Tampere University

Pallab Ganguly

COMPARATIVE ANALYSIS BETWEEN GENERIC AND MATCH APPROACHES FOR IMPLEMENTATION OF A SAFETY FUNCTION IN A CONTROL SYSTEM USING ISO 13849-1:2015 GUIDELINES.

Master Thesis
Faculty of Engineering and Natural Sciences
Examiner: Prof.Reza Ghabcheloo
Examiner: Dr. Niko Siltala
April 2020

ABSTRACT

Pallab Ganguly: Comparative analysis between Generic and MATCH approaches for implementation of a safety function in a control system using ISO 13849-1:2015 guidelines.
Master Thesis
Tampere University
Automation Engineering
April 2020

The motivation behind the thesis work was to grow awareness for safety in mobile machines as well as to popularize the Machine Application Tool Chain commonly known as MATCH. MATCH is a software suite developed by Hydac Software GmbH which is specially tailored for application development in mobile machines.

The thesis aimed to implement a safety-critical software function in a control system using ISO 13849-1:2015 guidelines with MATCH and Generic approaches and compare the two different approaches. The generic approach is not specific to any software suite application and this method can be commonly used to develop a safety or a non-safety function in control systems. A comparison of two different approaches was done by the Qualitative analysis and usability study method. Some common features used in both the approaches like lines of code, quality of documentation, experience, and skills required to implement the safety function was compared in Qualitative analysis. Effort (as a measure of time) involved to implement the safety function in a control system was used as a comparison parameter. Further, a user study with four programmers was conducted to get an insight knowledge of both the approaches. Where two programmers specialized with MATCH and two with Generic approaches were interviewed, respectively. V-model methodology was followed for implementing the safety function.
Implementing a safety function with two different approaches and comparing them was the main objective so practical application of the safety function was out of the scope of the thesis work. An imaginary application of controlling the boom cylinder of a spraying machine with a double-acting cylinder was therefore considered to provide a possible real-life example of the safety function. The hardware components used during implementation were a double signal redundant joystick as an input, Electronic control unit (ECU) as logic and a 4/3 proportional solenoid valve as an output. Performance level (PL) for controlling a boom cylinder application in a spraying machine is PLc according to EN 12001:2012. Hardware components were selected so that the PL of the entire channel was PLc or higher standard (for e.g. PL d or PL e). Detailed analysis for developing a safety function for a control system was done based on the qualitative analysis and user studies.

It was concluded that MATCH implementation was less complex in terms of coding, required less experience and skills to develop a safety function in a control system as compared to Generic implementation. The documentation with the MATCH approach was more time-efficient, created better readability, and better organized compared to the Generic approach. Efforts involved to create the documentation for both the approaches was the same by the developer.

Keywords: Safety-critical software function, MATCH, Performance Level (PL), control systems, ISO13849-1:2015.

PREFACE

This thesis is made as a completion of the master education in Automation Engineering from Tampere University, Finland. The complete thesis work was supervised by **Hydac Oy**, a member of the HYDAC Group which offers expertise in hydraulics, lubrication, electronics and process technology to the Finnish mechanical engineering industry and end users with more than 34 years of local experience. Now Hydac Oy is also actively working on development of control systems mainly focusing on mobile machines.

Several persons have contributed academically, practically and with support to this master thesis. I would therefore firstly like to thank my supervisor **Veli-Matti Jortikka** for his belief that I was competent to work with control systems as this field was quite new to me with very limited practical knowledge. Despite his busy schedule he made sure I got enough resources for my thesis work and constantly guided me whenever needed during my thesis. **Risto Haapala**, the person without whom I think I couldn't have completed this thesis. During my six months of thesis work he taught me the basics of Programmable logic controller, taught the technicalities needed to develop a safety function in a control system. I am truly grateful to him for investing his precious time and knowledge for training me and giving an insightful understanding of the whole topic. **Antti Pasanen**, from **Technion Oy** (part of **HYDAC** Group) a very experienced person in the field of control systems also guided me with some technicalities while developing the safety function.

Furthermore, I would like to thank **Dr. Niko Siltala** and **Prof Reza Ghabcheloo** from Tampere University for their big help throughout the entire process. Niko guided me throughout the entire thesis-writing process giving his valuable feedback. I appreciate the time he invested and the patience he has shown during my thesis work.

Finally, I would like to thank my family for being helpful and supportive during my time studying Automation Engineering at Tampere University despite some serious personal challenges.

Tampere, 28th April 2020

Pallab Ganguly

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

ACB	Auto Code Builder
AgPL	Agricultural Performance Level
AOPD	Active optoelectronic protective device
API	Application Programming Interface
ANSI	American National Standard Institute
BSP	Board support package
CAN	Controller area network
CCF	Common cause failure
DC	Diagnostic Coverage
ECU	Electronic control unit
FMI	Failure mode indicator
I/O	Input / Output
ISO	International Standard Organization
IDE	Integrated Development Environment
K15	Terminal 15 (ignition switch)
MATCH	Machine Application Tool Chain
MST	Mobile/Machine Service Tool
$MTTF_D$	Mean time to dangerous failure
NVMEM	Non-Volatile Memory
PCB	Printed Control Board
PDT	Project Definition Tool
PL	Performance Level
PL_r	Required Performance Level
PLC	Programmable logic controller
PWM	Pulse Width Modulation
SIL	Safety Integrity Level
SRP/CS	Safety related parts of control system
TSE	Test & Simulation Environment
WD	Watch Dog

.

1. INTRODUCTION

With every passing time mobile machines are becoming more and more advanced to increase its productivity and efficiency. Along with an increase in productivity and efficiency the machine manufacturing industries are now concerned about the safety of machines. The objective of companies according to the author to increase safety in machines is to prevent severe damage to machinery and also protect the workers from fatal injuries that can occur through moving parts of a machine (especially mobile machines) in the workplace. One of the key areas to focus on making a machine safe is to develop a safety function in a control system. In current scenario automation companies for example BR automation is strengthening their safety levels from SIL 2 (Safety Integral Level of type 2) or PL c (Performance Level of type c) towards SIL 3 and PL e [1]. SIL has four types of safety levels 1, 2, 3, and 4 where 1 is considered to have the lowest safety level and 4 being the highest. Similarly, PL also has five different levels PL a, b, c, d, and e where 'a' is considered the lowest and 'e' has the highest safety level. Whenever some new technology is implemented or planned to be implemented in mobile machines, the first thing a software developer for the control system thinks or should think is safety. How safe is the machine in case of fault occurrence? What is the Performance Level that can be achieved to develop a safety-critical function in a control system? etc. Part of the questions are answered by making a Risk assessment study of the system. A combination of occurrence probability and the severity of damage is termed as risk. Risk assessment is a process to identify the hazards which can be associated with the mobile machine. Hazards in the context of safety can be defined as the factors that can cause damage to machines as well as accidents of workers in the workplace. Analyzing the nature of the risk involved, if it a sever or a non-sever risk is shown in Figure 4. Hazard detection and risk assessments are the key factors to determine the necessary level for the safety function. Moreover, unfortunately, there is a big difference between practical implementation and theoretical instructions available as well as uncertainty for implementing machine safety and risk assessment [2]. There can be many reasons behind the lack of safety functions one of which BR automation has described that implementing a safety function is quite a complex task and time consuming so, they are currently using pre-certified functional blocks which makes the task easier to implement [1]. The thesis work is concentrating on safety functions in a mobile machine only, so it is important to know what mobile machines are. A machine is generally an assembly of different parts intended to perform different functions. Mobile machines are self-driven, or operator driven vehicles specially tailored to perform tasks on the road, on agricultural

fields, or in the forests. Before implementing any safety function in a machine hazard identification is very important. Hazards in a machine occur due to main reasons under different categories like mechanical, ergonomic, physical. Mechanical hazards occur from the entanglement with unguarded moving parts of a machine-like crushing, cutting. Ergonomic hazards imply physical conditions that can cause risk to the musculoskeletal system. Physical hazards include noise, vibration, the difference in temperatures, electricity, and many more [3]. Let us imagine a parking brake on a PLd microcontroller. The safe state of the brake is fixed, and the controller detects an error at one of its outputs and turns off all power outputs which results in no actuator movement. This is a simple example of machine safety which can be implemented to avoid hazardous situations [4].

In the United States, accidents of workers in mining industries due to mining and mobile equipment has increased over 20 years. Some serious accidents involved getting run over or struck by moving machines, getting entangled in rotating parts were recorded. Ensuring the safety of workers in coal mining industries has become a challenging task [5]. From the year (2000-2015) almost thirty-two deaths have been recorded from coal mining industries in the United States. PDS (proximity detection system) detected that accidents mainly occurred due to collision with mobile machines. PDS mainly aims to monitor the fatalities to reduce similar incidents in the future. According to NIOSH (National Institute of Occupational Safety and Health) researchers if a machine can automatically stop before hitting a miner then the fatality rate can be reduced up to 78% [6]. From the accidents which have occurred in the United States in coal mining sectors it is very evident that machine safety is very important to avoid fatal and non-fatal accidents. The accidents both fatal and non-fatal can be avoided to a large extent if risk analysis followed by risk assessment is planned and executed which results in risk reduction. The procedures of risk assessment are clearly stated in machine directive under ISO 12100:2010 which must be strictly followed during the design phase of the machine [7].

One of the objectives for implementing a safety-critical function in mobile machines is to provide safety to the workers and improve the lifecycle of the machine. Several international organizations are constantly monitoring technological advancements to provide the preferred information to the designers and developers to get a better and clear understanding of these systems. Revised standards were implemented in 2009 with a new machine directive 2006/42/EC [8].

The purpose of making international standards is to help the designers to achieve safety while designing a machine. International standards provide a basic guideline for different

methodologies, terminologies, and different applications of principles. The developers with their experience and skills analyze different risks associated with the machine which are both fatal and non-fatal based on-machine applications. Guidelines are provided to identify, estimate, and evaluate different types of risks associated with every phase of the machine life cycle.

ISO 13849-1:2015 standard intends to provide guidelines to designers and developers for designing and analyzing the control system of a machine. There are certain parts allocated in a machine that is termed as “safety-related parts of control system” (SRP/CS) that provides safety-related input signals and generates safety output signals. The safety-related parts can be an integral part of a control system or separate from it. The capability of Safety-Related Parts in a Control System (commonly referred to as SRP/CS) to produce desired output under different circumstances is measured by its Performance Level (PL) and is generally defined as the probability of a number of dangerous failure occurrence per hour. The probability that a dangerous failure of safety function can occur in an hour depends upon many factors. Some of the major reasons can be hardware and software structures, the reliability of different components which are calculated by “mean time to dangerous failure”($MTTF_D$), to some extent failure detection measured by Diagnostic coverage(DC) and also considering external factors like environmental adversities or some major design faults, etc. [9].

Nowadays, in industries demand for functional safety, proper documentation (use of Test Cases and Use Cases), traceability (relation between requirements, use, and test cases) is increasing gradually. The thesis work aims to find a preferred approach to implement a safety function in a control system which is easier to implement and involves less effort. So, a safety function is implemented with two different approaches MATCH (Machine Application Tool Chain) and Generic, and a comparative analysis was done to determine the most preferred approach. MATCH is a development environment (commonly termed as a software suite) specially tailored for developing applications for mobile machines. In addition to that, the MATCH assists the operator throughout the entire lifecycle of the machine, which minimizes the loss of information during product changes. The toolchain provides PLd certified functional block, thus reducing the effort and time of the developers to develop a safety function. Now testing and maintenance of main input parameters is easier with Graphical User Interface (GUI). Documentation of development process and less programming is required by the system integrator (an individual or organization that has some expertise in uniting part of subsystems into an entire system and guaran-

teeing that those subsystems work together) [10]. A generic approach is a universal approach to implement a safety or a non-safety functionality in control systems and is not bounded by any specific development platform.

1.1 RESEARCH QUESTIONS

i) What safety factors (or parameters) are needed to be considered for choosing hardware components for developing a safety function? How they are calculated? How the parameters help in determining a specific Performance Level for a system?

ii) What will be the preferred method (MATCH or Generic) for the implementation of a safety-critical function in a control system? Based on which parameters the most appropriate method of implementation is decided?

1.2 CONTRIBUTION OF THE THESIS WORK

i) In the current scenario implementation of safety, a function is quite a complex and lengthy process. The main aim of the author in the thesis is to provide a preferred approach that is simpler and less time consuming for implementing a safety function in control systems. A comparative analysis was done to determine the preferred approach between MATCH and Generic implementation. Comparison comprised of Qualitative analysis and usability study and based on the conclusive evidence a result is drawn. This result from the thesis work can prove to be a milestone for mobile industries that are facing challenges to implement a safety functionality due to complexities and more efforts involved in the process of implementation.

1.3 METHODOLOGIES USED

V-model methodology was chosen for implementing a safety function in the thesis work. The reason behind this is that it is simpler and easy to implement and works perfectly for projects where requirements are clearly defined. Moreover, ISO 13849-1:2015 also states the use of a V-model methodology for developing a safety function. But the main advantage is that testing activities like planning and designing of use and test cases are done before actual implementation which saves time and has a higher success rate. MATCH based software implementation supports V-model methodology and it was wiser to use the same methodology for both the approaches. A qualitative methodology was used to compare MATCH and Generic approaches for implementing a software function. Usability study was performed to have a valuable insight of MATCH and Generic implementation methods from experienced users.

1.3.1 V-Model Methodology

The basic principle behind the V-model structure is that implementation is performed in a sequential manner which can be better termed as Verification and Validation. Development and the testing phases are planned parallelly. There are different phases in the V-model structure as shown in Figure 1 they are software specification, system design, module design, and coding. The first phase of a V-model is a product specification analysis. With proper communication with the customer the actual expectations and requirements to develop a software function are understood. The uppermost layer of the V-model builds a foundation of a software development process. When the specifications and requirements according to customer need for a system are clear then the main hardware components are selected according to the specific safety level. Then a high-level architecture is designed based on the system design which is then followed by a low-level architecture giving more detailed information about the functional modules that will be implemented. The lowest part of the V-model is coding which is based on the low-level architecture. System integrators often prefer to use V-model over other methods because it is well documented, and system requirements are clearly defined and have less complexities [11].

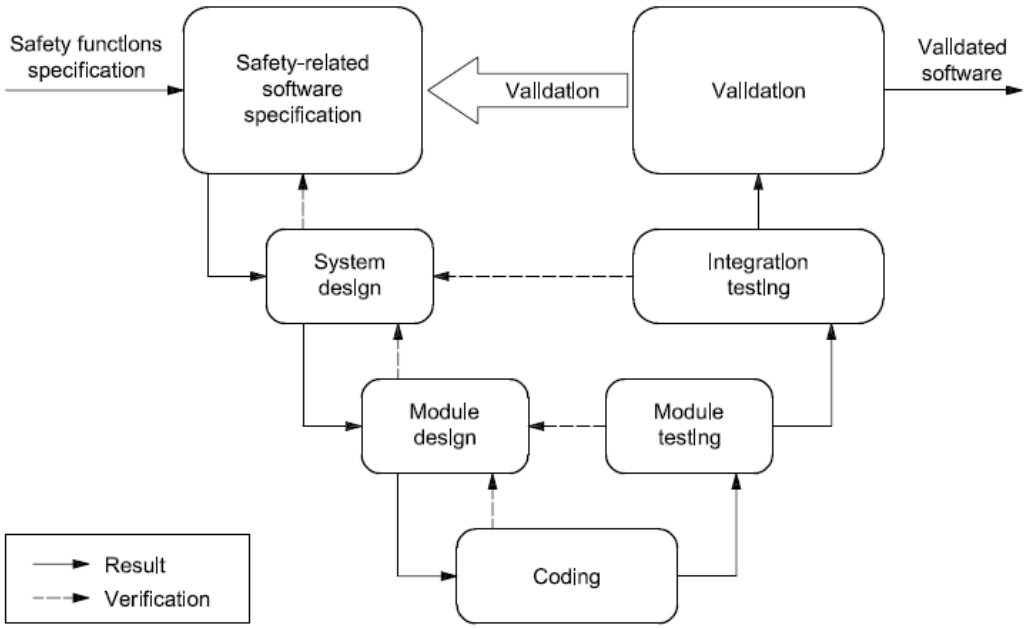

Figure 1.*V-Model for software lifecycle. [9, p.21, Figure 6]*

The main objective of the Comparison study was to figure out the preferred approach for implementing a safety function in a control system based on Qualitative analysis and usability study.

1.3.2 Comparative Study (Qualitative analysis)

Qualitative analysis is a methodology that analysis two or more methods or ideas. It shows how the methods are similar or different from each other based on facts and data. Qualitative approach in comparative analysis is more frequently used compared to the Quantitative approach. There are generally two different types of comparison in comparative study, Descriptive and Normative comparison. Descriptive comparison focuses on the uniformity between two or more methods. In general practice, there is no intension to make changes in the method, rather it avoids doing so. Whereas, in Normative approach aims to detect the flaws between two or more cases and try to improve the present state or make possible suggestions to improve it soon. In the thesis work, the Normative approach is followed [12].

1.3.3 Usability Study

Usability study helps to get a more insightful knowledge of a user experience. From the study a deep understanding of the user, their intentions, and future objectives could be understood. Some valuable understanding can be gained through usability study some of them are,

- Distinct difference between two or more design or implementation methods
- How usability is important?
- Which method is more user friendly?

Questions similar to the above can be easily understood through a usability study [13].

1.3.4 Combined Methodology

Figure 2 below illustrates the combined methodology used to link the research question with the different methodologies to pave a way final for a conclusive result.

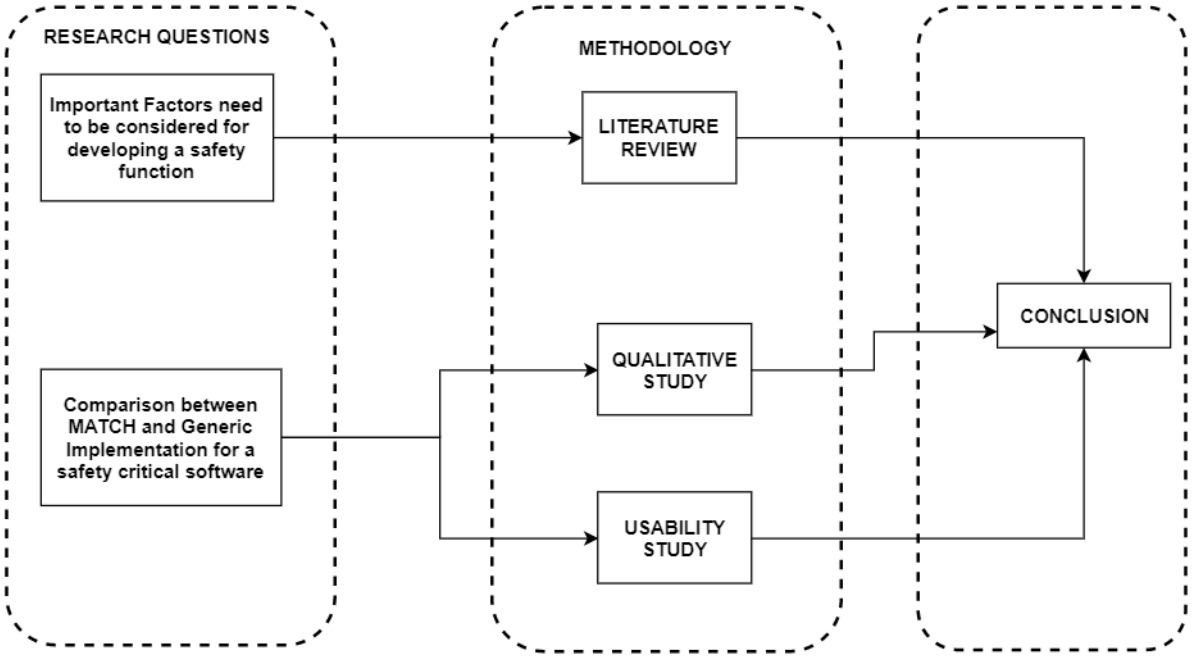

Figure 2. *Overall Methodology*

1.5 THESIS TOPIC OVERVIEW

The main chapters of the thesis work are literature review which is based on the importance of machine safety, use of MATCH toolchain and its safety features, Generic implementation and its safety features, detailed understanding of machine standards according to which the safety function is implemented. Then different methodologies and tools used are explained in more detail along with the materials used in the thesis work. One of the main chapters of the thesis is implementation and analysis. The final and the last chapter is the conclusion where the key findings of the thesis are highlighted and (recommendation) suggested future work from the author's point of view.

2. IMPORTANCE OF MACHINE SAFETY

The section mainly aims to focus on various aspects of machine safety first of which is risk estimation study and what are the consequences in terms of hazards. Further, it continues to describe a few fatalities that can occur if safety in machines is ignored and what can be a possible step through risk assessment to avoid accidents caused due to lack of study. Since the thesis focus on safety function in control on a simple example is elaborated at the end of the section.

2.1 Risk estimation study and its consequences

Companies or research organizations aim to develop a safety function to reduce the unacceptable risks or hazards associated with the system. Moreover, it is often recommended to implement the risk analysis in the early phase of a system development if enough information is available. Thus, the order of analyzing the risk is very important. According to ISO 12100:2010 standard, which is currently replaced by ISO 14121-1, the risk associated with a specific channel or a system is first analyzed. Analyzing includes deciding the limits for the system, identifying the hazards involved, and then estimating the intensity of risk that is identified. The final process is to reduce the risk as much as possible.

The basic methodology of identifying and reducing risk is described shortly, which gives a clearer understanding of the steps a designer must consider.

i) Calculate the limits of the machine, i.e different machines have different operation modes, limits caused due to mishandling etc.as described in ISO 12100:2010

ii) Connecting and analyzing different hazardous situations with corresponding hazards.

iii) Estimate the risk associated with each hazardous situation due to its hazards.

iv) Estimate or reduce the risk factors linked with different hazardous using different protective measures [8].

According to IEC TR 62062 standards, hazards estimated in a machine is based on following risk parameters,

- severity of injury (Table 1 illustrates the scale for determining the severity)
- Probability of reoccurring a hazard concerning time
- Possible extent a hazard can be avoided or limited [14]

Table 1. *Consequences with Severity level [14, p.769, Table 1]*

Consequences	**Severity**
Irreversible: death, losing an eye or arm	4
Irreversible: broken limb(s), losing a finger(s)	3
Reversible: requiring attention from a medical practitioner	2
Reversible: requiring first aid	1

Programmable Logic Control (PLC) is nowadays mainly responsible for developing the functional safety requirements in a machine. Risk analysis plays a crucial role in determining the safety requirement (Performance Level) so it has become a very important part of machine development [14].

Heavy earth-moving machines and especially heavy combination vehicles are more prone to road safety risks because of lower roll and yaw stability limits. The reason behind it is its large size and inertia, and high center of mass. Advanced technologies such as Roll Stability Control (RSC), Electronic Stability Control (ESC), and many others along with strict road regulations are made to reduce the risk factors for such vehicles.

Some common issues correlated with heavy mobile machines are,

- The motion and mechanism of these machines are quite complicated because of its structure and weight

  It is quite challenging to monitor or control commercial vehicles
- Commercial vehicles can either have yaw or roll stability depending upon the load and driving environment.

Over the past few decades, efforts are constantly made to improve safety in control systems, many advancements are made in braking systems such as RSC and ESC [15].

2.2 Why Machine safety is important?

There can be non-fatal as well as fatal accidents from moving machinery if safety is ignored. Some of them are listed below [16],

- Workers or people around can be stuck and injured by moving machines or ejected parts
- There can be both fatal and non-fatal consequences from sharp protruding edges of a machine
- Irreversible injuries which are often fatal can happen due to uncontrolled moving parts or fixed parts of a machine
- When there is a lack of fault monitoring, sometimes the machine becomes unreliable and can cause injuries
- Accidents due to fatal and non-fatal can also occur due to a lack of proper knowledge, experience, and training.

Certain steps can be taken to avoid mishaps to happen to some extent. Before a machine is used commercially, a proper risk assessment must be performed according to ISO 12100 standards. Some of the methods followed for risk analysis are listed below [16],

- Proper inspection of the machine which includes safeguard inspection and fault detection must be made. Safeguards include physical guards against sharp

edges or other parts of the machine that can harm people or two-hand control etc are included in safeguards

- Safety control function must be implemented to ensure safety in machines especially mobile machines. Norms for regular maintenance and inspection of a machine. Maintenance is performed on certain features which on failure can cause unacceptable risks
- Installation of static machine parts in a stable state
- The working environment of the machine must be carefully chosen and circumstances, where risk factors are more, must be avoided [16].

2.3 Safety in Control systems

A manifestation of the latest invention in safety control systems is directed towards lessening the chance of mishaps in terms of accidents in vehicles. Another viewpoint is to give a strategy for diminishing or maintaining a strategic distance from driver interruption during conceivably risky conditions experienced while working on a vehicle. As an example of one encapsulation of the current innovation of a safety control system for vehicles, a communication device which has at least one input reachable and an output device which is communicable within the vehicle. An operating sensor is capable of sensing at least one condition that can be identified with vehicle activity. A controller must have proper communication with the sensor and if any parameters detected by the sensor outside of the specified limit that can cause hazard or possible risk, then the controller immediately cuts off the output. The communication between the operator and the device is also suppressed this can prevent any unwanted risk or hazard [17]. In most cases safety in control systems is characterized by the reliability of individual subsystems. Their reliability is reached because of the utilization of different systems where various technologies are applied. Safety systems should then incorporate not just all components which are a part of independent systems (such as sensors, input/output devices) but also includes reliability of safety functions [18].

When designing any machine, designers need to consider many criteria. One of them is the safety of machinery and is also applicable for Performance level($_{required}$) (PLr) for every safety function. The recently planned system that complies with PLr must be also applicable to the current framework. The system, which plays out the safety limit comprises input, logic, and output devices. An example of an emergency stop which is one of the most important safety function in a control system. It can be explained through a simple example by considering an input, Logic, and output parts. Suppose an input device detects an external signal and it sends it to the logic unit where it is processed. If

there is an error is detected in the system, the output part provides a required intervention i.e it power supply of the machines is cut off and the machine stops [19].

3. STANDARDS FOLLOWED FOR IMPLEMENTATION OF A SAFETY-CRITICAL FUNCTION

In this section of the thesis standards that are needed to be followed for implementing a safety function in a control system are explained. ISO 12100:2010 is a standard which explains the general principles of risk assessment and risk reduction. ISO 13849-1:2015 describes how to design a safety-related part of a control system and Misra C standard a subset of C language consists of a set of guidelines for coding a safety function in C language.

3.1 ISO 12100:2010

The objective of ISO 12100:2010 standard is to provide developers a proper guideline to design a machine that is safe and appropriate for use. To ensure a safe design of a machine a proper plan of risk assessment and risk reduction must be executed as shown in Figure 3. The risk assessment depends upon many factors which a designer must take into consideration such as,

- Conditions that determine the limits of a machine such as different machines have different operating modes, the involvement of different users and possible mishandling of machines, etc... The experience, training of a user operating the machine are a few limitations that need to be considered
- Detecting the hazards and associated hazardous situations
- Examining the risk involved and steps to reduce the risk
- Eliminating the risk or hazards through protective measures.

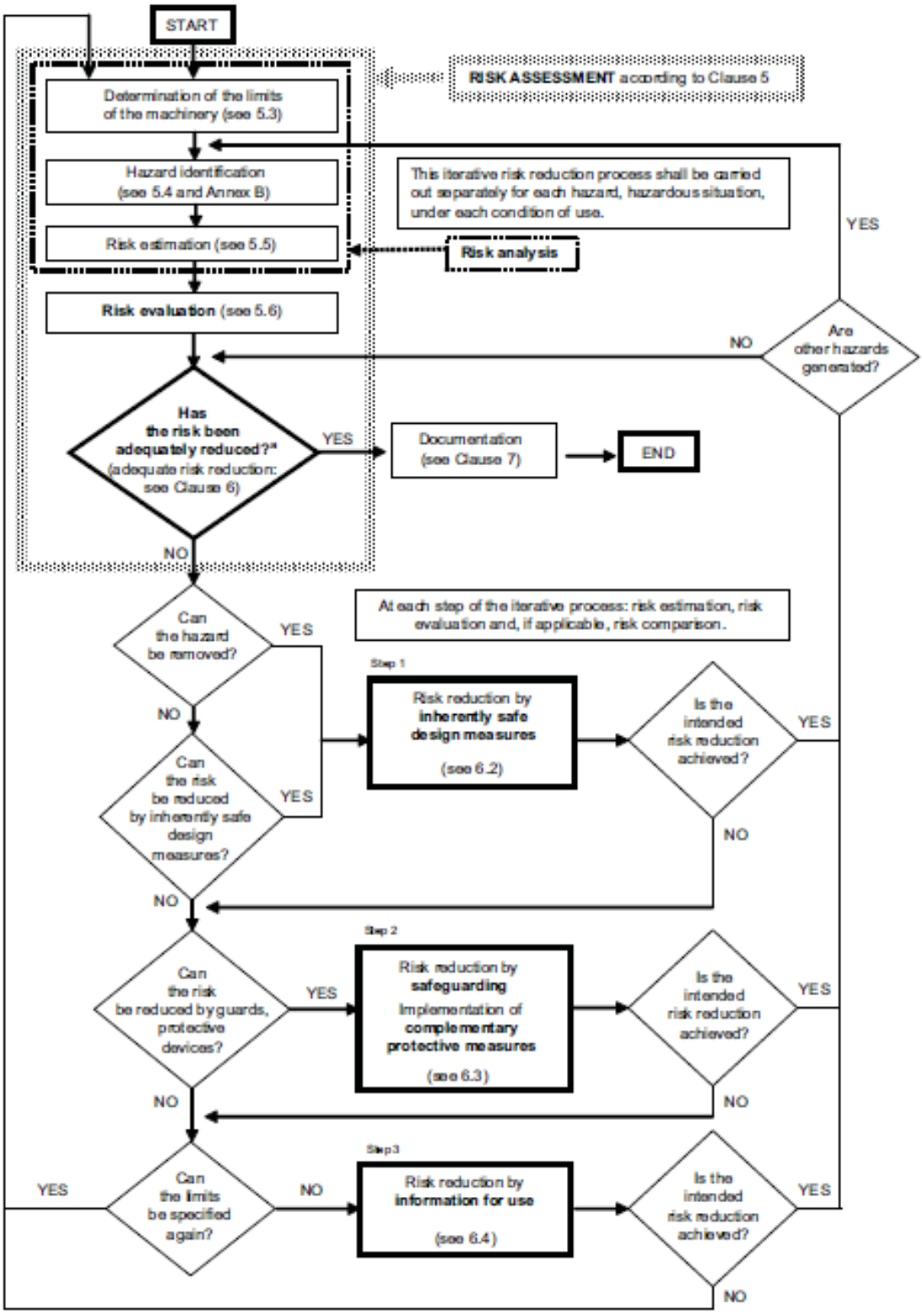

***Figure 3**. Schematic Diagram of risk reduction process [7, p.15, Figure 1]*

In the context of machine safety there are three types of safety standards according to machine directive, Type A, Type B, Type C.

- Type A also called basic standards, as the name suggests provides an overall concept for the design and general characteristics that can be implemented in a machine.
- Type B is named as a general safety standard and describes a bit more specific safety features or different safeguards that can be applied in a wide range of machines. Type B is again divided into two subcategories: Type B1 and Type B2.

Type B1 deals with safety-related to distance, surface temperature, and noise while Type B2 deals with different safeguard methods like two-hand control or using pressure-sensitive devices, etc.

- Type C safety requirements aim for a very specific type of machine or a group of machines with similar features [7].

3.2 ISO 13849-1:2015

ISO 13849 standard is based on Type B1. The goal of a system architect behind designing a control system of a mobile machine is to make it safe. The idea behind implementing an SRP/CS (Safety Related Part of Control systems) is to reduce the possible risks or hazards as much as possible. Different machines as well as different safety functions inside a machine has a specific PL which needs to be achieved to make the machine safe. There are different levels of risks that possibly can occur in a machine because of human behavior or technical faults. It is difficult to predict all possible risks that can happen during the lifecycle of a machine, but the majority of the risk factors can be avoided by reliable data and studying past accidents that have happened on similar machines. Comparable machines mean, it has similar safety functions, simpler modes of operation, and the same technology used. Figure 4 elaborates the decisions are made to choose a preferred PL level of a system [9.]

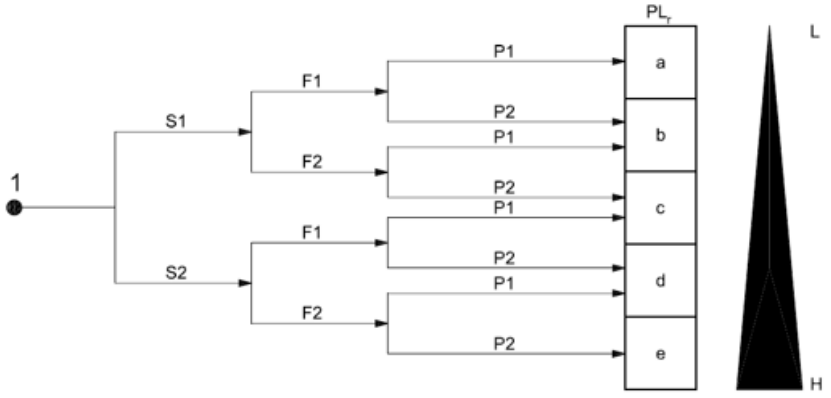

Figure 4.*Decision making graph to choose PL [9, p.45, Figure A.1]*

Where,

1 starting point for evaluation of safety function's contribution to risk reduction

L low contribution to risk reduction
H High contribution to risk reduction
PL_r required performance level

Risk parameters:

S severity of injury

S1 slight (normally reversible injury)
S2 serious (normally irreversible injury or death)
F frequency and/or exposure to hazard
F1 seldom-to-less-often and /or exposure time is short
F2 frequency-to-continuous and/or exposure time is long
P possibility of avoiding hazard or limiting harm
P1 possible under specific conditions
P2 scarcely possible

Figure 4 describes the different levels of hazards that can occur, along with their severity is it reversible or irreversible injury. It further illustrates the fact of how much extent a hazard can be avoided. So, these factors are considered while selecting a performance level of a system. It should be noticed that this method is not the only one to estimate the performance level of a specific safety function or for a whole system. Calculating the $MTTF_{d,}$ DC, CCF values of a system, and then choose the performance level is also a commonly used method, and this method is followed to implement a safety function in the thesis [9].

In context to safety-related embedded software, Chapter 4.6.2 of ISO 13849-1:2015 guides us on how to implement a safety function, guidelines from these parts were followed while implementing the safety-critical function in the thesis. Implementation of the safety function was done with two different methods one with MATCH approach and other with a GENERIC way. For a safety-related application software (SRASW), a safety function must have contained the following characteristics:

- development lifecycle with verification, validation activities.
- Documentation of specialization and specification. Proper documentation of High level, the low-level architecture of the entire safety-critical function. Selecting a category of the system, calculating the Mean Time to Dangerous Failure ($MTTF_D$), Diagnostic coverage (DC), Common cause failure (CCF) as shown in chapter 6.2 (of ISO 13848-1:2015 standard) system specifications.
- Modular and structured coding.
- Requirements both functional and non-functional must be written. Implementation is done following the use and test cases. During the implementation, use and test case are designed and then implementation is accordingly for two different methods.

- Functional testing such as black-box testing [9].

3.2.1 Performance Level of Safety-Related Parts in a Control System

Performance Level is determined from the safety-related parts of a control system. To select a preferred performance level for a system different factors such as $MTTF_d$, DC, and CCF these factors are calculated for each component as well as for the entire system. Apart from considering the above factors, the behavior of the safety function under faulty situations, safety functionalities added in the software developed is also taken into consideration. Table 6 illustrates the relation between PL with $MTTF_d$, CCF, and DC values [9].

3.2.2 Mean Time to Dangerous Failure

When considering a safety-critical safety function, $MTTF_d$ is a portion which deals with failure modes in terms of years, which can lead to serious hazards to personnel, environment, or equipment. $MTTF_d$ is divided into three levels (low, medium, and high) as shown in Table 4, and safety function should be considered as a separate channel while calculating the $MTTF_d$ values. Low $MTTF_d$ signifies the occurrence of a dangerous failure is more likely to happen earlier compared to $MTTF_d$ with medium and high values. So, it is always preferred to choose components or design a safety channel with high value [9].

3.2.3 Diagnostic Coverage

Diagnostic coverage in control systems means the detected dangerous failure divided by the sum of dangerous and detected dangerous failure. Detected dangerous failure are failures which are detected and can lead to loss of safety functions whereas dangerous failures are estimated failure that can occur. There are four different levels of diagnostic coverage and is measured in terms of percentage as shown in Table 5 Diagnostic coverage less than or equal to 60 is not considered in any system. High diagnostic coverage is considered best and low is not recommended in general [9].

3.2.4 Common Cause Failure

Common case failure occurs generally when multiple mostly identical components fail due to different reasons. The estimation of CCF value is calculated for the entire system. Table F.1 from ISO 13849-1 explains clearly the measures which are taken into consideration while calculating the CCF score corresponding to each measure. If the score is less than 65 then it is not considered to be a part of safety function implementation [9].

3.2.5 Category

While developing a safety-critical function in control systems a specific architecture consisting of different safety-related block diagram are used. These architectures belong to a specific Category, like Category B, Category 1, 2,3, and 4 respectively. Generally, all the machines are mapped with a category to make a safety function of a control system. An example of category 2 architecture is described in Figure 5.

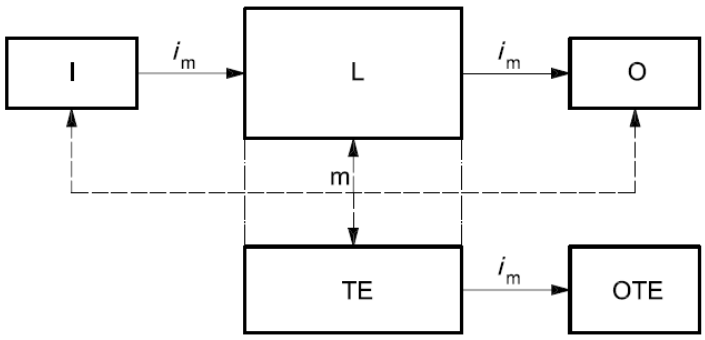

Figure 5*.Category 2 architecture [9, p.35, Figure 10]*

Where,

I_m interconnecting means

I input device, e.g. sensor

L logic

M monitoring

O output device, e.g. main contactor

TE test equipment

OTE output of TE [7]

3.3 Misra C standard

Misra C guidelines are a part of C language which provides an opportunity to reduce or remove errors in coding. For the development of any safety-related software many times it is recommended to use a language subset, similarly, Misra C standard is also used for development purposes with high integrity and reliability. Misra C standard which is followed during the code implementation is generally used in a software development process. Although it can be used separately but is recommended to be used in a process, to ensure that the entire process is completed correctly. Some of the key areas of Misra standard are:

- Software Requirement which includes safety requirements. It ensures that the code implemented is clear without any ambiguity as well as correct.
- The requirements are correct and there are no extra functionalities to ensure no complexities.

Misra C standard allows flexibility in terms of deviation. But deviation or exceptions made in implementation must be properly documented as well as authorized. According to section 6.2 of Misra C standard, guidelines are categorized into three different sections, ‘mandatory’, ’required’, and ‘advisory’. Mandatory guidelines should be followed without any exceptions. Required guidelines follow the same rules as mandatory but some formal deviation as described in section 5.4 of Misra C standard can be allowed. But it may depend totally on the organization or the project whether the exceptions are allowed or not. Whereas advisory guidelines do not give the liberty to violet the rules of the standard but there are a lot of flexibilities allowed and formal deviation rule is not followed [20].
A literature study was done for both MATCH based approach as well as Generic approach. In chapter 4 the author tries to point out some important MATCH toolchains their role in developing safety-critical software as well as some safety functionalities defined in MATCH. Whereas, chapter 5 shortly points out the key features of the TTC 500 controller based on which the Generic implementation was done.

4. MATCH TOOL CHAIN AND ITS SAFETY FEATURES

MATCH is a software suite to develop a safety function in a control system. The main components of the Machine Application Tool Chain (MATCH) are PDT (Project Definition Tool), MATCH CORE, MST (Machine Service Tool), TSE (Test and Simulation Environment) [10].

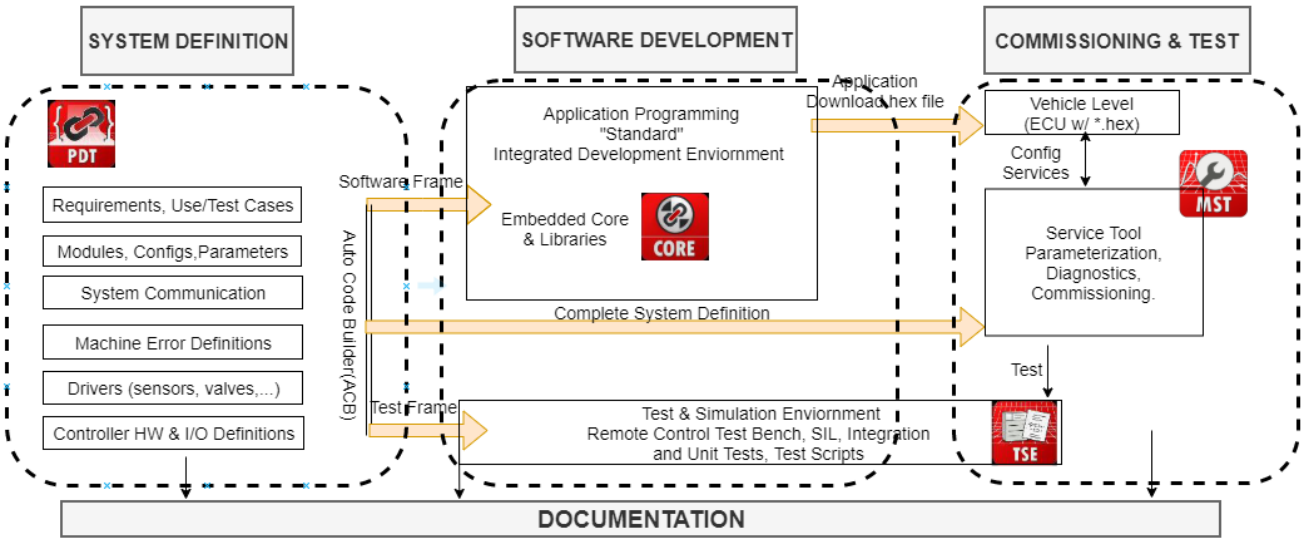

Figure 6.*Standard IDE of MATCH [10, p.9, modified diagram]*

Figure 6 contains an overview of the standard IDE (Integrated Development Environment) for application development in MATCH. This is the tool for writing, testing, and building the PC simulation and application code and binaries. The IDE comes with autocompleting and an integrated online help to the MATCH core libraries for comfortable development [10].

4.1 Short Description of PDT, CORE, MST, and TSE

4.1.1 PDT

The Project Definition Tool is the first tool of the Machine Application toolchain. PDT is a Windows software that supports the software development and documentation of ECU (Electronic Control Unit) applications on the machine level. The PDT supports the V-model development process. The V-model includes the Requirement Management, and the specification of the test- and use-cases. The PDT considers especially the demands of the functional safety and supports vehicle applications with a safety level up to SIL2 / PLd / AGPLd (Agricultural Performance Level d) [21].

4.1.2 MATCH CORE

MATCH defines the interface for the application programming. MATCH core is the heart of “Mobile Application Tool Chain”. The project library of MATCH core is located between the application development environment interface and board support package of the ECU [10].

4.1.3 MST

The MST is a PC program for mobile controller ECU maintenance. Depending on the version the MST supports one or more ECUs and displays.

There are the following different versions available:

- Service: displays ECU hard- and software- information for up to eight ECUs. Reads errors and states from the ECU. Updates the ECU firmware. Supports up to eight access level. Supports the project files from the Project Definition Tool. Supports the management of the parameter lists, factory settings, errors, I/O diagnostics, and CAN messages.
- Developer: contains all features from the Service version. Additional: download of the project files, support of error tests, and a CAN bus data plot function.
- Designer a version for screen design and translation customization.
- Ultimate the version for power-user with a complete unlimited functionality [22].

4.1.3 TSE

Software testing is a way to assess the quality of the software and to reduce the risk of software failure in operation. Software testing includes many different activities: execution, planning, analyzing, designing, and test implementation, reporting test progress and test results, and evaluating the quality of a test object. Test and Simulation Environment is an application for MS Windows. The TSE is part of the machine application toolchain. It takes over the test and simulation part in the MATCH [23]. Due to inadequate training during thesis work, the use of TSE was not applied in the thesis work. MST was mainly used for testing the software function.

4.2 MATCH core and layer

MATCH is a software suite for the development and maintenance of control systems for mobile machines. The MATCH Core environment is capable of planning, development which also includes safety features and services during a machine life cycle. The MATCH Core's library covers both the development environment and Board Support Package (BSP) of ECU. BSP is an interface between MATCH Core and ECU. [10]

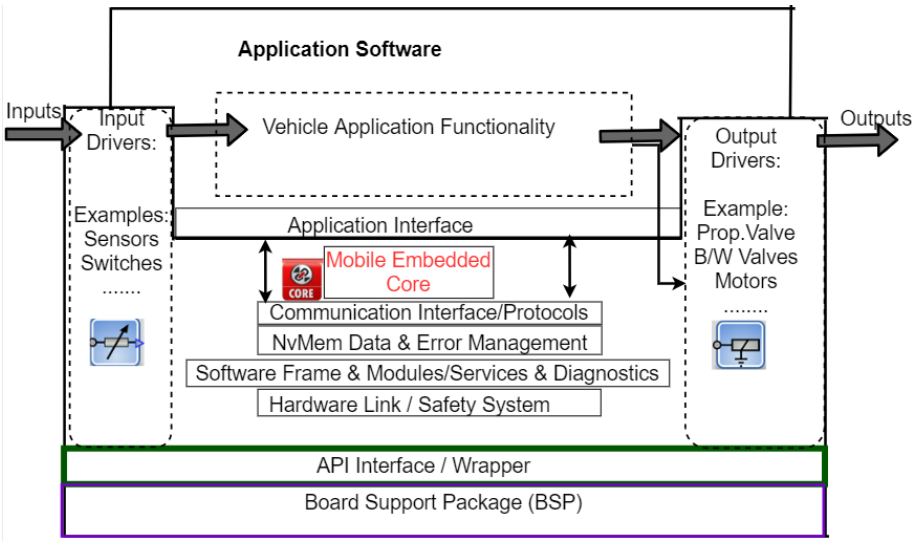

Figure 7.*MATCH Core Overview [10, p.12, modified diagram]*

Figure 7 above elaborates that MATCH Core is divided into two parts, Application Software and MATCH Core combined with ACB (Auto Code Builder) framework.

- Applications that are written by system integrator is termed as Application Software.
- The library blocks from which the input/output devices are built are also capable of communication interfaces/ protocols, NvMem data, handling error managements, and theses are done using the project definition tool.

The system integrator must be competent enough to use all the MATCH functionalities such as defining variables, structs, etc. Along with the application, the MATCH Core library can perform the following functions and services:

- CAN communications using different protocols
- Error handling and management.
- EEPROM management for parameter handling
- Data is transferred between RAM and NvMem and is handled using a database list [10].

Between the BSP and the application, the MATCH core forms a layer for better communication between the two. From Table 2, we can clearly understand that MATCH core uses BSP functionalities to make the interface much simpler to use for an application program. Moreover, the system integrator must be aware of the input/output pins in an ECU hardware as well as consider the operational requirements and specifications of ECU.

Table 2. MATCH Layers

Application (Layer 4)
MATCH core (layer 3)
BSP (layer 2)
ECU hardware (layer 1)

Block library consists of library elements for Input/Output functionalities and library elements for signal processing. MATCH offers predefined software bricks, each of them is designed to completing one task. Software bricks consist of signal and functional blocks and the main objective is to reduce the complexities associated with an application and coding efforts. The signal and functional blocks are tested and safety certified. Signal blocks, for example, LUT (Look Up Table) and functional blocks are for example input blocks like ADC (Analog to Digital Convertor), input current, input voltage blocks, etc. whereas, output blocks consists of DAC (Digital to Analog Convertor), Proportional output blocks, etc. Signal blocks may or may not be a part of the library block as they are common structures. Like signal blocks the library blocks also belong to common structures for block objects. Library blocks unlike signal blocks are capable of handling errors and may also contain other signal functions. Every block function, both signal, and the functional block has a different interface for MATCH environment and to support different

API functions. Input, output, and functional blocks are the main types of block objects. The input and output blocks are connected to the respective devices. Functional blocks are capable of error identification, parametrization. Each block can be divided mainly into two parts based on mathematical operations and control of hardware components [10].

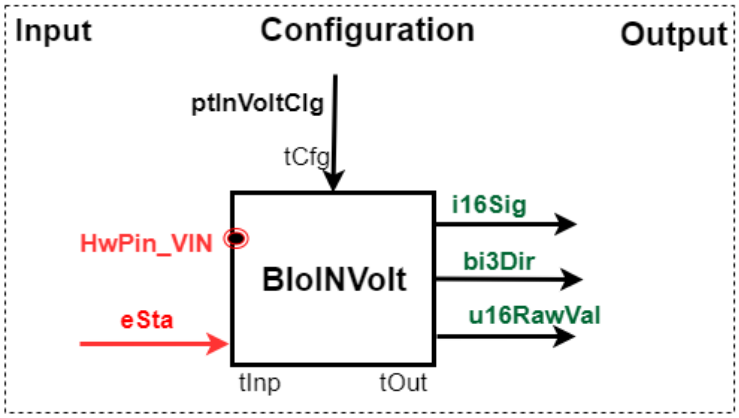

Figure 8.*MATCH functional block [10, p.112, Figure 3]*

In the double input voltage block shown in Figure 8 is an INVolt Block where the input is the voltage, obtained from the input pin, and eSta is a general convention of calling an input parameter. The outputs from the block use the tOut as suffix.

The initial structures of the block implemented using PDT before calling the API functions are called configurations. The role of an interface is to sustain the current block settings and the interaction between the block API and application.

Some of the ECU's which are supported by the MATCH toolchain is the HY-TTC family. Most of the controllers from TTC are safety certified few of them are HY-TTC 30S-H,32S-H, and HY-TTC 5XX family (510,540,580). The controllers from the TTC 5XX will be used in the thesis work. Many different components are integrated into a single ECU, each of them can perform a specific task. The main components are:

Main ECU and WatchDog.

Different sensors like temperature sensor, supply detector, and battery monitoring Input and Output pins both analog and digital CAN interfaces and memory allocations like RAM; Flash memory, etc [10].

4.3 SAFETY FUNCTIONALITIES IN MATCH

Increasing demands of the market and strict conditions on the functional safety of the machines create the need for a preferred development process. MATCH toolchain provides an already tested certificated software frame and functional blocks that reduce the development and test efforts. The toolchain supports the system integrator during development, test, and maintenance of applications running on electronic control units. System integrators before completing a project must consider that the MATCH middleware does not provide safety for the entire system. MATCH embedded middleware contains

MATCH Core with certified tools and MATCH library blocks. The system integrator has the responsibility for the functionality and safety of the developed application software. So, the safety of both functional and non-functional requirements must be verified on an end-user level. The end-user is the individual who utilizes the product or equipment after it has been completely evolved, promoted, and introduced. Auto-code builders, library blocks, and service tools in MATCH middleware are safety certified. The safety certification is based on ISO 13849 (Pld), ISO 25119/DIN EN 16590(AgPld), IEC 61508 (Sil II). Before the system is considered as complete, the developer must verify both safety standards and levels that are needed for the system to achieve a specific goal. Moreover, fail-safe behavior in a safe state is supported by MATCH middleware. In the thesis work as stated before, HY-TTC 5XX any one of the types of controller belonging to this family is used. The safety certifications for the HY-TTC 5XX family is up to Sil II and PL d according to IEC 61508 and DIN EN 13849 standards, respectively. The role of the system integrator concerning safety is very crucial. Both functional requirements and developed software applications must be properly tested. If any software callback error occurs, the system integrator is held responsible [24].

SIL error blocks handle all the error reactions and safety functionalities when they are activated. The error activation occurs when the value of a specified data is not within a predefined range. The BSP (Board Support Package) can keep track of the internal electronic circuit of the PCB as its functions relate to the safety error objects of the MATCH Core. The BSP also controls the temperature of the ECU hardware. If the temperature goes above the maximum limit, the hardware of the ECU enters the safe state. To restrict the system to enter the safe state a different application reaction with a smaller range can be implemented before it reaches the maximum temperature. There are limits set for under and overloads to detect the behavior of the system values before it enters the safe state of an ECU. Similarly, for sensor supply there are defined threshold values to analyze the critical behavior of a system. A definite protocol is followed for the controller to shut down. Reset of all the applications before the actual shut down is termed as soft reset. As a result of soft reset there can be some unwanted values in static variables. So, it is recommended that the global and static global variables are reset in the shutdown phase. In between a shutdown phase, all the outputs must be turned OFF to ensure a safe state in case of a soft reset and on the output side. Error detection from the internal electronic circuit of the printed control board as well as different components that are attached to it are supported by safety-critical Input and Output pins. When an error occurs in an internal circuit of a PCB hardware, then it automatically enters in a safe state. Some specific wiring problems like “short to power” or “open load or short to ground” which causes some safety errors can be handled in two ways: [24]

- Faulty pin behavior is detected by the BSP of a controller.
- Detection Method of the MATCH library takes care of the failure reactions.

One of the main functions of MATCH safety is its safe state. Whenever there is any occurrence of an error in the safety-related control parts, the system enters a safe state. It is very important to monitor the safety-related components, as a failure of any of those can cause hazardous situations. There are some controller hardware components which are termed as safety-critical they are:

- Watchdog
- RAM
- Power Supply
- Flash
- Communication between Watchdog and Main CPU inside an ECU
- Temperature monitoring

Depending upon the application the components perform, the MATCH middleware activates safety characteristics to prevent some hazardous events. Some of the characteristics related to board safety are:

- Sensor supply
- Watchdog timing
- Temperature monitoring

The I/O pins have safety features too. Some of Input and Output pins included in MATCH are:

- Analog/Digital inputs, 0…5/16/32V
- Digital frequency / timer inputs
- PWM outputs 2A, that includes current measurement inputs.
- Analog sensor supply, if any Analog input is defined as safety-critical.

When some error occurs, the system goes to a safe state, as a result of that all the outputs of an ECU is turned OFF, but the system runs simultaneously. If any hazardous situation occurs in ECU hardware, then the execution of the system might stop completely. If the time taken to complete an application cycle exceeds the maximum limit, then the watchdog in the controller will shift to a safe state [24].
All the errors that occur while the applications are running are stored in an error history list. Occurrence of a specific error during one lifecycle is also counted. These functions are performed by error blocks within the MATCH toolchain. For securing the error history list it is possible to store it twice so that the data is not erased during the emergency

shutdown phase. To make more preferred error detection, the error blocks can include more than one error elements [24].

4.4 Project Definition Tool

Project Definition Tool (PDT) is the first tool in the MATCH toolchain. The software application framework of PDT generates multiple outputs such as data and error management, diagnostic and service tool interface. Multiple language interface for a vehicle can also be implemented in PDT. Project documentation is very important as it gives a clear vision of the project. PDT provides both software and vehicle documentation which includes different use and test cases. Figure 9 illustrates the development application at the machine level. System definitions defined in PDT are compatible with all ECU's. The definition includes CAN messages, error messages, and parameter definitions. The pin definitions are given separately for each ECU. System and pin definitions are a part of the application code frame [21].

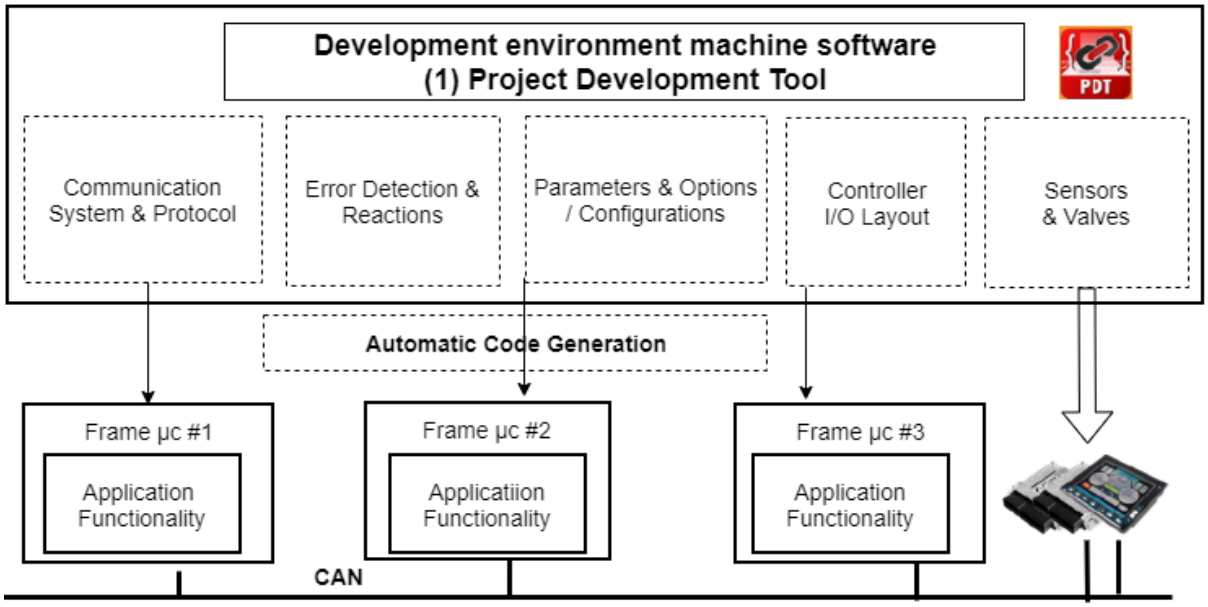

Figure 9.*Overview of application code frame [21, p.12, modified diagram]*

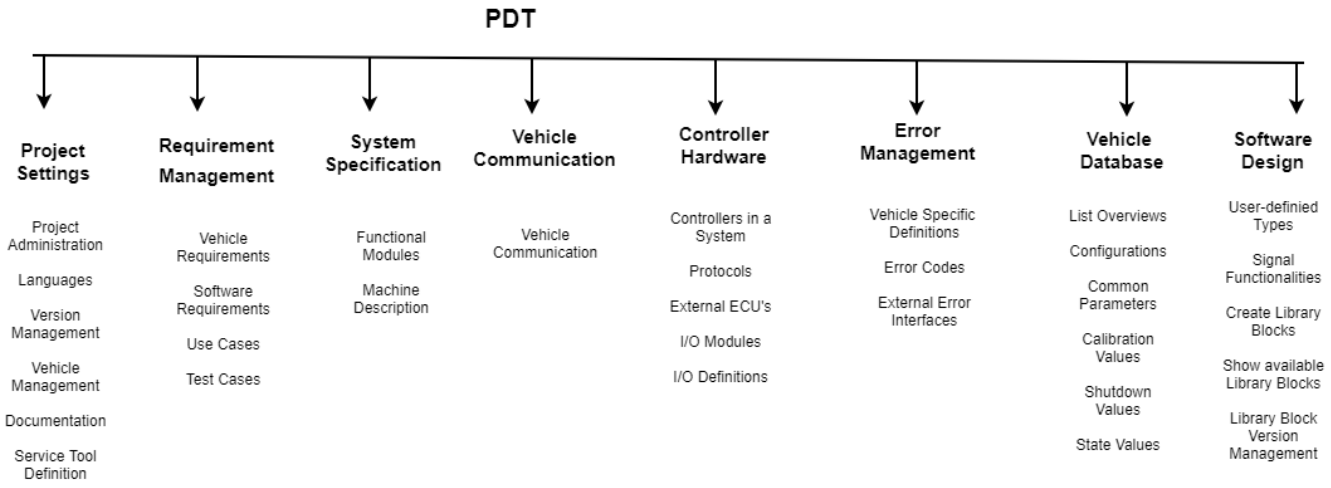

Figure 10.P*DT overview*

Figure 10Figure 10, gives an overview of a PDT tree. Project Settings, Requirement management, system specification, vehicle communication, controller hardware, error management, vehicle database, and software design are different features of PDT which are described briefly [21].

4.4.1 Project Settings

Inside Project settings, information's regarding its applications, system integrator, layout for language support, documentation, and interface for machine service tools are available.

4.4.2 Requirement Management

All the essential requirements which are needed for project development are managed inside PDT. In a project, requirements are estimated after hazard calculation and risk analysis. Important principles and terms related to the project development are derived from the V-model structure. Steps followed during a project development phase is described in V-model. Every V-model structure consists of different layers. In this case, we use four different layers inside a V- model as shown in Figure 11 below.

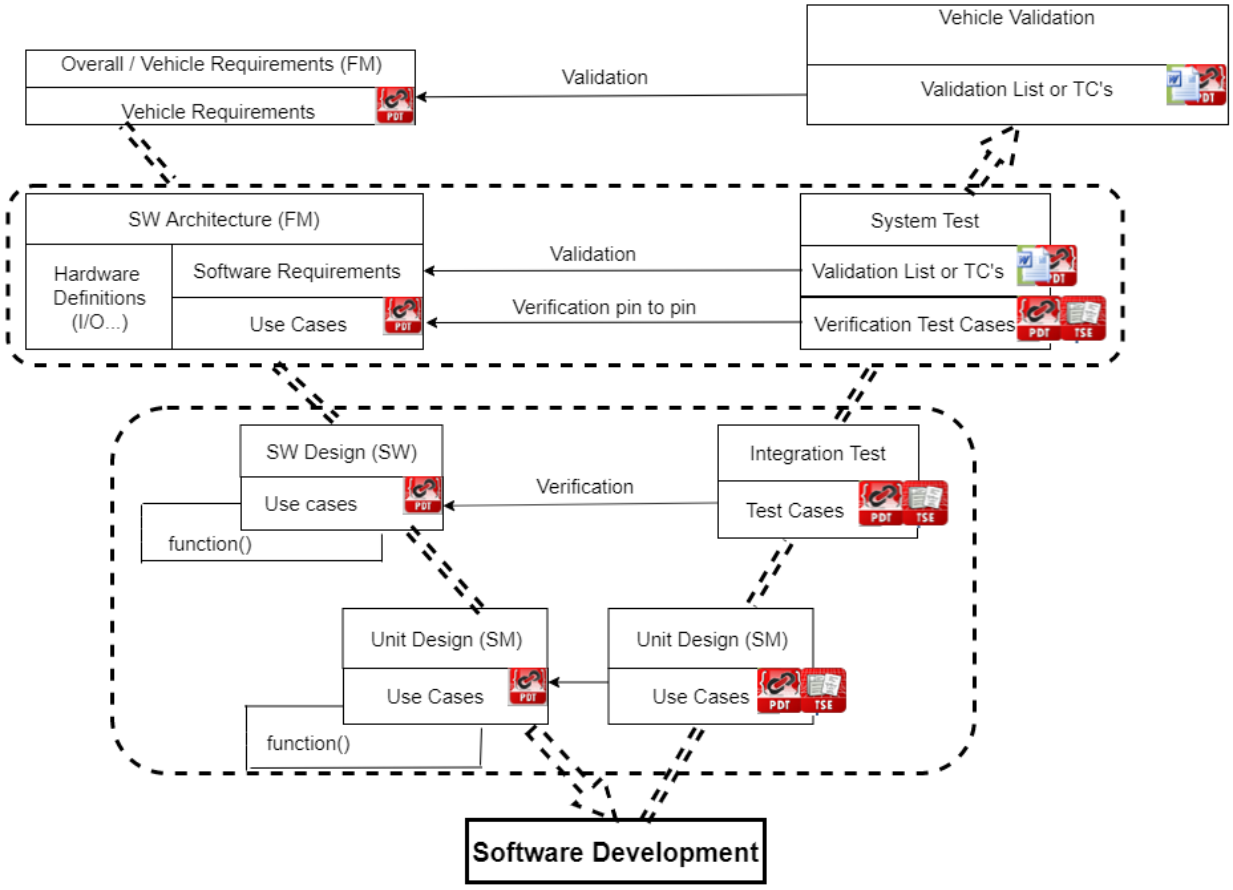

Figure 11.*V-Model based on system requirements for MATCH [21, p.97, Figure 1]*

Vehicle Layer, which forms the highest layer in the V-model structure with requirements in the first step and validation at the last step of development- After the Vehicle layer, follows the Architecture layer.

- Architecture layer consists of software architecture and system tests. Further, according to use cases the hardware system definitions and software requirements are implemented inside software architecture. During the development phase, the simulation of the pin to pin test can be conducted in TSE.
- The software design layer comes after the Architecture layer. The left development module consists of software design and the right side with an integration test
- The Unit design layer is responsible to perform small tasks independently for example a simple task with one function and no callbacks.
- The lowest layer is the software development where the actual coding for the software development occurs.

4.4.3 System Specification

The functional module and machine descriptions are two different labels that PDT supports. The functional module is a label within an application code. It helps to distinguish between different machine functions. The functional module is independent of different ECUS'. The machine description label provides an internal view of the machine which mainly aims for service technicians to operate.

4.4.4 Vehicle Communication

Vehicle communication contains information on CAN (Controller Area Network). It is a robust vehicle bus that allows communication between different ECUs', displays, input devices, and other components. The message-based protocol designed for multiplex electrical wiring within automobiles. Vehicle communication divides CAN communication into different nodes:

- CAN bus architecture is defined in Network topology
- CAN bus and its properties are defined by CAN ports
- All the CAN messages are stored in the CAN database

4.4.5 Controller hardware

The controller in system function in PDT provides a clear overview of all the ECUs' used in a single project. It allows us to add or delete them from a system whenever needed. Features of an ECU is displayed inside the system. Different protocol definitions like DM(x) (Diagnostic Message) protocol, UDS (United Diagnostic Services) protocol are also displayed within the system. UDS Protocol allows access to error information parameters and other predefined services by a tool connected to a CAN bus. Most importantly, it displays the Input/ Output pin definitions of the ECU.

4.4.6 Error Management

Error Management is broadly classified into Vehicle specific definition, Error Codes, and External error interface. The main responsibility of error management is to handle the detected error blocks.

Inside vehicle specific definitions there are multiple functions among which Set Conditions, Restricted Mode, and Release Condition are used mostly.

- Set condition is used for selecting errors that need to be handled
- Restricted mode is defined for central based error detection
- After the errors are handled, they are released free by release condition function

Error code, a part of error management is again sub-categorized into three parts namely Application Error List, Error List management, and Error Detection Method [21].

- When a system integrator assigns some error description, Failure Mode Integrator (FMI), and other specific properties to an error block from the error management it is termed as application error list.
- Error List Management stores and handles errors for every ECU in a system.
- Error Detection Method detects if any error has occurred in a single library block.

External Error Interface is an error message list other than the master list with all error codes defined in PDT. DM1 message List, Proprietary list are external error lists.

4.4.7 Vehicle Database

The vehicle database one of the advantages in MATCH. It makes all the variables and arrays used in the application code visible and can also be modified during application runtime with the help of MST. Old valves at the same time are backed up in the database. Vehicle database performs the following functions:

- It makes all the variable and arrays in a database list visible
- Adds new and manages existing database list
- Defines default datasets
- Defines factory settings

4.4.8 Software Design

The role of software design is to define constants and enumerations for a specific application, define block information, set software modules, etc. PDT plays a crucial part in developing a safety function in a control system. Starting from system specifications which includes the use and test cases, CAN messages, Error detection and the most important part which gives MATCH an upper hand compared to other methods is its safety certified functional blocks. These features are all used during the implementation phase of the thesis [21].

4.5 Machine Service Tool (MST)

MST is a PC program for mobile controller maintenance. Depending upon the version of MST it can support up to 8 ECUs' as well as displays. MST has four different versions available. They are:

- In service version both the hardware and software information of up to 8 ECUs' can be supported. It also displays the errors and states of an ECU. Service level is capable of supporting factory settings, Input /Output diagnostics, CAN messages, and many more.

- Developer version supports all the features of service version in addition to that it can download the project files, performing error tests and can also plot CAN bus data function plots.
- Designer version is capable of screen design and translation customization.
- Ultimate version supports unlimited functionalities of MST

Error Information Menu displays the most important error information's in MST. Error information includes error definition, active errors, list of all the errors that occur in a machine. While the system is running if the error pops up it is possible to disable the active errors or view the error history. During the software development phase, the system integrator can impose some conditions and test the software. The conditions are defined in PDT and later imported to MST [22].

5. ELECTRONIC CONTROL UNIT (HY-TTC 500) AND ITS SAFETY FUNCTIONS

The chapter aims to focus on the important concept of both hardware and software features inside an HY-TTC 500 ECU. It also provides some basic information about the safe state and safety functionalities of the controller.

5.1 Overview of HY-TTC 500 and its safety

HY-TTC 500 is a family of programmable electronic control units for sensors and actuator management. There are lots of I/O modules that are compatible with a wide range of sensors and actuators. The main objective to use the HY-TTC 500 family is that they are specially designed to be used in vehicles and heavy machines that operate in rough environments and extreme weather conditions.

HY-TTC 500 family consist of (HY-TTC 508, HY-TTC 510, HY-TTC 520, HY-TTC 540, HY-TTC 580, HY-TTC 590, and HY-TTC 590E). Currently HY-TTC 540 Ecu is used for the thesis work. The only difference between them is the I/O driver functions and party memory allocations but they are used for implementing the same safety mechanisms with regards to safety functions. The platform of HY-TTC integrates the Ecu with a safety-critical system. A system whose failure or malfunction can be fatal or leads to serious injury is termed as a safety-critical system. HY-TTC 500 is a generic control unit, which allows a diversified safety-critical application [25].

5.1.1 Hardware and software safety concepts

Both the hardware and software of the HY-TTC platform offers safety-critical applications they are:

- ECU Hardware: Hy-TTC 500 family is equipped with a main CPU and safety companion is form an external watchdog. It is used as a diagnostic and debugging interface which is very important for safety-critical systems.

- Bootloader: The bootloader is responsible for application software download, checking and then starting the application is it activated after the reset operation is performed.
- Board support package: It offers a startup code for the main CPU making a bridge between the application software and command files such as RAM, Flash and stack area
- FPGA bitstreams: HY-TTC has an additional logic IC which is used to get some feedback values for safety-critical outputs. The FPGA functions can be updated by flashing a separate FPGA bitstream to the ECU [25].
- I/O driver library: HY-TTC interface also provides a separate I/O library in C language consisting of different header files. The I/O drivers include,

  i) Software functions allowing access to Hy-TTC 500 interface and I/O from the application software. The application software includes PWM outputs, analog inputs, CAN communication, etc.)

  ii) Diagnostic modules that perform various I/O and CPU- internal tests at the start-up phase as well as during runtime [29].

5.2 Safe State and Safety Functions

In a safe state, there will be no current applied to the safety-critical outputs of the ECU, which implies when an error occurs the safety-critical outputs will be switched off. The safety-critical components are components which can cause possible hazardous conditions in case of failure. The components are as follows:

- Main CPU (including RAM and Flash)
- Safety Companion (Watchdog)
- Power supply and internal supply voltages
- Internal temperature monitoring

HY-TTC 500 platform is capable of distinguishing between different errors concerning momentary failure reaction. Error can be mainly categorized between Fatal and non-fatal errors.

Fatal errors are considered fatal when the safety program cannot be implemented anymore regardless of the actual application. When the safe program execution per second is unaffected due to some error then it is considered as non-fatal. This error can non-fatal because they might be located outside of the ECU, meaning an open circuit of a safety-critical actuator or it can be related to a contained subsystem of the ECU. When a fatal error occurs, the application software is notified by a notification callback from the I/O driver diagnostics. Whereas, when a non-fatal occurs the I/O driver executes an application callback. In the case of non-fatal error, the safety function does not enter a safe state [26].

For the Generic implementation of a safety function in the thesis work, TTC 500 safety characteristics and its I/O drivers are used in the thesis. Some of the most important safety-related data structures which are implemented are IO_ADC_SAFETY_CONF (Input Output Analog to Digital Safety Configuration) which stores all the relevant safety configurations for the ADC input, IO_DRIVER_SAFETY_CONF (Input Output Driver Safety Configuration) this is used to the safety-critical applications to the I/O drivers.

6. IMPLEMENTATION

The objective was to implement a safety functionality in a spraying machine with a Generic and MATCH based approach. Implementation of the safety function was based on EN 12001:2012 standards. EN 12001:2012 is a safety requirement standard for implementing a safety function in heavy earth moving or stationary machines are of type C which means only applicable for a specific group of machines like spraying, conveying, placing, and delivery line machines. According to the safety requirements the control system safety should have a minimum of **PLc** standard. Since the main objective of the thesis work was to implement and compare a safety function with two different approaches. So, implementation of the safety function and comparative study based on Qualitative analysis and user studies was mainly focused on. For a better understanding of a reader an example from a real working environment was given where the developed safety function could be implemented. To demonstrate one of the applications that can be formed by the safety function an imaginary situation was considered where a double-acting cylinder machine was assumed to control the boom cylinder of a spraying machine with the help of a double signal joystick and a 4/3 proportional solenoid valve, taking into consideration the safety requirements. V-model methodology was followed to implement the safety-critical function. To reach a minimum of PLc performance level factors likes $MTTF_d$, DC, CCF values of hardware components were calculated to ensure that the

required performance level was achieved for developing the safety software function. Misra C standards were followed for coding in the Generic approach. To ensure the code was implemented correctly in Generic implementation testing of the functional modules was done in the UNITY platform. Moreover, it was necessary not only for Generic but MATCH implementation to validate the results from each step as described in V-model architecture. MATCH testing was done with the MST tool and with PCAN_view software. A PCAN-view is a software which is used for monitoring CAN messages like viewing, transmitting, or recording CAN data [27]. With the help of PCAN-view the developer could view all the CAN message from inputs and outputs and monitor them to ensure the outputs and inputs of the CAN messages are according to developers' expectations.
Misra C guideline was followed for coding a safety function for Generic implementation. Some of the guidelines were violated but only belong to the advisory category and with proper supervision of experts. UNITY is designed to run a test suite, written in C language according to ANSI standard. Unity supports most embedded compiler traits. The key feature of unity is it has a set of assertions to perform test cases. Assertions are statements that the system developer expects to be true about the embedded system [28].

6.1 Overview of Comparative Study

The objective of a comparative study was to find a preferred approach in terms of implementing a safety-critical function in a control system. A comparative study was performed by two different methods, Qualitative analysis, and usability study. Before the actual implementation started specific parameters for comparing were decided which were common for both the implementation methods. Quality of documentation for both the methods, lines of code, testing both the methods are important factors to be compared. Efforts (as a measure of time taken) needed to achieve the parameters were recorded. The author himself was also the actor as well as the observer at the same time to compare the efforts needed in Qualitative analysis. So, to ensure that the efforts involved in comparing the parameters were done properly the actor ensured to keep track of time while starting each phase of implementation (like literature study, documentation, coding etc.) for each method separately. Efforts that were common for both the implementation approach was not taken into consideration like literature study of machine directives, designing the system architecture, choosing appropriate hardware components according to safety levels (shown in chapter 6.3). Efforts were calculated based on followed parameters:

- The effort needed for related literature study of MATCH related documents (like PDT, MST, MATCH, and safety manuals), similar efforts needed for literature study in the Generic approach (TTC 500 user, safety, I/O manuals) were recorded specifically.

- Efforts involved in documentation for MATCH (details in MATCH documentation is elaborately explained in chapter 6.2.1) and Generic documentation
- When the coding started separately the actor recorded the starting and ending time separately for both the methods

There were obstacles and challenges the author had to face to fulfill the objective of the thesis. Challenges were faced mainly during implementation. Some of the major obstacles are listed below:

- Getting the DC and $MTTF_D$ values for the hardware manufacturers which are generally not open-source took a bit of time.
- During MATCH and Generic implementation (especially error handling in the Generic approach and implementing some safety features inside PDT in MATCH approach) some difficulties were faced by the author from getting proper support from the specialist remotely.

As stated earlier by BR automation that one of the challenges to implement a safety function is, it is time-consuming. So, the effort needed to implement the safety function with both MATCH and Generic approach was measured. The authors from his point of view compared the expertise needed which meant coding skills, technical knowledge, training required for implementing a safety function using both the methods. If any approach needs too much training, experience, and skills then usually there is a lot of efforts involved in the process, so the aim was to find which approach demanded less effort. Further, to implement any safety function documentation is very important so the quality of documentation in terms of readability, elaborate and descriptive information's of all factors like Error handling, Input/Output pin configuration, CAN message details, etc. was measured in terms of efforts involved to make a preferred documentation with both MATCH and generic approach. Lines of codes and complexities required for implementing the safety function with both the methods was measured in terms of efforts involved. Lesser code with the same output from both the methods signifies the approach is simpler to implement with lesser complexities so lines of codes were compared. Usability study was performed by interviewing four developer specialized in MATCH and Generic methods. The objective of a usability study was to get a valuable insight into the knowledge of both the approaches for implementing a function in a control system through interviewing experts. Two engineers with expertise in MATCH and two engineers experienced in Generic implementation was interviewed. Based on the answers from user study and Qualitative analysis a conclusive result was drawn to figure out the most preferred approach to implement a safety-critical software function.

6.2 Overview of Implementation phase

The main objective of the thesis work was to compare MATCH and Generic approaches for implementing a safety function in a control system. For that a preferred safety function was decided that could be implemented using both the approach taking into consideration the resources available which means the input, output, and logic parts. The author did not have much experience in implementing a safety function so, the literature study of machine standards (mainly ISO 13849-1:2015) and coding standard (Misra C) was done. For Generic implementation, an initial literature study mainly focusing on the controller manual, I/O driver manual and safety manuals based on which the safety function was implemented. A similar study was also done for MATCH based approach (MATCH user manual, MATCH safety manual, PDT manual etc.). V-model methodology was chosen to implement a safety function for both the approaches. Since, MATCH already supports V-model methodology it was wiser to choose the same for the Generic approach also. Moreover, ISO 13849-1:2015 standard also recommends the use of a V-model methodology form implementing a safety function. After, all the literature studies were done a high-level architecture was designed based on input, logic, and output components. Specific components were chosen based on safety requirements as shown in Table 3. Then requirement was common for both the implementation since the same safety function was planned to be implemented. Then a more detailed low-level architecture was designed based on the requirements that needed to be followed for implementing the safety function.

6.2.1 MATCH Implementation Overview

The first step for implementing any software function in MATCH start with creating a project in PDT. The developer needs to add the controller that was used as a logic unit for implementing the software function. Currently MATCH is compatible with TTC controllers only, but exceptions can be made through a special developer contract from the company. Then requirement was added in Requirement Management according to the function that needs to implement followed by Use and Test cases. The developer then connected the Input and Output pins inside controller hardware. Inside PDT there already exists a list of I/O pins available in the printed control board. So, the developer needs to connect the inputs and outputs which are used in the project. MATCH provides pre-certified functions blocks such as InVoltDbl (Double Input Voltage) which was used as an input, Prop (Proportional) output, etc. Errors corresponding to these blocks are already defined inside these blocks. After selecting the preferred blocks that were needed for implementation, CAN messages were defined manually by the developer inside vehicle communication. Restriction Mode which is a pre-defined machine reaction caused by an

“active” error code is also added by the developer. The application checks the restricted mode and if the restricted mode is active, it handles the application errors for example the developer can set the output to zero if restriction mode is active. After all the necessary information was included in the PDT project, the developer after building the project an Auto Code Builder (ACB) was generated. The Code generated from PDT is in C language and follows some coding standard similar to Misra C standard and it contains all information of the functionals blocks (such as Errors, restriction mode etc.), CAN messages that are created inside PDT. The logic behind the task based on the requirement now was implemented. Testing of the software function is mainly done with the TSE tool but due to lack of training, testing was done through PCAN-view and MST tool. Documentation is automatically generated from the PDT project after it is built. It contains all the information the developer included in the PDT project such as requirements, use and Test cases, I/O pins, CAN messages, Error list, etc.

6.2.2 Generic Implementation Overview

The first step the author implemented was to start documenting the use case then the corresponding test cases that need to be tested to find out whether the ´safety function was implemented correctly or not. Figuring out first how the CAN messages are defined followed by the input and output signals was quite a challenging task initially. Coding the safety function was done following guidelines from Misra C standard. After the code was implemented a separate test assertion was designed in Unity to test the functionalities according to the use and test case that were defined earlier.

The author was also the observer at the same so, a proper track of time to implement both the methods was kept starting from documentation till the end of the testing phase.

6.3 System Requirements

For implementing a safety function for a control system, the system integrator should develop a system that is preferred to perform a specific task then define the system requirement based on safety criteria. Similarly, a system consisting of input, control system, logic, and output was developed.

In the implementation phase of the thesis V-model approach was followed for implementing a safety-critical safety function, the structure is explained in Figure 1 and Figure 11 the topmost layer of the V-model is System specification. Before the actual implementation, the components were selected to meet the required safety level. The hardware components used for implementation are cross monitoring input signal joystick as an analog input and Ecu as a logic unit and a 4/3 proportional valve as output.

6.3.1 Safety requirement calculation

Component specifications are as follows:

***Table 3**. COMPONENTS USED WITH SAFETY SPECIFICATIONS*

COMPONENT TYPE	$MTTF_D$* (No. of years)	DC* (in %)	CCF* (in points)	COMPONENT ROLE
Caldaro C15 Joystick	520.8	90	>=65	Input Device
TTC 540 ECU	42.33	90.39	>=65	Logic
P4WEH 10 bis 32	150	----------	>=65	Output device

*$MTTF_D$- Mean Time to Dangerous Failure

*DC – Diagnostic Coverage

*CCF- Common Cause Failure

*PL- Performance Level

The DC value for the Joystick and the valve is estimated from Annex E Table E.1 from ISO 13848-1:2015.

The DC value of the proportional valve is not taken into consideration as without a position sensor or a similar type of sensor it is not possible to detect the spool position weather it is closed or opened. Due to the limited scope of thesis work sensors are not used to detect the spool position of the valve so DC value is not considered.

CCF values are calculated form ISO 13849-1:2015 standards referring to Annex F from Table F.1 which is explained briefly in the latter part.

The formula to calculate the $MTTF_D$ value is referred from ISO 13849 is shown below:

$$\frac{1}{\mathrm{MTTF_D}} = \sum_{i=1}^{\bar{N}} \frac{1}{\mathrm{MTTF_{D}}_i} = \sum_{j=1}^{\bar{N}} \frac{n_j}{\mathrm{MTTF_{D}}_j} \quad \text{..... (Equation 1)}$$

Where,

$MTTF_D$ is for the complete system

N refers to the number of components used.

Hence the $MTTF_D$ for the system is:

$$\frac{1}{MTTF_D} = \frac{1}{MTTF_{D1}} + \frac{1}{MTTF_{D2}} + \frac{1}{MTTF_{D3}} \text{..... (Equation 2)}$$

$MTTF_{D1}$= 85.9 years. (The value is obtained from Caldaro C15 Joystick datasheet)

$MTTF_{D2}$= 42.33 years. (The value is obtained from TTC 500 safety manual)

$MTTF_{D3}$= 150 years. (The value is obtained from HYDAC P4WEH 10 bis 32 datasheet)

From Equation 2 and the $MTTF_D$ values of the components the $MTTF_D$ value of the system is:

$$\frac{1}{MMTF_D} = \frac{1}{520.8} + \frac{1}{42.33} + \frac{1}{150} = 0.032 \text{.................................. (Equation 3)}$$

$$MTTF_D = 31\ Years \text{.. (Equation 4)}$$

According to ISO 13849 standard the estimate of average Diagnostic coverage for a system is shown in the equation below:

$$DC_{avg} = \frac{\frac{DC_1}{MTTF_{D1}} + \frac{DC_2}{MTTF_{D2}} + \ldots + \frac{DC_N}{MTTF_{DN}}}{\frac{1}{MTTF_{D1}} + \frac{1}{MTTF_{D2}} + \ldots + \frac{1}{MTTF_{DN}}} \text{.............. (Equation 5)}$$

When N = number of components in a system =3

$$DC_{avg} = \frac{NUMERATOR}{DENOMINATOR} \text{... (Equation 6)}$$

$$\text{NUMERATOR} = \frac{90}{520.8} + \frac{90.39}{42.33} = 2.308 \text{.......................... (Equation 7)}$$

$$\text{DENOMINATOR} = \frac{1}{520.8} + \frac{1}{42.33} = 0.0255 \text{ .... (Equation 8)}$$

Therefore, combining Equation 7 and Equation 8 we get:

$$DC_{avg} = \frac{2.308}{0.0255} = 90.51\% \text{.. (Equation 9)}$$

According to safety standards described in ISO 13849, $MTTF_D$ values above 30 years are considered as high, similarly, DC_{avg} values between 90 and 99 percent are considered as Medium as referred from Table 4 and Table 5 respectively. CCF is >= 65 for the safety function. The CCF value is obtained from Table F.1 in ISO 13849-1:2015 standard taking into considerations factors like Separation, Design/application/experience, Assessment/analysis, Competence/training, Environmental conditions and other influences which adds up to 80 score, only diversity is not considered as the safety function I have considered does not satisfy that. According to Equation 4 and Equation 9 the $MTTF_d$ and DC_{avg} values are 31 years and 92.18% respectively from Tables 4 and 5 from ISO 13849:1-2015 standards shown below we can choose the preferred $MTTF_d$ and DC denotation for the safety function. Table 6 elaborated in figure 6 of ISO 13849-1:2015 and comparing the values of $MTTF_D$, DC_{avg} and the Category is of type 2. So, the Performance Level (PL) of the system is **PL d**.

***Table 4**. MMTFd classification for each channel [9, p.17, Table 4]*

$MTTF_D$	
Denotation of each channel	**Range of each channel**
Low	3 years ≤ $MTTF_D$ < 10 years
Medium	10 years ≤ $MTTF_D$ < 30 years
High	30 years ≤ $MTTF_D$ ≤ 100 years

***Table 5**. DC classification for each channel [9, p,17, Table 5]*

DC	
Denotation	**Range**
None	DC < 60 %
Low	60 % ≤ DC < 90 %
Medium	90 % ≤ DC < 99 %
High	99 % ≤ DC

***Table 6**. PL evaluation for the whole system [9, p.19, Table 6]*

Category	B	1	2	2	3	3	4
DC_{avg}	none	none	low	medium	low	medium	high
$MTTF_D$ of each channel							
Low	a	Not covered	a	b	b	c	Not covered
Medium	b	Not covered	b	c	c	d	Not covered
High	Not covered	c	c	d	d	d	e

6.3.2 System Architecture

The category of the Joystick using is 3 according to safety standards and is explained in Figure 12. TTC 540 follows category 2 safety according to their safety manual description.

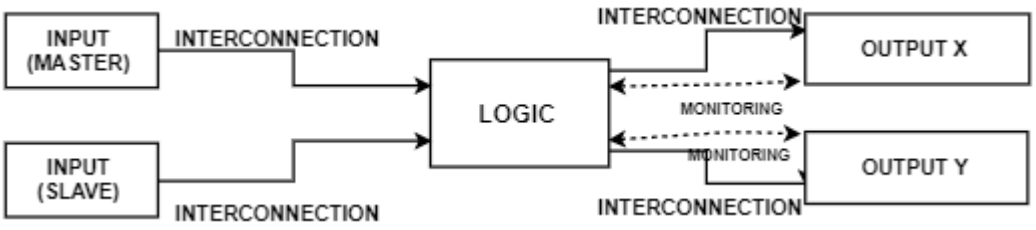

***Figure 12.**Category 3 architecture of an Independent double output Joystick*

For implementation of a control system, Category 2 was chosen which was best suited for the system as there is one logic and one testing platform. The testing platform normally for MATCH platform is Test and Simulation Environment (TSE) but due to limited time for thesis and lack of training testing in TSE was not possible, so testing was done through Machine Service Tool (MST) and PCAN-view. For testing in GENERIC imple-

mentation UNITY was used, as it supports the testing of embedded software that is written in C programming language. The high-level system architecture is shown in Figure 13 followed by Low-level architecture shown in Figure 14 which gives a clearer view of the system.

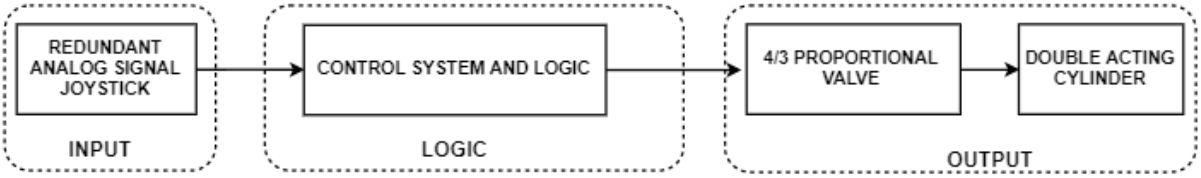

***Figure 13**. High level architecture*

High-level architecture helps to provide an overview of the entire system and giving an idea of the overall system. Whereas low-level architecture provides step by step information on the development process. The software architecture involved in the system is described in a more detailed manner.

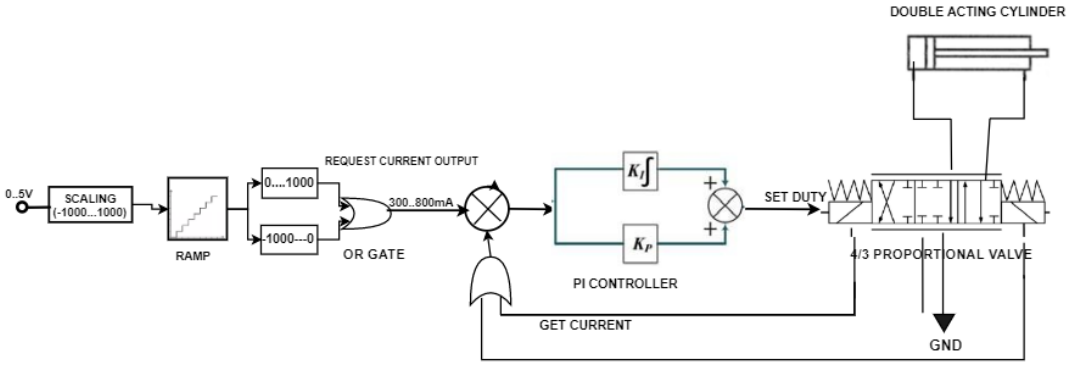

***Figure 14.**Low level architecture*

The purpose of implementing a low side PWM output is when an overcurrent is detected, the low side is open which safeguards the electric circuit from internal damage. In the thesis work the actuator as shown in Figure 15 is a proportional valve, one side is connected to high side output, and the other end to the low side output.

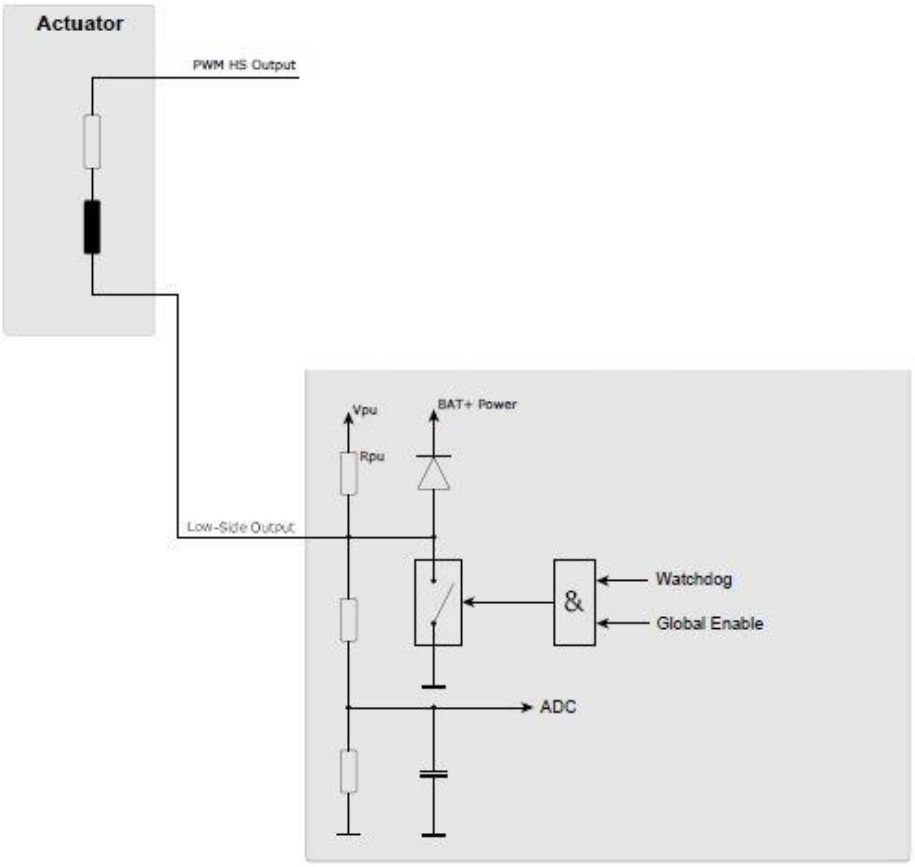

***Figure 15**.Low side digital output, from table 4.31 TTC 500 User Manual*

6.4 Coding and Documentation

Chapter 6.4 mainly focuses on the bottom-most layer of the V-model methodology which is coding for developing a safety-critical function. Coding was implemented separately with two different approaches Generic and MATCH, respectively. Documentation of both approaches was made to make it readable and increase the traceability during the development phase of the safety function.

6.4.1 Generic coding and documentation

Before any actual development of software, and the overall requirement is necessary which provides the necessary information to a developer about the key features that need to be implemented in software. In the thesis a safety-critical software function in a control system is implemented so it is important to figure out the overall requirement the system needs to fulfill. below will show the overall requirement which are designed for the safety function. Requirements include safety functions whose failure can result to risk. Each safety function must be associated with a performance level. The functional requirements are described below:

***Table 7**. Overall Requirement*

Requirements	Performance Level	Description
R1	PLc	When some error occurs in the safety function either from output or input the system stops immediately i.e the output is zero (SAFE STOP)
R2	PLc	There will be uniform acceleration or deceleration in the speed, sudden jerk in the system is prevented (UNIFORM ACCELERATION)
R3	PLc	Double voltage input function block, where the upper and lower limits are set as well as checking the permissible deviation between slave and master voltage. If there is more deviation error occurs
R4	PLc	Proportional output block, where the PI controller is programmed to limit the duty cycle

Coding is a core part of the implementation phase, as it is the bottommost layer of a V-model methodology, which was followed in the thesis. For coding, a safety-critical function in a control system in a generic way, TTC 500 I/O driver libraries were used. C programming was used as a programming language. MISRA C guideline was followed while coding the safety functions. Implementation was done based on requirements as elaborated in Table 7. Table 8 shows the list of use cases that are used for developing the safety functions that are derived based on the requirements. Use cases are designed before the actual coding of the safety function according to the V-model methodology. Use cases provide mainly details of safety and non-safety requirement but in the thesis work only safety requirements were focused on. It provides traceability in the development process.

***Table 8.** USE CASES*

Use Case	Description
SR1	Error in input Pins (Watchdog errors and PIN errors)
SR2	Error in Output Pins (Watchdog errors and PIN errors)
SR3	Using Ramp function to ensure there is no jerk in the system maintaining uniform acceleration or deceleration
SR4	Checking if there is a permissible deviation between slave and master voltage

SR5	Checking the upper and lower limits of master voltage.
SR6	Checking the PI controller for both the valves

Where,
SR Safety Requirement

Table 9 provides information about the input and output pins used during software implementation.

Table 9. *PIN Information in Generic approach*

Pin	Description	Pin Information
IO_PIN_103	Joystick x-axis master voltage	Ratiometric 0….5V Analog Input
IO_PIN_127	Joystick x-axis slave voltage	Ratiometric 0….5V Analog Input
IO_PIN_177	Proportional output for solenoid 1	High side PWM output
IO_PIN_165	Proportional output for solenoid 2	High side PWM output
IO_PIN_241	Low side output for solenoid 1	Low side digital output
IO_PIN_251	Low side output for solenoid 2	Low side digital output
IO_PIN_234	Sensor supply for master Voltage	Sensor supply 5V
IO_PIN_247	Sensor supply for slave Voltage	Sensor supply 5V

At first a CAN message is written to an ECU. Figure 16 below is an example that demonstrates how a CAN message is initialed and written. Generic implementation was done with TTC I/O drivers, where the length of a data accept is 8 bits so, a message of 16 or 32 bits are divided into segments of 8 bits and wrote in a CAN message. It demonstrates the initialization of a CAN message with IO_CAN_Init function which passes on parameters as shown below.

```
// Spliting 16 bits into segments of 8 bits each
void split_16bites(ubyte2 param, ubyte1* splt_1, ubyte1* splt_2){

    *splt_1 = param & 0xFF;
    *splt_2 =param >> 8;
  }
//  CAN message for transmitting */
   IO_CAN_Init(IO_CAN_CHANNEL_0, IO_CAN_BIT_250_KB, 0,0,0,0);
   IO_CAN_ConfigMsg(&Raw_scaledVolt,IO_CAN_CHANNEL_0,IO_CAN_MSG_WRITE, IO_CAN_STD_FRAME,0,0);
  //MASTER & SLAVE voltage (Raw and scaled voltage)
  split_16bites(u16Volt_master, &can_frame.data[0], &can_frame.data[1]);
  split_16bites(u16Volt_slave, &can_frame.data[2], &can_frame.data[3]);
  split_16bites(Master_volt, &can_frame.data[4], &can_frame.data[5]);
  split_16bites(scaled_volt_Master, &can_frame.data[6], &can_frame.data[7]);
    can_frame.id = 1;
    can_frame.id_format = IO_CAN_STD_FRAME;
    can_frame.length = 8;

  IO_CAN_WriteMsg(Raw_scaledVolt,&can_frame);
```

Figure 16.*Initialization and execution of CAN messages*

```
IO_ErrorType IO_CAN_Init ( ubyte1 channel,
                           ubyte2 baudrate,
                           ubyte1 tseg1,
                           ubyte1 tseg2,
                           ubyte1 sjw,
                           ubyte1 brp
                         )
```

Where,

Channel	CAN channel
Baudrate	Baud rate in kbit/s
Tseg1	Time segment before sample point (3 ... 16)
Tseg2	Time segment after sample point (2 ... 8)
Sjw	Synchronization jump width (1 ... 4)
Brp	Baud rate prescaler (1 ... 64)

```
IO_ErrorType IO_CAN_ConfigMsg ( ubyte2 *const handle,
                                ubyte1        channel,
                                ubyte1        mode,
                                ubyte1        id_format,
                                ubyte4        id,
                                ubyte4        ac_mask
                              )
```

Where,

Handle	Returns the message object handle
Mode	Mode for CAN Message, for the thesis it is IO_CAN_MSG_WRITE
Id_formate	Format of message identifier, for the thesis it is IO_CAN_STD_FRAME
Id	CAN message identifier
Ac_mask	CAN acceptance mask

In the above a CAN message named Raw_scaledVolt is shown where the Data Length Code (DLC) is 8 and messages like u16volt_master, u16volt_slave, Master_volt and scaled_volt_Master are written in the CAN message.

The analog inputs from the joystick are initialized by IO_ADC_ChannelInit function where the analog input values are read from the pin. The code below illustrates the implementation in the thesis:

```
//  initialize the ADC inputs with safety configurations*/
    temp_adc.adc_val_lower =7;
    temp_adc.adc_val_upper =93;
    temp_adc.redundant_channel = IO_PIN_NONE;
    IO_ADC_ChannelInit(IO_PIN_103,IO_ADC_RATIOMETRIC,IO_ADC_RANGE_5V,IO_ADC_NO_PULL,IO_SENSOR_SUPPLY_0,&temp_adc);
    IO_ADC_ChannelInit(IO_PIN_127,IO_ADC_RATIOMETRIC,IO_ADC_RANGE_5V,IO_ADC_NO_PULL,IO_SENSOR_SUPPLY_1,&temp_adc);
```

IO_ADC_ChannelInit function passes parameters like adc_channel where the specific pin information is given which is attached to the printed control board, the type of voltage input which is ratiometric which is this case, range of the voltage which is 0 to 5000mV so IO_ADC_RANGE_5V is added since there is no pull-up or pull-down resistor added so IO_ADC_NO_PULL is passed inside the function, analog inputs are added with the sensor supply of 5V since the initialization is done under safety configuration so that modules can be checked internally if the inputs parameters are invalid range so temp_adc parameters with adc_val_lower and upper values are passed which are measured in terms of percentage. From the FIGURE above we can check that the adc_val_lower =7 which signifies 7% of the lower limit and 93% to maximum limit are within the valid range. The redundant channel is set as IO_PIN_NULL which is set to 0 because the inputs are two analog redundant signals which are named as master and slave, for redundant signals the parameter is set to 0. As the objective is to develop a safety-critical function in a control function so, IO_DRIVER_SAFETY_CONF is added. It is a structure that is used for the configuration of a safety-critical function in an IO driver. The parameters which are passed in the IO_DRIVER_SAFETY_CONF are shown in Figure 17.

```
/* safety configuration */
IO_DRIVER_SAFETY_CONF safety_conf = {0};

//     cycle period: 10 ms with 25% percent window size, no resets, 30 ms glitch filter, with error
//     * and notification callback */-----------------------------------------------------------------
safety_conf.command_period      = 10000;
safety_conf.window_size         = SAFETY_CONF_WINDOW_SIZE_25_PERCENT;
safety_conf.reset_behavior      = SAFETY_CONF_RESETS_DISABLED;
safety_conf.glitch_filter_time  = 30;
safety_conf.error_callback      = APPL_ErrorCb;
safety_conf.notify_callback     = APPL_NotifyCb;

//     initialize IO driver with safety configuration */
io_error = IO_Driver_Init(&safety_conf);
```

Figure 17*.Safety configuration*

Inside the safety configuration structure command_period which is time in [us], the interval between two consecutive software cycles, window size which is a watchdog window size, rest_behavior which reset the watchdog for this thesis implementation it is disabled, glitch_filter_time which signifies that only if an error condition persists after expiration of this time range, an error reaction is considered and most importantly error_callback and notify_callback is passed into the structure. Error_callabck is a non-fatal error and notify_callabck is a fatal error. These callbacks are explained in more detail in the later phase. To create a functional block, an ADC_status function was de-

veloped which takes dual independent joystick inputs in terms of slave and master voltage. The function checks the error, boundary, neutral, and deviation conditions according to the information provided in the datasheet as shown in Figure 18 below.

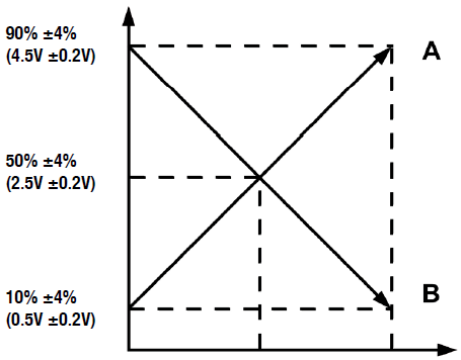

Figure 18*.Tolerance limits with max and min voltage from C15 product sheet*

The code implementation of the ADC_status function is shown in Figure 19. The ADC_status was developed to get the raw voltages (both salve and master) output from the double signal redundant joystick as an input of the function then calculate all the limits and its deviation and return a single output voltage (in this case it was master voltage).

```
/* The function takes two inputs in terms of Master and Slave voltage ,
 *  checks the error limits, boundary conditions most importantly deviation and returns
Deviation is taken as the summation of measurement error, Zero input input error and propoortional error
since from the caldaro datasheet the tolerance limit is +-200mV , from ttc 500 manual the Zero input error is +-15mV
and the proportional error is 0.2% of the total analog input +-10mV */
ubyte2 ADC_Status(ubyte2 master, ubyte2 slave){
    ubyte2 output =0;
    const ubyte2 REFERENCE_LINE = 2500;
    const ubyte2 MINIMUM_LIMIT = 300;
    const ubyte2 MAXIMUM_LIMIT =4700;
    const ubyte2 PERMISSIBLE_LOW = 500;
    const ubyte2 MAX_PERMISSIBLE_LOW = 700;
    const ubyte2 LOW_PERMISSIBLE_HIGH = 4500;
    const ubyte2 PERMISSIBLE_HIGH = 4500;
    const ubyte2 DEVIATION_LIMIT =450;
    const ubyte2 DEADBAND_MIN = 2300;
    const ubyte2 DEADBAND_MAX =2700;
    sbyte2 slave_volt =(slave-REFERENCE_LINE);
    ubyte2 deviation;
    sbyte2 slave_mirror = REFERENCE_LINE- slave_volt;
    deviation = abs(master - slave_mirror);
    if (master < MINIMUM_LIMIT || master > MAXIMUM_LIMIT){
        output = UBYTE2_MAX_VALUE;
    }
    else if ( slave < MINIMUM_LIMIT || slave > MAXIMUM_LIMIT){
        output = UBYTE2_MAX_VALUE;
    }
    else if (deviation > DEVIATION_LIMIT){
        output = UBYTE2_MAX_VALUE;
    }
    else
    {
        if (master>=MINIMUM_LIMIT && master < MAX_PERMISSIBLE_LOW){
            if (master >=MINIMUM_LIMIT && master <PERMISSIBLE_LOW){
                output = PERMISSIBLE_LOW;
            }
            else if (master >=PERMISSIBLE_LOW && master < MAX_PERMISSIBLE_LOW){
                output = master;
            }
        }
        else if (master >=DEADBAND_MIN && master <= DEADBAND_MAX){
            output =REFERENCE_LINE;
        }
        else if (master > LOW_PERMISSIBLE_HIGH && master <= MAXIMUM_LIMIT){
            if (master > LOW_PERMISSIBLE_HIGH && master <=PERMISSIBLE_HIGH){
                output = master;
            }
            else if ( master > PERMISSIBLE_HIGH && master <=MAXIMUM_LIMIT){
                output = PERMISSIBLE_HIGH;

                }
            }
            else
            {
                output = master;
            }

        }
        return output;
    }
```

Figure 19. *ADC_Status Function*

For calculating the permissible deviation from the joystick in ratiometric mode, conversion error of the HY-TTC 500 and the sensor supply error are included. For the calculation of the total measurement error, the error of HY-TTC 500 and the error of the ratiometric sensor (joystick tolerance) must be added [13]. From Figure 18 it is clear that the maximum voltage output can be 4500mV with ± 200 mV tolerance minimum is 500mV with ±200mV tolerance. The zero input error is ±15mV according to the TTC500 manual, which is added along with the Proportional error of ±0.2%. The middle position is at 2500mV, tolerance limit being the same. For that reason the implementation was computed such that anything above 4700mV or below 300mV was considered an error, further tolerance limit anything between 2300 to 2700 mV was considered the neutral position of the joystick. The deviation limit was chosen to be 450mV based on the tolerance limit of the joystick and Zero input error and proportional error. The output goes the master voltage as a slave is used to compare the deviation between the two voltages. Now scaling is done to the output of the ADC_status function which is the master voltage coming out from the joystick from -1000 to 1000. As well as a part of scaling is also performed to scale the requested ramp output current which is scaled from 300 to 800 mA. The code shown in Figure 20 below illustrates the implementation of the function as well as the formula used to scale. According to the requirements a ramp function is then implemented to prevent a sudden increase or decrease of output scaled voltage from the joystick. A step increase or decrease with a step size equal to 1 is implemented, Figure 21 below shows the implementation inside the software.

```
 /*Formula used for scaling (Max value is 4500mV, Min Value is 500 mV and scaling is done between -1000 to 1000:
 (Input- Minimum Value)
----------------------------* (Maximum scaled Value- Minimum scaled Value)+ Minimum Scaled value
(Maximum value- Minimum Value)          */

 sbyte2 scaled_master(ubyte2 master){
     const  ubyte2 Minimum_volt = 500;
     const sbyte2 minimum_scaled_value =-1000;
     sbyte2 Old_range_master = (master-Minimum_volt);
     sbyte2 scaled_volt_Master = (Old_range_master)/2 +minimum_scaled_value;
     return scaled_volt_Master;
}
 /*Formula used for scaling (Max value is 0, Min Value is 1000 and scaling is done between 300mA to 800mA:  */
 sbyte2 request_current(sbyte2 postive_scaled){
     const ubyte2 minimum_scaledcurr =300;
     sbyte2 reqst_curr = (postive_scaled)/2 +minimum_scaledcurr;
     return reqst_curr;
 }
```

Figure 20.*Scaled Function*

```
// Ramp function of scaled master voltage
sbyte2 ramp_output(sbyte2 scaled_master){
   static sbyte2 ramp =0;
   ubyte2 STEP_SIZE =1;
   sbyte2 ramp_out1;
   if (scaled_master-ramp  > STEP_SIZE){
       ramp_out1 =ramp+1;// Increase by 1

   }
   else if (scaled_master - ramp < STEP_SIZE){
       ramp_out1 =ramp-1; //decrease by 1
   }
   else{
       ramp_out1 = scaled_master;
   }
   ramp=ramp_out1;
   return ramp_out1;
}
```

***Figure 21**.Ramp Function*

After the ramp function is implemented, a 4/3 solenoid proportional valve is connected with a low side Pulse Width Modulation (PWM) output. The objective is to control the direction of the spool position in a valve due to current passing through the solenoid. As far as the P4WEH 10 bis 32 datasheet the current range is from 300 to 800 mA approximately so, the lowest current of 300mA and maximum of 800mA is considered during the implementation. The IO_PWM_InitWithLowside function initializes the PWM with low side and parameters that are passed inside the function are pwm_channel which is the pin configuration, PWM frequency (50Hz .. 1000Hz, only predefined frequencies with a period of an integral multiple of 1ms, 0.5ms or 0.25ms are possible), Polarity of the output signal, Indicate if a margin should be applied or not (for safety configuration this parameter is set as TRUE), similar to the ADC initialization IO_PWM_SAFETY_CONF checks the internal module like current check which is set as true so that it checks if there is any overload inside the PWM output or. A limiting current of 4000mA is set to prevent overcurrent etc.

The initialization of two PWM outputs is shown in Figure 22

```
IO_ErrorType IO_PWM_InitWithLowside ( ubyte1                    pwm_channel,
                                      ubyte2                    frequency,
                                      bool                      polarity,
                                      bool                      diag_margin,
                                      IO_PWM_SAFETY_CONF const *const safety_conf
                                    )
```

```
// initializing PWM output*/
    temp_pwm.enable_current_check = TRUE;
    temp_pwm.current_limit = 4000;
    temp_pwm.low_side_channel = IO_DO_15;//IO_PIN_NONE;
    // PWM initialization with Low side output
    io_error =IO_PWM_InitWithLowside(IO_PIN_177,200, TRUE, TRUE,&temp_pwm);

    temp_pwm_1.enable_current_check = TRUE;
    temp_pwm_1.current_limit = 4000;
    temp_pwm_1.low_side_channel = IO_DO_08;//IO_PIN_NONE;
    // PWM initialization with Low side output
    io_error =IO_PWM_InitWithLowside(IO_PIN_165,200, TRUE, TRUE,&temp_pwm_1);
```

Figure 22:Two PWM output initialization

Now, when the PWM is initialized, the task is implemented to control the spool of the proportional value by passing a current through the solenoid. For that a PI controller is implemented. A PI controller works as a feedback control loop which calculates the error signal (for the current system it is the desired current) by taking the difference between the output of the system (in this case it is Get current) and setpoint which is the desired ramp output current in the present context).

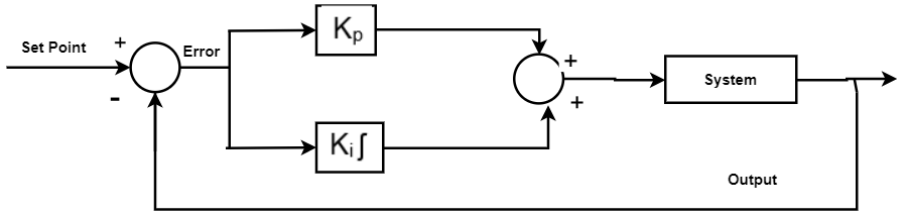

Figure 23.*Controller schematic diagram*

Figure 23 above shows the schematic diagram of a PI controller. During the implementation the K_p gain and to calculate K_i gain constants like dt, epsilon, proportional gain values are tuned from trial and error method to obtain a desired current value for the system. Figure 24 illustrates the implementation of a PI controller and the use of functions like get current and set duty. Get current function is used to return the measured current for a given channel in the context of the present system it is currently measured from PIN 177 and PIN 165. Set duty function sets the duty cycle of a PWM output. A duty cycle is the amount of time the signal is in a high state as a percentage of the total time it takes to complete one cycle.

```
        // Scaling of ramp_out between 300-800 mA
        Reqst_ramp_out = request_current(scaled_ramp_Master);
        // GETDUTY      &&&     GETCURRENT
        curr_out = Reqst_ramp_out-curr;
        pos_currout =abs(curr_out);
        if (abs(curr_out) > epsilon){
            integral = integral + abs(curr_out)*(dt/10);
            PI_output = P_Gain*abs(curr_out)+I_GAin*integral;
            get_duty_P =IO_PWM_SetDuty(IO_PIN_177, PI_output ,NULL, NULL);
            PI_output_1 =0;
            get_duty_N =IO_PWM_SetDuty(IO_PIN_165, PI_output_1 ,NULL, NULL);
            split_16bites(PI_output, &can_frame4.data[4], &can_frame4.data[5]);
            split_16bites(get_duty_P, &can_frame4.data[6], &can_frame4.data[7]);
        }
        else
            get_duty_P =IO_PWM_SetDuty(IO_PIN_177, PI_output ,NULL, NULL);
            PI_output_1 =0;
            get_duty_N =IO_PWM_SetDuty(IO_PIN_165, PI_output_1 ,NULL, NULL);
            split_16bites(PI_output, &can_frame4.data[4], &can_frame4.data[5]);
            split_16bites(get_duty_P, &can_frame4.data[6], &can_frame4.data[7]);
}
  // Measuring GET CURRENT FOR BOTH THE SOLENOID
  get_curr =IO_PWM_GetCur(IO_PIN_177, &curr, &fresh_curr1);
  get_curr_2 =IO_PWM_GetCur(IO_PIN_165, &curr_2, &fresh_curr2);
```

Figure 24*.Task implementation with PWM output*

The HY-TTC 500 I/O driver provides 2 application callbacks, which can be configured when calling the general driver initialization function IO_Driver_Init ().

The callbacks are executed depending on the error type before the safe state is entered:

- Error callback - executed for non-fatal errors
- Notification callback - executed for fatal errors

In case of error callback, the severity of an error in a high side PWM or a digital output is treated equivalent to a low side safety switch. Whereas, in notification callback there is no significant delay as no actions are taken directly. The purpose of this callback is to notify the application that a critical error has occurred, and the system will be triggered to a safe state. That means it sends debugging messages in the form of CAN messages. Table 10 and Table 11 shows the Error codes from the watchDog and corresponding I/O pins as per the descriptions which are implemented in the software. WatchDog error detection mainly happens during the startup phase but I/O pin error detection occurs during the runtime. Figure 25 and Figure 26 elaborates on the implementation of the callback function in an implementation.

Table 10. *WatchDog Errors*

PIN	Error Code	Description
PIN 103 PIN 127	1. DIAG_E_ADC_RANGE 2. DIAG_E_ADC_3MODE_SWITCH_TEST	1. ADC measurement range check error. This error is raised when the ADC value falls outside the range defined in the safety configuration of the pin. 2. Start-up test error with ADC mode selection switches. If this error is raised, the ADC channel in the ‘error_device` parameter is not functional.
PIN 177 PIN 165	1. DIAG_E_PWM_SHORT_CIRCUIT 2. DIAG_E_PWM_OPEN_LOAD) 3. DIAG_E_PWM_CURRENT 4. DIAG_E_PWM_FEEDBACK	1. PWM start-up test detected a short circuit 2. PWM start-up test detected an open load. 3. PWM current check detected an overcurrent. 4. PWM pulse or period feedback out of range.

Table 11. *Input and Output Errors from PIN*

INPUT ERRORS	Error Codes	Descriptions
PIN 103 PIN 127	1. IO_E_INVALID_CHANNEL_ID 2. IO_E_CH_CAPABILITY 3. IO_E_STARTUP 4. IO_E_REFERENCE 5. IO_E_DRIVER_NOT_INITIALIZED *There are many other input errors some important are listed above	1. The given channel id does not exist 2. The given channel is not an ADC channel 3. The input is in the startup phase 4. A reference voltage (sensor supply or internal reference) is out of range. 5. The common driver init function has not been called before.

OUTPUT ERRORS	Error Codes	Descriptions
PIN 177 PIN 165	1. IO_E_DRIVER_NOT_INITIALIZED 2. IO_E_STARTUP 3. IO_E_SHORT_GND 4. IO_E_OPEN_LOAD_OR_SHORT_BAT 5. IO_E_SAFE_STATE 6. IO_E_INTERNAL_CSM *There are many other Output errors some important are listed above	1. The common driver init function has not been called before 2. The output is in the startup phase 3. A short circuit has been detected 4. Open load or short circuit to battery has been detected 5. The PWM channel is in a safe state 6. An internal error occurred

The WatchDog errors are checked by the callback application but the errors from the PINs both input and outputs are checked by the IO_ADC_Get function and IO_PWM_SetDuty functions for input and output PINs respectively. The return values of these functions are mentioned in Table 11 above. The code was implemented in such a manner if there were no errors detected during runtime then the function gives the desired result else the voltage output for the PIN or the set duty for the PWM output is set as zero. An example of the implementation of both the input channels are shown in the code below where ADC1 and ADC2 has return types for error codes shown in Table 11 from the input channels.

```
if (ADC1 == IO_E_OK && ADC2 == IO_E_OK){
    u16Volt_master = (ubyte2)Input_master_voltage;
    u16Volt_slave = (ubyte2)Input_slave_voltage;
}
else{
    u16Volt_master = 0;
    u16Volt_slave = 0;
}
```

```
/* ================================================================================================ */
/*    Function Name:          APPL_ErrorCb                                                          */
/* ================================================================================================ */

static ubyte2 APPL_ErrorCb(ubyte1 diag_state,ubyte1 watchdog_state,DIAG_ERRORCODE * const error){

    ubyte2 action;
    switch (error->device_num)
    {
        case IO_PIN_103:
            if ((error->error_code == DIAG_E_ADC_RANGE) || (error->error_code == DIAG_E_ADC_3MODE_SWITCH_TEST))
            {
                action = DIAG_ERR_DISABLE_SSW0;
            }
            else
                action = DIAG_ERR_NOACTION;
            break;
        case IO_PIN_127:
            if ((error->error_code == DIAG_E_ADC_RANGE) || (error->error_code == DIAG_E_ADC_3MODE_SWITCH_TEST))
            {
                action = DIAG_ERR_DISABLE_SSW0;
            }
            else
                action = DIAG_ERR_NOACTION;
            break;
        case IO_PIN_177:
            if ((error->error_code == DIAG_E_PWM_SHORT_CIRCUIT)|| (error->error_code == DIAG_E_PWM_OPEN_LOAD)
                    ||(error->error_code == DIAG_E_PWM_CURRENT)|| (error->error_code == DIAG_E_PWM_FEEDBACK))
            {
                action = DIAG_ERR_DISABLE_SSW0;
            }
            else
                action = DIAG_ERR_NOACTION;
            break;
        case IO_PIN_165:
            if ((error->error_code == DIAG_E_PWM_SHORT_CIRCUIT)|| (error->error_code == DIAG_E_PWM_OPEN_LOAD)
                    ||(error->error_code == DIAG_E_PWM_CURRENT)|| (error->error_code == DIAG_E_PWM_FEEDBACK))
            {
                action = DIAG_ERR_DISABLE_SSW0;
            }
            else
                action = DIAG_ERR_NOACTION;
            break;
        default:
            action = DIAG_ERR_SAFESTATE;
            break;
    }
    return action;
}
```

***Figure 25**. Error Callback function*

```
/* ============================================================================================== */
/*    Function Name:          APPL_NotifyCb                                                       */
/* ============================================================================================== */

static void APPL_NotifyCb(ubyte1 diag_state,
                          ubyte1 watchdog_state,
                          DIAG_ERRORCODE * const error)
{
    /* no decision can be done here */
//    (void)diag_state;
//    (void)watchdog_state;
//    (void)error;
    can_frame6.data[0]= diag_state;
    can_frame6.data[1]= watchdog_state;
    can_frame6.data[2]= error->device_num;
    can_frame6.data[3]= error->error_code;
    can_frame6.data[4]= (ubyte1)(error->faulty_value >>24);
    can_frame6.data[5]= (ubyte1)(error->faulty_value >>16);
    can_frame6.data[6]= (ubyte1)(error->faulty_value >>8);
    can_frame6.data[7]= (ubyte1)(error->faulty_value);
}
```

***Figure 26**. Notify Callback function*

Testing is done to validate the results during the software development lifecycle according to V-model Methodology. Testing verifies that the system which is developed meets the functional and non-functional requirements according to user needs. It also validates that the system is implemented properly. Test cases are designed to test the safety functions. Before designing a test case for a software, use cases are structured so that it provides information about a task or tasks expected to be performed by the software that is defined in safety and non-safety requirements.

Table 12 shows the list of test cases used during the development of the safety function.

***Table 12**. TEST CASES*

Test Cases	Description	Tested
TS1_1 WATCHDOG ERRORS	DIAG_E_ADC_RANGE DIAG_E_ADC_3MODE_SWITCH_TEST Checked the two Analog input errors from Pin 103 and 127	04.03.2020
TS1_2	Analog Input PIN errors	11.03.2020
TS2_1 WATCHDOG ERRORS	DIAG_E_PWM_SHORT_CIRCUIT DIAG_E_PWM_OPEN_LOAD DIAG_E_PWM_CURRENT DIAG_E_PWM_FEEDBACK Check four different high side PWM error from Pin 165 and 177	04.03.2020
TS2_2	PWM Output PIN errors	11.03.2020
TS3	Ramp function tested	28.02.2020
TS4	Upper and lower limits of Master voltage is tested	20.02.2020

TS5	Deviation tested between slave and master voltage	24.02.2020
TS6	PI output for both the proportional valves	20.03.2020
TS7	Scaling function	28.02.2020

Where,
TS Test Case

Mapping is done between Requirements, use, and test cases. The purpose of mapping is the traceability of the implementation phase. The software is developed according to the requirements and to fulfill the requirements, use and test cases were designed and with the help of mapping the developer can monitor and verify that all the requirements are implemented and tested.

Table 13*. Mapping of Requirements, Use and Test Cases in Generic Implementation*

OVERALL REQUIREMENT	USE CASES	TEST CASES
R1	SR1, SR2	TS1_1, TS1_2, TS2_1, TS2_2
R2	SR3	TS3
R3	SR4, SR5	TS4, TS5, TS7
R4	SR6	TS6

Software testing is done in two segments, one with Unity and the other by real-time testing and obtaining results through PCAN-view software. The functions which are tested are elaborated with the Test codes. ADC_Status function tests the limit of the raw voltage coming from the joystick (master and slave) and sets a limit of low and high boundaries as shown in the code below. Most importantly it checks the deviation between the slave and master voltage and if the deviation crosses the permissible limit it sends a MAX_VALUE to the can message indicating that the deviation is more than expected. Followed by a ramp function, the purpose of which limit sudden increase or decrease of output voltage, the scaled function is implemented to scale the output voltage as well as the request ramp output. Functions were tested using boundary conditions as well as using some random values. Boundary tests are performed directly on, above or below the inputs and outputs respectively Figure 27, Figure 28 and Figure 29 below shows test assertions of the test using unity testing platform.

```
// Testing max (4500mV) , min(500mV) and in between values of master voltage scaled to -1000--1000
void test_scaledMaster(void){
        TEST_ASSERT_EQUAL_INT16(-1000, scaled_master(500));
        TEST_ASSERT_EQUAL_INT16(-900, scaled_master(700));
        TEST_ASSERT_EQUAL_INT16(0, scaled_master(2500));
        TEST_ASSERT_EQUAL_INT16(899, scaled_master(4299));
        TEST_ASSERT_EQUAL_INT16(900, scaled_master(4300));
        TEST_ASSERT_EQUAL_INT16(1000, scaled_master(4500));
}
// Testing max curr, min curr and between value scaled between 300..800mA
void test_requestedCurrent(void){
        TEST_ASSERT_EQUAL_INT16(300, request_current(0));
        TEST_ASSERT_EQUAL_INT16(550, request_current(500));
        TEST_ASSERT_EQUAL_INT16(800, request_current(1000));
}
```

***Figure 27**.Testing of Scaled function*

```
 void test_rampOutput(void){
     // initial state ramp memory = 0
         TEST_ASSERT_EQUAL_INT16(1, ramp_output(1000));
         TEST_ASSERT_EQUAL_INT16(2, ramp_output(1000));
         TEST_ASSERT_EQUAL_INT16(1, ramp_output(-1000));
         TEST_ASSERT_EQUAL_INT16(0, ramp_output(-1000));
         TEST_ASSERT_EQUAL_INT16(-1, ramp_output(-1000));

 }
```

***Figure 28**.Testing of Ramp function*

```
// Since the allowance is between 4500 to 4700 mV. Checking the allowance limits
void test_joystick_tooHighDeviation(void) // check boundary within limit 100
{
	//TEST_ASSERT_EQUAL_UINT16(UBYTE2_MAX_VALUE, ADC_Status(700,4200));
	TEST_ASSERT_EQUAL_UINT16(650, ADC_Status(650,4375));
	TEST_ASSERT_EQUAL_UINT16(750, ADC_Status(750,4225));
	TEST_ASSERT_EQUAL_UINT16(UBYTE2_MAX_VALUE, ADC_Status(500, 3000));
	TEST_ASSERT_EQUAL_UINT16(UBYTE2_MAX_VALUE, ADC_Status(2500, 350));
	TEST_ASSERT_EQUAL_UINT16(UBYTE2_MAX_VALUE, ADC_Status(350, 2500));
	TEST_ASSERT_EQUAL_UINT16(4500, ADC_Status(4501, 499));
}

void test_joystick_validInput(void)
{	// Random inputs within the range
	TEST_ASSERT_EQUAL_UINT16(2500, ADC_Status(2500,2500));
	TEST_ASSERT_EQUAL_UINT16(4500, ADC_Status(4500,500));
	TEST_ASSERT_EQUAL_UINT16(500, ADC_Status(500,4500));
	TEST_ASSERT_EQUAL_UINT16(600, ADC_Status(600,4400));
	TEST_ASSERT_EQUAL_UINT16(610, ADC_Status(610,4420));
	TEST_ASSERT_EQUAL_UINT16(4300, ADC_Status(4300,700));
	TEST_ASSERT_EQUAL_UINT16(4299, ADC_Status(4299,699));
}
//From the Caldaro Datasheet the deadzone is 50% of the max voltage,
// which is 2500mV + or-200mV as a allowance so the range is between 2300 to 2700mV
void test_joystick_deadband(void)
{	// Deadband is between 2300 to 2700mV checking the boundary condition within the deadband
	TEST_ASSERT_EQUAL_UINT16(2500, ADC_Status(2599, 2401));
	TEST_ASSERT_EQUAL_UINT16(2500, ADC_Status(2600, 2400));
	TEST_ASSERT_EQUAL_UINT16(2500, ADC_Status(2401, 2599));
	TEST_ASSERT_EQUAL_UINT16(2500, ADC_Status(2400, 2600));
	TEST_ASSERT_EQUAL_UINT16(2299, ADC_Status(2299, 2701));
	TEST_ASSERT_EQUAL_UINT16(2500, ADC_Status(2699, 2301));
}

// testing invalid values, exact LL/HH boundary values, valid boundary values,
//exact L/H boundary values, valid values
// test deadband

// From the Caldaro Datasheet the maxlimit is (90+4)% or (90-4)%,
//the maximum voltage is 5000mV so 90% of it is 4500mV.
//Taking an allowance of 4% is 180mV. So for the test convience it is rounded to 200mV.
//So themaximum limit before it shows error is (4500+200=4700mV)
void test_joystick_highHighValues(void)
{	// Checking boundary condition since the maximum limit is 4700 and the lowest is 300
	TEST_ASSERT_EQUAL_UINT16(500, ADC_Status(301, 4699));
	TEST_ASSERT_EQUAL_UINT16(UBYTE2_MAX_VALUE, ADC_Status(299, 4701));
	TEST_ASSERT_EQUAL_UINT16(500, ADC_Status(300, 4700));
	// Checking boundary condition since the maximum limit is 4700 and the lowest is 300
	TEST_ASSERT_EQUAL_UINT16(4500, ADC_Status(4699, 301));
	TEST_ASSERT_EQUAL_UINT16(UBYTE2_MAX_VALUE, ADC_Status(4701, 299));
	TEST_ASSERT_EQUAL_UINT16(4500, ADC_Status(4700, 300));
}

void test_joystick_highValues(void)
{	// Checking boundary condition since the maximum limit is 4700 and the lowest is 300
	TEST_ASSERT_EQUAL_UINT16(4500, ADC_Status(4699, 301));
	TEST_ASSERT_EQUAL_UINT16(UBYTE2_MAX_VALUE, ADC_Status(4701, 299));
	TEST_ASSERT_EQUAL_UINT16(4500, ADC_Status(4700, 300));
}

void test_joystick_lowValues(void)
{
	TEST_ASSERT_EQUAL_UINT16(500, ADC_Status(301, 4699));
}

void test_joystick_lowLowValues(void)
{	// Checking with low values greater than the min set limit
	TEST_ASSERT_EQUAL_UINT16(UBYTE2_MAX_VALUE, ADC_Status(299, 500));
	TEST_ASSERT_EQUAL_UINT16(UBYTE2_MAX_VALUE, ADC_Status(700, 299));
}
```

***Figure 29**.ADC status test*

Testing of PI controller is done through PCAN-view by simulation since it cannot be tested in offline mode. Since the maximum output current from the PWM was set to 800mA, so the objective was to test the set duty at approximately maximum current output and a valid current between maximum and minimum.

1. When the current output is 795 mA, HEX valve 5534 corresponding to decimal number is 21812. So, the Set duty is 21812 when the current is 795mA.

2. When the current output is 405 mA, HEX valve 2E65 corresponding to decimal number is 11877. So, the Set duty is 14687 when the current is 405 mA.

Error information is also viewed from PCAN-view, if any error occurs which are listed in Table 10 and Table 11 during the runtime or during startup it can be monitored through PCAN-view.

6.4.2 MATCH DOCUMENTATION AND CODING

Project Definition Tool is one of the MATCH tools which provides the foundation of documentation needed to implement a software function. All the information involved in a project is automatically generated by PDT after building the project. Documentation includes system specifications, use and test cases, input and output pin information, CAN messages, Error handling information. System specifications include an overall requirement, use, and test cases, and CAN messages are the documentation that the developer inputs manually inside a PDT project and a documentation in a pdf format is generated automatically. Errors associated with the input and outputs pins are automatically generated in a documentation. Overall functional requirements provide necessary that needs to be implemented in software. The developer needs to input the description and Vehicle Functionality manually in a PDT project, the Safety Level, Safety Function is already present as a pre-defined list and the developer needs to select the appropriate Safety Level and function according to requirements. Table 14, Table 15, and Table 16 below show the functional safety requirements:

Requirement 1: Requirement Safe Stop when some input or output error occurs in the safety function

***Table 14**. Requirement for Safe Stop*

ID	SR-S2
Version	1.0
Description	Safe Stop when some input or output error occures in the safety function
Priority	1
Safety Level	PLc
Safety Function	Safe Stop 1
Vehicle Functionality	If there is any error inside the safety function the ouput, which in this case is a Proportional valve is zero.
Safe State	-
Failure Reaction Time	50 ms
Author	PGA
Functional type of Requirement	Functional
Safety type of Requirement	Safety

Requirement 2: Requirement Safe acceleration or deceleration

***Table 15.** Requirement for Safe acceleration or deceleration*

ID	SR-S3
Version	1.0
Description	Safe accelaration or deccelaration when the operator uses maximum speed or tries to reduce it
Priority	1
Safety Level	PLc
Safety Function	Safely Limited Accelleration
Vehicle Functionality	To Vehicle moves with a constant accelaration or deccelaration
Safe State	-
Failure Reaction Time	50 ms
Author	PGA
Functional type of Requirement	Functional
Safety type of Requirement	Safety

Requirement 3: Requirement The boom cylinder should move in the desired direction according to the operator.

***Table 16**. Requirement for Safe Direction*

ID	SR-S5
Version	1.0
Description	The boom cylinder should move in the desired direction according to the operator
Priority	1
Safety Level	PLc
Safety Function	Safe Direction
Vehicle Functionality	With the postive values of the joystick output the boom cylinder expands and with negative values it extracts
Safe State	-
Failure Reaction Time	50 ms
Author	-
Functional type of Requirement	Functional
Safety type of Requirement	Safety

The input and output pins used during software development is shown in Table 17 below.

***Table 17**. Input and Output Pins*

540 Inputs and Outputs

Pin	*Electrical Symbol*	*Description*	*Pin Configuration*	*Block Type*	*Block Name*
Analog Input 3 Mode (VIN:5V / CIN:25mA / RES:100k)					
103	103	PIN_103	PINTYP_VIN	InVoltDbl	Joystick
104	104	PIN_104	PINTYP_NA	-	-
105	105	PIN_105	PINTYP_NA	-	-
106	106	PIN_106	PINTYP_NA	-	-
127	127	PIN_127	PINTYP_VIN	InVoltDbl	Joystick
Low-Side Digital Outputs (DOU_C / VIN / DIN)					
251	251	PIN_251	PINTYP_DOU	DAC	DAC_02
241	241	PIN_241	PINTYP_DOU	DAC	DAC_01
High-Side PWM Outputs (PWM_C / DOU_C / DIN)					
153	153	PIN_153	PINTYP_NA	-	-
177	177	PIN_177	PINTYP_PWM	PRO	PRO_01
156	156	PIN_156	PINTYP_PWM	-	-
180	180	PIN_180	PINTYP_NA	-	-
159	159	PIN_159	PINTYP_NA	-	-
183	183	PIN_183	PINTYP_NA	-	-
186	186	PIN_186	PINTYP_NA	-	-
162	162	PIN_162	PINTYP_NA	-	-
189	189	PIN_189	PINTYP_PWM	-	-
165	165	PIN_165	PINTYP_PWM	PRO	PRO_02

List of all the I/O pins are well defined inside PDT, a developer needs to just attach the input and output pins which are used in a project. After the I/O pins are defined, CAN messages are written under Vehicle communication in PDT. Figure 30 below illustrates an example of a CAN message for pulse width modulation output current. Since all the CAN messages are declared similarly so documenting all in the thesis would be unnecessary.

CAN Messages for Bus ServiceBus

PropValveCurr

DLC (Bytes)	*Full Identifier (hex)*	*Cycle Time [ms]*	*PGN*	*TX*	*Byte Order*
8	18FF0000	50	65280 (0xFF00)	540	Intel

Byte	*Start Bit rel(absol.)*	*Length*	*Parameter*	*Scale*	*Unit*	*Offset*	*Min Value*	*Max Value*
1	1 (1)	16 bits	ControlCommand1	1	[-]	0	0	65535
3	1 (17)	16 bits	ActualCurrent1	1	[-]	0	0	65535
5	1 (33)	16 bits	ControlCommand2	1	[-]	0	0	65535
7	1 (49)	16 bits	ActualCurrent2	1	[-]	0	0	65535

Byte	*Parameter*	*Description*
1	ControlCommand1	Output Control Command for PWM 1
3	ActualCurrent1	Output Actual measured values for PWM 1
5	ControlCommand2	Output Control Command for PWM 2
7	ActualCurrent2	Output Current command for PWM 2

Figure 30.*CAN message initialization in PDT*

Errors can occur due to faults in a system, so it is very important to identify errors and minimize as much as possible to reduce risks and hazards. Errors can occur from different sources such as input and output pins errors, error reactions, and safety functionalities when they are activated are called Safety Integral Level (SIL) errors and CAN message errors. Errors are automatically defined after the hardware connection in a printed control board, functional blocks and when CAN messages are added into the Project definition tool. The functional blocks are PLd certified. The functional blocks which are used in the project are input voltage double block and proportional blocks. Errors associated with the software development are listed in Table 18, Table 19, and Table 20.

***Table 18**. SIL Errors*

DTC	*Description*	*Prio*	*Failure Reaction*	*Debounce Set Error*	*Debounce Release*
1012:31:255	Application execution time reached task time limit	Yellow	RM00_ProValve	0 ms	0 ms
1013:31:255	Battery voltage fell below lower threshold	Yellow	RM00_ProValve	500 ms	1000 ms
1014:31:255	Batttery voltage exceeds upper threshold	Yellow	RM_NONE	500 ms	1000 ms
1015:31:255	Temperature at lower threshold	Yellow	RM_NONE	500 ms	1000 ms
1016:31:255	Temperature at upper threshold	Yellow	RM_NONE	500 ms	1000 ms
1017:31:255	Sensor Supply S1 Low	Yellow	RM_NONE	500 ms	1000 ms
1018:31:255	Sensor Supply S1 High	Yellow	RM_NONE	500 ms	1000 ms
1019:31:255	Sensor Supply S2 Low	Yellow	RM_NONE	500 ms	1000 ms
1020:31:255	Sensor Supply S2 High	Yellow	RM_NONE	500 ms	1000 ms
1021:31:255	Sensor Supply 5V Low	Yellow	RM_NONE	500 ms	1000 ms
1022:31:255	Sensor Supply 5V High	Yellow	RM_NONE	500 ms	1000 ms
1023:31:255	Primary fault page incorrect - second fault page loaded correctly	Yellow	RM_NONE	0 ms	0 ms
1024:31:255	List load defect	Yellow	RM_NONE	0 ms	0 ms
1025:31:255	List store defect	Yellow	RM_NONE	0 ms	0 ms
1026:31:255	DM_LIST_OVERFLOW	Yellow	RM_NONE	0 ms	0 ms

***Table 19**. Input Errors*

DTC	*Description*	*Prio*	*Failure Reaction*	*Debounce Set Error*	*Debounce Release*
9000:3:255	Master input signal short to power	Yellow	RM_NONE	500 ms	1000 ms
9001:4:255	Master input signal short to ground	Yellow	RM_NONE	500 ms	1000 ms
9002:4:255	Slave input signal short to power	Yellow	RM_NONE	500 ms	1000 ms
9003:3:255	Slave input signal short to ground	Yellow	RM_NONE	500 ms	1000 ms
9004:26:255	Deviation of signals out of limit	Yellow	RM_NONE	500 ms	1000 ms
9005:14:255	Limp mode active	Yellow	RM_NONE	500 ms	1000 ms
9006:24:255	Parameter of input char NOT monoton	Yellow	RM_NONE	500 ms	1000 ms
9007:12:255	Unknown internal error	Yellow	RM_NONE	500 ms	1000 ms

***Table 20**. Output Errors*

DTC	*Description*	*Prio*	*Failure Reaction*	*Debounce Set Error*	*Debounce Release*
9008:5:255	Open load / open circuit	Yellow	RM00_ProValve	500 ms	1000 ms
9009:5:255	Short circuit to power supply / battery	Yellow	RM00_ProValve	500 ms	1000 ms
9010:6:255	Short circuit to ground	Yellow	RM00_ProValve	500 ms	1000 ms
9011:26:255	Deviation of current control	White	RM_NONE	500 ms	1000 ms
9012:24:255	Internal error (software or hardware error)	Yellow	RM00_ProValve	500 ms	1000 ms
9018:5:255	Open load / open circuit	Yellow	RM00_ProValve	500 ms	1000 ms
9019:5:255	Short circuit to power supply / battery	Yellow	RM00_ProValve	500 ms	1000 ms
9020:6:255	Short circuit to ground	Yellow	RM00_ProValve	500 ms	1000 ms
9021:26:255	Deviation of current control	White	RM_NONE	500 ms	1000 ms
9022:24:255	Internal error (software or hardware error)	Yellow	RM00_ProValve	500 ms	1000 ms

Use and test cases play a very important role in software development especially when developing safety-critical software. Use cases provide necessary information about the system specification and important functions that need to be implemented in software. So, coding of a software function is done based on the use cases defined. Whereas test cases validate whether the use cases defined are implemented properly during the software development. Each use case can be linked with one or more test cases as shown in Table 21, Table 22 and Table 23.

***Table 21**. Use case Id SR-S2_3*

ID	SR-S2_3
Requirement	SR-S2
Version	0.1
Description	If any error occures from the joystick like Short to Ground/Power, Deviation the valve is closed
Safety Level	PLc
AuthorGuid	PGA
Phase	-
Test Cases	SR-S2_3_3, SR-S2_3_6, SR-S2_3_7, SR-S2_3_8, SR-S2_3_9, SR-S2_3_10

***Table 22**. Use case Id SR-S3_5*

ID	SR-S3_5
Requirement	SR-S3
Version	0.1
Description	There will be gradual increase or descrease of accelaration by implementing RAMP function
Safety Level	PLc
AuthorGuid	PGA
Phase	-
Test Cases	SR-S3_5_5

***Table 23**. Use case Id SR-S5_4*

ID	SR-S5_4
Requirement	SR-S5
Version	0.1
Description	If any error occures from the Proportional valve like Short to Ground/Power, Open load the valve is closed
Safety Level	PLc
AuthorGuid	PGA
Phase	-
Test Cases	SR-S5_4_4, SR-S5_4_11, SR-S5_4_12, SR-S5_4_13

After all the necessary information is added in a PDT project, then an Auto-code builder (ACB) is generated after the PDT project is built. The ACB generates the C code for the library block usage and set the default values for the blocks. To implement the requirements coding is done inside the auto-code builder. CAN messages are written in the code that is defined in the project definition tool, mainly for testing and debugging

purpose from PCAN-view. Figure 31 illustrates the implementation of CAN messages inside an ACB.

```
// debug messages (for checking if code works correctly)
gCsr_tJoystickInpOut.u16RawInputMaster =      gInVoltDbl_tJoystick.tOut.u16RawValMaster;
gCsr_tJoystickInpOut.u16RawInputSlave =       gInVoltDbl_tJoystick.tOut.u16RawValSlave;
// Error status behavior
gCsr_tJoystickInpOut.u16ErrorStatus =         gInVoltDbl_tJoystick.tOut.tBehErrSta.u16ErrSta;
// Output scaled voltage , scaled from -1000... 1000
gCsr_tJoystickInpOut.i16scaled_sig =          gInVoltDbl_tJoystick.tOut.tOutVal.i16Sig;
gCsr_tPropValveCurr.u16ActualCurrent1=        gPro_tPRO_01.tOut.u16Ia;
gCsr_tPropValveCurr.u16ControlCommand1 =      gPro_tPRO_01.tOut.u16Ic;
gCsr_tPropValveCurr.u16ActualCurrent2=        gPro_tPRO_02.tOut.u16Ia;
gCsr_tPropValveCurr.u16ControlCommand2 =      gPro_tPRO_02.tOut.u16Ic;
// Error status behavior
gCsr_tErrorStatus.u16Error_state1 =           gPro_tPRO_01.tOut.tBehErrSta.u16ErrSta;
gCsr_tErrorStatus.u16Error_state2 =           gPro_tPRO_02.tOut.tBehErrSta.u16ErrSta;
gCsr_tJoystickStatus.bi3Joy_Dir_x =           gInVoltDbl_tJoystick.tOut.tOutVal.bi3Dir;
// Enumeration to define the Main phases of the Core application
gCsr_tJoystickStatus.i16ePhase=               g_ePhase;
```

***Figure 31**.Can messages in MATCH*

Now, a ramp function is implemented out of the scaled voltage from the joystick. Implementation of Ramp function is first done by creating a structure named *tRampAdr* inside a separate header file then a ramp function is created inside *eAppStartUp* function then initialization is done in *eAppInit* function and then the ramp function is called in a cyclic function *vAppRun*, as shown in Figure 32 below.

```
typedef struct {
    TSigRampAdr tRampAdr;
} TGlobalVariables;

ERetVal eAppStartUp( TVoid )
{
    ERetVal eRet;
    eRet = eSigRampCreate(&g_tGlobal.tRampAdr, "ramp");
    return R_OKAY;
}
/*Calculation / Execution with 2 time constants
 * Use if required 2 different delays:
 * one for increment in both (positive & negative) ranges [0 --> YMax] & [0 --> YMin]
 * another for decrement in both (positive & negative) ranges [YMax --> 0] &[YMin --> 0]
 */
Ramp_output =i16SigRampTS2(&g_tGlobal.tRampAdr,scaled_sig,100,100);
TInt16  scaled_sig;
scaled_sig = gInVoltDbl_tJoystick.tOut.tOutVal.i16Sig;
if ((scaled_sig == I16_ERROR) || (scaled_sig == I16_UNDEF))//Error or undefined values
{
    scaled_sig =0;
}
ERetVal eAppInit( TVoid )
{
    ERetVal eRet;
    eRet = eSigRampInit(&g_tGlobal.tRampAdr, 10, RAMPTYPE_LINEAR, 0, -1000, 1000 );
    return R_OKAY;
}
```

***Figure 32.**Ramp function in MATCH*

If there is an error or undefined values in the output scaled voltage, then the output of the joystick is set as zero. The figure below shows the implementation of the code.

To run a safety function, the output errors must be checked and restricted. In Table 20 it is seen that RM00_ProValve is added to the output errors. RM00_ProValve is a restriction mode which is a predefined machine reaction caused by an “active” error code. When the restriction mode is active then the valves are controlled automatically and the input to both the solenoids is set to zero, so that no current is passed through the solenoid. Figure 33 inside the code.

```
//VcoresetErrRelCond,boCoreGetErrRestMode
//functions passes  parameters from PDT (user defined error condition in RESTRICTION MODE)
 // Setting condition for REstriction Mode
if (boCoreGetErrRestMode(RM00_ProValve)== TRUE){
    //PROSTA_OFF controls the proportional valve automatically
    gPro_tPRO_01.tInp.eSta =PROSTA_OFF;
    gPro_tPRO_02.tInp.eSta =PROSTA_OFF;
    eBloDacSetVal(&gDac_tDAC_01,0);
    gPro_tPRO_01.tInp.u16Ic = 0;
    eBloDacSetVal(&gDac_tDAC_02,0);
    gPro_tPRO_02.tInp.u16Ic = 0;
 }
else{
    eBloDacSetVal(&gDac_tDAC_01,1);
    eBloDacSetVal(&gDac_tDAC_02,1);
}
```

***Figure 33**.Restriction Mode Implementation*

When the errors are checked then current is passed to both the solenoids but one at a time. That means when current is passed through one solenoid then the current in the other solenoid is zero. The scaled voltage which is ramped is further scaled between 300 to 800 mA using function *i16CalcLinear2Dots* as shown in Figure 34.

```
// FIRST SOLENOID
if (Ramp_output >0 && Ramp_output <=1000) {
    //function calculates the i32Cust value (x-value) over the i32Val value (y-value) in an linear function
    current_requested=  i16CalcLinear2Dots(Ramp_output, 0, 300, 1000, 800);
    u16_currentreqst = u16AbsI16(current_requested);
    gPro_tPRO_01.tInp.u16Ic = u16_currentreqst*10;
    gPro_tPRO_02.tInp.u16Ic =0;
    gPro_tPRO_01.tInp.eSta =PROSTA_ON;
}
//  SECOND SOLENOID
else if (Ramp_output >= -1000 && Ramp_output < 0){
    //function calculates the i32Cust value (x-value) over the i32Val value (y-value) in an linear function
    current_requested=  i16CalcLinear2Dots(Ramp_output, 0, 300, -1000, 800);
    u16_currentreqst = u16AbsI16(current_requested);
    gPro_tPRO_02.tInp.u16Ic = u16_currentreqst*10;
    gPro_tPRO_01.tInp.u16Ic = 0;
    gPro_tPRO_02.tInp.eSta =PROSTA_ON;
}
```

***Figure 34**.PWM task implementation in MATCH*

Testing of software functions is important as it checks if the functionalities are executed correctly. So, test cases are designed to meet the software requirements. are few test shown below Table 24,

Table 25 and Table 26.

***Table 24**. Test case Id SR-S2_3_3*

ID	SR-S2_3_3
Use Case	SR-S2_3
Version	0.1
Description	Master input signal short to power
Safety Level	PLc
AuthorGuid	PGA
Phase	-
Software Module	AppGlobal
Status	Pass
Constraints	-
Test Date	12/19/2019
Test Type	UnitTest
TesterGuid	RHA

***Table 25**. Test case Id SR-S2_3_9*

ID	SR-S2_3_9
Use Case	SR-S2_3
Version	0.1
Description	Deviation of signals out of limit
Safety Level	PLc
AuthorGuid	PGA
Phase	-
Software Module	AppGlobal
Status	Pass
Constraints	-
Test Date	12/19/2019
Test Type	UnitTest
TesterGuid	RHA

***Table 26**. Test case Id SR-S3_5_5*

ID	SR-S3_5_5
Use Case	SR-S3_5
Version	0.1
Description	Implementation of Ramp function for constant accelaration
Safety Level	PLc
AuthorGuid	PGA
Phase	-
Software Module	AppGlobal
Status	Pass
Constraints	-
Test Date	12/4/2019
Test Type	UnitTest
TesterGuid	PGA

Similarly, test cases based on master input signal short to ground, slave input signal short to power, slave input signal short to ground, open load / open circuit for PWM 1 and 2, unknown internal error, short circuit to a power supply/battery for PWM 1 and 2, short circuit to ground for PWM 1 and 2 and Internal error (software or hardware error) for PWM 1 and 2 are also tested. Manual testing is done by monitoring the CAN messages through PCAN-view and with the Machine service tool (MST). When a CAN message is viewed through a PCAN-view it shows all the important information (like input voltage, ramped output, Errors from WatchDog as well as pins etc.) that need to be monitored. If any unexpected error messages or unexpected values are displayed in the PCAN-view the developer can make changes in the code. In MST all the I/O values, an error message is displayed also. Some of the test cases are shown in Table 24, Table 25, and Table 26.

Mapping was done with the MATCH approach also with the purpose have traceability between the requirements, use, and test case like the Generic approach.

***Table 27**. Mapping of Requirement, Use and Test Cases in MATCH implementation*

OVERALL REQUIREMENTS	USE CASES	TEST CASES
SR-S2	SR-S2_3	SR-S2_3_3, SR-S2_3_6, SR-S2_3_7, SR-S2_3_8, SR-S2_3_9, SR-S2_3_10
SR-S3	SR-S3_5	SR-S3_5_5
SR-S5	SR-S5_4	SR-S5_4_4, SR-S5_4_11, SR-S5_4_12, SR-S5_4_13

Table 27 illustrates the mapping between Requirements, Use, and Test Cases. SR-S2 and SR-S5 are input and output errors respectively which are checked with the corresponding test cases and SR-S3 is the Ramp function for uniform acceleration of the machine.

7. RESULTS AND ANALYSIS

Results were analyzed based on a comparative analysis between MATCH and the Generic approach for developing a safety function. A comparative study consisted of Qualitative analysis and user studies. Efforts involved in acquiring required knowledge or skills, quality of documentation, and lines of codes required for implementing the safety-critical function in the thesis were considered in the Qualitative analysis method. User experience study with software developers involved in control system software development for both MATCH and Generic approach was performed.

7.1 Comparison of Implementing a safety function with both methods

Some of the key features for comparison were previous knowledge and expertise for implementing a safety function in a control system, complexity, and lines of codes involved and documentation.

- Previous knowledge and Expertise needed to implement a safety function

  MATCH provides a well-defined documentation for its safety features and all its tools like PDT and MST. The Application Program Interface (API) is well defined and easy to understand. **MATCH handles many features automatically** so it very easy for a developer to develop a safety function. **Deep understanding of the middleware is not necessary**, if a developer understands the block approach and has a basic knowledge in programming then developing a basic safety function is simpler, like which is implemented in the thesis. For example, considering double analog signal as an input voltage block and PWM as a proportional output block and with the input and output a simple safety function can be developed with less expertise compared to the Generic approach. The deviation from the input signal and errors associated with the blocks are automatically handled by MATCH. Further, considering the proportional output the implementation of a PI controller, the output current, set duty, as well as the errors associated with the proportional block, are automatically handled by the PDT in MATCH so it makes the implementation of any safety function with MATCH much easier for a developer. On the other hand, in the Generic approach, **a deep knowledge of the input and output blocks are needed**. All the features that are handled

automatically with MATCH are **implemented manually** like the errors associated with the input and output blocks, deviation from the analog signal, implementation of a PI controller, checking errors from the set duty of a PWM output, etc.

- Efforts involved in Documentation of a safety function

  Documentation is the most important part while implementing a safety function. With MATCH approach documentation is done with much ease as the developer can input the requirements, use and test cases, I/O pin information, functional blocks, and CAN messages inside a PDT project according to requirements needed to be performed by the safety function. In MATCH based implementation the developer needs to input a description of different requirements that are necessary for implementing the safety function, the safety function, and the safety is chosen from a pre-defined list in the requirements. A unique ID is generated randomly corresponding to every requirement. The developer needs to manually input the Use case description, title and then link it with the corresponding requirement ID which was created before. Then test cases are designed by linking the corresponding use cases. Suppose a use case consists of an input error check, then the test case will contain all the errors that can occur inside the input block. The errors in the input block already exist in PDT so the developer just needs to link it with the corresponding use cases. Similarly, in the case of CAN messages the developer needs to define the datatype of the CAN message from a pre-existing list of datatypes, length of the message, etc. PDT already contains all the information of the I/O pins available in the PCB so, linking of the I/O pins which are used in the system is only required. After building the project **documentation is automatically generated**. Auto-generated documentation in PDT is shown in Table 14, Table 15, Table 16, Table 17 etc. Whereas, in the Generic approach similar documentation is made manually as shown in Table 7, Table 8. Table 10, etc. Documentation in the GENERIC approach was made to provide only the necessary information needed to implement a safety function. The efforts involved to make the documentation for both the MATCH and Generic approaches were almost similar. But consider the detailed information of error message both from input, output pins, I/O pins, clear and well-documented requirements, use, and test case that were generated MATCH documentation was very organized. MATCH provides better readability because it describes each requirement, use, and test cases separately for a better understanding of the reader. Efforts needed to create requirements, use and test cases were nominal. So, MATCH is more

time-efficient and generates more detailed information of the system which same effort as in the Generic approach.

- Lines of code and Complexities for coding a safety function
  As stated, earlier MATCH handles many features automatically, so the actual coding is much less. While implementing the safety function approximately **85-90 lines of actual coding** is done if the comments and line gaps are ignored. On the other hand, with the Generic approach considering all the error handling, safety, and different other functions are implemented the complexity of the code increase, a lot more commenting is required to make it user readable. For the safety function to be implemented approximately **350 lines of code** were written in the comments and line gaps are ignored which makes it almost 4 times more coding than the MATCH approach. Further for programming a safety function in Ge
  neric approach Misra C standard is followed. Developer does not need to con centrate on any specific standard of coding in MATCH because the ACB which is generated from the PDT project already follows some safety coding guidelines. Table 28 elaborates in detail some of the major differences in coding which implementing a safety function with MATCH and Generic approaches

Table 28. *Comparison of Code complexities in both approaches*

Functionalities Implemented	**MATCH approach**	**Generic approach**
Analog Double Voltage Input	MATCH provides a pre-existing double input voltage block which includes all the errors associated with it. The developer just needs to add the limits of the input (for eg. in the thesis work the limits both upper and lower of the joystick) and the deviation from the joystick is handled automatically the developer just needs to add a percentage value for the deviation. When the ACB is generated the developer can call the block by	To implement a double input voltage signal the developer needed first initialize inputs with TTC I/O functions. Then a separate function named ADC_Status was developed which checks the limits of the input signals, checks the deviation from the input as shown in Figure 19. The deviation check was a challenging part since it needs few considerations such as

	just referencing the block name inside C file and implement the required logic.	the joystick tolerance limit and well as input error has to be considered. The lines of code to implement the function and initial the input signals were approximately 70 lines of C coding.
PI controller	Inside the Proportional output block, which is existing inside PDT already, the PI controller is included so the developer does not need to implement a PI controller separately.	In the Generic approach a PI controller is implemented where the Proportional and Integra, epsilon values are tuned to get the desired output value which is set duty value for the proportional valve.
Error Handling	Errors are defined in all the input and output blocks inside PDT. So, the developer does not need to define it separately. Error handling is done automated in MATCH. So, there is no chance of any error to be missed out from any I/O modules.	In Generic implementation which was done with TTC 500 controller, all the watchdog, as well as a pin, are implemented manually as shown in Table 19 and Table 20. In generic implementation there is a high probability of missing out possible errors during implementation.

7.2 User Experience Study

User studies are done by interviewing four programmers, two of them specialized in MATCH currently developing safety function in a control system and two of them specialized in a Generic way of implementing a function in a control system. During the interview, it was observed the engineers implementing a function with the Generic approach are not using safety function. Interviewee's identity is confidential and is not disclosed in the thesis work. A fixed set of questionnaires was asked to all the four programmers. The questions asked are:

1. What are the advantages and disadvantages of using this (MATCH/Generic) approach for implementing a safety/non-safety function in a control system?
2. Approximately what is the average time taken to complete an implementation of a safety function? Prerequisites: All the requirements from the customer is clear
3. Experience and coding skills required for implementing a basic safety function
4. What was the overall learning experience?
5. Overall experience for developing functions in a control system
6. What can be the possibilities are of improvements with the current approach?

Interviewees were categorized into participants 1,2,3 and 4. Where Participants 1 and 2 are specialized with MATCH approach and Participant 3 and 4 in the Generic approach.

***Table 29**. Participant 1 user study*

QUESTIONS	PARTICIPANT 1
1	The participant is from a mechanical background so for him it is easier to work with MATCH because with MATCH a deep understanding of the middleware is not needed as it handles many features automatically. Error management is one of the best features in the MATCH toolchain, it helps to achieve a desired performance level in the context of safety requirements in mobile machines. Since MATCH is quite a new tool, so there are areas of improvement. According to him, the Requirement management must be improved in the future
2	For him implementing one simple safety feature takes around 3 to 4 days. That includes a couple of input and output pins, least documentation, and basic testing of the code.
3	He mentioned according to him only Basic understanding of a programming language is enough for implementing a simple function
4	The learning process for him was quite smooth and easy, but he again mentioned it varies from person to person also. The MATCH application program interface is easy to understand, getting used to different MATCH tools and developing a simple function was not so difficult for him
5	6 months experience
6	Requirement management must be improved as it does not focus on a methodological approach of implementation for example V-model, and he mentioned that currently, work is going on to improve its features.

***Table 30**. Participant 2 user study*

QUESTIONS	PARTICIPANT 2

1	He mentioned that MATCH is specially designed for mobile applications, which means it focuses on the applications that are used in mobile application development designed for developing a machine. Unified Diagnostic Services (UDS), database arrangements, and Automatic service tools are the key advantages of using MATCH. According to him there is no such disadvantage but there are limitations such as it supports only specific hardware, currently working closely with TTControl. But different controllers can also be implemented with MATCH provided a separate development contract is made with the company.
2	For implementing a complete safety-critical function for the complete system with detailed documentation and testing an approximation of 3 months is required.
3	Basic coding skills are required such as knowledge of pointers, structures, certain data types, and cyclic function is required.
4	Learning a MATCH approach according to him is very simple if someone grabs the idea behind it. MATCH works behind the idea of block approach, how the input and output blocks work, what are cyclic calls etc. then working with MATCH environment is very simple
5	6 years
6	Usability of PC tools, since the MATCH approach is very solid according to him and C environment is also good, so it is seen among customers that they are happier if the requirement of coding is minimized. Import and export of data, copying projects, switching between hardware can be improved. Some of the improvements are currently on progress and some need to be improved.

***Table 31**. Participant 3 user study*

QUESTIONS	PARTICIPANT 3
1	The difficult part in the implementation according to him is to specify functions and writing specifications before the actual coding is implemented
2	If the requirements are mentioned, then coding does not take much time (non-safety coding), as the functions are not safety functions so only preliminary testing is done to verify if the code is running correctly or not mainly testing is done in a real environment.
3	Knowledge and training differ from functions to functions. If any function is associated with a mechanical movement, then proper experience and training are needed to implement a function. Good knowledge of coding is required
4	According to him the best learning can be achieved while working in a real environment with proper guidance and training.
5	15 years
6	Improvement regarding testing is needed, in the future detailed testing will be done

***Table 32**. Participant 4 user study*

QUESTIONS	PARTICIPANT 4
1	According to him when a new function is implemented it must not have a negative impact on the exiting functionalities, meaning the existing code should not break. Because the previous code is tested and verified. With the inclusion of new functions if the existing code produce error then the whole functionalities must be debugged which is a challenging part. The easiest part is the coding. While coding lot of functionalities is taken into consideration, so he splits the code into segments so adding or debugging becomes easier
2	Approximately 2 weeks to add minor features, basic documentation, and basic testing. Basic testing is done with some hardware components and a controller. After that the function is sent to the field to be tested.
3	According to him skills and knowledge increase with time and experience and depend upon a person. For starting basic knowledge is enough
4	In the beginning, it was quite tuff to understand the technicalities and with the coding as well but with experience and time everything was fine
5	5 years
6	According to him there was no specific mention of improvement in the current approach but yes, he too agreed that more detailed testing in the future is necessary.

7.3 Analysis of Results

Comparative parameters were used based on the implementation of a safety function between MATCH and Generic approaches. A user study was also conducted with experienced programmers involved in both methods of implementation. Analyzing the results from both comparative and user studies were done. Based on the analysis the most preferred approach for implementing the safety-critical function according to the author was concluded.

7.3.1 Analysis based on Implementation with two different approaches

Practical implementation took around **3 weeks to get the desired output in MATCH** whereas, in the **Generic approach it took around 11 weeks** to get the same output. Time measured includes the documentation and coding (which includes testing for both the approaches) only for implementing the safety function because system requirements were common for both the methods. MATCH generated a good quality of documentation in no time compared to the Generic approach. There were a lot of challenges while implementing the safety function in the Generic approach. Mainly challenges came from the coding part. MISRA C standard was followed while implementing the functions, un-

derstanding the standard, and figuring out the optimum way of implementing the functionalities took a lot of effort. Apart from that the deviation calculation encountered from the joystick took quite a bit of time and was challenging. The concept of implementing a PI controller function inside a proportional output block was a bit tricky. Moreover, from developing the functional modules, error detection methods also took a lot of time, especially to figure out the proper way of implementing error callback, notify callback functions, and testing errors from the input and output pins. Whereas in MATCH implementation functional blocks that are pre-certified are presented in PDT so, when an ACB was generated from the PDT project it already contained all the information of errors, and other functionalities only the logic needed to be implemented which was quite easy compared to the Generic approach. In terms of documentation, the PDT project also generates automatic documentation with a very detail description of all necessary parameters, if the same quality of documentation is made from the beginning, according to the author it takes approximately 3 times more effort compared to MATCH documentation.

7.3.2 Analysis based on User Study

After considering the answers from the interviewees in an unbiased way, it was concluded that there were advantages as well as disadvantages in both the approaches. The key **advantage of** using the MATCH approach is that it automatically handles some features so, it is not always needed for a developer to have a deep understanding of the middleware which makes tasks easier to implement. Further, it has some **key features like error management, UDS, and database arrangement** which makes the development of a safety function faster to implement. Whereas in the Generic approach there were no distinct features, but the working **environment is robust that implies importing and exporting functionalities** from one project to the other can be done easily, this approach is not restricted to a hardware manufacturer. Talking about the **disadvantages** which can better be termed as limitations also, since MATCH is comparatively a new mobile toolchain application where still lots of improvements can be made some of them are **Requirement management, import, and export of projects,** etc. and along with it one important limitation of MATCH is that it is compatible with TTControl hardware's only. But one interesting fact was discovered in the course of the interview was that, since MATCH is specially tailored for mobile application and its interface is not limited to any specific hardware application so if a separate development contract is made with the company then it is possible to use other manufacturers' ECU also. The **major disadvantage** is the **Generic approach** while interviewing the programmers was that there was a **lack of safety** in the implementation of a software function in a control system.

For that reason, the coding was comparatively easier to implement. Because to implement a safety function according to the author as described in Table 28 needs all of the considerations and involved more complexities. Basic documentation as well as basic testing of software function to check if the code is running correctly was conducted.
Moreover, while interviewing participants 1 and 2 it was observed that they both have different roles in the software implementation inside the company. Participant 1 is very new with the MATCH environment with only 6 months of experience so his role is to implement few features inside a safety function not developing an entire safety functionality for a system whereas, participant 2 is an experienced guy with 6 years of experience so he can develop a complete safety functionality of a system. So, while asked about the time taken to implement a safety functionality both answered according to their respective roles inside the company.

7.3.3 Conclusion of Results based on Analysis

After performing a Comparative analysis between MATCH and Generic approach for developing a software function, according to the author **MATCH is a much more preferred approach** than a Generic approach. The reason behind this is that the efforts that are needed to implement the safety function in MATCH is almost 3 times less compared to Generic approach. MATCH offers pre-certified functional blocks which makes error handling and error detection every simple. Along with it the functional blocks also contain some pre-existing features as described in Table 28. Coding in MATCH is very less so a developer with basic coding skills can implement a safety function with MATCH toolchain whereas, it is not possible with the later. Documentation with MATCH is very simple and more detailed with the minimum amount of inputs from the developer. Moreover, considering the safety side with MATCH implementation a minimum level of safety is always implemented because the functional blocks are PLd certified whereas, in the Generic approach the developer can opt for skipping the safety functionalities while developing software in the control system. From the usability study the author observed there was a clear absence of safety features while implementing a software so, according to the developers the coding was simple. Because safety functionalities are implemented in the Generic approach the coding very more complex compared to a non-safety function.

8. CONCLUSION AND FUTURE WORK

For implementing a safety-critical function in a control system ISO 13849-1:2015 standard guidelines were followed throughout the development phase. Misra C guidelines for coding the software were followed. Detailed description of system architecture, hardware components common for both MATCH and Generic approaches were shown for developing a safety function in a control system. Factors that influenced the safety standard (like $MTTF_{D,}$ DC, CCF) of hardware components selected for implementing the safety function were described. The calculation of different factors to get a desired Performance Level for the system was presented in chapter 6.3. Some important parts of the code (mainly implementing functional requirements) for developing the safety function was also presented. V-model methodology was followed in both the approaches for implementing the safety function. A comparative study was performed in chapter 7.1 and chapter 7.2 to find out the most preferred approach for implementing a safety function in a control system. Qualitative analysis and Usability study were done as part of a comparative study. In Qualitative analysis the two different approaches were analyzed based on previous knowledge, experience, and skills required, documentation quality, and lines of coding involved during the implementation of the safety function. Effort (as a measure of time taken) was used as a measuring parameter to compare both the approaches. Usability study was performed by interviewing programmers specialized in both MATCH and Generic approaches, respectively. The purpose of the study was to get an insight into knowledge regarding real-life working challenges, advantages, etc. while working MATCH or Generic methods. According to the author's findings from the implementation of a safety function as well as the information from the interviewees the future focus on improving the Requirement Management in PDT is important. Because for now it does not give a clear view of the methodology followed for implementation (like in the thesis V-model is followed).

Analyzing the results was conducted in an unbiased manner based on a comparative study from implementation as well as from the usability study. Based on the result analysis a conclusive opinion according to the author (who was the actor as well as the observer at the same time) was drawn. MATCH based approach is time-efficient for implementation, involved fewer complexities in coding example lines of code is less in MATCH approach and provides a more detailed documentation of software implementation in no time as compared to Generic approach.

8.1 Summary of Important points

***Table 33**. Summary of key findings*

KEY FINDINGS FROM THESIS WORK	
1	$MTTF_D$, DC, CCF are important factors according to ISO 13849-1:2015 that must be considered for developing a safety function. Along with that the Category of architecture used based on inputs, logic, and outputs for a system should also be considered. Based on these parameters the Performance Level of the system is determined.
2	Some specific coding guidelines must be strictly followed for developing any safety function in a control system. Misra C guidelines were followed for implementing safety function in Generic approach similar guidelines were also followed in the MATCH approach as well.
3	For developing a safety function detailed documentation is very important. Clear and elaborate information on requirements, Use and test cases, pin information, etc. Traceability and readability is very important throughout documentation. Generic documentation is not time efficient. MATCH documentation is time-efficient with less inputs needed manually from the developer
4	User study cannot give a concrete proof for preferring any approach. Interviewee's opinions may vary from each other although they are working with a similar approach. User studies mainly provides some insight knowledge of the approach based on a real-life working environment.
5	From the Comparative study, the author who was the developer and observer at the same time concluded that MATCH approach is a preferred approach for implementing a safety function in a control system

8.2 Future Work

Currently, up-gradation and improvement of MATCH features to make it more customer-friendly application is going on. One of the key areas to focus on future work could be to improve the TSE tool in MATCH with better documentation. Further, improvements in import and export of projects could be done easily to make it more user friendly. While in the Generic approach a serious concern was the lack of proper testing noticed by the author while interviewing experts working in the Generic method. A common tendency to avoid implementing safety functionalities was seen mainly due to its complexity in

terms of more detailed documentation, extra coding, etc. resulting in more time. The consequences of a non-safety function in a control system can be very severe.
So, the most optimum solution is when the time required to complete the development process is reduced without compromising the quality (in terms of safety standard) of the product. MATCH plays an important role in reducing the time taken (better time-efficient than Generic approach) for developing a safety function and testing yet maintaining the same quality of the product.

The thesis work helps to find some strong arguments for using MATCH as a software suite to develop a safety function in mobile machines. So, this information can be used to popularize the toolchain among industries and convenience the manufacturing mobile machine industries that it can be a beneficial tool for them. Since many companies are facing challenges to implement a safety function with Generic approach. In the future, hopefully MATCH software suite will become more popular among mobile machine industries, and the safety of machines will be given much more priority compared to what it is given currently.

REFERENCES

[1] https://www.br-automation.com/en/about-us/customer-magazine/2018/20189/safety-for-mobile-machines/ [BR Automation, access date 17.04.2020]

[2] Tiusanen, R., Hietikko, M., Alanen, J., Pátkai, N. and Venho, O., 2008. System safety concept for machinery systems. *VTT Tiedotteita-Research Notes*, *2437*.

[3] Chinniah, Y., 2015. Analysis and prevention of serious and fatal accidents related to moving parts of machinery. *Safety science*, *75*, pp.163–173.

[4] Functional Safety of Mobile Machines, 23.10.2018. Presenter: Elmar Wulf. Hydac Software GmbH (presentation slides on functional safety)

[5] Ruff, T., Coleman, P. and Martini, L., 2011. Machine-related injuries in the US mining industry and priorities for safety research. *International journal of injury control and safety promotion*, *18*(1), pp.11–20.

[6] Noll, J., DeGennaro, C., Carr, J., DuCarme, J. and Homce, G., 2017, November. Causal Factors of Collision Accidents Involving Underground Coal Mobile Equipment. In *ASME 2017 International Mechanical Engineering Congress and Exposition*. American Society of Mechanical Engineers Digital Collection.

[7] "12100: Safety of machinery–General principles for design–Risk assessment and risk reduction (ISO 12100: 2010)." (2010).

[8] Porras-Vázquez, Alberto, and Julio-Ariel Romero-Pérez. "A new methodology for facilitating the design of safety-related parts of control systems in machines according to ISO 13849: 2006 standard." *Reliability Engineering & System Safety* 174 (2018): 60–70.

[9] ISO 13849-1:2015, Safety of machinery — Safety-related parts of control systems — Part 1: General principles for design. ISO, EN.

[10] User Manual for Machine Application Tool Chain (MATCH), version 1.14, 15th November 2018.

[11] https://www.tutorialspoint.com/sdlc/sdlc_v_model.htm [tutorialspoint, date of access: 29.11.2019].

[12] Bukhari, S.A.H., 2011. What is Comparative Study. *Available at SSRN 1962328*.

[13] Albert, W., Tullis, T. and Tedesco, D., 2009. *Beyond the usability lab: Conducting large-scale online user experience studies*. Morgan Kaufmann.

[14] Hietikko, Marita, Timo Malm, and Jarmo Alanen. "Risk estimation studies in the context of a machine control function." *Reliability Engineering & System Safety* 96.7 (2011): 767–774.

[15] Bin Li and Tianjun Zhu.,2018." Advanced safety technologies and control systems for heavy commercial vehicles: a survey". *In book: Advances in Engineering Research. Edition: 1. Chapter: 2. Publisher: NOVA Science Publisher.*

[16] https://www.hse.gov.uk/toolbox/machinery/safety.htm# [Health and Safety Executive (HSE), date of access: 11.12.2019]

[17] *Naboulsi, M.A., Act-Ip, 2018. Safety control system for vehicles based on driver health. U.S. Patent 10,081,317.*

[18] Gabriška, D., 2016. Software requirements for the control systems according to the level of functional safety. *Journal of Applied Mathematics, Statistics and Informatics*, *12*(1), pp.25–32

[19] Zahálka, J., Tůma, J. and Bradáč, F., 2014. Determination and Improvement of Performance Level of Safety Function of Emergency Stop for Machinery. *Procedia Engineering*, *69*, pp.1242–1250.

[20] MISRA C:2012 Guidelines for the use of the C language in critical systems.

[21] User Manual for Project Definition Tool. For the PDT version 2.9, January 9, 2019

[22] User Manual for Machine Service Tool, version 1.07, December 21,2018.

[23] Test and Simulation Environment User Manual Version 1.3, August 28, 2019.

[24] Safety manual for MATCH Safety version 1.17, November 20, 2019.

[25] HY-TTC 540 System Manual Programmable ECU for Sensor Actuator Management, Version 01.04, release date 28th June 2017.

[26] Safety Manual for HY-TTC 500, version 1.10.0, Author- Oliver Praprotnik, date 9.5.2017

[27] https://www.peak-system.com/PCAN-View.242.0.html?&L=1# [PCAN-view, date of access 30.03.2020].

[28] http://www.throwtheswitch.org/unity [Unity, date of access: 20.03.2020].

[29] I/O Driver Manual for HY-TTC 500, product version 2.10.1, 29th June 2016.