# Multi-Domain Fault Simulation Using Virtual Prototypes

**Dissertation**
der Mathematisch-Naturwissenschaftlichen Fakultät
der Eberhard Karls Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt von
M.Sc. Raghavendra Koppak
aus Sri Ramnagar, Karnataka, India

Tübingen
2021

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Eberhard Karls Universität Tübingen.

| | |
|---|---|
| Tag der mündlichen Qualifikation: | 19.07.2022 |
| Dekan: | Prof. Dr. Thilo Stehle |
| 1. Berichterstatter: | Prof. Dr. Oliver Bringmann |
| 2. Berichterstatter: | Prof. Dr. Joachim Gerlach |

# DECLARATION

I hereby declare that I alone wrote the doctoral work submitted here under the title "Multi-Domain Fault Simulation Using Virtual Prototypes", that I only used the sources and materials cited in the work, and that all citations, whether word for word or paraphrased are given as such.

Raghavendra Koppak
Tübingen, 21.12.2021

“A goal is a dream with a deadline.”

- Napoleon hill

“Treat failure as a lesson on how not to approach achieving a goal, and then use that learning to improve your chances of success when you try again. Failure is only the end if you decide to stop.”

- Richard Branson

“Education is the most powerful weapon which you can use to change the world.”

- Nelson Mandela

# KURZFASSUNG

Industrielle elektronische Systeme, die die Fertigungsprozesse sowie die Bewegungsteuerung der Fertigungsmaschinen in modernen, hochautomatisierten Produktionsanlagen steuern, sind hochkomplex geworden und kombinieren eine Vielzahl von elektronischen, elektrischen und mechanischen Komponenten, die während ihres Betriebs reibungslos zusammenwirken müssen. Die hohen Anforderungen bei der Entwicklung zukünftiger Produktionsanlagen werden insbesondere an sogenannte Motion-Control-Systeme gestellt, die beispielsweise die schnelle und präzise Positionierung und Bewegungssteuerung von Förderbändern und Roboterarmen übernehmen. Diese Systeme müssen Störungen aller Art bewältigen und die Sicherheit des Bedienpersonals und die Integrität der Maschine jederzeit gewährleisten. Aber was passiert, wenn z.B. einzelne Chips in der Steuerung eines Roboters ausfallen, ein Motor aufgrund eines Lagerschadens überhitzt oder ein Sensor fehlerhafte Daten liefert? Ist es möglich, die Steuerungssoftware dieser Systeme so zu entwickeln und zu testen, dass bei Störungen an einer oder mehreren Komponenten das Gesamtsystem immer in einem sicheren Zustand verbleibt? Ist es ferner gewährleistet, dass Menschen in der Nähe dieser Maschinen nicht geschädigt und teure Komponenten, wie Motoren oder Roboterarme, nicht zerstört werden? Bis heute basieren die Endtests dieser Systeme hauptsächlich auf realen Prototypen, um den korrekten und sicheren Betrieb zu sicherzustellen.

In der konventionellen Systementwicklung sind daher reale Prototypen und umfangreiche Systemtests notwendig. Diese Prototypen sind jedoch erst in späteren Phasen des Entwurfsprozesses verfügbar. Diese Tests sind bereits heute aufwendig und teuer und sind nicht in der Lage alle möglichen Fehlerarten vollständig abdecken, da bestimmte Fehler in der realen Hardware nicht provoziert werden können. Außerdem kann die späte Ausführung der Tests zu langen Iterationsschleifen führen, falls bei den abschließenden Tests Schwachstellen festgestellt werden. Noch anspruchsvoller wird die Validierung für die hochflexiblen, selbstkonfigurierenden, selbstheilenden Steuerungssysteme zukünftiger Fertigungsszenarien mit Fokus auf die Automatisierung kleiner Losgrößen, die eine wesentlich höhere Konfigurierbarkeit der Systeme und eine engere Interaktion zwischen Mensch und Maschine erfordern.

Die Fehlerinjektion in Low-Level-Simulationsmodelle ist weit verbreitet, um einen sicheren Betrieb unter unerwarteten Bedingungen, wie z.B. Eigenausfällen der Elektronik

oder umweltbedingten Ausfällen, zu validieren. Für eine schnelle Simulation der Systeme ist jedoch eine hohe Abstraktion der Modelle erforderlich und die Fehlersimulation bietet einen Kompromiss zwischen Genauigkeit und Simulationsleistung. Virtuelle Prototypen (VPs) ermöglichen es, Systemtests in einem frühen Entwicklungsstadium durchzuführen. Aber sie leiden immer noch unter einer unzureichenden Werkzeugunterstützung und Methodik. VPs haben z.B. keine Simulator Unterstützung mit inhärenten Fehlerinjektionstechniken. Darüber hinaus nutzen Ingenieurgruppen aus verschiedenen Bereichen verschiedene Modellierungssprachen und -Werkzeuge, um Modelle zu entwickeln und ihre Entwürfe in der Industrie zu bewerten.

In dieser Arbeit wird eine kombinierte Technik zur Modellierung von Fehlern in Multi-Domain-Systemen vorgeschlagen, die eine frühzeitige Validierung dieser heterogenen Systeme ermöglicht. Die Validierung erfolgt mittels virtueller Stresstests durch Zusammenführung verschiedener Domänen in eine Simulationsplattform. Es werden geeignete Techniken beschrieben, um physikalische Komponentenmodelle so zu erweitern oder zu entwickeln, dass sie Fehlerinjektionsmöglichkeiten beinhalten, bevor sie in eine virtuelle Plattform importiert werden. Dabei werden nur physikalische Komponentenmodelle berücksichtigt, die mit MATLAB/Simulink modelliert werden, da nicht-digitale Teile des Systems in der Industrie häufig mit MATLAB/Simulink modelliert werden. In dieser Dissertation wird die im Rahmen des EffektiV-Projekts entwickelte TLM-basierte Fehlerinjektionsinfrastruktur genutzt, um durch die Entwicklung geeigneter Fehlermodelle Fehler in digitale Bauteile zu injizieren. Darüber hinaus wird ein generisches Fehlerinjektions-Framework vorgeschlagen und implementiert, um Fehlerinjektionsexperimente auf der Basis virtueller Prototypen durchzuführen. Mit diesem Framework werden die Fehler in die heterogenen Komponenten der industriellen Systeme (Motion Control Systeme) entweder interaktiv während der Simulationslaufzeit aktiviert oder als Regressionstests mit Hilfe von Skripten ausgeführt. Des Weiteren werden Nachbearbeitungstechniken vorgeschlagen, um die Ergebnisse der Regressionstests automatisch auszuwerten. Ferner werden die vorvalidierten Fehlermodelle in ein Hardware-in-the-Loop (HiL)Testsystem bei Siemens integriert, wodurch eine abschließende Validierung der Fehlersicherheit der Systeme vor der Produktfreigabe ermöglicht wird.

Die vorgeschlagenen Methoden werden am Beispiel einer industriellen Motorsteuerung mit zwei virtuellen HiL- und einem HiL-Demonstrator evaluiert: (i)

Industrielle Motorsteuerung mit Motorsignalaufbereitung mittels Beschleunigungssensor. (ii) Industrielle Motorsteuerung und Förderbandanwendung basierend auf dem V-REP (Virtual Robotic Experimentation Platform) Simulator. (iii) HIL-Testsystem basierend auf einem Siemens SINAMICS G120 Antrieb. Es konnte gezeigt werden, dass die Ergebnisse dieser Arbeit zu einer Anpassung der zukünftigen Produktentwicklung für industrielle elektronische Systeme bei der Siemens AG führen kann.

***Schlüsselwörter***: Industrie 4.0, Cyber Physical Systems, Virtueller Prototyp, Fehlersimulation, Hardware-in-the-Loop (HIL), Virtuelle Hardware-in-the-Loop (vHIL), Physische Komponentenmodelle, Fehlerübertragung, SystemC/TLM, Multi-Domain Simulation, MATLAB/Simulink, Simscape, Validierung.

# Abstract

Industrial electronic systems that control manufacturing and machine motion in modern, highly automated production facilities have become highly complex, which combine a variety of electronic, electrical, and mechanical components that need to interact smoothly during their operation. The high critical issues in the development of future production plants are imposed by motion control systems, which manage the fast and most accurate positioning and motion control of conveyor belts and robot arms, for instance. These systems are required to properly cope with failures of all kinds to guarantee safety of operators and machine integrity at any time. But what happens, if for instance single chips in a robot‘s control unit fail, if a motor due to a bearing damage overheats or a sensor delivers faulty data? Is it possible to develop and test the control software of these systems in a way that in case of faults in one or more components the total system always remains in a safe state? Is it guaranteed that humans near to these machines are not harmed and expensive parts like motors or robot arms are not destroyed? As of today, the final tests of those systems are mainly based on physical prototypes to ensure the correct and safe operation.

In conventional system development, physical prototypes and extensive system tests are needed. However, those prototypes are available only in later phases of the design process. These tests are complex and expensive already today and are not able to completely cover all possible kinds of failures, as certain failures cannot be provoked in real hardware. Moreover, the late execution of the tests may cause long iteration loops in case weaknesses are detected in the final tests. The validation becomes even more challenging for the highly flexible, self-configuring, self-healing control systems of future manufacturing scenarios with focus on automation of small lot sizes, which requires much higher configurability of the systems and closer interaction between humans and manufacturing machines.

Fault injection into low level simulation models is widely used for validating the satisfactory operation under unexpected conditions like intrinsic failures of electronics or failures caused by the environment. However, a high level of abstraction of models is required for fast simulation of the systems and fault simulation is a trade-off between accuracy and simulation performance. Virtual prototypes (VPs) provide a possibility to perform system tests in an early stage of development. But they still suffer from less tool support and methodology features e.g. VPs lack simulator support with inherent fault-injection features.

Additionally, engineering groups of different domains exploit different modeling languages and tools to develop models and evaluate their designs in industry.

In this work, combined techniques to model faults in multi-domain systems are proposed which make early validation of these heterogeneous systems possible. The validation is performed by conducting virtual stress tests after bringing together different domains into one simulation platform. Suitable techniques are described, to enhance or develop physical component models such that they incorporate fault injection possibilities before being imported into a virtual platform. Only physical components models that are modeled using MATLAB/Simulink are considered, as non-digital parts of the system are often modeled in industry using MATLAB/Simulink. The TLM-based fault injection infrastructure developed under the EffektiV project is used to inject faults in digital parts by developing appropriate fault models. Furthermore, a generic fault injection framework is proposed and implemented to conduct fault injection experiments based on virtual prototypes. Using this framework, the faults in heterogeneous parts of the industrial systems (motion control systems) are either activated interactively during simulation runtime or executed as regression tests using scripts. Also, the post processing techniques are proposed to automatically evaluate the results of regression tests. Later, these pre-validated fault models are integrated in a hardware-in-the-loop (HIL) test system at Siemens, which enables a final validation of the systems' safety against faults before product release.

The proposed methodologies are evaluated using an industrial motor control application example with two virtual HIL and one HIL demonstrators: (i) Industrial motor control system with motor signal conditioning using acceleration sensor. (ii) Industrial motor control system along with conveyor-belt application based on the V-REP (Virtual Robotic Experimentation Platform) simulator. (iii) HIL test system based on a Siemens SINAMICS G120 drive. It has been shown that the contributions from this study provided further directions in future product development for industrial electronic systems at Siemens AG.

***Keywords***: Industrial 4.0, Cyber Physical Systems, Virtual Prototype, Fault Simulation, Hardware-in-the-loop (HIL), Virtual Hardware-In-the-Loop (vHIL), Physical Component Models, Fault Transfer, SystemC/TLM, Multi-Domain Simulation, MATLAB/Simulink, Simscape, Validation.

# PREFACE

This dissertation has been submitted to the Eberhard-Karls-University of Tuebingen in partial fulfilment of the requirements for the doctoral degree at computer science department (Wilhelm-Schickard-Institute for Computer Science). The work has been carried out at the DF TI AT (Advanced Technologies) department, Siemens AG Nuremberg under the supervision of Prof. Dr. Oliver Bringmann, Prof. Dr. Joachim Gerlach (Department of Computer science, Albstadt-Sigmaringen University) and Dr. Andreas von Schwerin.

# ACKNOWLEDEGEMENT

I would like to take this opportunity to remember all those who gave me the possibility to develop, to learn, to know myself better and finally to make life a memorable and unforgettable experience during this PhD thesis and beyond.

I express my gratitude to my colleagues at Siemens AG in the Department of DI, AT, ATS who supported me in my research work and maintained a pleasant working atmosphere and my supervisor Prof. Dr. Oliver Bringmann for all the invaluable inputs, suggestions, and support throughout this work to make this a success.

I would like to specially thank Prof. Dr. Joachim Gerlach for taking time to review my work and supporting me in the dissertation formalities.

My most grateful thanks are reserved for my parents, Laxminarayana and Vimala Rani, for their continued love and support. Finally, a special to my loving wife, Soumya, for being my inspiration and motivation through what has been an arduous project of my life till now. I am forever grateful for my caring, patient, and supportive family.

TABLE OF CONTENTS

**DECLARATION** **III**

**KURZFASSUNG** **V**

**ABSTRACT** **1**

**PREFACE** **3**

**ACKNOWLEDEGEMENT** **4**

**LIST OF FIGURES** **10**

**1 INTRODUCTION** **1**

**1.1 Robust Motion Control Systems** **1**

1.1.1 Traditional Development 2

1.1.2 Design and Validation Challenges, Parallel Development 3

**1.1 Research Contributions** **6**

**1.2 Thesis Outline** **7**

**2 BACKGROUND** **9**

**2.1 Electronic System Level (ESL)** **9**

2.1.1 ESL Design Methodology 9

2.1.2 System Level Design using SystemC 10

2.1.3 Virtual Prototypes 11

**2.2 Digital Component Modeling** **14**

2.2.1 Introduction 14

2.2.2 Transaction Level Modeling (TLM) 14

**2.3 Analog and Mixed-Signal (AMS) Modeling** **19**

2.3.1 Introduction 19

2.3.2 Analog and Mixed-Signal Modeling Languages 19

**2.4 Model-Based Design and In-the-loop Technologies** **23**

2.4.1 Model-in-the-loop (MIL) 26
2.4.2 Software-in-the-loop (SIL) 26
2.4.3 Processor-in-the-loop (PIL) 27
2.4.4 Hardware-in-the-loop (HIL) 28
2.4.5 Virtual Hardware-in-the-loop (vHIL) 30

**2.5 Fault Injection and Simulation 31**

2.5.1 Fault Classification on Temporal Behaviour 31
2.5.2 Fault Injection Current Techniques 32

**2.6 Functional Safety Standards 34**

2.6.1 IEC61508 34
2.6.2 ISO 26262 35

**3 STATE-OF-THE-ART AND RESEARCH CONTEXT 37**

**3.1 Related work 37**
**3.2 Motion Control Systems (MCS) 40**
**3.3 Exemplary Faults in MCS 43**

3.3.1 Exemplary Faults in the Analog Hardware 43
3.3.2 Exemplary Faults in Mixed-Digital/Analog Hardware 44
3.3.3 Exemplary Faults in the Communication Bus 45
3.3.4 Exemplary Faults in the Software 46
3.3.5 Exemplary Faults in the Application 46
3.3.5 Hardware Induced Software Errors 47

**3.4 Fault Injection using Virtual Prototypes 48**
**3.5 Aim of the Research and its Objectives 48**
**3.6 Drawbacks of Existing Solutions and Research Methodology 49**

**4 MULTI-DOMAIN FAULT MODELING AND SIMULATION 50**

**4.1 Introduction 50**
**4.2 Virtual Stress Tests 52**

4.2.1 Fault Scenario Identification and Fault Specification 52

4.2.2 Identifying Critical System States 56

**4.3 Analog and Mixed-Signal Fault Modeling 57**

4.3.1 SystemC-AMS and MATLAB/Simulink 59
4.3.2 Abstract Fault Models and Fault Interface Design 60
4.3.3 Fault Simulation in Abstract Models 62

**4.4 Fault Transfer 63**

4.4.1 Motor Control Application 64
4.4.2 Examples of Fault Transfer 67

**4.5 Digital Fault Modeling based on TLM 95**

4.5.1 State-of-the-art 95
4.5.2 Fault Injection for TLM Abstraction 96
4.5.3 Fault Interface Design and Injector 98

**4.6 Multi-Domain Virtual Platform 101**

4.6.1 Preparing the Simulink Models for Fault Injection 101
4.6.2 Simulink Model Integration 102
4.6.3 Generic Fault Injector 105
4.6.4 Mapping Fault ID's 107

**4.7 Automated Fault Injection Tests and Results Evaluation 107**

4.7.1 Motivation 107
4.7.2 Automated Comparison of the Simulation Results of Fault Injection Tests 108
4.7.3 Post-Processing Scripting 109

**5 IMPLEMENTATION ASPECTS AND RESULTS 110**

**5.1 Industrial Motor Control Application 111**

5.1.1 Control Software 111
5.1.2 PWM Generator 112
5.1.3 Simscape Power electronics 113
5.1.4 Simulink Power electronics 114

5.1.5 Motor Module 117

**5.2 Simulation Performance Test 119**

**5.3 Automated Fault Injection Tests and Results Evaluation 121**

5.3.1 Automated Simulations using Tcl Scripts 121

5.3.2 Configuration Strategy 121

5.3.3 Simulation Control Scripting 122

5.3.4 Automated Fault Injection Tests 122

5.3.5 Example 123

5.3.6 Conclusion 126

**6 MOTION CONTROL SYSTEM: CASE STUDIES 127**

**6.1 Motor Condition Monitoring using Acceleration Sensor from RB 128**

6.1.1 Setup 128

6.1.2 Fault Injection GUI 129

6.1.3 Test Scenario's 130

6.1.4 Conclusion 133

**6.2 Conveyer-Belt Simulation using V-REP Simulator 134**

6.2.1 Setup 134

6.2.2 Test Scenario's 136

6.2.3 Conclusion 138

**6.3 Hardware-in-the-loop (HIL) 138**

6.3.1 Setup 139

6.3.2 Characteristics of the Setup 141

6.3.3 Fault Injection 142

6.3.4 Conclusion 145

**7 CONCLUSION AND OUTLOOK 146**

**7.1 Summary 146**

**7.2 Contributions 147**

7.2.1 Fault Injection in Heterogeneous Component Models 147
7.2.2 Fault Transfer 148
7.2.3 Automated Stress Tests and Post Processing 148
7.2.4 Case Studies of Motion Control System 149

**7.3 Outlook** **149**

7.3.1 ASAM XIL Standard for Fault Effect Simulation 149
7.3.2 Improved Workflow for Virtual Prototypes with Stress Tests 150

**REFERENCES** **151**

# LIST OF FIGURES


**Figure 1: Typical Motion Control System with Sensor, PLC and CU [3]** **2**

**Figure 2: Traditional Product Development Flow** **3**

**Figure 3: Hardware/Software Parallel Development** **4**

**Figure 4: SystemC language architecture [14]** **10**

**Figure 5: Simulation speed versus accuracy trade-off [13]** **15**

**Figure 6: Use cases and abstraction levels [13]** **16**

**Figure 7: TLM-2.0 communication [13]** **17**

**Figure 8: AMS Extensions for the SystemC Language Standard [9]** **20**

**Figure 9: The V-model of the Systems Engineering Process** **24**

**Figure 10: Typical control system** **25**

**Figure 11: Model-in-the-loop** **26**

**Figure 12: Software-in-the-loop** **27**

**Figure 13: Processor-in-the-loop** **28**

**Figure 14: Hardware-in-the-loop** **29**

**Figure 15: Virtual Hardware-in-the-loop** **30**

**Figure 16: Drive Model of a Motion Control System** **41**

**Figure 17: Power electronics of a power module** **42**

**Figure 18: Fault Scenario Template** **53**

**Figure 19: Fault Specification Template** **54**

**Figure 20: Fault Scenario Example** **54**

**Figure 21: Fault Specification Example** **55**

**Figure 22: Speed Profile of a Motor** **56**

**Figure 23: Fault models of capacitor and transistor** **59**

**Figure 24: Analog Fault Model (AFM)** **60**

**Figure 25: Control Interface and Fault Type** **62**

**Figure 26: Motor Control Application Setup** **65**

**Figure 27: Simulation Output (Simscape)** **66**

**Figure 28: Simulation Output (Simulink)** **66**

**Figure 29: Speed Comparison** **67**

**Figure 30: Fault Selection in Inverter and Motor** **67**

**Figure 31: Simulation Output (Transient State) – Simscape (F1)** **69**

**Figure 32: Motor Currents Zoomed (Transient State) – Simscape (F1)** **69**

**Figure 33: Phase-to-Phase Short-Circuit Fault Model without Protection – Simulink (F1)** **70**

**Figure 34: Simulation Output (Transient State) - Simulink (F1)** **71**

**Figure 35: Motor Currents Zoomed (Transient State) – Simulink (F1)** **71**
**Figure 36: Speed Comparison without Protection - Transient State (F1)** **72**
**Figure 37: Speed Comparison without Protection - Steady State (F1)** **72**
**Figure 38: Motor Currents Zoomed (Steady State) – Simscape (F1)** **73**
**Figure 39: Motor Currents Zoomed (Steady State) – Simulink (F1)** **73**
**Figure 40: Phase-to-Phase Short-Circuit Fault Model with Protection – Simulink (F1)** **74**
**Figure 41: Motor Currents Zoomed (Transient State) – Simscape (F1)** **74**
**Figure 42: Motor Currents Zoomed (Transient State) – Simulink (F1)** **74**
**Figure 43: Speed Comparison with Protection - Transient State (F1)** **75**
**Figure 44: Speed Comparison with Protection - Steady State (F1)** **75**
**Figure 45: Motor Currents Zoomed (Steady State) – Simscape (F1)** **76**
**Figure 46: Motor Currents Zoomed (Steady State) – Simulink (F1)** **76**
**Figure 47: Motor Currents Zoomed (Transient State) – Simscape (F2)** **77**
**Figure 48: Motor Currents Zoomed (Transient State) – Simulink (F2)** **77**
**Figure 49: Speed Comparison without Protection - Transient State (F2)** **77**
**Figure 50: Speed Comparison without Protection - Steady State (F2)** **78**
**Figure 51: Motor Currents Zoomed (Steady State) – Simscape (F2)** **78**
**Figure 52: Motor Currents Zoomed (Steady State) – Simulink (F2)** **78**
**Figure 53: Speed Comparison with Protection - Transient State (F2)** **79**
**Figure 54: Speed Comparison with Protection - Steady State (F2)** **80**
**Figure 55: Motor Currents Zoomed (Transient State) – Simscape (F2)** **80**
**Figure 56: Motor Currents Zoomed (Transient State) – Simulink (F2)** **80**
**Figure 57: Motor Currents Zoomed (Steady State) – Simscape (F2)** **81**
**Figure 58: Motor Currents Zoomed (Steady State) – Simulink (F2)** **81**
**Figure 59: Speed Comparison with Protection - Steady State (F4)** **82**
**Figure 60: Motor Currents Zoomed (Steady State) – Simscape (F4)** **83**
**Figure 61: Motor Currents Zoomed (Steady State) – Simulink (F4)** **83**
**Figure 62: Speed Comparison with Protection - Transient State (F4)** **83**
**Figure 63: Motor Currents Zoomed (Transient State) – Simscape (F4)** **84**
**Figure 64: Motor Currents Zoomed (Transient State) – Simulink (F4)** **84**
**Figure 65: Speed Comparison with Protection - Steady State (F5)** **84**
**Figure 66: Speed Comparison with Protection - Transient State (F5)** **85**
**Figure 67: Motor Currents Zoomed (Steady State) – Simscape (F5)** **85**
**Figure 68: Motor Currents Zoomed (Steady State) – Simulink (F5)** **86**
**Figure 69: Motor Currents Zoomed (Transient State) – Simscape (F5)** **86**
**Figure 70: Motor Currents Zoomed (Transient State) – Simulink (F5)** **86**
**Figure 71: Motor Currents Zoomed (Transient State) – Simscape (F6)** **87**
**Figure 72: Speed Comparison with Protection - Transient State (F6)** **87**
**Figure 73: Motor Currents Zoomed (Transient State) – Simulink (F6)** **88**

**Figure 74: Speed Comparison with Protection - Steady State (F6)** **88**
**Figure 75: Motor Currents Zoomed (Steady State) – Simscape (F6)** **89**
**Figure 76: Motor Currents Zoomed (Steady State) – Simulink (F6)** **89**
**Figure 77: Motor Currents Zoomed (Transient State) – Simscape (F8)** **89**
**Figure 78: Motor Currents Zoomed (Transient State) – Simulink (F8)** **90**
**Figure 79: Speed Comparison with Protection - Transient State (F8)** **90**
**Figure 80: Motor Currents Zoomed (Steady State) – Simscape (F8)** **91**
**Figure 81: Motor Currents Zoomed (Steady State) – Simulink (F8)** **91**
**Figure 82: Speed Comparison with Protection - Steady State (F8)** **91**
**Figure 83: Motor Currents Zoomed (Steady State) – Simscape (F9)** **92**
**Figure 84: Speed Comparison with Protection - Steady State (F9)** **92**
**Figure 85: Motor Currents Zoomed (Steady State) – Simulink (F9)** **93**
**Figure 86: Speed Comparison - Steady State (F10 - 50 % Reduction)** **94**
**Figure 87: Speed Comparison - Steady State (F10 - 20 % Reduction)** **94**
**Figure 88: TLM Injectable Socket** **97**
**Figure 89: TLM Injectable Socket Block Diagram [13]** **98**
**Figure 90: Fault Class Information** **99**
**Figure 91: Bit Position Information** **99**
**Figure 92: Fault Injection Examples** **100**
**Figure 93: Normal Line** **101**
**Figure 94: Branched Line** **101**
**Figure 95: Simulink System with Connector Blocks to Interface to SystemC** **102**
**Figure 96: Exporting Simulink to SystemC** **104**
**Figure 97: Example of an interpolated reference curve with the assigned tolerance rectangles** **108**
**Figure 98: Motor Control Application Block Diagram** **111**
**Figure 99: Control Software** **112**
**Figure 100: Protection Mechanisms** **112**
**Figure 101: PWM Generator** **113**
**Figure 102: Power electronics (Simscape)** **114**
**Figure 103: LC filter, DC link and Inverter (Simulink)** **115**
**Figure 104: LC filter + DC-link (Simulink)** **116**
**Figure 105: Inverter module (Simulink)** **117**
**Figure 106: Synchronous Motor** **118**
**Figure 107: Motor in dq-coordinates** **118**
**Figure 108: Flux-Torque table** **119**
**Figure 109: Simulation Platform** **119**
**Figure 110: Simulation Performance** **120**
**Figure 111: TCL Script** **123**
**Figure 112: Virtual Drive Platform for Fault Simulation** **124**

**Figure 113: Format for Fault Injection Tests** **124**
**Figure 114: Motor Speed Output** **125**
**Figure 115: Curve Comparison Output** **126**
**Figure 116: Simulation Platform for Motor Condition Monitoring** **128**
**Figure 117: Fault Injection GUI** **129**
**Figure 118: Scenario 1 - Simulation Output** **131**
**Figure 119: Scenario 2 - Simulation Output** **132**
**Figure 120: Scenario 3 - Simulation Output** **133**
**Figure 121: Simulation Platform with Belt-conveyor** **134**
**Figure 122: Virtual Conveyor Belt** **135**
**Figure 123: Simulation Output without Fault Injection** **136**
**Figure 124: Simulation Output with Fault Injection** **137**
**Figure 125: HIL Setup with SINAMICS G120 with CU240 and PM240** **139**
**Figure 126: CU Hardware with Virtual PM** **141**
**Figure 127: HIL Demonstrator Setup** **142**
**Figure 128: SINAMICS STARTER User Interface** **143**
**Figure 129: ControlDesk Experiment** **143**
**Figure 130: Fault detected in SINAMICS STARTER** **144**
**Figure 131: Motor Voltages and Currents measured using an Analog Oscilloscope** **145**

This page is intentionally left blank.

# 1 Introduction

## 1.1 Robust Motion Control Systems

Production plants of the future for industrial production 4.0 [1] [2], are highly complex, intelligent systems that consist of a variety of heterogeneous components: software, microelectronics, power electronics, sensor technology and actuators. A typical motion control system shown in Figure 1 in which the conveyor belt and robotic arms are coordinated for fast and very precise position and path control to pick and sort the items moving on the belt-conveyor.

The basic components in a typical motion control system are the application controller (PLC – Programmable Logic Controller), the drive (the control unit (CU) along with power electronics), the motor, the mechanical elements, and the feedback device or position sensor. The path planning or trajectory calculations are performed in the PLC, which sends low-voltage command signals to the drive, which in turn applies the necessary voltage and current to the motor, resulting in the desired motion. Sometimes feedback devices on the motor or the belt-conveyor are used to notify the drive or the controller with specific details about the actual movement of the motor shaft or the belt-conveyor. This feedback data is used to increase the accuracy of the motion, and can be used to compensate for dynamic changes that may occur on the belt-conveyor. The robot-arms are required to pick "on-the-fly" objects moving on conveyor belts; the instantaneous location of moving objects is computed by the vision system acquiring images from a stationary, down looking camera. Motor condition monitoring is constant monitoring of the motor’s condition which enables changes in the monitored operating parameters to be recorded at a very early stage of fault development. The acceleration sensor is used for the condition monitoring of the motor to detect vibrations and jerking.

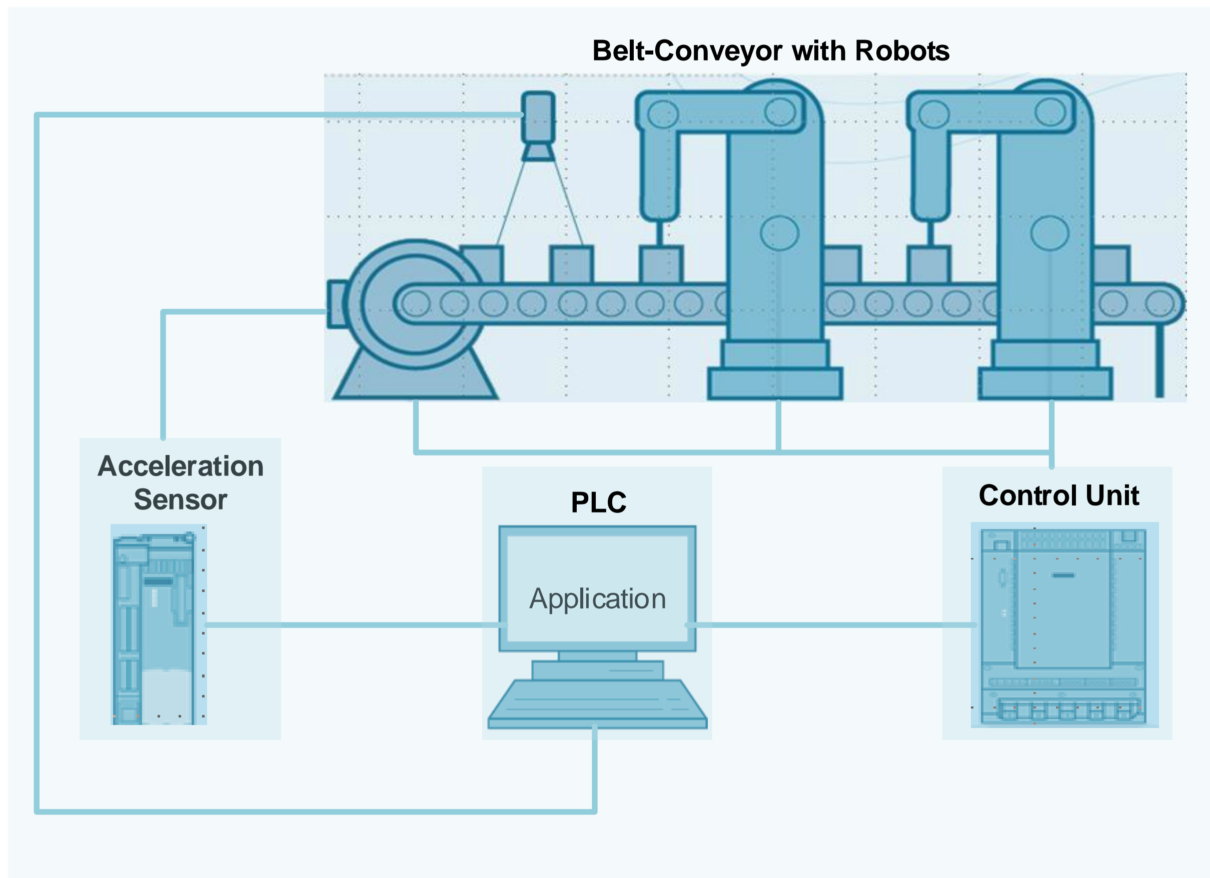


**Figure 1: Typical Motion Control System with Sensor, PLC and CU [3]**

### 1.1.1 Traditional Development

Figure 2 shows the traditional development flow of motion control systems, which was slow paced, sequential and embedded software development was dependent on the availability of physical boards. When the software content of an embedded system was relatively small, it was safe to wait for hardware to become available to start developing the software. Teams that wanted to get started sooner used register-transfer level (RTL) descriptions to represent the hardware, but that process was painfully slow. Register Transfer Level (RTL) abstraction is used to create high-level representations of a circuit, from which lower-level representations and ultimately actual wiring can be derived.

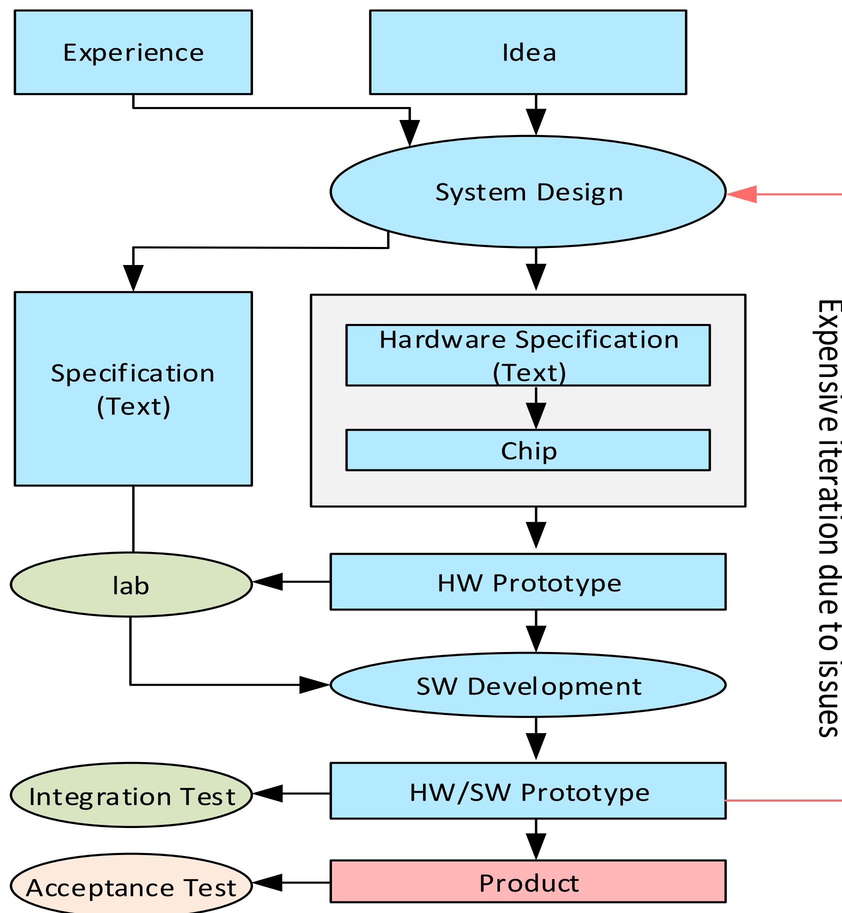


**Figure 2: Traditional Product Development Flow**

### 1.1.2 Design and Validation Challenges, Parallel Development

A well-proven way of dealing with the complexity of engineering systems is to find a suitable abstraction level to work with, since at high abstraction level details can be masked or hidden and a big picture view of the system can be achieved, without getting lost in details [3] [4]. Time-to-market pressures and system complexity force reconsideration in how these systems are designed. Systems design companies are increasingly designing at higher levels, which require understanding and validating software earlier in the process. This has led to the *hardware/software* parallel development i.e., hardware (right) and software (left) are developed parallel as shown in Figure 3.

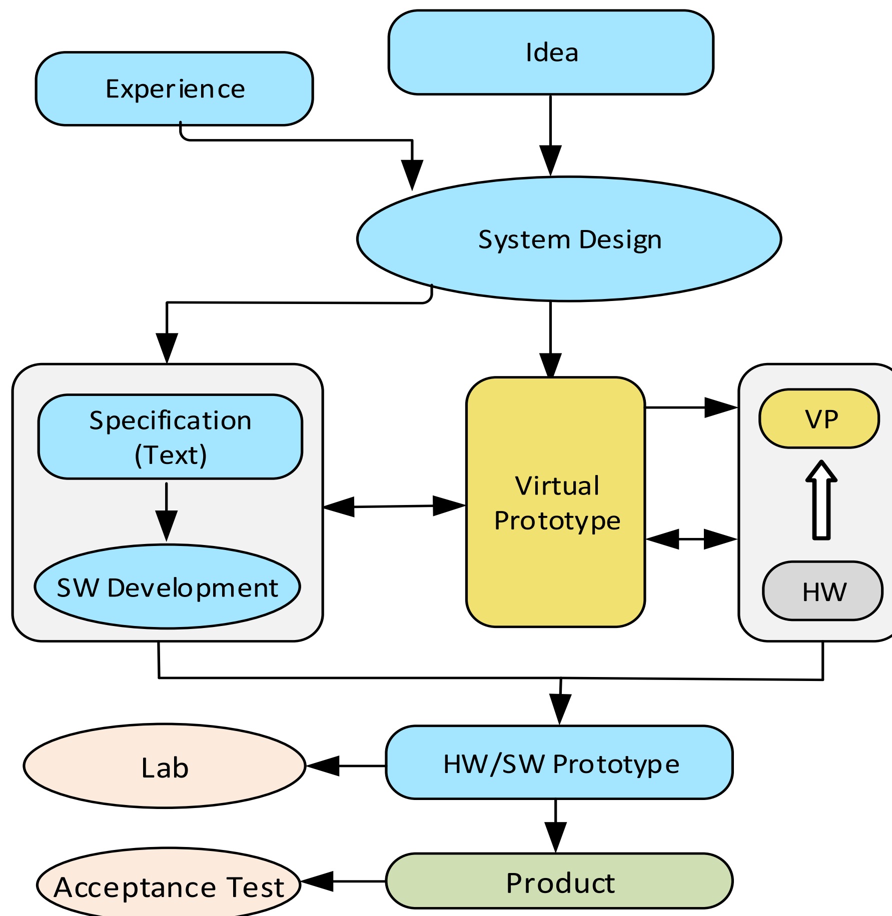


**Figure 3: Hardware/Software Parallel Development**

In parallel *hardware/software* development, hardware and software teams work in lock-step, communicating regularly, and continuously integrating hardware and software. The simple way to think about this trend is that important tasks that were done later in the design flow are now being started earlier. The software development begins before hardware is completed by using hardware simulation models of electrical and electronic components replacing their physical counterparts. Prime examples of hardware/software parallel development are the efforts in software development that are early enough to contemplate hardware changes, i.e., for e.g., hardware optimization and hardware dependent software optimization.

The motion control systems shown before have to cope with failures of all kinds to guarantee safety of operators and machine integrity at any time. Validation of this ability

today is still mostly done by final integration and system tests on hardware after development. These tests are already complex and expensive today and are not able to completely cover all possible kinds of failures, as certain failures cannot be provoked in real hardware. Moreover, the late execution of the tests may cause long iteration loops in case weaknesses are detected in final tests. Early and comprehensive statements about system behaviour under fault conditions are necessary and should be integrated into the product development, a way to counter these pressing problems.

ESL (Electronic System Level) design and verification [5] is an established electronic design methodology at most of the world's leading SoC design companies and it is being used increasingly in system design. With ESL methodologies, the SoC designs are described and analysed at a level of abstraction at which the functional behaviour can be described without considering many details of the hardware implementation. Virtual prototyping is an ESL technique where a software simulation of the entire hardware platform is created using simulation models of the various blocks in the system [4]. A virtual prototype is a software functional model that implements the behaviour of the real hardware device and provides a lot of advantages. The models can be debugged and traced to capture all interface and internal hardware states and better controllability is supported which enables developers to modify hardware behaviour's for software and system validation. However, virtual prototyping works well for software development but falls short when more detailed hardware models are required and it is plagued by model availability, its creation cost and effort. VPs also suffer from less tool support and methodology features e.g., VPs lack simulator support with inherent fault-injection features which considerably hinders the development and validation of complex industrial electronic systems.

Due to the heterogeneous nature of system components in industrial electronic systems, full-system simulation is necessary since the isolated view of individual subsystems comprising a closed-loop control system does not suffice to represent and thus validate the overall functionality. As strong interdependencies between subsystems exist, the interactions between them are of importance [6], which highlights the need for multi-domain simulation. Multi-domain simulation is the ability to efficiently and accurately simulate systems by consideration of different domains (e.g., thermal, electrical, mechanical, hardware/software,

etc.) together within one simulation or a coupled co-simulation [7]. Therefore, fast and accurate time-domain simulations are crucial in today's heterogeneous systems simulation, design exploration, and verification. Hence, the full-system simulation at higher levels is performed provided it enables realistic system design and testing. Designers currently use high-level system design languages, such as SystemC, to model only the digital parts of a system. Simulink [8] and SystemC-AMS [9] are two well-known modeling languages used to model non-digital components of a system and its building blocks behaviourally.

## 1.1 Research Contributions

In this work, we propose a virtual stress test methodology [10] based on fault injection into multi-domain virtual platforms to ease and speedup system validation against failures. Multi-Domain Virtual prototypes (MDVPs) [11] can be used for early validation by performing fault effect simulation abstracting the hardware components which the software is developed for. This allows the control software to protect against a wide variety of errors through extensive stress tests and also protection even against errors that up to now could not or barely been tested with real hardware setups. Thus, the security of systems is additionally increased despite its rapidly growing complexity.

A common methodology to perform efficient fault effect simulation of heterogeneous system parts are necessary to successfully validate and also to provide comprehensive statements about system behaviour in presence of faults. We define techniques to model faults across different domains, which can be used to conduct fault injection simulations in multi-domain virtual prototypes. These techniques are used to enhance the physical component models before integrating them into the virtual platform. Non-digital parts of the system are often modeled in industry using MATLAB/Simulink. For the digital parts of the system we use transaction-level modeling (TLM) -based fault injection techniques [12] by defining appropriate fault models. But fault simulations in such systems can only be done based on virtual prototypes at high abstraction levels, which allow for execution of real SW stacks within only minutes of simulation time. At high abstraction level, it is difficult to validate fault injection as these models are not accurate enough to simulate the physical behaviour.

The key to capturing physical system behaviour in presence of faults is to have an accurate and appropriate multi-domain simulation model of a physical system, which incorporates important mechanisms such as electrical effects, magnetic effects, mechanical loading etc. Modeling these mechanisms is difficult and more importantly simulating such a system is very slow. We propose to go from an accurate simulation model to allow for consideration of the true physical effects to the abstract representation which is fast and detailed enough to simulate the behaviour of the overall system. Once we have both the accurate and the abstract simulation models, the effect of a particular fault can be transferred by following these steps. (i) Simulate the accurate model in presence of the fault and capture the behaviour (ii) Transfer the fault behaviour onto the abstract model and simulate the overall system. We also propose and develop a generic fault injection framework based on virtual prototypes for fault effect simulation of industrial motion control systems. Once the models are integrated into a virtual platform, the automated stress tests are carried out using scripts or each individual test interactively.

## 1.2 Thesis Outline

Chapter 2 provides background knowledge for the remainder of this thesis. It starts by explaining the ESL methodology and system level design along with virtual porotypes. After that, we discuss the modeling of digital and analog/mixed-signal parts of the industrial systems in detail. Later, we will look at the model-based design and in-the-loop technologies. Finally, we briefly discuss the functional safety standards IEC61508 and ISO26262.

Chapter 3 describes the state-of-the-art relevant for work presented in this thesis. It also describes the motion control systems (MCS) along with providing exemplary faults in heterogeneous component parts in MCS. Later, fault injection current techniques are discussed. At the end, details of the research context which includes aim and research objectives along with methodology are given.

Chapter 4 describes the proposed concept for successfully conducting stress tests based on virtual prototyping. This chapter defines modeling techniques for analog and analog/mixed signal models to incorporate fault models for using inside virtual prototypes. Abstract fault models for analog and analog/mixed signal and also digital parts based on TLM

are defined. A concept to transfer fault behaviours from electrical to data-flow models is proposed and presented with examples. A generic fault injection framework to carry out fault injection tests using automated scripts with post processing is described.

Chapter 5 presents the implementation details of an industrial motor control application and automated fault injection tests along with results.

Chapter 6 presents the demonstrators for industrial use cases. These include two virtual HIL; condition monitoring of motor using an acceleration sensor and a conveyor belt application using a virtual robotic experimentation platform (V-REP simulator). Finally, HIL demonstrator based on SINAMICS G120 drive with fault injection is presented.

Chapter 7 presents the conclusions on what has been achieved in this work and proposes future prospects.

# 2 Background

## 2.1 Electronic System Level (ESL)

ESL (Electronic System Level) design and verification [5] is an established electronic design methodology at most of the world's leading SoC design companies and it is being used increasingly in system design. With ESL methodologies the SoC designs are described and analysed at a level of abstraction at which the functional behaviour can be described without many details of the hardware (RTL) implementations. Hence many engineering tasks and design optimizations can be sufficiently accomplished more quickly, efficiently and cheaply than at the RTL.

### 2.1.1 ESL Design Methodology

The "design gap" shows the spread between technology capabilities and hardware design capabilities. The rising "design gap" demands for continuous improvements of the design productivity. One of the most critical issues is how the available chip has to develop even more complex integrated circuits and systems under fixed time-to-market and quality constraints. Here, designers describe an integrated circuit with an HDL (e.g. Verilog or VHDL) at RTL. There, the design is (almost) automatically synthesized down and circuit tool suites are applied using techniques such as tracking and reporting information about the code coverage, or performing a constraint random simulation. Two basic approaches could help to improve the design productivity: The first one lifts the design level to a higher more abstract level above RTL. The second approach introduces a design reuse methodology by means of (third-party) IP components. Modern flows combine both approaches to maximize benefits.

The SystemC TLM standard [13] does not focus on abstraction levels. In fact, the standard mentions particular use cases, such as software development, software performance analysis, or hardware architecture analysis. These use cases are supported by two different

coding styles, i.e. loosely-timed and approximately-timed modeling. Coding styles guide the designer in model writing using particular programming interfaces.

### 2.1.2 System Level Design using SystemC

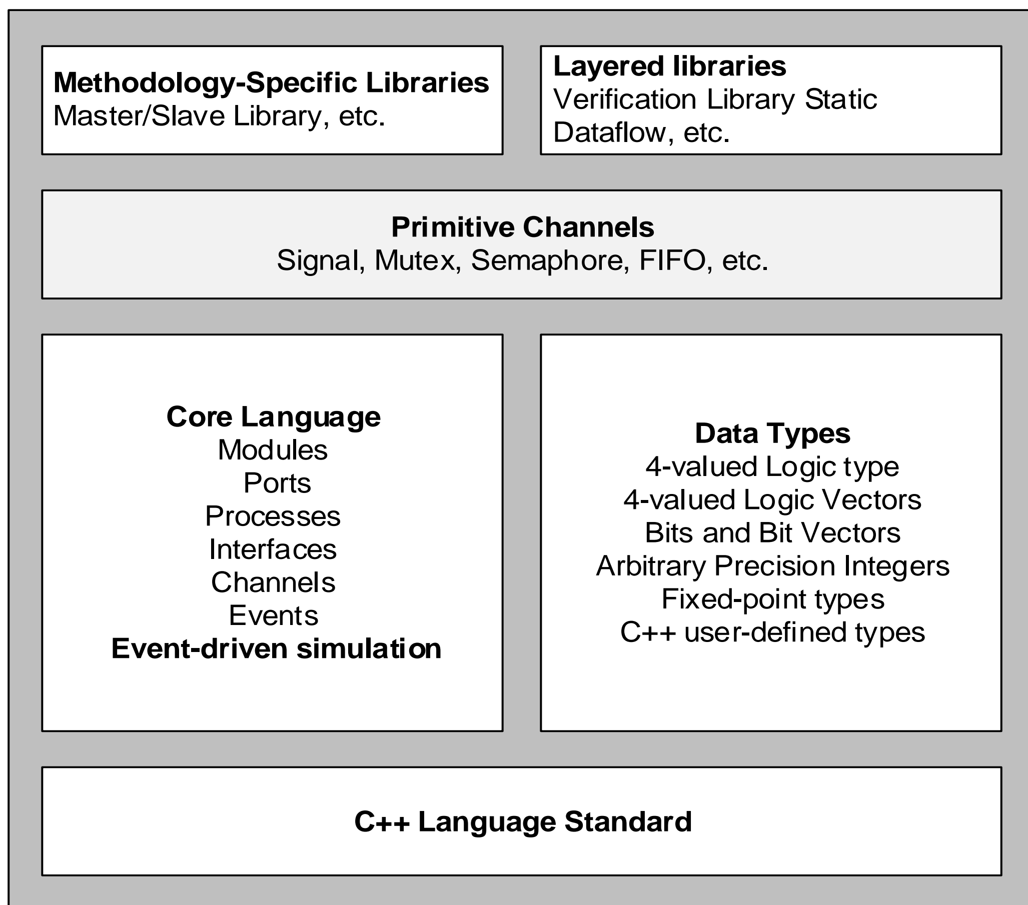


**Figure 4: SystemC language architecture [14]**

SystemC is the most accepted system level modeling language for system on chip design in the Electronic Design Automation (EDA) community. SystemC was introduced in 1999 by the Open SystemC Initiative (OSCI), which in 2011 merged with the Accellera Systems Initiative [15]. SystemC is an ANSI (American National Standards Institute) standard C++ class library [14] that allows modeling and dynamic verification of system level designs in various modeling abstractions. This includes classical RTL hardware modeling up

to transaction-level design. SystemC together with standard C++ software development tools are used to create system-level behavioural and architectural models of embedded systems. They provide hardware and software development teams a virtual platform for design, verification and test purposes without the need of hardware prototypes.

SystemC is both a system level design language and an even-driven simulation kernel. Figure 4 shows the layers of the SystemC library. The base layer highlights the fact that SystemC is built on top of C++, which makes it compatible with standard compilers and software development tools. The SystemC standard defines the three middle blocks. It defines the simulation kernel and the core language which together provide the main mechanisms for HW/SW co-design.

SystemC also defines the data types and elementary channels supporting libraries. The data type library is used for hardware modeling and for certain kinds of software programming, such as bits and bit vector data types for hardware and fixed-point data types for software implementations. Elementary channels include basic communication models widely applicable for hardware and software modeling. Finally, the upper blocks are examples of MoC and methodologies supported by SystemC, but are not included as part of the standard. They provide additional support for specific design methodologies and can be extended or form part of other standards. Further information regarding SystemC can be found in [16] and [17].

### 2.1.3 Virtual Prototypes

A virtual prototype [4] is an executable software model of a hardware/software system that runs on a host computer. These are system level simulation models that emulate (mimic) the behaviour of hardware prototypes and execute unmodified production code and provide higher debugging and analysis efficiency. Virtual prototypes are composed of system level models of processing elements and peripherals, such as memories, buses, interrupt controllers, etc. Their early availability, binary software compatibility and high execution performance enable virtual prototypes to be used to develop, debug, integrate and validate system software long before the first set of physical development targets becomes available and in most cases, even before the chip hardware design is done.

A virtual platform is a fully functional software representation of a hardware design that encompasses a single- or multi-core SoC, peripheral devices, I/O and even the user interface. The virtual platform runs on a general-purpose PC or workstation and is detailed enough to execute unmodified production code including drivers, the OS and applications at reasonable simulation speed. Users have articulated the need for virtual platforms to not be slower than 1 tenth of real time to be effective for embedded software development. The achievable simulation speed depends on the level of model abstraction, which also determines the platform's accuracy.

Virtual platforms can be used in most stages of a design. The main applications of virtual prototypes are during the development phase. They are especially useful in the following cases: software-driven verification and software development. Software-driven verification is equivalent to software-in-the-loop testing, where production code can be verified inside a virtual platform along with a simulated environment. This facilitates the verification process without the need of real hardware prototypes and experimental setups. In early design stages, virtual platforms are used as executable specification models that capture HW and SW requirements at a high abstraction. Due to their high abstraction, they can be made available in less time and can serve as golden reference models for further development and refinement stages.

Virtual platforms are also very useful for software development. Initial software applications and drivers can be developed and tested using virtual platforms. This allows the identification of software bugs and communication bottlenecks, which might be too complicated to find in real prototypes. Virtual platforms can also be useful after the deployment of a product. For instance, they may be used by a software designer to verify software updates in the form of firmware or higher level functionalities, done on multiple versions of deployed products which may not be physically available at the moment of testing the update. Aside from the previously stated verification benefits, virtual prototypes enable many other testing capabilities such as performance optimization, power analysis and fault injection.

### 2.1.3.1 Processor Models

Processor models use instruction set simulators (ISS) [18] to emulate the behaviour of the implementation model. By being an abstract representation of the RTL model, the simulation performance gained is typically several orders of magnitude faster than the implementation model. This allows hardware and software designers to run more application code or more comprehensive verification suites to prove the design. Processor models are system level descriptions of processing elements, such as microcontrollers, DSP's used in embedded systems. They are responsible for the simulation of binary code compiled for particular processor architectures and for their communication with other components inside a virtual platform. The processor models are composed of structural and behavioural descriptions. Structural descriptions contain architectural details of a processor such as functional units, caches, registers, counters, etc. Behavioural descriptions correspond to a software application that is loaded into the system model. Timing information is afterwards obtained by the combined interaction of structural and behavioural descriptions. The behaviour and timing information of a processor model is dictated by an Instruction Set Simulator (ISS). An ISS is used to perform binary translation of a software application complied for a specific microprocessor or DSP [19] instruction set and to execute it in a host computer.

Synopsys [4] virtual platforms provide comprehensive models of physical platforms. Virtual platforms combine high-speed processor instruction-set simulators and fully functional C/C++ transaction-level models (TLM) of the hardware building blocks to provide a high-level model of the hardware for early software development. The functionally accurate ARM Instruction Set Models are available from Synopsys which are fully validated against ARM processor designs and include modeling of advanced ARM technologies such as TrustZone and VFP (Vector Floating Point) [20]. Fast Models [21] uses Code Translation (CT) processor models, which translate ARM instructions into the instruction set of the host dynamically, and cache translated blocks of code. This and other optimization techniques, for instance temporal decoupling and Direct Memory Interface (DMI), produce fast simulation speeds for generated platforms, between 20-200 MIPS on a typical workstation enabling an

OS to boot in tens of seconds. They allow full control over the simulation, including profiling, debugging and tracing.

## 2.2 Digital Component Modeling

### 2.2.1 Introduction

The growing complexity in modern SoC (System on Chip) [22] is forcing the industry to look for design methodologies above RTL that can be used for architectural analysis and embedded software development. In system level design of these SoCs, communication takes the central role. Therefore a methodology for modeling communication at a higher abstraction level has become important. TLM is one such methodology that promises to be used in system level SoC modeling for early software development, architectural analysis and functional verification [23].

### 2.2.2 Transaction Level Modeling (TLM)

TLM is a transaction based modeling approach founded on high-level modeling languages such as SystemC. It highlights the concept of separating communication from computation within a system. TLM aims at enabling system level simulation of large systems, for which RTL simulation would require an unacceptable amount of time. This goal is achieved by abstracting from signal level communication and modeling complex communication operations as atomic transactions, thereby reducing the number of events to be processed by event-driven simulators.

TLM defines a transaction as a data transfer or synchronization between two modules at an instant. The definition of transaction is refined as a structure that is specific to bus protocol. The application of TLM is not tied with one programming language.

#### 2.2.2.1 Overview of TLM2.0 Standard

TLM enables simulation of complete hardware-software systems at much higher speeds. The simulation of TLM model is often orders of magnitude faster compared to RTL and low level

models. TLM models are used for different purposes by different people. Abstraction level of the model and their accuracy depends on use case and it is the clear trade-off between simulation speed and accuracy as shown in Figure 5.
Virtual platform created using TLM at different abstraction levels can be used for different purposes (see Figure 6).

- For architectural exploration and performance analysis, especially of large systems, where the efficiency of the high-level TLM model enables rapid simulation.
- As a platform for early application software development, since the TLM definition of the hardware functionality can have enough detail for software to run on it and be available months before a detailed RTL implementation.

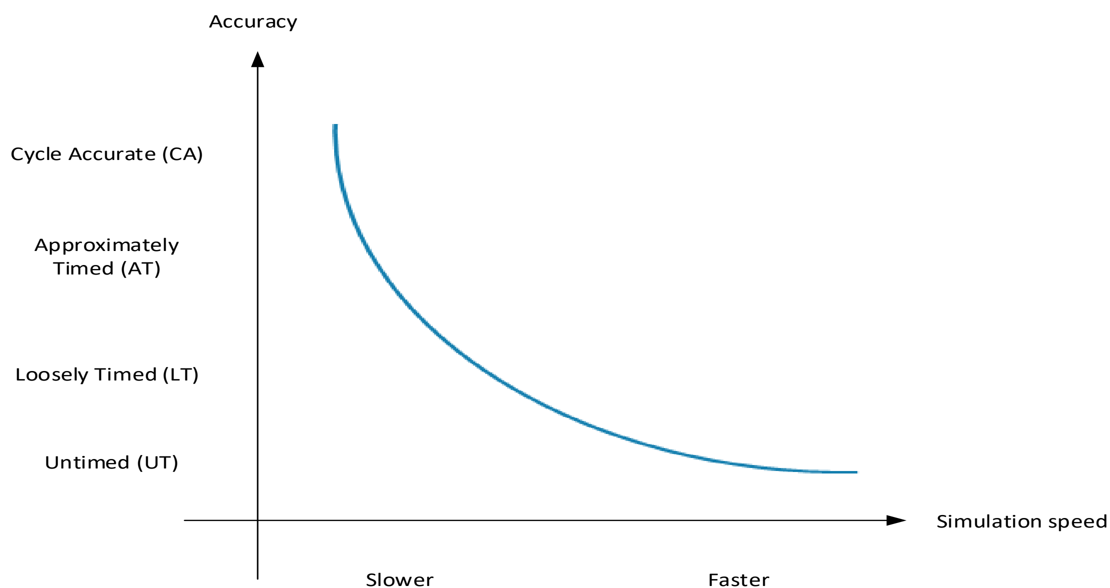


**Figure 5: Simulation speed versus accuracy trade-off [13]**

As a golden reference model for hardware verification, since TLM wrappers can be used to create a consistent interface to functional blocks whose detailed implementation is evolving.

The precision or correctness of a model in replicating the intended behaviour and activities of a system-under-design is determined by modeling accuracy of a given modeling approach. There are two decisive factors which determine the accuracy of a model, communication data granularity and timing accuracy.

The precision or correctness of a model in replicating the intended behaviour and activities of a system-under-design is determined by modeling accuracy of a given modeling approach. There are two decisive factors which determine the accuracy of a model, communication data granularity and timing accuracy.

- Communication data granularity: This criterion reflects the fineness of the data carried by the communication structure of a model. In the order of increasing accurateness, the data granularity can be broadly categorized into application packet, bus packet, and bus size levels.

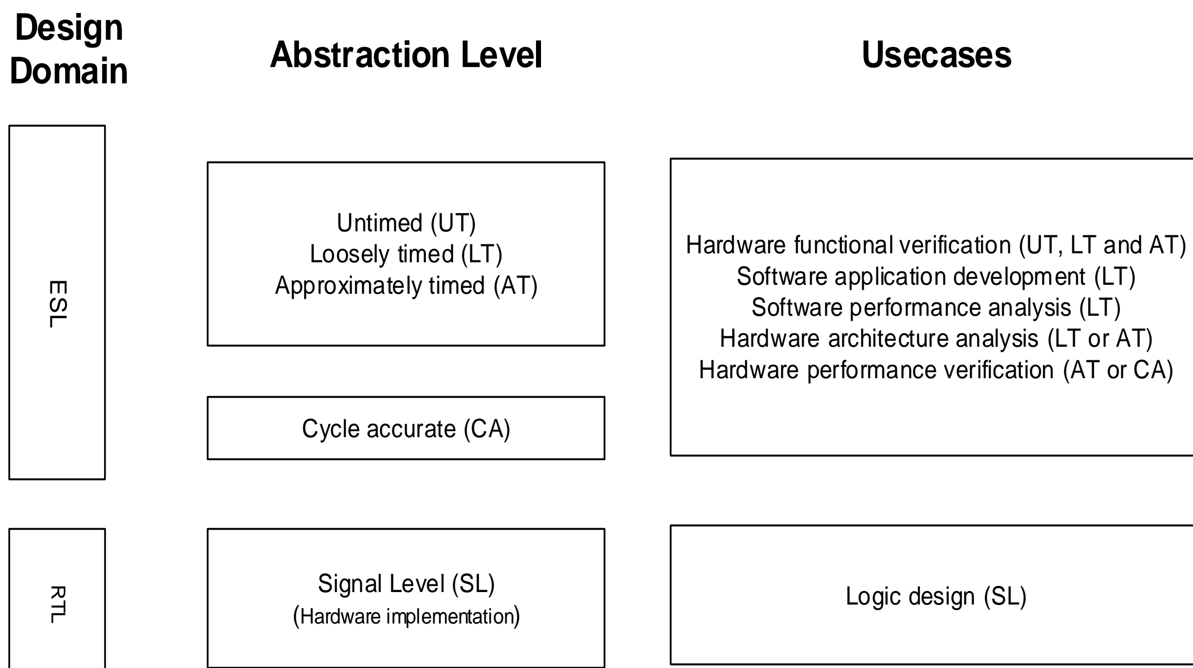


**Figure 6: Use cases and abstraction levels [13]**

- Timing accuracy: Timing accuracy of a TLM model determines the fidelity of the model to the intended timing behaviour. Based on the timing accuracy, TLM models can be broadly classified into timed and untimed models. These models are tailored for different purposes. Untimed TLM model is an architectural model targeted specifically at early functional software development and functional verification where timing annotations are not necessary. The objective of this model is high simulation speed. On the other hand, timed model is less abstract and it focuses on simulation accuracy. It can be conceptually perceived as a scale of two extremes, i.e. untimed model and CA (Cycle Accurate) model. The timing accuracy of TLM model is increased as we move from untimed to CA level end as shown in Figure 6.

Although the AT coding style enables the creation of more accurate models than the LT coding style, the amount of communication timing details that can be modeled is still limited as it uses only four transaction phases. For SoC design use cases such as detailed performance analysis and hardware/software validation that require even more accuracy, the base protocol of the AT coding style is insufficient. Typically these use cases require the communication to be accurate at the level of the system clock. At this cycle-accurate (CA) level of abstraction, the approximately-timed base protocol must be replaced with a more specific communication protocol [24]. For specific communication protocols that require additional timing points, the base protocol can be extended with custom phases.

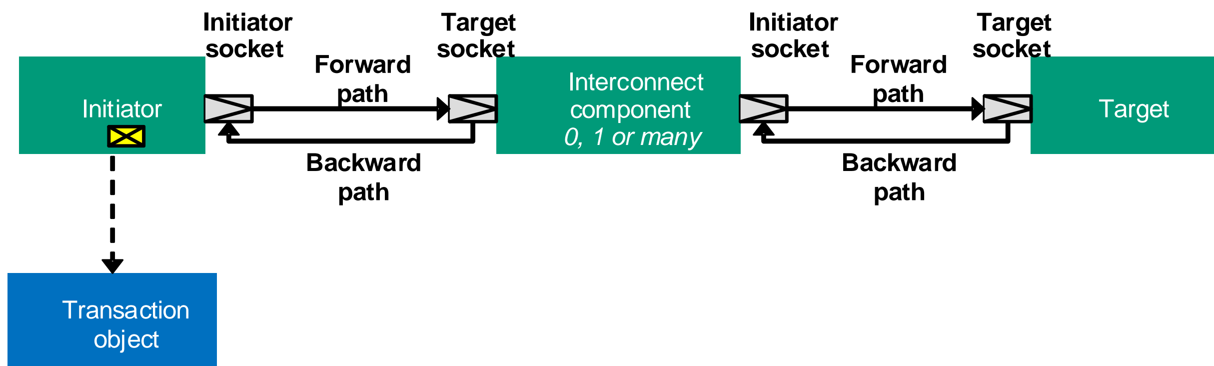


**Figure 7: TLM-2.0 communication [13]**

The SystemC TLM2 standard distinguishes between coding styles and interfaces instead of defining abstraction levels for each particular use case. A use case can be for instance software development, software performance analysis or hardware verification. The coding styles guide the designer in system modeling where the interfaces define low-level programming mechanisms. The TLM2 standard defines two coding styles: loosely-timed and approximately-timed that is supported by particular blocking and non-blocking transport interfaces.

- A module can act in three different ways:
- Initiator: This module type creates new transactions and passes them to the channel by calling a predefined interface method.
- Target: A module of this type receives transactions and executes them according to the target module task.
- Interconnect: This component type forwards a transaction and possibly modifies it. So it acts as initiator and target at the same time.

The transportation path of a transaction that is going from an initiator to a target is also called the forward path. The opposite direction is named the backward path. Over the backward path, the target informs the initiator about the transportation state. Either the modified transaction object is returned or a specific backward method is called explicitly. Two socket types encapsulate the connection between components. The initiator socket enables interface calls on the forward path by a port and on the backward path by an export. The target socket offers the same mechanism in case of a backward path.

TLM-based methodology is increasingly used to improve the design productivity of complex systems. The communication interfaces enable TLM to achieve separation of communication from computation and interoperability between components. By incorporating TLM in a SoC design flow, it is possible to model systems at various abstraction levels.

## 2.3 Analog and Mixed-Signal (AMS) Modeling

### 2.3.1 Introduction

Virtual prototyping is a well-known technique applied in digitally-oriented ESL design methodologies. The objective is to create a reference platform of the complete system or IC architecture (the “prototype” part) by an executable description captured as an abstract model (the “virtual” part), which is then simulated. In this way, virtual prototyping provides software developers and system architects an environment for software development, architecture exploration, or HW/SW co-design. However, virtual prototypes based on purely digital models and model descriptions may not offer an efficient way to capture analog behaviour, which is often an integral part of the embedded system. This can be a serious drawback, as the interfaces to the outside world which are analog in nature – are thus not adequately modeled or even not modeled at all. Examples of these analog interfaces are sensors and actuators or the power supply and management unit of an integrated circuit. Furthermore, for optimized system architectures containing analog, digital, and software functionality, the software or firmware often directly interacts with analog/mixed-signal (AMS) hardware. Therefore, the correctness and robustness of the system in terms of its architecture, functional aspects, and timing aspects cannot be validated in the analog or digital domain only hence mixed-signal simulations are needed.

### 2.3.2 Analog and Mixed-Signal Modeling Languages

VHDL-AMS, Verilog-AMS [25], and SystemC-AMS allow modeling of discrete and continuous-time signals or a combination of both. All the three afore said HDL languages can represent AMS systems at a higher level of abstraction by bringing down the simulation time while providing the intended functionality of the design. Simulink (Simscape) or other interpretation tools can be used to model AMS systems if supporting libraries and functions are available. Simulink has an in built analog tool set which can be used for AMS modeling and the accuracy results are comparable to that of spice simulation results. In the following section, we will briefly look at SystemC-AMS and Simulink (Simscape [26]).

### 2.3.2.1 SystemC-AMS

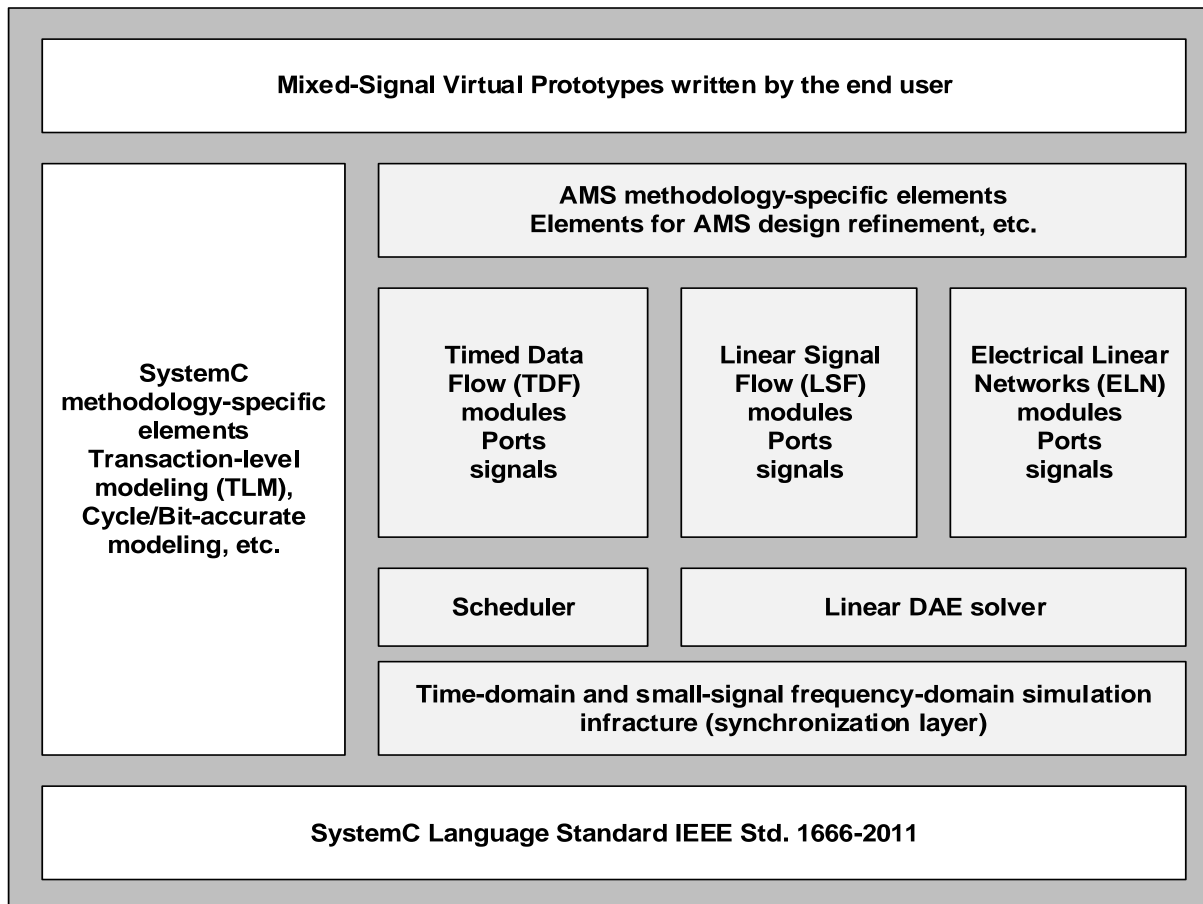


**Figure 8: AMS Extensions for the SystemC Language Standard [9]**

The SystemC-AMS language is an extension to the SystemC language. It enables analog and mixed-signal modeling and simulation capabilities. SystemC-AMS is not yet defined by an international standard, although a detailed documentation [9] and its respective proof-of-concept simulator [27] are available. SystemC-AMS extensions are fully compatible with the SystemC language standard as shown in Figure 8. The architecture of the SystemC-AMS language follows a layered approach built on top of the SystemC kernel. The user layer corresponds to the two top levels of Figure 8 and supports the following MoC's (Models of Computation): Electrical Linear Networks (ELN), Linear Signal Flow (LSF) and Timed Data Flow (TDF). The solver layer provides a linear DAE solver for ELN and LSF models and a scheduler for TDF models. The synchronization layer is responsible for embedding MoC

descriptions and their solvers/schedulers into data flow cluster processes [28] which are able to co-simulate with SystemC's discrete-event simulator.

The ELN MoC supports the modeling of non-causal continuous-time models described as electrical networks. Electrical network are built by the instantiation and interconnection of basic passive components derived from available macro models, such as resistors, capacitors and inductors, as well as sources and monitors. ELN does not support the use of non-linear elements (e.g. transistors and diodes), although there are some workarounds [29] [30]. A further limitation relies on the available linear DAE solver provided in the recent SystemC-AMS proof-of-concept simulator. The solver implements simple fixed-step numerical integration methods (a combination of backward Euler and trapezoid methods [31]) which may lead to numerical instabilities. The LSF MoC supports modeling of continuous-time systems described in a causal form using the following basic blocks: additions, multiplications, integration and delay. The connection of such blocks is used to define systems of equations, similarly to MATLAB /Simulink, which can be solved by the available linear DAE solver. The same limitations with respect to the linear DAE solver apply as before. The TDF MoC is an implementation of the Synchronous Data Flow (SDF) [32] principle, where processes are statically scheduled according to production and consumption rates. The advantage of TDF is the possibility to describe applications using a MoC similar to SDF. This is very useful for modeling digital signal processing algorithms. A further benefit is high simulation efficiency, since the TDF static scheduler reduces the dynamic overhead imposed by the discrete-event kernel of SystemC. SystemC-AMS is not yet a fully capable multi-domain simulator. It provides basic modeling capabilities for AMS systems in the electrical domain, but it lacks the modeling support for describing other type of physical energy domains. In addition, the available linear DAE solver in proof-of-concept simulator [27] is not robust enough, although improvements on this sense have been investigated [31].

#### 2.3.2.2 Simulink (Simscape)

Simulink is commercial software from Mathworks [8] for modeling, simulation and analysis of dynamic systems. It is a graphical programming environment used for creating models with block diagrams. It is capable of simulating models using different solvers and automatically

generating code. The basic elements in Simulink are blocks and lines. Basic blocks are mandatory units to perform computation or display functions, such as Add, Memory, Scope, etc. The designer builds hierarchical systems by encapsulating basic blocks into subsystems. Lines (also called edges or channels) are used to connect blocks and have register semantics (non-destructive read, destructive write). It has a large library of function blocks and users can also create their own customized blocks. By creating systems and subsystems in different blocks it is easy to organize a system and to get a clear overview of the different parts of the system, in a way that is often not available when using text based editors. Simulink provides several solvers to compute the model that contains continuous and/or discrete states. It is very efficient to use Simulink during early design stages for algorithm exploration. Simulink is often used in MBD (Model-Based Design). MBD is a process that enables fast and cost-effective development of dynamic systems, including control systems, signal processing, and communications systems.

Simscape [33] is one such toolbox which allows you to quickly create models of physical systems within the Simulink environment. With Simscape, you create physical component models based on physical connections that are directly integrated into block diagrams and other modeling paradigms. Simscape provides a set of block libraries and special simulation features especially for modeling physical systems that consists of real physical components. It is accessible as a library within the Simulink environment. They model systems such as electric motors, bridge rectifiers, hydraulic drives and cooling systems by combining essential components in a scheme. Simscape add-on products provide more complex components and analysis capabilities. From these different physical domains you can create models of your own custom components.

Simscape provides a foundation library with blocks from physical components for the domains mechanical, magnetic, electrical, hydraulic and thermal elements, as well as Simscape Multibody (formerly SimMechanics) for 3D mechanical systems, such as robots, vehicle suspensions, construction equipment, and aircraft landing gear. The following toolboxes are available, Simscape Driveline (formerly SimDriveline) for rotational and translational mechanical systems, Simscape Electronics (formerly SimElectronics) toolbox for electronic and mechatronic systems, Simscape Fluids (formerly SimHydraulics) for fluid

systems and Simscape Power Systems (formerly SimPowerSystems) for electrical power systems. To release designs to other simulation environments, consisting of HIL or vHIL systems, Simscape supports C-code generation. Transaction-Level Model Generation feature of HDL Verifier [34], when used with Simulink Coder automatically generates IEEE 1666 SystemC TLM 2.0 compatible transaction-level models. Generated SystemC models have a TLM 2.0 compliant interface with a target socket that uses the TLM 2.0 generic payload. You can select options for memory mapping, processing times as well as input and output buffering.

## 2.4 Model-Based Design and In-the-loop Technologies

Modeling is the step between the collection of high level requirements and implementation. Models allow testing and verification to be done continuously, in parallel with system design and implementation. In the early design stages, one can develop behavioural models to clarify and define detailed low level requirements. Such models may have the basic architecture of the solution, but are independent of the target platform. A model used to capture key requirements and to demonstrate correct behaviour in simulation, as well as to demonstrate traceability to high level requirements, is often referred to as executable specification. Further development of the executable specification and the addition of implementation details lead to the definition of a model that represents a final implementation. Often, such a model is optimized for code generation. It honours the data types, the target architecture and even the required coding style. Changes require a verification process that ensures the change introduced in the model for production code generation does not change the model’s behaviour.

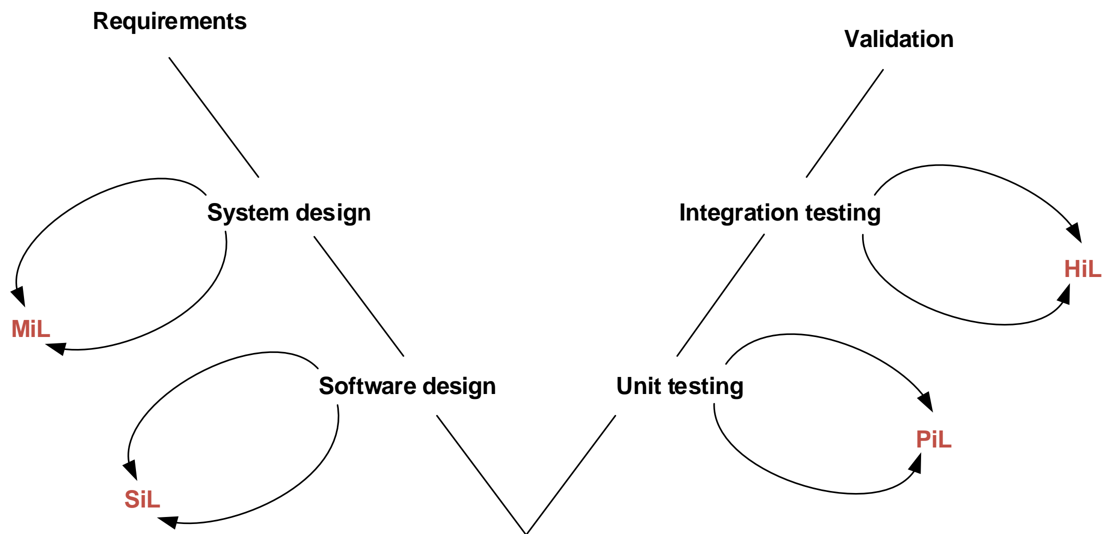


**Figure 9: The V-model of the Systems Engineering Process**

Model-based design (MBD) is a well-established approach to design embedded control systems [35]. By using MBD, engineers can find errors earlier in the design process and create high-performance motor control systems. It enables designers to start verification and testing with models of these components thereby saving design time, reducing costs and also improving overall system quality with accuracy and performance. MBD and in-the-loop methods are widely used today in the industry to develop and test control strategies. In-the-loop testing techniques allow one to reuse the model test cases and test environment for execution with the production. There are several different kinds of workflow approaches. As the integration level changes during development, the individual test execution environment changes as well. A workflow approach that takes this into consideration is the V-model, which links early development activities to their corresponding testing activities later on. The V-model that can be used as a design process of a project and it is illustrated in Figure 9.
Conventionally, the left side of the V-model represents the embedded system design phases, while the right side represents the validation and verification phases of the embedded system. The first step is the **requirement analysis**. This phase is about establishing what the ideal system has to perform, without determining how the software will be built or designed. This

is done in the section where goal specifications and requirements are made. The second phase of the V-model is the **system design**. This is the phase where the developers analyse and understand the business of the proposed system by studying the user requirements documents and try to enable the requirements and specifications. This is done by testing the model continuously using Model-in-the-loop (MIL) [36] and coming up with solutions to the requirements and specifications. The third stage is where the code is written or generated. This is the **software design** stage of the V-model. To make sure that the code is working correctly it is tested using Software-in-the-loop (SIL) [37] [38]. On the right side of the V-model are **unit testing**, **integration** and **validation**. The unit testing stage is about testing the processor. This will not be a large part in this project since a PLC that is guaranteed to work by the manufacturer is used. In other cases, where the construction of the processor is part of development, Processor-in-the-loop (PIL) [39] [40] testing can be made in this stage. The integration part is where the generated code is integrated with the hardware and Hardware-in-the-loop (HIL) [41] [42] testing is made. In this phase tests are made to see if the controller that was made in the system design stage works in the system. This links System Design together with the Integration Testing part. The last step is to validate if the results fulfils the requirements.

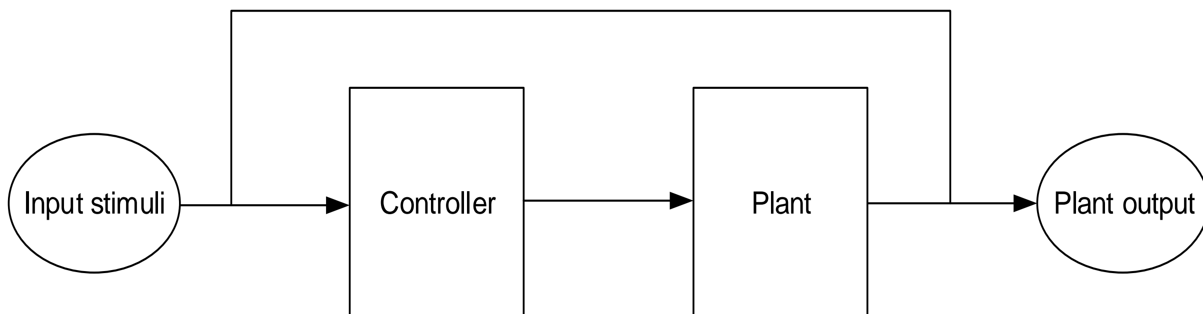


**Figure 10: Typical control system**

In a typical MBD workflow of designing a control system as shown in Figure 10: Typical control system, it is common to model a controller along with the plant it is designed to control. Plant is the common term for a model of the physical system and its environment.

The starting point is the creation of the model of the controller and the plant in a modeling tool. Then, it is common to run simulations inside the modeling tool with simulated inputs and outputs to test the design as it evolves. As the project progresses, the controller model will need to migrate to the real electronic system of software and hardware, while the plant model will be replaced by real physical systems and tested in the real environment.

### 2.4.1 Model-in-the-loop (MIL)

MIL [36] consists of mixing the physical plant model (including mechanical, electrical, thermal, etc., effects) with the control strategy at the algorithmic level (typically using state machines). This creates a complete mechatronic model that can be simulated together to find out whether the behaviour of the control logic is correct. Very often a tool that is used for this is Simulink from the company Mathworks. Initial tests can be created at this level. These tests should be reusable for the next subsequent phases. Once the MIL simulation shows that the results are correct, the control part is implemented (typically in software). Implementing the control model can be done manually by writing C code or automatically by using code generation tools.

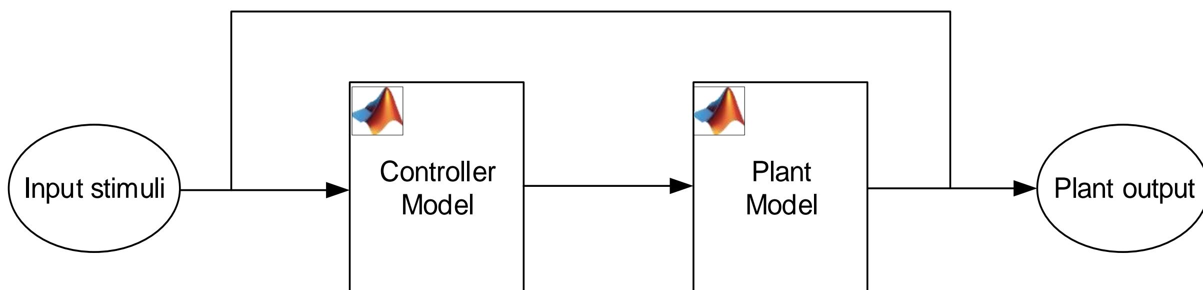


**Figure 11: Model-in-the-loop**

### 2.4.2 Software-in-the-loop (SIL)

SIL testing involves executing the production code for the controller within the modeling environment for non-real-time execution with the plant model and interaction with the user. The code executes on the same host platform that is used by the modeling environment. A code wrapper of the generated code provides the interface between the simulation and the

generated code. The problem with SIL is that the C code is compiled for the host PC instead of the target control unit (CU). This may introduce differences in precision due to different data types (saturation and overflow effects), as well as other problems due to resource limitations on the CU (memory and processing power).

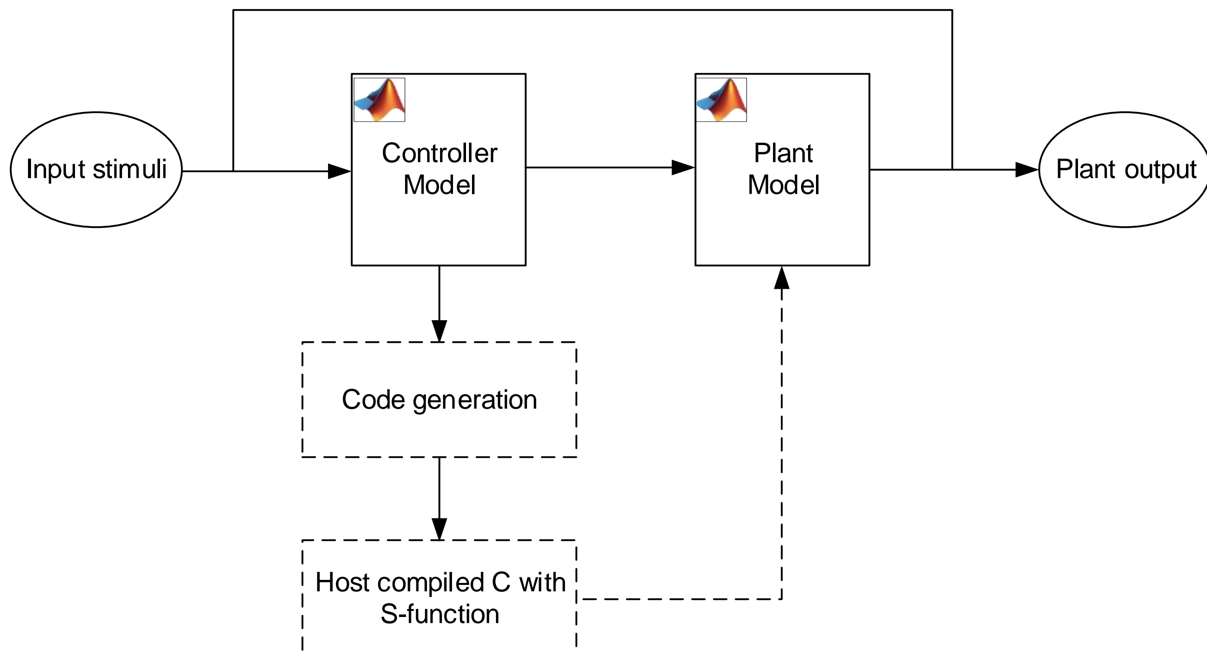


**Figure 12: Software-in-the-loop**

### 2.4.3 Processor-in-the-loop (PIL)

PIL is used to overcome some of the limitations of SIL. PIL still focuses on the control algorithmic implementation, but this time the C code is compiled for the targeted processor architecture. The execution of the software code is synchronized with the simulated physical plant in order to reuse the same tests. Memory allocation and execution time can be measured at this point. The main problem of PIL is that in reality a control function is not targeted to a single MCU architecture, but too many different MCUs depending on the specific requirements and variant of the end product. Combined with that people working with SIL are not familiar or keen to work with hardware environments hence making this approach a limited adoption.

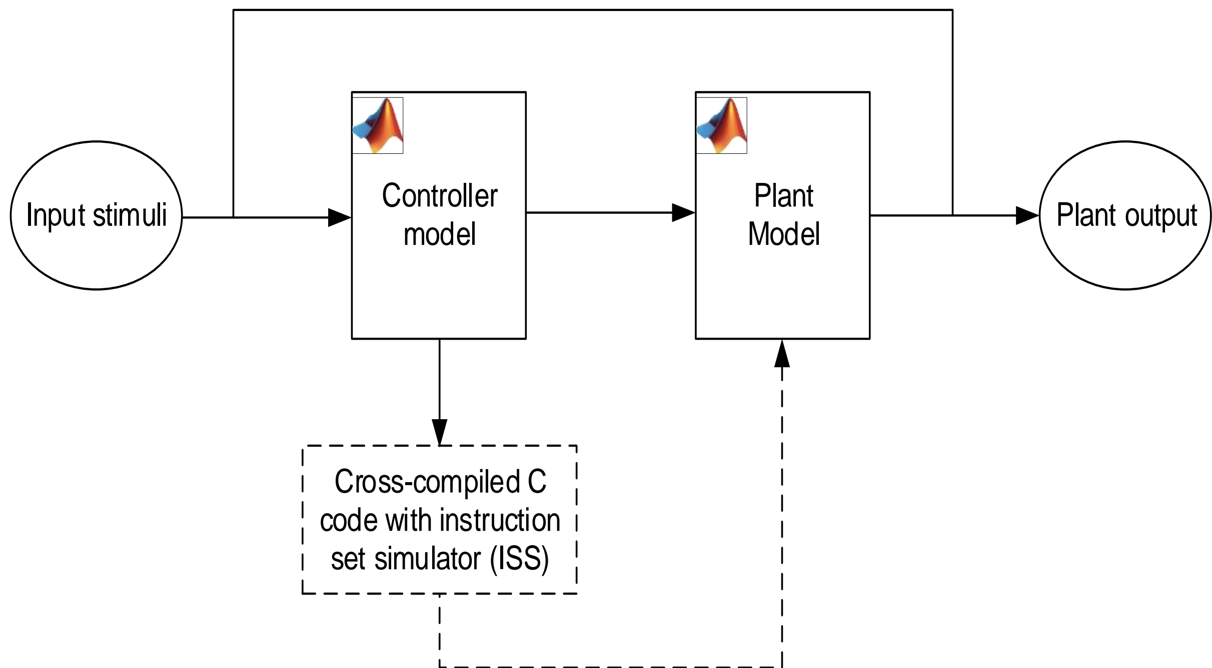


**Figure 13: Processor-in-the-loop**

### 2.4.4 Hardware-in-the-loop (HIL)

HIL simulation is a technique that is used for the development and testing of control systems which are used for the operation of complex machines and systems. HIL simulation provides an effective platform by adding the complexity of the plant under control to the test platform. The complexity of the plant under control is included in test and development by adding a mathematical representation of all related dynamic systems.

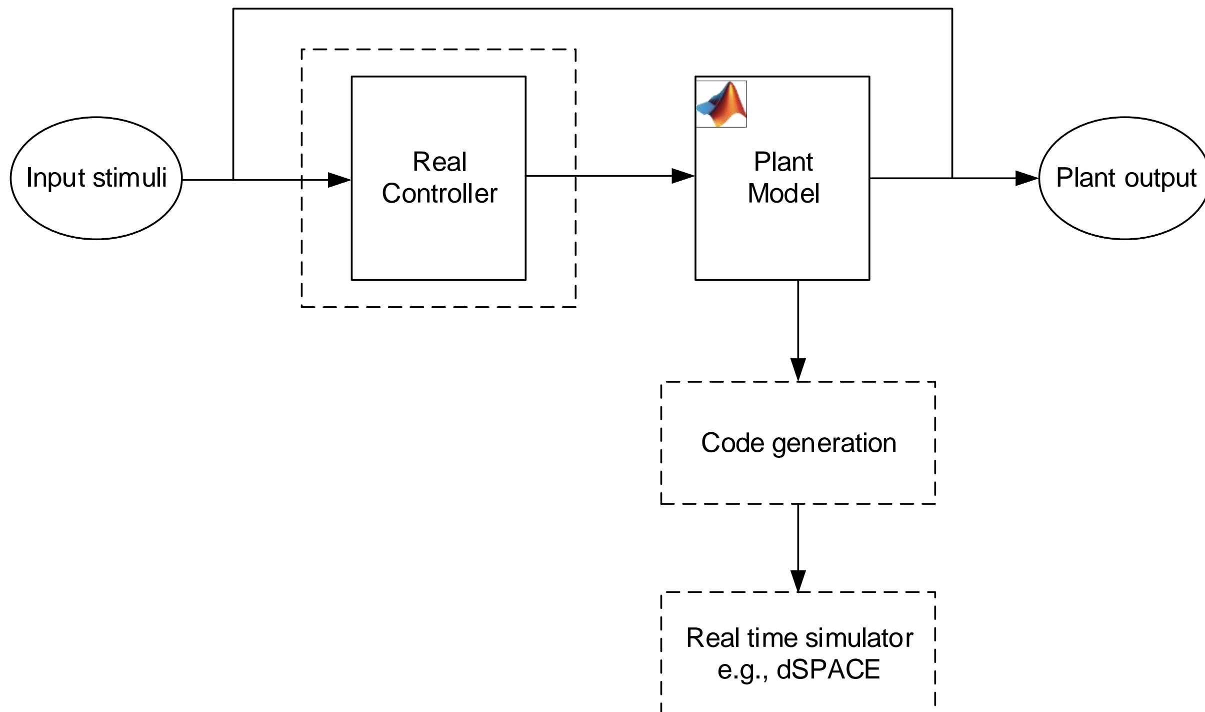


**Figure 14: Hardware-in-the-loop**

In HiL simulation, a production controller is tested against a real-time simulation of the plant (motors and machine). This capability is useful in cases where access to the actual system is limited or unavailable – for example, motors attached to large industrial machines such as printing presses or packaging equipment. HIL simulation is also invaluable when it is dangerous to test the plant's full operational envelope. Consider the risks of trying out a complex motor control algorithm in an industrial environment. If something goes wrong, a system failure can damage equipment and endanger people nearby. Testing the production controller against a real-time simulation of the motors and machine is far better. Just as important, hardware-in-the-loop simulation can be used to fully exercise system diagnostics – for example, emergency condition detection and shutdown procedures – which might be difficult or impossible to test on the motor and/or machine itself.

HiL Current Limitations:

- Limited access due to limited number of HIL systems (cost and access)

- Limited visibility and controllability of HW and SW
- Hard to deploy in regression
- Complex to setup

### 2.4.5 Virtual Hardware-in-the-loop (vHIL)

The vHIL [43] using a virtual prototype (VP) can add multiple benefits to a standard development flow. In the context of an "in-the-loop" methodology, a VP removes the dependency on the ECU hardware and enables earlier software integration and testing. This creates a smoother path between MIL, SIL and HIL. With a vHiL environment, a virtual ECU model is connected in closed-loop to the same plant model that runs, for instance, on Simulink. The same tests applied on the MIL and SIL phases can be now executed on the vHIL environment. More importantly new tests can now be created by the software testing teams without having to wait for an ECU prototype or having to allocate time on a HIL environment. The vHIL accelerates integration, testing and also improves quality through parallel regressions.

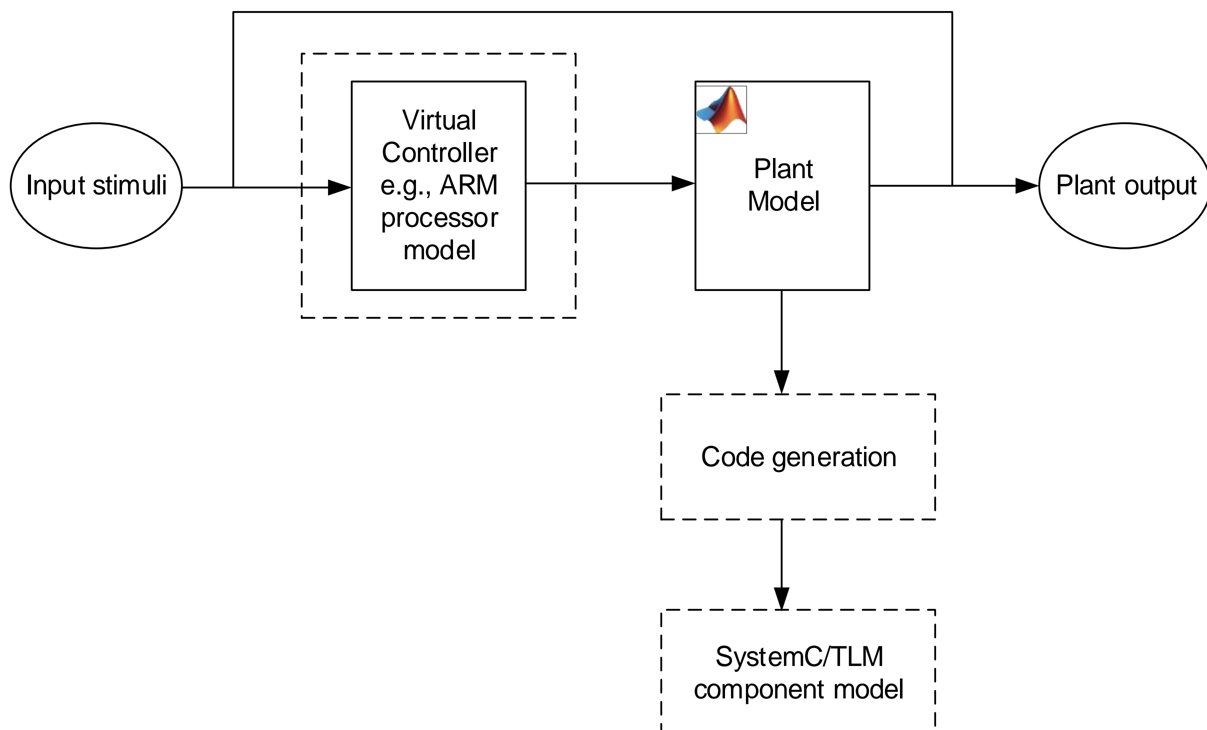


**Figure 15: Virtual Hardware-in-the-loop**

#### 2.4.5.1 Advantages over HIL

- Start early: before CU HW is available
- Reduced costs: easier deployment and setup costs
- Improve productivity: Non-intrusive SW debug and analysis
- Improve quality: more testing coverage (scripting for automated regression execution)

## 2.5 Fault Injection and Simulation

A system may not always perform the function it is intended for. The causes and consequences of deviations from the expected function of a system are called the factors to dependability. The Fault injection (FI) is defined as a validation technique for the dependability of fault tolerant systems. It is used to assess robustness, including the evaluation of error handling mechanisms. The causes and consequences of deviations from the expected function of a system are called the factors to dependability.

- Fault is a physical defect, imperfection, or flaw that occurs within some hardware or software component.
- Error is a deviation from accuracy or correctness and is the manifestation of a fault.
- Failure is the non-performance of some action that is due or expected.

When a fault causes an incorrect change in a machine stage, an error occurs. Although a fault remains localized in the affected code or circuitry, multiple errors can originated from one fault site and propagate throughout the system. When the fault-tolerance mechanisms detect an error, they may initiate several actions to handle the faults and contain its errors. Otherwise, the system eventually malfunctions and a failure occurs.

### 2.5.1 Fault Classification on Temporal Behaviour

The faults that arise during system operation are best classified by their temporal behavior, as permanent, intermittent, and transient:

- **Permanent**: Faults remain in existence indefinitely if no corrective action is taken. Many of these are residual design or manufacturing faults. Those that are not most frequently occur during changes in system operation, for instance, after system start-up or shutdown, or as a result of a catastrophic environmental disturbance such as a collision.
- **Intermittent**: Faults appear, disappear, and reappear repeatedly. They are difficult to predict, but their effects are highly correlated. Most intermittent faults are due to marginal design or manufacturing. The system works well most of the time, but fails under typical environmental conditions.
- **Transient**: Faults appear and disappear quickly, and are not correlated with each other. They are most commonly induced by random environmental disturbances.

### 2.5.2 Fault Injection Current Techniques

The fault injection techniques have been recognized for a long time as necessary to validate the dependability of a system by analysing the behaviour of the devices when a fault occurs. Several efforts have been made to develop techniques for injecting faults into a system prototype or model. Most of the developed techniques fall into these following categories.

#### 2.5.2.1 Hardware-based Fault Injection

It is performed at the physical level. This is typically done by modifying the value of the hardware input/output pins (with contact fault injection) and also by creating external disturbances with electromagnetic interference, heavy ion radiation, etc. (without contact fault injection). The advantage of hardware fault injection techniques is the ability to access some locations that are not easy to access by other techniques. It has several disadvantages such as high risk of damage to the system under study, needs a special hardware which in turn incurs additional costs and moreover, the results are difficult to collect and observe.

#### 2.5.2.2 Software-based Fault Injection

It uses an implementation model and is attractive because they don't require expensive hardware. Although the software approach is flexible, the software-based fault-injection techniques [44] [45] are limited since they can only inject errors on those locations accessible by the software. The biggest problem with software-based fault injection is that it involves changing the software by inserting code to cause errors, which means it might act differently when compared with the production software.

#### 2.5.2.3 Simulation-based Fault Injection

It uses an implementation model to perform the experiments. Simulation-based fault injection [46] has the advantage of having full access to all hardware elements in the system. Without being intrusive it has full observability, controllability and is fully deterministic. The down side of this level of simulation is that they are extremely slow. This makes them unusable for more complex fault scenarios where software must be taken into account.

#### 2.5.2.4 Emulation-based Fault Injection

This technique [47] [48] [49] has been presented as an alternative solution for reducing the time spent during simulation-based fault injection campaigns. It is based on the exploration of the use of Field Programmable Gate Arrays (FPGAs) for speeding-up fault simulation and exploits FPGAs for effective circuit emulation. This technique can allow the designer to study the actual behaviour of the circuit in the application environment, taking into account real-time interactions. However, when an emulator is used, the initial VHDL description must be synthesizable and optimized to avoid a costly emulator and also to reduce the total running time during the injection campaign. Other drawbacks include the cost of a general hardware emulation system and the implementation complexity of a dedicated FPGA based emulation board. A low cost can be reached but at the expense of a reduced speed of the fault injection campaign.

There are certain merits and demerits of every method stated above. Detailed information about this is presented in [50].

## 2.6 Functional Safety Standards

Functional safety [51] [52] means the prevention of unjustifiable risks that can arise from hazards caused by malfunctions in electrical or electronic systems. Systems comprised of electrical and/or electronic elements have been used for many years to perform safety functions in most application sectors. Computer-based systems generically referred to as programmable electronic systems are being used in all application sectors to perform non-safety functions and increasingly to perform safety functions. Functional safety is a concept applicable across all industry sectors. It is fundamental to the enabling of complex technology used for safety-related systems. It provides the assurance that the safety-related systems will offer the necessary risk reduction required to achieve safety for the equipment.

To achieve functional safety, the risk of hazards caused by system malfunction must be removed. Safety standards ensure that associated risks are reduced or removed to meet safety requirement levels. The functional safety standards include IEC-Standard IEC-61508 [53] for the general industry and ISO-Standard ISO-26262 [54] for road vehicles. These standards define the appropriate safety lifecycle and Safety Integrity Levels (SILs) to develop hardware, software and provide a safety analysis with supporting confirmation measures and processes.

IEC 61508 was developed for the industrial automation industry, but derivatives for other industries such as rail (EN 50128), medical (IEC 62304) and machinery (IEC 62061) are already existing. IEC 61508 is still used for commercial vehicles – especially off-highway vehicles.

### 2.6.1 IEC61508

IEC61508, functional safety of electrical/electronic/programmable electronic safety-related systems, is designated by IEC as a generic standard and a basic safety publication. This means that industry sectors will base their own standards for functional safety on the requirements of IEC61508. This International Standard sets out a generic approach for all

safety lifecycle activities for systems comprised of electrical and/or electronic and/or programmable electronic (E/E/PE) elements that are used to perform safety functions. This unified approach has been adopted in order that a rational and consistent technical policy be developed for all electrical-based safety-related systems. A major objective is to facilitate the development of product and application sector international standards based on the IEC 61508 series.

The IEC61508 framework consists of 7 parts:

- IEC61508-1 (General requirements): defines the activities to be carried out at each stage of the overall safety lifecycle, as well as the requirements for documentation, conformance to the standard, management and safety assessment.
- IEC61508-2 (Requirements for electrical/electronic/programmable electronic safety-related systems): Interprets IEC61508-1 for hardware.
- IEC61508-3 (Software requirements): Interprets IEC61508-1 for software.
- IEC61508-4 (Definitions and abbreviations): Contains definitions and abbreviations in the standard.
- IEC61508-5 (Examples of methods for the determination of safety integrity levels): provides risk analysis and demonstrates the allocation of safety integrity levels (SIL's). The Safety Integrity Level or SIL is a measure for the quantification of risk reduction.
- IEC61508-6: Guidelines on the application of IEC61508-2 and IEC61508-3
- IEC61508-7 (Overview of measures and techniques): provides brief descriptions of techniques used in safety and software engineering, as well as references to sources of more detailed information about them.

### 2.6.2 ISO 26262

ISO 26262 [54] is the adaptation of IEC 61508 to comply with needs specific to the application sector of electrical and/or electronic (E/E) systems within road vehicles. This adaptation applies to all activities during the safety lifecycle of safety-related systems comprised of electrical, electronic and software components.

Safety is one of the key issues of future automobile development. New functionalities not only in areas such as driver assistance, propulsion, in vehicle dynamics control, active and passive safety systems increasingly touch the domain of system safety engineering. Development and integration of these functionalities will strengthen the need for safe system development processes and the need to provide evidence that all reasonable system safety objectives are satisfied. With the trend of increasing technological complexity, software content and mechatronic implementation, there are increasing risks from systematic failures and random hardware failures. ISO 26262 includes guidance to avoid these risks by providing appropriate requirements and processes.

System safety is achieved through a number of safety measures, which are implemented in a variety of technologies (e.g. mechanical, hydraulic, pneumatic, electrical, electronic, programmable electronic) and applied at the various levels of the development process.

# 3 State-of-the-Art and Research Context

## 3.1 Related work

As discussed in section 1.1 Robust Motion Control Systems, comprehensive system tests and early applicable stress tests are an important component for the protection and qualification of electronic systems in industrial automation, especially where fast moving parts are to be controlled, i.e. in motion control systems. In this application domain, fault safety and functional safety play an important role, which is also reflected in several standards (e.g., IEC 61508, IEC 61511 etc.). System tests are now carried out on extremely complex test setups in which the motion control products to be qualified are constructed in various configurations with a number of standard machines. In these tests, the response of the system is validated to various errors e.g. interrupted communication lines, failure of the power supply of individual devices and the like. These tests are not only very complex, lengthy and expensive, but also cover a limited range of applications. This is due on the one hand to the relatively few machines available in the test compared to the almost infinite number of possible combinations in the field. On the other hand, many important errors, e.g. subtle errors in electronic circuits (such as a bit-flip in a memory cell) or the heat running and blocking of a mechanical bearing can hardly be represented in these tests.

A wide-spread approach to address this problem is the so-called Failure Mode and Effect Analysis (FMEA), which involves reviewing as many components, assemblies, and subsystems of a system as possible to identify failure modes, and their causes and effects [55] [56] [57] [58]. A support for this qualitative process by comprehensive simulation of fault effects and stress scenarios is so far not available or only at lower abstraction levels, such as the error simulation at gate level. Although, the provision of approaches to high-level error simulation has been called for several times since 2011, for e.g., at the Design Automation

Conference (DAC) 2011, the solutions still remain. At higher abstraction levels, there are solutions for model-based fault analysis using abstract black box component models used in current delivery projects, e.g. CHESS [59], SAFE [60] and SPES_XT [61] in particular with regard to current safety standards (for example ISO 26262) as well as an integration in a model-based design sequence for different application domains. The aim of the model-based fault analysis is to determine the failure probability of a system, taking account of the error rates of all components by applying established techniques for reliability analysis, e.g. the fault tree analysis (FTA). A number of monographs on the reliability analysis [62] [63] and on fault tolerance [64] [65], can be referred to in this area, which discuss this subject intensively.

In [15] a system was developed for rapid simulation in the industrial system (IEEE Std. 1666) in the field of modeling and simulation of electronically based systems based on ESL (Electronic System Level). Through the availability of the open source C++ library, the language allows the development of open, tool-independent system C / C++ libraries and the integration of any C++ modules. The bus abstraction standard TLM (Transaction Level Modeling), which is now integrated in the current IEEE Std. 1666, was developed for the further simulation acceleration. Furthermore, SystemC-AMS [66] provides extensions for the modeling and simulation of analog systems and open system libraries that implement the essential parts of the UVM (Universal Verification Methodology) standard for test environments [67]. Since SystemC is completely based on C++, proprietary C / C++ modules can be seamlessly integrated into the simulation. Thus, any model-based tool chains can also be used provided they have code generation features. For e.g., Simulink [8] or UML (Unified Modeling Language) [68] with C / C++ code generators such as Simulink Coder [69] or TargetLink [70] respectively.

In the context of the rapid co-simulation of HW / SW systems, the area of so-called virtual prototyping (VP) has developed over the last few years, which essentially means the integration of software emulators with ESL simulators, e.g., Synopsys Virtualizer [71]. Since the emulators perform the binary code of the target platform, they are based on the virtual model of the processor for early software development using the tool chains of the target platform. This includes the development of hardware-related software, such as device drivers, as advanced techniques allow execution accelerators to perform complete images (application

software, operating system, etc.). Meanwhile, Cadence, Synopsys and Mentor Graphics offer VP environments. Alternatively, there are also approaches to integrate SystemC with open source emulators, such as QEMU [72] [73] [74].

The fault simulation had its origin at gate level in order to be able to detect production errors in the production of circuits. An error based on a fault model is injected into the circuit and for a given test pattern, a simulation is performed to determine whether there is a difference between fault-free and faulty circuit behaviour. This procedure with the necessary methods for the generation of test patterns is also very well understood for the lower levels (gate level and register transfer level) [75] [76] [77] [78]. A first method of fault simulation for SystemC has been proposed in [79], but only low abstraction-level and structural errors are addressed. The increase of the fault simulation to higher levels of abstraction took place only in the first approaches [80] [68] [81]. The inclusions of software, metrics for the detection of fault coverage, the generation of scenarios as well as a methodology out of the requirements are not addressed. General methods of fault injection for SystemC models based on mutations were presented in [82] [83]. However, no reference has been made to known hardware fault models and thus the continuity to the lower levels is not guaranteed. Additionally, an attempt to use fault injection techniques for dependability evaluation of software functional model is done with development of MODIFI tool [84]. MODIFI (or MODel-Implemented Fault Injection tool) extends the fault injection methodology to behaviour models in Simulink. The tool allows for introducing single or multiple points faults on behavioural models, the fault tolerant system properties are studied by analysing faults leading to failure. Even though experiments have been performed on a variety of dataflow-algorithms of software described in Simulink but the primary focus was on the control software of an automotive application. Furthermore, so far no approaches are known which provide fault simulation for heterogeneous systems e.g. the effects of errors in analog components, on the digital circuit components and the reaction of the application software. A special application of fault simulation on embedded software has been proposed in [85]. However, software operates without considering the underlying hardware mutated, so that no realistic statements are possible for a given hardware. To validate the specification of the new MOST network standard, a first TLM-based method was developed in [86] fault simulation,

which is specifically designed for a specific application use case and not directly on the problems addressed here are generalized.

In this work, we propose a virtual stress methodology based on virtual prototypes to validate the functional safety of motion control systems. Using this methodology it will be possible to carry out system tests at an early stage and more importantly it is possible to protect the reaction of motion control systems against faults which could hardly be provoked in real hardware setups.

## 3.2 Motion Control Systems (MCS)

Motion control systems contain both analog and digital circuitry parts and the associated software as shown in Figure 16. The performance of a motion control system depends on the interaction of these components. The analog components include the power module, the encoder and the motor. The digital components are the control unit, the pulse logic control and monitoring with the associated software. Examples of work machines are conveyor belts, fans, pumps, printing machines, industrial robots, crane systems, wind power plants.

Power electronics of a power module converts an input alternating voltage of a fixed frequency and amplitude into a variable output voltage with variable frequency and amplitude. The components of the PM power section are shown in Figure 17. The control unit and the pulse logic will determine and regulate the frequency and amplitude of the alternating voltage at the motor that drives the work machine. The power rectifier of the PM power section in Figure 17 consists of an uncontrolled three-phase bridge circuit. Their task is to convert the alternating voltage of the power supply into a (pulsating) DC voltage, which is smoothed by the intermediate circuit capacitor. The inverter consists of three bridge divisions, each consisting of two IGBTs and two diodes. The inverter converts the intermediate circuit voltage into a three-phase pulsating output voltage.

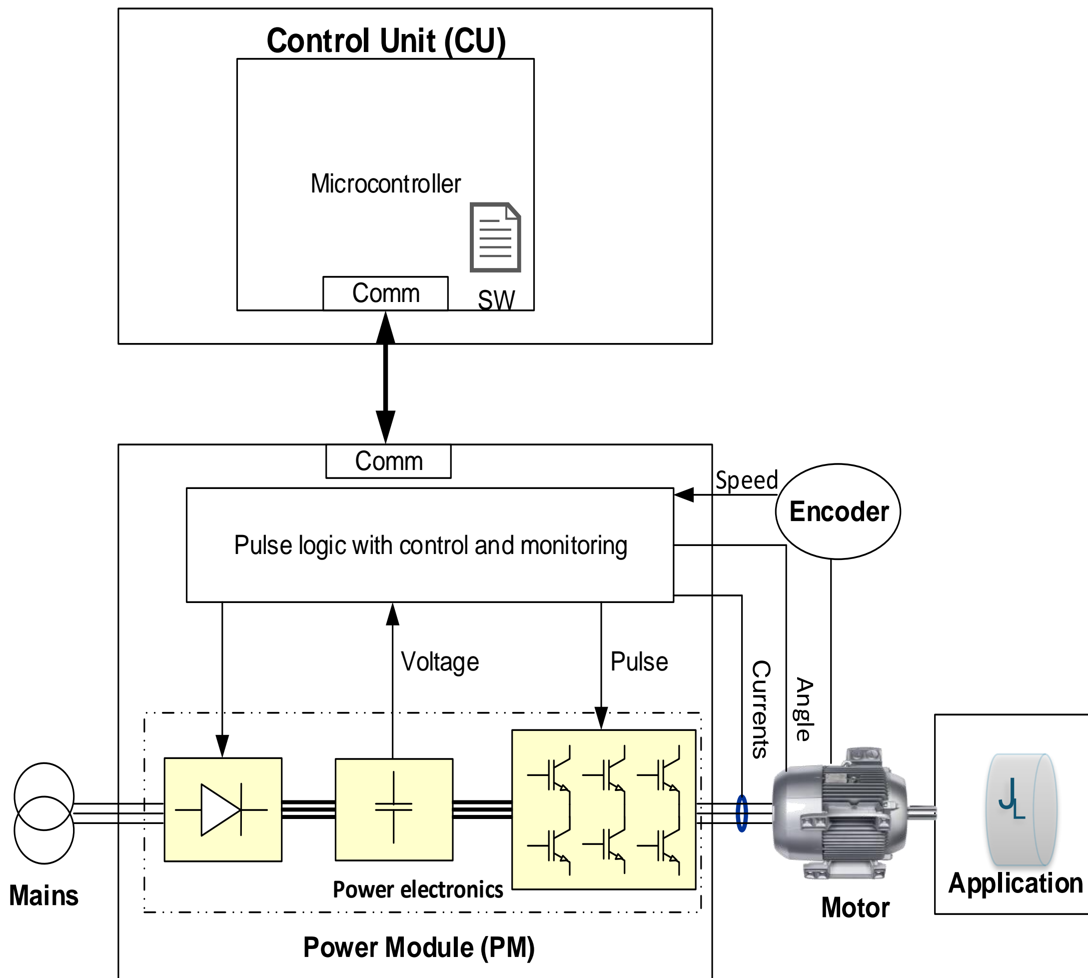


**Figure 16: Drive Model of a Motion Control System**

The encoder in Figure 16 provides the speed and position information required for the current control to the pulse logic of the power module. The electronics for control and monitoring will determine the control signals of the electric drive from the *set point* values of the control unit and the *actual* values from the frequency converter of the power module and from the sensor.

The motor of an electric drive converts electrical energy into mechanical energy. Since the requirements for electric motors are extremely diverse, there are both DC motors and three-phase motors. For three-phase motors, a distinction is made between synchronous motors [87] and asynchronous motors [88]. Synchronous motor is a typical AC electrical motor that is capable of producing synchronous speeds. In these motors, both the stator and

the rotor rotate at the same speed thus achieving synchronization. The basic working principle is, when the motor is connected to the mains, electricity flows into the stator windings, producing a rotating electromagnetic field. A synchronous motor is started by applying AC power to the stator and an external DC power then is applied to the rotor coils after the motor reaches maximum speed. This produces a strong, constant magnetic field in the rotor, which locks in step with the rotating magnetic field of the stator. This is in turn induced on to the windings in the rotor which then starts rotating. As a result of this interlocking, the motor has either to run synchronously or not run at all.

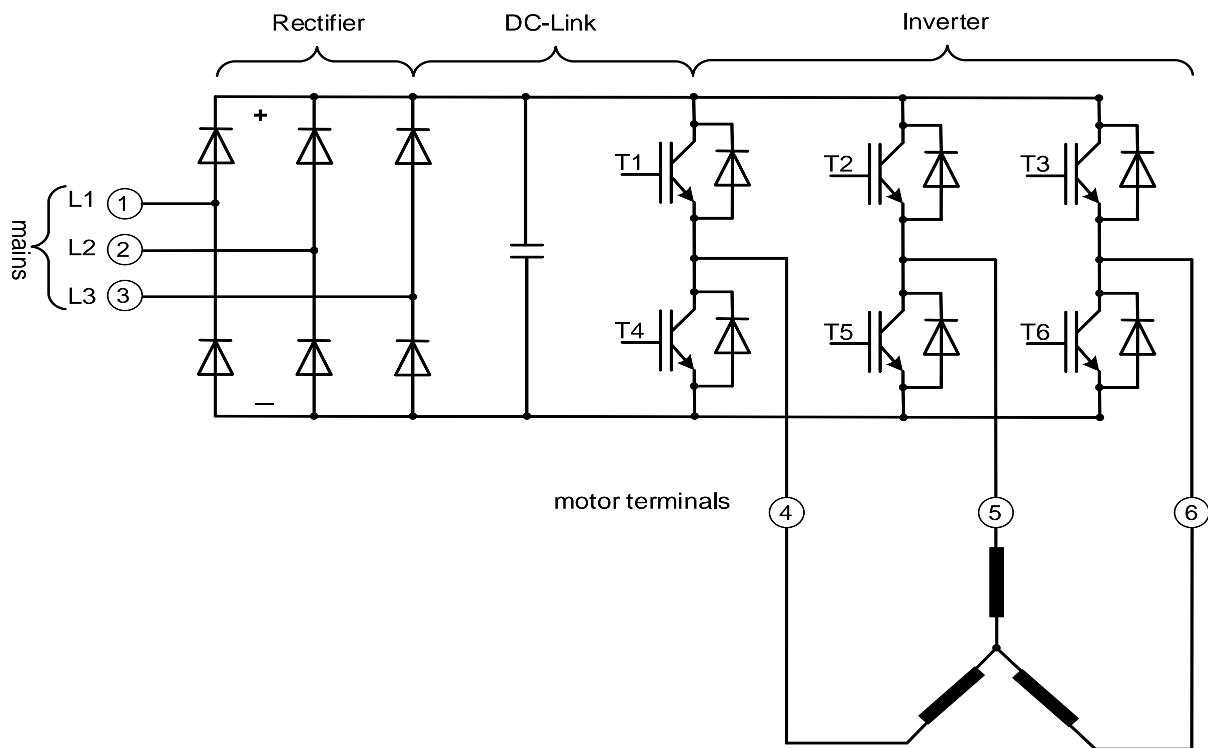


**Figure 17: Power electronics of a power module**

The working principle of asynchronous motors is almost same as to that of synchronous motors except that it has no external exciter connected to it. Asynchronous motors also known as induction motors which also run by the principle of electromagnetic induction. In asynchronous motors there is no external device connected to excite the rotor in asynchronous motors and hence, the rotor speed depends on the varying magnetic induction. This varying electromagnetic field causes the rotor to rotate at a speed lower than that of the

stator's magnetic field. Since the speed of rotor and the speed of stator's magnetic field vary these motors are known as asynchronous motors. The motor is generally connected to a work machine via a rotating shaft with a clutch. Using the mechanical energy the motor performs application-specific tasks, e.g. the material transport (conveyor belt).

## 3.3 Exemplary Faults in MCS

A comprehensive list of software and analog or digital hardware faults that can occur in motion control systems and their respective effects can be found together with selected error scenarios in motion control applications in [Ref]. We list some of them as examples in this section.

### 3.3.1 Exemplary Faults in the Analog Hardware

The analog components of a motion control system as shown in Figure 16 are power electronics, motor and encoder.

- One broken input terminal of the motor
    - Fault ID: HW_Analog_001
    - There is no input voltage at exactly one at the input terminals of the motor
    - Reduced power and unbalanced network load
- Two broken input terminals of the motor
    - Fault ID: HW_Analog_002
    - There is no input voltage at exactly two at the input terminals of the motor
    - Drive failure

- A diode to the positive pole is short-circuited in the mains rectifier of the PM power section.
    - Fault ID: HW_Analog_003
    - Intermediate circuit voltage breaks
    - Drive failure
- The intermediate circuit capacitor of the PM power section is short-circuited.

  - Fault ID: HW_Analog_004
  - No smoothing of the DC voltage generated by the mains rectifier
  - Drive failure
- In the inverter of the PM power section, an IGBT is short-circuited.
  - Fault ID: HW_Analog_005
  - Intermediate circuit voltage breaks
  - Drive failure
- Sensor defective.
  - Fault ID: HW_Analog_006
  - No or incorrect actual values are passed to the pulse logic for control and monitoring
  - drive failure
- Short circuit between the stator windings of different voltage phases in the asynchronous motor.
  - Fault ID: HW_Analog_007
  - An overcurrent is generated in the motor
  - drive failure

### 3.3.2 Exemplary Faults in Mixed-Digital/Analog Hardware

The digital components of a motion control system (including sigma-delta converters) as shown in Figure 16 are Pulse logic with control and monitoring, and microcontroller hardware.

- Hardware error in the control unit or in pulse logic.
  - Fault ID: **HW_AMS_001**
  - The PM performance part receives incorrect control signals from the pulse logic
  - Drive failure
- No clock signal at the output of the sigma-delta converter for the intermediate circuit voltage.

    - Fault ID: **HW_AMS_002**
    - No digitization and therefore no measurement of the DC link voltage possible
    - Drive failure
- Incorrect clock frequency at the output of the sigma-delta converter with at least 1 phase of the 3-phase inverter output voltage.
    - Fault ID: **HW_AMS_003**
    - Error measurement of the voltage of a phase
    - Drive failure
- Faulty bit stream at the output of the sigma-delta converter with at least one phase of the 3-phase inverter output voltage.
    - Fault ID: **HW_AMS_004**
    - Error measurement of the voltage of a phase
    - Drive failure
- The pulse logic generates a short turn-on or switch-off pulse for the inverter IGBTs.
    - Fault ID: **HW_AMS_005**
    - This leads to overheating and destruction of the affected IGBTs
    - Drive failure
- The pulse logic generates a short circuit with each switching of at least one of the three phases in the PM inverter (for e.g., from 1 to 0).
    - Fault ID: **HW_AMS_006**
    - Drive failure

### 3.3.3 Exemplary Faults in the Communication Bus

The data exchange between the Control Unit (CU) and the Power Module (PM) takes place via the communication bus (K-Bus) (see Figure 16).

- K-Bus cable defective or interrupted.
    - Fault ID: **K_BUS_001**
    - No data exchange between CU and PM is possible
    - Drive failure

- K-Bus cable is too long.
    - Fault ID: **K_BUS_002**
    - Sporadic errors can occur during the data exchange between CU and PM
    - Drive failure is possible
- The CU parameterization does not match the PM parameterization.
    - Fault ID: **K_BUS_003**
    - No data exchange between CU and PM is possible
    - Drive failure

### 3.3.4 Exemplary Faults in the Software

With the software (SW) of the Control Unit (CU), the microcontroller (μC) can calculate the set point values for the pulse logic (see Figure 16).

- Software does not correctly support the operating system.
    - Fault ID: Software_001
    - Application is not executable
    - Drive failure
- Faulty command processing due to a bit-dumper in the command memory.
    - Fault ID: Software_002
    - Application crashes
    - Drive failure
- Software processes unauthorized data.
    - Fault ID: Software_003
    - Application crashes
    - Drive failure

### 3.3.5 Exemplary Faults in the Application

For the execution of the tasks from applications, the work machines are necessary to perform the mechanical work. Examples of work machines are conveyor belts, fans, pumps, printing

machines, industrial robots, crane systems, wind power plants, etc. The application module is shown in Figure 16. We list some example fault scenarios in conveyor belts.

- Conveyor belt is accelerated too much or decelerated.
    - Fault ID: **Appl_001**
    - Material can fall from the conveyor belt
- Product jam on the conveyor belt.
    - Fault ID: **Appl_002**
    - Further processing of the product by subsequent processes is endangered
- Conveyor belt blocked.
    - Fault ID: **Appl_003**
    - Drive overload
    - Shutdown is necessary

### 3.3.5 Hardware Induced Software Errors

Hardware failures can modify software and/or induce totally unpredictable results in the software. While the failure mechanism is within the hardware, the software is erroneous because it has new and unintended reaction. For example, a hardware bit perturbation can result in an incorrect instruction or data, or a jump to an incorrect memory location. We list some example fault scenarios.

- Exposure of microcontroller to cosmic radiation
    - Fault ID: **HW_SW_001**
    - one or more bits of program code or HW registers or from RAM or Flash are set incorrectly, software registers are changed
    - slight deviations from the nominal behavior or in the case no error impact
    - massive deviations from the desired behavior with harmful potential or SW crash
- Hardware part of a sensor in the system is defect
    - Fault ID: **HW_SW_002**
    - It may induce an unintended reaction from the application software

ISO26262 explicitly mentions possibilities for fault effect simulation as an effective measure for the achievement of safety targets: Clause 5: 10.4.5 "Fault injection testing aims at introducing faults in the hardware product and analysing the response. This testing is appropriate whenever a safety mechanism is defined. Model based fault injection (e.g. fault injection done at the gate-level netlist level) is also applicable, especially when fault injection testing is very difficult to do at the hardware product level. For example, showing the response of safety mechanisms to transient faults inside hardware parts, such as a microcontroller, is very difficult to do with fault insertion at the hardware product level since it would require irradiation tests."

## 3.4 Fault Injection using Virtual Prototypes

A virtual prototype (VP) is a fast simulation model of the digital hardware e.g., control unit (CU). This model enables fast simulation, while being able to execute exactly the same binary software as the target CU. Because a virtual prototype is a software simulation model which helps to address the previously issues by providing the following advantages [89].

- full access to internal and external hardware elements (that have been modeled), as well as software,
- high observability i.e., all hardware and software events recorded and correlated,
- high controllability i.e., faults can be triggered by software, hardware or time events
- complete repeatability i.e., simulations are completely deterministic and
- faults reside in the simulation framework and do not go into release code

## 3.5 Aim of the Research and its Objectives

The aim of this work is to propose and develop methodologies for efficient fault simulation for industrial electronic systems consisting of system parts of heterogeneous domains (digital/analog/mechanical) using virtual prototypes and also in the context of hardware-in-

the-loop system. The aim of the research was further decomposed into the following objectives:

- Creation of concepts for the efficient simulation of digital, analogous electrical and mechanical system components, which enable fault simulation for real systems.
- Integration of heterogeneous system parts in to a virtual platform (multi-domain virtual platform).
- Integration of fault injection into analog and mixed-signal components in to a virtual platform and also in the context of hardware-in-the-loop simulations.
- Prepare a virtual and hardware-in-the-loop prototype systems for demonstration

## 3.6 Drawbacks of Existing Solutions and Research Methodology

The system qualification has a special role to address possible errors constructively and proactively in the development of future industrial electronic systems. The early detection and correction of the errors is of central importance. However, early error detection is not possible without the detailed consideration of all components along the entire value chain. Fault simulation techniques are today established at a very low level of abstraction or in the context of real hardware test setups. The consideration on a low level of abstraction does not allow a comprehensive system view and also test setups are only available in late development phases which involve considerable effort. Hence, an early and comprehensive consideration of all relevant failure scenarios is not feasible.

Virtual prototyping in the industry has for some time received the greatest attention and acceptance. Meanwhile in the system development virtual representations of hardware are increasingly used. On the one hand these virtual models, in contrast to real hardware, provide better insight into the system, while simulating a variety of scenarios efficiently. On the other hand one of the concerning things about virtual prototyping has been the separation between speed and accuracy. Therefore, our goal is to raise the existing fault simulation techniques to the virtual system-level without compromising on accuracy i.e., accurate simulation of faults on abstract system models. Hence, for quality assessment, this includes the modeling of faults in abstract heterogeneous component models as well as the assessment of the impact of

disturbances on the system in order to identify faults and deficiencies in the virtual prototype early in the design process and to prevent them.

# 4 Multi-Domain Fault Modeling and Simulation

## 4.1 Introduction

Future industrial electronic systems (IES) have to cope with a wide spectrum of applications, which demand the design of much more complex and trustworthy systems. As a result of growing complexity and more stringent requirements, early design space exploration and full-system simulation is crucial for these systems with traditionally long development cycles and rush market demands. IES are of multi-domain nature which in general consists of digital electronic components with embedded software (SW) on single or multiple processor cores

along with analog electronics, mechanical components and the environment. The interaction between these heterogeneous subsystems becomes very crucial. Multi-domain simulation is the ability to efficiently and accurately simulate systems of different domains (e.g., thermal, electrical, mechanical, hardware/software, etc.) together within one simulation [7].

Considering the complexity of IES, simulations are preferable to pure mathematical analysis. Engineering groups of different domains in industry exploit different domain languages and tools to model and evaluate their designs, which may not be familiar to everyone. It is error-prone and inefficient to work with these domain languages and tools directly to design and evaluate overall systems. Given the multi-domain nature of IES, it is more appropriate either to bring different domains into one simulation [90] or use a heterogeneous simulation environment (co-simulation) [91] [92] to study system dynamics. We propose to bring together different domains into one simulation platform in a closed loop. Moreover, these systems are required to properly cope with failures of all kinds to guarantee safety of operators and machine integrity at any time. As of today, the final tests of those systems are mainly based on physical prototypes to ensure the correct and safe operation of the system. In conventional system development, physical prototypes and extensive system tests are needed. However, those prototypes are available only in later phases of the design process. These tests are complex and expensive already today and are not able to completely cover all possible kinds of failures, as certain failures cannot be provoked in real hardware. Moreover, the late execution of the tests may cause long iteration loops in case weaknesses are detected in the final tests. We propose a virtual stress methodology based on virtual prototypes to validate the functional safety of motion control systems. Using this methodology it will be possible to carry out system tests at an early stage and more importantly it is possible to protect the reaction of motion control systems against faults early in the development and also against faults which could hardly be provoked in real hardware setups.

We identified the following tasks in order to successfully carry out virtual stress tests, which will be discussed in the remainder of this chapter.

- Fault scenario identification and fault specification
- Identifying critical system states

- Analog and mixed-signal fault modeling
- Digital fault modeling
- Integration of heterogeneous system parts in a virtual platform
- Infrastructure/generic framework for fault injection in a virtual platform
- Automated fault injection tests and post processing

## 4.2 Virtual Stress Tests

To perform virtual stress tests i.e., to analyse the behaviour of a system under the influence of faults, a model of the system is required which describes for each fault its impact on system, the time at which the fault occurs (fault activation time) and how long it lasts (fault duration). Hence as a prerequisite it is needed to list the possible fault scenarios which are of interest on system behaviour level.

### 4.2.1 Fault Scenario Identification and Fault Specification

| System component | Name of the system component |
|---|---|
| Function | Functionality of the system component |
| Abstraction | Abstraction level at which the fault is modeled |
| Fault | Abnormal condition that can cause an element or an item to fail [54] |
| Error | Discrepancy between a computed, observed or measured value or condition, and the true, specified or theoretically correct value or condition [54] |
| Failure | Termination of the ability of an element to perform a function as |

| | |
|---|---|
| | required [54] |
| Impact | Effect on system functionality. Possible values; high, low, process dependent |
| Comments | Additional information for user |

**Figure 18: Fault Scenario Template**

A widely used method in industry to identify all possible failures in a system is failure mode and effects analysis (FMEA) [56] [57]. We use the FMEA reports to extract fault scenarios and derive their corresponding fault specifications which aid in fault modeling. We define the template shown in Figure 18 to list the possible fault scenarios in a system and the template for derived fault specification is shown in Figure 19.

| **Fault ID** | **Unique identifier** |
|---|---|
| Abstraction | Transaction level, signal level |
| Modeling | SystemC, SystemC-AMS, Simulink, ... |
| Fault model | Fault model e.g. open-circuit, short-circuit, stuck-at, value-drift, delay, ... |
| Fault type | Fault type based on timing behaviour: transient, permanent... |
| Time-window (Period) | Time interval in which the fault occurs at least once. Not relevant for permanent faults |
| Duration | Fault active time in one time window (duration, period). Not relevant for permanent faults |

| Comments | Additional information for user |
|---|---|

**Figure 19: Fault Specification Template**

| **System component** | **Variable Frequency Drive (VFD)** |
|---|---|
| Function | 3-phase motor control using AC voltage dependent on frequency and amplitude |
| Abstraction | System level; Simulink or SystemC-AMS |
| Fault | Broken-wire fault on one of the three input phases |
| Error | No input (value) on one of the three input phases |
| Failure | Reduced performance and unsymmetrical network load |
| Impact | Process dependent |
| Comments | VFD converts from a 3-phase AC voltage at the input to the frequency and amplitude adjustable 3-phase AC voltage at the output, e.g. for the operation of three-phase motors |

**Figure 20: Fault Scenario Example**

| **Fault** | **Broken-wire fault on one of the three input phases** |
|---|---|
| Fault ID | HW_Analog_0001 |
| Abstraction | Signal level |
| Modeling | Simulink |
| Fault model | Open-circuit |

| Fault type | Permanent |
|---|---|
| Time-window | Not applicable |
| Duration | Not applicable |
| Comments | The motor operation depends on load when one of the 3-Phase input lines is broken |

**Figure 21: Fault Specification Example**

An industrial electronic system, e.g. motor control application is a typical example of multi-domain system, which in general consist of components of multiple domains, i.e. digital electronic components with embedded software (SW) running on one or multiple processor cores as well as analog electronics, mechanical components and the environment. Various numbers of faults can occur on these multi-domain system components. An example fault scenario of broken-wire fault on one of the three input phase lines and its corresponding fault specification are shown in Figure 20 and Figure 21 respectively.

### 4.2.2 Identifying Critical System States

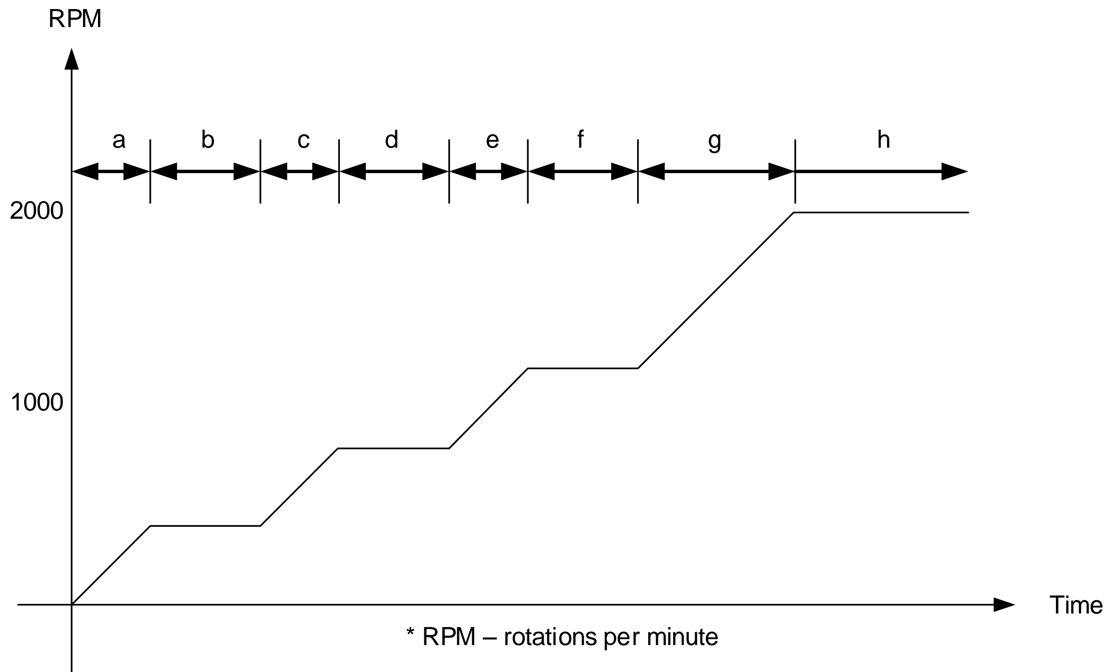


**Figure 22: Speed Profile of a Motor**

Another step necessary for conducting fault injection campaigns is identifying the critical states of a system, which helps to extract the fault activation times for an application use case. It is a challenging task as each output system state is dependent on various factors, e.g., application use case, environment, system parameters etc. and for complex systems it is hard to consider all the system use cases at once. Hence, it is important to develop a framework which is independent of this step in order to use the same infrastructure with minimal modifications for different use cases. An example speed profile of a motor is as shown in Figure 22. As seen the motor output has three different states ramping or transient (a, c, e and g), and steady (b, d, f and h). We can extract fault activation times from this speed profile, which helps in conducting fault injection campaigns.

As strong inter-dependencies between components across domains exist in multi-domain systems, possibilities to combine different faults within and across domains are necessary to validate the system. Hence, it is important that the infrastructure for fault

injection should provide generic mechanisms for the users to not only activate simple predefined faults but also to combine multiple faults in order to make fault injection campaigns possible in multi-domain systems.

## 4.3 Analog and Mixed-Signal Fault Modeling

Virtual prototypes based on purely digital models and model descriptions may not offer an efficient way to capture analog behaviour, which is often an integral part of the embedded system. The fault model complexity in analog and mixed-signal circuits is different from that in digital circuits. In digital circuits, the stuck-at fault model is widely used at gate level [93]. In contrast, in analog circuits accurate analog fault models are not always available. Also, describing the good and faulty circuit for all types of faults at higher levels such as the behaviour or the macro-model level is a very complicated task and still remains a challenge in analog circuit testing. Several fault models at different abstraction levels are proposed. Furthermore, probability methods are often not efficient because the statistical distributions of analog faults, generally, are not known with enough precision to accurately predict the fault coverage of a test set. The information provided in the literatures can be used for making test decisions, creating fault models, generating fault lists, and calculating fault coverage in fault simulation. A comprehensive structured approach for testing and fault diagnosis of analog and mixed-signal (AMS) circuits and systems have not yet materialised. The basic problem with analog IC fault diagnosis is the absence of efficient fault models [94], component tolerances and non-linearities. It is difficult to arrive at a general fault model like the stuck-at models for the digital circuits.

Different failures modes exist in analog domain such as degraded performance, functional failures, open circuit, short circuit etc. Degradation faults depend mainly on variations of certain parameters of the components used in a circuit from its nominal values. This may be due to manufacturing defects, process variations, change in the environment or ambient temperature and/or wear out due to aging. Functional faults, on the other hand, are based on the fact that a circuit may continue to function, but some of its performance specifications may lie outside their acceptable ranges.

We can broadly categorize the types of faults occur in analog circuits as parametric and catastrophic faults. Parametric faults occur in circuit due to some manufacturing defects (change in some parameter like due to doping level and due to oxide thickness). Due to parametric faults in circuit the tolerance of component will vary to certain value. In these types of faults the circuit output may or may not be changed. Because the value of component will increase or decrease to certain value. This type of fault can be removed with the help of knowing the tolerance of component. Catastrophic faults will completely change the output of the circuit as these faults which eventually will lead to a short or an open circuit. These are also called hard faults. Due to these types of faults the behaviour of system changes drastically. Parametric faults are more realistic but the list of faults to simulate is nearly impossible. Hence it is necessary to identify a set of faults to simulate before preparing the models [95]. In most of the studies, the faults are modeled mostly as open, short, and variable component values. However, component value changes are usually significant in these failure modes. As a result, a faulty value with a value ten times larger or ten times smaller is a reasonable assumption in generating the fault list. Open and short faults are only the extreme cases of these two. Therefore, it is also important to test these variations (parametric faults) along with open and short faults.

In our study, we have mainly chosen to use open and short fault models for analog fault modeling as most defects seen in manufactured ICs are shorts and opens (except in the most advanced process nodes). Also, in the state-of-the-art production processes, shorts are 3-10X more likely than opens [96]. Additionally, 80 to 90 percent of analog faults involve shorted and open in electrical components such as capacitors, diodes, transistors [62]. Our goal is to simulate faults in analog/mixed-signal abstract component models at system level.

In analog models, short circuit and open circuit should be considered as resistive values according to the technology and process [97]. Open faults are hard faults in which the component terminals are out of contact with the rest of the circuit creating a high resistance at the incidence of fault in the circuit. Addition of a high resistance in series with the component (e.g., capacitor or diode) can simulate the open faults. Short faults, on the other hand, are a short between terminals of the component (effectively shorting out the component from the circuit). A small resistor in parallel with the component can simulate this type of fault for the

component. Parametric faults can also be simulated by having an appropriate deviated resistance value. In this study we have chosen the fault models of devices such as capacitor and transistor as shown in Figure 23. The capacitor has one resistor in series (Rs) and one in parallel (Rp) in order to simulate open and short circuits respectively. Similarly a transistor has 6 resistors.

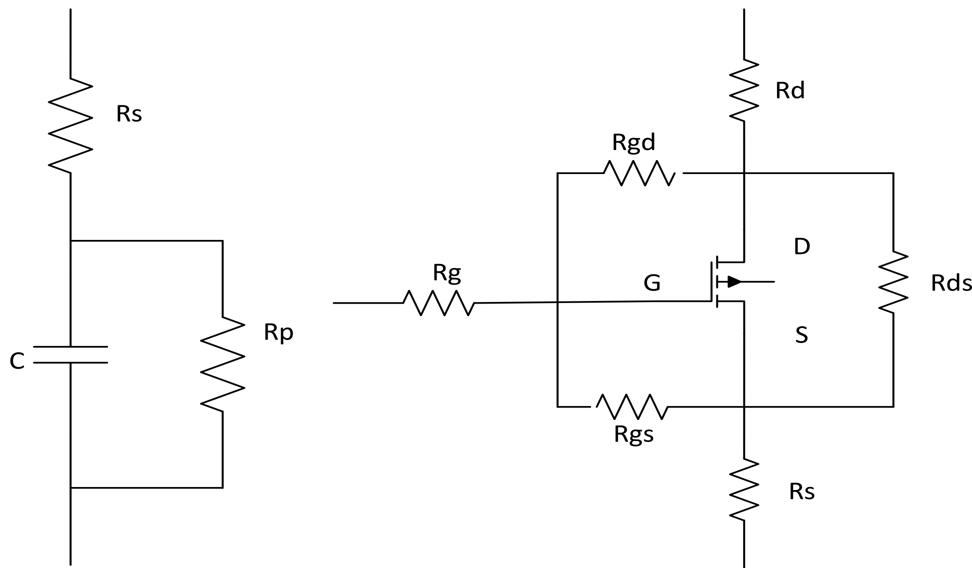


**Figure 23: Fault models of capacitor and transistor**

### 4.3.1 SystemC-AMS and MATLAB/Simulink

One of the important concepts of fault diagnosis in analog ICs is the use of model-based observer scheme. Most popular approaches to model analog and mixed-signal components on system behavioural level are using the SystemC language [98] along with the analog and mixed-signal (AMS) extension (SystemC-AMS [9]) and MATLAB/Simulink [8], which has become a de-facto standard in industry. SystemC-AMS extensions introduce three different models of computation to support AMS behavioural modeling at different levels of abstraction. It offers various advantages such as interportability, ease of integration into virtual prototypes, open-source, etc. On the other hand, MATLAB/Simulink is a good candidate for simulating behavioural macromodels as it provides various libraries and toolboxes for the modeling flexibility along with code generation and integration facilities. It also supports physical component and electrical power systems modeling with Simscape and

Simscape Power Systems (formerly SimPowerSystems) toolboxes respectively. Also, recently introduced HDL verifier [34] toolbox supports SystemC/TLM component generation for virtual platform development.

Both MATLAB/Simulink and SystemC-AMS have their own strengths. This dissertation focuses on MATLAB/Simulink models as it has become the de-facto standard in industry for modeling and simulation of physical systems and the weighted factor being the most common modeling tool used at Siemens (i.e., model availability).

### 4.3.2 Abstract Fault Models and Fault Interface Design

The aim of test generation (fault injection) is to minimize production tests and improve test quality by choosing an optimal set of test patterns. This task is well understood for digital circuits, for which automatic test pattern generators (ATPGs) assume a fault model (stuck-at, stuck-open, delay faults, etc.) and generate tests based on these. However because of the complex nature of analog circuits, a direct application of digital fault models proves to be inadequate in capturing the faults behaviour. Hence analog test selection has to be approached in a rather ad-hoc way. Sometimes circuits tend to be over tested to avoid shipping a faulty product, while, at other times, the tests may be inadequate and moreover we look at the system on behaviour level. The first step towards developing an analog testing methodology is to develop comprehensive analog fault models. The proposed standard analog fault model (AFM) for behaviour models is shown in Figure 24.

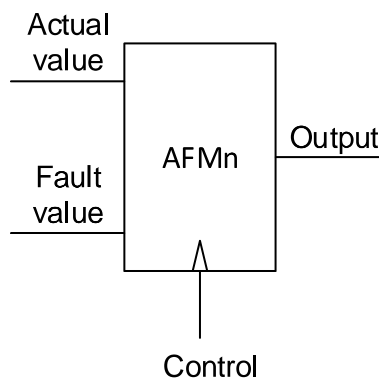


**Figure 24: Analog Fault Model (AFM)**

Each AFM implements a failure mode by manipulating signals between blocks in a model. As a general remark valid for all AFMs presented is that at each time step when a fault is not activated the output value is equal to the non-faulty value (actual value) ; else the actual value assumes the value determined by the AFM (fault value). Some of the AFM's are listed below (1-3 AFM's are boolean and 8-10 are real or integer),

- AFM 1: Forces the actual value (boolean) to be "1"
- AFM 2: Forces the actual value (boolean) to be "0"
- AFM 3: Inverts the actual value (boolean)
- AFM 4: Forces the actual value (real/integer) to be "0"
- AFM 5: Forces the variable's actual value (real/integer) to a constant value, provided as a parameter
- AFM 6: Increases the variable's nominal value (real/integer) by an amount, provided as a percentage change
- AFM 7: Decrease the variable's nominal value (real/integer) by an amount, provided as a percentage change
- AFM 8: The actual value (real/integer) is multiplied by a fixed value, provided as a parameter, at each time step
- AFM 9: Makes the actual value (real/integer) drift away from the nominal value by a fixed amount, provided as a parameter, at each time step
- AFM 10: Makes the actual value (real/integer) to keep the nominal value – at the time step when the failure occurred – for as long as the failure is active

The temporal aspect (transient, intermittent, permanent) is also embedded in the AFM and it is controlled by the additional control signal as shown in Figure 25. The *time-window* and *duration* are multiples of simulation time-step of the fixed-point solver. For e.g., consider a Simulink model with simulation time-step of 100 us, the value of time-window field equals 10 and duration equals 5. The value of time-window is calculated as 10*100 us and duration as 5*100 us.

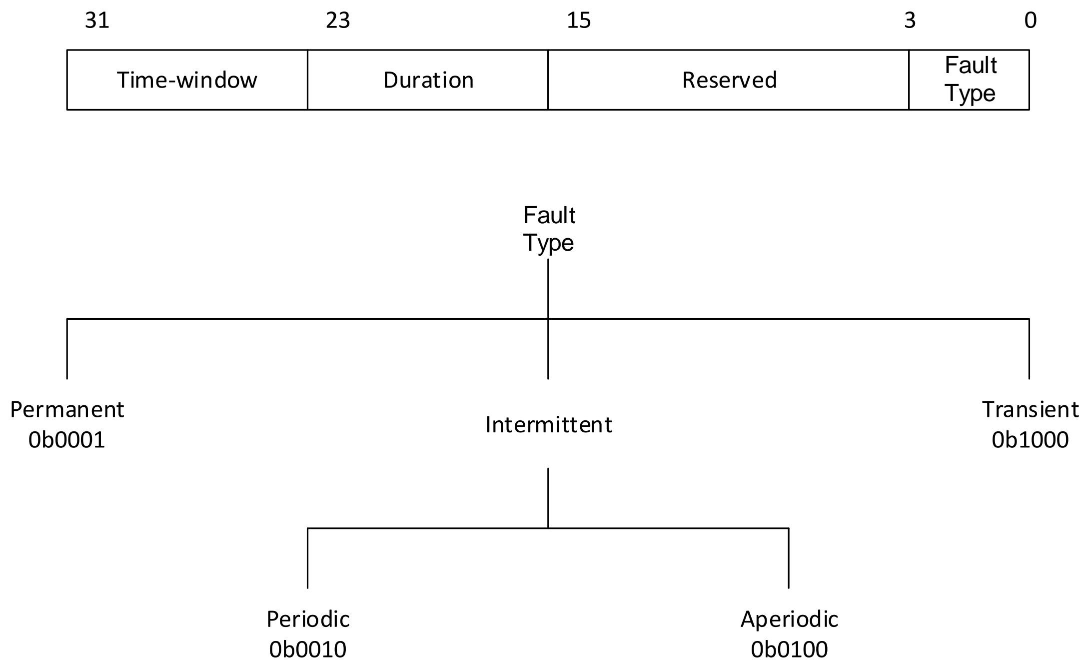


**Figure 25: Control Interface and Fault Type**

### 4.3.3 Fault Simulation in Abstract Models

Fault models are anomalies in the system and hence, they inherently cannot be accurately described with the means of standard electrical primitives. As technology continues to shrink, fault models are becoming more and more complicated. As the result, fault injection is becoming harder, less accurate and in some cases impossible to perform. Designers are usually forced to approximate fault behaviours to be able to add their effect to the circuit and in doing so, they would reduce precision. For a reliability researcher or engineer, it is very important to have a precise description of faults for an abstraction level i.e., abstraction level depends on use case and be able to rapidly model and implement new fault models as the design or technology evolve.

Abstract models of MATLAB/Simulink are based on signal-flow graphs. The bi-directional voltage-current relationship known from kirchoffian networks must be separated into input- and output only signals, where by voltages are considered as input and current as output. Electronic devices such as resistors cannot be directly attached to voltage ports; rather feedback loops are utilized to model the voltage drop correctly. For this reason, data flow

models are not appropriate for electronic design, whereas they are preferred environment for use cases such as system simulations, software validation, etc.

The amount of details to be incorporated inside the models will certainly depend on system use case. For use cases such as system simulation and software validation using virtual prototypes, it is not necessary to incorporate each and every physical effect of the electrical system as this will cost a lot of simulation performance (simulation time) but in turn has a drawback of being less accurate making it difficult to simulate fault effects accurately. Consider a motor control application system modeled on behaviour level using Simulink; it is difficult to simulate electrical faults like short-circuit between motor-phases, open-circuit (setting a phase value to ‘0’ might not have an open-phase effect), etc. Hence, a better approach to solve this issue is to capture the fault behaviour on right abstraction level and transfer it on to behaviour model, we call it as *fault transfer*. Although more accuracy to simulate faults would be possible in the simulation we choose Simscape electrical model to capture the fault behaviour as this would give enough insight on the abstraction level of the virtual prototypes (VP).

Why fault transfer?

- It is not possible to accurately simulate the faults such as electrical faults in abstract behavioural models.
- Although the electrical faults can be simulated in a virtual platform by integrating Simscape models using code generation feature, they are very slow for virtual prototyping use case (refer Simulation Performance Test).

In HIL (hardware-in-the-loop) systems, only the abstract model can be used due to stringent real time requirements.

## 4.4 Fault Transfer

We propose to go from an accurate simulation model to the abstract representation, which is fast and detailed enough to simulate the behaviour of the overall system. Once we have both the accurate and the abstract simulation models, the effect of a particular fault can be transferred by following these steps.

- Simulate the accurate model in presence of the fault and capture the behaviour.
- Transfer the fault behaviour onto the abstract model and simulate the overall system.

We use Simscape toolbox from Mathworks for electrical model simulation, which is closest to that of SPICE simulations (The variable-step solver, ode23t is closest to the solver that SPICE traditionally uses [99]). Incorporating every physical effect certainly will slow down the simulation speed. Hence, these Simscape models does not necessarily incorporate every physical aspect, but should be detailed enough to carry out simulations on system level. For abstract representation Simulink data-flow models are used. We simulate Simscape models to capture the fault behaviour, which are relatively accurate representation and transfer the fault effect onto an abstract representation (Simulink data-flow model).

### 4.4.1 Motor Control Application

The power electronics components (selected by the red dotted rectangle as shown in Figure 26) are modeled in both Simscape and Simulink.

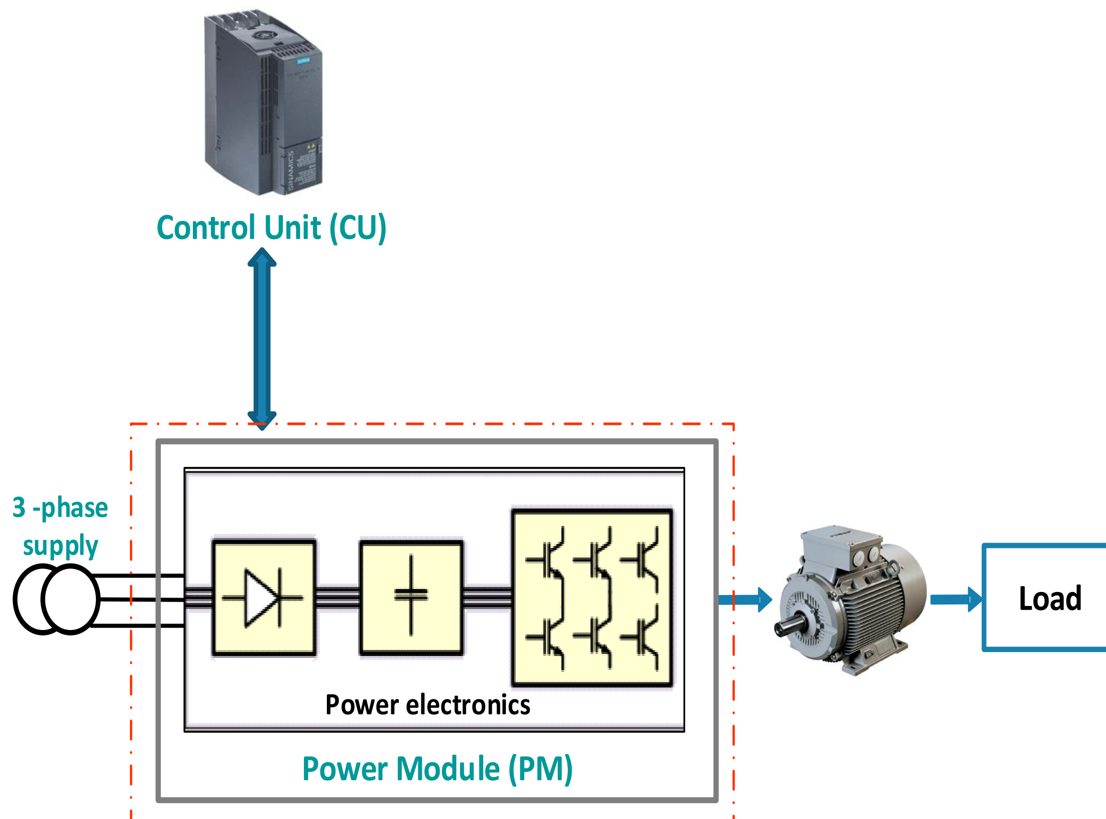

**Figure 26: Motor Control Application Setup**

### 4.4.1.1 Simulation and Results Analysis

Once we have both the Simscape and Simulink models, the initial step is to validate that both are functionally equivalent (without any fault injection). Figure 27 and Figure 28 show the simulation output with omega, torque and stator currents of Simscape and Simulink models respectively.

The comparison graph of motor speed is shown in Figure 29 respectively. From the speed comparison graph, we can clearly see that the signals in the graphs are overlapping only with the slight variations. These slight variations are due to differences in modeling artefacts in Simscape and Simulink. Hence we conclude from the simulation results that both the models are functionally equivalent.

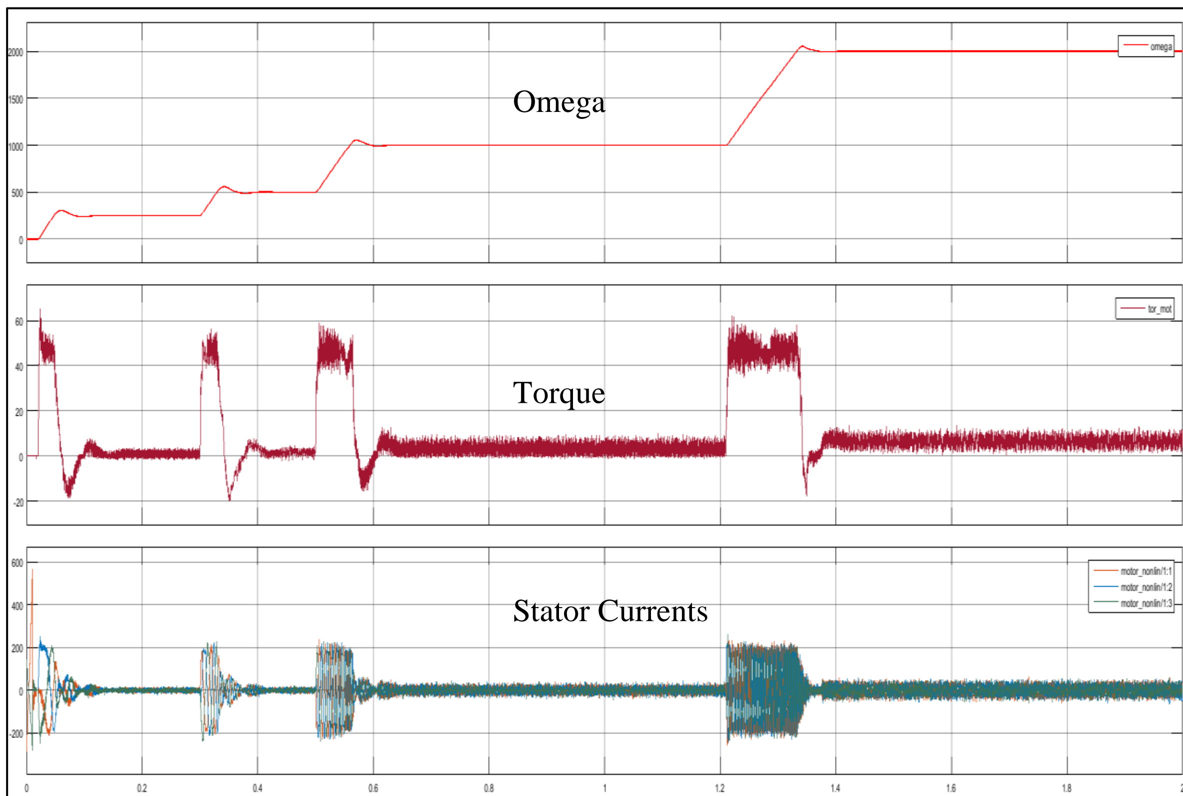

**Figure 27: Simulation Output (Simscape)**

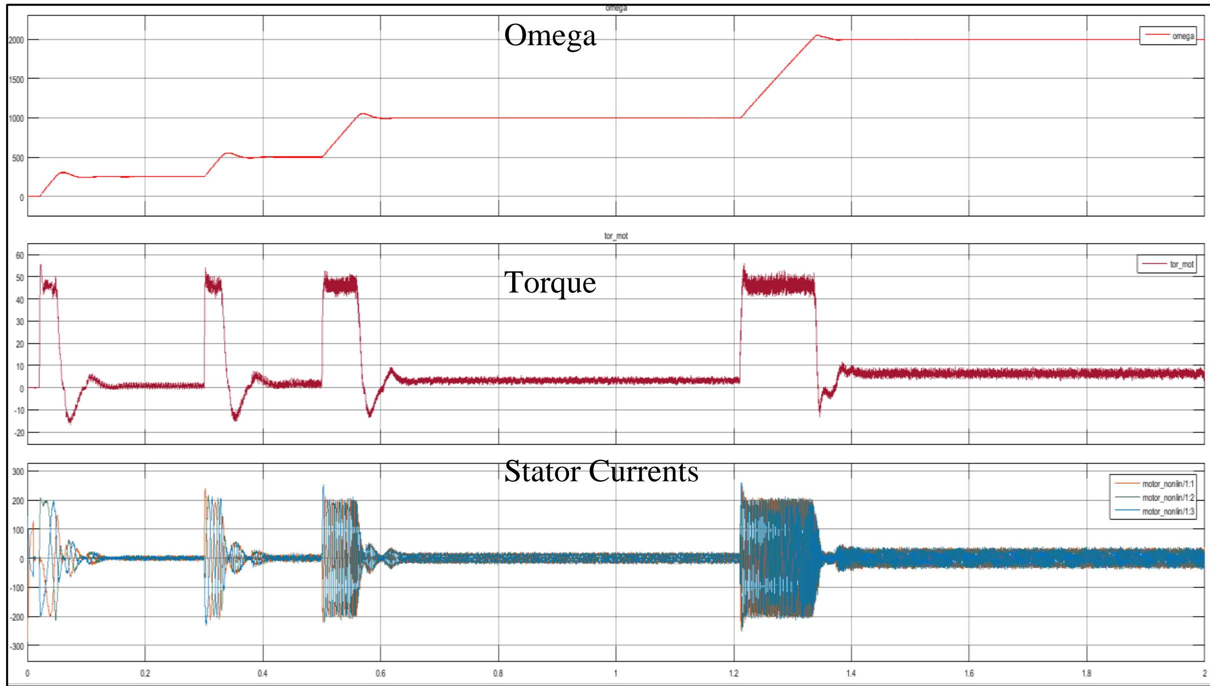


**Figure 28: Simulation Output (Simulink)**

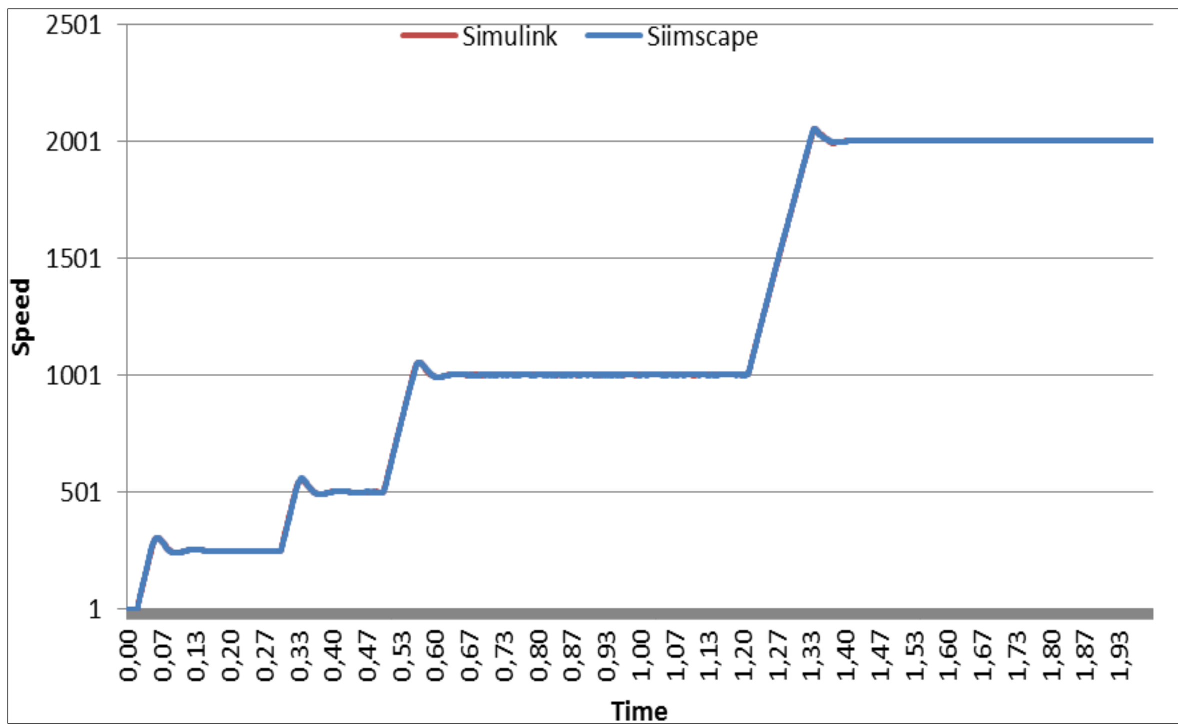

**Figure 29: Speed Comparison**

## 4.4.2 Examples of Fault Transfer

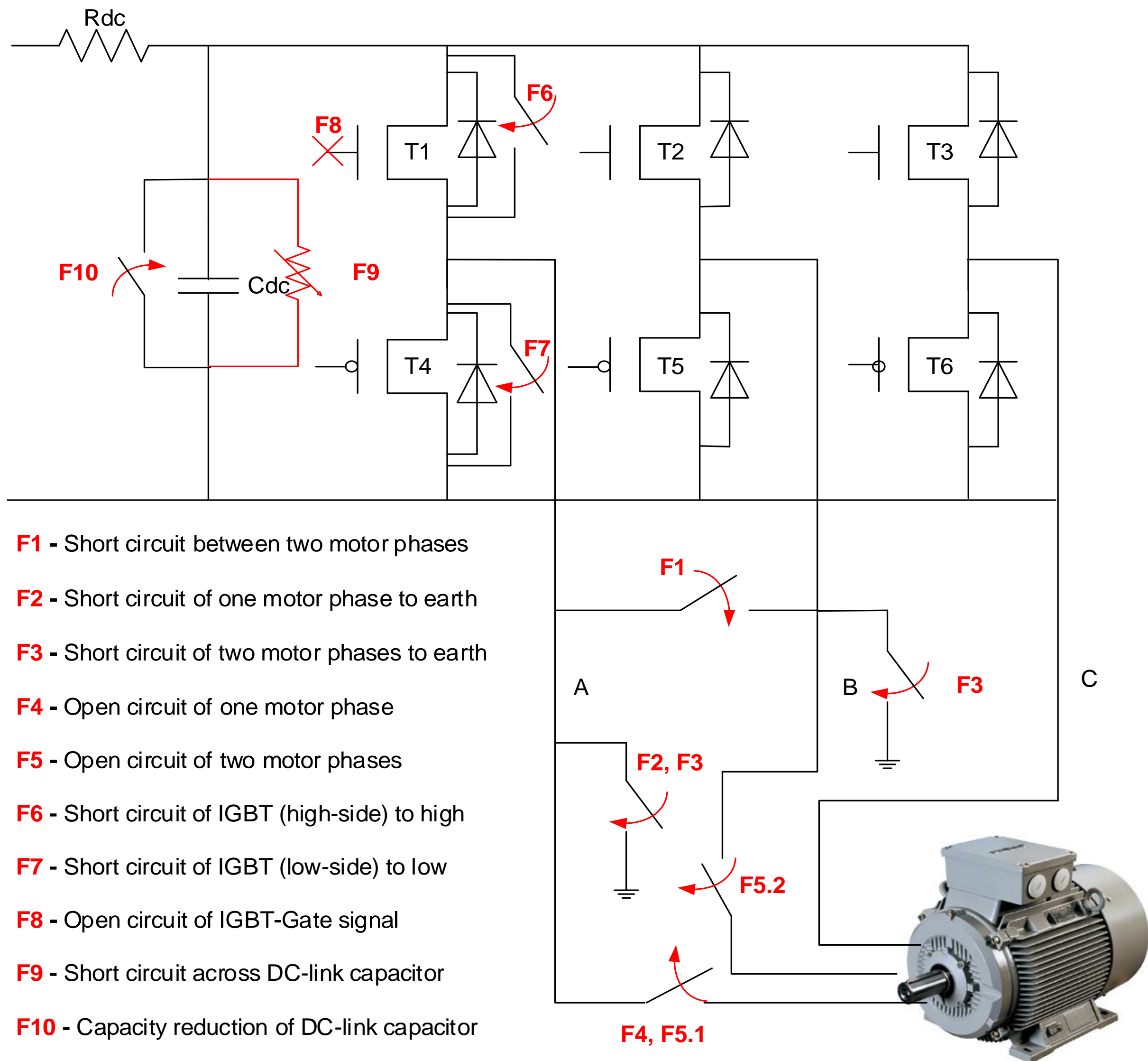


**Figure 30: Fault Selection in Inverter and Motor**

The most of electric motor drive system malfunctions deal with power converter failures, resulting in a significant decrease of a drive performance [84][85][86]. Along with DC-link capacitor faults, that make up more than 60% deviations, which are related to the power converters [87], IGBT transistor failures are the vast majority ones. These faults are mainly

caused by an ageing process, which is intensified due to a thermal stress of the transistors, when the inverter operates under wide range of environment temperature variations and a changeable load condition [88]. From two types of the transistor malfunctions, namely short and open-circuit faults, the first ones are more destructive for the power converters than the second ones because they can result in a high current flow through the DC-link circuit and a complementary transistor of the faulty converter phase, which leads to a capacitor destruction. We have selected various faults as shown in Figure 30 that could occur in power converter and also on motor phases for fault transfer process. In the electrical model the hard-faults like open or short are simulated using a switch (open or close) and variable resistor for simulating both hard-faults and parametric faults like capacitance reduction of dc-link capacitor.

In order to validate the fault transfer, it is necessary to consider different system output states as the behaviour of the system in presence of faults is dependent on system output state. We consider one of the transient (a, c, e and g) and one of the steady states (b, d, f and h) as examples from the motor speed profile shown in Figure 22

The effects of selected faults on motor speed output and stator currents are demonstrated in the experiments of this section. In order to prove the flexibility of the fault transfer methodology, we consider the following two cases assuming protection mechanisms i.e., fault remedial strategies (see Figure 100) might not be present or not completely developed,

- Protection mechanisms are not present
- Protection mechanisms are present

#### 4.4.2.1 Short circuit between two motor phases (F1)

The short circuit between motor phases fault is simulated by using a switch connected between them in electrical model as shown in Figure 30 whereas for data-flow model it is not straightforward as the input phase lines to the motor are not electrical. Hence, we propose to simulate the fault F1 in electrical model and extract the information on how the motor input phases are affected and model this behaviour in data-flow model.

### *Protection mechanisms are not present*

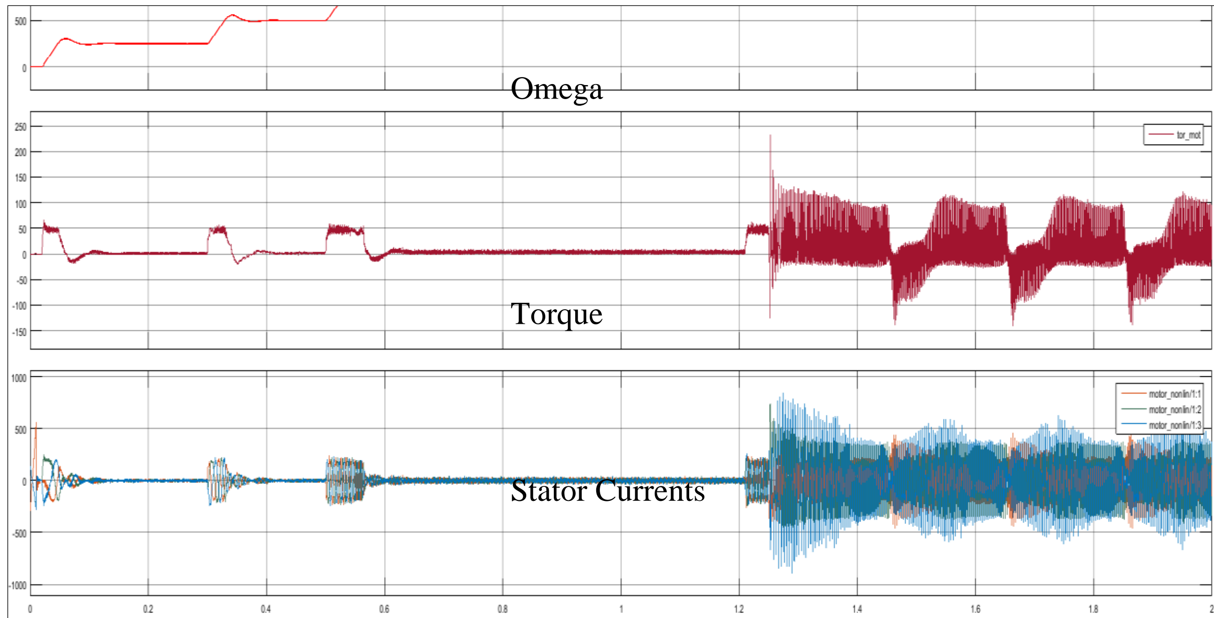


**Figure 31: Simulation Output (Transient State) – Simscape (F1)**

The simulation output during two-phase short circuit with no protection mechanisms is as shown in Figure 31. When the fault happens at t=1.25s (transient state), high transient stator currents are detected in motor (see Figure 32) and the motor is unstable.

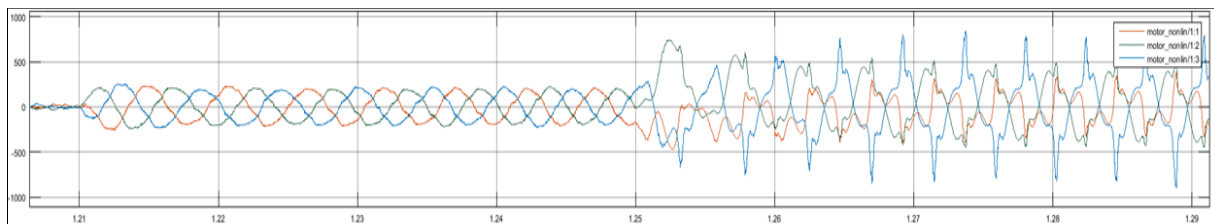

**Figure 32: Motor Currents Zoomed (Transient State) – Simscape (F1)**

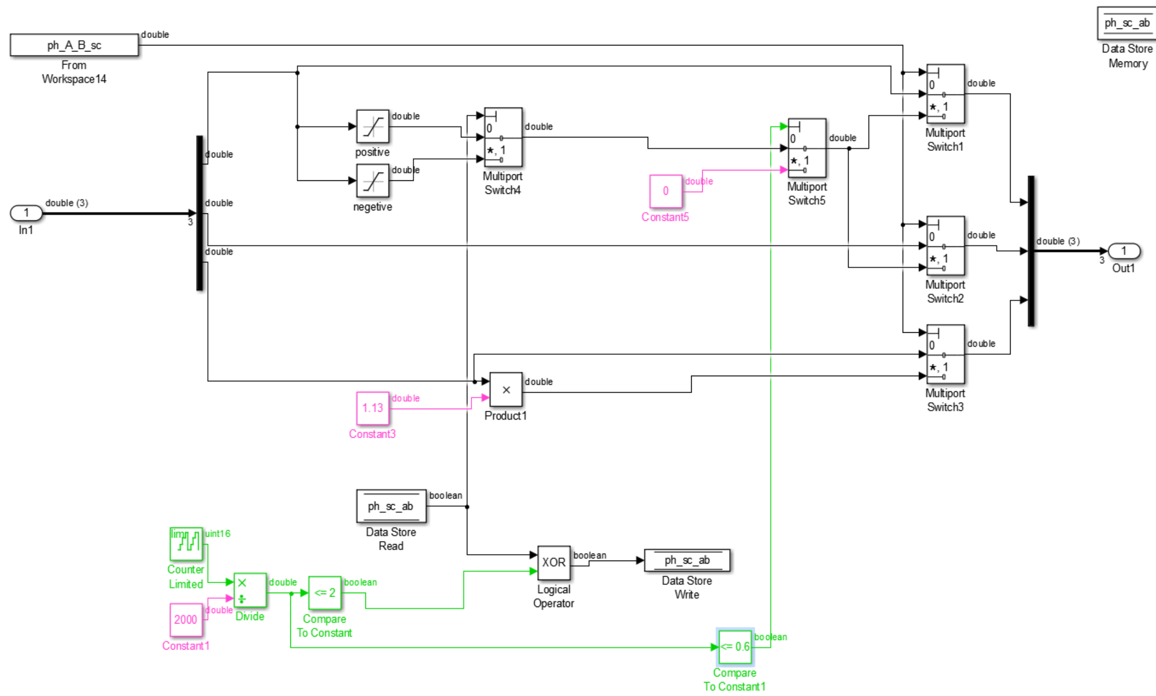


**Figure 33: Phase-to-Phase Short-Circuit Fault Model without Protection – Simulink (F1)**

The behaviour of input phase voltages with the fault is extracted manually and modeled in Simulink (data-flow) model as shown in Figure 33. Now the short circuit between motor phases fault is simulated in Simulink (data-flow) model at t=1.25s (transient state) as shown in Figure 34. The motor output speed comparison is shown in Figure 36, as clearly seen the motor behaviour is very similar and slight variations are due to modeling artefacts in Simscape and Simulink. The stator current signals (zoomed) of Simscape and Simulink models are shown in Figure 32 and Figure 35 respectively.

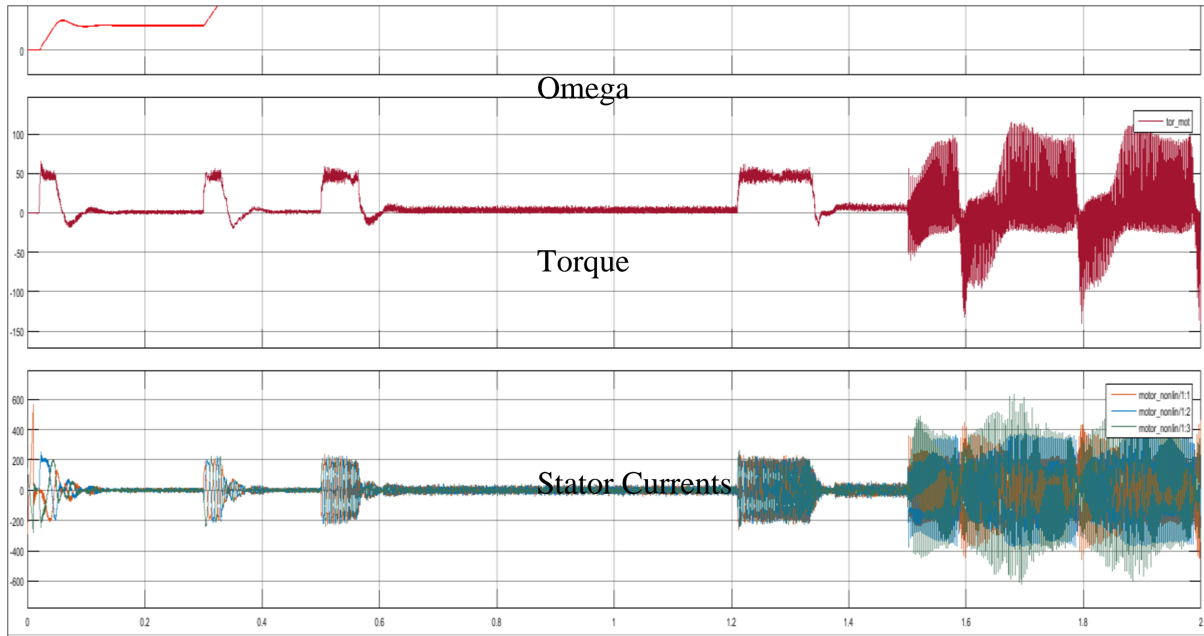


**Figure 34: Simulation Output (Transient State) - Simulink (F1)**

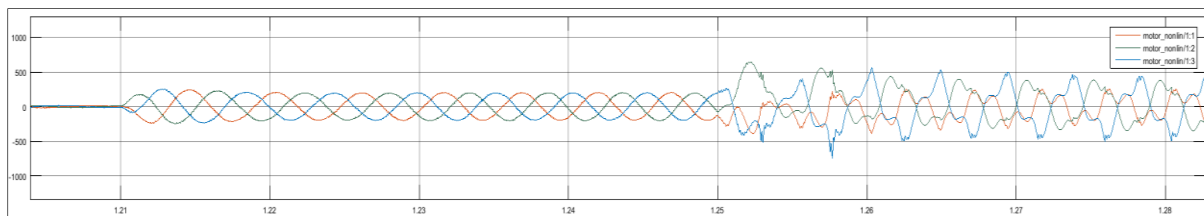

**Figure 35: Motor Currents Zoomed (Transient State) – Simulink (F1)**

Similarly, the short circuit between motor phases fault without protection mechanisms is simulated at t=1.5s (steady state) and the speed comparison graph of Simscape and Simulink is shown in Figure 37. The slight variations in speed curves are due to modeling artefacts in Simscape and Simulink. After the phase short circuit fault happens, motor is unstable and high currents are detected in the motor phase components (see Figure 38 and Figure 39).

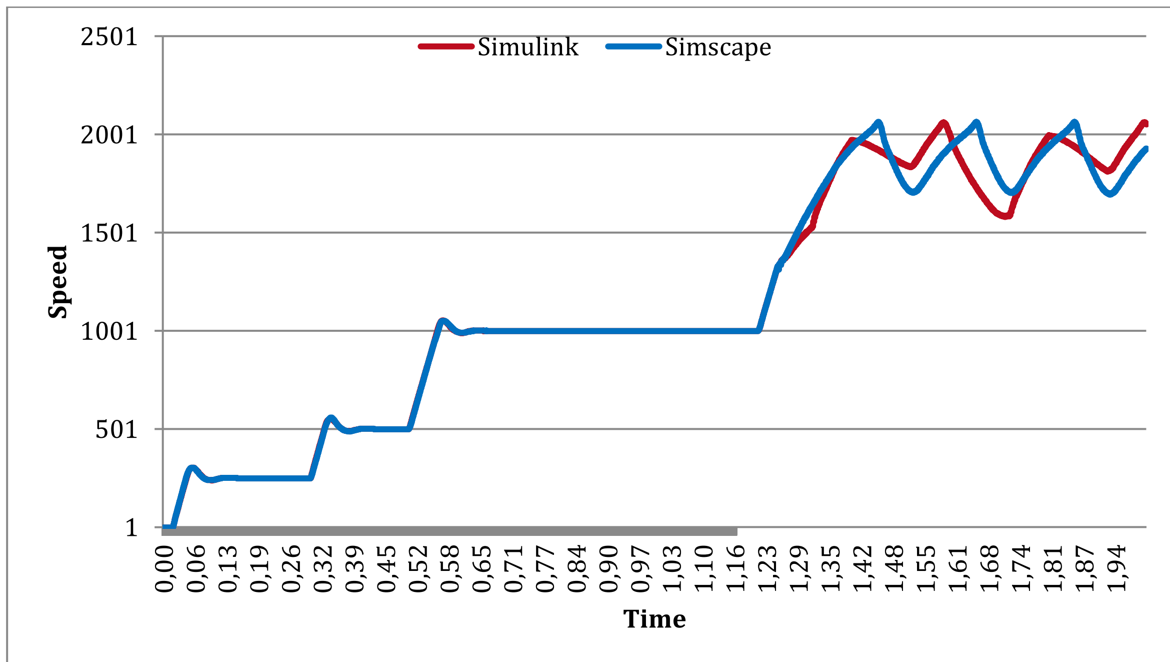


**Figure 36: Speed Comparison without Protection - Transient State (F1)**

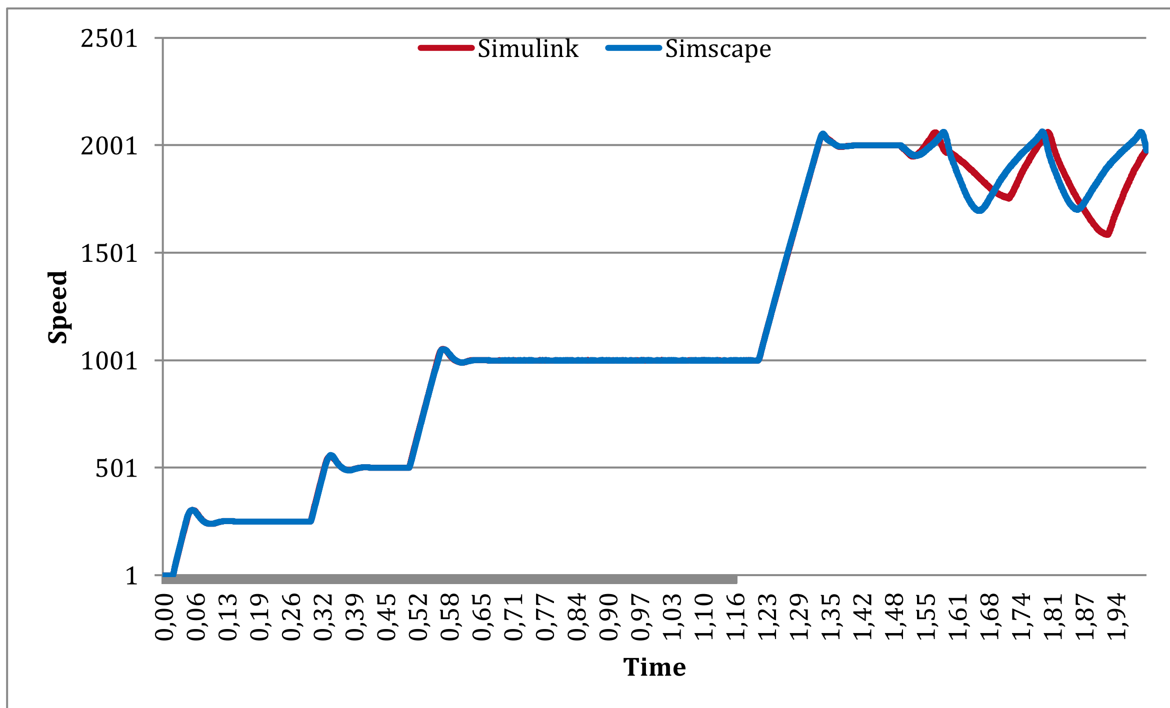


**Figure 37: Speed Comparison without Protection - Steady State (F1)**

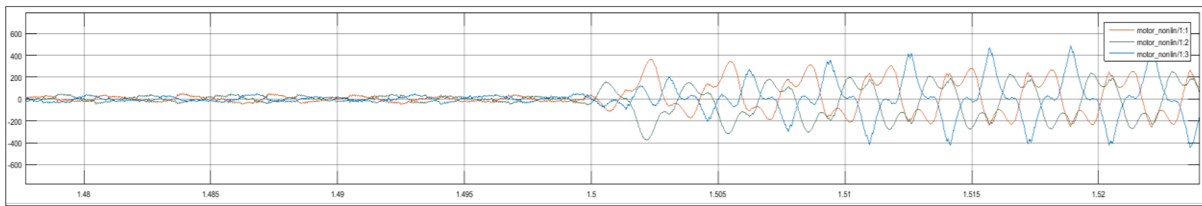


**Figure 38: Motor Currents Zoomed (Steady State) – Simscape (F1)**

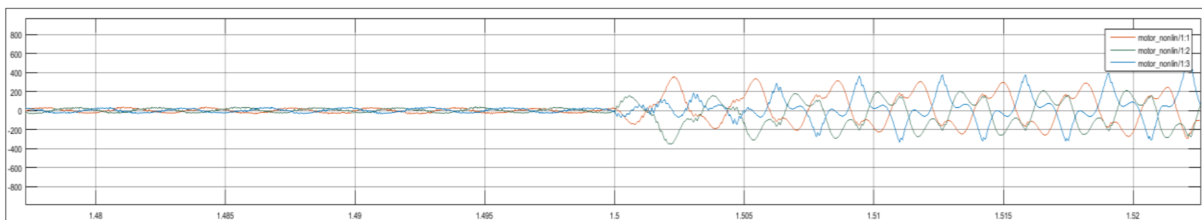


**Figure 39: Motor Currents Zoomed (Steady State) – Simulink (F1)**

### *Protection mechanisms are present*

In this test the two-phase short circuit with protection mechanisms is simulated and the behaviour of input phase voltages with the fault is extracted manually and modeled in Simulink (data-flow) model as shown in Figure 40. When the fault occurs at t=1.25s (transient state), high transient stator currents are detected in two of the motor currents and the current in the third phase is almost zero (see Figure 41). The hardware i.e., all six IGBT's are shutoff by the protection mechanism due to these high current components and the motor starts to decelerate abruptly towards zero.

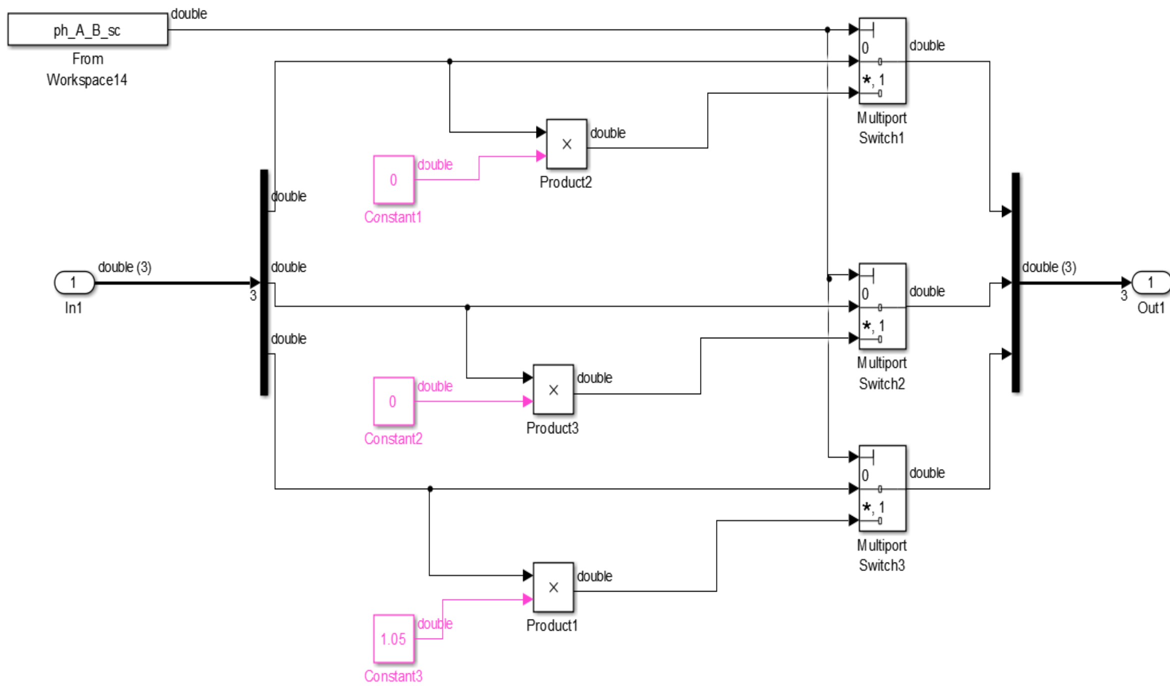


Figure 40: Phase-to-Phase Short-Circuit Fault Model with Protection – Simulink (F1)

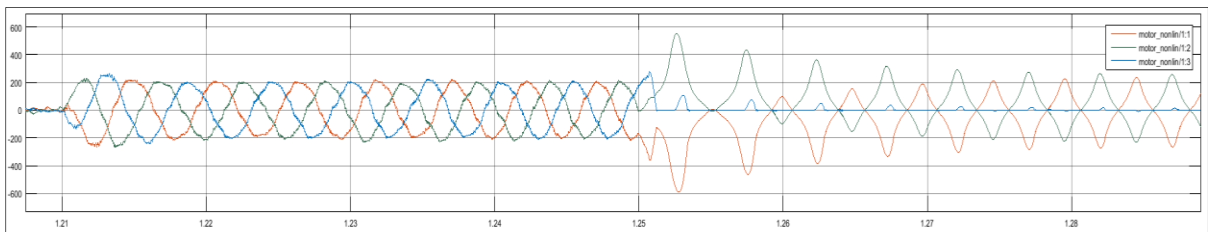


Figure 41: Motor Currents Zoomed (Transient State) – Simscape (F1)

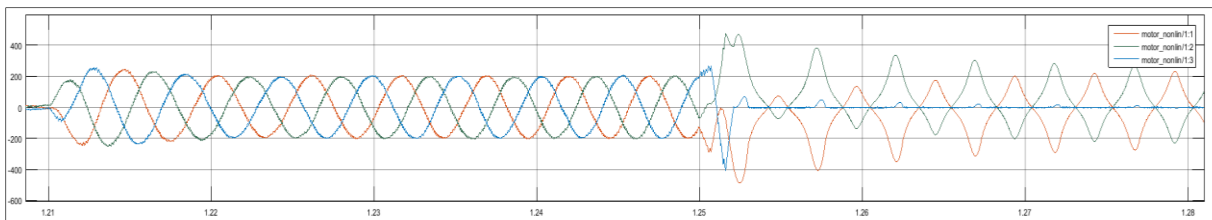


Figure 42: Motor Currents Zoomed (Transient State) – Simulink (F1)

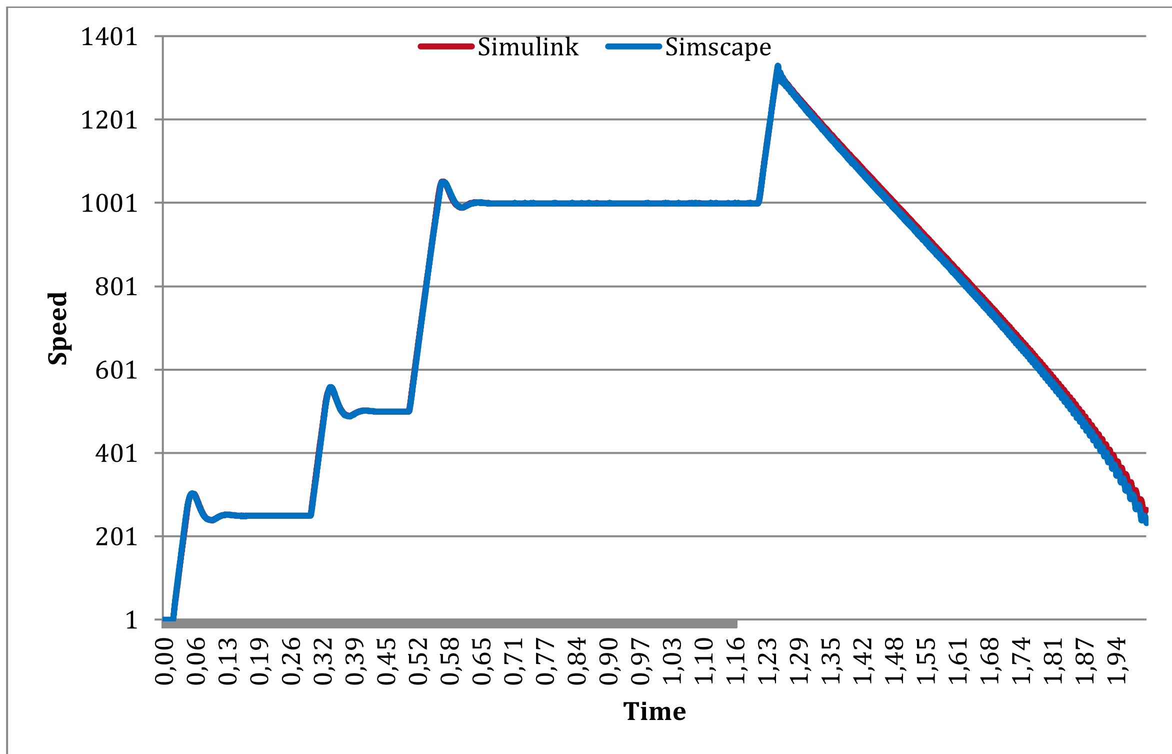


**Figure 43: Speed Comparison with Protection - Transient State (F1)**

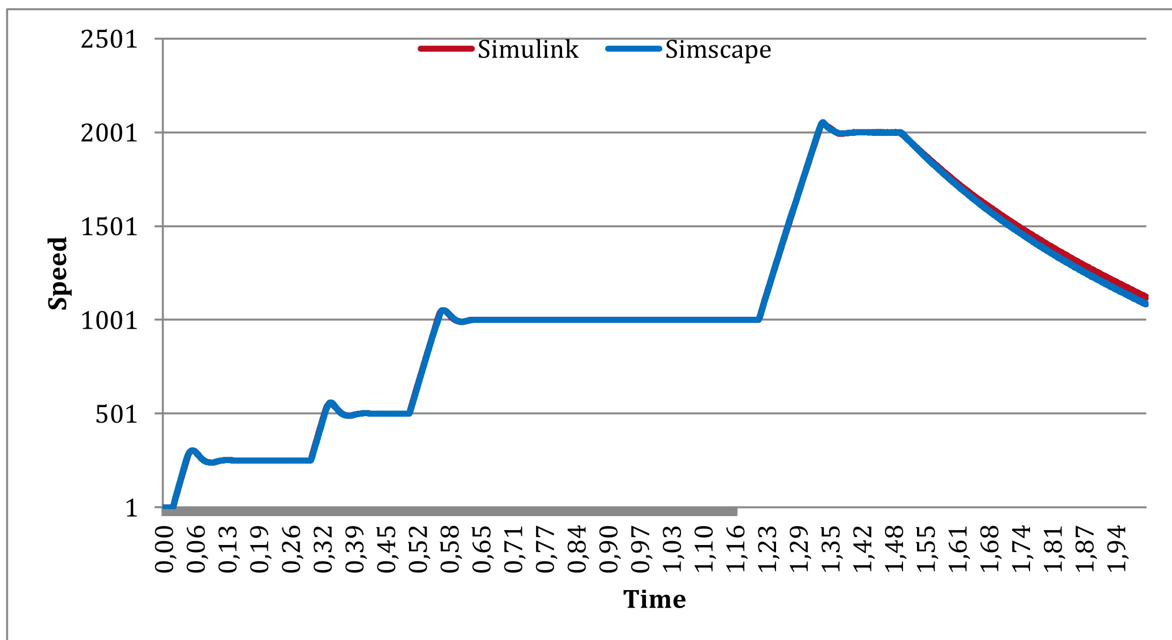


**Figure 44: Speed Comparison with Protection - Steady State (F1)**

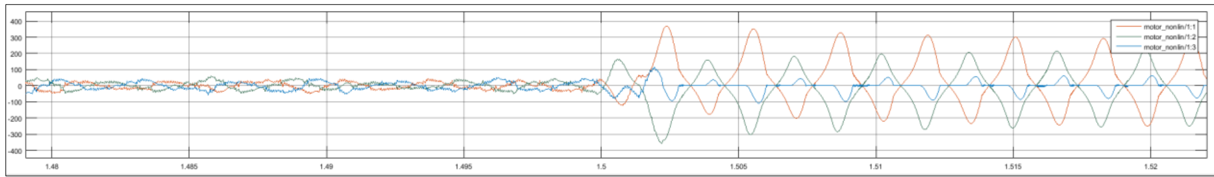

**Figure 45: Motor Currents Zoomed (Steady State) – Simscape (F1)**

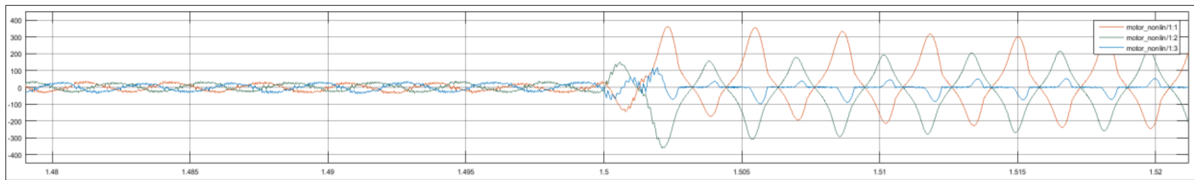

**Figure 46: Motor Currents Zoomed (Steady State) – Simulink (F1)**

## 4.4.2.2 Short circuit of one motor phase to earth (F2)

The short circuit between motor phases fault is simulated by using a switch connected between the phase and ground lines (see Figure 30) in electrical model whereas for data-flow model it is simulated by modifying the corresponding phase component to zero.

### *Protection mechanisms are not present*

The speed comparison curve of Simscape and Simulink during one phase short circuit to ground without protection mechanisms is shown in Figure 49 and Figure 50. The oscillation in motor speed is clearly seen when fault happens at t=1.25s (transient state), whereas when fault is introduced at t=1.5s (steady state), the motor output is relatively stable due to more energy drawn from the two healthy phases.

The phase currents are as shown in Figure 47, Figure 48, Figure 51 and Figure 52. The current is reduced to low value in the faulty phase and high currents are seen in the healthy phases due to high energy (voltage) drawn by the control software algorithms.

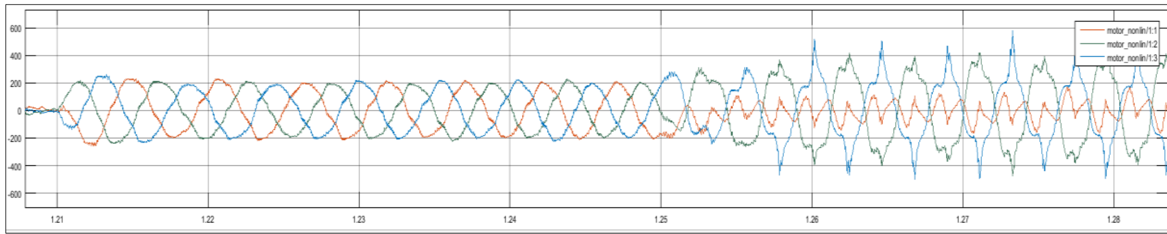

**Figure 47: Motor Currents Zoomed (Transient State) – Simscape (F2)**

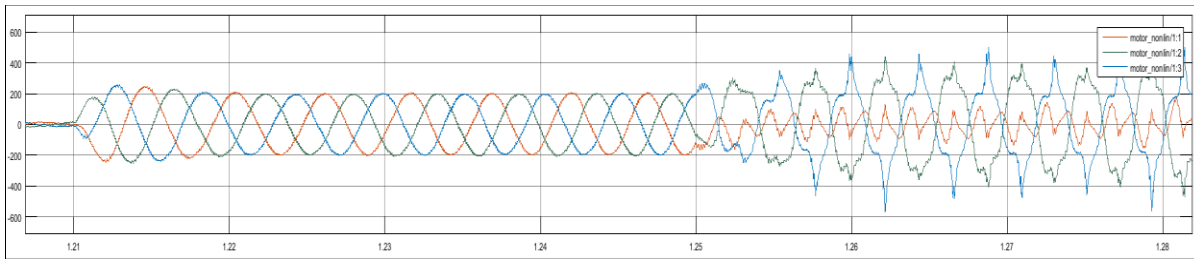

**Figure 48: Motor Currents Zoomed (Transient State) – Simulink (F2)**

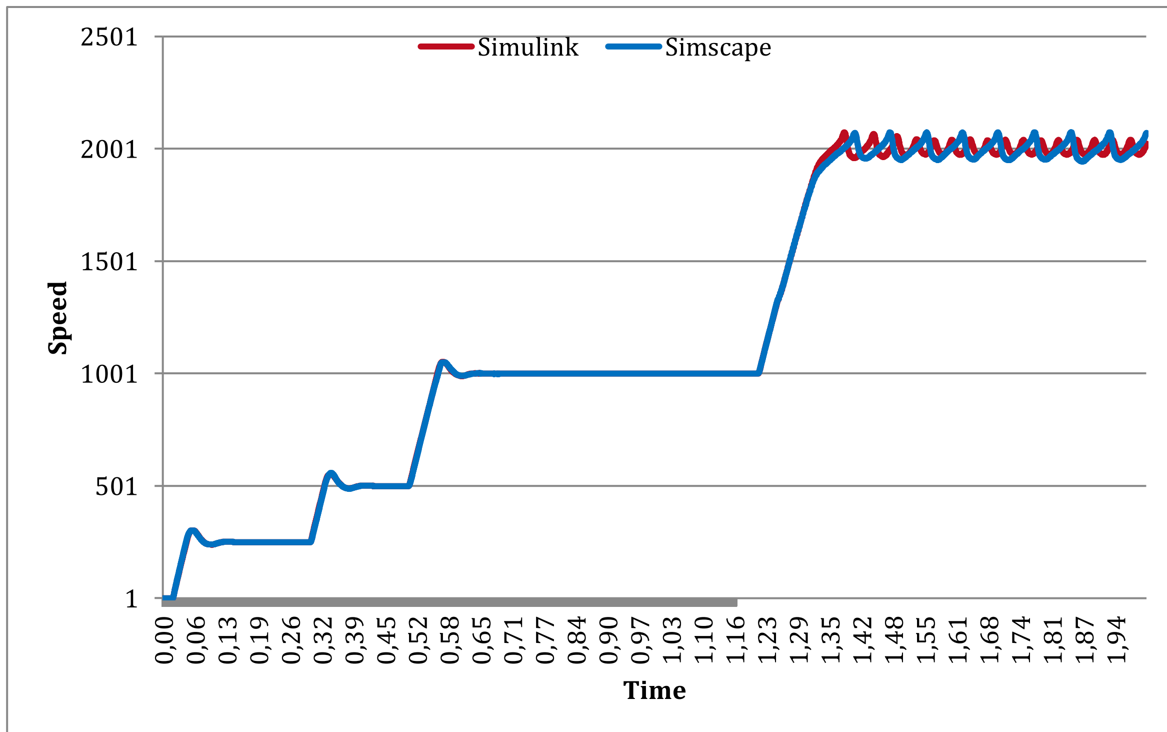


**Figure 49: Speed Comparison without Protection - Transient State (F2)**

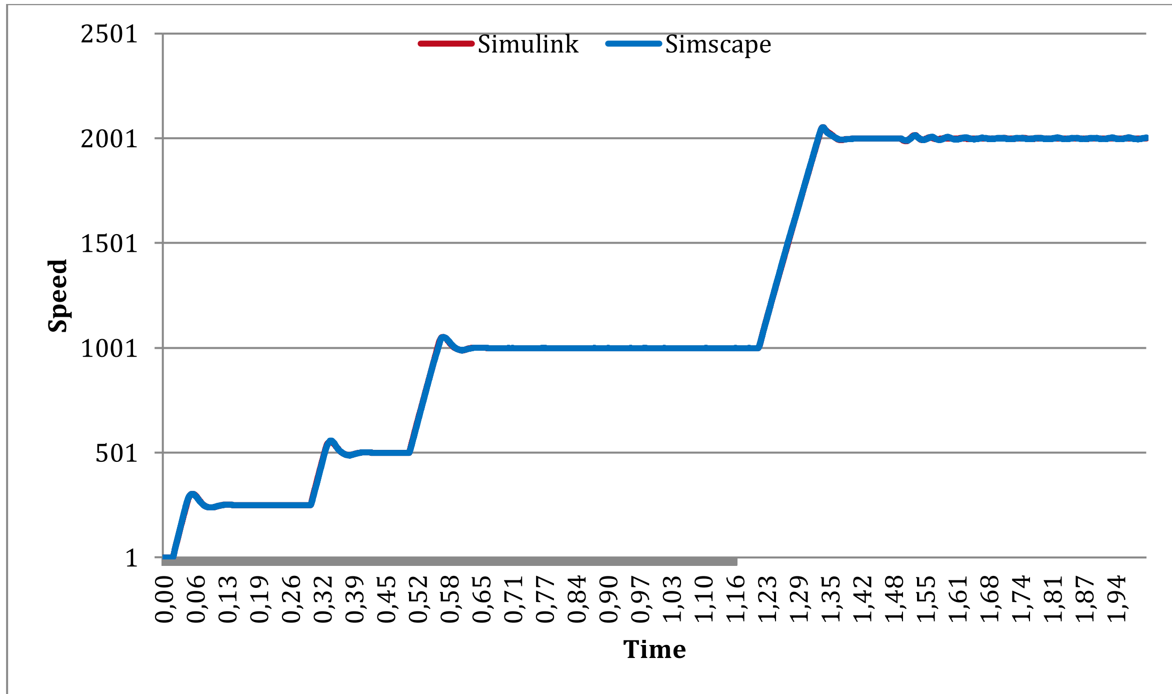


**Figure 50: Speed Comparison without Protection - Steady State (F2)**

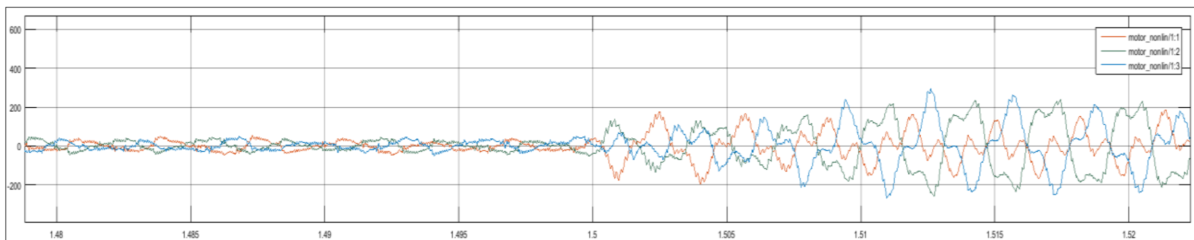

**Figure 51: Motor Currents Zoomed (Steady State) – Simscape (F2)**

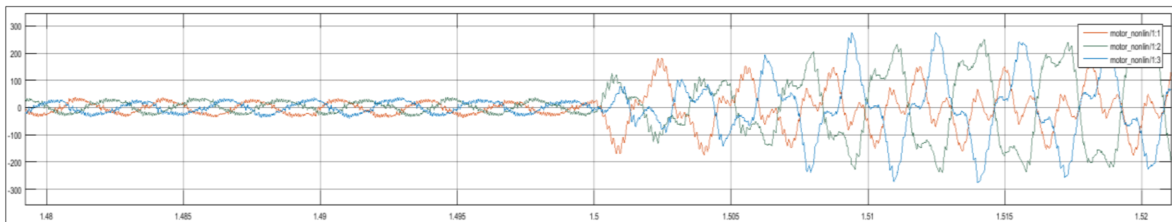

**Figure 52: Motor Currents Zoomed (Steady State) – Simulink (F2)**

### *Protection mechanisms are present*

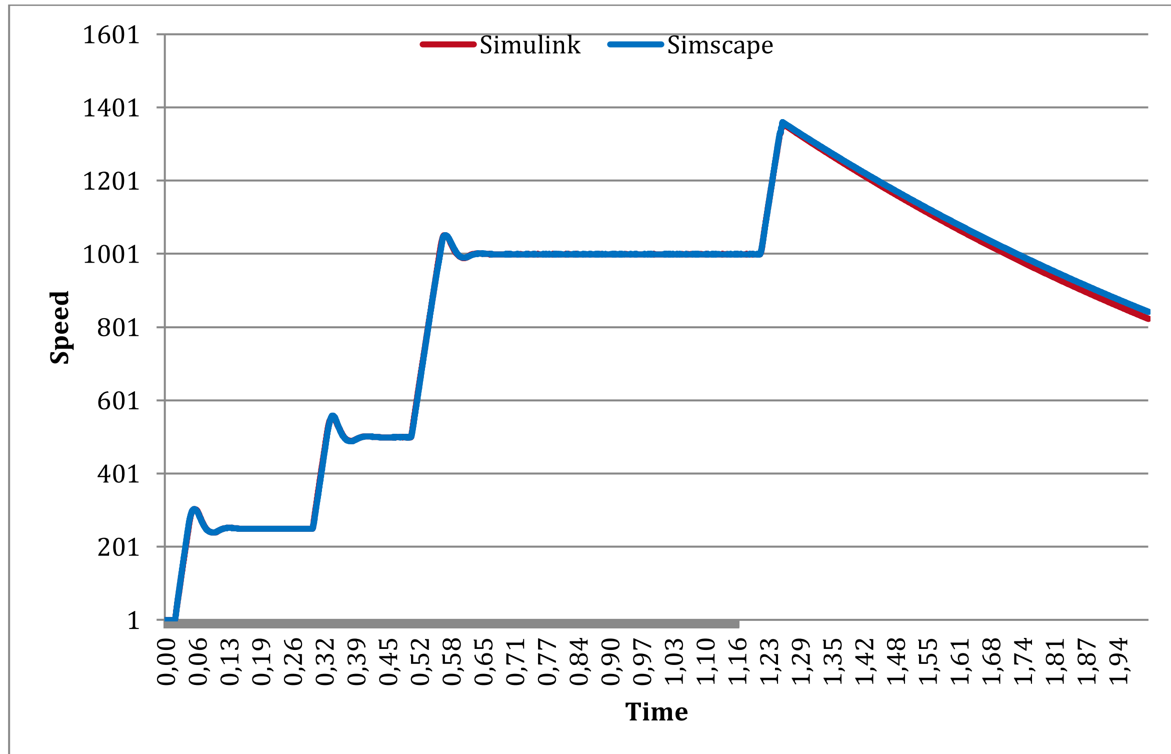


**Figure 53: Speed Comparison with Protection - Transient State (F2)**

The speed comparison curve of Simscape and Simulink during one phase short circuit to ground without protection mechanisms is shown in Figure 53 and Figure 54. When fault happens at t=1.25s (transient state), the high values in the phase currents is detected by the protection mechanism and the hardware is shutoff. Hence the motor speed gradually reduces towards zero. The phase currents are shown in Figure 55 and Figure 56, as clearly seen a high value of current is detected in one of the phase after the fault is introduced and all the phase currents are set to zero after that.

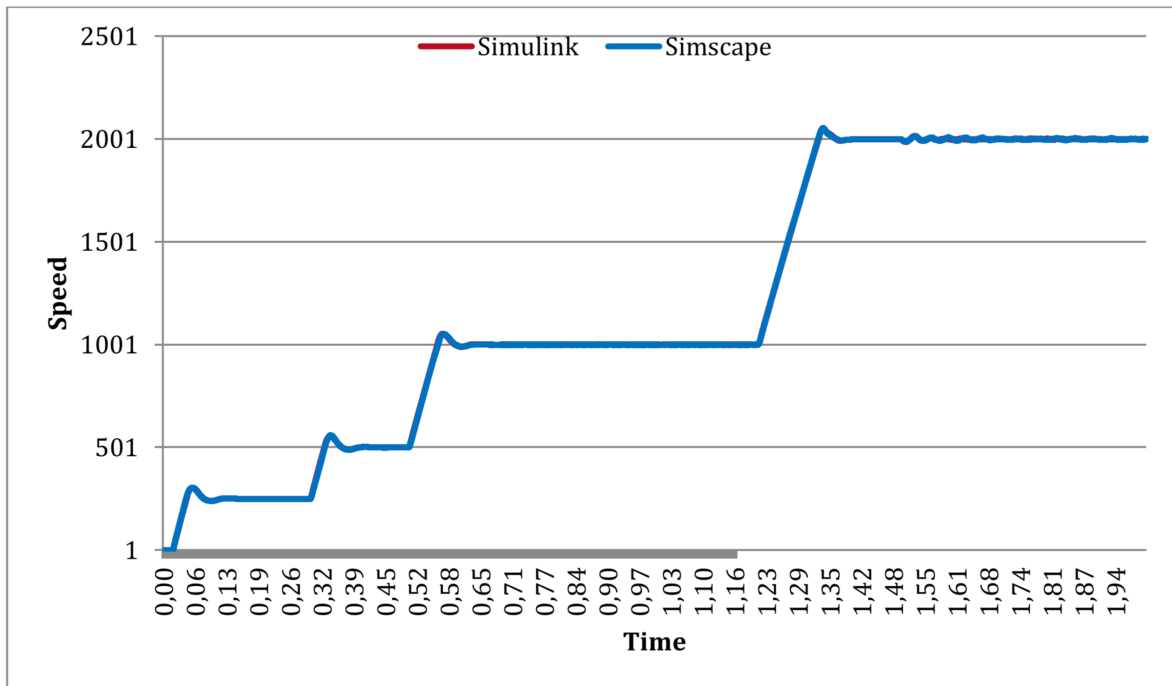


**Figure 54: Speed Comparison with Protection - Steady State (F2)**

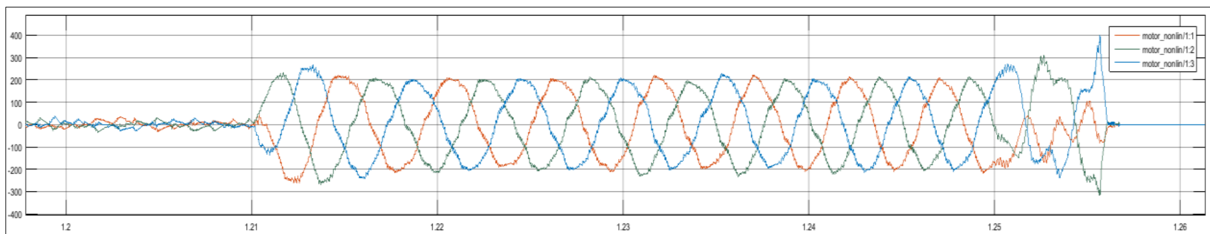

**Figure 55: Motor Currents Zoomed (Transient State) – Simscape (F2)**

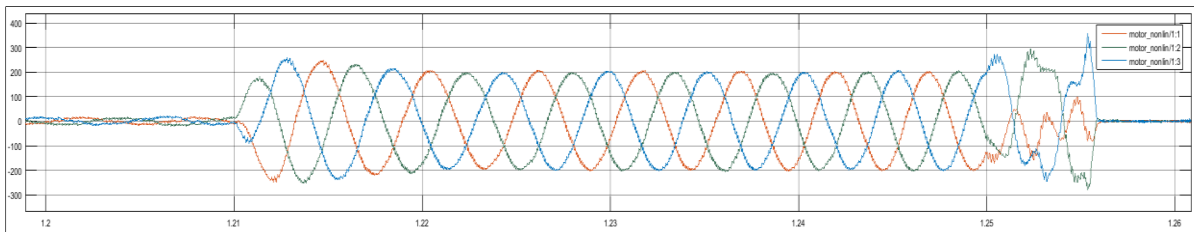

**Figure 56: Motor Currents Zoomed (Transient State) – Simulink (F2)**

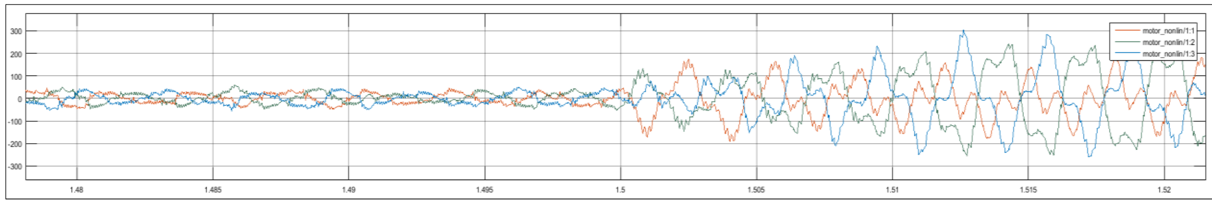

**Figure 57: Motor Currents Zoomed (Steady State) – Simscape (F2)**

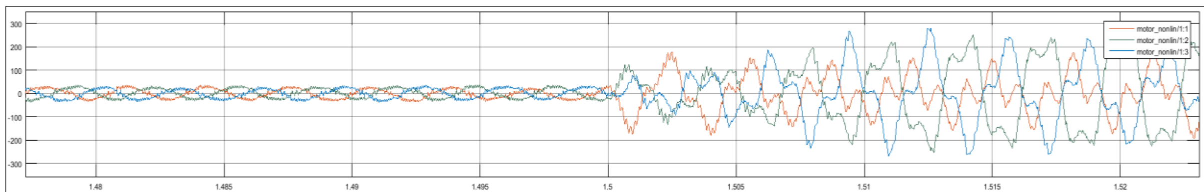

**Figure 58: Motor Currents Zoomed (Steady State) – Simulink (F2)**

For the remaining faults F3 to F10; we consider only the case with protection mechanisms being a part of control software.

#### 4.4.2.3 Short circuit of two motor phases to earth (F3)

The short circuit between motor phases fault is simulated by using a switch connected between phase lines and ground (see Figure 30) in electrical model whereas for data-flow model it is simulated by modifying the corresponding phase component to zero. In this test, two of the three motor phases are shorted to ground with protection. When two of the phases are shorted to ground irrespective of whether it is transient or steady state, the motor works mainly as generator and speed gradually reduces to zero.

#### 4.4.2.4 Open circuit of one motor phase (F4)

The open circuit of motor phases fault is simulated by using an open switch (see Figure 30) in electrical model whereas for data-flow model it is not straightforward as the input phase lines to the motor are not electrical. It cannot be simulated in data flow model by setting the phase value to zero, since its meaning is dependent on modeling. Open circuit on phase line means

the faulty phase should not deliver any energy (voltage). Hence a better strategy to simulate this fault in case of data flow model is switching off both the IGBT's of inverter leg which corresponds to the faulty phase line.

An open circuit fault on one phase is activated when the motor is running under steady-state set speed conditions. High currents are drawn on other two phases as shown in Figure 60 and Figure 61 to operate the drive without any problems. Whereas, when the same fault is activated while motor being accelerated as in at t = 1.25s, hardware shutoff is activated due to detection of overcurrent condition. Figure 59 and Figure 62 show the speed comparison curves of Simscape and Simulink in steady (t=1.5s) and transient (t=1.25s) states respectively.

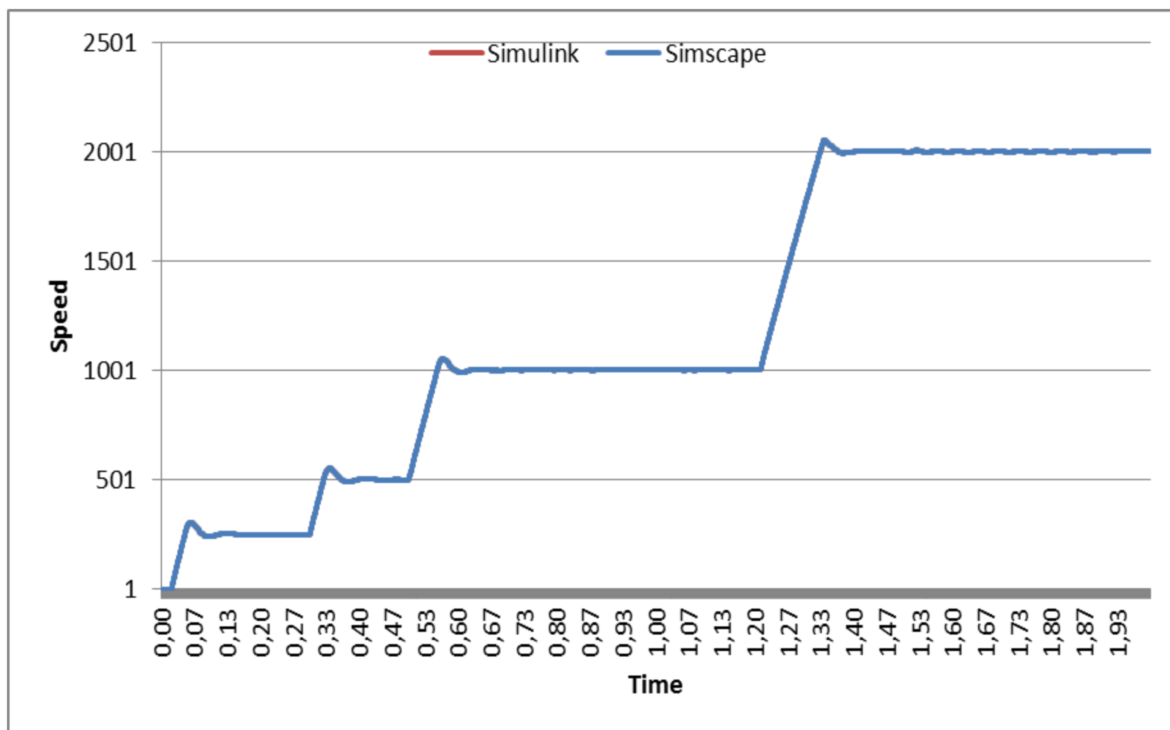


**Figure 59: Speed Comparison with Protection - Steady State (F4)**

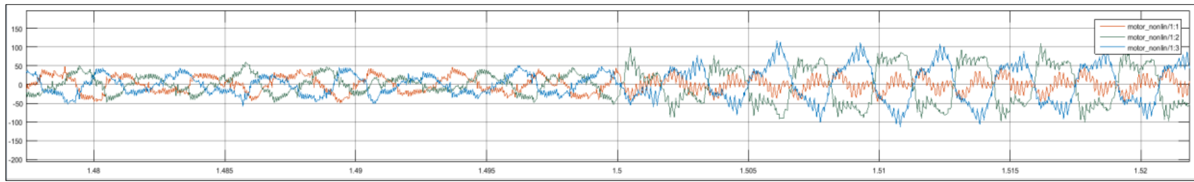

**Figure 60: Motor Currents Zoomed (Steady State) – Simscape (F4)**

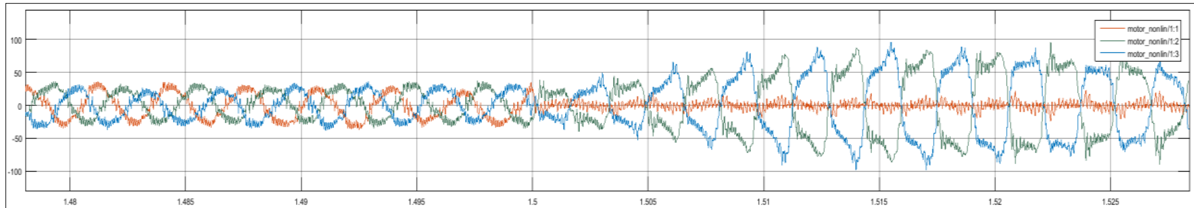

**Figure 61: Motor Currents Zoomed (Steady State) – Simulink (F4)**

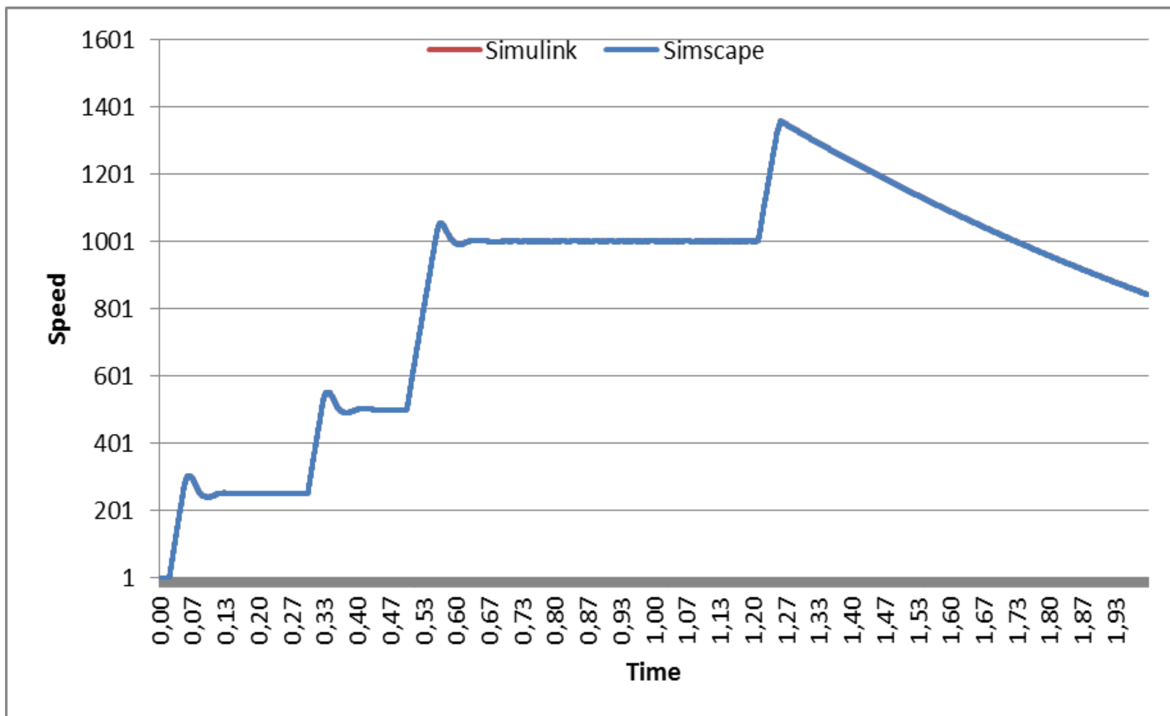


**Figure 62: Speed Comparison with Protection - Transient State (F4)**

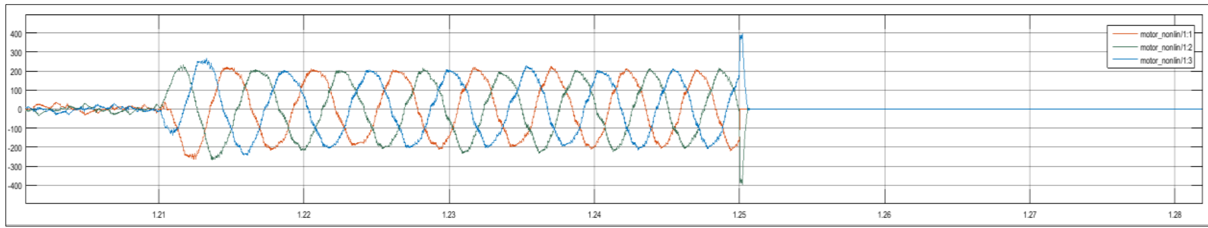

**Figure 63: Motor Currents Zoomed (Transient State) – Simscape (F4)**

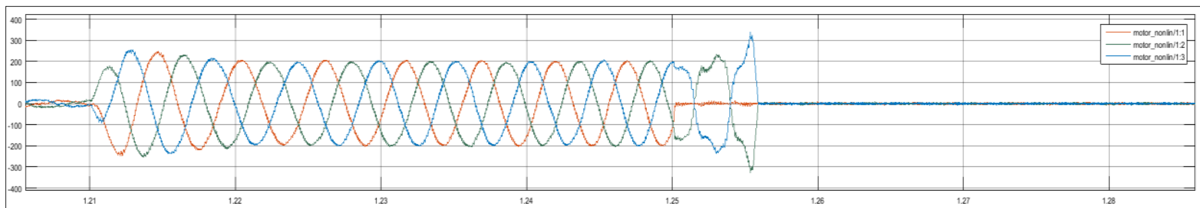

**Figure 64: Motor Currents Zoomed (Transient State) – Simulink (F4)**

### 4.4.2.5 Open circuit of two motor phases (F5)

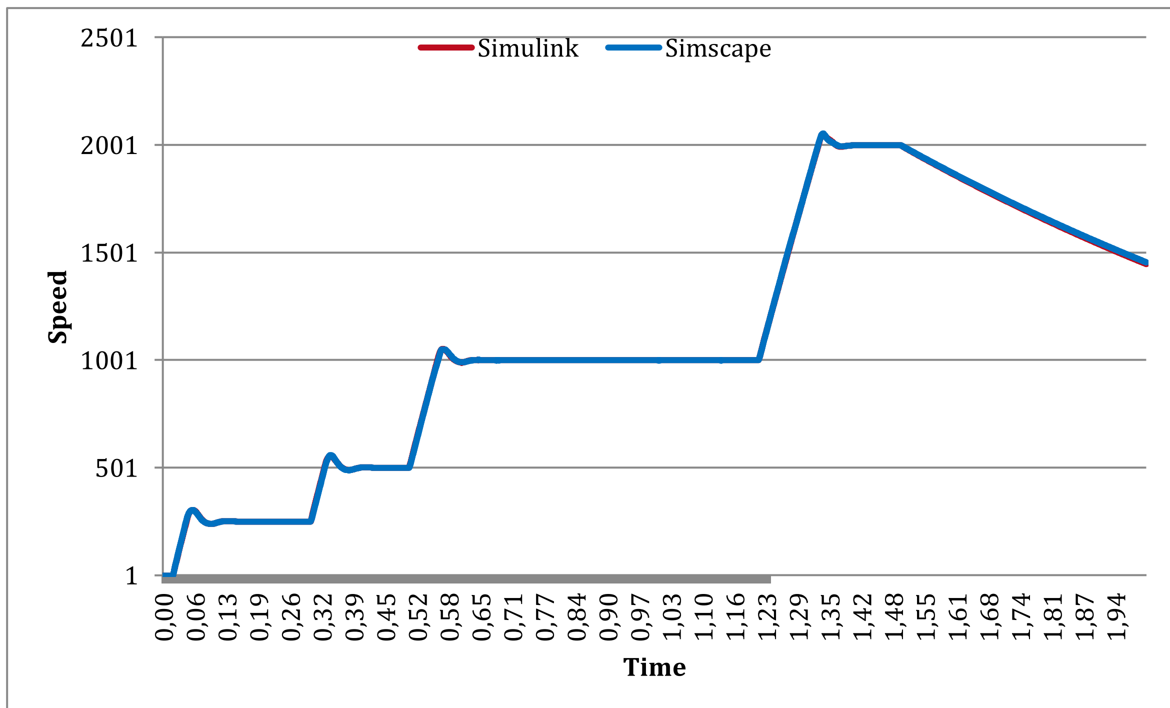


**Figure 65: Speed Comparison with Protection - Steady State (F5)**

The open circuit of motor phases fault is simulated by using an open switch (see Figure 30) in electrical model whereas for data-flow model similar to fault F4, both the IGBT's of inverter legs which corresponds to the faulty phase lines are switched off. Irrespective of system state either transient or steady, the motor cannot be driven anymore with this fault. As soon as fault is activated, motor speed slowly damps down towards zero. The speed comparison curves of Simscape and Simscape in steady and transient states are shown in Figure 65 and Figure 66 respectively. The motor currents are shown in Figure 67, Figure 68, Figure 69 and Figure 70.

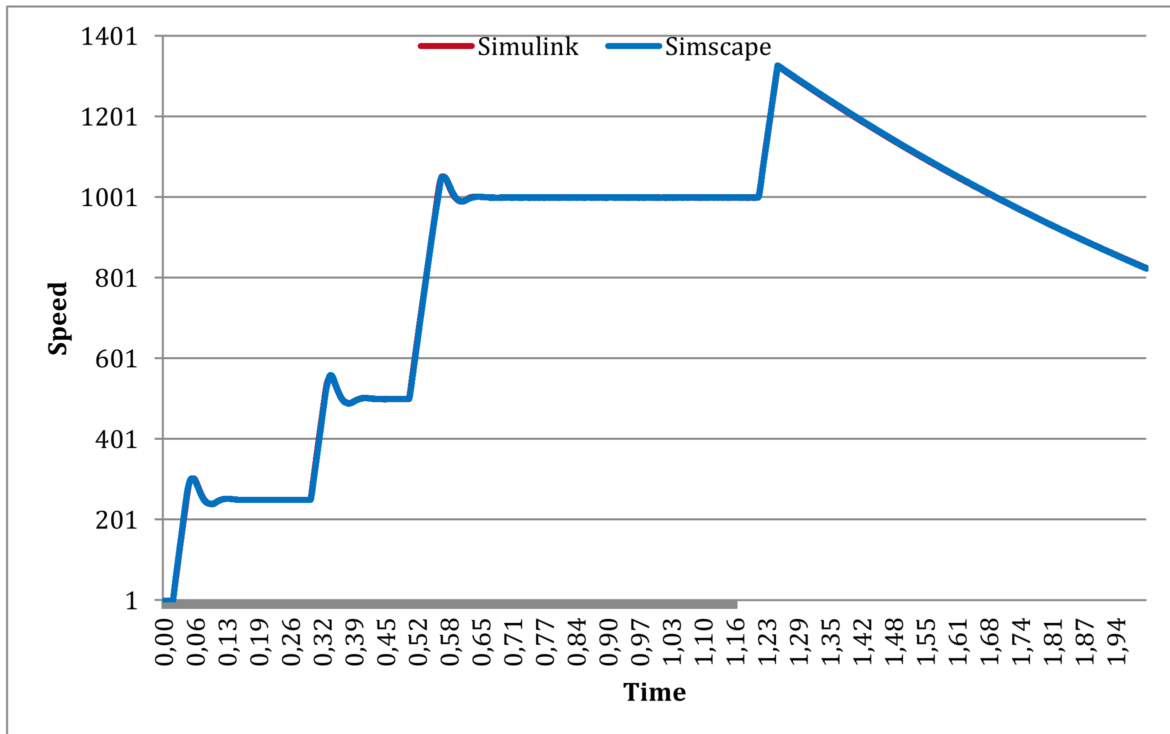


**Figure 66: Speed Comparison with Protection - Transient State (F5)**

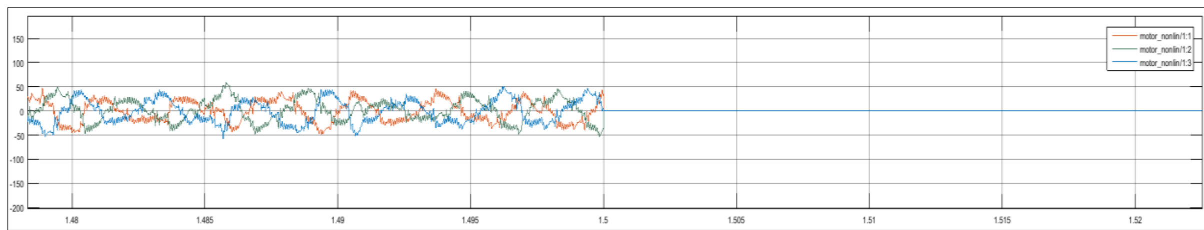

**Figure 67: Motor Currents Zoomed (Steady State) – Simscape (F5)**

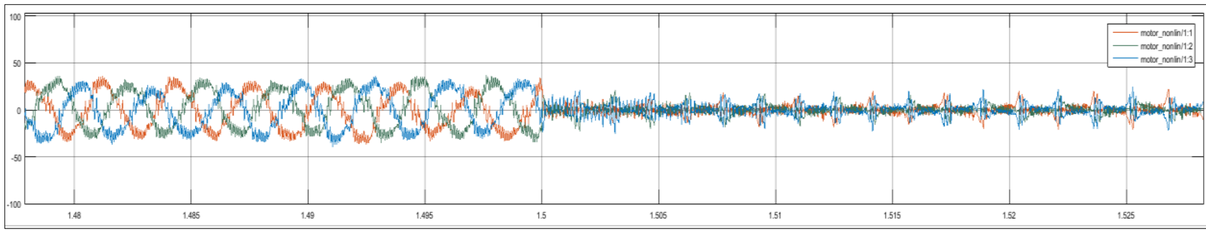


**Figure 68: Motor Currents Zoomed (Steady State) – Simulink (F5)**

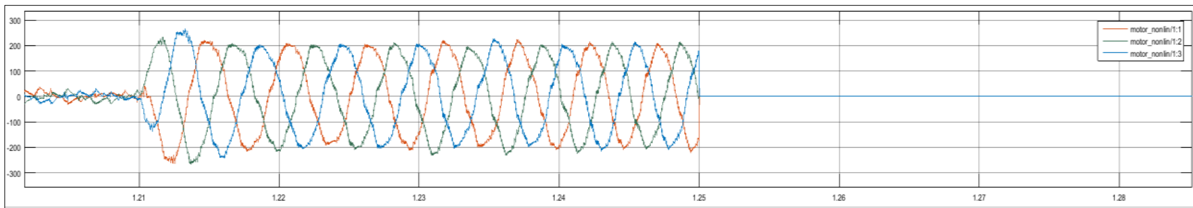


**Figure 69: Motor Currents Zoomed (Transient State) – Simscape (F5)**

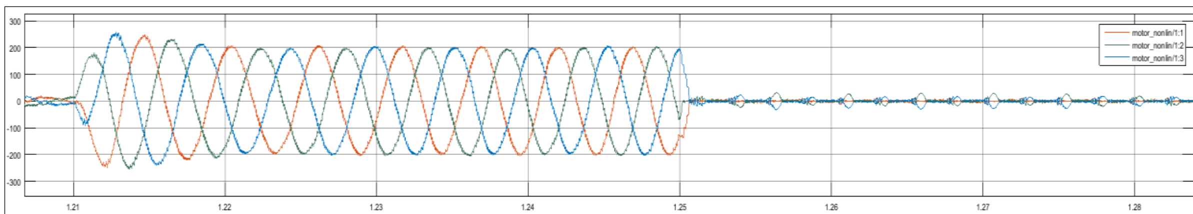


**Figure 70: Motor Currents Zoomed (Transient State) – Simulink (F5)**

### 4.4.2.6 Short circuit of IGBT (high-side) to high (F6)

The IGBT short circuit puts extreme stress on the inverter switching devices and therefore requires immediate attention of the protection circuit. An IGBT might fail due to current stress or voltage stress. The failure due to current occurs when the device is carrying load current whereas the voltage failure occurs when the device is switching off the load current.

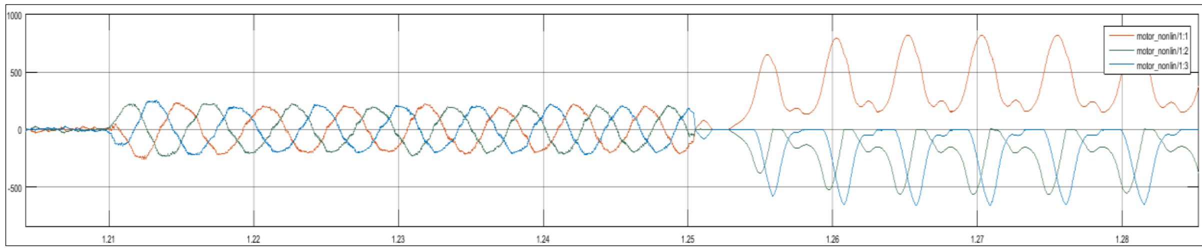


**Figure 71: Motor Currents Zoomed (Transient State) – Simscape (F6)**

The short circuit of IGBT1 (high-side) fault is simulated by using a closed switch (see Figure 30) in electrical model. The short circuit IGBT (high-side) means, one transistor on that inverter leg cannot switch-off. A short-circuit on IGBT1 condition is introduced in data-flow model by always switch-on high-side transistor and switch-off the complementary transistor IGBT4 thereby resulting in short-circuit through dc-link capacitor.

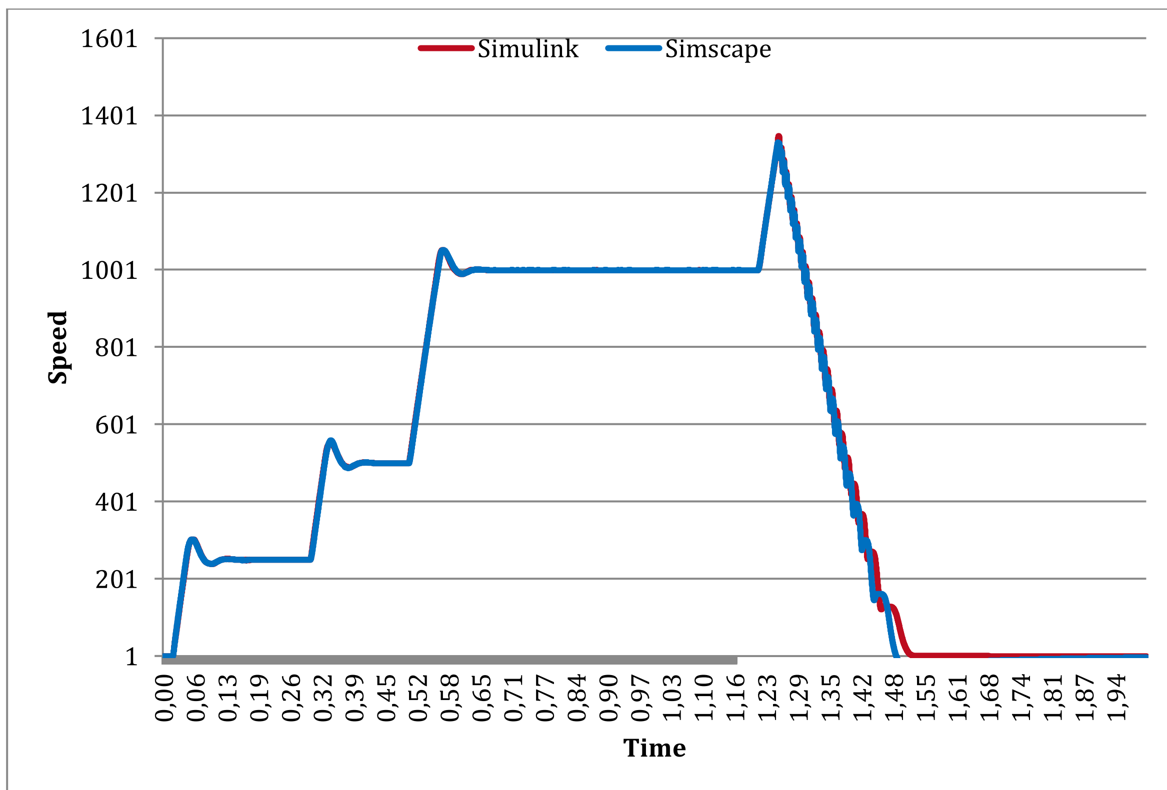


**Figure 72: Speed Comparison with Protection - Transient State (F6)**

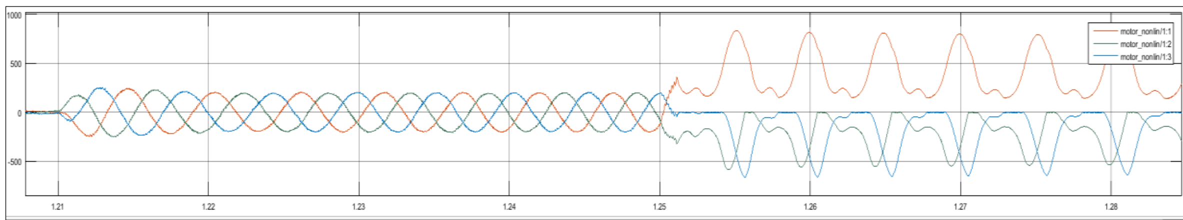


**Figure 73: Motor Currents Zoomed (Transient State) – Simulink (F6)**

The stator currents are shown in Figure 71, Figure 73, Figure 75 and Figure 76. As soon as fault is injected, the current in phase ‘A’ gets substantially positive and other two phases ‘B’ and ‘C’ are negative. Because of the high positive and negative torque pulses the motor speed drastically reduces. The motor speed comparison curves in transient and steady states are shown in Figure 72 and Figure 74 respectively.

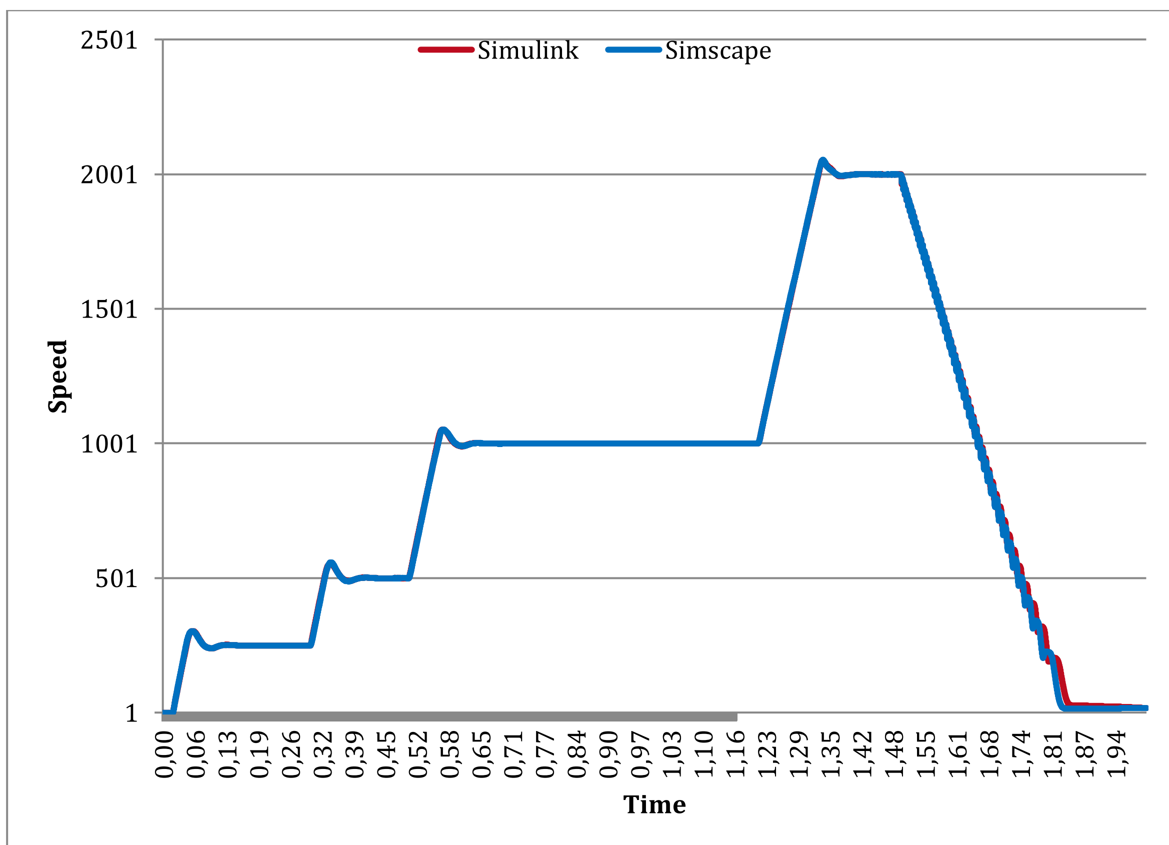


**Figure 74: Speed Comparison with Protection - Steady State (F6)**

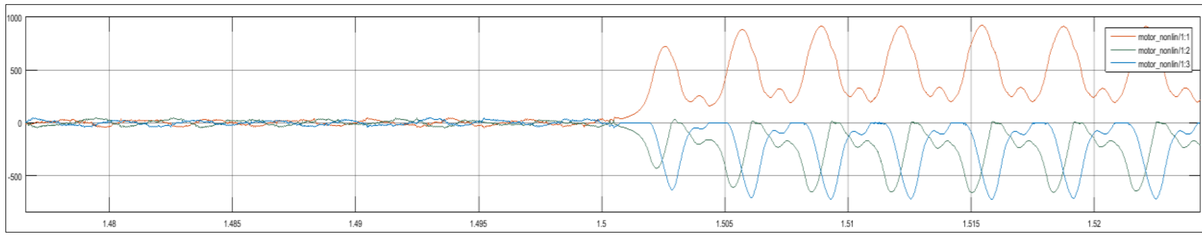


**Figure 75: Motor Currents Zoomed (Steady State) – Simscape (F6)**

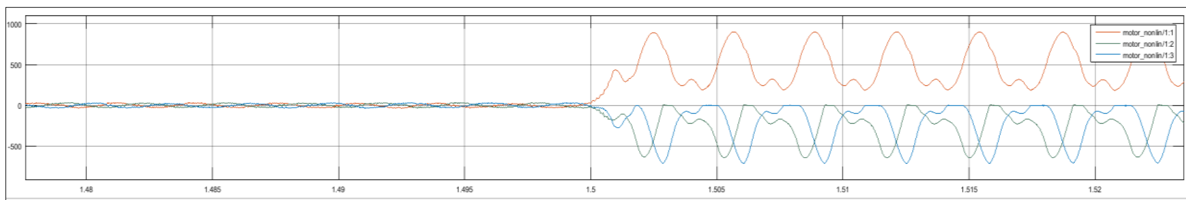


**Figure 76: Motor Currents Zoomed (Steady State) – Simulink (F6)**

### 4.4.2.7 Short circuit of IGBT (low-side) to low (F7)

The short circuit of IGBT4 (low-side) fault is simulated by using a closed switch (see Figure 30) in electrical model. In case of short circuit across IGBT (low-side), one transistor on that leg cannot switch-off. Similar to F6, it is introduced by always switching on the low-side transistor IGBT4 and switching off the complementary transistor IGBT1 thereby resulting in short-circuit through dc-link capacitor.

### 4.4.2.8 Open circuit of IGBT-gate signal (F8)

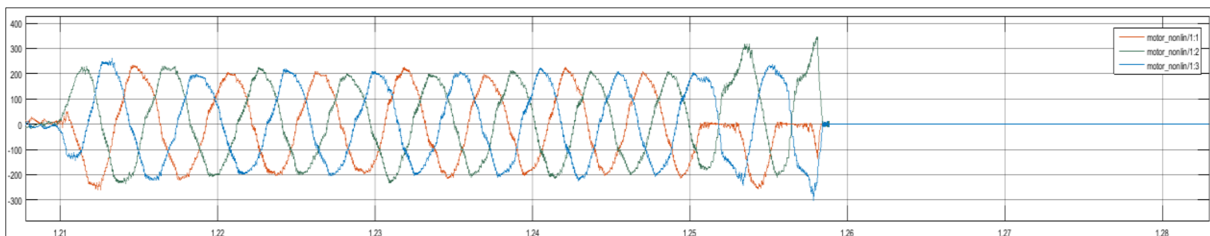


**Figure 77: Motor Currents Zoomed (Transient State) – Simscape (F8)**

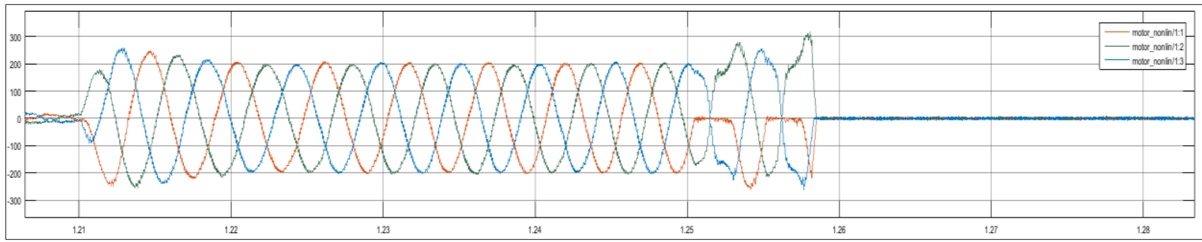


**Figure 78: Motor Currents Zoomed (Transient State) – Simulink (F8)**

The inverter transistors (IGBT's) are normally controlled by isolated base drive amplifiers. Malfunctioning of one of these units can result in a missing base drive. Since the faulty IGBT is inoperative, the output phase voltage of the inverter leg is determined by the polarity of current and the switching pattern of the complementary IGBT of the same inverter leg.

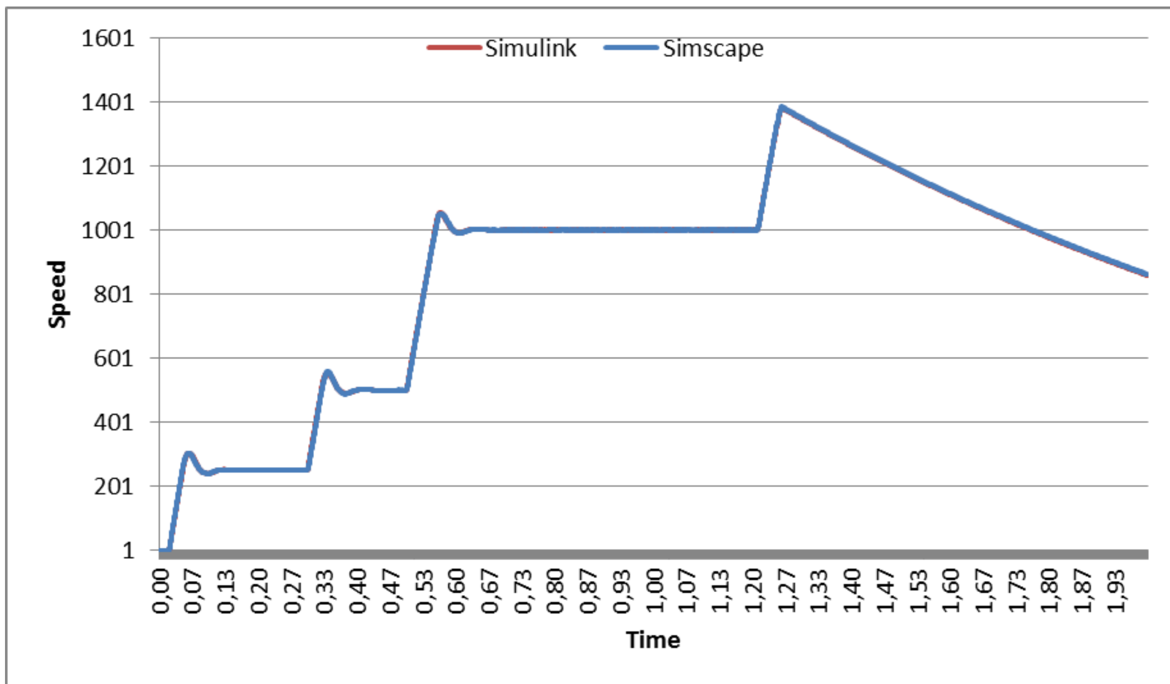


**Figure 79: Speed Comparison with Protection - Transient State (F8)**

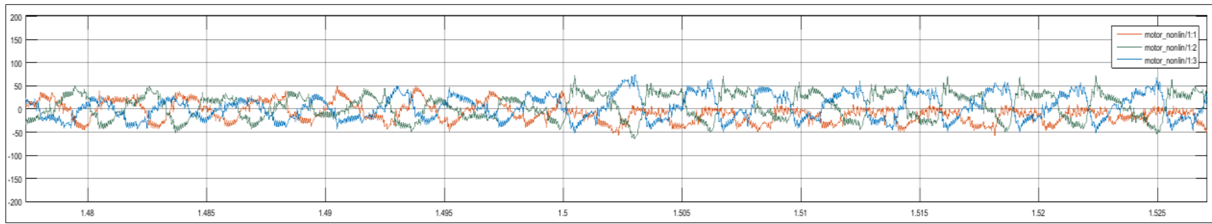

**Figure 80: Motor Currents Zoomed (Steady State) – Simscape (F8)**

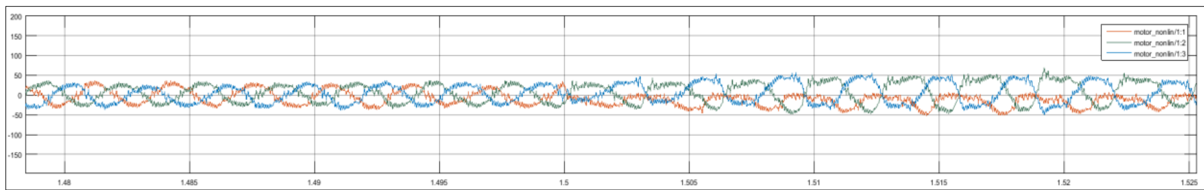

**Figure 81: Motor Currents Zoomed (Steady State) – Simulink (F8)**

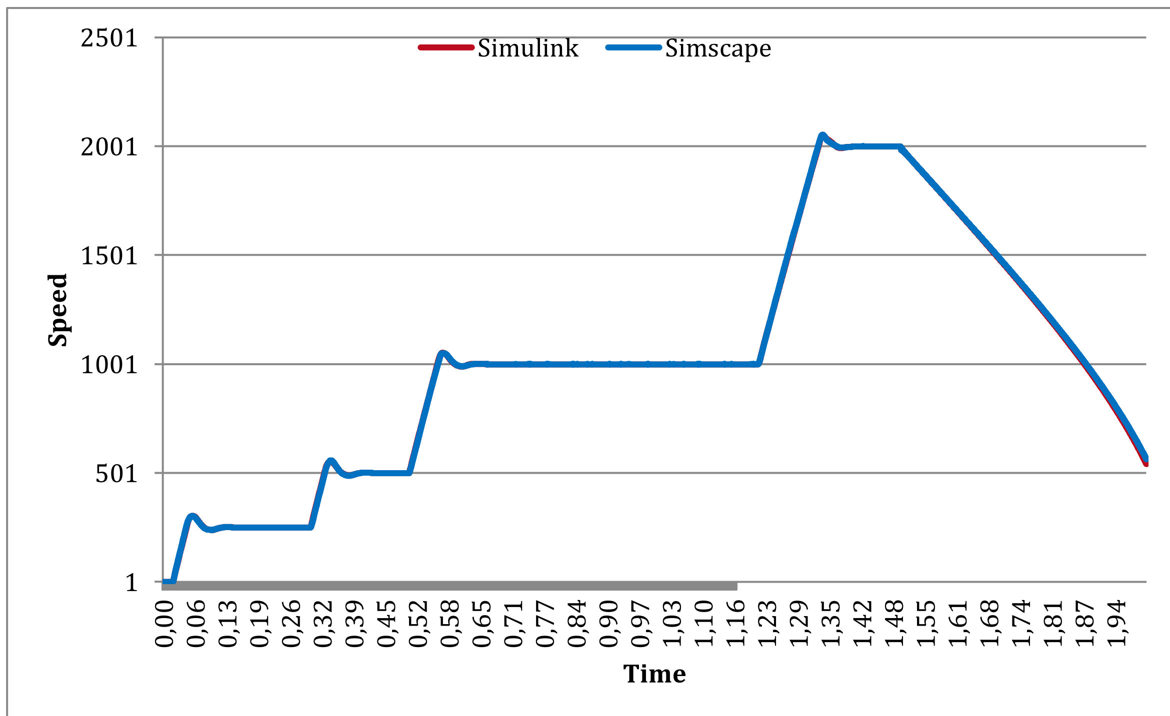


**Figure 82: Speed Comparison with Protection - Steady State (F8)**

### 4.4.2.9 Short circuit across DC-link capacitor (F9)

The short circuit across dc-link capacitor fault is simulated by using a closed switch (see Figure 30) in electrical model. In case of data flow AFM4 fault model (refer 4.3.2 Abstract Fault Models and Fault Interface Design) is used to set the dc voltage (capacitor output voltage) to zero.

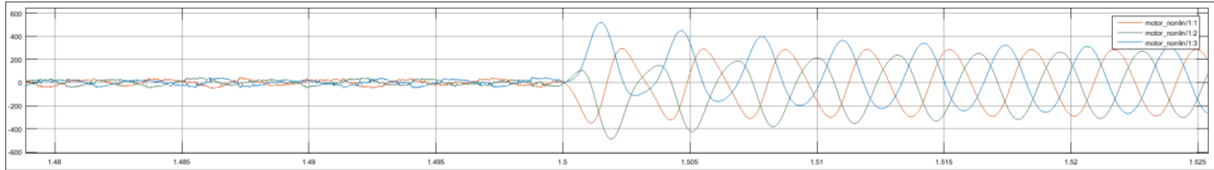

**Figure 83: Motor Currents Zoomed (Steady State) – Simscape (F9)**

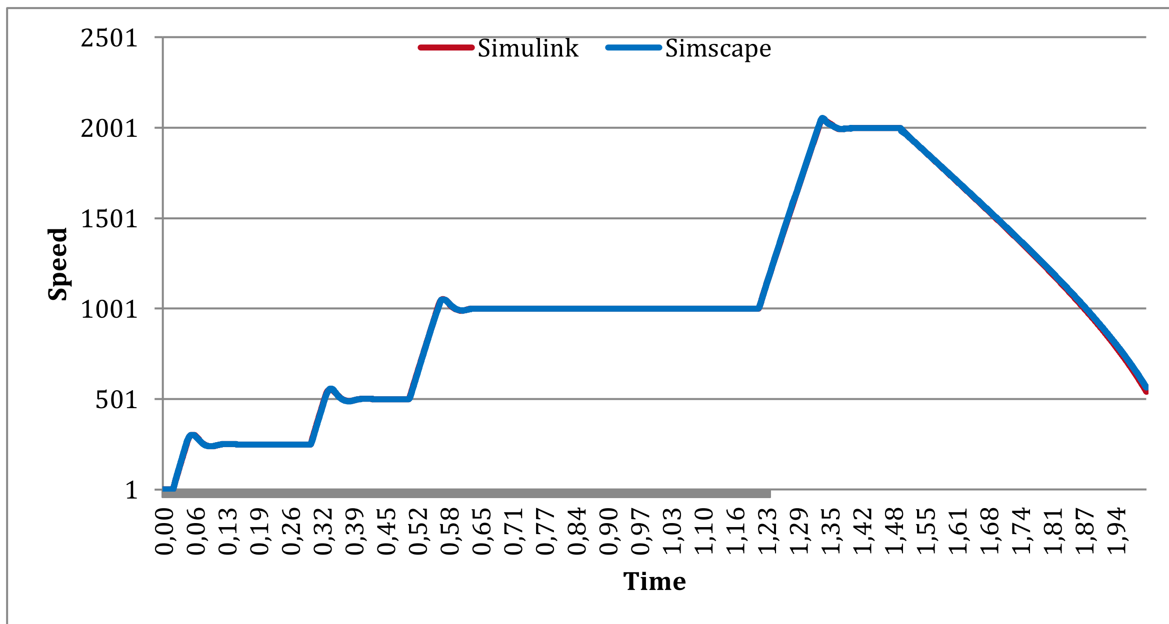


**Figure 84: Speed Comparison with Protection - Steady State (F9)**

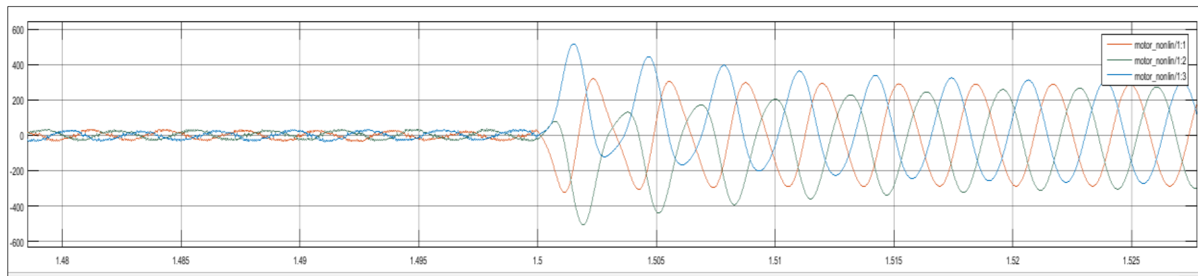


**Figure 85: Motor Currents Zoomed (Steady State) – Simulink (F9)**

## 4.4.2.10 Performance Degradation of Electrical Components

Electrical components tend to degrade and fail faster under high electrical and thermal stress conditions that they are often subjected to during operations. As these components undergo aging and lose, partially or totally, their desired functions. If not effectively monitored and controlled, degradation of these components may impair their performance characteristics and lead to a reduction in reliability of associated systems. To maintain adequate performance the system, it is essential to check the component tolerance levels and also monitor, to take appropriate preventive actions. As an example, the degradation of capacitor is presented in the following section.

**Capacity reduction of dc-link capacitor (F10):**

Capacitors tend to degrade their performance over time. Parametric faults are those changes that cause performance degradation of the circuit and these faults involve parameters' deviations from their nominal value that can consequently quit their tolerance band. In this experiment, we carry out a few capacity reduction tests for dc-link capacitor.

The capacity reduction of dc-link capacitor fault is simulated by using a variable resistor (see Figure 30) in electrical model. In case of data flow AFM7 fault model (refer 4.3.2 Abstract Fault Models and Fault Interface Design) is used to decrease the dc voltage (capacitor output voltage) by a specific percent.

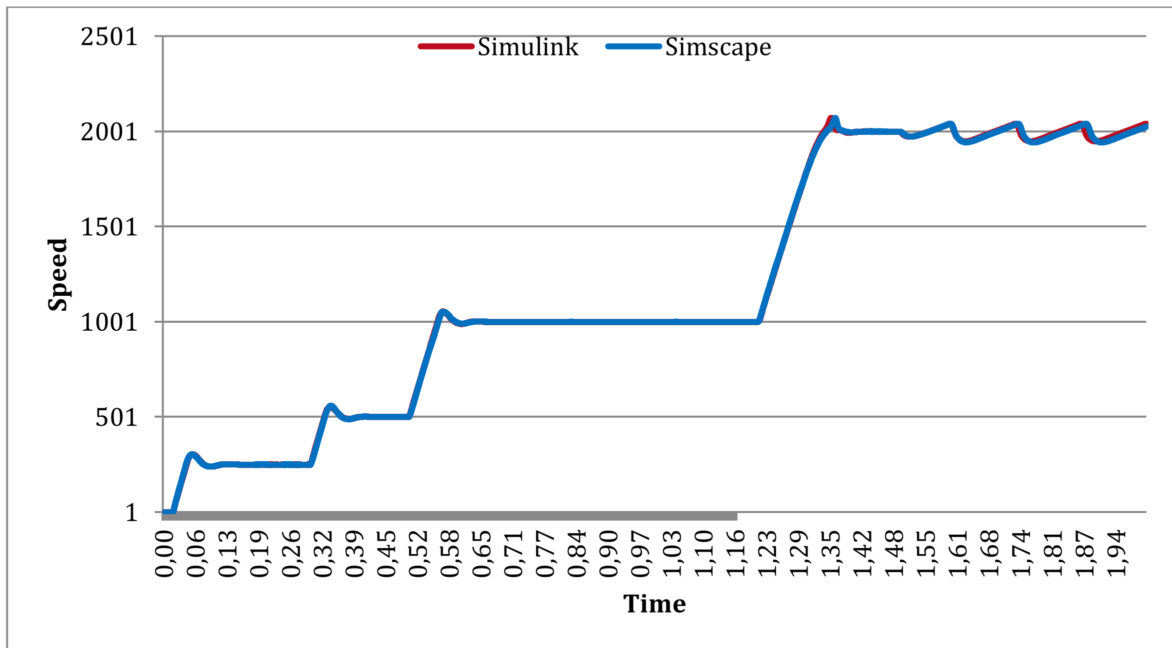


**Figure 86: Speed Comparison - Steady State (F10 - 50 % Reduction)**

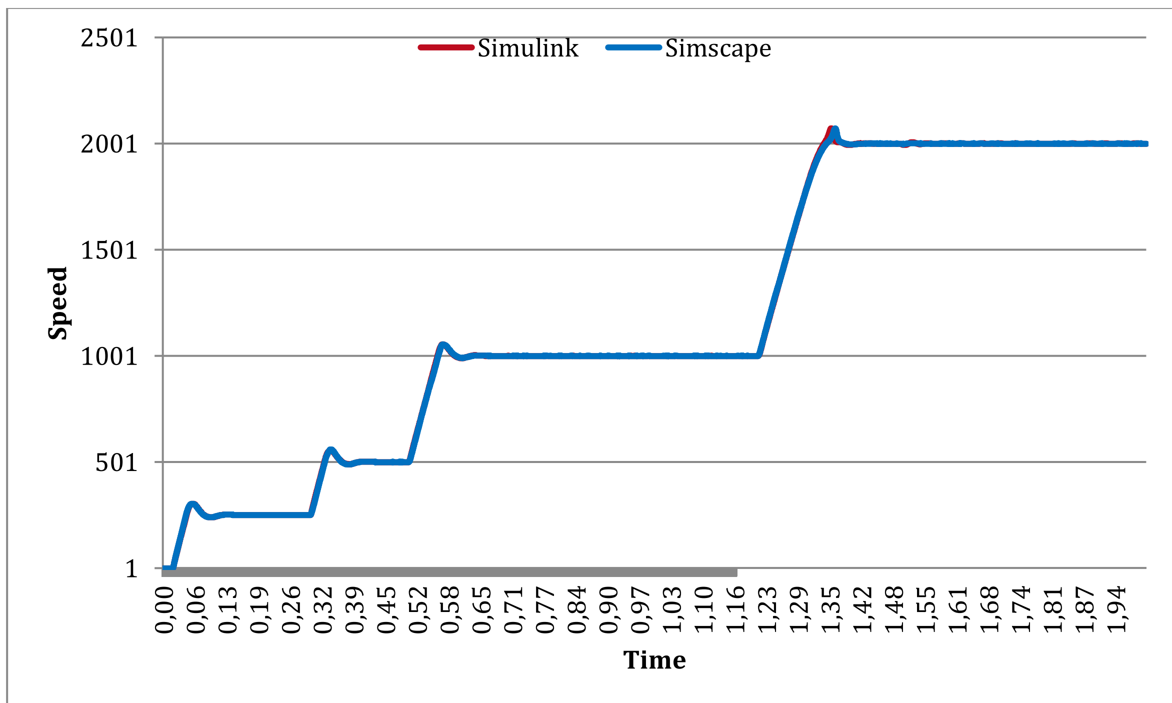


**Figure 87: Speed Comparison - Steady State (F10 - 20 % Reduction)**

#### 4.4.2.11 Conclusion

In this section, how the fault behaviors are transferred from Simscape electrical (detailed) models to Simulink data-flow (abstract) models are discussed. Various faults in the inverter module of an industrial motor control system are presented. This generalized idea independent of modeling abstraction and application, could be applied to faults, to transfer their behavior from accurate models to an abstract models.

## 4.5 Digital Fault Modeling based on TLM

While several fault-injection tools have been developed over the years for gate-level and register-transfer level (RTL), fault-injection mechanisms for transaction-level models are still a relatively new topic with limited research.

### 4.5.1 State-of-the-art

- [79], [100]: An effective fault injection at gate level was researched.
- [101]: Different fault injection techniques and strategies are highlighted.
- [82]: proposes a mutation model for perturbing transaction level modeling (TLM) SystemC descriptions. In particular, the main constructs provided by the SystemC TLM 2.0 library have been analysed, and a set of mutants is proposed to perturb the primitives related to the TLM communication interfaces.
- [102]: proposes a multi-level fault simulation method which combines the accuracy of gate-level fault simulation and the simulation speed of behavioural models. A transaction-level model of the system is augmented by precise gate-level models of components which are subject to fault injection.
- [103]: provides a methodology that leverages state-of-the-art techniques for efficient fault simulation of structural faults together with transaction-level modeling.
- [104] [105]: A non-intrusive fault-injection technique is presented in which C++ virtual function tables are modified to extend the TLM transport interfaces with fault-injection capabilities. The modification is achieved by mutating the executable TLM

models of a LEON3 CPU. To add fault injection capabilities to the models, this approach augments the TLM initiator and target sockets. However, the approach is platform and compiler dependent. These dependencies are overcome in [106] by applying code mutation on the source code of TLM models. Therefore, faults are introduced directly into the models without manipulation of the C++ virtual-function tables. However, this latter method is intrusive because of its requirement to change the original code.

- [12]: The fault injection techniques presented are focused on fault injection into TLM-based VPs and target the extension of TLM sockets, payload, and interfaces with fault injection capabilities, which enable abstract fault-model definitions and non-intrusive fault injection during a simulation. This approach has the advantages of platform and compiler independent, and enables VP fault-model development outside of the TLM model. We use these injectable sockets by defining appropriate fault models to enable not only the modeling of simple faults like single bit flips but also double and triple bit flips.

### 4.5.2 Fault Injection for TLM Abstraction

In TLM-based VPs [10], modules exchange information using generic payload objects. The generic payload has attributes which are used as basis for modeling communication protocols. The attributes include command, address, data, byte enables, single word transfers, burst transfers, streaming and response status. Faults can be injected by modifying generic payload attribute fields. It is evident, that the faults happening in the bus signals will lead to data transaction errors and finally cause a system failure. The meta-model-based fault-library concept proposed in [107] is a systematic approach to define fault models for TLM abstraction level. For the fault-library to be widely applicable, the metamodel must be as abstract as possible, such that it can serve as foundation for libraries addressing not only different domains, but also different levels of abstraction. In our work, we consider mainly bit-flips (single, double, and triple) and stuck-at faults as these represent the smallest observable change at this abstraction level. By using the meta-model-based fault library

concept, the fault models could be extended with formalized semantic to describe the correlations between the different abstraction layers.

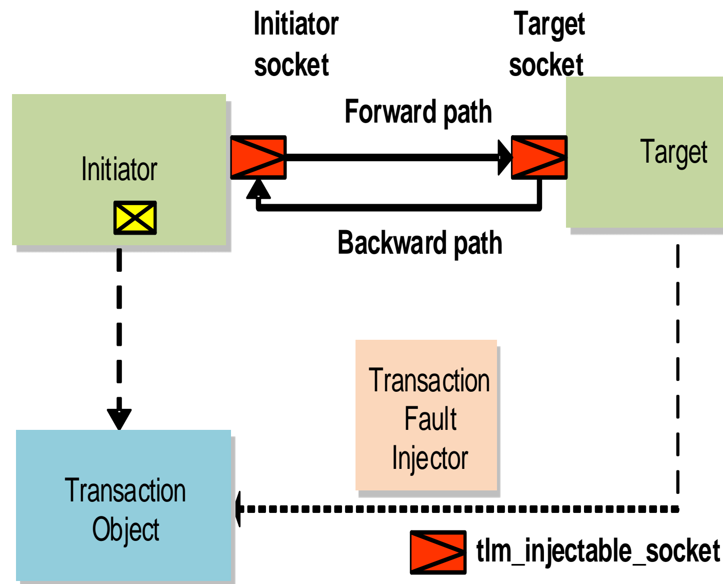


**Figure 88: TLM Injectable Socket**

To provide TLM-based VPs with fault-injection capabilities, interconnects have been embedded into the standard TLM initiator and target sockets to sabotage the transport methods normal functionality (e.g., blocking transport - *b_transport*, non-blocking transport - *nb_transport_fw*, *nb_transport_bw*). Three hooks (pre-hook, override-hook, post-hook) are provided inside the saboteur-like transport methods of interconnects to add more flexibility to the standard TLM transport flow (Figure 89). These hooks do not consume simulation time and fault injection is enabled by connecting a fault injector to one of the available hooks. The pre-hook gets called before the actual transport method, while the post-hook is called after the transport method is executed. The override-hook replaces the target socket's transport implementation completely and acts like an external (non-intrusive) mutant. The fault injector is a user-defined class derived from a TLM interface with specialized transport method through which a payload's contents are changed during the simulation. By modeling the VP faults inside the fault injector and connecting the injector to a specific TLM block, fault injection is performed without changing the TLM block's original code.

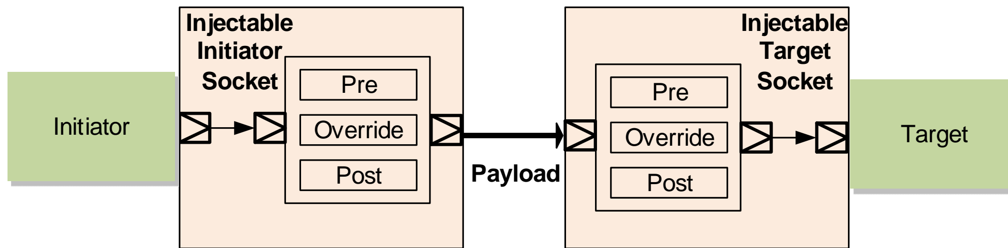


**Figure 89: TLM Injectable Socket Block Diagram [13]**

For using TLM injectable sockets it is necessary to model the fault injector and attach it to one of the hooks. We identify fault class and bit position information to inject faults in the transaction payload. The fault class is used to determine which field of the transaction a fault is injected into TLM transaction, and type of fault (error type in Figure 90, on the bottom-left, i.e., single-bit, double-bit, etc.). The possible fault classes on payload fields are shown in Figure 90, on the bottom-right.

### 4.5.3 Fault Interface Design and Injector

In addition, the information of the triple-bit-fault is entered in the field TBF (fault class information in Figure 90). The tlm_injectable_socket is reported by the bit position parameter to which bit of the corresponding field an error is injected. The bit position information is used to decide of which bit of the corresponding field an error is injected. As shown in Figure 91, the bit position information is divided into three fields, each of which is responsible for a particular bit error.

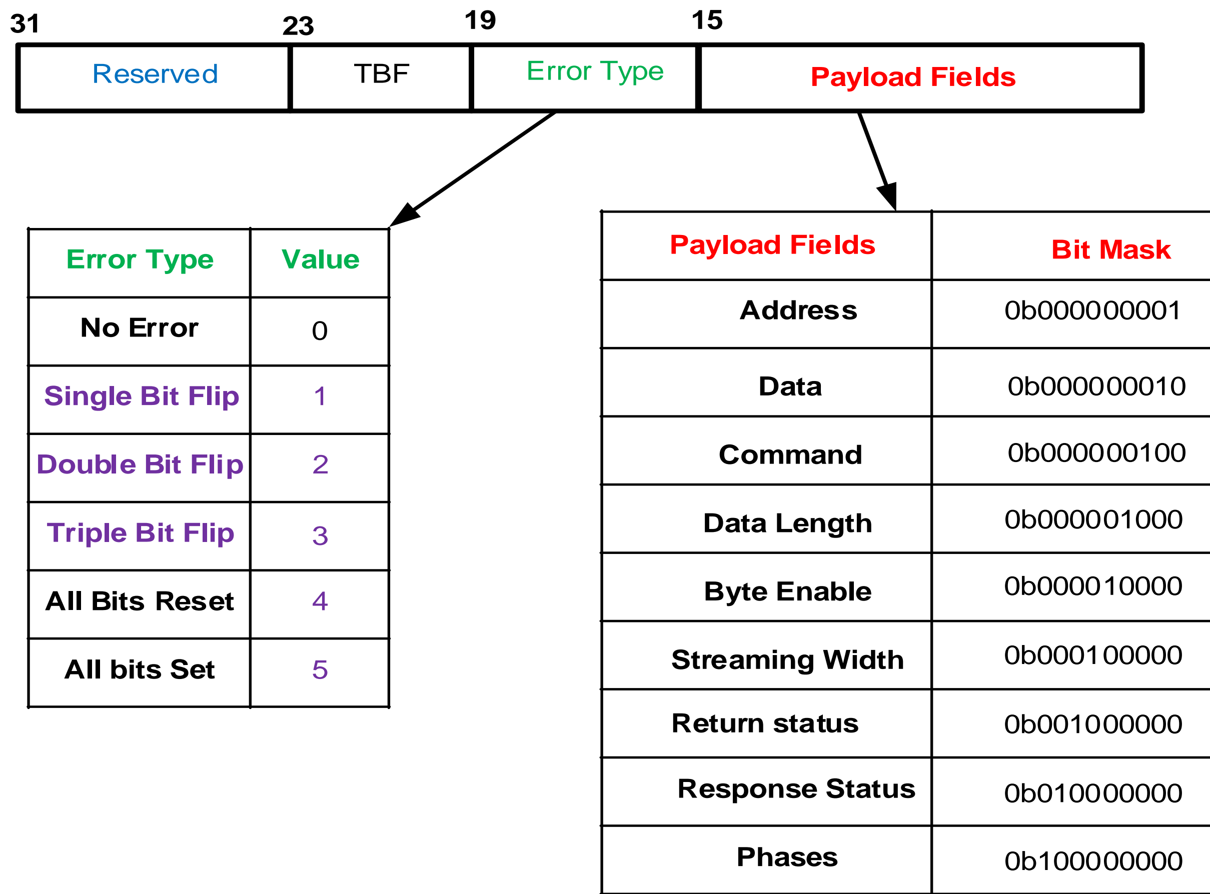


Figure 90: Fault Class Information

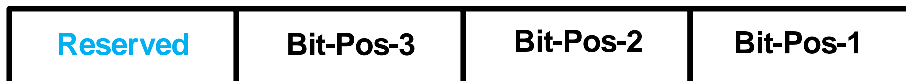


Figure 91: Bit Position Information

| No. | Type of Fault | Bit-Pos-3 | Bit-Pos-2 | Bit-Pos-1 | TBF | Error Type | Payload-Fields |
|---|---|---|---|---|---|---|---|
| 1 | No Fault | X | X | X | X | X | 0bXXXX XXX0 0000 0000 |
| 2 | Single Bit Flip | X | X | 0 – 2^16 | X | 0001 | 0bXXXX XXX0 0000 0001 |
| 3 | Single Bit Flip | X | X | 0 – 2^16 | X | 0001 | 0bXXXX XXX0 0000 0010 |
| 4 | Double Bit Flip | X | 0 – 2^16 | 0 – 2^16 | X | 0010 | 0bXXXX XXX0 0000 0011 |
| 5 | Double Bit Flip | X | 0 – 2^16 | 0 – 2^16 | X | 0010 | 0bXXXX XXX0 0100 0100 |
| 6 | Triple Bit Flip | 0 – 2^16 | 0 – 2^16 | 0 – 2^16 | X | 0011 | 0bXXXX XXX0 0010 1001 |
| 7 | Triple Bit Flip | 0 – 2^16 | 0 – 2^16 | 0 – 2^16 | 0110 | 0011 | 0bXXXX XXX0 0001 0100 |
| 8 | Triple Bit Flip | 0 – 2^16 | 0 – 2^16 | 000..111 | X | 0011 | 0bXXXX XXX0 0000 0001 |
| 9 | Reset all bits | X | X | X | X | 0100 | 0bXXXX XXXX XXXX 0001 |
| 10 | Set all bits | X | X | X | X | 0101 | 0bXXXX XXXX XXXX 0101 |

X -> don't care

**Figure 92: Fault Injection Examples**

This method makes it possible to combine up to three different types of errors in a fault injection command. Figure 92 shows a series of different fault injection possibilities. For example: at number 4, a fault injection command contains two errors (bit-flips), a fault is injected to the bit position 1 (bit-pos-1) of address, and another is in bit-position 2 (bit-pos-2) of data. Another example at number 7, fault information contains three errors (bit-flips), in which bit-pos-1 and bit-pos-2 corresponds to two faults in payload field '*data*' and bit-pos-3 corresponds to a fault in payload field '*byte enable*'. This information is embedded in the TBF = **0110**, where value of two LSB bits corresponds to '*data*' payload field and value of other two bits corresponds to '*byte enable*' payload field. Number 9 and 10 will reset and set all bits of a particular field(s).

# 4.6 Multi-Domain Virtual Platform

## 4.6.1 Preparing the Simulink Models for Fault Injection

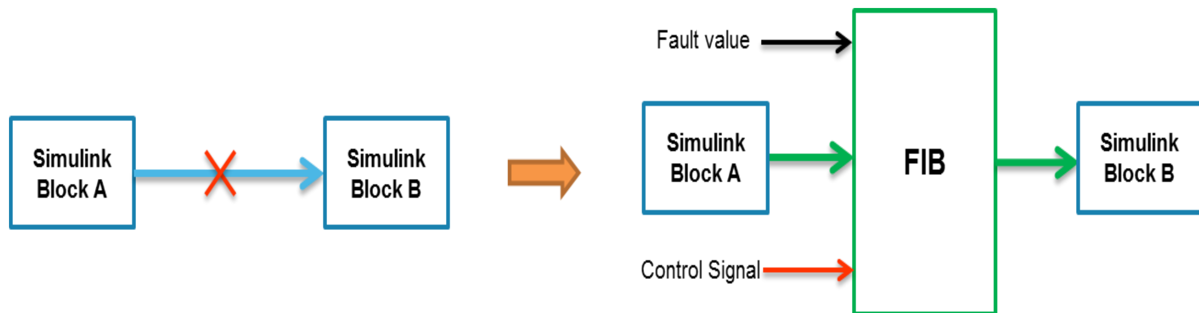


**Figure 93: Normal Line**

The principal idea is to extend the Simulink model by adding additional blocks to provide a mechanism to inject fault values. This is achieved by providing a possibility to insert a fault value and also a control mechanism to select either of them. Finally, control and fault signals are externalized as input ports in a top-level system, so that the values can be injected into the system.

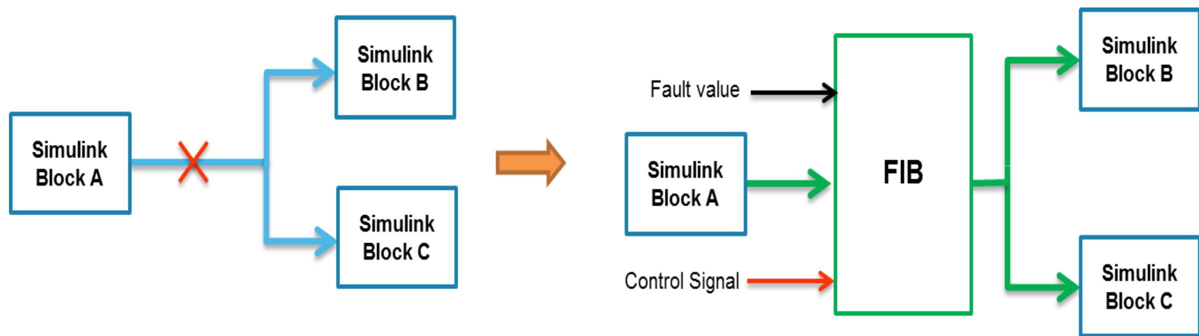


**Figure 94: Branched Line**

A new fault injection block (FIB) is inserted for each line of interest, which simulates the effect of faults. The output of FIB is equal to the actual value (non-faulty value), when the fault-injection is not activated. A simple example is shown in Figure 93, in which a fiblock is inserted between source and destination Simulink blocks, A and B respectively. For a branched line as shown in Figure 94, a FIB can be either inserted close to source or

destination blocks. In order to avoid an additional FIB, it is inserted close to Simulink source block A. A FIB is nothing but AFM (refer 4.3.2 Abstract Fault Models and Fault Interface Design).

### 4.6.2 Simulink Model Integration

Once the Simulink model has been adapted by adding appropriate FIB's for fault injection, it is imported into a virtual platform. In this section, different solutions for integrating Simulink models into a virtual platform are discussed.

#### 4.6.2.1 Virtualizer-Simulink Interface (VSI)

The third-party tools support package of Synopsys-Virtualizer provides connector blocks that connects to a signal in the Simulink simulation model on one side and interface to SystemC ports on the other side as shown in Figure 95. It shows a small Simulink system with one input and one output. Both signals are routed to connector blocks.

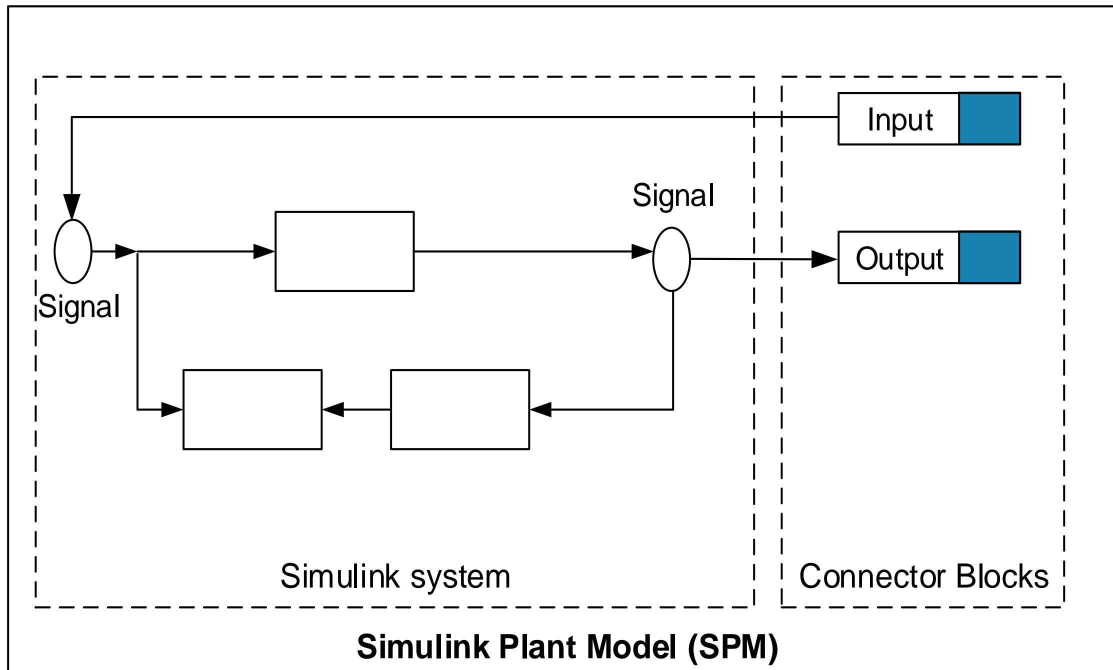


**Figure 95: Simulink System with Connector Blocks to Interface to SystemC**

- Co-simulation: The SPM and the SystemC simulation run in separate processes and communicate with each other by means of interprocess communication (IPC). The advantages of this flow are short turnaround times and full availability features of Simulink. The third-party integration package includes a generic connector block library for Virtualizer to interface a SystemC platform with Simulink. The first flow is to use these generic blocks and instantiate them for every corresponding connector block/module on the Simulink side. In the co-simulation flow, both simulators run in their own process. The name of the connector blocks must be unique within the Simulink model. The connector blocks transport data from the interprocess communication layer to the connector blocks on Simulink side and the SystemC model. In the Co-simulation flow, Simulink acts as the master and SystemC is a slave.
- Export: In this flow, the SPM is exported into a SystemC peripheral using Simulink Coder. The resulting set of header files and libraries is linked to the SystemC simulation. The resulting simulation runs in a single process without IPC. Advantage of this flow is the higher simulation speed compared to the co-simulation flow.

The interface to the SystemC simulation is done by connector blocks that wrap the Simulink protocol to the SystemC counterparts. The export flow adds registers (fifo) between the Simulink system and the connector blocks and creates a library with a SystemC interface as shown in Figure 96. This Simulink SystemC peripheral is shown in the dotted-line box on the right side. The original Simulink scheduler is responsible for driving the simulation of the Simulink system with the fifo's being the boundary, while the SystemC scheduler drives the rest of the simulation. In this setup, the SystemC scheduler acts as the master, while Simulink is the slave. This is different from the co-simulation flow where Simulink is the master. With regard to the timing and synchronization, it must be explicitly performed, i.e., for instance, using an additional interface module between Simulink and SystemC.

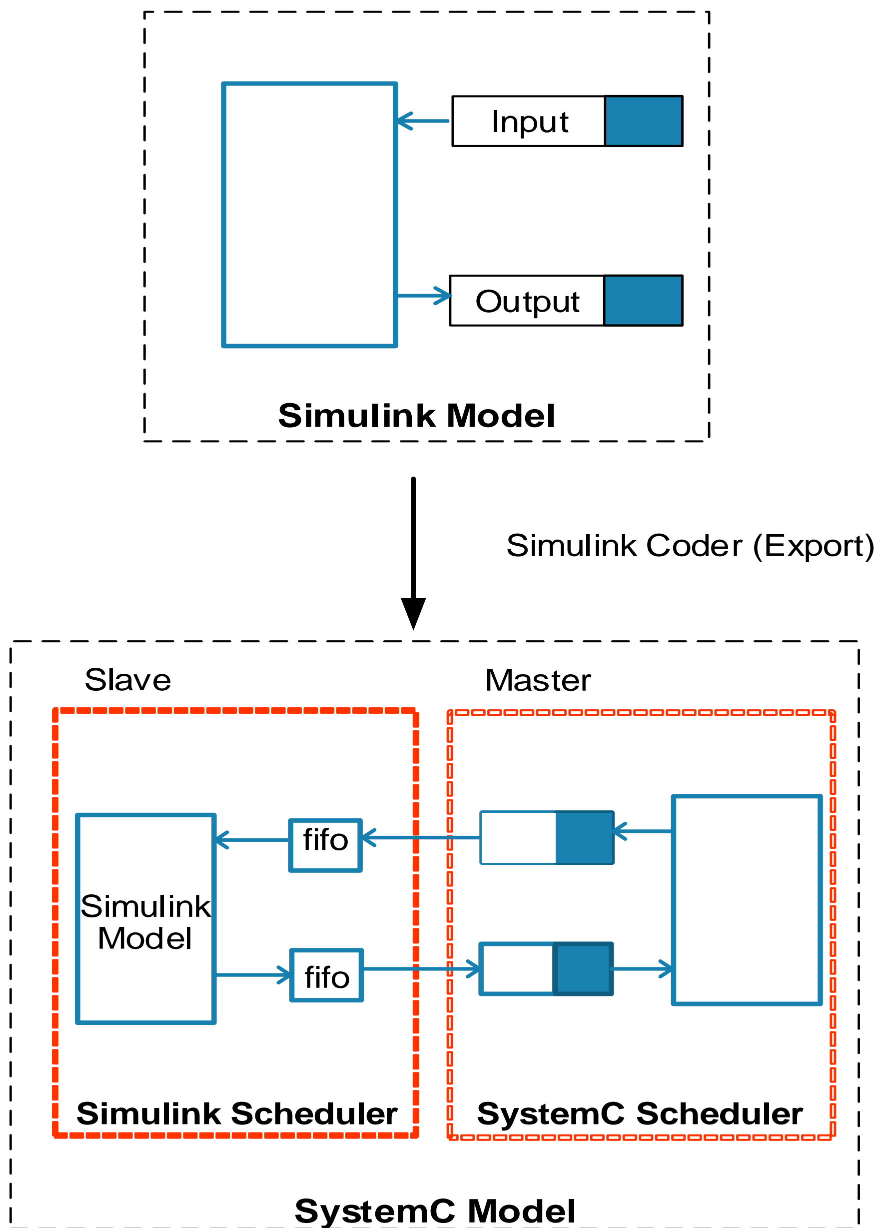


**Figure 96: Exporting Simulink to SystemC**

#### 4.6.2.2 Simulink Model as SystemC Module with TLM Standard Socket Interfaces

HDL Verifier™ (R2015b) from MathWorks [34] lets you create a SystemC Transaction Level Model (TLM) that can be executed in any OSCI-compatible TLM 2.0 environment, including a commercial virtual platform.

#### 4.6.2.3 Comparison of Solutions to Export Simulink Model to Virtual Platform

- ***As SystemC Module with Signal Interfaces***: The integration is very simple, since the script to import the Simulink model into virtual platform are already generated during the code generation, but the limitation is for each signal one needs an interface block, for larger Simulink modules increases the number of signals to interface.
- ***As SystemC Module with TLM Standard Socket Interfaces***: With this solution the generated module is very compact because it has only standard TLM2.0 socket as interface. But the importing into virtual platform and linking the module as static library after compiling has to be done manually and also an extra interface module has to be written to encode and decode payload information.

In our work, we have chosen the solution with exporting "*As SystemC module with TLM standard socket interfaces*", since the number of faults signals to interface are high.

### 4.6.3 Generic Fault Injector

Once the multi-domain parts of the system after fault modeling are integrated into a virtual platform, to perform fault injection tests i.e., to activate faults in heterogeneous system components, a generic mechanism is necessary to interact with the system during simulation run.

#### 4.6.3.1 Command Processor

Synopsys SystemC Modeling Library (SCML) is a TLM 2.0 compliant API library for creating and integrating user-defined SystemC TLM models into virtual prototypes. The SCML command processor [108] is an SCML object that connects the interactive debugger to the simulation to execute commands from the debugger during simulation. The command processor works locally within a specific component in a model, and can execute multiple commands, each with a specific set of arguments. Thus, the SCML command processor is also suitable for controlling the fault injection. The SCML commands were implemented using *SCML_COMMAND_PROCESSOR* and *SCML_ADD_COMMAND* macros.

**SCML_COMMAND_PROCESSOR:**

This macro is used to indicate which method should be called when the external debugger sends a command to the object. The macro takes exactly one argument: the name of the command handler method. The return type of this method should be *std::string* and it should accept one parameter of type *const std::vector<std::string> &*, which contains the command and its arguments, if any. It is guaranteed that the command handler method is only called for commands that have been declared using the SCML_ADD_COMMAND macro and for which the number of arguments lie within the bounds declared by the SCML_ADD_COMMAND macro. The string that is returned by the command handler method is displayed by the debugger.

**SCML_ADD_COMMAND:**

This macro is used to declare the commands that can be executed by the SCML Command Processor.

An example implementation of an SCML command '**faultx**' is as follows:

```
SC_MODULE(MyModule)
{
        SC_CTOR(MyModule) {
        // ProcessCommand method is invoked When a debugger sends a command to this
object
                SCML_COMMAND_PROCESSOR (ProcessCommand);
                SCML_ADD_COMMAND ("faultx", 0, 1, "allowed values<1,0>", "activate
faultx fault injection");
        }
```

```
};
// ProcessCommand function
std::string MyModule::ProcessCommand (const std::vector< std::string >& cmd)
{
        std::string command = cmd[0];
        if (command == "faultx ") {
                outMsg << faultx_faultinjection(cmd);
        }
};
// 'faultx_faultinjection' implementation
std::string MyModule::faultx_faultinjection (const std::vector< std::string >& cmd)
{
        std::string switch_val = cmd[1];
        if (switch_val == "1") {
        // activate faultx fault injection
        }
        else if (switch_val == "0") {
        // de-activate faultx fault injection
        }
        else  {
        // wrong input parameter
        }
}
```

### 4.6.4 Mapping Fault ID's

Each fault signal is assigned an identifier after fault modeling and these identifiers are mapped to fault ID's from the fault scenarios (see 3.3 Exemplary Faults in MCS). For example, **HW_Analog_001** is mapped to *FAULT_SIG_001* whereas **HW_Analog_002** is mapped onto two signals *FAULT_SIG_001* and *FAULT_SIG_002*, each corresponding to one motor terminal.

## 4.7 Automated Fault Injection Tests and Results Evaluation

### 4.7.1 Motivation

Virtual stress tests or simulated system tests with fault injection are performed in hardware and software at an early stage to provide comprehensive statements about system behaviour under fault conditions and also to prove that the entire system behaviour is always safe also in the event of errors. Hence, there are many simulation results with and without fault injection,

to compare. The time required for these comparisons is very high but with an automated evaluation it can be significantly reduced. In the case of an automated evaluation, the simulation results with fault injection are compared with reference values from simulations without fault injection.

### 4.7.2 Automated Comparison of the Simulation Results of Fault Injection Tests

A procedure for the automated comparison of simulation results with fault injection and without fault injection is described in the following section. It is also available in the form of a C++ program as a prototype as well as tcl package. The implementation details are presented in the next chapter. This prototype is suitable to be integrated in different script environments for an automated comparison of simulation results. To be able to compare the result of a simulation with fault injection (i.e., a measurement curve) with the result of a simulation without fault injection (i.e., with a reference curve), the following actions are to be performed:

1. Read the reference curve
2. Reading the measurement curve

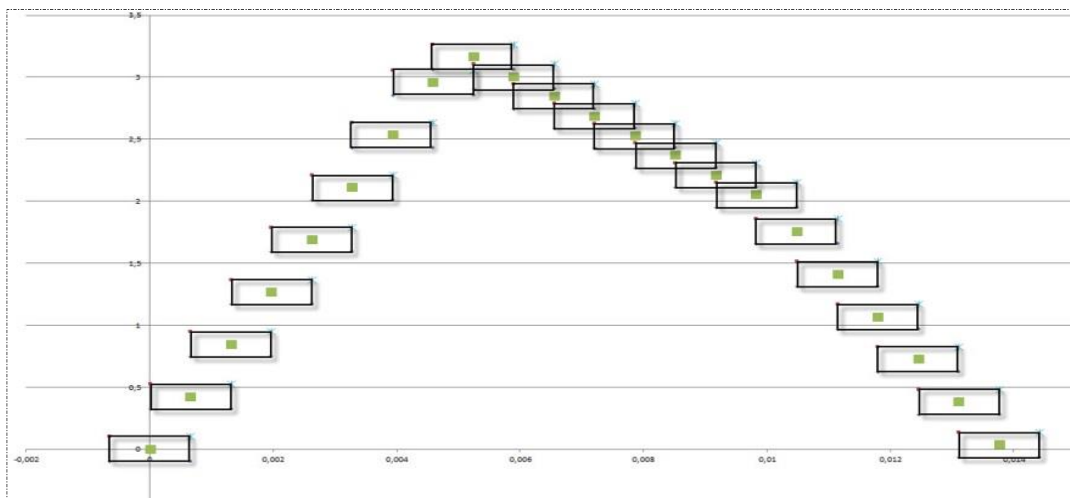

**Figure 97: Example of an interpolated reference curve with the assigned tolerance rectangles**

3. Replace the reference curve with an interpolated reference curve in which equidistant time points are assigned to all reference points.
4. Replace the readout curve with an interpolated measuring curve in which equidistant time points are assigned to all measuring points
5. A tolerance rectangle as shown in Figure 97 is assigned to each point of the interpolated reference curve.
6. Parameterize the size of the tolerance rectangle.
7. The tolerance curve of the interpolated reference curve is used to determine whether the reference curve (i.e., the simulation results without error injection) and the measurement curve (i.e., the simulation results with error injection) differ or whether the fault injection has not changed the behaviour. For this purpose, a check is made for each point of the interpolated measuring curve as to whether the measuring point lies within one of the tolerance rectangles of the interpolated reference curve.
8. Determination of the percentage of tolerance rectangles which contain at least one measuring point of the interpolated measuring curve. As long as this percentage is greater than or equal to a predetermined target percentage, the reference curve and the measurement curve are considered to be the same curves, i.e., the fault injection did not change the system behaviour.

### 4.7.3 Post-Processing Scripting

It allows one to record and extract the simulation results for analysis. In the scripting API, there is a sequence of steps you take in working with your simulation results:

1. First, choose which simulation run you are interested in.
2. Then choose which recorded result to work with, like a trace of a port or some statistics on a bus.
3. Then you can either export the actual data records of that result to external files or view using VP explorer GUI.

# 5 Implementation Aspects and Results

In this chapter we present the implementation details of the industrial motor control application and also the results of automated fault inject tests presented in section 4.7 Automated Fault Injection Tests and Results Evaluation.

## 5.1 Industrial Motor Control Application

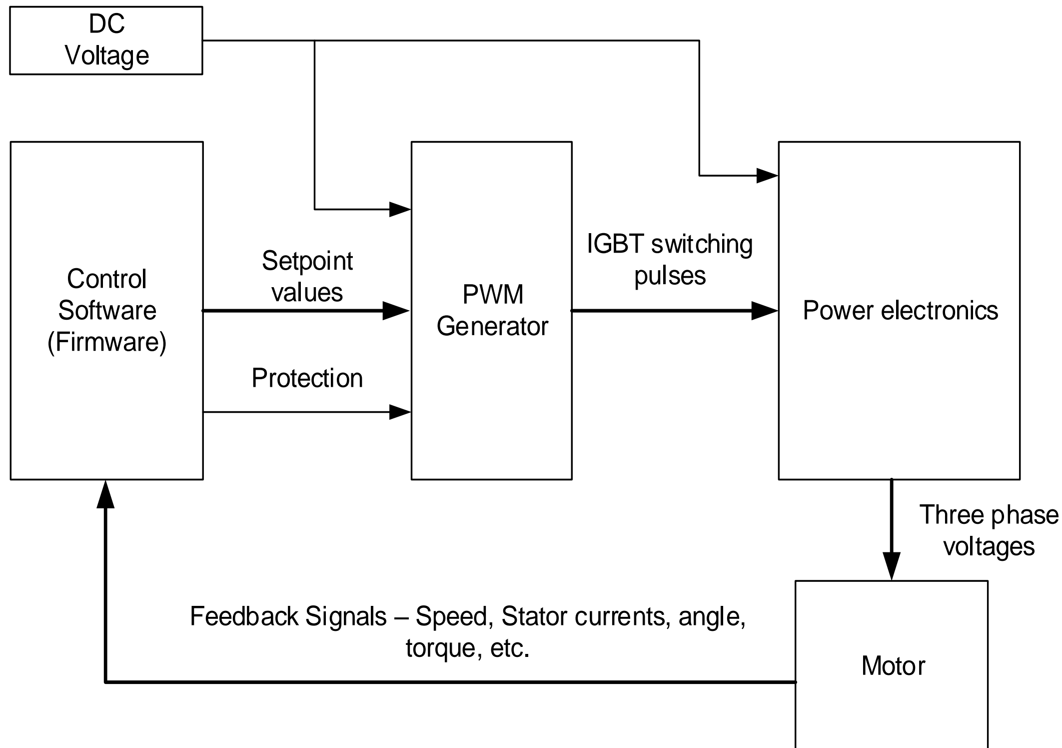


**Figure 98: Motor Control Application Block Diagram**

Block diagram of closed-loop motor control application is shown in Figure 98. The mains components include control software, PWM generator, power electronics and motor.

### 5.1.1 Control Software

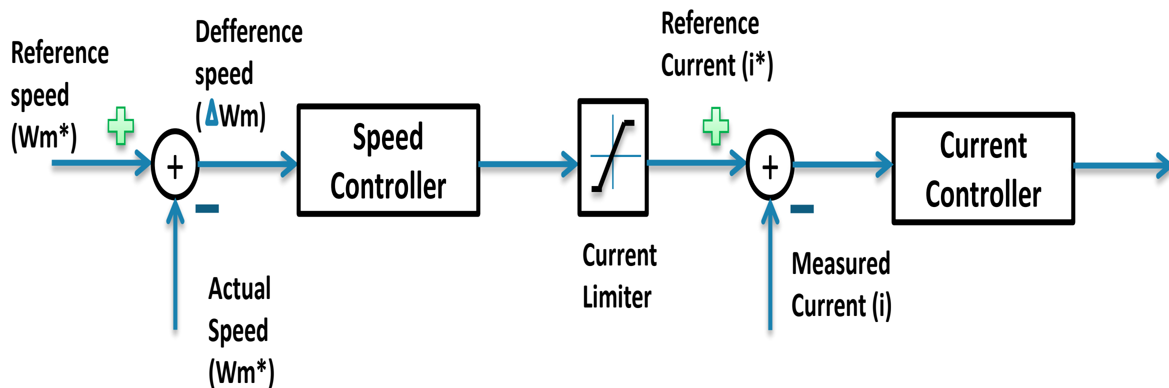

**Figure 99: Control Software**

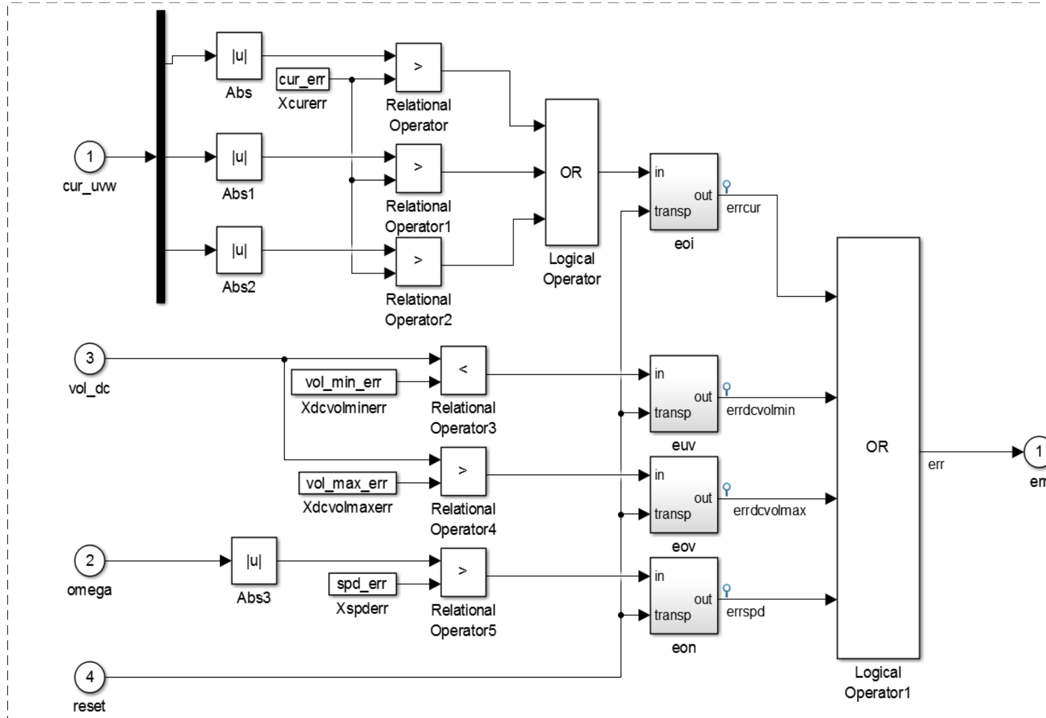


**Figure 100: Protection Mechanisms**

The control software consists of speed control algorithm and protection mechanisms. The primary function of the control software is the speed control of the motor. The control algorithm implemented in the control software consists of two control loops as shown in Figure 99, an internal current control and an external speed control. The motor current is directly proportional to the motor torque. It is correspondingly increased by the control at a necessary acceleration or an increased load on the motor shaft in response to a deviation of the actual from the desired speed, and vice versa. The protection mechanisms (shown in Figure 100) are based on monitoring motor currents, dc-voltage (minimum and maximum) and speed (maximum).

## 5.1.2 PWM Generator

The computation of the three duty-cycles according to the space-vector modulation is as shown in Figure 101. Balancing of the zero-pointers is performed by subtracting the middle-

value of the extrema of the three phases and then by division with the DC voltage. This is followed by a modification of the duty cycle corresponding to the lock time (Tlock) and a conversion into PWM signals by comparison with a reference with PWM frequency (frq_pwm). The enable signal 'en' at the bottom is used by the control software to release or block the pulses.

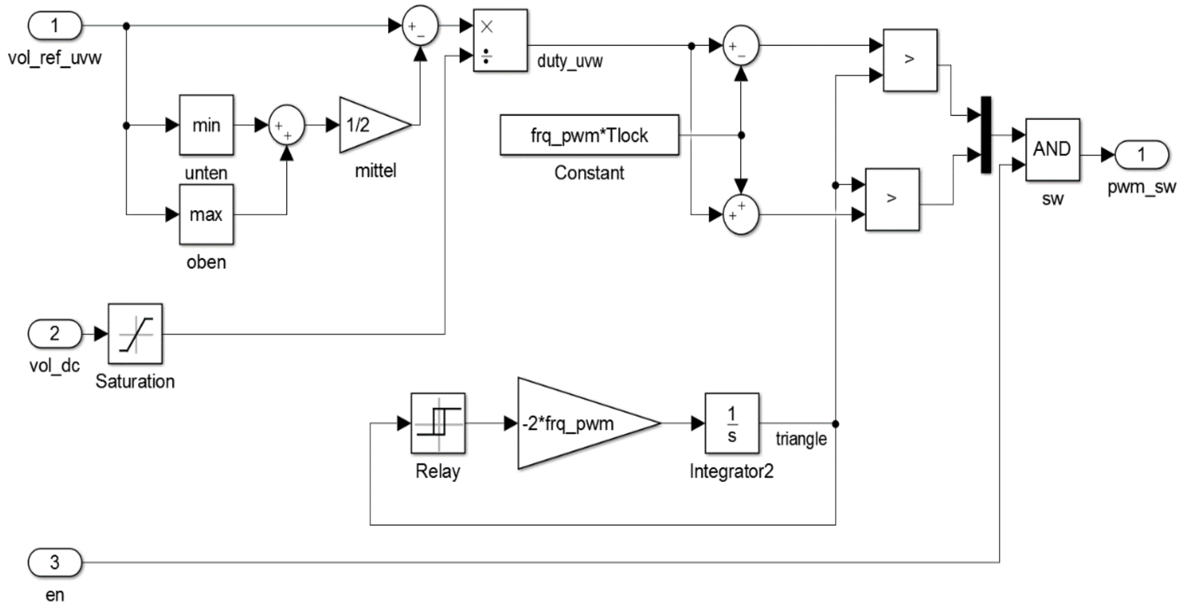


**Figure 101: PWM Generator**

## 5.1.3 Simscape Power electronics

The power electronics part consists of LC filter, DC link capacitor, and the inverter modules as shown in Figure 102. The small circles are electrical ports for which node and mesh equations are set up. The feed (left) is controlled by a controlled voltage source which is controlled by a physical signal. A converter converts the Simulink signal vol_bat into this physical signal. Various sensors provide the voltages and currents for the Simulink outputs. The intermediate circuit receives its ground reference via two resistors. The Solver block allows settings for the Simscape Solver for this network. The semiconductors are modeled with finite on and off resistors. This results in a rigid equation system, which requires a corresponding solver thereby increasing the computation time.

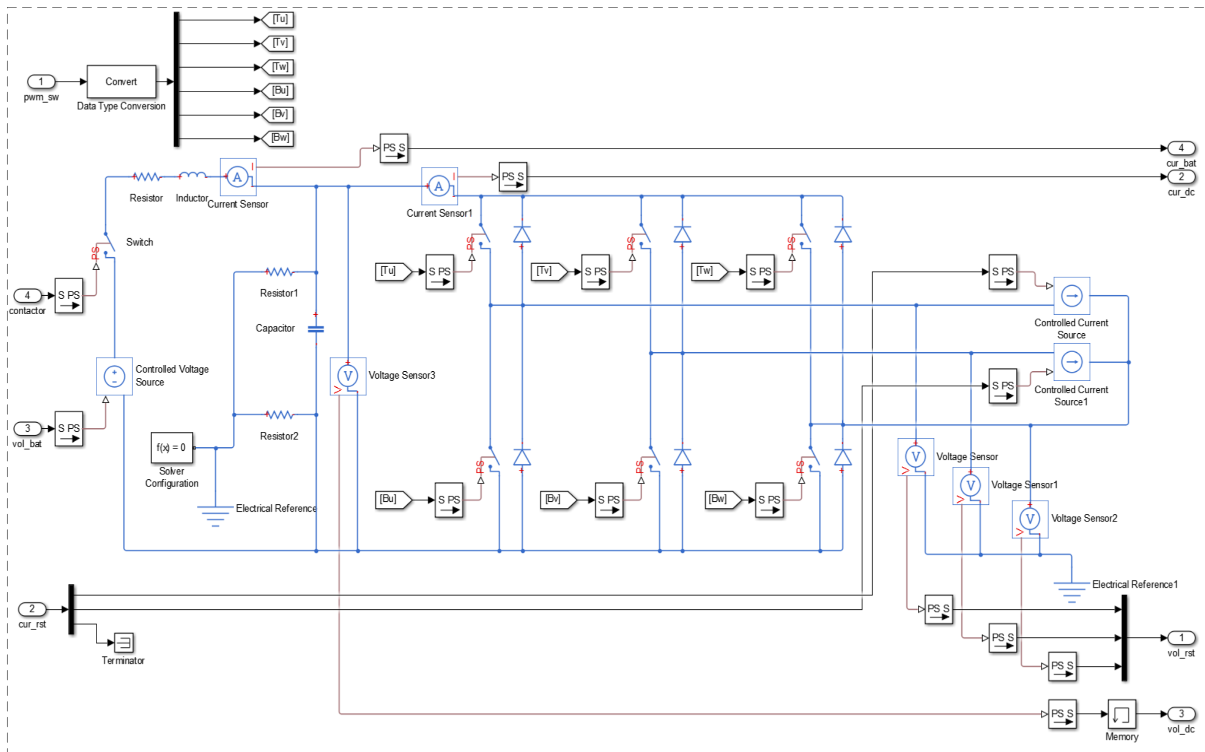


**Figure 102: Power electronics (Simscape)**

## 5.1.4 Simulink Power electronics

For the virtual prototyping application, the Simscape power electronics module is not appropriate, since the speed of simulation is a major factor. The Simulink power electronics module is abstracted into sub-components LC filter, DC link capacitor, and the inverter modules as shown in Figure 103.

LC filter along with DC-link is shown in Figure 104. There is an integrator for the two state variables coil current and capacitor voltage, which integrates the voltage across the coil or the current into the capacitor. The opening of the battery contactor is modeled by resetting the coil current to zero. The switch allows bridging the LC filter so that it can also be simulated with a constant DC voltage.

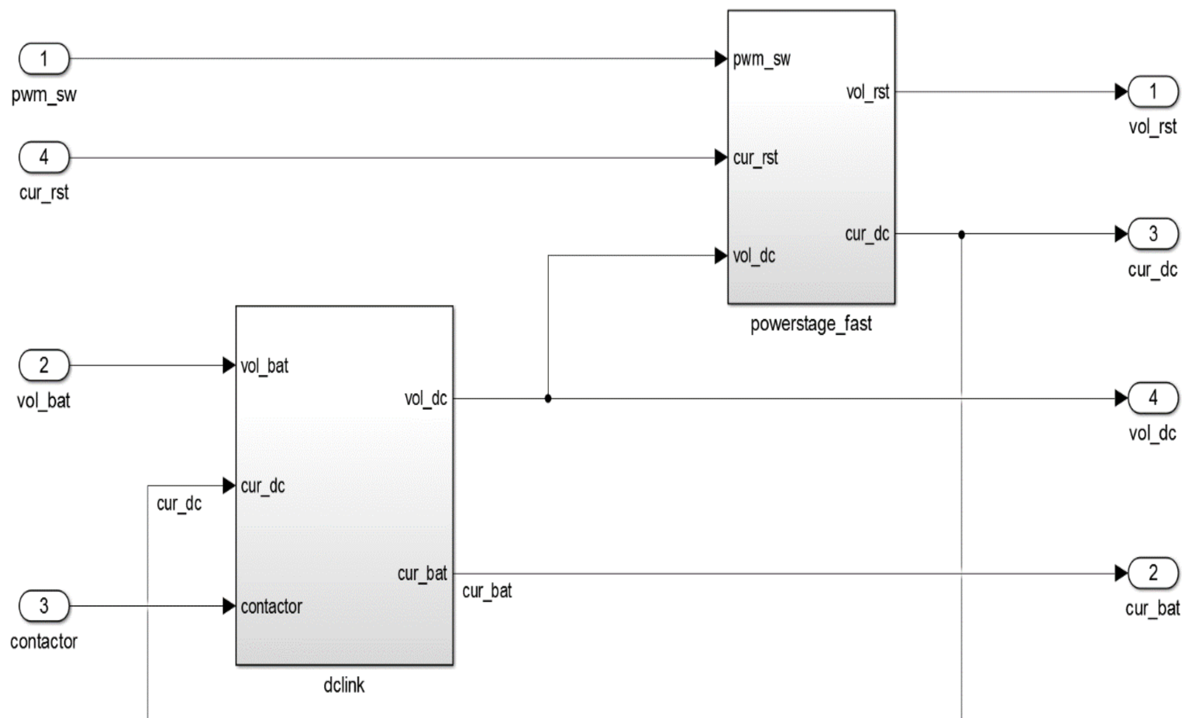


**Figure 103: LC filter, DC link and Inverter (Simulink)**

The inverter module is shown in Figure 105. The switch (switch2) switches the output terminals to the upper or lower DC bar, where the reference potential is placed in the center. Before this, the forward voltage of the semiconductors is subtracted. This is determined as a characteristic curve depending on the phase current. The positive part of the characteristic curve corresponds to the transmission behaviour of the IGBT and the negative part is of the diode. Therefore the characteristic curve is reversed to the phases connected downwards.

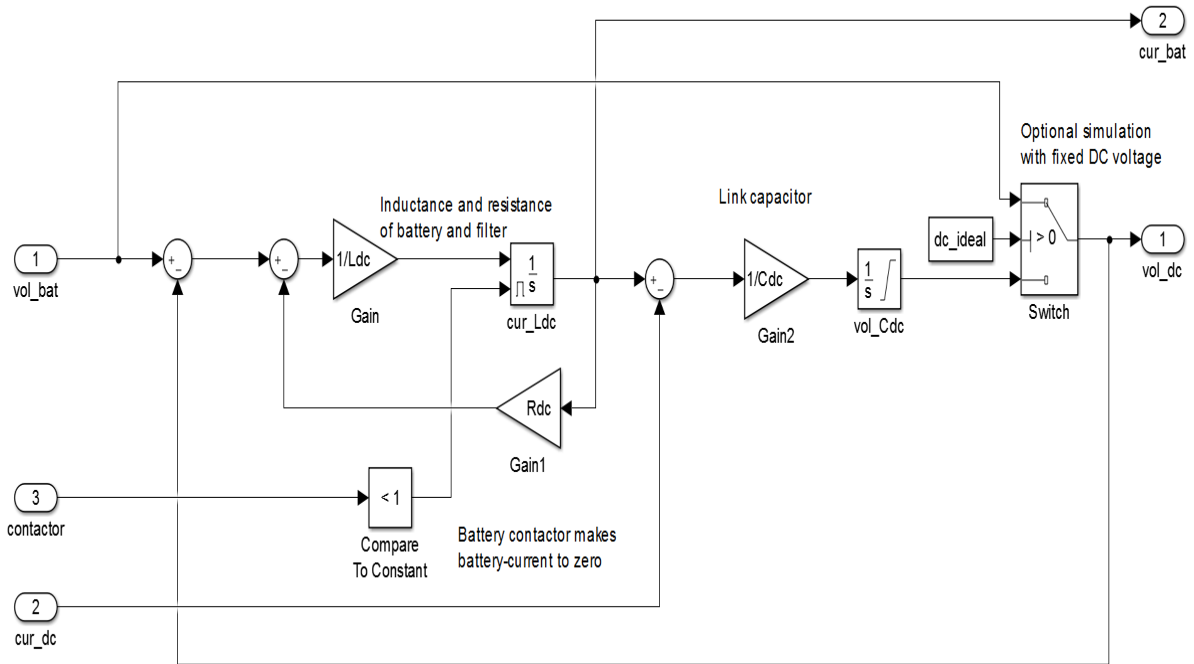


**Figure 104: LC filter + DC-link (Simulink)**

The DC current results from the sum of the currents of the up-connected phases or the negative sum of the down-connected phases, thus it is equal to half the total sum. The switching state of the phases is uniquely determined in the case of active switching signals, i.e. in this case, the flip-flop is set directly. During the interlocking time or in the case of pulse interrupts, the state is determined by the current: the upper diode conducts with a negative current, the lower diode when the current is positive.

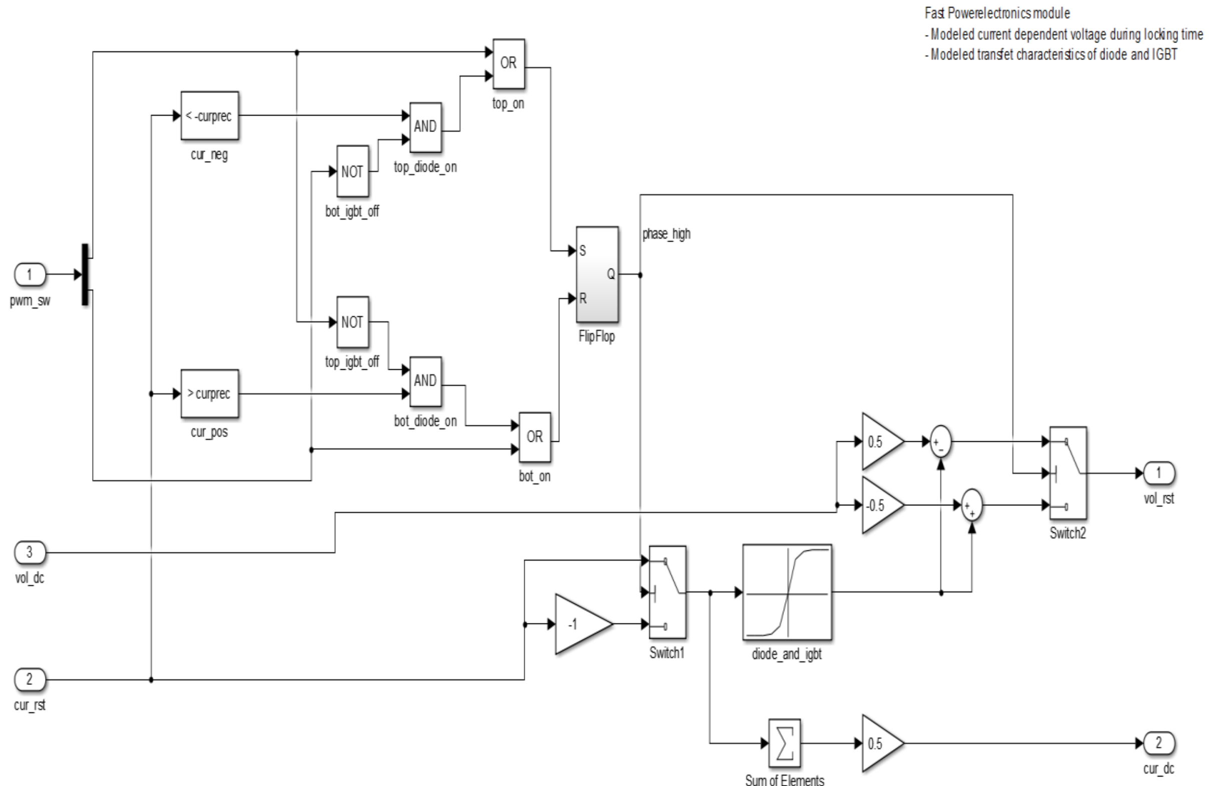


**Figure 105: Inverter module (Simulink)**

## 5.1.5 Motor Module

The motor model shown in Figure 106 simulates a permanent-magnet synchronous motor with flow characteristic fields. First, the terminal variables are transformed into rotor-oriented dq coordinates. The mode of operation is dictated by the sign of the torque (positive for motor mode, negative for generator mode). In dq-coordinates, the flux is formed by integrators from the voltage (see Figure 107). In the case of characteristic fields, the current is stored as a function of the flux, and the torque is determined from the cross product of flux and current (see Figure 108). Trailing losses (iron and friction) are not considered in this model.

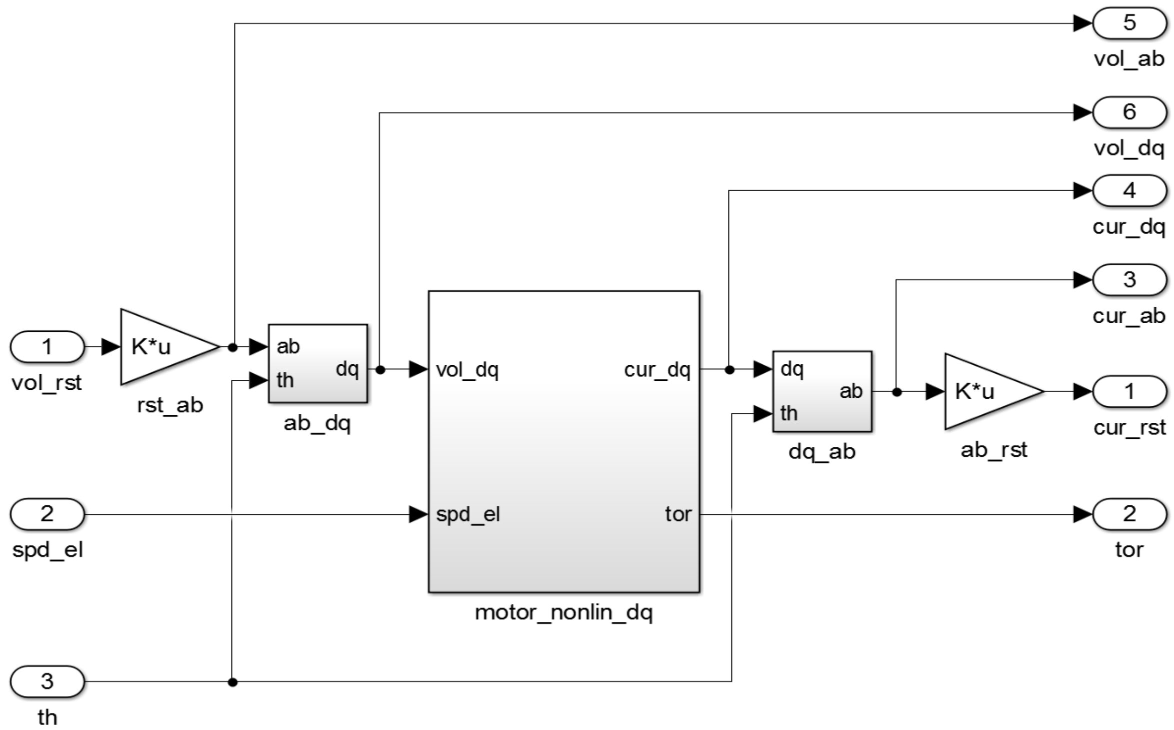


**Figure 106: Synchronous Motor**

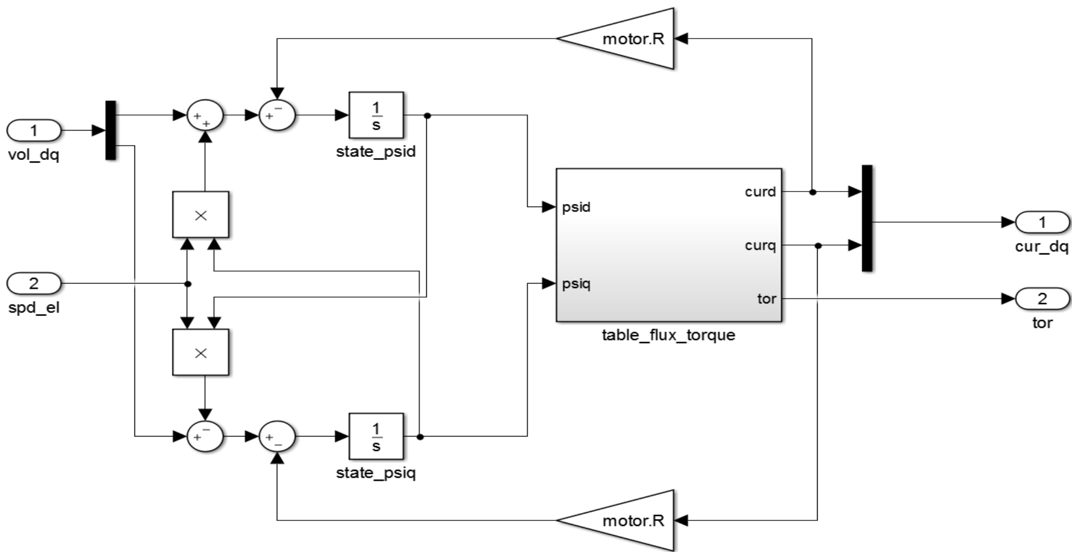


**Figure 107: Motor in dq-coordinates**

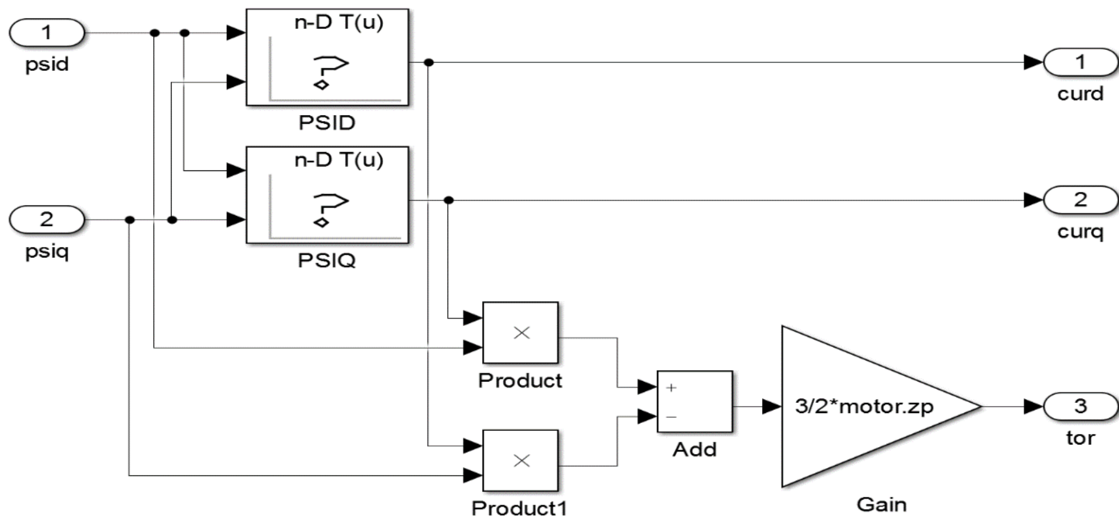


Figure 108: Flux-Torque table

## 5.2 Simulation Performance Test

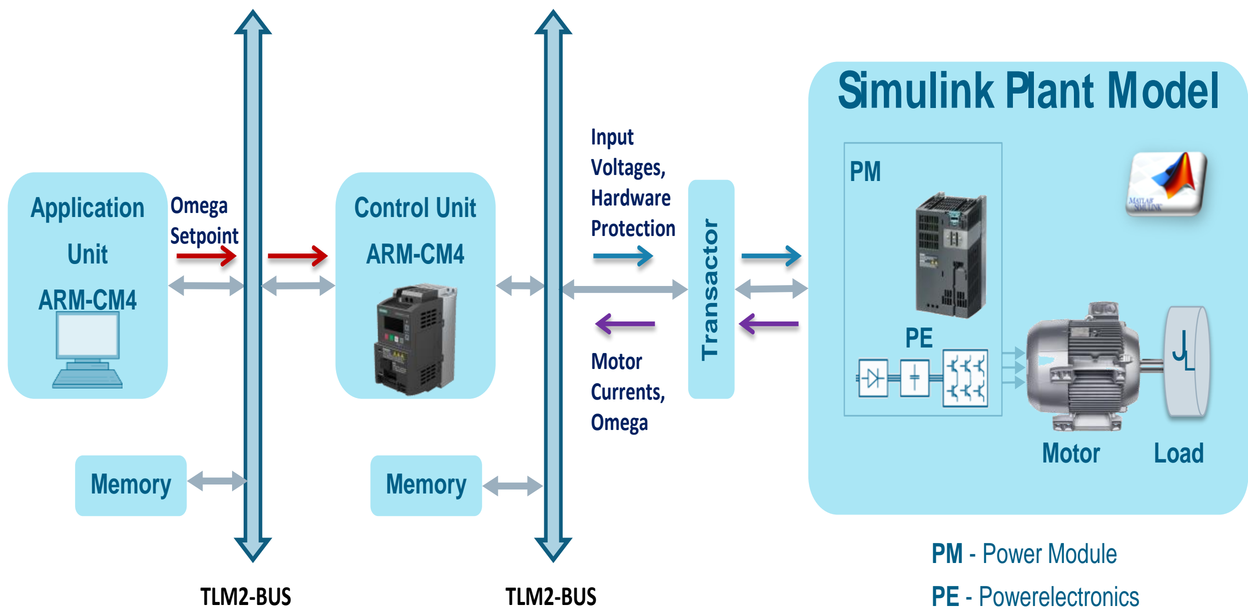


Figure 109: Simulation Platform

In this test, we compare the simulation performance of Simulink (dataflow) and Simscape (electrical) models inside a virtual platform. The dataflow and electrical power electronics of the setup is already discussed in the subsection Motor Control Application.

The simulation platform for motor control application is shown in Figure 109. It has been implemented in the Synopsys Virtualizer and contains models for the components application unit (AU), control unit (CU), power module (PM) [pulse width modulator (PWM) + power electronics (PE)] and motor along with load. The simulation platform here includes models for an AU for controlling the application and a virtual drive platform (VDP) for controlling the motor. VDP includes CU, power electronics along with motor module and load. AU and CU are built around ARM fast model cortex-m4 processor.

The AU controls the entire process by sending instructions to the motor to make it rotate at a certain speed. In the case of problems which occur and are detected in the application, the AU sends commands to the control of the drive (to the VDP) to stop the motor. The ARM CU runs the control software algorithms to control the motor. The transactor is a generic adapter component which decodes the information received on its inputs and forwards it.

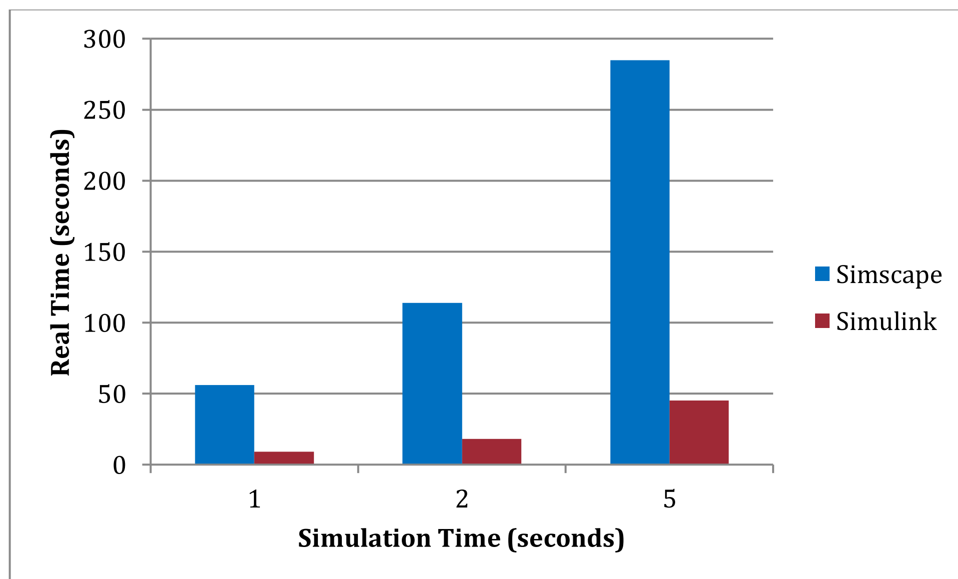


**Figure 110: Simulation Performance**

The dataflow model is integrated into the virtual platform using the HDL Verifier as a SystemC / TLM component (Simulink Plant Model). The electrical model is integrated into the virtual platform by C-code generation of Simscape model and writing an additional SystemC wrapper. The simulation performance graph is shown in Figure 110. The real-time factor (real-time / simulation time) of Simulink and Simscape are close to 10 and 60 respectively which means to simulate 1 second of real time, it takes 10 seconds for Simulink whereas 60 seconds for Simscape. Note that only power electronics part of the Simulink and Simscape models is different. We conclude for this experiment a significant performance is gained in case of Simulink (data-flow) model.

## 5.3 Automated Fault Injection Tests and Results Evaluation

The implementation details of the concept for automatic evaluation of results of the fault-effect simulations described in previous chapter (refer 4.7 Automated Fault Injection Tests and Results Evaluation) are presented here.

### 5.3.1 Automated Simulations using Tcl Scripts

VP explorer [71] [109] from Synopsys provides TCL (Tool Command Language) [110] interface to interact and control the virtual platform simulation. For e.g., using TCL commands one can create breakpoint at particular simulation time point and also attach callbacks when a breakpoint is hit. There are various functions one can use to write automated tests.

### 5.3.2 Configuration Strategy

To analyse different properties of your simulation, VP Explorer allows you to connect and configure monitors to different blocks in the system. It depends on the simulation which monitors can be connected to which blocks. You can attach monitors to your platform to probe aspects of your platform's state in order to trace or analyse it. Monitors can be attached at any point after the end of elaboration, but can only be attached to points in your platform

that have been instrumented for this purpose. The Synopsys SystemC simulation kernel and Synopsys IP models have been instrumented to support monitors.

### 5.3.3 Simulation Control Scripting

VP Explorer offers various procedures related to controlling the simulation such as start/stop/pause simulation, creating breakpoints, attach callbacks to breakpoints, read simulation information etc. It also offers many ways to interact with the blocks that make up your simulation. For e.g., model, which corresponds to a *sc_module*, *sc_port*, *sc_export*, or *scml_memory* instance in the simulation. The blocks can be addressed by using means of a hierarchical instance names, and specific commands can be sent to these blocks.

### 5.3.4 Automated Fault Injection Tests

Figure 111 shows the overview of the main script in which the following actions are performed:

1. Read input files – 'trace signals', 'fault injection tests' and 'script configuration parameters'. Format for fault injection tests is shown in **Fehler! Verweisquelle konnte nicht gefunden werden.** in which each 'test number' corresponding to one simulation run and each row corresponds to a fault. Multiple faults can be activated using the same test number for the multiple rows e.g., *test number 2*. Each fault has 'fault id', fault interface (see Figure 25: Control Interface and Fault Type ), 'injection time – when to inject fault' and 'fault duration – how long fault is active'. The script configuration parameters are mainly to make the script generic i.e., independent of the simulation platform.
2. Configuration: setup monitors for trace signals, load curve comparison tcl package
3. Reference (golden) simulation run (without fault injection) and store result
4. Execute each fault injection test as separate simulation and store the result
5. Perform the curve comparison of each fault injection test with reference simulation and store the results
6. Show the output signal waveforms (using a GUI such as '*gnuplot*')

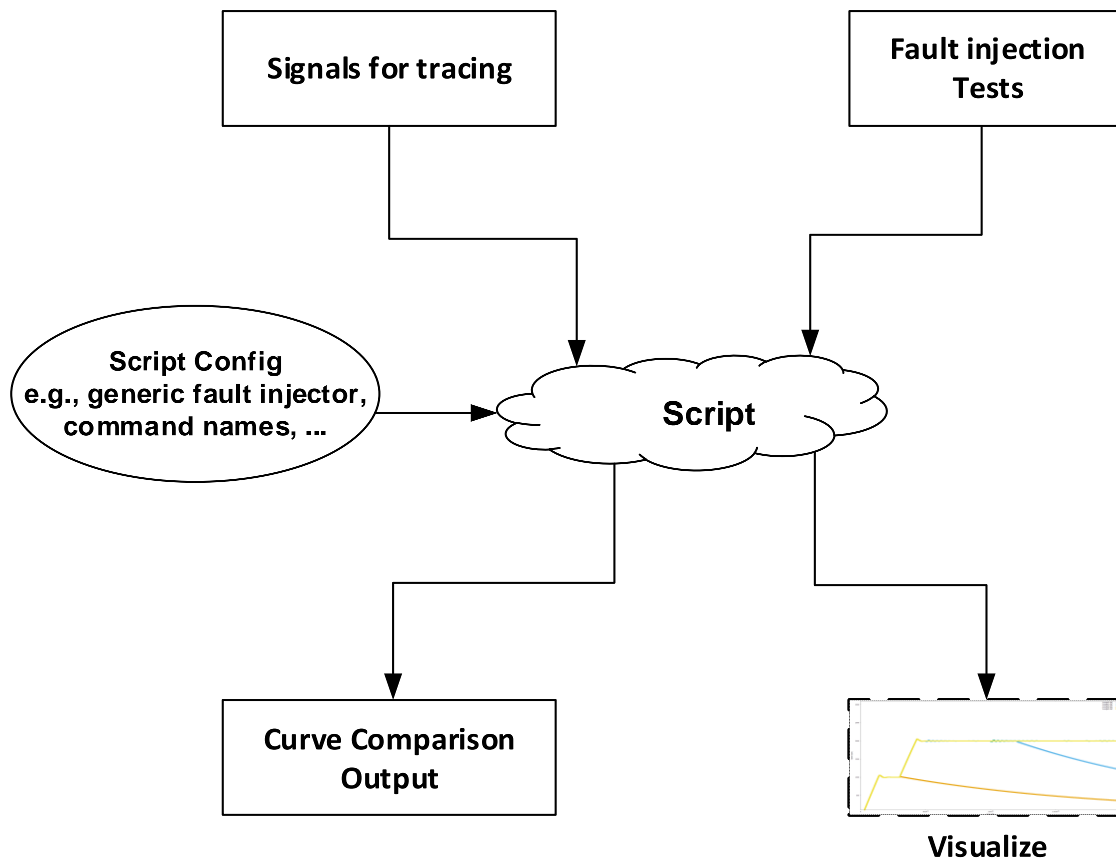


**Figure 111: TCL Script**

### 5.3.5 Example

The virtual drive platform (VDP) is shown in Figure 112. The generic fault injector module is used for fault injection by defining and implementing necessary commands. The commands modify the '*fault memory*' inside the Simulink interface module for 'analog fault injection' in which each location inside the '*fault memory*' is mapped to fault signals in the Simulink plant model (SPM). The faults in the TLM transaction for 'digital fault injection' are introduced by modifying the transaction inside 'tlm_injectable_target_socket'.

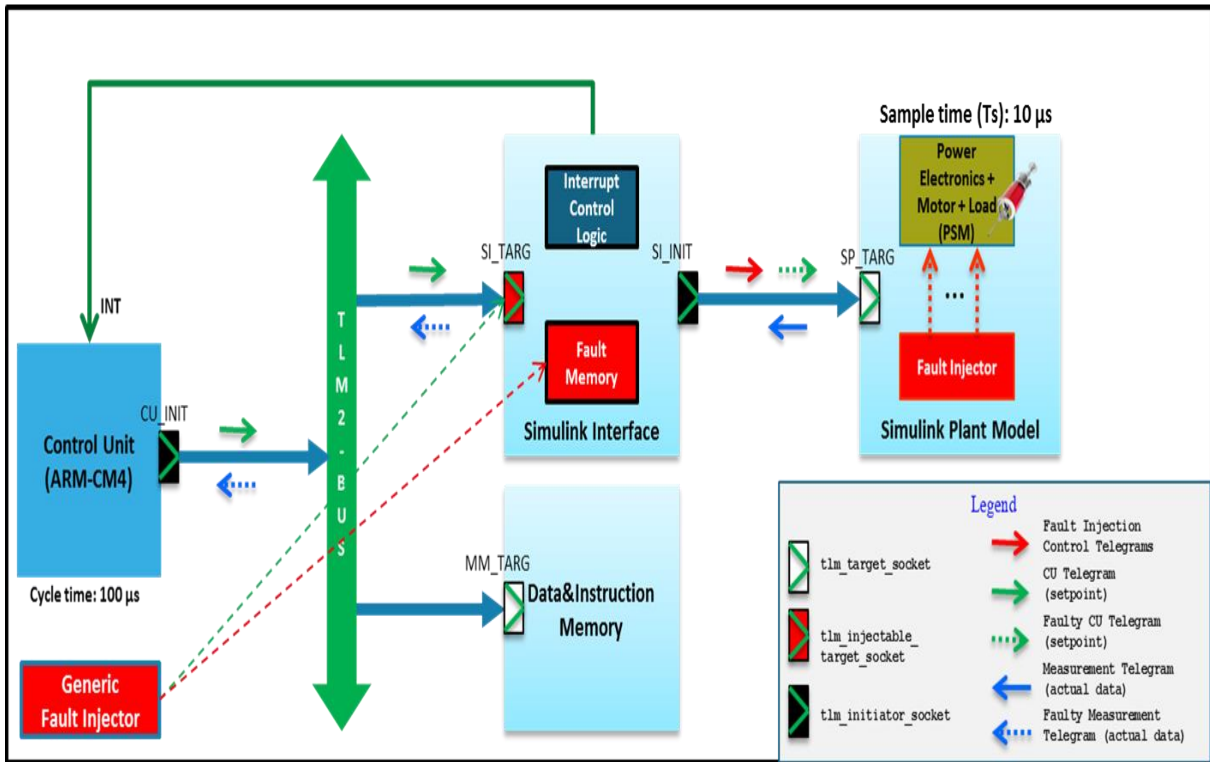


**Figure 112: Virtual Drive Platform for Fault Simulation**

| Test Number | Fault ID | Fault Interface | Injection Time (ms) | Fault Duration (ms) |
|---|---|---|---|---|
| 1 | FAULT_SIG_001 | 1 | 1000 | 500 |
| 2 | FAULT_SIG_002 | 1 | 500 | 1000 |
| 2 | FAULT_SIG_003 | 1 | 1200 | 500 |
| 3 | FAULT_SIG_002 | 1 | 300 | 1000 |
| 4 | FAULT_SIG_002 | 0x0A020002 | 1000 | 10 |

**Figure 113: Format for Fault Injection Tests**

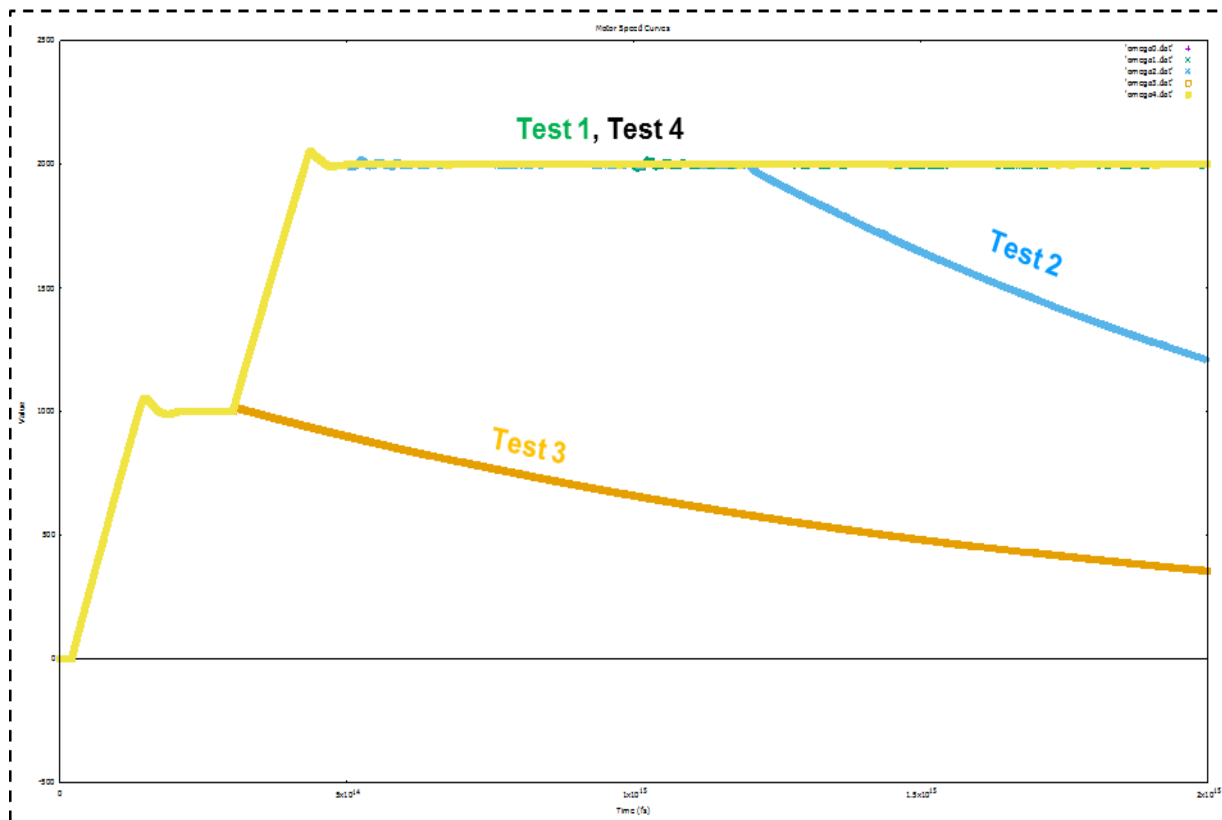


**Figure 114: Motor Speed Output**

Figure 113 shows an example input for fault injection tests for analog faults i.e., inside Simulink plant model (SPM), which includes a total of four tests with each row, corresponds to a single fault. Three of the tests have one fault except test 2 which has 2 faults. In test 4, a transient fault is activated at 1000 ms with fault interface value of 0x0a020002, which means in an interval (period) of 0x0a (10)*Ts, fault is active for 2*Ts (duration). 'Ts' is the sample time of Simulink module (analog parts).

The curve comparison output and the motor speed output signal are shown in Figure 114 and Figure 115 respectively. The 'max_matching_rate' will give you the percentage of matching points with comparison to reference curve (without fault injection). The tolerances 'delta_x' and 'delta_y' must be configured appropriately in order to interpret the results of curve comparison. In our example, the output curve of tests 1 and 4 with ~70% matching look very similar whereas tests 2 and 3 with ~63% and ~22% matching are clearly differentiable.

Hence, the tolerance rectangles must be carefully parameterized for using and interpreting the results of curve comparison.

| Test | Method | dalta_x | dalta_y | max_matching_rate | interpolation_num |
|---|---|---|---|---|---|
| 1 | 0 | 0.01 | 0.7 | 0.702 | 4000 |
| 2 | 0 | 0.01 | 0.7 | 0.629907476869 | 4000 |
| 3 | 0 | 0.01 | 0.7 | 0.215053763441 | 4000 |
| 4 | 0 | 0.01 | 0.7 | 0.70525 | 4000 |

**Figure 115: Curve Comparison Output**

### 5.3.6 Conclusion

The simulation results of automated fault injection tests are presented with an example which allows the comparison of simulations with and without fault injection. The automated fault injection framework provides the users configuration possibilities for preparing the tests. At the end, the results are visualized with the help of a plot tool for further analysis.

# 6 Motion Control System: Case Studies

When using virtual stress tests in the development of intelligent motion control systems, different system configurations must be considered, in which digital and analog system components or even virtual system parts and real hardware can be investigated in co-simulations depending on the task. The aim is to be able to prove that the reaction of the system to certain faults which occur within a system or which act on a system from the outside takes place safely. The system reaction includes hardware and software procedures for fault handling. For example, in virtual system tests, it can be examined whether the motion control system responds correctly to signals from the acceleration sensors for the condition monitoring of the motor or how faults of these components affect the overall system. Also, errors in the digital part, e.g. the control unit (such as the failure of a PROFINET, DRIVECLIQ or CAN transceiver or the reception of faulty signals via one of these interfaces) as well as the response of the entire motion control system to these errors. This makes virtual system integration possible, i.e., the simulative execution of extensive system tests on intelligent motion control systems already in development. The final system tests (HIL) on the real hardware, which cannot be omitted entirely for safety reasons, then usually only serve to confirm the correct function. These tests can then be carried out essentially with the same test sequences used in the virtual system test. Hence we developed various demonstrators for different purposes mentioned here.

# 6.1 Motor Condition Monitoring using Acceleration Sensor from RB

## 6.1.1 Setup

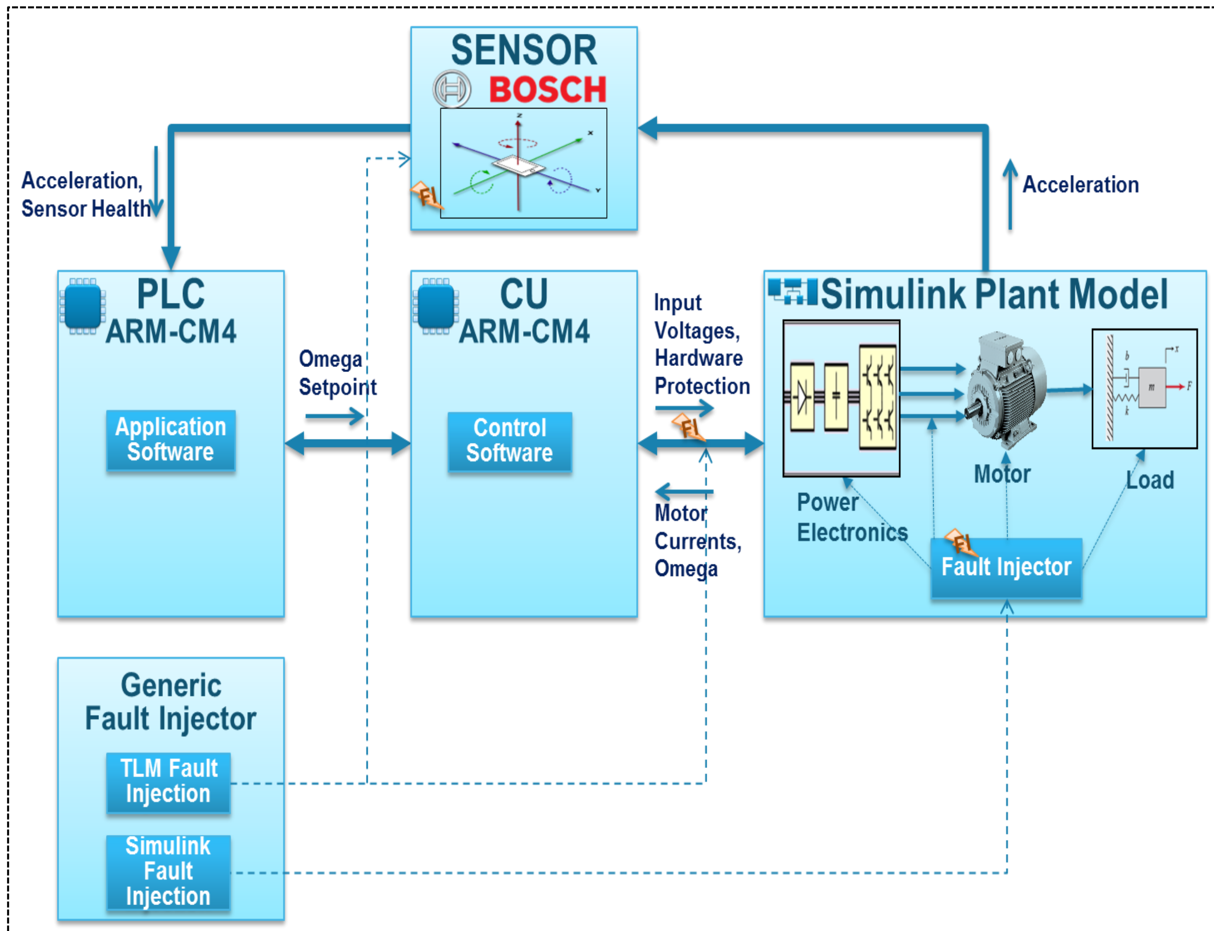


**Figure 116: Simulation Platform for Motor Condition Monitoring**

Simulation Platform for Condition Monitoring of motor using acceleration sensor is shown in Figure 116. The first case study discusses a sensor module from Bosch [111] which is integrated into the virtual platform. The sensor is used for precise monitoring of motor wear and tear using vibrations. The sensor measures up to 35 g with a resolution of 2 mg and the sampling frequency is 285 kHz. The sensor module takes the acceleration value as analog input and the output is sent over TLM bus using SPI protocol. The output also includes fault

information whether a fault is detected inside the sensor module. The sensor has internal protection mechanism, when a fault occurs it is registered internally. During initialization this status signal must be checked in order to make sure it is working properly. The sensor module provides possibilities to inject faults in its analog, digital and software parts.

### 6.1.2 Fault Injection GUI

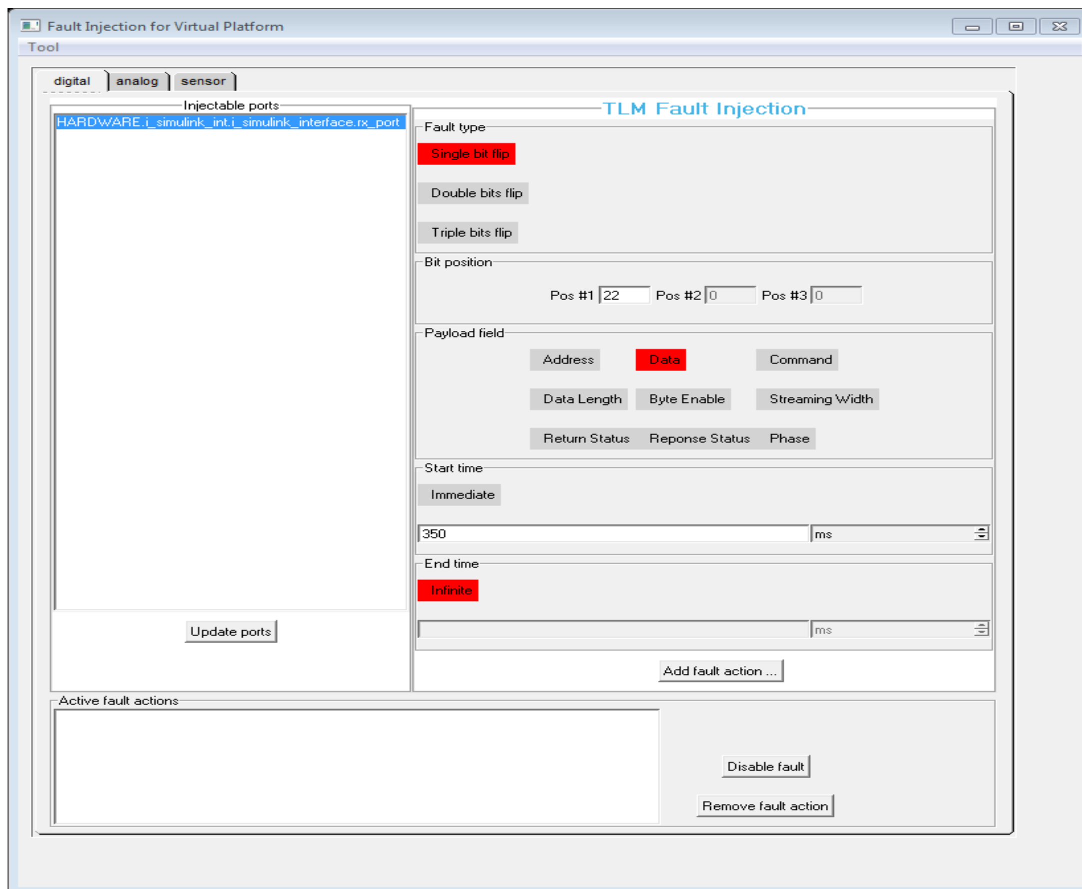


**Figure 117: Fault Injection GUI**

A tcl based GUI is developed to inject faults in the virtual platform is as shown in Figure 117. The first tab we see *digital (digital parts in the system- TLM based Fault Injection)*, second tab *analog (analog parts in the system)* and the last tab is *sensor* (faults inside sensor). The user has the possibility to query for injectable ports, analog fault signals and also fault

injection possibilities in sensor on each tab before configuring them to activate during simulation runtime. Multiple faults can be activated in a single simulation run. Faults on *TLM based Fault Injection side* has parameters such as fault type (single/double/triple bit-flip), bit positions, payload fields, activation time and deactivation time. A fault on analog side has parameters such as fault type (permanent/transient/intermittent), period (optional) and duration (optional), activation time and deactivation time. The sensor module also supports different fault injection possibilities such as *range check*, *bit flips etc.*

### 6.1.3 Test Scenario's

In this setup the following three different fault scenarios are demonstrated,

I. Scenario 1
   a. Sensor is not configured correctly
   b. Motor vibrations are started to intensify
   c. Problem is not detected by sensor
   d. System is not aware of the problem
   e. Hence no reaction of the system but problem detected by simulation

II. Scenario 2
   a. Sensor is configured correctly
   b. Motor vibrations are started to intensify
   c. Sensor output shows the intensified vibrations
   d. System is aware of the problem
   e. System reacts by stopping the motor

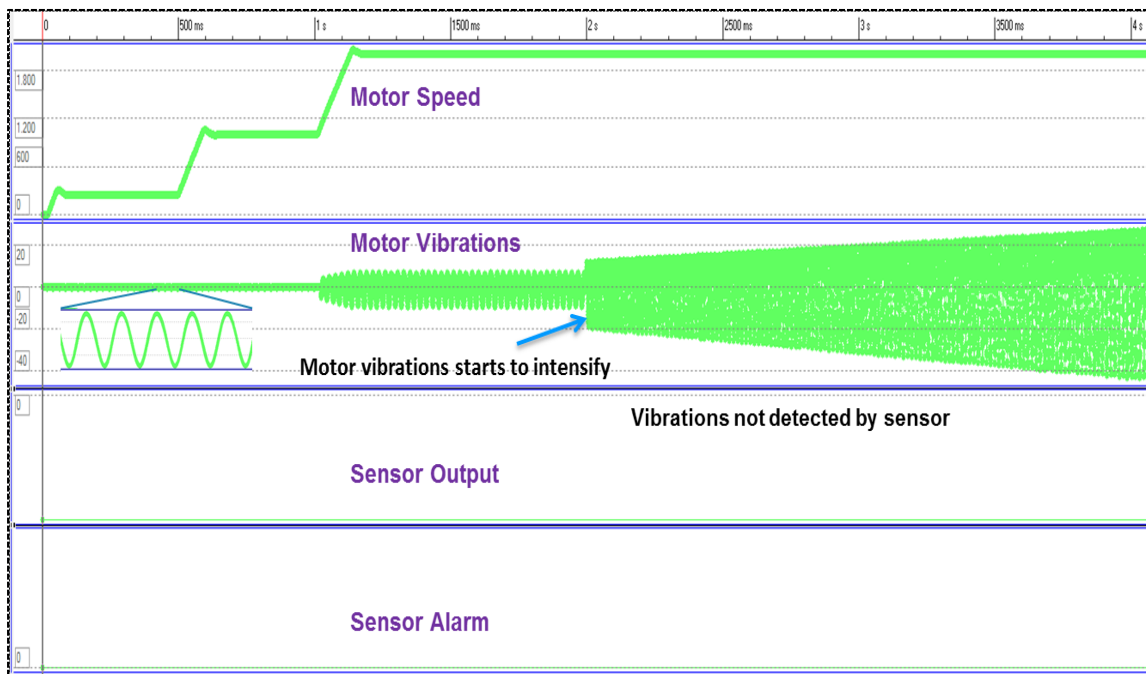


**Figure 118: Scenario 1 - Simulation Output**

III. Scenario 3

a. Sensor monitoring is switched on
b. Fault is injected in sensor
c. Sensor sends an alarm signal
d. PLC-SW shuts off the motor-torque

Figure 118 shows the results of the simulation for scenario 1. First, the motor runs stably, almost no vibrations, i.e., the acceleration remains in the range up to 0.1 mg, up to 1000 RPM (revolutions per minute). As soon as the engine rotates faster than 1000 rpm, it starts to vibrate. The amplitude and frequency of the vibration depends on the rotational speed. The acceleration values by the vibration are in the range between 5 mg and 10 mg. At the time t = 2000 ms, the sensor begins to vibrate (as a result of the fault injection). However, the sensor does not provide an increased acceleration signal because it is configured incorrectly. This makes the problem (the increasing engine vibrations) unrecognizable for the system (the PLC), which subsequently leads to a serious problem of the whole system, even to the destruction of plant parts or a risk to the life and limb of the operating personnel at the

production plant. In the simulation, this problem can becomes apparent e.g. is easily detected by an automatic comparison between sensor input (which is known in the simulation, unlike the real system) and sensor output so that the user of the fault effect simulation can notice and correct the faulty configuration of the sensor.

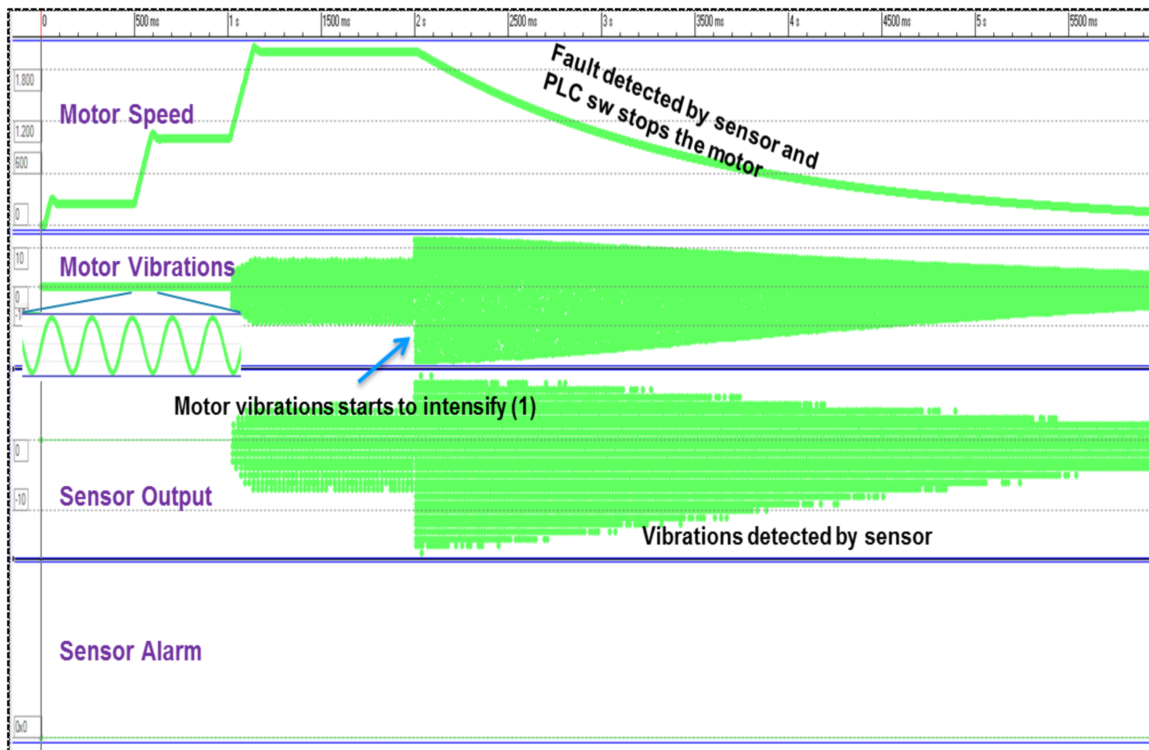


**Figure 119: Scenario 2 - Simulation Output**

In scenario 2, the simulation result of which is shown in Figure 119, the motor also begins to vibrate at time t = 2000 ms. In this case, the sensor functions as expected, supplies a signal that can be evaluated so that the system (the PLC) detects the fault and switches off the motor. Figure 120 shows the results for scenario 3. In this case, a fault is injected into the sensor at the time t = 2000 ms, the sensor detects the internal fault and sends an alarm together with the measured values for the vibration to the PLC. The system (the PLC) reacts by turning off the motor. The engine speed decreases accordingly.

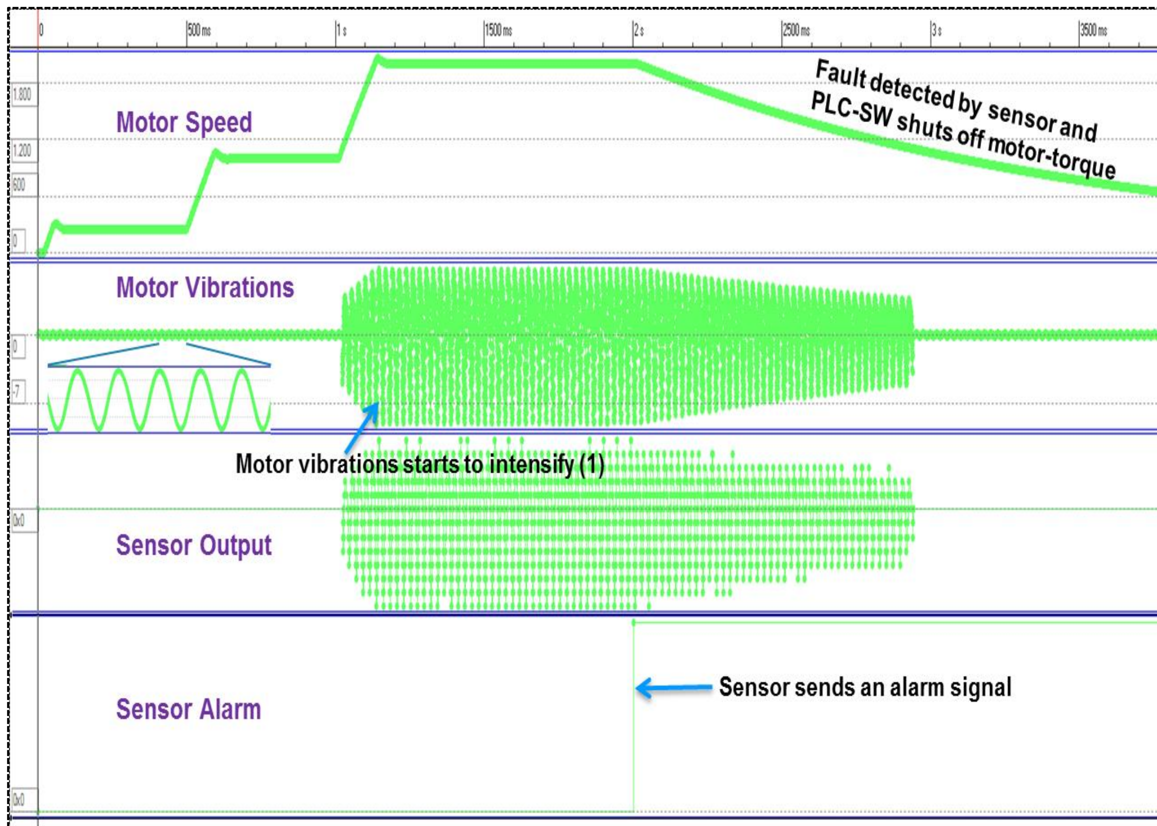


**Figure 120: Scenario 3 - Simulation Output**

### 6.1.4 Conclusion

In this case study, it is demonstrated in detail, not only how the faults are injected in the drive platform but also in the third-party vendor modules (Sensor module from Bosch).

# 6.2 Conveyer-Belt Simulation using V-REP Simulator

## 6.2.1 Setup

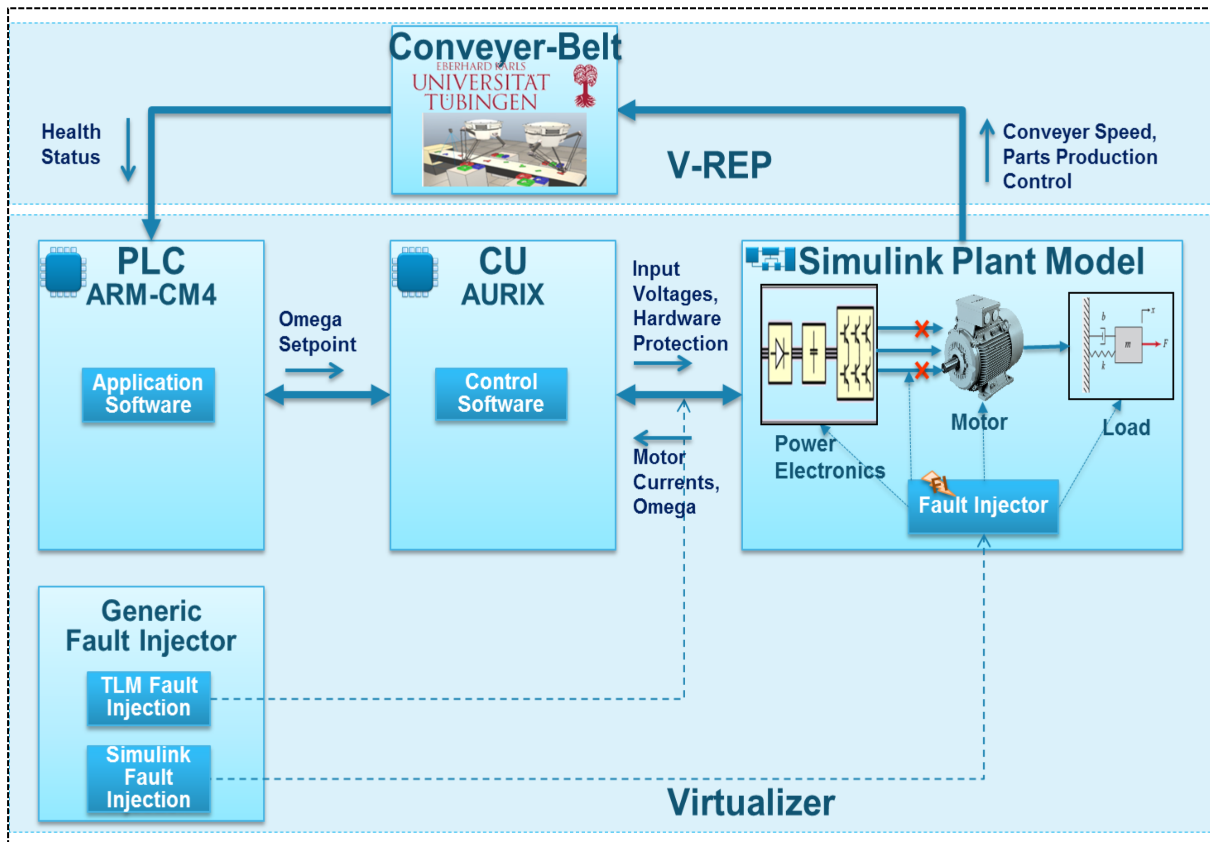


**Figure 121: Simulation Platform with Belt-conveyor**

The simulation platform (Virtualizer) shown in Figure 121 includes models for a PLC (Programmable Logic Controller) for controlling the manufacturing process, i. the application (conveyor belt), a virtual drive platform (VDP) for controlling the motor as well as the model of the conveyor belt as an application. The PLC controls the entire process by sending instructions to the CU to turn the motor at a certain speed; the motor, which in turn drives the conveyor belt. In the case of problems detected in the application, the PLC sends commands to the control unit of the drive (to the VDP) to stop the motor. Generic fault injector is used to trigger errors that can be injected into the VDP during the simulation.

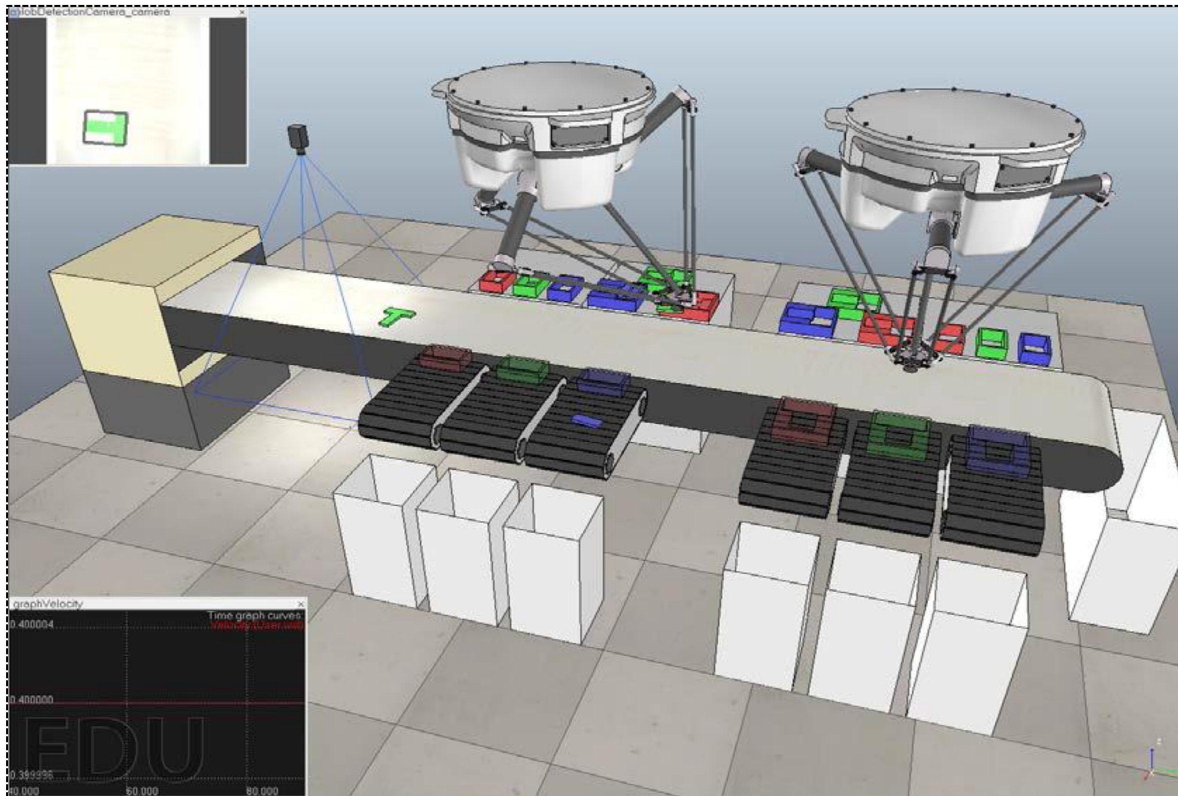


**Figure 122: Virtual Conveyor Belt**

The V-REP [112] simulation (virtual conveyor belt) shown in Figure 122 consists of a production unit which produces T- and I-shaped parts in green, red and blue and places these on the conveyor belt. This is followed by the optical sensor, which implements both position detection and part detection, so that the parts are gripped by the following grippers and sorted according to the shape and colour if the sequence is not affected. The conveyor belt is controlled via a virtual drive platform.

In an intelligent drive system, faults which may occur in the drive must be recognized by the drive itself and must be handled with a corresponding fault response. This includes either stopping the drive or switching it to a torque-free state, and, on the other hand, defined information (an alarm) about the occurrence of the fault should be passed to the PLC i.e., an alarm signal should be passed on to the PLC, so that corresponding fault reaction routines can be triggered by the application software. If this information (alarm) is not passed on to the

other components, they will continue to work despite faults, e.g. further produce parts, even though the belt-conveyor is already stopped, leading to undesirable behaviour.

### 6.2.2 Test Scenario's

In this demonstrator platform, a number of different faults can occur in every component but our focus is only on occurring faults in VDP. When a fault is detected in drive platform, this information should be sent to conveyor-belt module so that appropriate action is taken i.e., in presence of faults both the motor/conveyor-belt and piece production must be stopped. Whenever a fault is detected in the drive operation, a fault detected flag is set and this information is passed on to the conveyor-belt application.

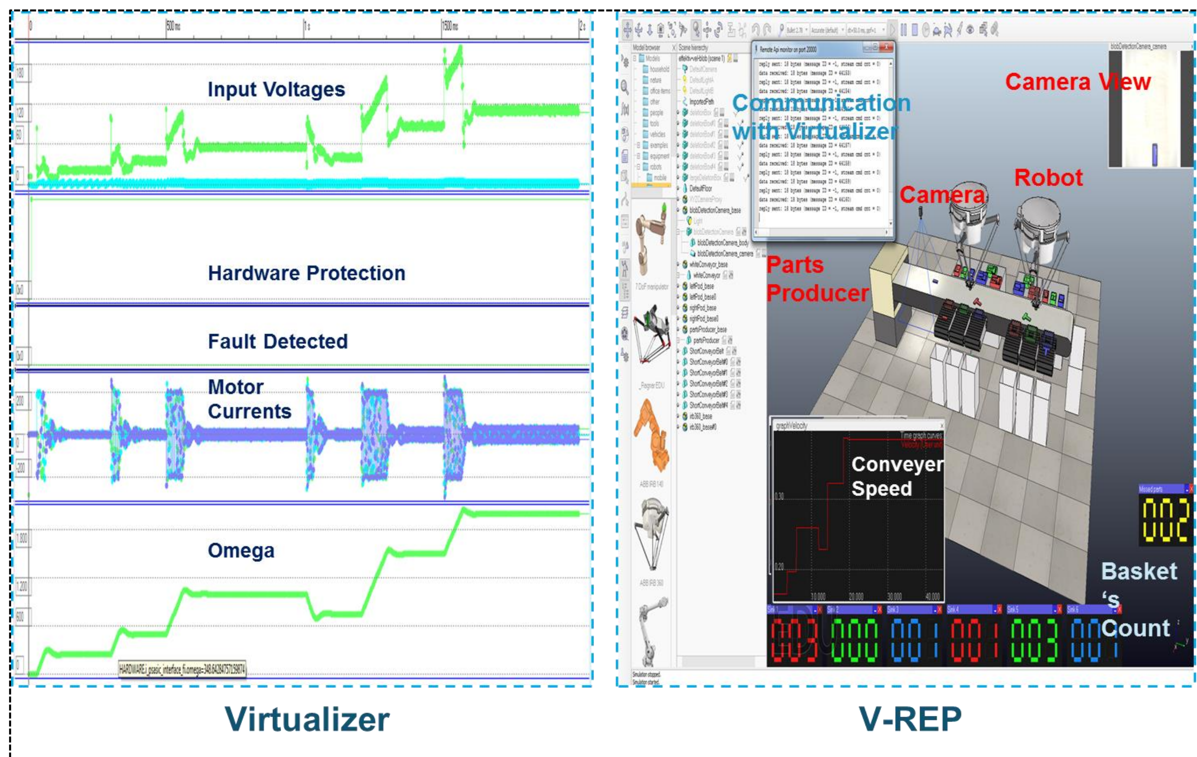


**Figure 123: Simulation Output without Fault Injection**

The simulation output in Figure 123 shows the output of both Virtualizer and V-REP simulations without fault injection. The virtual connection of motor to the conveyor-belt can be observed in '*omega'* and '*conveyor speed'* signals.

When one of the three phases of supply voltage to the motor is broken, the control system will draw more current in the other phases. Only if the current exceeds the limits then the motor will be shut off by protection mechanisms. Whereas when two of the three phases of supply voltage to motor are broken, it will be shut off as it will not be able to drive it anymore.

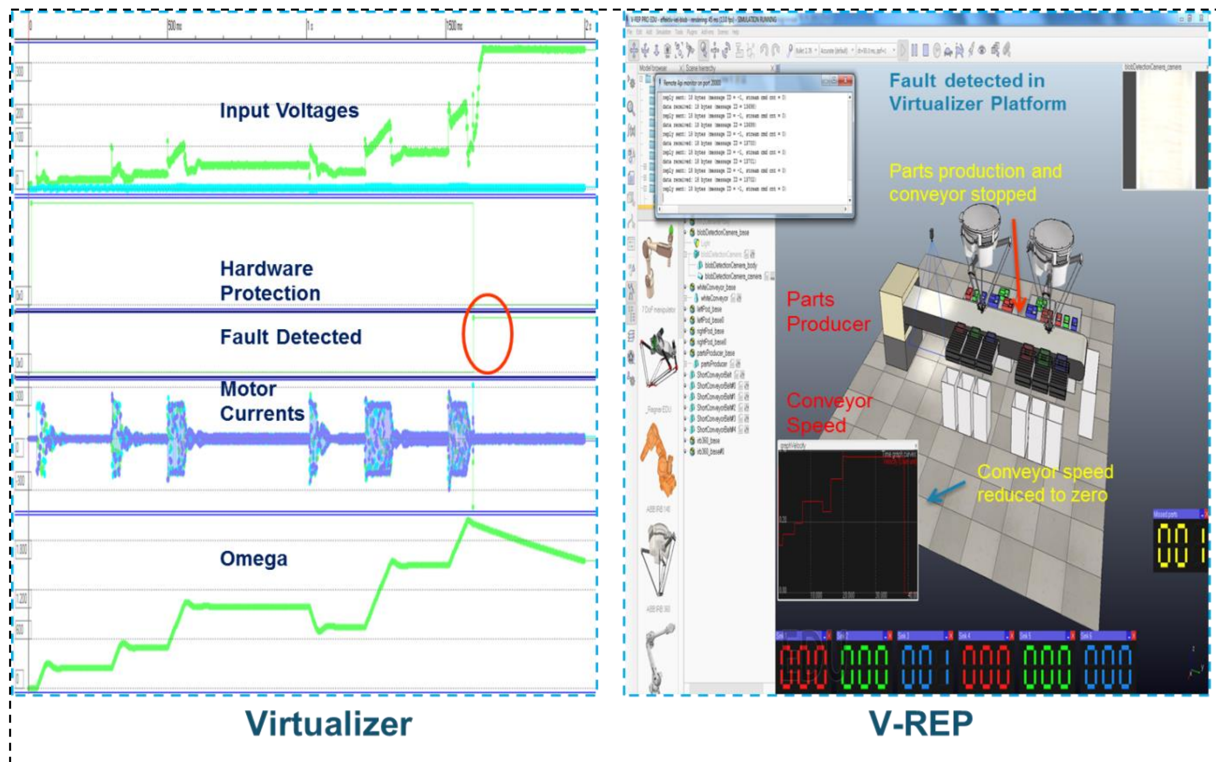


**Figure 124: Simulation Output with Fault Injection**

The simulation output is shown in Figure 124 and the example fault injection scenario is demonstrated as follows,

- At t = 1500 ms, a transient fault (loose contact) occurs on one of the 3 input motor terminals for a short duration of 50 ms, this fault is not detected as the system is still able to drive the motor.

- At t = 1550 ms, a permanent fault on one phase of the 3 input motor terminals is activated. The system is still able to drive the motor as currents drawn are within limits.
- At t = 1600 ms, a permanent fault on second phase of the 3 input motor terminals is activated. Now, system cannot drive the motor and the over-current is detected by protection mechanisms in the control software.

Hence at t = 1600 ms the motor is shutoff and the parts production is stopped. The motor speed will drop gradually to zero.

### 6.2.3 Conclusion

In this case study, with the help of the methods developed in the work, it is demonstrated in detail how the faults are simulated in the conveyor-belt application. The behavior of the system along with the application under influence of faults is studied and using these results counter measures can be developed to ensure correct function of an intelligent motion control system in the event of a fault.

## 6.3 Hardware-in-the-loop (HIL)

The final step of validating a newly developed embedded electronic system, in particular in the domains of automotive and industrial motion control, is to use a HIL set-up, where the final implementation of the embedded control electronics is connected to a real-time simulation system. Also in such a configuration, fault injection into the physical components that run in the real-time simulation, is needed to validate the proper implementation of previously developed failure reaction mechanisms in final hardware and SW. In case the adapted Simulink model used before in the virtual prototype is abstract enough to be executed in real time, it can be re-used for this task. It has to be imported into the real-time simulation system and the fault injection has to be controlled by proper means. In our investigations, we use the modular dSPACE real time simulation system with DS1006 processor board.

### 6.3.1 Setup

SINAMICS G120 [113] is a modular frequency inverter system that is designed to provide precise and cost-effective speed/torque control of three-phase motors. A G120 system essentially comprises of two function units: Control Unit (CU) and Power Module (PM). The CU controls and monitors the PM and the connected motor using several different control types that can be selected. Comprehensive protection functions provide a high degree of protection for the PM and the motor.

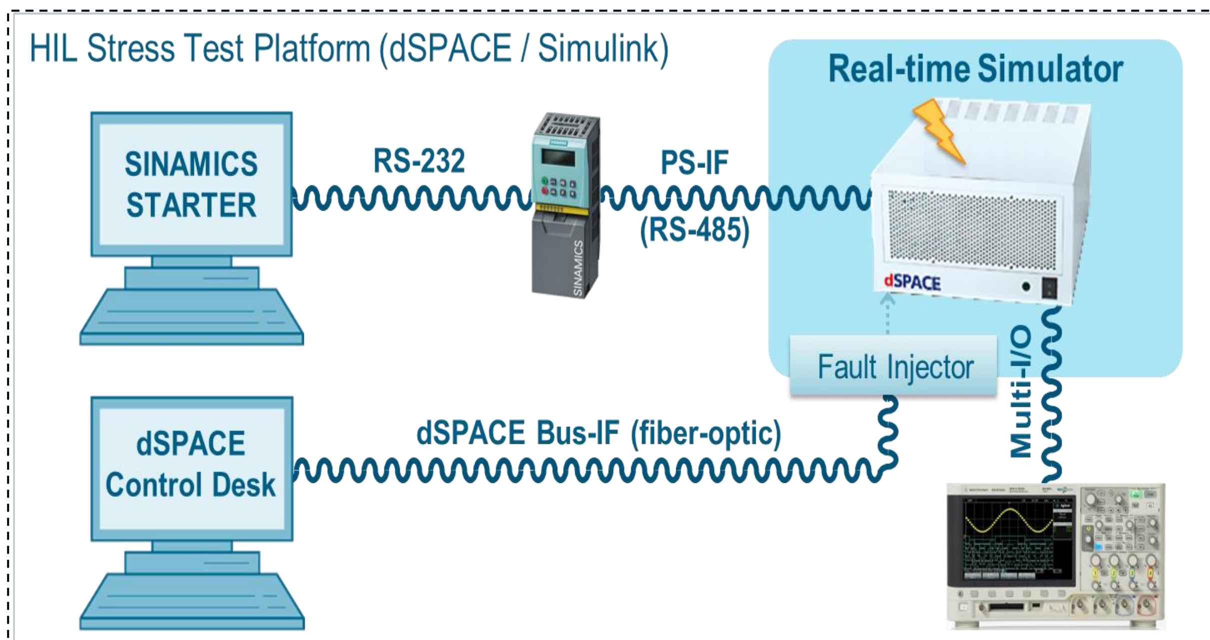


**Figure 125: HIL Setup with SINAMICS G120 with CU240 and PM240**

The HIL setup is based on G120 (shown in Figure 125) which consists of the following components:

- Control Unit (CU240S): The CU controls and monitors the PM and the connected motor in various control modes which can be selected as required. It supports communication with local or central control as well as with monitoring devices.
- Power Module (PM240): The Power Modules (PM) is available for motors with a power range of between 0.37 kW and 250 kW. It features state-of-the-art IGBT technology with pulse-width-modulated motor voltage and selectable pulse frequency.

Comprehensive protection functions provide a high degree of protection for the PM and the motor. IGBT technology and pulse-width modulation are used to ensure reliable and flexible motor operation. The PM consists of ASIC and power electronics components.

- Motor: A three-phase induction (asynchronous) motor is used.
- SINAMICS STARTER (Version - V4.4.1.0): The STARTER commissioning tool [114] features a project Wizard that guides you step-by-step through the commissioning process. Configuring the drive setup (CU, PM and motor) using the PC is user friendly and faster. The speed of the motor is specified via the STARTER software tool using the analog input interface board.
- DS1006 Processor Board with DS2201 Multi-I/O Board: The DS1006 processor board is one of the processors that form the core of the modular dSPACE [115] hardware. It is our flagship for very complex, extensive, computer-intensive models for virtual simulations. In such simulations, the ds1006 provides the computing power for the real-time system and also acts as an interface to I / O boards and the host PC. It can be directly connected to all dSPACE I / O boards via the PHS bus. The ds2201 multi-i/o board as an external interface to time varying signals.
- ControlDesk: ControlDesk is the dSPACE experiment software for seamless ECU development. It performs all the necessary tasks and gives you a single working environment, from the start of experimentation right to the end.
- Interfaces (serial bus, PHS bus …): The STARTER software and the CU are interfaced using serial cable (RS232). CU and PM are interfaced using power stack interface (PS-IF: RS-485).
- Analog Oscilloscope: For observation of varying signals from dSPACE system interfaced through the ds2201 multi-i/o board.

### 6.3.2 Characteristics of the Setup

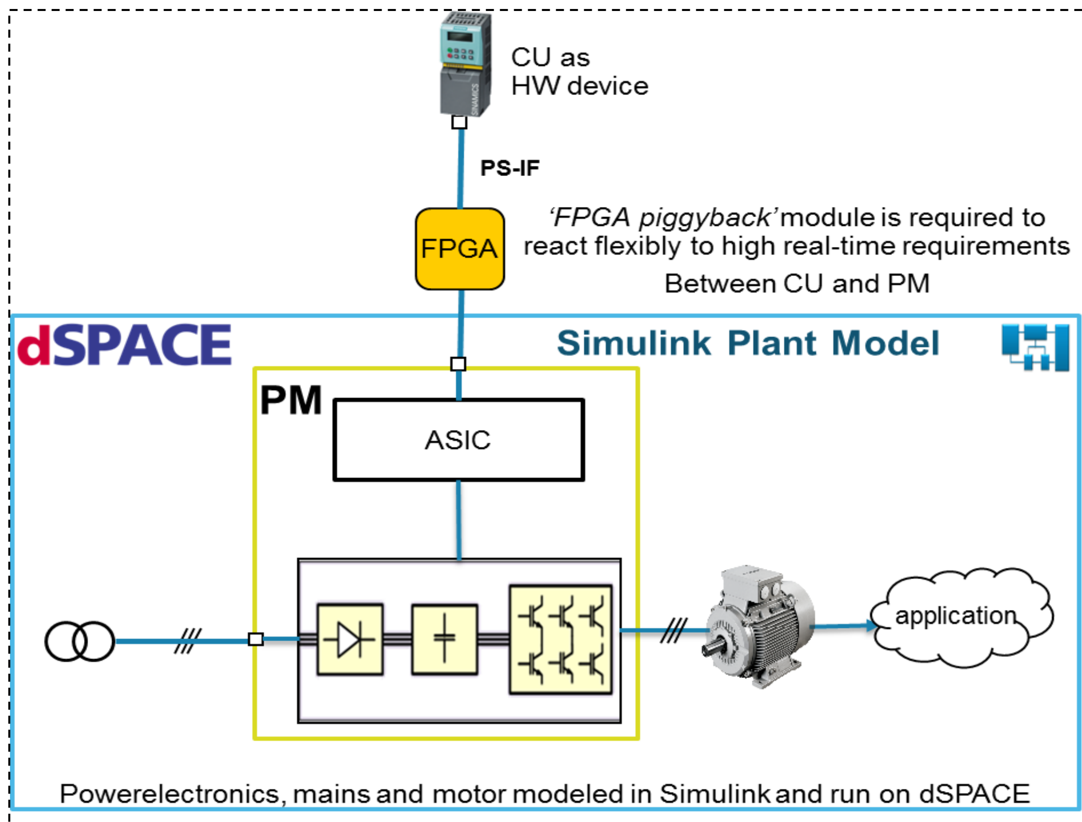


**Figure 126: CU Hardware with Virtual PM**

A real hardware device CU240S is used as a CU which is integrated in real world environment. The PM module (ASIC + power electronics) along with mains and motor are modeled in Simulink. The CU is coupled with PM via serial communication interface.

Typical use cases of this setup are *software validation* and *virtual system test of CU*. A FPGA piggyback module is connected between CU and PM to handle the high real time requirements of CU. Figure 127 and Figure 128 show the HIL demonstrator setup and user interface of SINAMICS STARTER software tool.

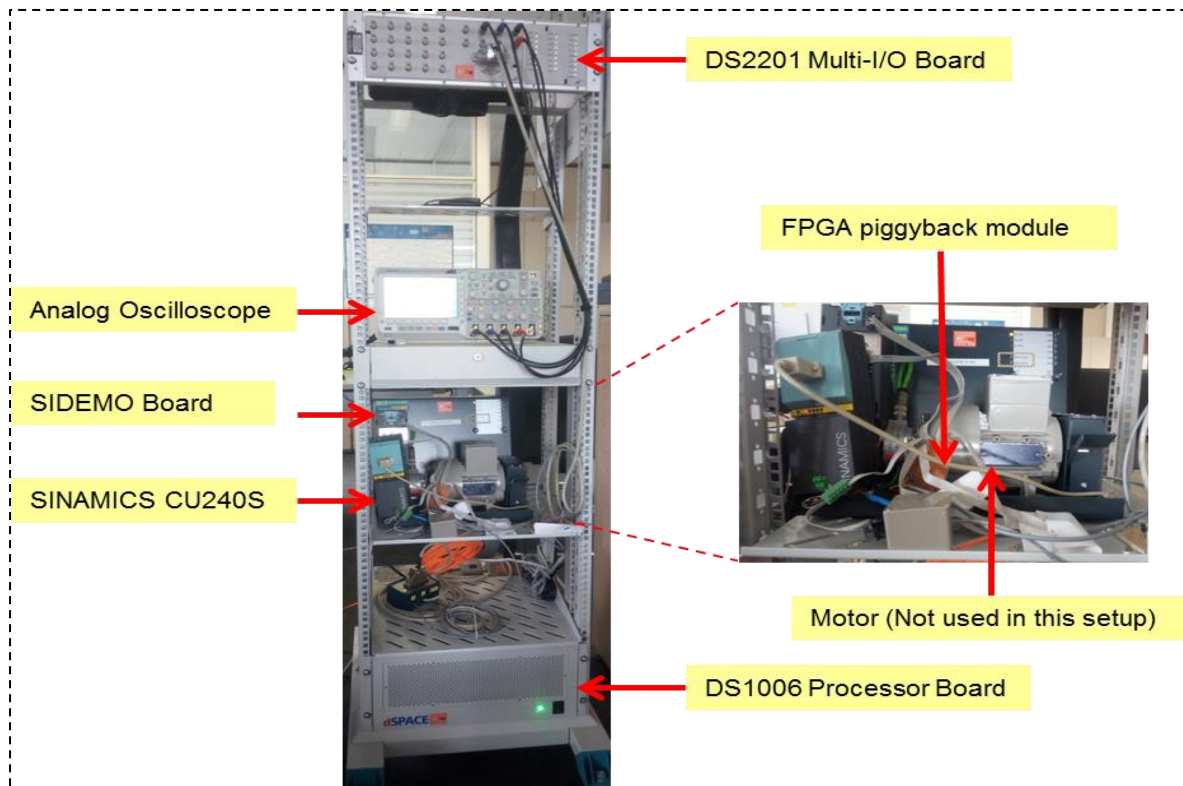


**Figure 127: HIL Demonstrator Setup**

### 6.3.3 Fault Injection

The Simulink model is extended (see 4.6.1 Preparing the Simulink Models for Fault Injection) for fault injection; it is prepared for execution on the dSPACE system by using the Real-Time Interface (RTI) library provided by dSPACE. The standard Simulink coder based flow is then used to upload the prepared model to the dSPACE real time simulator, which is connected to the new controller. The ControlDesk [116] software from dSPACE provides possibility to access and modify parameters inside the Simulink model. To control fault injection in such a set-up, ControlDesk is used to modify the fault values and control signals. The ControlDesk experiment is shown in Figure 129.

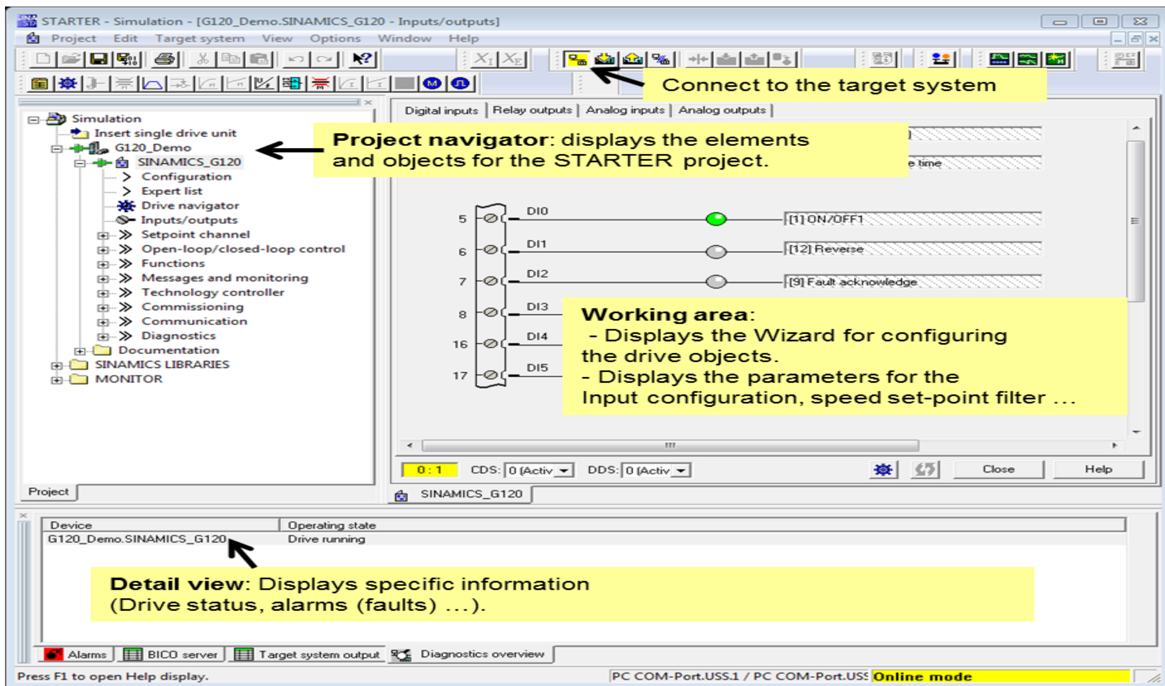


**Figure 128: SINAMICS STARTER User Interface**

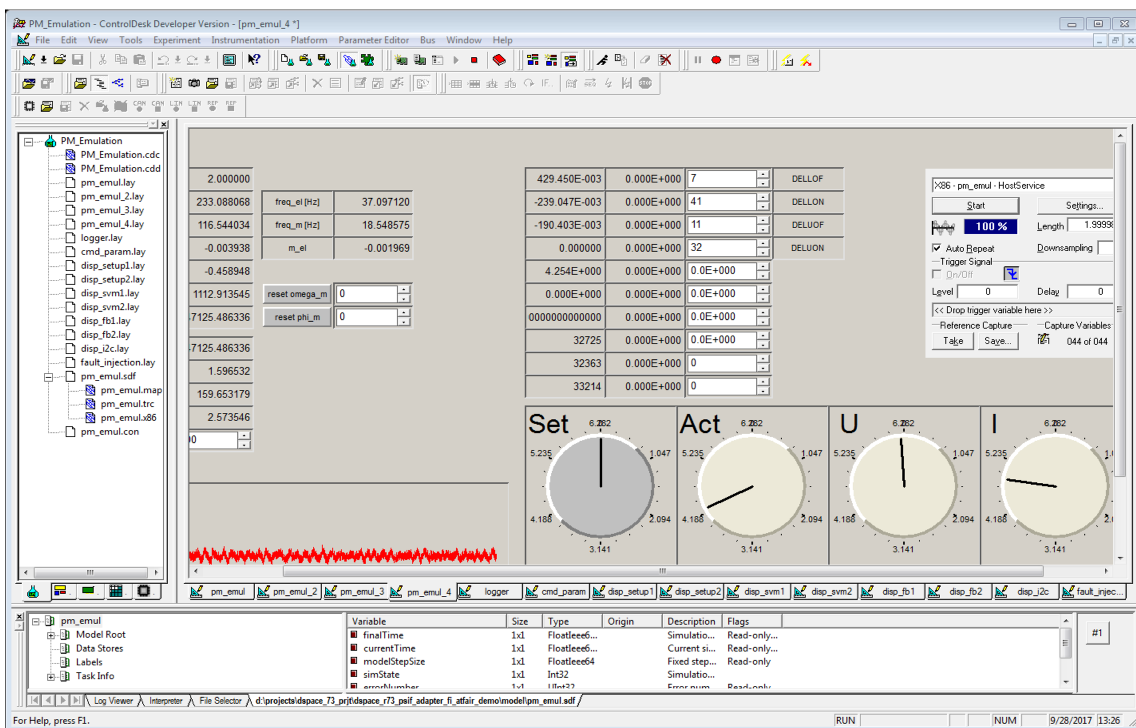


**Figure 129: ControlDesk Experiment**

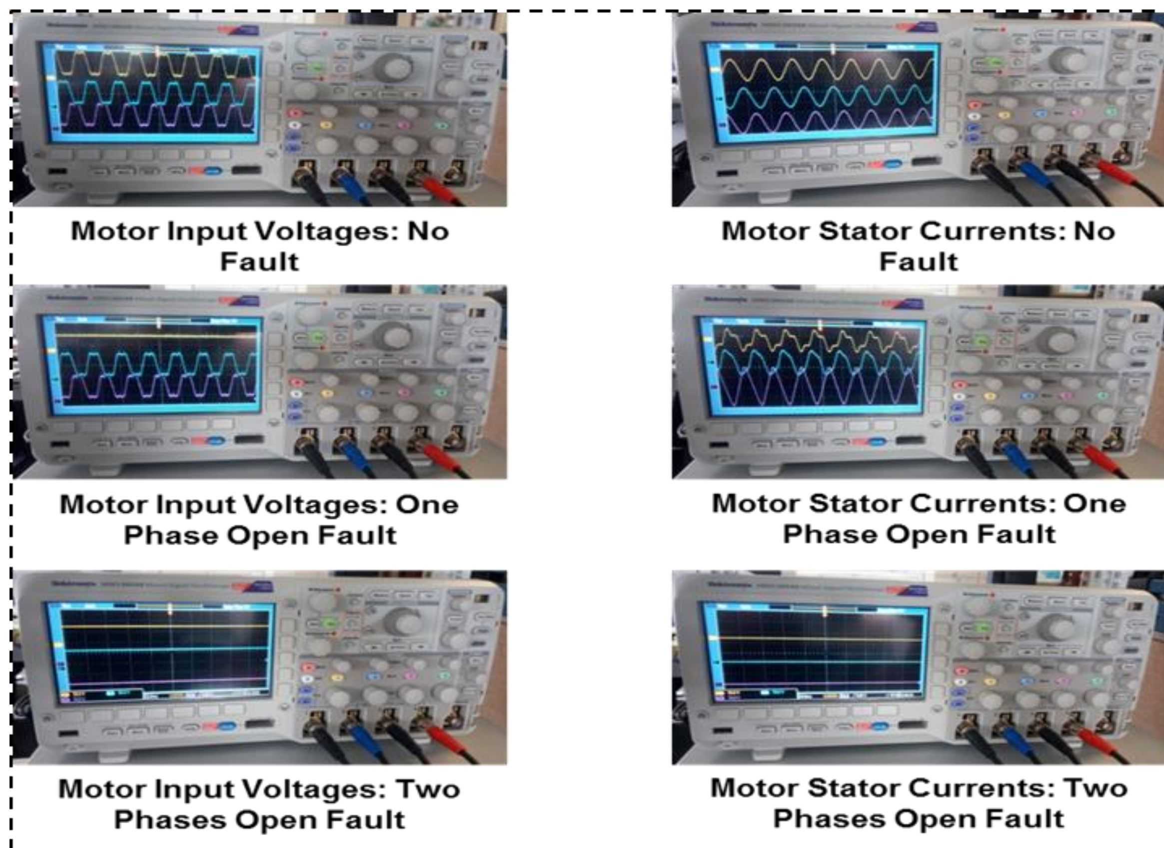


**Figure 130: Fault detected in SINAMICS STARTER**

Figure 130 shows the oscilloscope output with motor input voltages and the measured stator currents without fault injection and with fault injection. When one of the input phase lines broken fault is activated, the motor still continues to run but with reduced output power and also the disturbance is observed in the motor stator currents. Whereas when two input phase lines are broken, there is no rotating magnetic field to produce torque on rotor hence the motor deaccelerates. This shows the high applicability of the approach in terms of covering physical effects in an abstract model. As soon as this fault is activated, the reaction is observed instantly in the system and it is recorded in the SINAMICS STARTER commissioning tool as frequency consistency fault (see Figure 131).

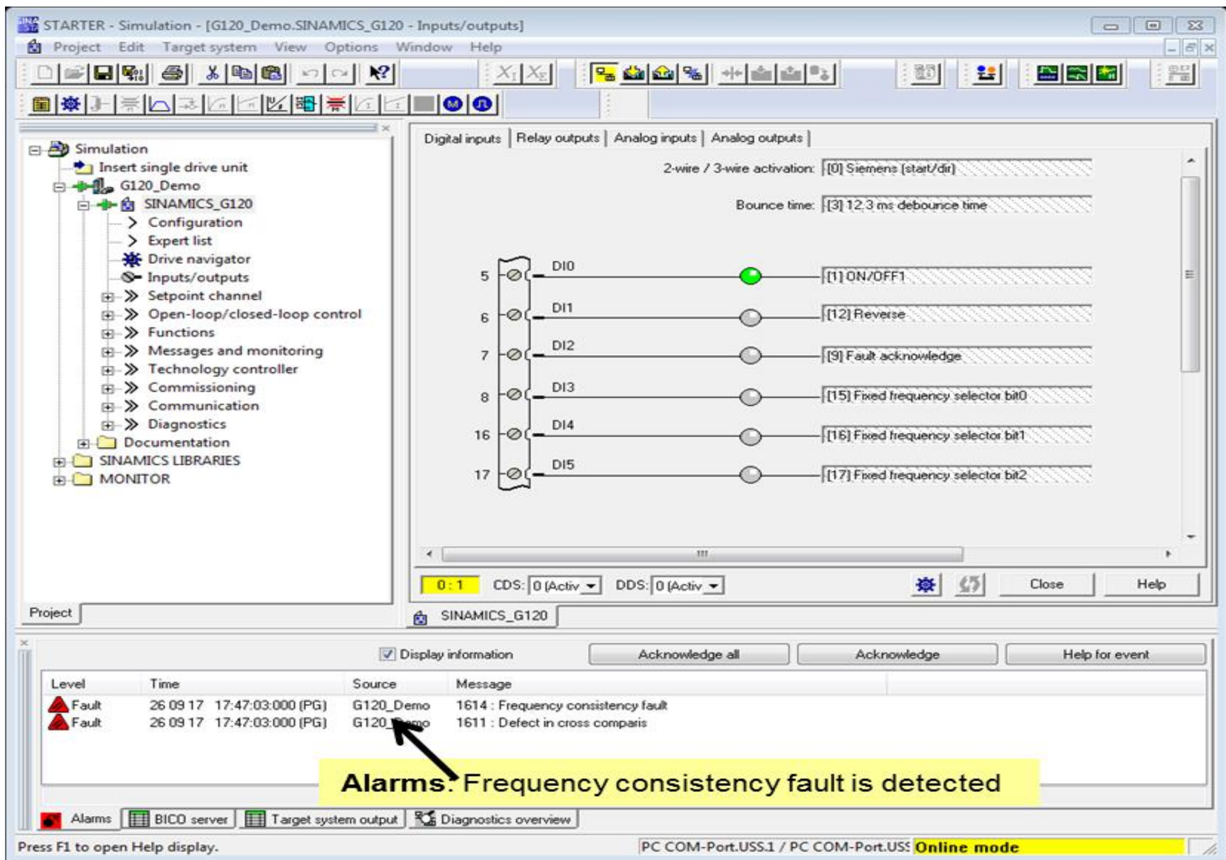


**Figure 131: Motor Voltages and Currents measured using an Analog Oscilloscope**

## 6.3.4 Conclusion

In this case study, the faults in physical component model (Simulink model) are simulated in a setup with SINAMICS CU with real (production) software running on it. It has been shown that, the same fault models which were developed for fault simulation in virtual platform can be used to simulate the faults in HIL setup.

# 7 Conclusion and Outlook

## 7.1 Summary

The manufacturing facilities of the future, the so-called Cyber-Physical Production Systems (CPPS) for Industry 4.0, are highly complex, intelligent systems consisting of a variety of components, which have to interlock smoothly. Even more complex is the embedded software, which controls these systems. The highly critical issues in the development of future production plants are imposed by motion control systems, which manage the fast and most accurate positioning and motion control of conveyor belts and robot arms, for instance. These systems are required to properly cope with failures of all kinds to guarantee safety of operators and machine integrity at any time. But what happens, if for instance single chips in a robot's control unit fail, if a motor due to a bearing damage overheats or a sensor delivers faulty data? Is it possible to develop and test the control software of these systems in a way that in case of faults in one or more components the total system always remains in a safe state? Is it guaranteed that humans near to these machines are not harmed and expensive parts like motors or robot arms are not destroyed? As of today, the final tests of those systems are mainly based on physical prototypes to ensure the correct and safe operation of the system. However, those prototypes are available only in later phases of the design process.

This thesis researched to overcome these constraints and developed a methodology for fault simulation of intelligent motion control systems based on virtual prototypes. The hardware simulation models of electrical and electronic components with fault injection possibilities are used replacing their physical counterparts for validation in early design phases. With the results it is now possible to carry out system tests, which were previously only possible in late design phases. Expensive iterations with rework, limited functionality or even the threat to the product success with the consequent impairment of the competitiveness can be avoided. The fault effect simulation developed in this work makes it also possible to protect the reaction of motion control systems against such errors, which could hardly be provoked with the test methods available before with real hardware constructions. This can

further increase the safety in manufacturing plants, despite the rapidly growing complexity of these systems. Previous solutions either worked exclusively on low levels of abstraction, thus allowing only a small system to be viewed or based on complex, real test setups, which do not allow an early and comprehensive analysis of all relevant fault scenarios.

## 7.2 Contributions

### 7.2.1 Fault Injection in Heterogeneous Component Models

Industrial electronic systems in general consist of heterogeneous components, digital electronics, analog electronics, mechanical components and the environment. MATLAB / Simulink is the most widely accepted tool as the development environment for control engineering in industry, both as a simulator on system level and as a fault analysis tool. The digital peripherals of microcontrollers are commonly developed as models at the transaction level in SystemC; this allows fast simulation due to model dependencies. A fast co-simulation of MATLAB / Simulink and SystemC supports early simulation of a heterogeneous system at system level for complex real-time applications or the MATLAB / Simulink models could be integrated into a SystemC based simulation environment.

A concept for the connection of the fault simulation techniques to heterogeneously abstracted platform models was developed. In order to enable fault simulations, a method was developed as to how fault injector blocks can be inserted into physical component models in MATLAB/Simulink. A TLM-based fault injection technique is used for digital component models by defining appropriate fault models. Further results were obtained during the integration of MATLAB/Simulink models into virtual platform and the control of the fault injection within the framework of these simulation platforms. In detail, this included the fault injection and integration of MATLAB / Simulink and SystemC-TLM models using the example of an industrial drive. The SystemC modules were generated from MATLAB / Simulink. Later, the abstract fault models of physical component models are transferred to hardware-in-the-loop (HIL) system to conduct fault simulation tests.

### 7.2.2 Fault Transfer

The key to capturing physical system behaviour in presence of faults is to have an accurate and appropriate multi-domain simulation model of a physical system, which incorporates important mechanisms such as electrical effects, magnetic effects, mechanical loading etc. Modeling these mechanisms is difficult and more importantly simulating such a system is very slow. On the other hand in abstract models, the functional behaviours are extracted which might result in lack of accuracy in order to simulate faults. For example, it is difficult to simulate electrical faults like short-circuit between phases voltage lines in abstract data flow models. We proposed a methodology to transfer fault behaviours from accurate simulation models to the abstract representation, which is fast and detailed enough to simulate the behaviour of the overall system. The following actions are performed. (i) Simulate the accurate model in presence of the fault and capture the behaviour (ii) Transfer the fault behaviour onto the abstract model and simulate the overall system.

### 7.2.3 Automated Stress Tests and Post Processing

With the help of virtual stress tests methodology proposed in this work, simulated system tests are carried out with fault injection in hardware and software at the earliest possible stage. As the number of tests is significantly high, the time and effort required to carry out is very high and it is considerably shortened by an automated tests execution and evaluation.

A method for the automated tests execution and evaluation of results of fault effect simulations was developed. The automated stress tests are performed with the TCL script with necessary inputs and the automated evaluation method was implemented in the C ++ programming language as a prototype in order to achieve an optimal evaluation performance. The prototype for automatic evaluation of simulation results was used in a TCL regression test environment by developing a TCL wrapper using the SWIG (Simplified Wrapper and Interface Generator) library i.e., it was integrated as a prototype into the script for automated stress tests. Using this method, the simulated measuring curves with fault injection are automatically evaluated for motion control systems and compared with the corresponding reference curves (without fault injection). The results are graphically visualized using plotting tool for analysis.

### 7.2.4 Case Studies of Motion Control System

In order to demonstrate our proposed methodologies, two virtual HIL demonstrators and one HIL demonstrator were established. These demonstrators were able to demonstrate in detail how, with the help of the methods developed in this work, the correct function of an intelligent motion control system can be ensured even in the event of a fault.

## 7.3 Outlook

### 7.3.1 ASAM XIL Standard for Fault Effect Simulation

ASAM XIL [117] is an API standard for the communication between test automation tools and test benches. The standard supports test benches at all stages of the development and testing process – most prominently model-in-the-loop (MIL), software-in-the-loop (SIL) and hardware-in-the-loop (HIL). The notation "XIL" indicates that the standard can be used for all "in-the-loop" systems. This has the advantage that it enables users to freely choose testing products according to their requirements and integrate them with little effort. The major goal of the ASAM XIL standardization effort is to allow for more reuse in test cases and to decouple test automation software from test hardware. As a consequence, the reuse of test cases within the same test automation software on different test hardware systems is achieved. This leads to a significant reduction of efforts for test hardware integration into test automation software. ASAM XIL focuses on the decoupling of test-cases from real and virtual test systems. It broadly covers all use cases in the testing area and is primarily used in conjunction with test automation tools on a client server base. The standard is used in test automation applications of all automotive E/E domains, e.g. drivetrain, steering, electric lighting, body electronics etc.

The concept of the ASAM XIL standard should be looked upon to standardize the test specification for vHIL and HIL platforms presented in our work for virtual stress tests automation.

### 7.3.2 Improved Workflow for Virtual Prototypes with Stress Tests

In [118] an improved workflow for the generation of virtual prototypes based on model-based design and automatic code generation is presented. An extended IEEE 1685 ™: IP-XACT [119] register description is the basis for MathWorks code generation tools to automatically generate a functional C / C ++ description of the signal processing algorithms as well as the interface description necessary for the integration into the SystemC TLM 2.0 based virtual platform. The register description uses the Synopsys® SystemC Modeling Library (SCML). These signal processing components, which were previously, designed using Model-Based Design and MathWorks MATLAB / Simulink, can then be integrated directly into a virtual platform. Through this workflow, both their structure is accelerated and the functional match between the algorithm design in MATLAB and Simulink and the simulation in the virtual platform is guaranteed.

Incorporating the information of faults in Simulink models in IP-XACT descriptions will completely automate the Simulink model integration into virtual platforms for virtual stress tests.

# REFERENCES

[1] Industry 4.0 - BMBF. [Online]. https://www.bmbf.de/de/zukunftsprojekt-industrie-4-0-848.html

[2] Plattform Industrie 4.0. [Online]. http://www.plattform-i40.de/I40/Navigation/EN/Home/home.html

[3] C. L. Lynn, "Basic elements of mathematical modeling," in *In Handbook of Dynamic System Modeling, P. A. Fishwick, Ed. Chapman & Hall/CRC, ch. 5*, 2007.

[4] Synopsys Inc. Virtual Prototyping. [Online]. https://www.synopsys.com/verification/virtual-prototyping.html

[5] F., Drechsler, R. Rogin, "Debugging at the Electronic System Level," *Springer Verlag*, 2010.

[6] C.M. Apostoaia, "Multi-domain System Models Integration for Faults Detection in Induction Motor Drive," *Electro/Information Technology*, pp. 388-393.

[7] R. Koppak, O. Bringmann, and A. von Schwerin, "Fault Injection in Multi-Domain Physical System Models at Different Levels of Abstraction," in *1st International Workshop on Resiliency in Embedded Electronic Systems (REES)*, Amsterdam, Netherlands, 2015.

[8] Mathworks. Matlab/Simulink. [Online]. http://www.mathworks.de/products/simulink/

[9] OSCI AMS Working Group. (2010) Standard SystemC AMS extensions Language Reference Manual. Release 1.0. [Online]. www.accellera.org

[10] Bringmann Oliver et al., "The Next Generation of Virtual Prototyping: Ultra-fast Yet Accurate Simulation of HW/SW Systems," in *Design, Automation and Test in Europe (DATE)*, 2010.

[11] R. Koppak, O. Bringmann, and A. von Schwerin, "Fault Injection Campaigns in Multi-Domain Virtual Prototypes," in *2nd International Workshop on Resiliency in Embedded Electronic Systems (REES)*, Lausanne, Switzerland, March, 2017.

[12] andrei Tabacaru Bogdan et al., "Fault-Injection Techniques for TLM-Based Virtual Prototypes," in *FDL*, 2015.

[13] OSCI. TLM-2.0: Language Reference Manual. [Online]. http://www.accellera.org/downloads/standards/systemc

[14] OSCI Working Group, "SystemC 2.0.1 Language Reference Manual, Revision 1.0," 2003.

[15] OSCI. SystemC. [Online]. http://www.accellera.org/downloads/standards/systemc

[16] Thorsten Grotker, *System Design with SystemC*. Norwell, MA, USA: Kluwer Academic Publishers, 2002.

[17] D. C. Black, J. Donovan, B. Bunton, and A. Keist, "SystemC: From the ground up.," *Springer Verlag*, 2009.

[18] Wikipedia. Instruction set simulator. [Online]. https://en.wikipedia.org/wiki/Instruction_set_simulator

[19] Analog Devices. Processors & DSP. [Online]. http://www.analog.com/en/design-center/processors-and-dsp.html

[20] ARM Holdings. ARM TrustZone and ARM Vector Floating Point (VFP). [Online].

http://www.arm.com/products

[21] ARM. ARM fast models. [Online]. http://www.arm.com/products/tools/models/fast-models/

[22] ARM Holdings. Custom SoC Design. [Online]. https://developer.arm.com/soc

[23] Accellera systems initiative. About SystemC TLM. [Online]. http://accellera.org/

[24] H. W. M. van Moll, Henk Corporaal, Victor Reyes, and Marleen Boonen, "Fast and accurate protocol specific bus modeling using TLM 2.0," in *Design, Automation & Test in Europe (DATE)*, 2009.

[25] F. Pêcheux and C, Lallement, "VHDL-AMS and Verilog-AMS as Competitive Solutions," *Springer, Boston, MA*, no. System Specification & Design Languages, 2003.

[26] MathWorks. Physical Systems Simulation. [Online]. https://www.mathworks.com/products/simscape.html

[27] Fraunhofer-Gesellchaft. SystemC-AMS PoC Beta 2. [Online]. https://www.eas.iis.fraunhofer.de

[28] A. Vachoux, C. Grimm, and K. Einwich, "Towards Analog and Mixed-Signal SOC Design with SystemC-AMS," in *In Second IEEE International Workshop on Electronic Design, Test and Applications (DELTA)*, 2004.

[29] S. Adhikari and C. Grimm, "Modeling Switched Capacitor Sigma Delta Modulator Nonidealities in SystemC-AMS," in *In Proceedings of the 2010 Forum on specification and Design Languages*, 2010, pp. 216–221.

[30] K. Caluwaerts and D. Galayko, "SystemC-AMS modeling of an electromechanical harvester of vibration energy," in *Forum on Specification, Verification and Design*

*Languages*, Sept. 2008, pp. 99–104.

[31] P. A. Hartmann, P. Reinkemeier, A. Rettberg, and W. Nebel, "Modelling control systems in SystemC AMS - Benefits and limitations," in *IEEE International SOC Conference (SOCC)*, Sept. 2009, pp. 263–266.

[32] E. Lee and D. Messerschmitt, "Synchronous data flow," in *Proceedings of the IEEE*, 1987.

[33] The MathWorks, Inc. Simscape. [Online]. https://de.mathworks.com/products/simscape.html

[34] MathWorks. HDL Verifier. [Online]. https://www.mathworks.com/products/hdl-verifier.html

[35] A. Frederiksen, "Model-Based Design of Advanced Motor Control Systems," Analog Devices, Inc., 2013.

[36] Çakmakc Melih, Li Yonghua, and Liu Shuzhen, "Model-in-the-Loop Development for Fuel Cell Vehicle," in *American Control Conference on O'Farrell Street*, San Francisco, CA, USA, 2011.

[37] Werner Stephan, Masing Leonard, Lesniak Fabian, and Becker Jürgen, "Software-in-the-Loop simulation of embedded control applications based on Virtual Platforms," in *FPL*, 2015.

[38] Osadcuks Vitalijs and Galins Ainars, "Software In the Loop simulation of autonomous hybrid power system of an agricultural facility," in *Engineering for rural development*, Jelgava, 2012.

[39] JC Juang et al., "Design, Implementation and Verification of Microsatellite Attitude Determination and Control Subsystem Based on Processor-in-the-loop," in *The 3rd*

*Nano-Satellite Symposium: conference proceedings*, Kitakyushu, Japan, 2011.

[40] S. Lentijo, A. Monti, and R. A. Dougal, "Design of DC Motor Speed Control through Processor-in-the-Loop Approach," in *Huntsville Simulation Conference*, Huntsville, AL, October 2003.

[41] P. Sarhadi and S. Yousefpour, "State of the art: hardware in the loop modeling and simulation with its applications in design, development and implementation of system and control software," *International Journal of Dynamics and Control*, 2014.

[42] C. Kleijn. Introduction to Hardware-in-the-Loop Simulation.

[43] Synopsys. Seamless Transition from Software in the Loop to virtual Hardware in the Loop with Virtualizer Studio. [Online]. https://www.synopsys.com/company/resources/newsletters/prototyping-newsletter/seamless-transition-software.html

[44] G.A. Kanawati, N.A. Kanawati, and J.A. Abraham, "FERRARI: a flexible software-based fault and error injection system," *IEEE Transactions on Computers*, pp. 248 - 260, 1995.

[45] A. Ademaj, P. Grillinger, P. Herout, and J. Hlavicka, "Fault tolerance evaluation using two software based fault injection methods," in *On-Line Testing Workshop, Proceedings of the Eighth IEEE International*, July 2002.

[46] Maha Kooli and Giorgio Di Natale, "A survey on simulation-based fault injection tools for complex systems," in *9th IEEE International Conference On Design and Technology of Integrated Systems In Nanoscale Era (DTIS)* , Santorini, Greece, 2014.

[47] A. Riefert, M. Sauer, M. Jörg, B. Wolfram, and B. Bernd, "FPGA -based Fault Injection Framework," in *VLSI Test Symposium (VTS)*, 2013.

[48] A. Mohammadi, M. Ebrahimi, A. Ejlali, and S. G. Miremadi, "SCFIT: A FPGA-based fault injection technique for SEU fault model," in *Design Automation and Test in Europe*, March, 2012, pp. 586–589.

[49] L. Reva, V. Kulanov, and V. Kharchenko, "Design fault inject ion-based technique and tool for FPGA projects verification," in *9th East-West Design and Test Symposium (EWDTS)*, September, 2011, pp. 191–195.

[50] H. Ziade, R. Ayoubi, and R. Velazco, "A Survey on Fault Injection Techniques," *The International Arab Journal of Information Technology*, vol. 1, no. 2, pp. 171–186, 2004.

[51] Robert Bosch GmbH. Functional Safety. [Online]. http://www.bosch-engineering.de/en/de/dienstleistungen/funktionale_sicherheit/funktionale_sicherheit_1.html

[52] Siemens AG. (2013) Introduction and Terminology for Functional Safety of Machines and Systems. [Online]. http://www.industry.siemens.com/topics/global/de/safety-integrated/maschinensicherheit/sicherheitsnormen/Documents/Introduction_and_Terminology_for_Functional_Safety_of_Machines_and_System_2013.pdf

[53] The International Electrotechnical Commission, Functional safety of electrical/electronic/programmable electronic safety-related, 2010.

[54] ISO 26262-10: Road vehicles — Functional safety, 2012.

[55] J. M. Stecklein, "NASA Johnson Space Center: Error Cost Escalation Through the Project Life Cycle," in *14th Annual International Symposium)*, Toulouse, France, June (2004).

[56] T. Tietjen, D.H. Müller, and A. Decker, "FMEA Praxis: Das Komplettpaket für Training und Anwendung," *Hanser Verlag*, 2011.

[57] M. Werdich, *FMEA - Einführung und Moderation.*: Springer-Verlag, 2011.

[58] Bertsche B, "Reliability in Automotive and Mechanical Engineering," *Springer Verlag*, 2008.

[59] CHESS: Composition with Gaurantees for High-integrity Embedded Software Components Assembly. [Online]. http://www.chess-project.org/

[60] SAFE: Safe Automotive soFtware architEcture. [Online]. http://www.safe-project.eu/

[61] SPES_XT: Software Plattform Embedded Systems "XT". [Online]. http://spes2020.informatik.tu-muenchen.de/spes_xt-home.html

[62] A. Birolini, "Quality and Reliability of Technical Systems," *Springer-Verlag*, 1994.

[63] Shooman Martin L, *Reliability of Computer Systems and Networks: Fault Tolerance, Analysis, and Design.*: Wiley-Interscience, 2001.

[64] Knight John, *Fundamentals of Dependable Computing for Software Engineers.*: Chapman and Hall/CRC Innovations in Software Engineering, 2012.

[65] Koren Israel and Mani Krishna C., *Fault-Tolerant Systems.*: Morgan-Kaufmann, 2007.

[66] Accellera.SystemC-AMS. [Online]. http://www.accellera.org/community/systemc/about-systemc-ams

[67] M.F.S Oliveira, Ch. Kuznik, W. Mueller, W. Ecker, and V. Esen, "A SystemC Library for Advanced TLM Verification," in *DVCon 2012, San Jose, USA*, 2012.

[68] Y. Vanderperren, W. Mueller, D. He, F. Mischkalla, and W. Dehaene, "Extending UML for Electronic Systems Design: A Code Generation Perspective," *Heterogeneous Embedded Systems - Design Theory and Practice, Springer Verlag*, 2010.

[69] MathWorks. (2016) Simulink Coder. [Online]. http://de.mathworks.com/products/simulink-coder/

[70] dSPACE.TargetLink. [Online]. http://www.dspace.com/de/gmb/home/products/sw/pcgs/targetli.cfm

[71] Synopsys. (2015, Nov.) Synopsys Virtualizer. [Online]. http://www.synopsys.com/Prototyping/VirtualPrototyping/Pages/virtualizer.aspx

[72] F. Bellard, "QEMU, a fast and portable dynamic translator," in *In Proc. of the USENIX Annual Technical Conference*, 2005.

[73] M. Gligor, Fournel N. N., and F Petrot, "Using binary translation in event driven simulation for fast and flexible MPSoC simulation," in *In Proc. of the Int. Conference on Hardware/software codesign and system synthesis (CODES+ISSS)*, Grenoble, France, 2009.

[74] W. Mueller, M. Oliveira, H. Zabel, and M. Becker, "Verification of Real-Time Properties for Hardware-Dependant Software," in *HLDVT2010*, Anaheim, 2010.

[75] H Wojtkowiak, "Test und Testbarkeit digitaler Schaltungen," *Teubner*, 1988.

[76] M.B. Santos and J.P. Teixeira, "Defect-oriented mixed-level fault simulation of digital systems on-a-chip using HDL and Design," in *Automation and Test in Europe Conference and Exhibition*, 1999.

[77] Z. Navabi, S. Mirkhani, M. Lavasani, and F. Lombardi, "Using RT Level Component Descriptions for Single Stuck-at Hierarchical Fault Simulation," *Journal of Electronic Testing*, vol. 20, no. 6, pp. 575-589, December 2004.

[78] R. Drechsler, S. Eggersglüß, G. Fey, and D. Tille, "Test Pattern Generation using Boolean Proof Engines," *Springer-Verlag*, 2009.

[79] S. Misera, H.T Vierhaus, and A. Sieber, "Fault injection techniques and their accelerated simulation in SystemC," in *In Digital System Design Architectures, Methods and Tools, 10th Euromicro Conference*, 2007.

[80] J. Perez, askasua M. Azkarate, and A Perez, "Codesign and Simulated Fault Injection of Safety-Critical Embedded Systems Using SystemC," in *European Dependable Computing Conference (EDCC)*, 2010, pp. 221 - 229.

[81] M.A. Kochte et al., "Efficient Simulation of Structural Faults for the Reliability Evaluation at System-Level," in *19th IEEE Asian Test Symposium (ATS)*, 1-4 Dec, 2010.

[82] N. Bombieri, F. Fummi, and G. Pravadelli, "A Mutation Model for the SystemC TLM 2.0 Communication Interfaces," in *Design, Automation and Test in Europe (DATE)*, 2008.

[83] Lisherness Peter and Cheng Kwang-Ting (Tim), "SCEMIT: A SystemC error and mutation injection tool," in *Design Automation Conference (DAC)*, 2010.

[84] R. Svenningsson, J. Vinter, H. Eriksson, and M. Trngren, *MODIFI - A MODel-Implemented Fault Injection Tool*.: Computer Safety, Reliability, and Security, p. 210 - 222, 2010.

[85] W. Lu and M. Radetzki, "SystemC-Fehlersimulation auf der Systemebene mit nebenläufigkomparativen," in *GI/GMM/ITG - Fachtagung Zuverlässigkeit und Entwurf*, Hamburg-Harburg , 2011.

[86] A. Braun, D. Lettnin, O. Bringmann, and W. Rosenstiel, "Simulation-Based Verification of the MOST NetInterface Specification Revision 3.0," in *Design, Automation and Test in Europe Conference and Exhibition*, Dresden, 2010.

[87] Siemens AG. Synchronous Motors. [Online]. https://w3.siemens.com/drives/global/en/motor/high-voltage-motors/synchronous-

motors/Pages/synchronous-motors.aspx

[88] Siemens AG. SIMOTICS M 1PH8 High Performance Induction and Main Spindle Motor. [Online]. http://www.industry.usa.siemens.com/drives/us/en/electric-motor/mc-motors/induction-motors/Pages/induction-motors.aspx

[89] Tom De Schutter, *Better Software. Faster!: Best Practices in Virtual Prototyping*.: Synopsys Press, 2014.

[90] Liyuan Zhang, Michael Glass, Nils Ballmann, and Jürgen Teich, "Bridging Algorithm and ESL Design: MATLAB/Simulink Model Transformation and Validation," in *Languages, Design Methods, and Tools for Electronic System Design. Springer-Verlag*, 2015.

[91] J. Boland, C. Thebeault, and Z. Zilic, "Using MATLAB and simulink in a SystemC verification environment," in *In preceedings of Design and Verification Conference*, 2005.

[92] B. Wang and J.S. Baras, "HybridSim: A Modeling and Co-simulation Toolchain for Cyber-physical Systems," in *17th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications*, Delft, The Netherlands, 2013.

[93] Abramovuci Miron, Breuer Melvin A, and Friedman Arthur D, "Digital Systems Testing and Testable Design," in *Computer Science Press*, 1990.

[94] N. Nagi and J.A. Abraham, "Hierarchical fault modeling for analog and mixed-signal circuits," in *Digest of Papers, 1992 IEEE VLSI Test Symposium*, April (1992).

[95] S. Sunter. Part1: Analog Fault simulation challenges and solutions.

[96] Kruseman B et al., "Defect Oriented Testing for Analog/Mixed-Signal Designs," in *IEEE Design and Test*, 2012.

[97] M. Sachdev, "A realistic defect oriented testability methodology for analog circuits," in *Journal of Electronic Testing: Theory and Applications, Vol. 6, pp. 265-276*, June (1995).

[98] IEEE Std.1666-2011 - Standard SystemC Language Reference Manual. (2011) [Online]. http://www.accellera.org/downloads/standards/systemc

[99] MathWorks. Simulating an Electronic System. [Online]. https://de.mathworks.com/help/physmod/elec/ug/simulating-an-electronic-system.html

[100] S. Moazzeni, S. Poormozaffari, and A. Emami, "An optimized simulationbased fault injection and test vector generation using VHDL to calculate fault coverage," in *In Microprocessor Test and Verification (MTV), 10th International Workshop*, Dec (2009).

[101] Bolchini Cristiana, Miele Antonio, and Sciuto Donatella, "Fault models and injection strategies in SystemC specifications," in *Digital System Design Architectures, Methods and Tools*, 2008.

[102] R. Baranowski et al., "Efficient Multi-level Fault Simulation of HW/SW Systems for Structural Faults," in *SCIENCE CHINA Information Sciences, Volume 54, Number 9, pp. 1784-1796*.

[103] M. A. Kochte et al., "Efficient Simulation of Structural Faults for the Reliability Evaluation at System-Level," in *Asian Test Symposium*, 2010.

[104] A. d. Silva and S. Sanchez, "Leon3 ViP: A virtual platform with fault injection capabilities," in *Digital System Design: Architectures, Methods and Tools (DSD), 13th Euromicro Conference on. IEEE*, 2010.

[105] A. d. Silva, P. Parra, O.R. Polo, and S. Sanchez, "Runtime instrumentation of systemc/tlm2 interfaces for fault tolerance requirements verification in software

cosimulation," in *Modelling and Simulation in Engineering, vol. 2014, p. 42*, 2014.

[106] V. Guarnieri et al., "Mutation analysis for systemc designs at tlm," in *Test Workshop (LATW), 12th Latin American*, 2011, pp. 1-6.

[107] Andrei Tabacaru Bogdan, Chaari Moomen, Ecker Wolfgang, and Kruse Thomas, "A Meta-Modeling-Based Approach for Automatic Generation of Fault-Injection," in *DVCon Europe*, 2014.

[108] Synopsys. SCML Reference Manual. Virtualizer Tool Suite Documentation.

[109] Synopsys, VDK Family for ARM® and DesignWare IP, Fast Software Bring-Up and Better Debug Control for ARMv7 and ARMv8 Processor-Based Designs.

[110] Tcl Developer Xchange. [Online]. https://www.tcl.tk/

[111] Bosch Sensortec. [Online]. https://www.bosch-sensortec.com/bst/products/motion/accelerometers/overview_accelerometers

[112] V-REP: Virtual Robot Experimentation Platform. [Online]. http://www.coppeliarobotics.com/

[113] Siemens. SINAMICS G120 standard inverters. [Online]. https://mall.industry.siemens.com/mall/en/de/Catalog/Products/10122015?tree=CatalogTree

[114] Siemens. (2006) Getting Started with the STARTER Commissioning Tool - 03/2006 Edition. [Online]. http://w3.siemens.com/mcms/mc-solutions/en/engineering-software/startdrive/pages/startdrive.aspx

[115] dSPACE. [Online]. https://www.dspace.com

[116] ControlDesk and Real-Time Interface (RTI). [Online]. http://www.dspace.com

[117] ASAM XIL Standard. [Online]. https://wiki.asam.net/display/STANDARDS/ASAM+XIL

[118] A. Mauderer, A. Schreiber, J. Chevalier, and J.H. Oetjens, "Automatisierte Integration von Simulink-Modellen in Virtuelle Plattformen," in *MATLAB EXPO*, München, DE, 2017.

[119] (2014) 1685-2014 - IEEE Standard for IP-XACT. [Online]. https://standards.ieee.org/findstds/standard/1685-2014.html

[120] A. Birolini, "Reliability Engineering: Theory and Practice," *Springer Verlag*, 2010.

[121] Riccardo Marino, Patrizio Tomei, and Cristiano, M. Verrelli, *Induction Motor Control Design*. London: Springer-Verlag London Limited, 2010.

[122] Yung-Yuan Chen, Chung-Hsien Hsu, and Kuen-Long Leu, "SoC-level risk assessment using FMEA approach in system design with SystemC," in *IEEE Symposium on Industrial Embedded Systems*, 2009.

[123] Lentijo, A., Monti, R. A., Dougal S., "Design of DC Motor Speed Control through Processor-in-the-Loop Approach," in *Huntsville Simulation Conference*, Huntsville, AL, October 2003.

[124] ARM Holdings. Fast Models Reference Manual. [Online]. https://developer.arm.com/products/system-design/fast-models

[125] S. Yang, D. Xiang, P. Bryant, L. Ran, and P. Tavner, "Condition monitoring for device reliability in power electronic converters: a review," in *IEEE Trans. on Power Electronics*, 2010.

[126] T. Orlowska-Kowalska and P. Sobanski, "Simple Sensorless Diagnosis Method for Open-Switch Faults in SVM-VSI-fed Induction Motor Drive," in *IEEE 39th Ann. Conf.*

*of the Ind. Electron. Soc.*, 2013.

[127] M. Alavi, D. Wang, and M. Luo, "Short-Circuit Fault Diagnosis for Three-Phase Inverters Based on Voltage-Space Patterns," in *IEEE Trans. on Ind. Electron*, 2014.

[128] L. Kwang-Woon, K., Jangho, Y. Myungchul, L. Kwang-Woon, and Y. JiYoon, "Condition Monitoring of DC-Link Electrolytic Capacitors in Adjustable-Speed Drives," in *IEEE Trans. on Ind. App.*, 2008.

[129] V., Forest, F. Smet et al., "Ageing and Failure Modes of IGBT Modules in High-Temperature Power Cycling," in *IEEE Trans. on Ind. Electron.*, 2011.

[130] edacentrum. (2013, October) EffektiV: Efficient Fault Simulation using virtual Prototypes for Qualification of Intelligent Motion-Control-Systems in Industrial Automation. [Online]. https://www.edacentrum.de/en/projects/EffektiV

[131] (2017) EffektiV: Project Final Report. [Online]. https://www.edacentrum.de/effektiv/en/content/the-project