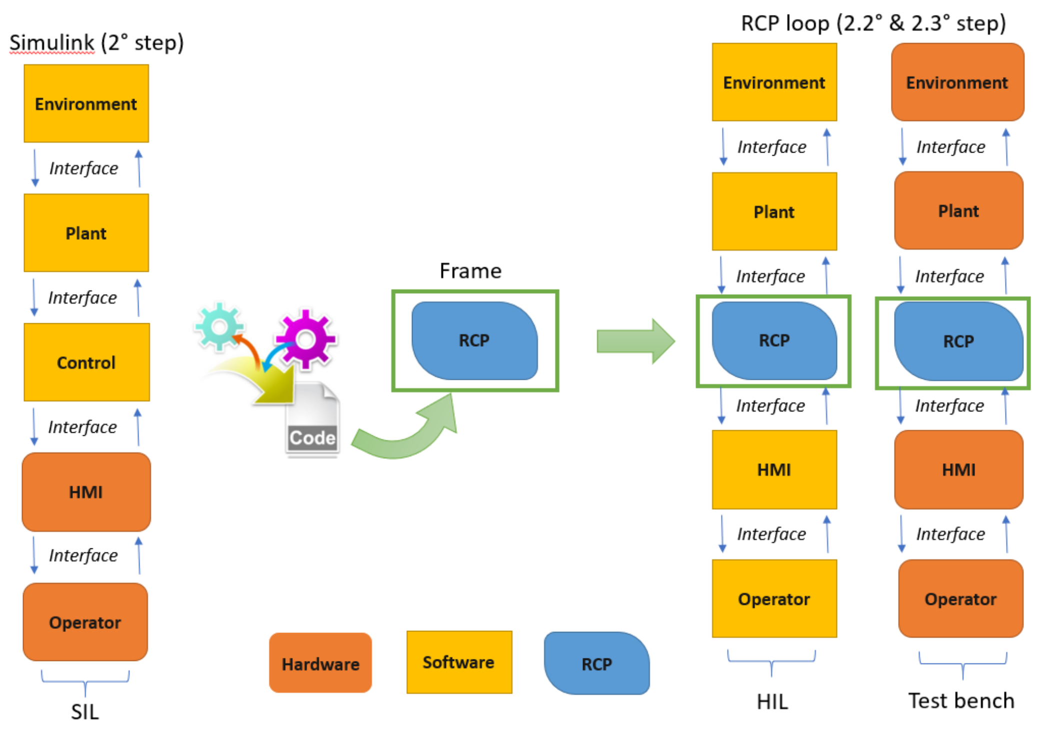


*Figure 108 Hybrid V-cycle steps*

The generated code is added in the appendix with description of settings used. We must see the customer requirements and we can see that by introducing running variance we have achieved to complete requirement no. 1.3 and 1.4.

After checking the requirements, we are sure that we have satisfied the customer requirements. Now we will move to the rapid control prototyping step in which we will run our program on the rapid control prototype device. This device will set out the specifications of the final device.

## 2.1. Rapid control prototype

The RCP device we are using is from Texas instruments which is launchpad cc2640r2f. the specifications of the device is given as follows:

### Specifications of the TI

I used code composer studio to program the device. In order to program the device we need to integrate the software files inside the code composer studio. The specification of the device and the related material about the code composer studio is given in appendix.

## 2.2. Production model testing

I generated the C code using the embedded coder provided in the Simulink. The generated code is to be integrated in the firmware (already developed by some other student). After the integration, I tested the model on our rapid control protype. Due to the shortage of time I was unable to complete this phase of integration of the complete software and my work was limited to the code generation for testing on the rapid control prototype.

## Conclusion:

- We created a development toolchain by integrating ISO2622, V-Cycle and Systems Engineering in one, that is what we called Hybrid V-cycle. This newly developed Hybrid-V-Cycle caters the needs for automotive component development considering all the safety standards now in place. this Hybrid V-cycle makes work easier to handle, thus increasing efficiency.
- We introduced an innovative approach for the architecture of the system that focuses on modularity and reusability for the software components developed during the project. This modular structure enables us to use this toolchain for several other fields, especially complex control engineering.
- We explained our process in detail by applying the newly developed toolchain on the simple example of digital filter.
- To further establish the significance of our toolchain we took a real-world example of Bat-Man project. It shows that how it can be useful in developing a real project considering all the function safety concepts along with the reusability and modularity of architecture of the system.
- For the future development this tool chain can be easily fine-tuned to cater with other different kinds of projects.

## Bibliografia

(2020, Gennaio 2). Retrieved from http://ftp.uni-kl.de/pub/v-modell-xt/Release-1.1-eng/Dokumentation/pdf/V-Modell-XT-eng-Teil1.pdf

Adelinde M. Uhrmacher, D. W. (2009). *Multi-Agent systems: Simulation and Applications.*

Brain Technologies. (s.d.). Preliminary analysis of BATMAN project. Torino.

cadence. (2019, 08). *cadence.* Tratto da https://www.cadence.com/content/dam/cadence-www/global/en_US/documents/solutions/automotive-functional-safety-wp.pdf

dspace. (2019). Tratto da dspace: https://www.dspace.com/en/ltd/home/applicationfields/our_solutions_for/bussimulation/bussimulation_usecases/rapid_control_prototyping.cfm

Instruments, N. (2019, 3 5). *white paper.* Tratto da https://www.ni.com/it-it/innovations/white-papers/11/what-is-the-iso-26262-functional-safety-standard-.html

Iso.org. (2018). *ISO online browsing platform.* Retrieved from ISO online browsing platform: https://www.iso.org/obp/ui/#iso:std:iso:26262:-1:ed-2:v1:en

*Jabil*. (2017). Retrieved from jabil: https://www.jabil.com/insights/blog-main/automotive-industry-trends-point-to-shorter-product-development-cycles.html

Jackson, C. (2019). *lifecycle insights.* Tratto da https://www.lifecycleinsights.com/tech-guide/model-based-development/

*MathWorks*. (s.d.). Tratto da Mathworks website: https://www.mathworks.com/help/ecoder/gs/v-model-for-system-development.html#brufb98-7

mitre.org. (2014, may). *mitre.* Tratto da https://www.mitre.org/publications/systems-engineering-guide/systems-engineering-guide/the-evolution-of-systems

Ross, H.-L. (2016). *Functional safety for road vehicles.* Springer International Publishing.

x-engineer.org. (2019). Tratto da https://x-engineer.org/graduate-engineering/modeling-simulation/model-based-design/essential-aspects-of-the-v-cycle-software-development-process/