

Christian Köhler

# Enhancing Embedded Systems Simulation

A Chip-Hardware-in-the-Loop Simulation Framework



Christian Köhler

Enhancing Embedded Systems Simulation

VIEWEG+TEUBNER RESEARCH

Christian Köhler

# Enhancing Embedded Systems Simulation

A Chip-Hardware-in-the-Loop Simulation Framework

VIEWEG+TEUBNER RESEARCH

Bibliographic information published by the Deutsche Nationalbibliothek  
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie;  
detailed bibliographic data are available in the Internet at <http://dnb.d-nb.de>.

Dissertation Technische Universität München, 2010

1st Edition 2011

All rights reserved

© Vieweg+Teubner Verlag | Springer Fachmedien Wiesbaden GmbH 2011

Editorial Office: Ute Wrasmann | Anita Wilke

Vieweg+Teubner Verlag is a brand of Springer Fachmedien.

Springer Fachmedien is part of Springer Science+Business Media.

[www.viewegteubner.de](http://www.viewegteubner.de)



No part of this publication may be reproduced, stored in a retrieval system or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the copyright holder.

Registered and/or industrial names, trade names, trade descriptions etc. cited in this publication are part of the law for trade-mark protection and may not be used free in any form or by any means even if this is not specifically marked.

Cover design: KünkelLopka Medienentwicklung, Heidelberg

Printed on acid-free paper

Printed in Germany

ISBN 978-3-8348-1475-3

# Acknowledgment

*No guilt is more urgent than to say, thanks.*

(Marcus T. Cicero, 106 B.C. - 43 B.C.)

I would like to comply with the given quotation by writing this acknowledgement. This thesis, which I wrote within the last three years at the Infineon Technologies AG in Munich, would not have been finished without the diverse support which I received.

First of all, I would like to mention my supervisor and team leader Albrecht at Infineon, who offered this possibility to me. Thank you Albrecht for the helpful suggestions, your openness, and your patience also to accept a complex solution on the way to the simple one.

Special thanks go to my doctoral advisor Prof. Herkersdorf from the Technical University of Munich. Thanks for your assistance as well as for your critical and perspicacious view at my approaches.

Many thanks go to Prof. Vierhaus from the Technical University of Cottbus. Thanks for your wide support of your current and your former students.

In addition, I would like to thank Prof. Gerndt for his spontaneous offer to act as additional co-examiner for my thesis.

A lot of thanks go to my colleagues at Infineon. Thank you all for your help to master the Infineon technologies and for your friendship. Thanks to Thomas, Andreas, Harry, Gerlinde, Udo, Jens, Angeles, Hans, Richard, and Michael and all the others who are not mentioned here.

One group of people is of extraordinary importance within the last weeks on a thesis: the proofreader. The proofreaders had the challenging task to find failures on about 160 content pages which I placed there very carefully. Thanks to Mario, Martin, Marcus, and Stefan, who are good friends of mine since my time in Cottbus.

Thanks go to Daniela and Nadja who very well demonstrated their knowledge of grammar on my thesis. In addition, I would like to mention and to thank my proofreaders Kirill and Paul from the U.S. Thank you all for your time!

Besides technical support, other assistance is needed to finish such work. I would like to thank my parents for their wide support and help during all the years.

Special thanks go to my brother who has the special ability to motivate me in many different ways. I am very proud that he finished his dissertation some weeks earlier than me. Thank you Micha!

Furthermore, I would like to thank all my friends. I will mention some of them: Nancy, who showed me more than once which are the possibilities of a single person to reinvent herself or himself; Lars, whose special and subtle humour can change a bad day to a good one; and last but not least Antje, who encouraged and supported me within the last years and who helped me discover new aspects of myself.

*Christian Köhler*

# Contents

<b>Acknowledgment</b>	<b>v</b>
<b>Contents</b>	<b>vii</b>
<b>List of Figures</b>	<b>xi</b>
<b>List of Tables</b>	<b>xv</b>
<b>Acronyms</b>	<b>xvii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Objective of the Thesis . . . . .	2
1.3 Thesis Organization . . . . .	3
<b>2 Related Work and Work Hypothesis</b>	<b>5</b>
2.1 Work Hypothesis . . . . .	5
2.2 Basics . . . . .	8
2.3 Possibilities of Hardware-Simulation-Coupling . . . . .	15
2.4 Additional Concepts . . . . .	23
2.5 Comparison of the Concepts . . . . .	26
2.6 Subproblems of Hardware-Simulation-Coupling . . . . .	26
<b>3 Connection between Hardware and Simulation</b>	<b>29</b>
3.1 Basics . . . . .	29
3.2 Related Work . . . . .	32
3.3 Exchange of Events between Microcontroller and Simulation	35
3.4 CHILS Event Exchange Mechanism . . . . .	42
3.5 Effects Caused by Coupling . . . . .	45
3.6 Summary . . . . .	53



<b>4</b>	<b>Interfaces between Microcontroller and Environment</b>	<b>55</b>
4.1	Related Work . . . . .	55
4.2	Interface Modelling and Abstraction . . . . .	58
4.3	Summary . . . . .	64
<b>5</b>	<b>Optimization – Coupling System Analysis</b>	<b>65</b>
5.1	Related Work . . . . .	66
5.2	Basics . . . . .	67
5.3	Formal Definitions . . . . .	68
5.4	Example . . . . .	74
5.5	Comparison of Different Coupling Systems . . . . .	77
5.6	Summary . . . . .	85
<b>6</b>	<b>Optimization – Analysis of the Real System and Environment</b>	<b>87</b>
6.1	Analysis of Algorithms . . . . .	87
6.2	Classification of Algorithms . . . . .	98
6.3	Analysis and Classification of Systems . . . . .	104
6.4	General Analysis . . . . .	109
6.5	Summary . . . . .	114
<b>7</b>	<b>Optimization – Runtime Analysis</b>	<b>117</b>
7.1	Metric Basics . . . . .	117
7.2	Metric Definitions . . . . .	118
7.3	Example . . . . .	122
7.4	Summary . . . . .	124
<b>8</b>	<b>CHILS Framework – Concept</b>	<b>127</b>
8.1	Functionality – CHILS Basics . . . . .	127
8.2	Microcontroller Requirements . . . . .	130
8.3	Range of Applications . . . . .	132
8.4	Features . . . . .	134
8.5	Optimization Flow . . . . .	139
8.6	Summary . . . . .	141
<b>9</b>	<b>CHILS Framework – Classification</b>	<b>143</b>
9.1	Software Models on Different Levels of Abstraction . . . . .	143
9.2	Embedding of Hardware in Simulations . . . . .	147
9.3	Comparison . . . . .	152
9.4	CHILS Performance and Accuracy . . . . .	159
9.5	Summary . . . . .	162

<b>10 CHILS Framework – Applications</b>	<b>163</b>
10.1 SystemC-Coupling . . . . .	163
10.2 Matlab/Simulink-Coupling . . . . .	168
<b>11 Summary and Outlook</b>	<b>175</b>
11.1 Summary . . . . .	175
11.2 Outlook and Future Work . . . . .	178
<b>A Formulas</b>	<b>181</b>
A.1 Chapter 5 . . . . .	181
<b>B Tables</b>	<b>183</b>
B.1 Chapter 3 . . . . .	183
B.2 Chapter 5 . . . . .	185
B.3 Chapter 9 . . . . .	187
<b>C Listings</b>	<b>189</b>
C.1 Chapter 8 . . . . .	189
C.2 Chapter 10 . . . . .	192
<b>D Datasheets</b>	<b>201</b>
D.1 Chapter 2 . . . . .	201
D.2 Chapter 5 . . . . .	202
<b>Bibliography</b>	<b>205</b>
<b>Glossary</b>	<b>217</b>

# List of Figures

1.1	CHILS Schematic Diagram . . . . .	3
1.2	Document Structure . . . . .	4
2.1	Simulation Performance [Goo08] . . . . .	6
2.2	OVP Core Model Performance in TLM 2.0 [OVP] . . . . .	7
2.3	Simulation Performance Challenge [DMMN03] . . . . .	7
2.4	System . . . . .	8
2.5	Processor-in-the-Loop . . . . .	22
2.6	Application of Simulation Techniques [Güh05] . . . . .	24
3.1	Simulator Coupling Adapted from [Bra06] . . . . .	32
3.2	Synchronization Model from [BNAA05] . . . . .	34
3.3	Simulation-Emulator Synchronization from [HSBG98] . . . . .	35
3.4	Alternating Execution . . . . .	37
3.5	Parallel Execution . . . . .	37
3.6	Alternating Execution – $\mu C$ Leads . . . . .	39
3.7	Alternating Execution – Simulation Leads . . . . .	39
3.8	Parallel Execution – Simulation Leads . . . . .	39
3.9	Alternating Execution – $\mu C$ Leads . . . . .	40
3.10	Alternating Execution – Simulation Leads . . . . .	40
3.11	Alternating Execution – $\mu C$ Leads . . . . .	41
3.12	Parallel Execution – Simulation Leads . . . . .	42
3.13	CHILS Event Exchange Setup . . . . .	43
3.14	CHILS Event Exchange Setup . . . . .	44
3.15	Compensation by Regaining of Lost Time . . . . .	53
4.1	SCE-MI Interface between SW Model and DUT [HKPS05] . . . . .	58
4.2	Module Concept . . . . .	61
4.3	ADC Module . . . . .	62
4.4	Receive Synchronization Driver - Monitor . . . . .	63
4.5	Send Synchronization Driver - Monitor . . . . .	64

5.1	HIL System . . . . .	65
5.2	HIL System and Real System . . . . .	67
5.3	MIMO System . . . . .	70
5.4	Heat-Sensor-in-the-Loop . . . . .	75
5.5	Heating System – Different Air Velocities . . . . .	77
5.6	Heating System – Different Materials . . . . .	78
5.7	System Comparison Modelling . . . . .	79
5.8	Scenario I – Results . . . . .	84
5.9	Scenario II – Results . . . . .	85
6.1	Profiling Flow . . . . .	93
6.2	Example-CFG-DFG . . . . .	95
6.3	Example-DFG . . . . .	96
6.4	Example-DFG-Condition-Labels . . . . .	97
6.5	Classification Flow . . . . .	101
6.6	Graph Matching – Covering the DFG . . . . .	102
6.7	Result-Recombination . . . . .	104
6.8	Rest Positions of a Mechanical System . . . . .	106
6.9	Numerical PID Control Analysis Results . . . . .	115
6.10	Numerical PID Control Analysis Results – Extract . . . . .	116
7.1	Measure of Changes of an Event Source . . . . .	120
7.2	Electric Motor Control via PID Controller . . . . .	122
7.3	Results Electric Motor Control . . . . .	123
7.4	Results Electric Motor Control (Detailed Section) . . . . .	124
8.1	TC1767ED $\mu$ C-EasyKit-Board . . . . .	128
8.2	CHILS Coupling Implementation between $\mu$ C and Simulation	129
8.3	CHILS Application in the V-Model . . . . .	133
8.4	CHILS Step Size Calculator . . . . .	139
8.5	Optimization-Flow . . . . .	140
9.1	VTB-PIL Coupling from [Len04] . . . . .	150
9.2	CHILS Synthetic Performance Values (Full Range of Appli- cation) . . . . .	161
10.1	SystemC Output . . . . .	164
10.2	CHILS Virtual Peripheral . . . . .	166
10.3	CHILS Virtual Peripheral Simulation Output . . . . .	169
10.4	TC1796 Simulink® Block . . . . .	171

10.5 Rocker with Air-Screw [Sch09] . . . . . 172

10.6 CHILS-Based Simulation and Real Control Path Measure-  
ments [Sch09] . . . . . 172

D.1 Block Diagram TC1796 . . . . . 201

# List of Tables

3.1	Basic Concepts [BNAA05]	30
3.2	Exchange Principles	38
3.3	STM Time Difference Measurements	47
3.4	GPTA Time Difference Measurements	48
3.5	Runtime of Additional CHILS Monitor Instructions	49
3.6	STM Time Difference (incl. and excl. Monitor Instructions)	52
4.1	Interface Abstraction [CGL <sup>+</sup> 00]	56
4.2	Interface Abstraction [JBP06]	57
4.3	$\mu$ C Interfaces Abstraction Levels	60
6.1	Database Size	105
6.2	Time for Database Filling	105
6.3	Time for Database Retrieval	105
6.4	Attributes of Nonlinear Elements [Wen07a]	108
7.1	Event Distribution Accuracy (Mean Values)	125
9.1	Abstraction Levels [JBP06]	144
9.2	$\mu$ C Modelling Abstraction Levels and Techniques	145
9.3	Performance of 6 Cylinder Engine Simulation with dSPACE	149
9.4	Comparison of CHILS with Different Hardware Solutions	153
9.5	Comparison of CHILS with Different Software Solutions	156
9.6	STM Time Accuracy Measurements	160
9.7	GPTA Time Accuracy Measurements	161
9.8	CHILS-MATLAB/Simulink Sample Application Performance	162
B.1	STM Time Difference Measurements – PWM Generation	183
B.2	STM Time Difference Measurements ( <i>isync/dsync</i> Difference)	184
B.3	System Comparison – Scenario I	185
B.4	System Comparison – Scenario II	186
B.5	Synthetic Performance Values (no Oversampling for Event Detection)	187

B.6 Synthetic Performance Values (10x Oversampling for Event Detection) . . . . . 188

D.1 DeskPOD™ Hardware Datasheets . . . . . 202

D.2 dSPACE HW Datasheets – DS2103 Multi-Channel D/A Board 202

D.3 dSPACE HW Datasheets – DS2211 HIL I/O Board . . . . . 203

D.4 dSPACE HW Datasheets – DS2211 HIL I/O Board . . . . . 203

D.5 dSPACE HW Datasheets – PHS Bus . . . . . 203

# Acronyms

## Symbols

$\mu\text{C}$

Microcontroller 1–3, 5–7, 9, 11, 13, 16, 18, 20–22, 25, 27, 35–46, 49, 50, 52, 54, 55, 59, 60, 64, 66, 78, 81–85, 87, 109, 114, 116–118, 122, 123, 125, 127–130, 132–137, 139, 141, 143–147, 149, 150, 152–159, 162, 163, 165, 166, 170, 171, 173, 175–179, 201

**A**

**ABS**

Anti-Lock Braking System 16

**ACSL**

Advanced Continuous Simulation Language 217

**ADC**

Analogue to Digital Converter 60, 61, 78–80, 82, 131

**ALU**

Arithmetic Logic Unit 217

**API**

Application Programming Interface 134, 136, 138, 141, 148, 180

**ASC**

Asynchronous Serial Channel 22, 59, 63, 64, 82, 83, 129

**C**

**CAN**

Controller Area Network 22, 64, 82, 83, 129, 131

**CFG**

Control Flow Graph 217



**CHILS**

Chip-Hardware-in-the-Loop Simulation 217

**CLCP**

Closed-Loop Characteristic Polynomial 110, 111

**CPU**

Central Processing Unit 5, 6, 22, 46, 47, 49–51, 130, 139, 149, 156, 159

**CSSC**

CHILS Step Size Calculator 218

**D****DAC**

Digital to Analogue Converter 78–80

**DAP**

Device Access Port 128, 159

**DAQ**

Data Acquisition 16, 18–20, 82, 148

**DAS**

Device Access Server 218

**DE**

Driving Environment 57

**DESS**

Differential Equation System Specification 13, 35

**DEVS**

Discrete Event Systems Specification 12, 35

**DFG**

Data Flow Graph 218

**DLL**

Dynamic Link Library 128, 163, 168

**DMA**

Direct Memory Access 131

**DSP**

Digital Signal Processor 5

**DTSS**

Discrete Time System Specification 12

**DUT**

Device under Test 57

**E**

**ECU**

Electronic Control Unit 218

**F**

**FADC**

Fast Analogue Digital Converter 61

**FIFO**

First In - First Out 56

**FPGA**

Field Programmable Gate Array 19–21, 25, 26, 35, 151, 153

**FS**

Full Scale 218

**FSR**

Full Scale Range 218

**G**

**GPIO**

General Purpose Input/Output 59, 61, 129, 131, 162

**GPTA**

General Purpose Timer Array 5, 45, 46, 61, 131, 159

**H**

**HDL**

Hardware Description Language 34, 58, 145

**HIL**

Hardware-in-the-Loop 219

**I****I/O**

Input/Output 2, 18–21, 78, 82, 128, 147, 148, 160

**IC**

Integrated Circuit 146, 147

**ISS**

Instruction Set Simulator 145, 219

**J****JTAG**

Joint Test Action Group 128, 149

**K****kNN**

k-nearest neighbors 219

**L****LSB**

Least Significant Bit 219

**LTI**

Linear Time Invariant 220

**M****MCD**

Multi-Core Debug 220

**MCDS**

Multi Core Debug System 220

**MIL**

Model-in-the-Loop 220

**MIMO**

Multiple Input Multiple Output 220

**MIPS**

Million Instructions per Second 220

**P****PC**

Personal Computer 5, 16–18, 23, 43, 49, 59, 81, 128, 134, 148, 151, 159, 162, 163

**PGA**

Programme Graph Analysis 221

**PID**

Proportional–Integral–Derivative 221

**PIL**

Processor-in-the-Loop 221

**PLI**

Program Language Interface 148

**PLL**

Phase-Locked Loop 221

**PWM**

Pulse-Width Modulation 32, 46, 159

**R****RCP**

Rapid Control Prototyping 221

**RTL**

Register Transfer Level 55–58, 144–146, 151, 152, 221

**S****SCE-MI**

Standard Component Emulator Modelling Interface 58, 60

**SIL**

Software-in-the-Loop 222

**SISO**

Single Input - Single Output 222

**SoC**

System-on-Chip 6, 11, 25, 26, 57, 89, 136, 143, 151, 175, 222

**SPICE**

Simulation Programme with Integrated Circuit Emphasis 222

**SSC**

Synchronous Serial Channel 22, 62–64, 82, 83

**STM**

System Timer 45, 46, 49, 159

**SVM**

Support Vector Machines 222

**T****TCS**

Traction Control System 16

**TLM**

Transactional Level Modelling 223

**U****UML**

Unified Modelling Language 223

**USB**

Universal Serial Bus 127, 128

**V****VLIW**

Very Large Instruction Word 223

**VTB**

Virtual Test Bed 149, 223

# 1 Introduction

The complexity of embedded hardware/software systems increases every year. For example, the currently implemented embedded systems in cars consist of 10 to 100 million transistors per chip, 1 to 10 million lines of code, and 10 to 100 microcontrollers ( $\mu$ Cs) per car. Especially the automotive industry requires high-end  $\mu$ Cs for Electronic Control Units (ECUs) to fulfil the government's requirements on emissions reduction as well as the customers' demands for comfort, safety and reliability. Today, most failures in cars are caused by electronic components and not by mechanical ones anymore [Jac03]. In addition, the cycle times for new products last only a few years in the automotive industry and just several months in the communications industry.

System modelling becomes even more important to handle the high demands on complexity, reliability, and short development time mentioned above. Model-based development offers possibilities like early exploration of system designs, rapid prototyping, and extended system verification, optimization and testing. It allows the engineers to develop software and hardware in parallel to reduce the cycle times. Furthermore, these systems are often a recombination or extension of existing parts to reduce the development effort.

## 1.1 Motivation

The present development in high-tech industry causes an exponential growth of system simulation performance requirements that cannot be covered by the rising simulation computer performance. The current method is to use different levels of abstraction, since a higher level of abstraction increases simulation speed. Modelling is a trade-off between high speed, high accuracy and low effort. It is easy to create a model covering two of these three attributes but it is nearly impossible to build high speed and high accuracy models spending only a small amount of effort. This trade-off is especially critical if system software development is supposed to begin

on the simulated system hardware. Because of this, software developers need highly accurate and fast models.

## 1.2 Objective of the Thesis

Many complex hardware/software systems are a combination of existing parts like  $\mu$ Cs, memory subsystems, interconnect structures, input/output (I/O) modules and hardware accelerators. The possibility to embed a real  $\mu$ C into a system simulation of a larger technical context would be a great benefit. Compared to a model, the real  $\mu$ C will run with high speed and full accuracy. This is an advantage especially for early software development. The challenge is to embed the  $\mu$ C into the simulation environment so that it is transparent to the system simulation whether it is a simulated or a real  $\mu$ C. This kind of solution needs to meet different demands, namely:

- The data exchange and synchronization between real hardware and simulated hardware has to be realized in an efficient and effective way.
- A certain accuracy of the coupling system has to be guaranteed.
- The hardware has to be coupled with different simulation environments.
- In order to reduce the amount of exchanged data a suitable level of abstraction for the  $\mu$ C interfaces and peripherals has to be found.
- Transparency of the coupling between simulated environment and the  $\mu$ C to the executed  $\mu$ C is needed to minimize the influence of the coupling to the simulation results.
- The hardware effort for coupling determines the practicability of the solution.

With respect to the existing concept of Hardware-in-the-Loop (HIL) simulation, the presented approach is called Chip-Hardware-in-the-Loop Simulation (CHILS) (see figure 1.1). HIL simulation is a technique that is used in the development and testing of complex embedded systems. In a HIL simulation, the represented system consists of the simulated part and a real part, the “hardware-in-the-loop”. CHILS meanwhile focuses on complex hardware/software systems designed with existing  $\mu$ Cs.

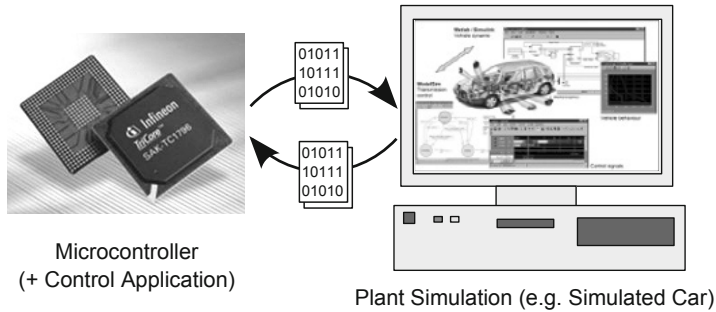


Figure 1.1: CHILS Schematic Diagram

## 1.3 Thesis Organization

The thesis organization is shown in figure 1.2. The following chapter includes the working hypothesis, related work and the basics of the thesis. Three main topics are identified: the connection between hardware and simulation, the representation of  $\mu\text{C}$  interfaces in the coupling of hardware and simulation, and the event exchange optimization between hardware and simulation. These topics are examined in the following five chapters. Chapter 3 introduces the  $\mu\text{C}$  hardware to simulation coupling<sup>1</sup> which is used to realize the CHILS approach. The representation of the  $\mu\text{C}$  interfaces, from the simulation point of view and from  $\mu\text{C}$  point of view, is explained in chapter 4. Chapters 5, 6 and 7 cover different aspects of the hardware to simulation coupling optimization. Chapter 8 gives an overview of the complete CHILS framework. Chapter 9 covers the classification of the CHILS framework. A comparison with other solutions is presented. Some applications for the CHILS approach are shown in chapter 10. Finally, the conclusions can be found in chapter 11.

<sup>1</sup> The term hardware to simulation coupling as well as the terms  $\mu\text{C}$  to simulation coupling and  $\mu\text{C}$  hardware to simulation coupling denote all the same setup of a coupling between a  $\mu\text{C}$  and a simulation environment on the PC.



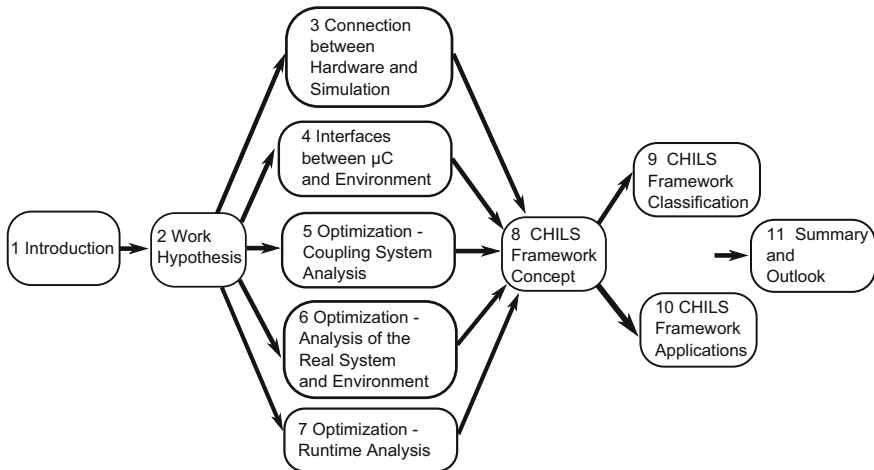


Figure 1.2: Document Structure

## 2 Related Work and Work Hypothesis

The dilemma of modelling complex hardware/software systems is the combination of the three main aspects: speed, accuracy and effort. The target is to have high speed models with high accuracy, combined with a low effort for model creation and application. For  $\mu$ C applications this means to have nearly 100 percent cycle accurate models, which are created or generated automatically, and which run in real time on a standard Personal Computer (PC). This requirement is nearly impossible to fulfill.

### 2.1 Work Hypothesis

Figure 2.1 shows performance measurements presented by the company *Target Compiler Technologies* [ReT] at the conference and exhibition *multi-coreEXPO* in 2008 [Goo08]. A cycle accurate Instruction Set Simulator (ISS) has a performance of about one Million Instructions per Second (MIPS). This is very fast for a cycle accurate model, but this model is just an ISS, so it contains no peripheral module models. A  $\mu$ C contains a lot of peripheral modules in addition to the main core, the Central Processing Unit (CPU). The target device used in this thesis, the high-end 32bit TriCore®  $\mu$ C<sup>1</sup> series shown in figure D.1, contains several complex peripherals like the General Purpose Timer Array (GPTA). The GPTA provides a set of timer, compare and capture functionalities, which can be flexibly combined to form signal measurement and signal generation units. The combination of over one hundred timers and thousands of trigger combinations between them makes this module hard to model. A speed-up in modelling is mostly achieved by reducing the events<sup>2</sup> and the state changes produced within a model. If hundreds of sources and sinks for events exist, for example inside the GPTA, the task becomes very difficult. A less accurate model, with the focus set to instruction accuracy and not in addition to timing

---

1 The TriCore is a 32-bit  $\mu$ C-Digital Signal Processor (DSP) architecture which is optimized for real-time embedded systems. Its main applications are engine control, body and safety applications within the automotive sector and industrial applications.

2 An event can be defined as an indicator for a change [ZPK00].

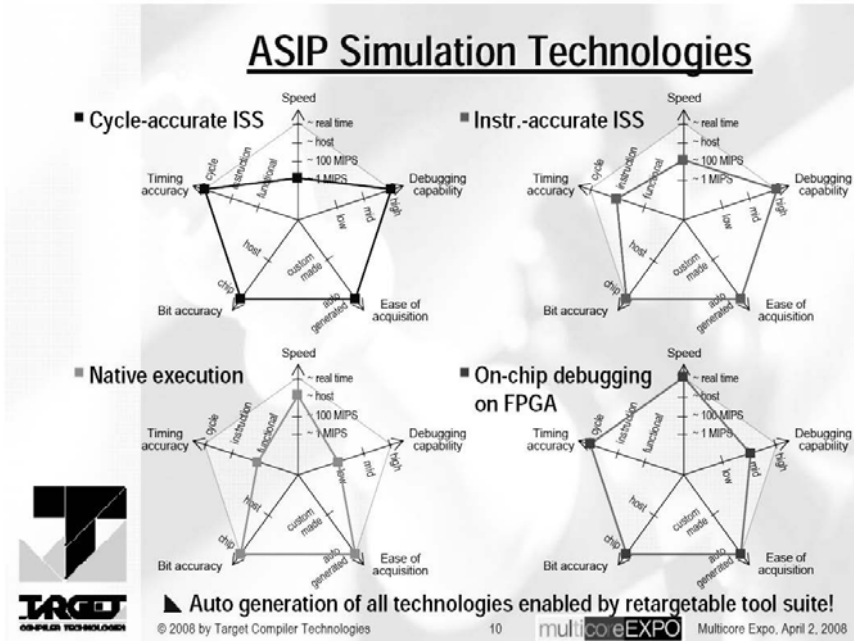


Figure 2.1: Simulation Performance [Goo08]

accuracy, can achieve about 100 MIPS (see figure 2.1). Such an instruction accurate model will be less time accurate, because of the simplification of the instruction execution process (pipelining is often not included) and of the memory access (the cache behaviour is not modelled). A complete  $\mu$ C modelled on that abstraction level can have a performance which is the tenth part of a model which only covers the CPU. Within the project *Open Virtual Platforms* [OVP] core models modelled in Transactional Level Modelling (TLM) 2.0<sup>3</sup> with a performance of some hundred MIPS can be found. The peak performance is higher than 1000 MIPS (see figure 2.2), but these are only models of processor cores without peripherals.

The raising abstraction level of models primarily addresses the biggest problem of current Systems-on-Chip (SoC) and  $\mu$ C modelling. The available performance grows slower than the demand for it, caused by the ris-

3 TLM 2.0 is the current standard in transaction level modelling. TLM is mostly connected with SystemC [Sys]. It is an interface standard that enables the interoperability and reuse of SystemC models at the transaction level.

TLM2.0/OVP	Single: ARM Core			MultiCore: 2 MIPS Cores			ManyCore: 24 ARC Cores		
Benchmark	Simulated Instructions	Run time	Simulated MIPS	Simulated Instructions	Run time	Simulated MIPS	Simulated Instructions	Run time	Simulated MIPS
linpack	678,061,601	1.33s	<b>435</b>	442,853,216	2.61s	<b>170</b>	100,419,903,936	348.56s	<b>288</b>
fibonacci	320,512,265	1.35s	<b>243</b>	1,075,056,992	4.21s	<b>255</b>	14,692,840,992	61.48s	<b>239</b>
Dhrystone	4,544,054,499	12.29s	<b>370</b>	1,010,144,280	2.36s	<b>427</b>	14,977,987,872	10s	<b>365</b>
peakSpeed1	5,600,001,624	4.93s	<b>1137</b>	11,200,002,620	9.36s	<b>1197</b>	144,000,042,864	147.83s	<b>974</b>
All measurements on 3 GHz Pentium									

Figure 2.2: OVP Core Model Performance in TLM2.0 [OVP]

ing complexity of such systems. “Simulation Can’t Keep Pace with Design Size” [DMMN03].

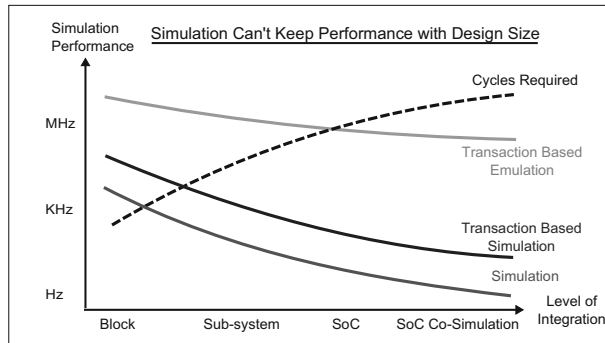


Figure 2.3: Simulation Performance Challenge [DMMN03]

Beyond the modelling itself, another important part is the verification of the correctness of the model. The model is useless if the functional correctness is not verified. The silicon test benches for  $\mu C$  verification will fail on a model of a high abstraction level, because the tests require a cycle accurate behaviour <sup>4</sup>.

All this leads to the idea to use the real  $\mu C$  as a replacement for a modelled  $\mu C$  inside of a system simulation. The  $\mu C$  is already verified and the maximum performance reachable with a  $\mu C$  is its natural performance. The overall system performance is primarily limited by the performance of the coupling system between real hardware and simulation.

<sup>4</sup> The test benches are composed of sets of input data and sets of expected output data including hard deadlines. The tests will fail if the system-under-test-response does not match with the expected response in time.

## 2.2 Basics

The introduction refers to the importance of system modelling and simulation and the complexity of doing so. This section introduces the motivation behind these techniques. Furthermore the different types of simulations are presented as well as the basics of modelling. Modelling is the task to find a suitable description of a system. The *Deutsches Institut für Normung* (German Institute for Standardization) defines in DIN 19226 [DIN] a **system** as follows:

**Definition 2.2.1.** *A system is an arrangement of entities in an observed context which are related to each other. This arrangement is separated from the environment by defined conditions.*

*Ein System ist eine in einem betrachteten Zusammenhang gegebene Anordnung von Gebilden, die miteinander in Beziehung stehen. Diese Anordnung wird aufgrund bestimmter Vorgaben von ihrer Umgebung abgegrenzt.*

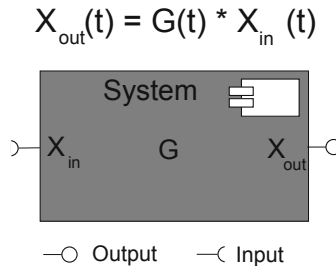


Figure 2.4: System

This is a general definition but it contains the important aspects. A system consists of related parts and it can be separated from its environment. In the book *Signale und System* (Signals and Systems) [Kie98] Uwe Kiencke defines a system as a construction which reacts on an input signal with an output signal (figure 2.4). A **signal** is defined as a physical quantity which changes over time and contains information. So another property of a system is the existence of inputs and outputs.

**Definition 2.2.2.** *In summary a system can be defined by (see [Abe03a]):*

- the system boundary (real physical boundary or imaginary boundary)
- input and output variables which pass the boundary

- *internal variables (the internal states)*
- *the behaviour which defines the relation between input, output and internal states*

DIN 19226 [DIN] includes in addition a definition for a **model**:

**Definition 2.2.3.** *A model is a mapping of a system or a process to another conceptual or representational system, which is created by using known principles, an identification or assumptions. The model has to represent the system or the process sufficiently accurately with respect to the question at hand.*

*Ein Modell ist die Abbildung eines Systems oder Prozesses in ein anderes begriffliches oder gegenständliches System, das aufgrund der Anwendung bekannter Gesetzmäßigkeiten, einer Identifikation oder auch getroffener Annahmen gewonnen wird und das System oder den Prozeß bezüglich interessierender Fragestellungen hinreichend genau abbildet.*

The phrase “sufficiently accurately” is the most interesting part of the definition. The needed accuracy depends on the field of application of the model. A model will never be identical with the system which is modelled. It is just a representation of the system or rather a representation of certain aspects of the system. Different modelling principles exist to model these certain aspects. If the target is to model a  $\mu\text{C}$  or a System-on-Chip (SoC), the modeller can fulfil the task for example in the physical domain by modelling the transistors and their charge, or he or she can write a model which realizes the pure functionality and no physical aspects.

### 2.2.1 Motivation for Modelling and Simulation

Now the question is why do we need to model and simulate a system? Due to the rising complexity of systems in all domains it is not reasonable to design and build real systems via trial and error from scratch. The effort in terms of time and cost would be too high. The design of real prototypes has to be reduced. The goal is to build a system “first time right”, so that the first prototype works properly. Modelling and simulation are the tools to reach that goal. André Lüdecke mentions in *Simulationsgestützte Verfahren für den Top-Down-Entwurf heterogener Systeme* [Lüd03a] the following motivations for simulations.

- The system specification can be validated very early (Am I building the product right?). The system design could be wrong because of system interactions which were not understood or not taken into account during the specification phase. A simulation can help to discover such failures and leads to a deeper understanding of a system.
- Components are sometimes not available in an early stage of system development. Especially if systems are developed in a top-down approach. At an early stage of development, only the specification of the whole system is available. Simulations can help to make early design decisions regarding the components and reduce the iterations in the design process. A big issue in the design of combined hardware/software systems is the hardware/software partitioning. The main question is which part of the system should be software (more flexibility) and which part should be hardware (more efficiency). A lot of research is done in this area (a short overview can be found in *Hardware/Software Codesign – Pedagogy for the Industry* [HTW<sup>+</sup>08]).
- Sometimes the observation of internal states of a system is not possible in an experimental environment. A simulation enables the user to observe all internal states, variables and parameters (as long as they are modelled). Moreover, the functional and the timing behaviour of the simulation will not change if the observation is running, unlike the behaviour of the real hardware.
- The costs are very high for measurements with real prototypes. A simulation is often cheaper than a real experiment (think about a car crash simulation in automotive industry). Experiments are mostly very time consuming and require cost intensive measurement equipment. The main purpose is to use experiments just for verification of the models.
- Experiments with real prototypes can be dangerous and destructive. Simulation enables to explore the system limits.

Simulation is a powerful tool in system design, but one has to be aware that a model is only a simplified representation of the system.

### 2.2.2 Modelling Basics

In general, the differentiation between **structural** and **pragmatic modelling** can be made (see [PLG02]). **Structural modelling** is also called **white-box modelling**. The inner structure of the systems is known. The modelling is done by abstracting the given system parts. At **pragmatic** or **black-box modelling** the inner structure is unknown. Only the interaction of the systems, the external behaviour or the reaction on signals can be observed. The modelling is done by the imitation of the observed behaviour as so-called **behaviour models**. This is also called behaviour modelling (see [Lüd03b]). Combinations of both approaches are called **grey-box modelling**. This is often the most efficient way.

**Behaviour models** represent the system by equations or tables, for example as a characteristic curve. The behaviour of a system can be described by underlying physics of the system or by measured or simulated input and output values. A typical example is the control path of the CHILS demonstrator system presented in chapter 10 in subsection 10.2.1. The demonstrator consists of a rocker with an air-screw attached, which is controlled by a TC1796  $\mu$ C. The rocker-air-screw system is modelled by determining the transfer function of the system by measurements.

**Structural models** consist of mostly hierarchical sub-models. These sub-models can also contain other sub-models or basic models. These basic models can be structural or behaviour models. A typical example of structural models are models of SoCs or  $\mu$ Cs. Both structural and behaviour models can be used on different levels of abstraction. A good overview can be found in Daniel D. Gajski's and Frank Vahid's book *Specification and Design of embedded Systems* [GVNG94].

Additional differentiations between **heterogeneous** and **homogeneous modelling** are possible. Modelling languages and simulation systems are often bound to special domains. In **heterogeneous modelling**, a specific language and/or simulation environment can be applied for each domain of a system to create the model. So it is possible to choose the language and/or simulation environment which fits best to the modelling task. This can reduce the effort for modelling enormously. Sometimes this technique is called **multi-language approach**. The major problem arising from multi-language modelling is the coupling of different simulation environments. This topic is discussed in detail in chapter 3.

In **homogeneous modelling** or **single-language modelling**, the system is described by only one modelling language. Only one simulator is needed



so problems caused by coupling are eliminated. The whole handling of the simulation is easier. Perhaps some parts of the system have to be described in a language that does not fit well to the domain of these parts. This can be a disadvantage.

### 2.2.3 Simulation Systems

Simulation is the method to perform experiments on models of systems. During the simulation the state of the model changes. This state change can be seen as **simulated time or simulation time**. This internal time is independent from the **simulation runtime**, the external time. It is called **realtime simulation** if the simulated time and the runtime of the simulation are identical. Furthermore, the (software) environment running the simulation is called a **simulator**.

Depending on the domain of modelling different types of simulation techniques and principles have been developed. Basic theoretical contemplations can be found in Bernard P. Zeigler's, Herbert Praehofer's and Tag Gon Kim's book *Theory of modelling and simulation - integrating discrete event and continuous complex dynamic systems* [ZPK00]. They differ between three respectively two basic types. These are:

**DESS** - Differential Equation System Specification - continuous in time and values

**DTSS** - Discrete Time System Specification - a time discrete system

**DEVS** - Discrete Event Systems Specification - it can be seen as more general version of DTSS <sup>5</sup>

An even more detailed hierarchy of simulation methods can be found in *Diskrete Simulation – Prinzipien und Probleme der Effizienzsteigerung durch Parallelisierung* written by F.Mattern and H. Mehl [MM89]. The main differentiation is also between **continuous** and **discrete simulation systems**, but the authors do not see Discrete Event Systems Specification (DEVS) as an even more general version of Discrete Time System Specification (DTSS).

---

<sup>5</sup> A change of a state can happen at every point of time, in contrast to a DTSS system. Time steps are variable and they will be adapted to the events. The point in time of the next event has to be known.

DESS systems are often called continuous simulation systems<sup>6</sup>. F.Mattern's and H. Mehl's hierarchy is defined as follows:

- continuous
- discrete
  - event-driven
    - \* event-oriented
    - \* activity-oriented
    - \* process-oriented
    - \* transactional
  - time-driven
    - \* quasi-continuous

The principles of continuous and discrete simulation are independent from the level of abstraction. Further explanations about different levels of abstraction are given in chapter 9. The CHILS Framework is compared with  $\mu$ C models of different abstraction levels. An overview about these simulation methods can be found in Stefan Eilers *Zeitgenaue Simulation gemischt virtuell-realer Prototypen* [Eil06].

### 2.2.3.1 Continuous Simulation

Time and values are continuous in a continuous simulation, so the model changes continuously. The models are often based on differential equation systems, which are solved by numerical integration. In reality the simulation is only quasi-continuous due to the numerical solving on a computer. The time steps, which are performed by the numerical solver, are often adapted to the differential equation system which is to be solved. This is caused by the non-exact solving behaviour of numerical algorithms. An exact analytical solution is not possible for general differential equation systems (see [Bec05]).

In continuous (or quasi-continuous) simulators like Simulink®, the numerical solver can be chosen by the user. There are different numerical methods: the single-step and the multi-step method. The accuracy of the

---

<sup>6</sup> In fact most continuous simulation system are only quasi-continuous due to the discrete nature of simulation computers. From now on the term continuous simulation or Differential Equation System Specification (DESS) is used for that kind of simulation system.

methods depends on their order. A method of a higher order usually leads to a higher computational resource consumption for calculation. Willi Törnig and Peter Spellucci provide a good overview about these methods in the book *Numerische Mathematik für Ingenieure und Physiker* [TS90].

### 2.2.3.2 Discrete Simulation

In a discrete simulation, the state changes of the model occur at determined points in time. The time jumps from one change to the other. The state change is caused by incrementing the simulation time, which is called **time-driven**, or caused by events, which is called **event-driven**. A time-driven simulation can run on an event-driven simulator by generating time-events (see definition in Bernard P. Zeigler's book [ZPK00]).

**Definition 2.2.4.** *An event is an indicator for a change.*

**Event-driven Simulation** In an event-driven simulation events, which are marked by time stamps, cause the state changes of the model. If the event-driven simulation contains only one model, which runs in one simulator, the functionality can be described as follows. A global event queue sorts the events based on their time stamps. The simulator chooses a non-processed event with the smallest proximity in time in difference to the actual simulation time. Then the simulation time is set to the time stamp of this event and the event is processed. The event processing changes the state of the model. New events can be produced and these events are sorted into the global event queue. If more than one model (and possibly more than one simulator) are used, the models have to be synchronized, because each model has its own simulation time. Stefan Eilers [Eil06] mentions two principles to obtain the causal order of the events: the **conservative** and the **optimistic principles**.

**Conservative principles** guarantee the maintenance of the causal order. The simulation time is only incremented if all relevant events can be determined. The simulation step size is chosen in the way that all events are processed to the exact point of time of occurrence. Sometimes this can cause deadlocks or slow down the simulation by producing empty synchronization events.

**Optimistic methods** are based on the assumption that violations of the causal order are seldom. The simulation is incremented even if not all relevant events are known. If an event is detected with a time-stamp

smaller than the actual simulation time, the simulation is set back to a point of time before the event occurs. Now the next step can be set exactly to the right point of time. This principle is called rollback or backtracking. Previous simulation states have to be stored. One version of the rollback simulation is the **time-warp method**.

These principles are basics of simulator coupling and Co-simulation. In chapter 3 *Connection between Hardware and Simulation* this topic is evaluated. In general, event-driven simulation is very efficient because computation time is only consumed if state changing events occur. Dead times with no changes do not consume computation time, unless all relevant events are known. A popular event driven simulator is the SystemC simulation framework [Sys] which is one of the target frameworks to couple with CHILS.

**Time-driven Simulation** The simulation time is incremented in fixed or variable steps (see [Eil06]). All events with a time stamp between the last and the present simulation point of time are processed. It is important that the time step is small enough, so that the events processed in the same interval only influence events in simulation future. Time-driven simulation is normally slower than event-driven simulation because time steps without processed events must not be ignored.

A general time-driven simulator works as follows ([HP02]). First the simulator initializes the system state and simulation time. As long as the simulation is not finished, the following steps have to be repeated: collecting statistics about the current state, handling events that occurred between the last step and incrementing the simulation time. Time-driven simulators are used for example in network simulations.

## 2.3 Possibilities of Hardware-Simulation-Coupling

The concept of hardware-simulation coupling has been existing for several years. Sheran Alles, Curtis Swick, Syed Mahmud and Feng Lin [ASML92] present an integrated HIL simulation systems which does not differ from actual systems. The simulation runs on standard PC hardware, while the coupling is realized with special I/O cards.

Paul Baracos defines the term HIL simulation in the following way [BMRJ01]:

**Definition 2.3.1.** *In HIL, either a simulated plant is wired to a real controller, or a real plant to a simulated controller.*

So two types of HIL simulations can be distinguished. In *Design of the Embedded Software Using Flexible Hardware-In-the-Loop Simulation Scheme* D. Virzonis, T. Jukna and D. Ramunas call these types **plant simulation** and a **controller behaviour simulation** [VJR04]. Other terms with identical meaning or specialization can be found in literature, too.

Within a **plant simulation** the mathematical model of the control object, called plant, is running on a general purpose computer. For example, the controller is connected via a Data Acquisition (DAQ) board with the simulation PC. The **controller behaviour simulation** inverts the situation. The prototype of a control programme runs on a general purpose computer which is connected to a real plant for example through a DAQ board. Sometimes special simulation computers instead of general purpose computers are used to run the simulation. The solutions of dSPACE [dsp] are one example (see subsection 2.3.2.2). They will be discussed later in this chapter.

### 2.3.1 Motivation for Hardware-in-the-Loop

HIL simulation is a widely used concept. Especially within the automotive industry, it is used mainly for rapid prototyping, test and optimization of systems. A typical example of HIL simulations is a connection between a complete ECU (including a  $\mu$ C) and a simulated car environment. The environment is the plant, while the ECU is the controller in this setup. [WMH04],[Güh05], and [PSF04] are typical examples for this application. Jae-Cheon Lee and Myung-Won Suh describe in [JCL99] a HIL simulator for testing an ECU for an Anti-Lock Braking System (ABS) and Traction Control System (TCS). Such an application is a typical HIL supported test scenario.

Several reasons lead to the development of HIL simulation systems (see [Abe03b], [WMH04] and [Rot04]).

- Critical scenarios can be tested without compromising hardware or people. Safety-critical components (such as drive-by-wire) can be tested without danger.

- Test automation is possible.
- HIL tests are cost-efficient tests. A scenario is easier to configure in a virtual environment than in a real environment. So, for instance, fewer test drives and test bench experiments are needed, and also fewer vehicle prototypes or vehicle components – resulting in cost savings.
- On one hand, the virtual environment can be easily modified, so a real control (the real ECU) can be tested in different control loops. On the other hand, a real control loop (for example an engine) can be easily tested with different simulated controllers (or control algorithms). Difficult ambient conditions (winter tests, rain, high or low temperatures) can be simulated by simple parameter changes.
- Tests are immediately repeatable, so component failures and associated emergency scenarios can be tested reproducibly.
- The hardware can be tested in parallel in different working environments (for example in a climate chamber).
- If some parts of a system are too complex to model, they can be replaced inside the simulation by the real hardware.
- The embedded real hardware can accelerate the whole simulation.
- The hardware and software of ECU can be tested at an early stage of development. A real engine or a real transmission is not needed.

### 2.3.2 Commercial Hardware-in-the-Loop Solutions

Especially the importance of HIL-simulation for the industry leads to a wide offering of commercial solutions. Simulation software like MATLAB®/Simulink® is well established to design complex systems. HIL solutions need to offer the interface between the simulation software and the “hardware-in-the-loop”. Depending on the field of application, the system also has to be real-time capable (for example for ECU-in-the-Loop scenarios). The following simulation frameworks are an overview about available commercial solutions. The principles are more or less identical. A standard PC or a dedicated simulation computer, often designed from standard PC hardware, hosts the simulation while the connection to the

hardware is realized by DAQ cards or external units. Classical HIL simulations require more often than not real-time capable simulations, because the hardware-in-the-loop, like an ECU, cannot be slowed down.

### 2.3.2.1 MathWorks™

MATLAB® from MathWorks™ [Mat] is a well known solution for technical computing. It offers great capabilities for numerical computations. Based on this computation solution, MathWorks™ delivers Simulink®. Simulink® is an environment for model-based design of complex dynamic systems. Systems can be designed on a high level by modelling the interactions between functional blocks. The calculations of the blocks are done by MATLAB®.

With xPC Target™, MathWorks™ offers a solution for rapid prototyping and HIL-simulation (see [xPC09]). xPC Target™ delivers I/O interface blocks for Simulink® models. These interface blocks represent I/O boards inside a PC. xPC Target™ executes the models under a realtime kernel on the PC. The “hardware-in-the-loop” is attached to the I/O boards.

MATLAB®/Simulink® is one example for a simulation environment which can be coupled to a  $\mu$ C via the CHILS approach. The CHILS demonstrator platform presented in chapter 10 *CHILS Framework - Applications* uses MATLAB®/Simulink® to simulate the control path. In addition, MathWorks™ offers an auto-code generator bundled with a Processor-in-the-Loop (PIL) solution<sup>7</sup> which is called *Link for TASKING®* [Lin09]. *Link for TASKING®* links MATLAB® and Simulink® to the TASKING [TAS] environment. Depending on the  $\mu$ C, the generated code runs on an ISS or a real device.

### 2.3.2.2 dSPACE

dSPACE [dsp] offers hardware and software for rapid control prototyping and HIL simulation. MATLAB®/Simulink® is used for high level modelling while dSPACE provides I/O hardware for hardware to simulation coupling and additional software for example for target code generation<sup>8</sup>

<sup>7</sup> PIL simulation is special kind of a HIL or a Software-in-the-Loop (SIL) simulation for complex embedded systems which consist of a plant part and a controller part. The control algorithm interacts with the model of the plant. It can be executed either on the real  $\mu$ C or on an ISS.

<sup>8</sup> A target code generator is able to transform the an high-level description of an algorithm into source code (for example C-code) for a target system (normally a  $\mu$ C).

from models. The dSPACE simulator hardware for HIL simulation is modular. In addition, I/O boards can be added. Complete solutions, for example for rapid control prototyping, are available. The automotive industry is one of the main dSPACE-customers. A typical dSpace solution is used for a comparison with the CHILS approach in chapter 5.

### 2.3.2.3 Modelica/Dymola

Modelica® [mod] is an object-oriented modelling language for complex physical systems, containing mechanical, electrical, electronic, hydraulic, thermal, control and electric power or process-oriented subcomponents that allows homogeneous modelling. Modelica® is more suitable for mixed dynamic systems with different types of components than MATLAB®/Simulink®.

Dymola® is a commercial modelling and simulation environment for Modelica. Interfaces to MATLAB®/Simulink® exist, so models can be embedded as special block diagrams (S-functions) into Simulink. Dymola® [dym] offers the capability of HIL-simulation in combination with dSPACE, xPC Target™ or RT-LAB. The Modelica® language is translated into C-Code which is compiled for the simulation execution.

Because of the multi-domain single language capabilities Modelica®, the language has a fast growing community. HIL-simulations are implemented for example by the working group of Professor Dr.-Ing. Clemens Gühmann (*Hardware-in-the-Loop simulation of a hybrid electric vehicle using Modelica/Dymola* [WG06a], *Synchronizing a Modelica Real-Time Simulation Model with a Highly Dynamic Engine Test-Bench System* [WG06b]).

### 2.3.2.4 National Instruments

National Instruments offers a variation of HIL solutions based on the LabVIEW simulation environment [Lab09]. LabVIEW offers a graphical programming language to program and to control the I/O interfaces from the simulation to the “hardware-in-the-loop”. LabVIEW can execute models of other simulation environments or it interfaces directly to MATLAB®/Simulink® for example. Various DAQ boards are provided including Field Programmable Gate Array (FPGA) boards.



### 2.3.2.5 Visual Solutions

Visual Solutions VisSim is, like Simulink®, a visual block diagram language for modelling and simulation of complex dynamic systems. Interfaces to MATLAB® are available. Simulink® blocks can be imported directly into VisSim. Identical to MathWorks™ solutions, it is possible to generate C-Code from the block model. The VisSim Embedded Controls Developer provides an integrated solution for rapid control prototyping based on  $\mu$ Cs from Texas Instruments. VisSim Real-TimePRO provides HIL-simulation capabilities. Analogue and digital I/O boards from different manufacturers are supported, as well as serial connections to programmable logic controllers or distributed control systems.

### 2.3.2.6 ETAS

ETAS [ETA] provides pre-configured HIL systems for example for power train control units. The PT-LABCAR [Anh06], one ETAS HIL solution, offers as I/O interfaces 300 signal pins and 50 additional high-current pins. The basic system is designed for testing ECUs that control 8-cylinder gasoline engines or 12-cylinder diesel engines. It can be extended for handling high-end applications, such as tests of ECUs that control 16-cylinder engines too.

## 2.3.3 Non-Commercial Hardware-in-the-Loop Solutions

Non-Commercial HIL solutions are mainly projects of universities. The general setup does not differ from the commercial solutions. A standard PC equipped with DAQ cards hosts the simulation. The DAQ cards establish the connection to the hardware. Also FPGAs are used as configurable I/O devices ([VGB04a],[BMG05]).

### 2.3.3.1 University of South Carolina – Virtual Test Bed

The Virtual Test Bed (VTB) [vtb], developed by the Electrical Engineering department of the University of South Carolina, is a software for prototyping of large-scale, multi-technical dynamic systems. The VTB embeds models from different simulation environments into a unified simulation environment. The VTB models can be created in different ways. For instance, they can be imported from other simulation tools, such as MATLAB®/Simulink®, SPICE or ACSL. The modular VTB solver can be used

with Windows, Linux and Mac OS-X. Different analogue and digital I/O boards from different manufactures are supported to build a HIL setup.

### 2.3.3.2 ETH Zürich – Generic Hardware-in-the-loop Framework

A generic HIL framework is designed by Marco Sanvido ([San02], [SCS02]). This framework adds new approaches in testing embedded systems based on temporal logic and fault generation.

### 2.3.3.3 University of Twente – Borderc project

In the context of the Borderc project (*Beyond the Ordinary: Design of Embedded Real-time Control*) HIL solutions were developed to support the design of embedded systems. The publications of Peter M. Visser, Marcel A. Groothuis and Jan F. Broenink present applications for embedded control systems ([VGB04a],[VGB04b]). 20-sim [20S] is used as the simulation environment. 20-sim is a modelling and simulation programme that can simulate the behaviour of dynamic systems, such as electrical, mechanical and hydraulic systems or any combination of these.

### 2.3.3.4 University of Karlsruhe – COMPASS

At the University of Karlsruhe a configurable modular rapid prototyping system called COMPASS has been developed for automotive systems ([BMG05], [Bie07]). The system uses FPGAs cards as versatile configurable I/O interfaces.

## 2.3.4 Coupling Concepts

For a general HIL simulation, there is only one possibility of designing it. A physical connection of each hardware interface is needed to capture signals and to stimulate the hardware. General HIL simulation does not only covers digital or analogue signals which are transmitted over electrical wires but also forces or heat. Let us think about a Damper-in-the-Loop simulation. The damper has to be stimulated with a real force and it produces an opposing force.

For a  $\mu$ C-in-the-Loop simulation this means that every input and output pin of the  $\mu$ C has to be connected to the simulation computer. Hundreds of connections are needed to do so. The advantage is that no special adaption

on the side of the  $\mu\text{C}$  is needed. The disadvantages are the demands on the coupling hardware and the real time capability of the simulation. The last disadvantage can be overcome by controlling the  $\mu\text{C}$  clock system, if it is possible to do so. The DeskPOD™ from the SimPOD company [Sima] is a hardware solution which uses this concept. Later on the CHILS approach is compared with the DeskPOD™ in chapter 5 in section 5.5.

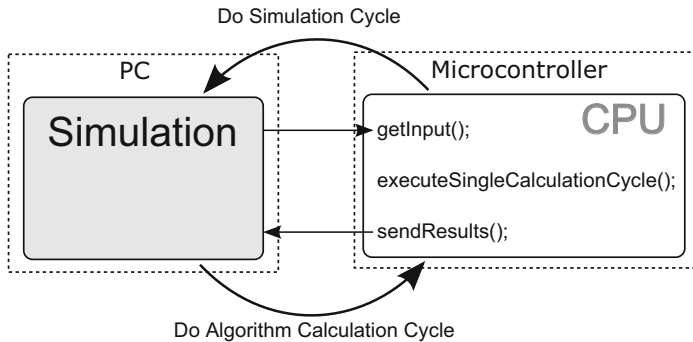


Figure 2.5: Processor-in-the-Loop

With  $\mu\text{Cs}$  as the “in-the-loop” hardware, there is another possibility for the simulation connection. If interfaces of the  $\mu\text{C}$  are stimulated from inside, the connection can be realized on a higher level of abstraction over a single physical connection. The only physical connection between hardware and simulation can for example be the debugger interface. The solution is very attractive because of the a low hardware effort for coupling. PIL solutions, based on evaluation boards, often use this simple connection over general serial interfaces or the debugger interface. But a PIL approach does not use any peripherals<sup>9</sup> of the  $\mu\text{Cs}$ . The data exchange with the simulation is realized by bypass functions at the beginning and the end of a single control-cycle of the algorithm (see figure 2.5). The *Link for TASKING* [Lin09] solution from MathWorks™ mentioned above is a popular example. Further details on PIL solutions are described in chapter 9 in section 9.2. Vase Klandzevski [KM06] also describes the usage of a serialized pin interface and internal stimulation as preceding work of the current approach.

<sup>9</sup> The peripherals of  $\mu\text{Cs}$  are just as complex as the CPU. Beyond standard communication interfaces, like Asynchronous Serial Channel (ASC), Controller Area Network (CAN) and Synchronous Serial Channel (SSC), peripherals include often complex timers for signal acquisition and generation or external bus systems.

## 2.4 Additional Concepts

Beyond the **HIL** simulation other techniques exist. These techniques cover overlapping areas of application within the development process. The three major types of simulation techniques, including **HIL** simulation, can be defined as: **System-Simulation** or **Model-in-the-Loop (MIL)**, **SIL** and **HIL** simulation. In the literature the use of the terms is often not consistent. For the definitions the system is assumed to consist of an object of control, the plant, and the controller.

### 2.4.1 System-Simulation

The **System-Simulation** has the target to map a real system or a system which is to be developed to a simulation. The system is mapped to a representation which consists of equations (continuous simulation) or event sources (event driven simulation). All components of the simulation are executed as simulation models (the object of control and the control) on a developing system. Often **Co-Simulation** techniques are used to simulate complete system. A **Co-Simulation** couples different simulation environments (for example a continuous and an event driven simulation environment) together. The motivation is to use a suitable simulation environment for each part of the system.

The hardware of the development system, typically a PC or Workstation, is usually not identical with the hardware for the real implementation. The term **MIL** is also used for System-Simulation (see figure 2.6).

### 2.4.2 Software-in-the-Loop Simulation

It is called a **SIL** simulation if the model of the (control) software is replaced by the real control algorithm<sup>10</sup>. The control algorithm interacts with the model of the plant. Co-Simulation can be applied for this purpose, too. One way is to couple an ISS, which can execute the control software of the final controller device, with the plant simulation. Prof. Dr. Ing. Abel's script *Rapid Control Prototyping* [Abe03c] defines that the control algorithm running on the developing hardware interacts with the real plant or parts

---

<sup>10</sup> For example, it is possible that the model of the control uses floating point operations while the real software uses fixed point operations. So the SIL simulation is more exact than the MIL simulation.

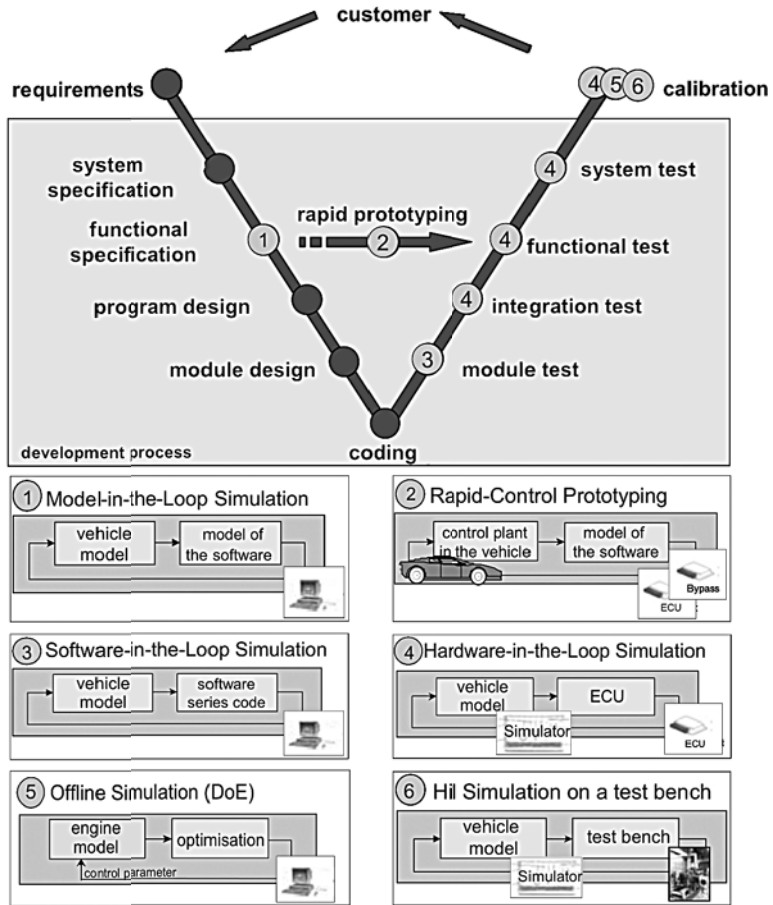


Figure 2.6: Application of Simulation Techniques [Güh05]

of the plant. In fact this is some kind of HIL simulation. The definition 2.3.1 introduced before is used here.

### 2.4.3 Hardware-in-the-Loop Simulation

In the automotive industry the term **HIL** simulation is mostly used for the coupling of an ECU and a vehicle model. As mentioned before the other possibility, the connection of the real vehicle and a simulated ECU, is also

a **HIL** simulation. In figure 2.6 the term **Rapid Control Prototyping (RCP)** is used for that setup. The real vehicle is controlled by a model of the software, which can rapidly be adapted and optimized.

## 2.4.4 Emulation and Emulator

An emulator can be defined as provider of an emulation.

An emulator duplicates (provides an emulation of) the functions of one system using a different system, so that the second system behaves like (and appears to be) the first system. This focus on exact reproduction of external behaviour is in contrast to some other forms of computer simulation, which can concern an abstract model of the system being simulated. ([Wika])

Lovic Gauthier and Ahmed Amine Jerraya mention in [GJ00] the hardware simulation or emulation as ‘classic’ methods to model a  $\mu$ C. Hardware simulations or emulations reproduce at gate level the circuit to be simulated. Often an emulator is fast enough to be plugged into a working target system as replacement of a future chip. Different variations of emulator hardware exist. A **processor based emulation** contains from tens of thousands to hundreds of thousands of Arithmetic Logic Units (ALU) with registers [Tur05]. These processing elements are scheduled to emulate the chip logic and the chip registers. **FPGA based emulators** map the SoC design to the configurable logic blocks of a FPGA. Another form are **Very Large Instruction Word (VLIW) based emulators** which are described for example in Gunter Haug’s thesis *Emulation synthetisierter Verhaltensbeschreibungen mit VLIW-Prozessoren* [Hau01] or in Jürgen Schnerr’s thesis *Zyklengenaue Binärkodeübersetzung für die schnelle Prototypenentwicklung von System-on-a-Chip* [Sch06b]. VLIW processors are a highly parallel architecture containing multiple ALUs. The parallelization is realized on compiler level.

Emulation is an often used technique to accelerate the verification of SoC systems (for example in[YMC00], [SR04], [HKPS05]) because the emulation is several magnitudes faster than a Register Transfer Level (RTL) simulation. In *Integrierte Simulation und Emulation eingebetteter Hardware/Software-Systeme* Stephan Schmitt presents [Sch05] an integrated FPGA based developing environment, which can be used to develop SoC software in a very early stage of SoC design. A direct simulation acceleration by FPGAs is also

possible. R. Siripokarpirom and F. Mayer-Lindenberg [SML04] presents a concept of *Hardware-Assisted Simulation*.

## 2.5 Comparison of the Concepts

Each of the four concepts supports the system development process. In figure 2.6 from [Güh05] each technique is mapped to different steps of the development process. The idea is to start the process with a System-Simulation or MIL simulation and replace the simulated components step-by-step by the real system parts. HIL simulation is an addition to the other techniques, but it can be used instead of a System-Simulation if the HIL components already exist, so that the design process does not start from the scratch for each component.

Emulation can accelerate simulations or replace a simulation completely. The limitation of emulation is caused by the emulator hardware. This can happen for example if a SoC design is too large to be synthesized for a FPGA-emulator<sup>11</sup>.

## 2.6 Subproblems of Hardware-Simulation-Coupling

The subproblems to be solved can be divided into three main areas. First off all, the exchange of information or events between real hardware and simulated hardware has to be realized. The term events is used because it is more common for simulation systems. This concerns the exchange modes, synchronization aspects and the hardware needed for the coupling. It also includes possibilities to connect the real hardware with different simulation environments. The interface abstraction, especially the level of abstraction, is a problem that is independent from the exchange of events itself. The level of abstraction defines what an event is and not how it is published. After all, the event exchange process can be optimized by analyzing the system.

---

<sup>11</sup> The synthesization maps the SoC design, which is normally described in hardware description language, to the resources of the FPGA platform.

### 2.6.1 Connection between Hardware and Simulation

The connection of hardware and simulation has to cover the following aspects:

- the exchange of events between real hardware and simulated hardware
- the support of different simulation environments
- and the hardware effort for coupling.

In subsection 2.3.4, two general types of connections are mentioned. The first way is to design a hardware interface which connects every pin of the  $\mu\text{C}$  to the simulation computer. This solution demands a lot of effort in the coupling hardware. Alternatively the connection can be realized with internal resources of the  $\mu\text{C}$  and uses a higher level of abstraction to exchange the events. The hardware effort is much lower but the  $\mu\text{C}$  has to support this mechanism. Chapter 3 discusses how CHILS uses this concept for the exchange of events between the  $\mu\text{C}$  and the simulation.

### 2.6.2 Interface Abstraction

Abstraction is the most effective way to speed up a simulation. If the events, which are exchanged inside a simulation, can be reduced, the simulation will be faster. This is done by raising the level of abstraction. The same issue, speeding up a simulation, is addressed if the events have to be exchanged between hardware and simulation. The task is to define the right level of abstraction for a certain application. Chapter 4 presents the CHILS concept of interface realization between the  $\mu\text{C}$  and the environment.

### 2.6.3 Event Exchange Optimization

The goal of the HIL simulation system analysis is to find an optimized setup for the data exchange between hardware and simulation. This is primarily a compromise between accuracy and performance. An optimized setup covers the type and amount of exchanged data and the step size between two data exchanges. The first claim can be obtained by choosing the highest level of interface abstraction which is suitable for the desired functionality. The second claim concerns the rate of changes of the exchanged data and the necessity of change distribution. Two techniques can be used, an analysis



of the running system and a pre-analysis of the system, which are presented in chapter 5, 6 and 7.

## 3 Connection between Hardware and Simulation

In the last section, the basics of simulation systems were presented. Now the basics of simulator coupling and coupling system approaches, primarily connections between different simulation systems, are discussed. Based on these considerations the event exchange concept will be chosen. The connection setup was partly presented on the *19th IASTED International Conference on Modeling and Simulation* in the paper *Chip Hardware-in-the-loop Simulation (CHILS) - Embedding Microcontroller Hardware in Simulation* [KMH08a].

### 3.1 Basics

The connection between hardware and simulation is based on principles used in simulator coupling for Co-Simulations. The principles cover the protocol part of the coupling and not the physical channel. Basic principles for the physical interface are explained in chapter 2 in subsection 2.3.4.

Continuous and discrete simulations differ in interpretation of time, communication and activation of simulation modules. Table 3.1 from [BNAA05] presents this relation. Coupling of discrete simulators is a straightforward task. The notifications of events have to be passed on from one simulator to another. In order to couple continuous simulators, the piecewise continuous signals have to be transmitted. The coupling of discrete and continuous simulators is even more complex. The different interpretations of time and communication have to be translated into each other.

#### 3.1.1 Simulator Coupling and Co-Simulation

Co-Simulation becomes necessary if a **heterogeneous modelling** approach is used to model the different domains of a system (see chapter 2 subsection 2.2.2). Stefan Eilers [Eil06] distinguishes between two main approaches. If

<b>Concepts / Model Type</b>	<i>Time</i>	<i>Communication Means</i>	<i>Process Activation Rules</i>
<i>Discrete</i>	Global notion for all modules of the system. It advances discretely when passing by time stamps of events.	Set of events (value and time stamp) located discretely on the time line.	Processes are sensitive for events.
<i>Continuous</i>	Global variable involved in data computation. It advances in integration steps.	Piecewise-continuous signals.	Processes are executed at each integration step.

Table 3.1: Basic Concepts [BNAA05]

the different modelling languages are translated into a common language, the simulation can run on one simulation engine. This is called **Single-Engine Co-Simulation**. The approach is fast and efficient as long as the transformation into the common language is efficient. So far no standard for a common language exists. If no common language exists, each module, written in another language, has to run on a suitable simulation engine. The simulation engines or simulators are coupled on a higher level. This setup is called **Multi-Engine Co-Simulation**. Different implementations of multi-engine coupling are known. In general, a simulation manager is implemented to control the different engines. This manager initializes the engines and synchronizes them. In a tightly coupled concept, the coupling is realized by a **simulation-backplane** or **simulation-bus**<sup>1</sup>. Other concepts use the message exchange mechanisms on operating system level or shared memories (see [Bra06] and [Lan06]). Christian Scholz [Sch04] uses the message passing interface *Parallel Virtual Machine* [PVM] to couple simulation engines which run on different computers in a network.

Many simulation systems allow to run user defined modules written in C or C++. So the coupling can be realized by writing interface modules.

<sup>1</sup> The terms Co-Simulation bus and a Co-Simulation backplane can be found in literature too.

A simulation engine cannot be completely controlled in every case by an external application. This can be a disadvantage for example if only fixed simulation time steps are possible<sup>2</sup>.

**Master-Slave Co-Simulation** is a term used for a Co-Simulation setup where one simulation engine controls the other one. The simulation manager is in fact the master simulation. The master controls the step size of the whole simulation and/or manipulates the event schedulers of the other simulators (event-driven simulation).

### 3.1.2 Synchronization of Co-Simulations

The synchronization concept is a key problem of simulation-to-simulation coupling as well as for hardware-to-simulation coupling. But simulation-to-simulation coupling has one advantage: it is more flexible because of the pure software nature of the setup. In *Hybride, tolerante Synchronisation für die verteilte und parallele Simulation gekoppelter Rechnernetze* [Rüm97] Markus Rümekasten defines the synchronization as follows. The synchronization shall prevent the events from one simulation to arrive too late in another simulation. It has to guarantee the maintenance of the causal order. The basics are mentioned in chapter 2 in subsection 2.2.3 *Simulation Systems*. One can distinguish between **conservative** and **optimistic principles**.

**Conservative principles** forbid the violations of the causal order. Each simulation has to know if another simulation plans to process events that concerns it. First, a simulation determine the time  $t$  of the next concerning external event. Second, this simulation continues until the point of time  $t$  is reached. Time  $t$  is the synchronization point in time. Third, the simulation waits for the arrival of the planned external event. A simulation manager can manage this synchronization by distributing the external events from one simulation to another. This technique can only be applied for event-driven simulations. The problems are deadlocks and decreased simulation performance caused by empty synchronization events<sup>3</sup>. The **lockstep principle** can also be seen as a conservative principle. The simulators are forced to execute synchronous time steps. The size of the time step is determined by the next internal or external event to be processed or by

---

<sup>2</sup> The solver of continuous simulation environments, like MATLAB®/Simulink®, are even more efficient if the step size is variable.

<sup>3</sup> Synchronization events are events which are used to synchronize different simulators. The synchronization events are set to points in time where an external event is supposed to occur. The synchronization event is called empty if no external events occur.

the smallest planned time step by any simulator within the Co-Simulation, if continuous simulators are included in the coupling. This is needed to prevent violations of the causal order, but the fact that all simulators are forced to compute each time step leads to many unnecessary computations. The advantage of the **lockstep principle** is that it can be applied in most cases.

**Optimistic principles** accept violations of the causal order. The violations are fixed at occurrence by rolling back the simulation to a previously saved state before the violation causing event occurred. The simulation runs again up to the point where the detected event occurs. A rollback or backtracking has high demands on the simulation engines. They have to support the state saving and the rollback. In addition, the concept requires a lot of memory to store the states.

**Hybrid principles** combine both approaches. An optimistic execution can be optimized by distribution of lookaheads to reduce rollbacks. It is also possible to run sub-models in an optimistic way, while realizing the communication between them in a conservative way.

## 3.2 Related Work

In *Schnittstellenmaske für die Kopplung unterschiedlicher Simulatoren über polymorphe Signale* [Lan06] and *Synchronisation von Simulatoren unter Berücksichtigung des Konzepts polymorpher Signale* [Bra06] so-called **polymorphic signals** are used to synchronize different simulators and exchange events between them. The attempt is to present a generalized synchronization concept. The idea of **polymorphic signals** was presented in [SGW04]. They are designed to realize a data exchange between different kinds of models. A conversion of types, value ranges and clock rates is done implicitly. The signals adjust to the model. A typical example is a Pulse-Width Modulation (PWM) signal. A high-level model just evaluates the PWM rate while a low-level model needs a series of pulses presenting the signal.

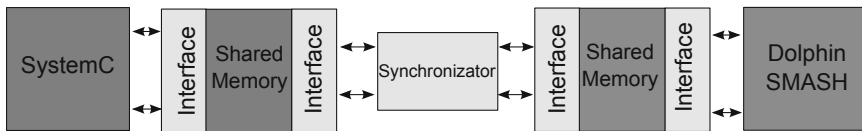


Figure 3.1: Simulator Coupling Adapted from [Bra06]

Clemens Braband [Bra06] defines four quality levels of simulator coupling. **Level 0**, the lowest quality level, synchronizes the simulators only when a read or write operation to the shared memory area occurs. The simulators are stopped at read and write accesses until all of them reach the same point in time. This simple technique allows the coupling of nearly all kinds of simulators. **Level 3**, the highest quality level, is equivalent to an event-discrete coupling. The simulators have to exchange all information of planned events, so no unnecessary time-steps or delta steps<sup>4</sup> are taken. Continuous simulators cannot be coupled at that level, because the changes of the signals are often not predictable. Mechanisms like rollback are not applied in this work. **Level 1** and **level 2** are approximations of **level 3**. Both levels requires simulation environments with adaptable time steps but without prediction of future events. **Level 1** coupling emulates delta steps by taking a very small time step, while **level 2** simulators are able to execute delta steps.

The research group of Professor Gabriela Nicolescu from the *Ecole Polytechnique de Montréal* works on discrete–continuous Co-Simulation environments. The focus objects are generic architectures for discrete-continuous simulation models, for example for a global validation in component-based heterogeneous systems design ([BNAA05] and [BBN<sup>+</sup>06]), and a formal definition of the internal architecture of simulation interfaces [GBNB06]. The examples are based on SystemC and MATLAB®/Simulink®. The group developed a tool for the automatic generation of simulation models. The inputs of the tool are the continuous models modelled in Simulink® and the discrete models written in SystemC. The output of the tool is a global simulation model, based on the simulation-bus approach. The interfaces to the simulation-bus are generated by the tool. The synchronization model used by the Co-Simulation is based on event detections [BNAA05]. The requirements are formulated as follows:

“The continuous simulator must detect the next discrete event (timed event) scheduled by the discrete simulator, once the latter has completed the processing corresponding to the current time...

The discrete simulator must detect the state events. A state event is an unpredictable event, generated by the continuous simulator, whose time stamp depends on the values of the state

---

<sup>4</sup> Delta steps are the state changes of discrete simulators. They consume no simulated time, but of course simulation run time.

variables (ex: a zero-crossing event, a threshold overtaking event, etc.).”

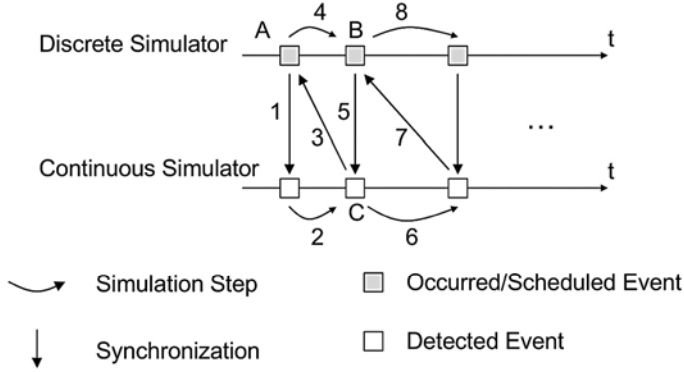


Figure 3.2: Synchronization Model from [BNAA05]

The synchronization model, shown in figure 3.2, is designed to couple two simulation engines. During the simulation, the context switches between both engines<sup>5</sup>. An implementation of rollback mechanisms is planned (see *Semantics for Rollback-Based Continuous/Discrete Simulation* [GNB08]). As described in the previous section, the **simulation-bus** is a common concept.

A special field of research are Hardware-Software Co-Simulation tools. Often a Hardware Description Language (HDL)-simulator, with models of the bus systems and peripherals, is coupled with a model of the software or a fast processor model, which execute the software. Bus-based architectures allow an easy partitioning of the whole system architecture by separating the model at the interfaces to the bus. In *Hardware-Software Co-simulation of Bus-based Reconfigurable Systems* [KV05] a cycle accurate co-simulator, as a typical setup, is presented. A C-model of a processor executes the system software while HDL models describe the bus and reconfigurable units. Because of the discrete nature of the setup and the existing interface, a synchronization of the simulation parts is easy to realize. Every access to the simulated bus can be seen as a single event (high-level transaction) or a sequence of events (low-level events). Commercial tools like Seamless [Sea] from Mentor Graphics [Men] are available for such systems, too.

<sup>5</sup> So the simulation engines are not executed in parallel.

In *Accelerated Logic Simulation by Using Prototype Boards* [HSBG98] Jürgen Haufe, Peter Schwarz, Thomas Berndt and Jens Große present an approach to speed-up simulations. A FPGA based prototype board is used for the hardware emulation. The synchronization concept is straightforward. The simulator sets the vector to the input of the emulator board. Afterwards the simulator waits for one or more hardware clock cycles, which are defined by the user, to get the output vector output as return. The simulator advances the simulation in the next step. Diverse other approaches use a similar approach for synchronization, for example the PIL setups presented in [JLD<sup>+</sup>04] and [FBP<sup>+</sup>07].

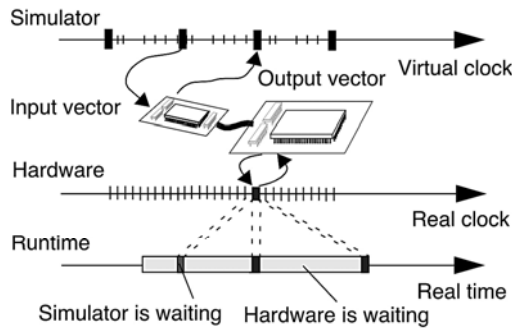


Figure 3.3: Simulation-Emulator Synchronization from [HSBG98]

### 3.3 Exchange of Events between Microcontroller and Simulation

These considerations lead to two basic settings of simulators to couple with. One setting is the coupling with a DESS-simulation, or continuous simulation, while the other one is the coupling with a DEVS-simulation. For each basic type another strategy can be applied.

Independent from the type of target simulation, the term “event exchange” is used for the data exchange between  $\mu C$  and simulation. The exchange mechanism operates with discrete steps, caused by the discrete nature of the  $\mu C$ , therefore the changes of a quasi-continuous signal can be interpreted as events. The following considerations cover the different tech-



niques for simulator coupling. The application of conservative principles, optimistic principles, and hybrid principles for coupling are discussed.

### 3.3.1 Continuous Simulation

The connection between a  $\mu C$  and a continuous simulation is a coupling of a discrete and a continuous system. This is comparable to the Co-Simulation setup which is used by Professor Gabriela Nicolescu's research group (see for example ([BNAA05] and [BBN<sup>+</sup>06])). In contrast to a discrete simulator like SystemC, additional restrictions exist. A prediction of future events, caused by the  $\mu C$  software, is not possible<sup>6</sup>, but of course the detection of events is possible. So it is not feasible to run the continuous simulation exactly to the point in time where an event occurs which is produced by the  $\mu C$ . A minor propagation delay of the event has to be accepted.

#### 3.3.1.1 Basic Coupling Principles

Two basic exchange principles exist. The first option is to run the simulation and  $\mu C$  alternately (see figure 3.4). The second one is to run both systems in parallel (see figure 3.5). It is obvious that the parallel execution has a higher performance at the same step size, but the alternating version is potentially more exact. Both versions are possible solutions from the simulation point of view. The next step size of the continuous simulation is known and in addition some simulators support rollbacks. The prediction of future events of the  $\mu C$  is not possible as mentioned. The events can be detected at occurrence. It is also possible to stop the execution of the  $\mu C$  if an event has been detected.

**Optimistic principles** cannot be applied. Not all simulation environments support rollbacks and the effort to implement rollbacks for a  $\mu C$  is too high<sup>7</sup>. **Conservative principles** need to be applied. The **lockstep principle** is the best option to be used. Both sides, the simulation and the  $\mu C$ , have to be forced to execute synchronous time steps.

Table 3.2 presents the exchange principles in dependence of the available resources. An alternating execution of simulator and  $\mu C$  makes sense if the

<sup>6</sup> A general prediction of the results, which a programme produces, is not possible. The programme has to be executed for detecting the read and write operations. Write operations to the peripherals can be seen as sources for events.

<sup>7</sup> The whole state of the  $\mu C$  has to be saved. That means to store and restore the values of all internal registers, memories and state machine positions.

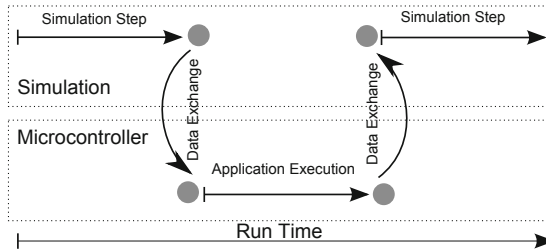


Figure 3.4: Alternating Execution

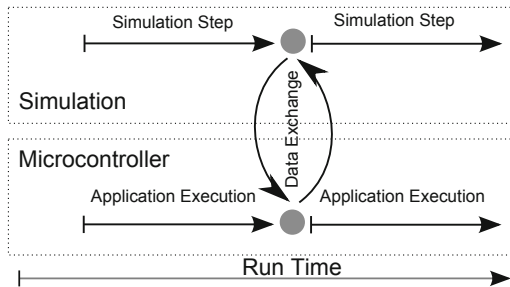


Figure 3.5: Parallel Execution

simulator uses rollbacks and/or the  $\mu\text{C}$  halts on events. A parallel execution is not possible because both sides may not run to the same point in time. The challenge is that one side has to follow the other side, so either the  $\mu\text{C}$  or the continuous simulation determines the next step. In figure 3.4 the continuous simulation leads the execution while the  $\mu\text{C}$  follows<sup>8</sup>. If the MC does not halt on events, they might be delayed or lost. This is true for a parallel run, too. Otherwise if the  $\mu\text{C}$  determines the next step, the step size could be too large or too small for the numerical solver of the simulation. A useful minimal and maximal step size has to be chosen in advance. Figure 3.6 and figure 3.7 present both possible scenarios. In figure 3.6 the  $\mu\text{C}$  leads the execution. The process starts with a data exchange of both systems (step 1). Information regarding the minimal and maximal step size can be exchanged. In step 2, the  $\mu\text{C}$  starts with the planned time step size marked as  $2^*$ . Before finishing the complete time step, the  $\mu\text{C}$  detects events and

<sup>8</sup> The  $\mu\text{C}$  must not halt on events, otherwise the time base of  $\mu\text{C}$  and simulation will differ.

<b>Simulation / Microcontroller</b>	<i>Next Step Size Known + No Roll Back</i>	<i>Next Step Size Un- known + Roll Back</i>
<i>Can Detect Events</i>	run in parallel	run alternatingly
<i>Can Detect Events + halt device</i>	run alternatingly	run alternatingly

Table 3.2: Exchange Principles

halts. The information about the already executed time step is sent to the simulation (step 3). The simulation executes a time step of the same size (step 4) and both systems are at the same point in simulation time. In step 5, the data, events and event occurrence information, are exchanged between the systems while in step 6 the  $\mu\text{C}$  executes the next time step.

Figure 3.7 presents the situation if the simulation is the leading element. In step 1, the first data exchange is executed. The simulation executes the integration step in step 2. Afterwards a rollback condition is detected and the simulation is set to the previous point in time. Step 4 is the adapted time step which is taken by the simulation. In step 5, the information about the executed time step is transferred to the  $\mu\text{C}$  which executes the time step in step 6. An event is detected by the  $\mu\text{C}$  during the execution. This event is delayed until the data exchange in step 7 is finished.

If no rollback is used by the simulator and the  $\mu\text{C}$  does not halt on events (only detects events), a parallel run makes sense. The simulation determines the size of the next step <sup>9</sup>. The information of detected  $\mu\text{C}$  events can influence this choice by reducing or extending the possible step size. The execution time of the  $\mu\text{C}$  application is adapted to the chosen step size. Figure 3.8 shows this kind of synchronization and data exchange. In step 1, the first data exchange is executed. Then in step 2, simulation and  $\mu\text{C}$  execute the time step in parallel. An event is detected by the  $\mu\text{C}$  during execution. The event is delayed until the next data exchange in step 3 is finished.

### 3.3.2 Event Discrete Simulation

The connection between a  $\mu\text{C}$  and a discrete simulation is the coupling of two discrete systems. The best case for coupling discrete system is an

<sup>9</sup> The numerical solvers of continuous simulation environments have an automatic step size adaption. Otherwise a constant step size can be chosen by the modeller.

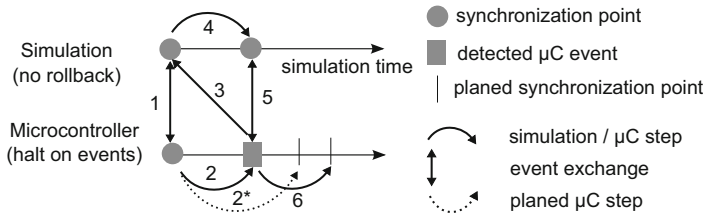
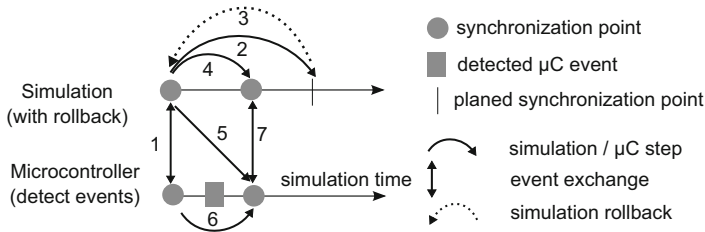
Figure 3.6: Alternating Execution –  $\mu$ C Leads

Figure 3.7: Alternating Execution – Simulation Leads

event-discrete coupling where the simulators exchange all information of planned events. As mentioned before, a prediction of future events on  $\mu$ C side is not possible but the detection of events is of course possible. So, only a lower coupling quality level, described in [Lan06] and [Bra06], can be achieved.

### 3.3.2.1 Basic Coupling Principles

The two basic exchange principles of the hardware to simulation coupling are identical with the continuous simulation coupling. An alternating run

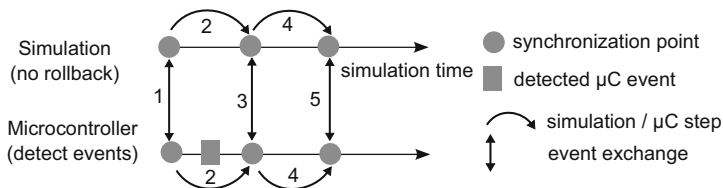


Figure 3.8: Parallel Execution – Simulation Leads

of the simulation and the  $\mu\text{C}$  is possible (see figure 3.4). The advantages and disadvantages are the same as in continuous simulation coupling. One side has to follow the other side, so either the  $\mu\text{C}$  or the discrete simulation determines the next step. Both variations can lead to lost or delayed events, so a useful minimal and maximal step size has to be chosen in advance.

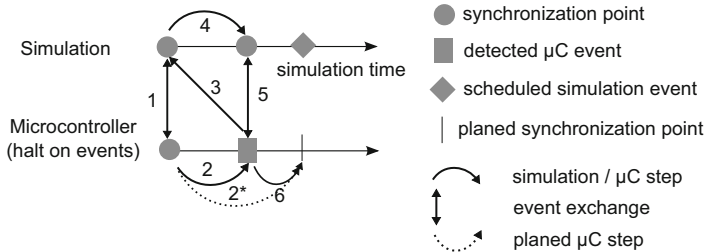


Figure 3.9: Alternating Execution –  $\mu\text{C}$  Leads

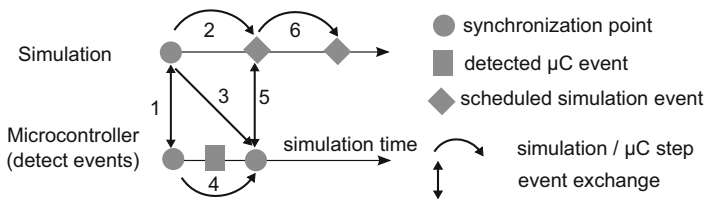


Figure 3.10: Alternating Execution – Simulation Leads

In figure 3.9, the  $\mu\text{C}$  leads the execution. Step 1 shows the initial data exchange between both systems. In step 2, the  $\mu\text{C}$  starts with the planned time step size marked as  $2^*$ . This step size is chosen by both systems. The discrete systems delivers the information about a future event for the  $\mu\text{C}$ . Then the  $\mu\text{C}$  detects an event and halts. The information about the already executed time step is sent to the simulation (step 3). The simulation executes a time step of the same size (step 4) and both systems are at the same point in simulation time. In step 5, the data, events and event occurrence information, are exchanged between the systems while the  $\mu\text{C}$  executes the next time step in step 6.

The challenge is to transmit the correct information for the next scheduled

event of the simulation. It would be very inefficient if internal events<sup>10</sup> influence the planned time step of the  $\mu C$ , too. On the other hand, an internal event can cause an external event<sup>11</sup>. This external event is lost or delayed in the worst case (see figure 3.11).

Figure 3.10 presents the situation if the simulation is the leading element. In step 1, the first data exchange is executed. The simulation steps forward to the next scheduled event in step 2. In step 3, the information about the executed time step is transferred to the  $\mu C$  which executes the time step in step 4. The event, detected by the  $\mu C$ , is delayed until the data exchange in step 5 is finished.

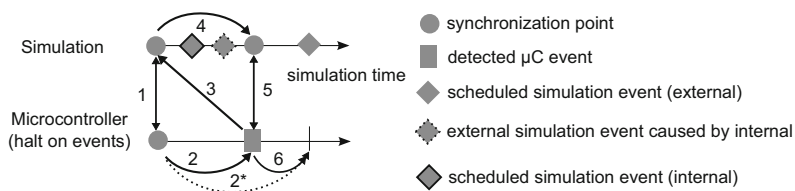


Figure 3.11: Alternating Execution –  $\mu C$  Leads

The second possibility is to run both systems in parallel (see figure 3.5). The event discrete simulation determines the step size depending on the planned events. The information of detected  $\mu C$  events can influence this choice. This can be done by feeding the event queue of the simulator with additional events. A constant step size for the synchronization of both systems can be implemented by creating such synchronization-events. Figure 3.12 shows this kind of synchronization and data exchange. In step 1, the first data exchange is executed. Then in step 2, simulation and  $\mu C$  execute the time step in parallel. During the execution, an event is detected by the  $\mu C$ . The event is delayed until the next data exchange in step 3 is finished.

### 3.3.3 Conclusions

Both variations of coupling can lead to lost or delayed events. It is advisable to choose the version which has the potentially better performance. So

<sup>10</sup> Internal events are not leaving the system boundaries of the simulation, so they have no direct influence on the coupled systems.

<sup>11</sup> This happens if an external events crosses the system boundaries.

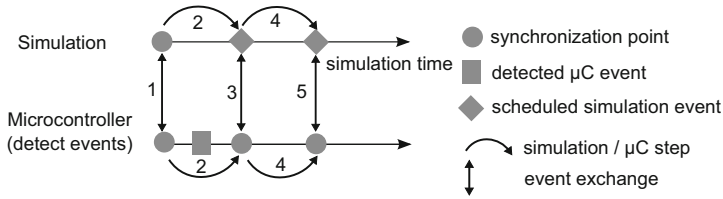


Figure 3.12: Parallel Execution – Simulation Leads

running the simulation and the  $\mu\text{C}$  in parallel is the most practical and high-performance way. The alternating run only has minor advantages. In addition, the modeller has to choose a useful minimal and maximal step size in advance, to prevent or reduce the effects of delayed or lost events. An efficient version to predict events on the  $\mu\text{C}$  is not possible, but information about detected events can be taken into account to adapt the step sizes.

It makes sense that the simulation determines the step size because continuous and discrete-event simulators have mechanisms to do so. The  $\mu\text{C}$  event detection yields additional information to adapt the time step. If the  $\mu\text{C}$  halts on events, other problems can occur. If writing operations, as sources of events, are located directly in a row, the stop-and-go can be very inefficient. In addition, the accuracy of execution is reduced enormously because normally a  $\mu\text{C}$  cannot be halted from one cycle to another. This effect is discussed later on in the section 3.5 *Effects Caused by Coupling*.

### 3.3.3.1 Levels of Simulator Coupling

Two levels of simulator coupling are defined for the current implementation, the **fixed-step** and the **variable-step** level. In both levels, the simulation and the  $\mu\text{C}$  run in parallel.

**fixed-step** The step size is chosen in advance for the  $\mu\text{C}$  and the simulation

**variable-step** The step size is chosen dynamically by the simulation. Additional event occurrence information from the  $\mu\text{C}$  can influence the step size. A minimal and maximal step size can be chosen in advance.

## 3.4 CHILS Event Exchange Mechanism

The event exchange mechanism of CHILS is based on the information presented in the last section. A parallel execution of simulation and  $\mu\text{C}$

application is chosen. The time step is fixed or variable, but it is determined before the step is performed. The modeller chooses the minimum and maximum step size before the run.

The data exchange, the exchange of events, and additional information is managed by a monitor application which is called the CHILS monitor. The CHILS monitor has to run independently from the user application on the  $\mu\text{C}$ . On the PC side, a corresponding part exists which handles the communication between the CHILS monitor and the simulation. This software is called the CHILS device. Based on the CHILS device, adaption layers for different simulation environments can be written. Figure 3.13 shows an exemplary data exchange between the  $\mu\text{C}$  and the simulation. Simulation and  $\mu\text{C}$  execute time steps of the same length, as can be seen in figure 3.14. The runtime of simulation and  $\mu\text{C}$  can differ, so the simulation does not need to be realtime-capable. On simulation side, the execution context switches between simulation and CHILS device, while on  $\mu\text{C}$  side it switches between CHILS monitor and user application.

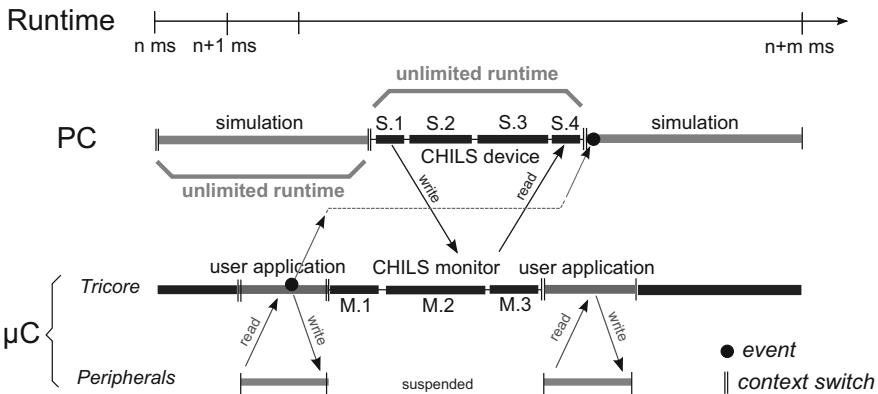


Figure 3.13: CHILS Event Exchange Setup

The phases of data exchange are defined as follows.

- M.1** The CHILS monitor captures the application output (the events).
- M.2** Afterwards the CHILS monitor waits for data exchange with the CHILS device.
- M.3** The CHILS monitor sets the new MC inputs for the application after the data exchange with the CHILS device is finished.



- S.1 The CHILS device gets the simulation output.
- S.2 This output is transferred as soon as the CHILS monitor is ready to receive.
- S.3 Right after transmitting the data to the CHILS monitor, the CHILS monitor device gets the output data from the application.
- S.4 Then the CHILS device sets the new simulation input.

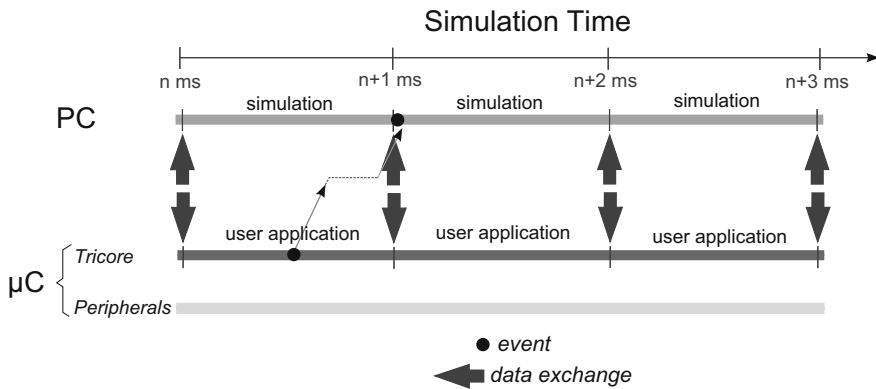


Figure 3.14: CHILS Event Exchange Setup

### CHILS Monitor Application Demands

1. The step size of the  $\mu$ C application step has to be configurable.
2. Mechanisms like event detection, event counting, and event delay measurement have to be available.
3. The CHILS monitor has to have the ability to set new inputs to the  $\mu$ C interfaces and peripherals.
4. The event sources have to be selectable to reduce the monitoring overhead and the amount of exchanged data to be exchanged.
5. CHILS monitor should not reduce the resources which are available for the user application<sup>12</sup>.

<sup>12</sup> The CHILS monitor has to be executed from a memory location which is not used by the user application. It should not reserve normal peripherals like timers.

### Adaption Layer and CHILS Device Demands

1. Adaption layer and CHILS device have to provide the  $\mu C$  interfaces for the simulation environment.
2. The interfaces of the  $\mu C$  have to be individually selectable.
3. A data conditioning of received and sent data, for example the conversion between different abstraction levels or data representation levels, has to be available.

## 3.5 Effects Caused by Coupling

The coupling system between the  $\mu C$  and the simulation influences the overall system. The main cause discussed above is the switching between the CHILS monitor and the CHILS device. The CHILS monitor suspends the peripheral units, but this can not be done immediately. A mechanism, called *Delayed Suspend* (see 3.5.2.2), takes care that all bus transfers to a peripheral unit are finished before the unit is suspended. Otherwise bus transfers could be lost. Other effects are caused by the cache system and by the pipeline architecture of the TriCore®. These factors cause a difference in the runtime of the disturbed and undisturbed application. In addition, the internal time generated by the  $\mu C$  timers, and the external time, which is the time of the simulation, differ. The following contemplations are based on the CHILS monitor implementation for the TC1796ED  $\mu C$ . The complete framework and some implementation issues are explained later in chapter 8 *CHILS Framework - Concept*.

### 3.5.1 Measurement of the Timing Difference

The time difference is measured by test applications executed by the  $\mu C$ . Different test scenarios are applied with focus on the System Timer (STM)<sup>13</sup> and the GPTA<sup>14</sup>. Based on the measurements, the **execution time accuracy** is calculated<sup>15</sup>. The **execution time accuracy** is based on the time the application needs to fulfill a certain task, so it is based on the runtime of a

<sup>13</sup> The STM is the standard timer of the TriCore® based  $\mu C$  series.

<sup>14</sup> The GPTA is a peripheral unit of TriCore® based  $\mu C$  series for signal measurement and signal generation. It provides a set of timer, compare and capture functionalities, which can be flexibly combined.

<sup>15</sup> Further examination can be done by executing the tests on the RTL model of the  $\mu C$ .

piece of code. The **execution time accuracy** results are presented in chapter 9 in section 9.4 *Performance and Accuracy*.

### 3.5.1.1 System Timer (STM)

The STM test application measures the runtime of a certain piece of application code. The basis is the undisturbed system. The measured time difference includes the effect of the *Delayed Suspend*, the cache, and the pipelining. The measurements in table 3.3 are taken at a CPU frequency of 150MHz, a system frequency of 75MHz, and a STM to system frequency rate of one. The CHILS monitor implementation on the TC1796ED  $\mu$ C is used as reference. The measurements show a difference of 22 to 38 CPU clock cycles per exchange. This difference depends on the CHILS monitor location<sup>16</sup> and on the application. The application code of the programme *NOPS* contains only a loop and nop instructions, while *Sieve* calculates prime numbers based on the algorithm *Sieve of Eratosthenes*<sup>17</sup>.

### 3.5.1.2 GPTA

The GPTA test application generates a PWM signal which is measured in the simulation environment. The measurements in table 3.4 are taken at a CPU frequency of 150MHz, a system frequency of 75MHz, and with the CHILS monitor implementation on the TC1796ED. The measurements show a difference of 37 to 52 CPU clock cycles per exchange, so the influence on the GPTA is larger than on the STM. The test application generates a PWM signal of 3750Hz and 1000Hz with a PWM rate of 20%. The signal is measured by a MATLAB®/Simulink® model. The CHILS monitor is executed from a non-cached or from a cached memory region. The results of a similar programme which uses the STM and generates a PWM manual can be found in appendix B in table B.1. As expected, the time difference of the PWM signal is between 24 and 31 CPU cycles per exchange.

The high frequency PWM requires a sampling which should be at least twice of the signal frequency. In addition, a higher oversampling rate is required to capture not only the base frequency but also the PWM rate. So a sample rate is chosen which is approximately ten to hundred times higher

<sup>16</sup> The CHILS monitor is executed either from a non-cached or from cached memory.

<sup>17</sup> The *Sieve of Eratosthenes* is an algorithm for finding all prime numbers up to a specified positive integer. The algorithm is based on striking out numbers from an integer list which are multiples of a prime number.

Application	Step Size (CPU Cycles)	Av. Run-time without Monitor (STM Cycles)	Av. Run-time (STM Cycles)	Monitor Activities	CPU cycles per exchange
<i>Monitor - Non Cached</i>					
NOPS - Flash	20000	35000117	35063562	3498	36.28
	10000	35000117	35130507	6991	37.30
	5000	35000117	35265506	13966	38.01
NOPS - SRAM	20000	35000080	35062326	3497	35.60
	10000	35000080	35123314	6990	35.26
	5000	35000080	35245757	13959	35.20
Sieve - Flash	20000	81812440	81943113	8172	31.98
	10000	81812440	82049670	16324	29.07
	5000	81812440	82368121	32600	34.09
Sieve - SRAM	20000	33406093	33465151	3338	35.39
	10000	33406093	33524101	6671	35.38
	5000	33406093	33641483	13323	35.34
<i>Monitor - Cached</i>					
NOPS - Flash	20000	35000117	35048392	3498	27.60
	10000	35000117	35095634	6989	27.33
	5000	35000117	35189947	13958	27.20
NOPS - SRAM	20000	35000080	35048338	3497	27.60
	10000	35000080	35095609	6990	27.33
	5000	35000080	35189921	13959	27.20
Sieve - Flash	20000	81812440	81906445	8172	23.01
	10000	81812440	81989499	16325	21.69
	5000	81812440	82221708	32600	25.11
Sieve - SRAM	20000	33406093	33451799	3338	27.38
	10000	33406093	33497451	6671	27.39
	5000	33406093	33588188	13323	27.34

Table 3.3: STM Time Difference Measurements

than the base frequency. The chosen step sizes of a hundred CPU cycles are at the limit of the CHILS approach, as the values from table 3.4 show.

Here we see a difference of more than 15% in signal period length in one scenario.

Monitor Step Size (CPU Cycles)	Sampling Rate	PWM Frequency (Hz)	Measured PWM Frequency (Hz)	Period Length Difference	CPU Cycles Difference per Exchange
<i>Monitor - Cached</i>					
4040	9.80	3750.00	3790.84	0.92%	37.12
440	91.01	3750.00	4131.73	9.25%	40.65
300	133.33	3750.00	4335.43	13.50%	40.51
15050	10.00	1000.00	1002.93	0.29%	43.65
1500	100.00	1000.00	1030.73	2.98%	44.71
<i>Monitor - NonCached</i>					
4040	9.90	3750.00	3797.44	1.08%	43.64
440	90.70	3750.00	4196.54	10.64%	46.93
300	133.33	3750.00	4435.41	15.45%	46.36
15000	10.00	1000.00	1003.89	0.35%	52.50
1500	100.00	1000.00	1035.32	3.41%	51.13

Table 3.4: GPTA Time Difference Measurements

### 3.5.2 Causes for the Timing Difference

The main causes for the differences are mentioned above: the *Delayed Suspend*, effects of cache system and pipeline and implementation issues of the CHILS monitor. The measured time difference accumulates all effects. It is not trivial to separate the time difference by the sources. The first step is to take a look at the CHILS monitor implementation. After releasing the peripheral suspend, additional actions have to be taken, so additional instructions are executed.

#### 3.5.2.1 Additional Instructions

The additional instructions executed after the suspend has been released (see listing 3.1) are the main cause of the time difference. A direct mea-

Sequence	CPU Cycles
<i>Monitor - Cached</i>	
Without <i>ret</i>	14
With <i>ret</i>	18
<i>Monitor - NonCached</i>	
Without <i>ret</i>	14
With <i>ret</i>	28

Table 3.5: Runtime of Additional CHILS Monitor Instructions

surement of the runtime of the instruction sequence with the *return from monitor* (*rfm*) statement is not possible due to the nature of the breakpoint trap mechanism<sup>18</sup>. Instead, the sequence was extended by the normal function return statement *ret*. The sequence without *ret* takes 14 CPU cycles at 150MHz CPU frequency, 75MHz system frequency, and STM timers at 75 MHz (see table 3.5). If a normal *return* (*ret*) statement is used instead of the *rfm* statement, the sequence takes 18 to 28 CPU cycles (code location at a cached or a non-cached area). In difference to the *rfm* statement, the *ret* statement restores 16 registers of the TriCore® (see *TriCore® 1 32-bit Unified Processor Core v1.3 Architecture Manual* [Tri02a]), the so-called upper context. These measurements are also influenced by pipeline effects, caching or bus accesses, so they are just an orientation.

Further considerations can be made if a look at the whole sequence is taken. The *rfm* consists of the following five operations, while six operations are needed by the monitor after the suspend is released (see listing 3.1).

- branch to a11
- restore PCXI from [DCX]
- restore PSW from [DCX + 4]
- restore a10 from [DCX + 8]
- restore a11 from [DCX + 12]

<sup>18</sup> The breakpoint trap mechanism is a specialized trap mechanism to switch from a user application to a debug monitor application. The debug monitor on the  $\mu\text{C}$  allows the PC debugger application to observe and manipulate  $\mu\text{C}$  resources. The *return from monitor* (*rfm*) statement is a dedicated instruction to switch back from the debug monitor application to the user application.

The minimal execution length of the sequence can be estimated by accumulation of the single instruction runtime. The instruction *mov.d*, the macro *LDA* and the *enable* instruction have an accumulated runtime of four CPU cycles. The runtime of the *dsync* and *isync* instruction varies <sup>19</sup>. The difference between two monitor implementations with and without *dsync* and *isync* is twelve CPU cycles (see appendix B section B.1) for the non-cached monitor version and three CPU cycles for the cached version. Upon the state of store buffers the *rfm* instruction needs four to five cycles if it is executed from an cached external memory (cache hit assumed). A normal access to the memory location, which is used by the monitor, takes ten to twelve CPU cycles. So the approximated runtime for the sequence is now 30-31 CPU cycles (non-cached) and eleven CPU cycles (cached with cache hit) or 21 CPU cycles (cached with cache miss).

These contemplations support the measured values in table 3.5. The approximated values are now subtracted from the runtime measurement values. Table 3.6 presents the results of this calculation in addition to the execution time accuracy based on definition 7.2.1.

### 3.5.2.2 Delayed Suspend

The so-called **Delayed Suspend** is a special feature of current TriCore®- $\mu$ Cs. The peripheral is not directly suspended if the suspend signal is applied. The suspend is delayed until all bus-transfers to the peripheral units are finished. The delay can be zero or a couple of CPU cycles and it cannot be predicted, because it depends on the application and its current state.

### 3.5.2.3 Pipelining

The delay caused by the pipeline stalls is hard to predict and to measure, especially if two competitive applications like the CHILS monitor and the user application run on the same CPU. But the maximal length of a stall is determined by the length of the pipeline. The TriCore® 1.3 pipeline has four stages, so the maximal delay is four cycles. It can be assumed that the test programme *NOPS* shows the largest influence on pipeline stalls caused

---

<sup>19</sup> The TriCore® architecture manual describes the operations as follows [Tri02a]: “*dsync* forces all data accesses to complete before any data accesses associated with an instruction, semantically after the *dsync*, is initiated... The *isync* instruction forces completion of all previous instructions, then flushes the CPU pipelines and invalidates any cached pipeline state before proceeding to the next instruction.”

```

1      // release suspend
2      mfcr    \%d15, $DBGSR
3      and     \%d15, 0xEF
4      mtcrr   $DBGSR, \%d15
5      // suspend is released (!)
6      // restore \%d15, \%a10 will be restored
7      mov.d   \%d15, \%a10
8      // load address fix
9      // (macro consists of two operations)
10     LDA \%a10, 0xD000D000
11     // data / instruction synchronization
12     dsync
13     isync
14     // enable interrupts
15     enable
16 rfm

```

Listing 3.1: Return from Monitor Sequence

by the switching between the CHILS and the user application. Hardly any pipeline stalls should occur if the programme is executed without disturbance. As mentioned before, the *isynch* operation flushes the CPU pipeline and forces all previous instructions to finish. So the runtime depends on the previously executed code and the actual state of the CPU.

### 3.5.2.4 Caching

If the monitor is executed from a cached area, a pollution of the cache is possible, so the cached instructions or data from the user application are overwritten. On the other hand, the execution time for the additional instructions needed by the CHILS monitor is reduced by ten CPU cycles (see 3.5). In table 3.6 a difference from two to eleven CPU cycles is caused between cached and non-cached execution. The effect strongly depends on the user application.

### 3.5.3 Possibilities for Compensation

Possibilities to compensate the measured time differences are rare. The effect of cache pollution can be solved by executing the CHILS monitor



Application	Step (CPU cles)	Size Cy- Cycles	CPU Cycles per Exchange	CPU (excl. Monitor Instr.)	Cycles approx.
<i>Monitor - Non Cached</i>					
NOPS - Flash	20000		36.28	6.28	
	10000		37.30	7.30	
	5000		38.01	8.01	
NOPS - SRAM	20000		35.60	5.60	
	10000		35.26	5.26	
	5000		35.20	5.20	
Sieve - Flash	20000		31.98	1.98	
	10000		29.07	-0.93	
	5000		34.09	4.09	
Sieve - SRAM	20000		35.39	5.39	
	10000		35.38	5.38	
	5000		35.34	5.34	
<i>Monitor - Cached</i>					
NOPS - Flash	20000		27.60	6.60	
	10000		27.33	6.33	
	5000		27.20	6.20	
NOPS - SRAM	20000		27.60	6.60	
	10000		27.33	6.33	
	5000		27.20	6.20	
Sieve - Flash	20000		23.01	2.01	
	10000		21.69	0.69	
	5000		25.11	4.11	
Sieve - SRAM	20000		27.38	6.38	
	10000		27.39	6.39	
	5000		27.34	6.34	

Table 3.6: STM Time Difference (incl. and excl. Monitor Instructions)

from a non-cached memory region, and by accepting less execution performance. The advantage is that the time difference is better predictable. The difference between the simulation time and  $\mu\text{C}$  internal time can be compensated by reducing the planned step size by the approximated step

size. This is easy to apply, but the available runtime for the user application is slightly reduced.

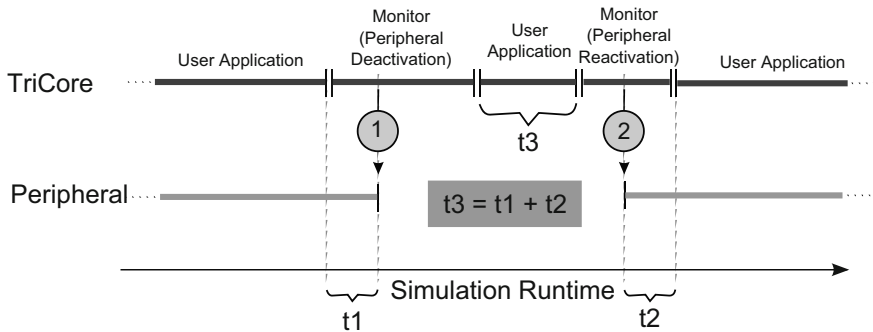


Figure 3.15: Compensation by Regaining of Lost Time

A more complex solution is needed to compensate this runtime difference. It would be possible to give the user application the chance to catch up the lost time. This can be done by suspending the peripherals while running the application. Figure 3.15 shows the basic principle. Caused by the *Delayed Suspend*, the peripheral runs  $t_1$  time units before it is suspended. After the CHILS monitor exchanged the data with the simulation, the peripherals are not immediately reactivated. First of all the CHILS monitor gives the application the chance to regain the lost time  $t_1$ , by running for  $t_3$  time units.  $t_3$  also includes the time  $t_2$ , which is lost because of the additional monitor instructions needed after releasing the peripheral suspend. The problem is that it has to be guaranteed that the user application did not access any peripherals while they are suspended<sup>20</sup>.

## 3.6 Summary

The hardware to simulation coupling concept used in this work is based on concepts of simulator coupling in Co-Simulation environments. The approach covers connections to continuous and discrete simulations which differ in interpretation of time, communication, and activation of simulation

<sup>20</sup> This technique can be applied if the debugging system supports the debug trap generation for peripheral accesses. The CHILS monitor could be called immediately if a peripheral access is detected.

modules. A conservative synchronization principle is chosen where both sides, simulation and  $\mu C$ , are forced to execute identical time steps. Lost or delayed events are accepted because a perfect event detection on  $\mu C$  side is currently not possible and only some simulation environments support a state rollback. The most practical and high-performance way is to run the simulation and the  $\mu C$  in parallel. Events are exchanged after a predefined time between simulation and hardware. A fixed step size and the variable step size is possible.

## 4 Interfaces between Microcontroller and Environment

The previous chapter explained the mechanism of data exchange between the  $\mu$ C and the simulation. The next step is to define which data has to be transferred and in which way it is done. These questions are depending on the interfaces of the  $\mu$ C. The discussion includes the representation of the interfaces from the user application point of view and from the simulation point of view. Especially for the simulation it is necessary to create models of the interfaces. In addition, a certain level of abstraction is needed to exchange the data in an efficient way. This also covers the representation of the data.

### 4.1 Related Work

The presented related work focuses on interface abstraction and interface modelling. A strong relation with simulation coupling techniques exists which were presented in the previous chapter.

The research group of Professor Gabriela Nicolescu from the *Ecole Polytechnique de Montréal* works on discrete–continuous Co-Simulation environments. In *An XML-based Meta-model for the Design of Multiprocessor Embedded Systems* [CGL<sup>+</sup>00] and *A Generic Wrapper Architecture for Multi-Processor SoC Cosimulation and Design* interface definitions with four levels of abstraction are used: **Service Level**, **Message Level**, **Driver Level** and **Register Transfer Level (RTL)** (see table 4.1). The levels are introduced according to the communication abstractions. The choice is a matter of the specification and the design process. The highest level of the communication specification are **Services** supplied to the environment. The communication is seen as the interaction of processes. A process can request a service from another process. Service-request models are very common in web-based software development.

The **Message Level** describes how processes communicate in the respect of concurrency, synchronization, and channel behaviour. Active chan-

Ab- straction Level	Communication			Typical Commu- nication Primitive
	<i>Media</i>	<i>Data Type</i>	<i>behaviour</i>	
<i>Service</i>	type-resolved dynamic net	namespaces + concrete and alge- braic data types	routing	request(print, device, file)
<i>Message</i>	active chan- nels with complex data structures	concrete generic data types	protocol conversion	send(data, disk)
<i>Driver</i>	logical inter- connections	fixed enu- merated data types	driver-level protocol	write(data, port) wait until x==y
<i>RTL</i>	binary signals	fixed bit- vector data types	transmission	set(value, port) wait(clock)

Table 4.1: Interface Abstraction [CGL<sup>+</sup>00]

nels are used to model the communication between connected modules. Different forms of communication are modelled by changing the channel behaviour. Messages contain concrete generic data types.

On **Driver Level**, system are modelled by interconnected modules which communicate through logical connections. The different modules exchange fixed enumerated data types like integers. Typical communication abstractions are master-slave buses or First In - First Out (FIFO) based point-to-point communications.

The **RTL** is defined as the lowest level of abstraction. The combinatory logic that controls the registers and any address decoding is explicitly defined. Binary signals are used as communication primitive.

A similar definition of interface abstraction levels can be found *Programming models and HW-SW interfaces abstraction for multi-processor SoC* [JBP06]. The presented concept defines five abstraction levels: **System Level**, **Virtual Architecture Level**, **Transaction Accurate Level**, **Virtual Prototype**

and **RTL** (see table 4.2). This definition is strongly related to the different abstraction levels of SoC modelling. The approach uses parallel programming models to abstract hardware-software interfaces in the case of heterogeneous SoCs. The authors define a programming model as a set of functions (or primitives) that can be used by software to interact with hardware.

<b>Abstraction Level</b>	<b>Communication Primitives</b>	<b>HW-Access Primitives</b>
<i>System Level</i>	data exchange e.g send/receive(data)	functional access to specific resources
<i>Virtual Architecture</i>	data exchange + synchronization for example Posix threads, lock/unlock (data)	specific I/O protocols related to architecture
<i>Transaction Accurate</i>	data access with specific addresses for example read/write(data, adr)	physical access to HW resources
<i>Virtual Prototype</i>	HW dependant I/O for example DMA and memory mapped I/O	physical I/Os
<i>RTL</i>	load and store	physical I/Os

Table 4.2: Interface Abstraction [JBP06]

The concept of **Polymorphic Signals** in [Lan06], [Bra06] and [SGW04] has been mentioned before. **Polymorphic Signals** are designed to realize a data exchange between different kinds of models. A conversion of types, value ranges and clock rates is done implicitly. The next step is to implement an implicit conversion between different levels of abstraction, too.

Transaction based interfaces for the connection between models on different levels of abstraction are another topic of research. In *A Transaction-Based Unified Architecture for Simulation and Emulation* [HKPS05], Soha Hassoun, Murali Kudluga, Duaine Pryor, and Charles Selvidge present a layered architecture for simulation and emulation. The architecture uses transactions to realize the communication between the Driving Environment (DE) and Device under Test (DUT). The environment, a high-level application, generates the test stimuli. The stimuli are translated by **transactors** into a sequence of cycle-level stimuli for the DUT.

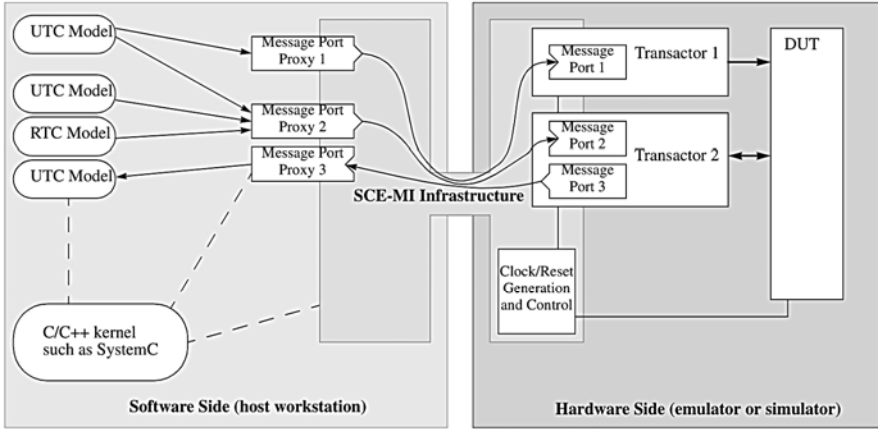


Figure 4.1: SCE-MI Interface between SW Model and DUT [HKPS05]

The Standard Component Emulator Modelling Interface (SCE-MI) standard also uses transactors to connect software models and structural hardware models for verification applications [PAB<sup>+</sup>05].

The SCE-MI provides a transport infrastructure between the emulator and the host workstation sides of each channel which interconnects transactor models in the emulator to C (untimed or RTL) models on the workstation. For purposes of this document, the term emulator can be used interchangeably with any simulator capable of executing RTL or gate-level models, including software HDL simulators.

A transactor transforms the data between two interfaces. It matches the hardware to the specific protocol interface, while high-level commands are provided for the software to perform specific actions. Damian Deneault and Lauro Rizzatti present an application of the SCE-MI standard in *Evaluating and improving emulator performance* [DR04].

## 4.2 Interface Modelling and Abstraction

The interface modelling and abstraction concept is independent from the event exchange concept introduced in the previous chapter. The exchange concept covers the question of how to exchange the data, while this chapter

addresses the question what is the data to be exchanged. This question depends on the available interfaces and the level of interface abstraction. The communication takes place between the user application, executed on the  $\mu\text{C}$ , and the simulation on the PC. The abstraction level of an interface model has to be chosen with respect to the data format expectations of user application and of the simulation. First of all the approach of interface abstraction is introduced, then the interface representation is discussed.

### 4.2.1 Interface Abstraction

There are two aspects to consider about interface abstraction: first the abstraction levels have to be suitable for the available  $\mu\text{C}$  interfaces and second, a higher level of abstraction can accelerate the simulation- $\mu\text{C}$  coupling by reducing the amount of data shared between the  $\mu\text{C}$  and the simulation environment.

The CHILS approach adapts the proposal for communication abstraction levels presented in [CGL<sup>+</sup>00]. Unlike this proposal, the Analogue-Level has been added. Four levels of interface abstraction are defined (see table 4.3) with respect to the available  $\mu\text{C}$  interfaces. The highest level of abstraction is based on messages. An higher level of abstraction, for example based on services, is not needed for the target application of CHILS.

In theory, every  $\mu\text{C}$  interface can be modelled on the Analogue-Level, but this would not be efficient. The CHILS device interfaces to the simulation environment are implemented on different levels of abstraction, depending on the interface type. For example the values of the General Purpose Input/Output (GPIO) ports are exchanged on Digital-Level as bit vectors. For the communication interfaces, the exchange can be done on Byte-Level or Message-Level as primitive or composed data-types. Nevertheless, it is possible to define for example the ASC on the Digital-Level, if the simulation models the serial communication protocol. Normally, concentrating on the the data bytes transmitted via ASC would be enough.

### 4.2.2 Interface Representation

The interface representation, a kind of interface model, provides a direct connection for the user application and for the simulation. So each interface module consists of two parts. One part is located on the  $\mu\text{C}$ , the other part is located on the simulation computer. The  $\mu\text{C}$  part has to provide memory locations for the input and output values of the user application. In an



Abstraction Level	Data Type	Example	Communication Primitive
<i>Message-Level</i>	non-fixed size generic data types	CAN	send(data, device)
<i>Byte-Level</i>	primitive fixed size data types	ASC/SSC	write(data, port) wait until x=y
<i>Digital-Level</i>	digital values, bit vector	Port/Pin	set(value,port) wait(clock)
<i>Analogue Level</i>	analogue values	A/D, D/A	stimulate(value, pin)

Table 4.3:  $\mu$ C Interfaces Abstraction Levels

ideal setup, the user application uses the same memory addresses as for the real communication. This is not always possible due to restrictions in accessibility of the peripheral registers. Instead of real registers the presented concept uses **virtual registers** for some of the peripherals. This is not a disadvantage because such adaptations can be realized in the driver software which is normally used to access peripherals.

The interface representation on the simulation computer, which is part of the CHILS device implementation, has to provide connection possibilities for the simulated  $\mu$ C environment. The type of provided and expected data types depends on the interface itself. Which type of data is chosen for each interface is explained later. Figure 4.2 shows the module concept and the bipartite representation. The user application reads and writes configuration data and payload data from virtual or real registers of the CHILS monitor interface part. The CHILS monitor transfers the data to the CHILS device. The interface part in the CHILS device evaluates and converts the data. The automatic conversion of data types between different abstraction levels is based on the same idea as the concept of **Polymorphic Signals** in [Lan06], [Bra06] and [SGW04]. A transactor-like technique (see SCE-MI standard [PAB<sup>+</sup>05]) can be used for implementation.

The communication of both interface module parts is realized on the highest level of abstraction which fulfills the demands on the interface. In order to demonstrate this idea, the Analogue to Digital Converter (ADC) implementation is discussed. The simulation produces a simulated ana-

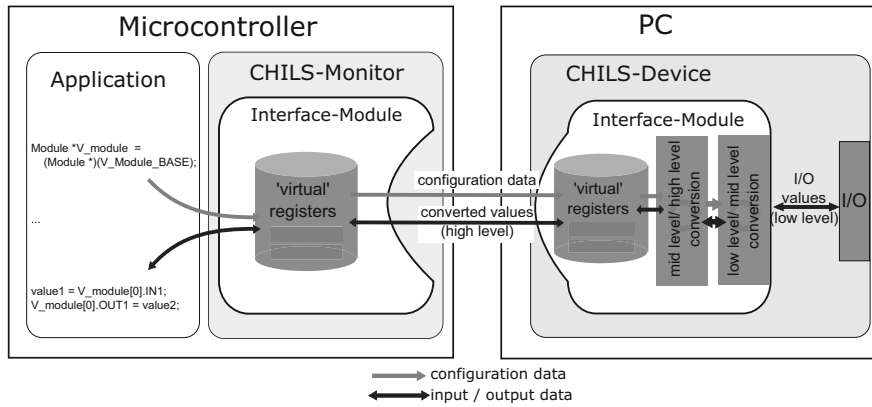


Figure 4.2: Module Concept

logue value as 32Bit or 64Bit floating point value to the ADC interface of CHILS device. It is possible to send this floating point value to the CHILS monitor, but the user application just expects a 8 to 12Bit value. So the conversion of the simulated analogue value is done in the interface part of the CHILS device. The necessary configuration data is provided by the CHILS monitor. Only the converted 8 to 12Bit value is send to the user application, instead of the 32Bit or 64Bit floating point value.

#### 4.2.2.1 Digital Interfaces

The implementation of the digital interfaces and peripherals, namely the GPIO and the GPTA, is straightforward. The GPIO output registers are directly writable, while the input registers can be set via a loop-back to the output registers. The GPTA uses the same input and output lines, so the GPTA signal can be acquired via GPIO registers. The user application uses the real registers in both situations. The data exchange is realized on the **Digital-Level**, so bit vectors are exchanged. The simulation receives and sends bits via single virtual pins.

#### 4.2.2.2 Analogue Interfaces

The analogue realization, which covers the ADC and the Fast Analogue Digital Converter (FADC), is shown in figure 4.3. The module part within the CHILS monitor uses virtual registers for configuration and communica-

tion. As mentioned before the AD conversion is done by the CHILS device part of the interface. The data exchange is realized on **Byte-Level**, so fixed size primitive data types are exchanged between the CHILS device and the CHILS monitor. But of course the simulation sets a simulated analogue value (32Bit or 64 Bit floating point value) as input to the CHILS device.

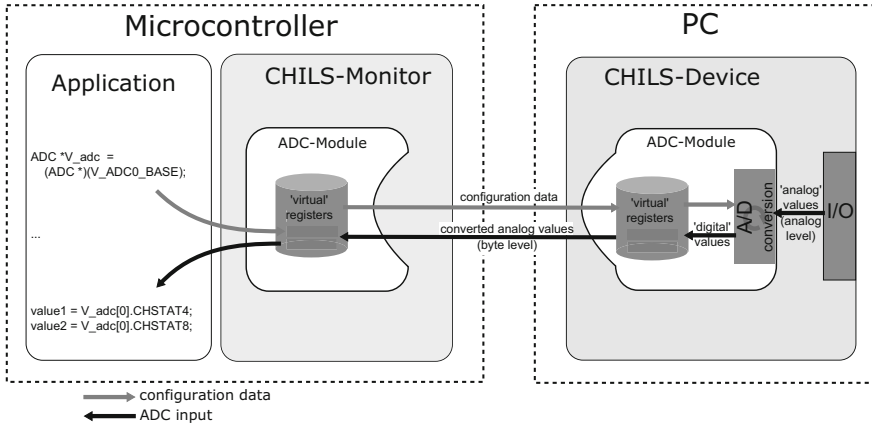


Figure 4.3: ADC Module

#### 4.2.2.3 Communication Interfaces

The communication interfaces require a synchronization of the exchanged data. Special virtual registers are defined to transfer data between the application and the CHILS device. Listing 4.1 shows the data structure for the SSC. The driver can send and receive fully synchronized data<sup>1</sup>. Figure 4.4 presents the receive synchronization concept. The driver signalsizes to the CHILS monitor (and the CHILS device) that it is ready to receive data (set *req*). Then the CHILS device sends available data from the simulation to the CHILS monitor. The CHILS monitor sets the data and the acknowledgment flag (set *data*, set *ack*) and resets the request (reset *req*). The acknowledgment flag is reset by the driver as soon as it reads the data (reset *ack*).

Figure 4.5 shows the send synchronization concept. The driver signalsizes to the CHILS monitor and to the CHILS device that it has new data to transmit and sets this data (set *req*, set *data*). Then the CHILS monitor

1 Of course the driver can ignore the flags and just read and write the input/output registers

```

1 typedef struct ssc_sync_t {
2     volatile unsigned int SEND;
3     volatile unsigned int RECEIVE;
4     volatile unsigned int SEND_ACK;
5     volatile unsigned int RECEIVE_ACK;
6     volatile unsigned int SEND_REQ;
7     volatile unsigned int RECEIVE_REQ;
8 } ssc_sync_t;

```

Listing 4.1: Virtual Registers for SSC Interface

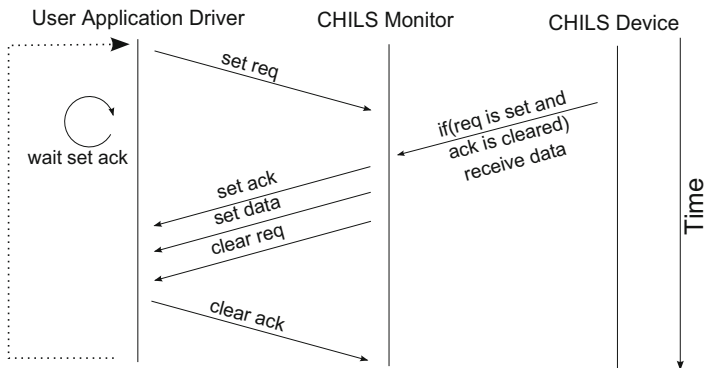


Figure 4.4: Receive Synchronization Driver - Monitor

sends the available data to CHILS device. The CHILS device transmits the data to the simulation and signals the CHILS monitor that the data has been transmitted. The CHILS monitor resets the request flag and sets the acknowledgment flag, so the data is only transmitted once (reset *reg*, set *ack*). The acknowledgment flag is reset by the driver as soon as possible (reset *ack*).

**ASC/SSC** ASC and SSC are realized via the described set of virtual registers. The data exchange is implemented on **Byte-Level**, so fixed size primitive data types are exchanged between the CHILS device and the CHILS monitor.

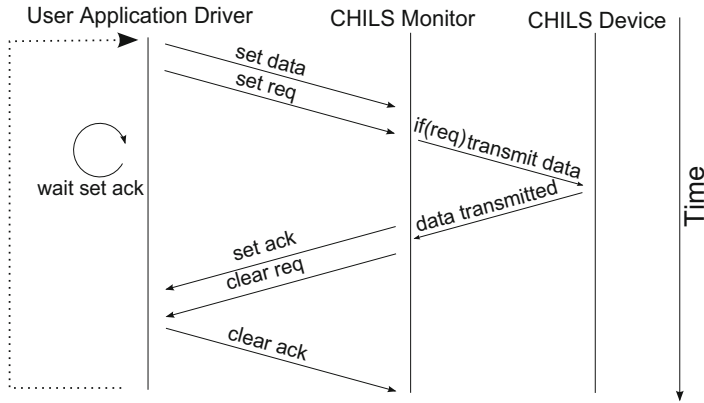


Figure 4.5: Send Synchronization Driver - Monitor

**CAN** The CAN peripheral<sup>2</sup> can be realized in the same way as the ASC and SSC. The virtual registers will contain the payload of a CAN message. So the data exchange can be implemented on **Message-Level**.

### 4.3 Summary

Four levels of interface abstraction are defined (see table 4.3) with respect to the available  $\mu\text{C}$  interfaces: Message-Level, Byte-Level, Digital-Level and Analogue Level. The highest suitable level of abstraction for an interface is chosen to reduce the amount of data shared between the  $\mu\text{C}$  and the simulation environment. This reduction accelerates the simulation- $\mu\text{C}$  coupling. Each interface module consists of two parts. One part is located on the  $\mu\text{C}$ , the other part is located on the simulation computer. The simulation computer part of each interface module is able to convert the data between the different levels of abstraction. The  $\mu\text{C}$  part of an interface consists of real or virtual registers, which are readable and writable by the user application. Normally a driver layer hides the implementation, so the user software does not have to be changed to run on the final system.

<sup>2</sup> The CAN is not implemented yet.

## 5 Optimization – Coupling System Analysis

The goal of the HIL simulation system analysis is an optimized setup for the data exchange between the hardware and the simulation (see figure 5.1). This is primarily a tradeoff between accuracy and performance. An optimized setup covers the type and amount of exchanged data and the step size between two data exchanges. The first claim can be obtained by choosing the highest level of interface abstraction which is suitable for the desired functionality. The second claim concerns the rate of changes of the exchanged data and the necessity of change/events distribution<sup>1</sup>. Two techniques can be used, an analysis of the running system and a pre-analysis of the system.

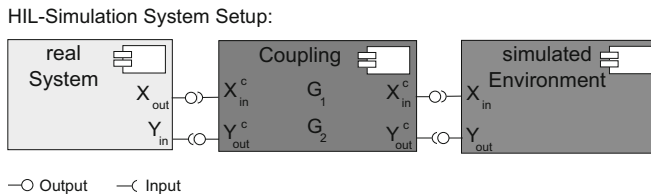


Figure 5.1: HIL System

**Runtime Analysis** The runtime analysis, or analysis within the runtime, focuses on the detection of non-distributed<sup>2</sup> changes of exchange values between hardware and simulation while the system is running. This information can be used to dynamically adapt the step size of the exchange to find a compromise between accuracy and performance. Runtime analysis is covered in chapter 7.

<sup>1</sup> An event can be defined as an indicator for a change [ZPK00].

<sup>2</sup> A change/event is called non-distributed if it is produced by a signal source but not distributed to the corresponding sink.

**Pre-Analysis** The pre-analysis of the system helps to determine the influence of errors<sup>3</sup> introduced by the coupling system. It depends strongly on the algorithms used on both sides (simulation and  $\mu C$ ) to process the exchanged data. Algorithm analysis is a part of numerical mathematics. The basis for  $\mu C$  algorithm analysis is the source code of the application (chapter 6). The pre-analysis based on the coupling system is described within this chapter. These analysis techniques have been partly presented in the paper *Determining the Fidelity of Hardware-In-the-Loop Simulation Coupling Systems* for the 2008 IEEE International Behavioural Modeling and Simulation Conference [KMH08b].

## 5.1 Related Work

The presented work is based on the **system theory** for the description of Linear Time Invariant (LTI) systems. Irwin W. Sandberg's article *A Perspective On System Theory* [San84] presents a general introduction to the development of system theory. System theory deals with the conversion of signals via systems. The authors of the book *Einführung in die Systemtheorie* (introduction into system theory) define a signal and a system as follows [GRS07].

**Definition 5.1.1.** *A signal is a function or a series of values which present information.*

**Definition 5.1.2.** *A system is the abstraction of a process or a structure which correlates several signals to each other.*

The LTI systems theory is an important subset of system theory. This part of system theory is well prospected and defined. The primarily mathematical representations of LTI, the **linear differential equations**, are easy to apply. In contrast to the linear systems theory, the theory of nonlinear systems is still a field of research with open questions, especially the mathematical part which is based on **nonlinear differential equations**. Several books about linear system theory can be found, for example *Signale und Systeme* (Signals and Systems) [Kie98] or *Signale, Prozesse, Systeme* (Signals, Processes, Systems) [Kar04].

---

<sup>3</sup> The term errors is used in the context of numerical mathematics. An error is the discrepancy between an expected exact value and the retrieved value. In a larger scope an error can be seen as a system state which does not matches the specification.

The attempt to formulate the problem of the theoretical accuracy of HIL simulations can be found [Bac05] , [Bac07] and [MMB07]. In *Quantifying the Accuracy of Hardware-in-the-Loop Simulations* the authors M. MacDiarmid and M. Bacic write that the “quantification of the accuracy of HWIL simulators presents unique challenges, and remains an open research problem”. The complete system, HIL simulator and real system are contemplated within the mentioned papers. The authors model the coupled system as a two-port network. Especially digital systems and control hardware systems are excluded from the approach.

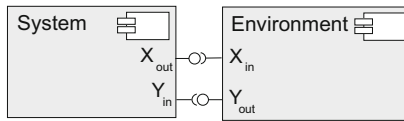
My attempt is to formulate the problem for general HIL systems, including the coupling system of an ECU-HIL simulation and similar solutions, like the CHILS approach. The focus is set to the HIL simulation coupling system itself without modelling the hardware and/or the simulation part.

## 5.2 Basics

The starting point is given by the complete system within its original environment (upper part of figure 5.2). The system output is the vectorial variable  $X_{out}$  corresponding to the vectorial variable  $X_{in}$ . The system input is the vectorial variable  $Y_{out}$  corresponding to the vectorial variable  $Y_{in}$ .

$$\begin{aligned} X_{in} &= X_{out} \\ Y_{in} &= Y_{out} \end{aligned} \quad (5.1)$$

Real System Setup:



HIL-Simulation System Setup:

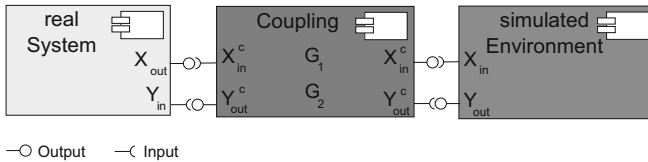


Figure 5.2: HIL System and Real System



In the HIL simulation the system remains the same, while the environment is simulated. The coupling system has the same input and output types as the real system and its environment. The coupling system, as a signal processing system, transforms the output of the real system and the output of the simulated environment. The transformation functions are  $G_1(t)$  and  $G_2(t)$ .

$$\begin{aligned} X_{in}(t) &= G_1(t) * X_{out}(t) \\ Y_{in}(t) &= G_2(t) * Y_{out}(t) \end{aligned} \quad (5.2)$$

The transformation can be reformulated as follows.

$$\begin{pmatrix} X_{in}(t) \\ Y_{in}(t) \end{pmatrix} = \begin{bmatrix} G_1(t) & 0 \\ 0 & G_2(t) \end{bmatrix} * \begin{pmatrix} X_{out}(t) \\ Y_{out}(t) \end{pmatrix} \quad (5.3)$$

An ideal coupling system does not change the transmitted signal, so the “hardware-in-the-loop” senses no difference to a connection with the real environment. The term **transparency** can be used for this purpose [Bac05]. A design goal of the coupling system would be to design the system as transparent as possible. That means that the transformation functions  $G_1(t)$  and  $G_2(t)$  are nearly one (equation 5.4).

$$\begin{bmatrix} G_1(t) & 0 \\ 0 & G_2(t) \end{bmatrix} \approx \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad (5.4)$$

The focus of interest now is how to measure the **transparency of the coupling system**?

## 5.3 Formal Definitions

Based on the general description of the coupling system the following definitions are introduced to measure the **transparency of the coupling system**.

### 5.3.1 Transparency and Fidelity Definition

The basis of measuring the transparency of the coupling system is a model of the coupling system. Unlike the approach in [Bac05] and [Bac07], it is not necessary to model the real system or its environment. It is feasible to model only the coupling system.

**Assumption 5.3.1.** *The coupling is assumed to be representable as a **LTI system**. This is a general approximation that is often used because non-linear systems are hard to model. Most of the non-linear systems can be approximated with linear models within their normal working range. Real world systems are mostly non-linear.*

**Definition 5.3.2.** *A system is called linear if and only if the following equations are valid for all input variables:*

$$a \cdot f(x_{in}) = f(a \cdot x_{in}) \quad (\text{homogeneity}) \quad (5.5)$$

$$f(x_{in1}) + f(x_{in2}) = f(x_{in1} + x_{in2}) \quad (\text{superposition property}) \quad (5.6)$$

*This is called the **linearity principle**. A system is called nonlinear if the linearity principle is not valid.*

For the next steps, we will use the **transfer function** within the frequency domain. The advantage is that the transfer function directly shows the relation of the input and the output signal.

**Definition 5.3.3.** *A LTI system can be described by the convolution of the input signal with the impulse response*

$$y(t) = g(t) * x(t) \quad (5.7)$$

Assuming that  $x(t)$  is the input signal and  $y(t)$  is the output signal of a Single-Input Single-Output (SISO) system. In the frequency domain, the corresponding Laplace transformed signals are  $x(s) = \mathcal{L}\{x(t)\}$  and  $y(s) = \mathcal{L}\{y(t)\}$ .

**Definition 5.3.4.** *The transfer function is defined as*

$$y(s) = h(s)x(s) \quad \text{and so} \quad h(s) = \frac{y(s)}{x(s)} \quad (5.8)$$

$y(s)$  and  $x(s)$  are polynomials of degree  $m$  (w.l.o.g.  $x(s)$  and  $y(s)$  have the same degree).

$$h(s) = \frac{y(s)}{x(s)} = \frac{b_0 + b_1s^1 + \dots + b_ms^m}{a_0 + a_1s^1 + \dots + a_ms^m} \quad (5.9)$$

**Lemma 5.3.5.** *In a completely transparent system, the input has to be identical to the output, so  $y(s) = x(s)$ . The **transparency of a signal transformation system** can now be defined by the difference of the two polynomials  $y(s)$  and  $x(s)$  of the transfer function  $h(s)$ .*

For the calculation of the transparency, I define a  $m + 1$ -dimensional space  $\prod^m$  over polynomials  $\sum_{i=0}^m a_i k^i$ . The power  $i$  stands for the space axis while the coefficients  $a_i$  are the values in each dimension  $i$ .

**Lemma 5.3.6.** *The difference between two polynomials  $y(s)$  and  $x(s)$  can be defined as the distance of the polynomials within the  $m + 1$ -dimensional space  $\prod^m$  over polynomials of the degree  $m$ .*

**Definition 5.3.7.** *A weighted distance  $d_w(x(s), y(s))$  with*

$$x(s) = a_0 + \dots + a_m s^m \quad (5.10)$$

and

$$y(s) = b_0 + \dots + b_m s^m \quad (5.11)$$

is defined as

$$d_w(x(s), y(s)) = \left\| \begin{pmatrix} a_0 \\ \vdots \\ a_m \end{pmatrix} - \begin{pmatrix} b_0 \\ \vdots \\ b_m \end{pmatrix} \right\|_w \quad (5.12)$$

with the weighted norm

$$\|\dots\|_w = \sqrt{w_0(a_0 - b_0)^2 + \dots + w_m(a_m - b_m)^2} \quad (5.13)$$

The weights  $w_i \geq 1$  are used to satisfy the influence of the different polynomial exponents on the whole polynomial difference. For the examples in section 5.4  $w_i = \sum_{j=0}^i j + 1$  is chosen.

The transparency is now defined for SISO systems. Most of the transfer systems are Multiple-Input Multiple-Output (MIMO) systems (figure 5.3).

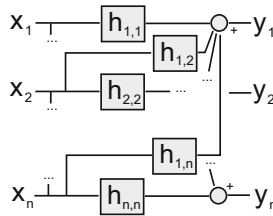


Figure 5.3: MIMO System

**Definition 5.3.8.** *MIMO systems can be described by a matrix of SISO transfer functions*

$$H(s) = \begin{bmatrix} h_{1,1}(s) & \cdots & h_{n,1}(s) \\ \vdots & \ddots & \vdots \\ h_{1,n}(s) & \cdots & h_{n,n}(s) \end{bmatrix} \quad (5.14)$$

with

$$Y(s) = \begin{pmatrix} y_0(s) \\ \vdots \\ y_n(s) \end{pmatrix} \quad (5.15)$$

and

$$X(s) = \begin{pmatrix} x_0(s) \\ \vdots \\ x_n(s) \end{pmatrix} \quad (5.16)$$

so

$$Y(s) = H(s)X(s) \quad (5.17)$$

**Remark 5.3.9.** *The transfer matrix is square because the vectors  $X(s)$  and  $Y(s)$  have the same size.*

The main diagonal elements represent the direct transfer functions between each input  $x_i$  and each output  $y_i$ . The other elements are couplings between different inputs and outputs ( $x_i$  and  $y_j$  with  $i \neq j$ ).

**Definition 5.3.10.** *The ideal transfer function matrix has a main diagonal containing ones. The other matrix elements are zero.*

$$\begin{pmatrix} y_1(s) \\ \vdots \\ y_n(s) \end{pmatrix} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & \ddots & 0 & \vdots \\ \vdots & 0 & \ddots & 0 \\ 0 & \cdots & 0 & 1 \end{bmatrix} \begin{pmatrix} x_1(s) \\ \vdots \\ x_n(s) \end{pmatrix} \quad (5.18)$$

This means there are no influences between different inputs and outputs, while the direct connection between each input and each output does not change the signal, so  $\frac{y(s)}{x(s)} = 1$ .

**Definition 5.3.11.** A norm  $\|h(s)\|_p$  over a polynomial quotient  $h(s) = \frac{y(s)}{x(s)}$  can be defined over the distance of  $x(s)$  and  $y(s)$  in  $\prod^m$ :

$$\left\| \frac{y(s)}{x(s)} \right\|_p = d_w(x(s), y(s)) \quad (5.19)$$

This norm can be used as a measure for the transparency.

**Remark 5.3.12.** If  $x(s)$  and  $y(s)$  are identical the distance is zero, so the system is fully transparent and the transfer function does not change the input signal. In all other cases the distance is larger than zero.

**Definition 5.3.13.** In addition, the difference between the upper polynomial  $y(s)$  and a zero polynomial can be defined as norm  $\left\| \frac{y(s)}{x(s)} \right\|_p^0$  over the distance of  $y(s)$  and 0 in  $\prod^m$ :

$$\left\| \frac{y(s)}{x(s)} \right\|_p^0 = d_w(0, y(s)) \quad (5.20)$$

**Definition 5.3.14.** A **matrix of transparency** can be defined as follows. The main diagonal contains the elements  $\|h_{i,i}(s)\|_p$  with  $1 \leq i \leq n$ , while the other positions are filled with elements  $\|h_{i,j}(s)\|_p^0$  with  $1 \leq i \leq n, 1 \leq j \leq n, i \neq j$ :

$$\begin{bmatrix} \|h_{1,1}(s)\|_p & \|h_{1,2}(s)\|_p^0 & \dots & \|h_{1,n}(s)\|_p^0 \\ \vdots & \ddots & \ddots & \vdots \\ \|h_{n,1}(s)\|_p^0 & \|h_{n,2}(s)\|_p^0 & \dots & \|h_{n,n}(s)\|_p \end{bmatrix} \quad (5.21)$$

An ideal matrix of transparency is the zero matrix.

$$\begin{bmatrix} 0 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \end{bmatrix} \quad (5.22)$$

**Definition 5.3.15.** With the help of a matrix norm we can now define a **transparency function**  $\text{tr}$  for a MIMO system transfer matrix. The Euclidean norm is chosen in the example in section 5.4.

$\text{tr}(H(s)) =$

$$\left\| \begin{bmatrix} \|h_{1,1}(s)\|_p & \|h_{j,i}(s)\|_p^0 \\ & \ddots \\ \|h_{i,j}(s)\|_p^0 & \|h_{n,n}(s)\|_p \end{bmatrix} \right\| \quad (5.23)$$

**Definition 5.3.16.** The *fidelity function*  $\mathfrak{f}\mathfrak{d}$  of a coupling system can now be defined by the transparency of the transfer function. The value of the fidelity ranges between zero and one.

$$\mathfrak{f}\mathfrak{d}(H(s)) = \frac{1}{1 + \text{tr}(H(s))} \quad (5.24)$$

The transformation performed by the coupling system (equation 5.3) is defined as follows:

$$\begin{pmatrix} X_{in}(t) \\ Y_{in}(t) \end{pmatrix} = \begin{bmatrix} G_1(t) & 0 \\ 0 & G_2(t) \end{bmatrix} \star \begin{pmatrix} X_{out}(t) \\ Y_{out}(t) \end{pmatrix} \quad (5.25)$$

**Definition 5.3.17.** For better visibility, a symbolic transformation ( $\star$ ) of the input signals can now be defined as multiplication by the fidelity matrix.

$$\begin{pmatrix} X_{in}(t) \\ Y_{in}(t) \end{pmatrix} = \begin{bmatrix} \mathfrak{f}\mathfrak{d}(\mathcal{L}\{G_1(t)\}) & 0 \\ 0 & \mathfrak{f}\mathfrak{d}(\mathcal{L}\{G_2(t)\}) \end{bmatrix} \star \begin{pmatrix} X_{out}(t) \\ Y_{out}(t) \end{pmatrix} \quad (5.26)$$

This can be interpreted as the proportionate information loss of the original input signal caused by the transformation. The fidelity is one, if the signal is not changed, so no information gets “lost”. An ideal symbolic transformation matrix is now

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad (5.27)$$

**Remark 5.3.18.** *The system is now defined as a continuous version. The same definitions can be made with a discrete system model (the Laplace transformation is replaced by the Z-transformation). In real coupling systems, a discrete part often exists, for example if the input/output values are exchanged via a digital connection.*

### 5.3.2 Application of the Coupling System Fidelity

The **fidelity function**  $\mathfrak{f}\mathfrak{d}(H(s))$  is an instrument to compare different HIL coupling solutions. Three steps are necessary to use the approach.

**First:** model the different coupling systems

**Second:** calculate  $\mathfrak{f}\mathfrak{d}(H(s))$  for each system

**Third:** compare the symbolic transformation matrixes using a matrix norm

It is also possible to find optimal parameters for a parameterized coupling system by starting an optimization process by applying the transparency function  $\text{tr}(H(s))$  and the symbolic transformation matrix. The optimization problem can be defined as  $\min_p(H_p(s))$ .  $p$  is a set of parameters of the transfer function  $H_p(s)$ . The same consideration can be made for discrete coupling systems.

**Definition 5.3.19.** *The transfer function for a discrete LTI SISO system can be defined by*

$$h(z) = \frac{y(z)}{x(z)} = \frac{b_0 + b_1 z^1 + \dots + b_m z^m}{a_0 + a_1 z^1 + \dots + a_m z^m} \quad (5.28)$$

The Laplace transformation is replaced by the Z-transformation.

## 5.4 Example

A heat-sensor-HIL simulation is taken as an example for a continuous coupling system<sup>4</sup>. In order to build a heat-sensor-HIL simulation a heating element is needed to transform the simulated heat into real heat for the sensor.

---

<sup>4</sup> The system example is based on a climate control, which is used in a practical course of control engineering at the HAW Hamburg [Huß08].

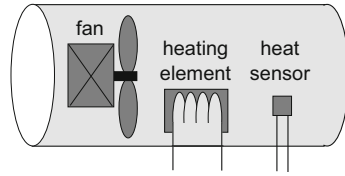


Figure 5.4: Heat-Sensor-in-the-Loop

**Definition 5.4.1.** *The heating element can be described by the following transfer function.*

$$H_h(s) = K * \frac{1}{1 + Ts} \quad (5.29)$$

The proportional coefficient  $K$  and the time constant  $T$  are depending on environmental variables like the specific heat capacity, density and velocity of the transfer medium and the heating element.

$$K = \frac{1}{c_m \gamma_m A v} \quad (5.30)$$

$$T = \frac{c_h}{c_m \gamma_m A v} \quad (5.31)$$

with

$c_m$  - heat capacity of the medium (air  $c_m = 1.01 \frac{Ws}{gK}$ )

$c_h$  - heat capacity of the heating element (steel  $c_h = 0.477 \frac{Ws}{gK}$ )

$\gamma_m$  - density of the medium (air  $c_m = 1293 \frac{g}{m^3}$ )

$v$  - velocity of the medium

$A$  - cross section surface of the pipe where sensor and heating element are located

$l$  - distance between heating element and sensor

The heat sensor and the heating element are positioned in a distance of  $l$  from each other. This causes the delay  $D = \frac{l}{v}$  in heat transportation. It is



assumed that the heat control system corrects the proportional coefficient by adding the correction coefficient

$$C = \frac{1}{K} \quad (5.32)$$

The transfer function of the complete coupling system is now.

$$H_h(s) = C * K * \frac{e^{-Ds}}{1 + Ts} \quad (5.33)$$

$e^{-Ds}$  can be approximated by the Fourier series

$$e^{Ds} = 1 + sD + \frac{s^2 D^2}{2!} + \dots \quad (5.34)$$

The fidelity function of the system is now calculated assuming an air velocity of  $v = 1m/s$ , a sensor distance of  $l = 0.1m$  and a pipe cross section surface of  $A = 0.1m^2$ . The calculation (equation 5.35) is done without units. The result is a system fidelity of  $\text{fd}(H_h(s)) = 0.847$ <sup>5 6</sup>.

$$\begin{aligned} T &= \frac{0.477}{1.01 * 1293 * 0.1 * 1} \\ T &= 0.00365 \\ D &= 0.1/1 \\ e^{0.1s} &= 1 + 0.1s + 0.005s^2 \\ H_h(s) &= \frac{1}{(1+0.1s+0.005s^2)*(1+0.00365s)} \\ H_h(s) &= \frac{1}{1+0.10365s+0.00537s^2+0.00002s^3} \\ \text{tr}(H_h(s)) &= \left\| \left[ \begin{pmatrix} 1 \\ 0.10365 \\ 0.00537 \\ 0.00002 \end{pmatrix} - \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \right]_w \right\| \\ \text{fd}(H_h(s)) &= \frac{1}{1 + \text{tr}(H_h(s))} \\ \text{fd}(H_h(s)) &= \underline{\underline{0.847}} \end{aligned} \quad (5.35)$$

Increasing the air velocity to  $v = 10m/s$  leads to better results of the fidelity function  $\text{fd}(H_h(s)) = 0.982$ . Figure 5.5 shows that the heating systems with increased air velocity follows the control input even better than the other system. It is obvious that the higher air velocity leads to a faster heat

<sup>5</sup> The highest reachable fidelity value of a coupling system is one.

<sup>6</sup> The calculation is done for clearness without units.

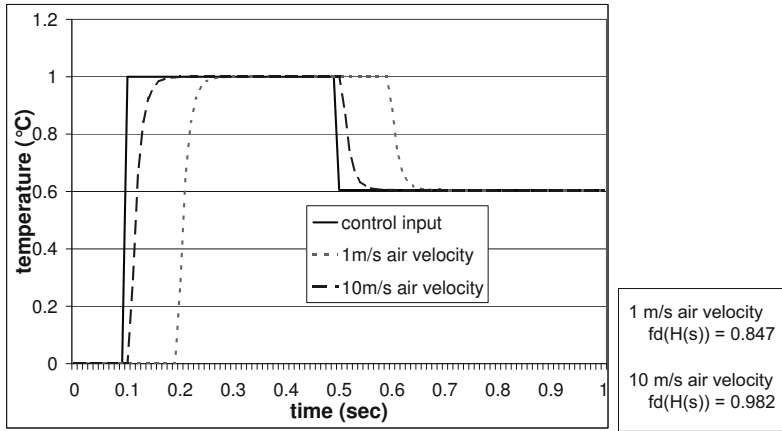


Figure 5.5: Heating System – Different Air Velocities

transport to the heat-sensor-in-the-loop, but it is not clear what happens if the material of the heating system itself is changed. The steel heating element in the original system is replaced by a copper heating element and an aluminium heating element. The heat capacity of copper is  $c_h = 0.381 \frac{W_s}{gK}$  while the heat capacity of aluminum is  $c_h = 0.896 \frac{W_s}{gK}$ . Without a system model it is hard to decide which material is the better choice for the system. The fidelity function of the system is calculated assuming the original settings with an air velocity of  $v = 1m/s$  and a pipe cross section surface of  $A = 0.1m^2$ . The resulting system fidelity is  $\bar{f}d(H_h(s)) = 0.848$  for the copper based system and  $\bar{f}d(H_h(s)) = 0.843$  for the aluminum based system. The copper heating element produces a higher system fidelity than the steel heating element and the aluminum heating element, but the delay in heat transportation is even more important for the fidelity. The three materials are leading to nearly the same results (figure 5.6). The fidelity value reflects this behaviour in a very good way.

## 5.5 Comparison of Different Coupling Systems

CHILS and other HIL solutions are only comparable if the solutions are intended for the same application. The presented pre-analysis approach is suitable to compare different HIL system implementations for the same application. The procedure is shown in figure 5.7. The coupling system is

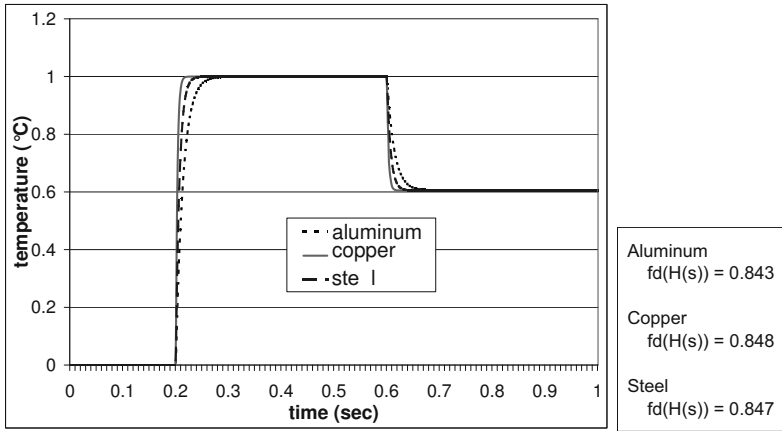


Figure 5.6: Heating System – Different Materials

divided in the analogue part and the digital part. Afterwards the fidelity values of both parts are calculated.

### 5.5.1 Simulation Scenario

Typical automotive systems require sampling times of 1ms while 0.5ms or 0.25ms are feasible for Formula One applications [dSp09]. Based on that information, sample times between 0.1ms and 1ms are chosen for a comparison with the focus on engine control, the main application of the TC1796/TC1766  $\mu C$ . The TC1796/TC1766  $\mu C$  series has hundreds of digital I/O lines and several analogue input lines. The simulation scenario respects these facts.

### 5.5.2 Interpretation of Data Sheets

Data sheets are a general source of information for a comparison. It is necessary to be able to interpret the data sheets correctly.

In the Texas Instruments application report *Understanding Data Converters* [Tex99] the following explanations to ADC and Digital to Analogue Converter (DAC) data sheets can be found.

**Least Significant Bit (LSB)** For ADC, the width of one conversion step is

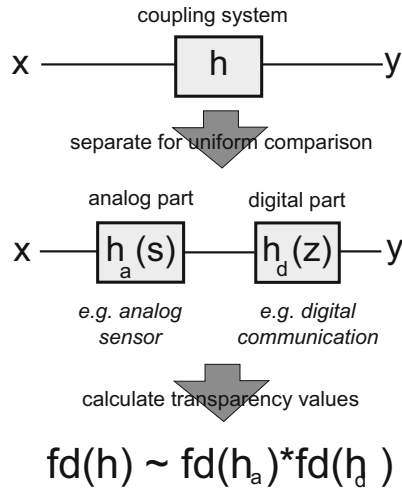


Figure 5.7: System Comparison Modelling

defined as one LSB,  $1LSB = \frac{FS}{2^n - 1}$ <sup>7</sup>. For a DAC, one LSB corresponds to the height of a conversion step between successive analogue outputs. The LSB is a measure of the resolution of the converter because it defines the number of units of the full analogue range.

**Full Scale Range (FSR)** The FSR defines the range of analogue values that can be generated or measured.

**Gain Error** The gain error is defined as the difference between the nominal and the actual gain points on the transfer function after the offset error has been corrected to zero. For an ADC, the gain point is the midstep value when the digital output is full scale, and for a DAC it is the step value when the digital input is full scale.

**Offset Error** The offset error is defined as the difference between the nominal and actual offset points. For an ADC, the offset point is the midstep value when the digital output is zero, and for a DAC it is the step value when the digital input is zero.

**Integral Linearity** The integral linearity error (sometimes seen as simply linearity error) is the deviation of the values on the actual transfer

<sup>7</sup> Full Scale (FS) defines the upper limit of convertible values of an ADC.

function from a straight line. For an ADC the deviations are measured at the transitions from one step to the next, and for the DAC they are measured at each step.

**Differential Linearity** The differential nonlinearity error (sometimes seen as simply differential linearity) is the difference between an actual step width (for an ADC) or step height (for a DAC) and the ideal value of one LSB. Therefore if the step width or height is exactly one LSB, then the differential nonlinearity error is zero.

**Absolute Accuracy Error** The absolute accuracy or total error of an ADC is the maximum value of the difference between an analogue value and the ideal midstep value. It includes offset, gain, integral linearity errors and also the quantization error in the case of an ADC.

**Settling Time** The settling time is the maximal time which a DAC needs to reach an output value of a defined accuracy after the digital input value changed.

**Conversion Rate** The conversion rate is the maximum number of conversion per second which an ADC can achieve.

### 5.5.3 Scenarios

Two scenarios for the comparison are defined. Scenario I contains only digital I/O lines. Setups with 16 or 128 digital input and output lines are compared. Scenario II also includes 8 or 32 analogue input lines.

Scenario I - Digital I/O only:

**Digital  $\mu C$  Input Lines:** 16, 128

**Digital  $\mu C$  Output Lines:** 16, 128

**Analogue  $\mu C$  Input Lines:** 0, 0

Scenario II - Digital I/O and Analogue I/O:

**Digital  $\mu C$  Input Lines:** 16, 128

**Digital  $\mu C$  Output Lines:** 16, 128

**Analogue  $\mu C$  Input Lines:** 8, 32

### 5.5.4 CHILS vs. DeskPOD™

The DeskPOD™ from the SimPOD company is a hardware box with individual connector boards to connect all I/O pins of the  $\mu$ C. DeskPOD™ couples the device with a simulator frontend on the PC. The solution addresses the market of early silicon validation and HW/SW-Co-verification. Normally, the DeskPOD™ has full control over the device by bypassing the PLL, so the  $\mu$ C can be started and stopped by switching the clock on or off. The DeskPOD™ has two modes of operation. In the engaged mode, the DeskPOD™ is under full control of the simulator. In this case DeskPOD™ and simulator exchange the data at each clock cycle, both simulated and real. The performance is limited to 5000-10.000 clock cycles per second.

In the disengaged mode, the DeskPOD™ is executing a cycle and checks for engage conditions or an engage request from the simulator. If the conditions are met it sends the data to the simulator and receives data from the simulator. Otherwise the DeskPOD™ executes the next clock cycle. Depending on the setup, the performance can be as low as 15.000 clock cycles per second and as high as 250.000 clock cycles per second.

A non preferred mode is to let the device run free without bypassing the PLL. So the DeskPOD™ has no control over the device. It will read and write values from and to the  $\mu$ C as fast as possible (5000-10.000 interactions per second). In that scenario the  $\mu$ C runs at full speed. Further details can be found in chapter 9 in section 9.2.

#### 5.5.4.1 DeskPOD™ Hardware

Based on the given technical datasheets (see table D.2.1 in chapter D subsection D.2.1) the following parameters for the LTI system representation can be extracted.

**Interactions between DeskPOD™ and Simulation:** 5000-10000 per sec <sup>8</sup>

**Network Latency:** approx. 0.25ms <sup>9</sup>

SimPOD could not deliver any data regarding the quality of the D/A converters.

<sup>8</sup> An average value of 7000 is chosen for the calculation (see chapter 9 in section 9.2).

<sup>9</sup> This is a typical value for a local area network.

### 5.5.4.2 Scenarios

In scenario I, which includes only digital I/O, the DeskPOD™ is bypassing the PLL. Therefore full control of the system is possible, so the maximum accuracy of the DeskPOD™ solution is reachable. The hardware to simulation coupling is cycle accurate but not very fast. The DeskPOD™ is assumed to run in the disengaged mode, which provides more performance.

In scenario II, which includes digital I/O and analogue I/O, the DeskPOD™ configuration has to be changed. The analogue I/O is needed, so the PLL of the  $\mu$ C cannot be bypassed because the ADC will probably not work at such low frequencies (5000-10.000Hz). The DeskPOD™ has no direct control of the device. The consequence is that the device runs at full speed and the DeskPOD™ exchanges with its maximum data rate. In this case events can be lost. This is a significant problem for communication interfaces like CAN, FlexRay or SSC and ASC. Especially bus interfaces like CAN and Flexray have to run synchronously to the bus system. Data losses, time-outs and varying bit lengths will break the connection. Furthermore, the simulation has to be realtime capable for scenario II.

## 5.5.5 CHILS vs. dSPACE

dSPACE [dsp] offers a wide variance of DAQ cards with digital and analogue inputs and outputs which are used within their simulation computer racks. Special interfaces like CAN are also supported. The user can connect up to 16 I/O boards via the the PHS bus, a special communication bus, within a simulation computer. dSPACE hardware is mostly used to couple a complete ECU with a simulation of the ECU environment, for example a combustion engine. The simulation and the simulator have to be real-time capable. A non real-time use case is not intended.

A real-time simulation of a 6-cylinder gasoline engine including I/O with the dSPACE Automotive Simulation Models Engine Simulation Package can achieve a cycle time of 0.15ms [dSp09]. Table 9.3 *Performance of 6 cylinder engine simulation with dSPACE* in chapter 9 shows some performance values of different simulation configurations.

### 5.5.5.1 dSPACE Hardware

The following dSpace Hardware is needed for the chosen scenarios:

- 3x DS4003 Digital I/O Board

- 1x DS2103 Multi-Channel D/A Board
- 1x DS1006 Processor Board
- 1x DS2211 HIL I/O Board

From the data sheets (see table D.2.2, D.2.2, D.3 and D.4 in chapter D subsection D.2.2) the following values can be interpreted as LTI system: gain error, conversion time and settling time. The other values are properties of nonlinear systems. This is the limitation of the presented approach.

Based on the given technical data sheets the following parameters for the LTI system representation can be extracted:

**D/A Converter Settling Time:** 0,010 ms

**D/A Converter Gain Error:** 0,2%

**Bus Type:** single master bus with 16 clients

**Bus Latency (PHS Bus):** 0,001 ms

The bus latency of the PHS bus is approximated based on its transfer rate and the bus structure<sup>10</sup>. Measured values from a real system are not available.

### 5.5.5.2 Scenarios

Unlike the SimPOD solution and the CHILS approach, all scenarios require a realtime capable simulation. Communication interfaces like CAN, FlexRay or SSC and ASC can be included with special I/O boards.

## 5.5.6 Comparison Results

The results in figure 5.8 and figure 5.9 show a good fidelity of all systems in the range of 0.1ms to 0.25ms exchange cycle time<sup>11</sup>. The DeskPOD™ system has a disadvantage because of the high network latency. The network latency influences the transfer of the halt-signal from the simulation in the disengaged mode, so the data transferred from the DeskPOD™ to the simulation and from simulation to the DeskPOD™ is delayed by this time. If the DeskPOD™ halts the  $\mu$ C because of an internal condition, the transfer

---

<sup>10</sup> The PHS bus is a single master bus, so no concurrency between different masters occurs.

<sup>11</sup> The weight functions for the fidelity calculation can be found in Appendix A.



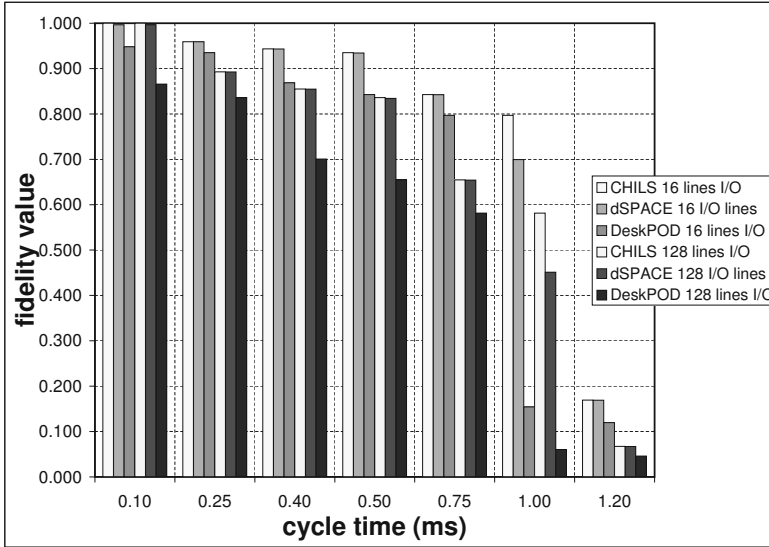


Figure 5.8: Scenario I – Results

is also delayed because the simulation has to be stopped in addition. In the free running mode without a PLL bypass all transfers are delayed by the transfer via network, too.

The dSPACE solution produces also a delay, which is caused by the internal bus system. However, this delay is much smaller than the delay caused by an ethernet connection used by the DeskPOD™ solution. The CHILS approach has the advantage that the transfer latency of the data, which is exchanged between hardware and simulation, does not influence the fidelity. While the data is transferred both side, the  $\mu\text{C}$  and the simulation, are halted.

The results reflect this behaviour. The CHILS approach normally has the highest fidelity value, so the quality of the coupling system itself is better. Especially at the rigid boundary of  $1\text{ms}$  cycle time, the differences are visible. The DeskPOD™ setup delivers a worse fidelity value because the real cycle time exceeds the boundary much more than the other systems ones.

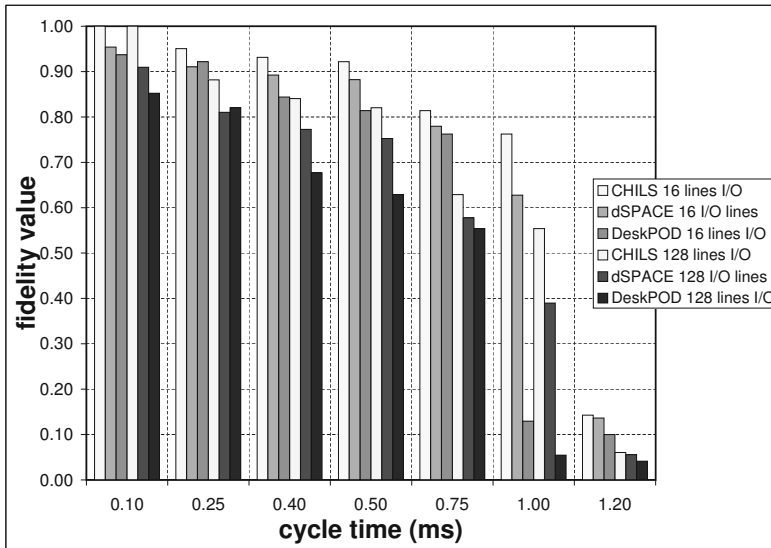


Figure 5.9: Scenario II – Results

## 5.6 Summary

The coupling system analysis calculates the fidelity of HIL simulation coupling systems in a formal way. The calculation is based on the transfer function in the frequency domain of the coupling system. SISO and MIMO systems are covered by this approach. The approach can be used to compare different HIL simulation coupling systems. An optimization process, which is based on the fidelity value, can be executed in order to find the best possible configuration of a coupling system.

The comparison between the CHILS hardware to simulation coupling systems and other HIL solutions, the dSPACE system, a classical HIL solution, and the DeskPOD™ from the company SimPOD, a special solution for  $\mu$ C to simulation coupling, shows that the CHILS approach has a higher coupling system fidelity and quality in a typical simulation scenario of a passenger car.

## 6 Optimization – Analysis of the Real System and Environment

The pre-analysis of the system helps to determine the **influence of errors introduced by the coupling system**. The error depends strongly on the algorithms used on both sides, the simulation side and  $\mu\text{C}$  side, in order to process the exchanged data. First, the current chapter presents the analysis of algorithms executed on the  $\mu\text{C}$ . Afterwards, concepts for the analysis of the simulated environment are presented. These analysis techniques were partly presented on the *16th International Conference Mixed Design of Integrated Circuits and Systems* in the paper *Chip Hardware-In-The-Loop Simulation (CHILS) Coupling Optimization Through New Algorithm Analysis Technique* [KMH09].

### 6.1 Analysis of Algorithms

Algorithm analysis is a part of numerical mathematics. A classical problem of numerical mathematics is the **estimation of the error introduced by the numerical calculation**. Most of the standard books about numerical mathematics like [SW93] cover error estimation. The **numerical error analysis** calculates the relative or absolute output error depending on the relative or absolute input error. The relation between input and output error is called **condition number**. Algorithms are tolerant to input errors if the condition numbers are small. A problem with a low condition number is said to be well-conditioned, while a problem with a high condition number is said to be ill-conditioned. The condition number is a good indicator to prove the error tolerance of an algorithm. Based on that knowledge it is the basis of the following algorithm analysis. The condition number calculation is very computationally intensive because it does not only depends on the operation itself but also on all input variables. There is more than one condition number for an algorithm. It is a spectrum of condition numbers depending on a spectrum of inputs. The numerical basics are presented in subsection 6.1.2, including the definition of errors and the calculation of

condition numbers. The next section presents related work on the topic of numerical algorithm analysis techniques.

### 6.1.1 Related Works

A numerical analysis normally focuses on a limited number of algorithms, because of the high complexity of calculations. The paper *A Comparison of Two Algorithms for Predicting the Condition Number* [HZ07] describes a method to predict the condition number of matrices by a combination of modified k-nearest neighbors (k-NN) algorithm and a Support Vector Machine Support Vector Machines (SVM). The condition number of matrices is a measure of stability or sensitivity of a matrix to numerical operations. It gives a bound of how accurate or inaccurate the solution of a given problem is if it is numerically solved. This is interesting for the numerical solving of differential equations in terms of stability analysis, but not for a general algorithm analysis. Marina Epelman and Robert M. Freund focus on the condition number complexity of an algorithm for resolving a conic linear system [EF97]. Such considerations for single algorithms are supposed to be very time-consuming and hard to solve.

In *Computational Graphs and Rounding Errors* [Bau74] computational graphs are used to calculate the error propagation of relative errors. The presented approach can be applied in general to all algorithms representable as computational graphs. The study of an algorithm is limited by the input data of the algorithm. Only the error propagation for a specific set of input data can be solved, but this is acceptable as long as the input range of data is known. A graph framework for computation idea is presented by Linnainmaa in 1976 to compute the Taylor series expansion of an accumulated rounding error with respect to the local rounding errors.

P. Y. Yalamov extends the idea in the article *Graphs and stability of algorithms* to study classes of algorithms. The work on graphs falls under the term of automatic error analysis, while the previously presented approaches are a combination of theoretical analysis and numerical experiments. Other approaches are summarized in the book *Accuracy and Stability of Numerical Algorithms* written by Nicholas J. Higham [Hig02a]. An automatic error analysis can be regarded as an optimization problem so direct search methods can be applied. Another idea is to perform the calculation based on intervals, which is called interval analysis. So the calculation directly includes the fuzziness of the variables. Affine arithmetic or affine analysis was invented to reduce the range explosion by calculat-

ing intervals. An application of affine analysis can be found in the paper *Floating-Point Error Analysis based on Affine Arithmetic* [FCR03].

Beyond the numerical analysis other methods are used to analyze programme structures. Often a combined Control Flow Graph (CFG)-Data Flow Graph (DFG) is used. For example, the applications are the design space exploration of SoC architectures described by Mario Schölzel in [Sch06a] and by Victor S. Lapinskii in [Lap01] or compiler techniques. A classical approach in compiler techniques is register allocation based on graph colouring, which can be found in [Bri92] and [BCT94]. This popular technique constructs a graph representing the constraints that the register allocator must preserve. The information is primarily taken from the data flow dependencies of the programme.

## 6.1.2 Numerical Basics

In [EMR87] the numerical stability is defined as follows:

**Definition 6.1.1.** *An algorithm applied to a numerical method is called stable, weak-stable or instable depending on whether an error admitted in the  $n$ -th step of calculation is decreasing, of the same order or growing in the following steps of exact calculation.*

*Ein Algorithmus für ein numerisches Verfahren heißt stabil, schwach stabil oder instabil, je nachdem ob ein im  $n$ -ten Rechenschritt zugelassener Rechnungsfehler bei exakter Rechnung in den Folgeschritten abnimmt, von gleicher Größenordnung bleibt oder anwächst.*

This definition describes the idea of analysis of the error propagation. An ideal calculation process is assumed which is disturbed by the introduced error. If the exact calculation leads to a fast growing error, the algorithm is not applicable to solve the numerical problem. The stability of an algorithm depends on the problem to solve. In *Accuracy and Stability of Numerical Algorithms* [Hig02b] Nicholas J. Higham mentions the Gram-Schmidt method as an example. The method is stable when used to solve the least square problem, but if it is applied to compute the orthonormal basis it can produce poor results.

### 6.1.2.1 Error Propagation

The analysis of the error, for example the input error of a programme, can be done by looking at the propagation of the absolute perturbation

$\Delta x = (\Delta x_1, \Delta x_2, \dots, \Delta x_n) \in \mathbb{R}^n$  of the argument  $x = (x_1, x_2, \dots, x_n)$  of a mapping  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  to the result  $f(x + \Delta x)$  [SW93]. If  $f$  is continuously differentiable then the absolute error is defined as:

$$|f(x + \Delta x) - f(x)| = \left| \sum_{j=1}^n \frac{\partial f}{\partial x_j} \Big|_{x+\tau\Delta x} (\Delta x_j) \right| \quad (6.1)$$

with  $\tau \in (0, 1)$

The factor of growths is primarily determined by the derivative. For the analysis of relative perturbation  $\varepsilon_k = \Delta x_k / x_k$  one can approximate that:

$$\left| \frac{f(x + \Delta x) - f(x)}{f(x)} \right| \sim \left| \sum_{j=1}^n \frac{\partial f}{\partial x_j} \Big|_{x+\tau\Delta x} \cdot \frac{\Delta x_j}{x_j} \cdot \frac{x_j}{f(x)} \right| \quad (6.2)$$

with  $\tau \in (0, 1)$

[SW93]

**Definition 6.1.2.** *The condition of a problem is defined as the worst case growth factor of the influence from the input error to the result error.*

The relative condition number  $c_k$  and the relative result error  $\varepsilon_y$  for a function  $y = f(x_1, x_2, \dots, x_n)$  with  $n \in \mathbb{N}$  and a relative perturbation of input arguments  $\varepsilon_k = \Delta x_k / x_k$  with  $k = 1, 2, \dots, n$  can be determined by [Ern07]:

$$\begin{aligned} \varepsilon_y &= \frac{f(x_1 + \Delta x_1, x_2 + \Delta x_2, \dots, x_n + \Delta x_n) - f(x_1, x_2, \dots, x_n)}{f(x_1, x_2, \dots, x_n)} \\ &\sim \sum_{k=1}^n c_k \varepsilon_k \end{aligned} \quad (6.3)$$

with

$$c_k = \frac{x_k}{f(x_1, x_2, \dots, x_n)} \frac{\partial f(x_1, x_2, \dots, x_n)}{\partial x_k}$$

### 6.1.3 Arithmetic Basic Operations

The condition number calculation for specific arithmetic operation is based on the numerical basics mentioned before. In [SW93], the development of the error for arithmetic basic operations is defined in the following way.

**Lemma 6.1.3.** *Let be  $x, y \in \mathbb{R} \setminus \{0\}$  and let be  $\circ$  one of arithmetic operations  $+$ ,  $\cdot$  and  $/$ . The relative errors  $\varepsilon_x = (\tilde{x} - x)/x$ ,  $\varepsilon_y = (\tilde{y} - y)/y$  and  $\varepsilon_\circ = (\circ(\tilde{x}, \tilde{y}) - \circ(x, y))/\circ(x, y)$  regarding two approximations  $\tilde{x}$  of  $x$  and  $\tilde{y}$  of  $y$  are described by*

$$\begin{aligned} \varepsilon_\circ &\doteq \varepsilon_x + \varepsilon_y && \text{for } \circ(x, y) = x \cdot y \\ \varepsilon_\circ &\doteq \varepsilon_x - \varepsilon_y && \text{for } \circ(x, y) = x/y \\ \varepsilon_\circ &\doteq \frac{x}{x+y} \varepsilon_x + \frac{y}{x+y} \varepsilon_y && \text{for } \circ(x, y) = x + y \neq 0 \end{aligned} \quad (6.4)$$

$\doteq$  means that products of errors can be ignored.

### 6.1.3.1 Combination of Arithmetic Basic Operations

The combination of the arithmetic basic operation is the basis of the condition number calculation of complete algorithms. An algorithm is normally defined by a sequence of basic operations. The following proof presents the equivalence of condition number calculation using the sequential calculation of basic operations and using one complex operation. The calculation is based on equation 6.3.

**Proof 6.1.4.** *Let  $f_1(x_1, x_2) = x_1 \circ_1 x_2$  be a function with the operator  $\circ_1$  and let  $f_2(y_1, x_3) = y_1 \circ_2 x_3$  be a function with the operator  $\circ_2$ . It is given that  $f_1(x_1, x_2) = y_1$  and  $f_2(y_1, x_3) = y_2$ . The function  $f_2$  gets the result of  $f_1$  as input. In addition  $f_3(x_1, x_2, x_3) = (x_1 \circ_1 x_2) \circ_2 x_3$  is a function which is equivalent to sequential computation of  $f_1$  and  $f_2$ , so  $f_3(x_1, x_2, x_3) = f_2(f_1(x_1, x_2), x_3)$  is true. The input error of  $x_n$  is given as  $\varepsilon_n$  with  $n \in \{1, 2, 3\}$ . The relative errors  $\varepsilon_{f_1}, \varepsilon_{f_2}, \varepsilon_{f_3}$  produced as a result of the functions  $f_1, f_2, f_3$  can be calculated as follows (see equation 6.3).*

$$\varepsilon_{f_1} = \frac{\partial f_1}{\partial x_1} \cdot \frac{x_1}{f_1(x_1, x_2)} \cdot \varepsilon_1 + \frac{\partial f_1}{\partial x_2} \cdot \frac{x_2}{f_1(x_1, x_2)} \cdot \varepsilon_2 \quad (6.5)$$

$$\varepsilon_{f_2} = \frac{\partial f_2}{\partial y_1} \cdot \frac{y_1}{f_2(y_1, x_3)} \cdot \varepsilon_{f_1} + \frac{\partial f_2}{\partial x_3} \cdot \frac{x_3}{f_2(y_1, x_3)} \cdot \varepsilon_3 \quad (6.6)$$

$$\varepsilon_{f_3} = \frac{\partial f_3}{\partial x_1} \cdot \frac{x_1}{f_3(x_1, x_2, x_3)} \cdot \varepsilon_1 + \frac{\partial f_3}{\partial x_2} \cdot \frac{x_2}{f_3(x_1, x_2, x_3)} \cdot \varepsilon_2 + \frac{\partial f_3}{\partial x_3} \cdot \frac{x_3}{f_3(x_1, x_2, x_3)} \cdot \varepsilon_3 \quad (6.7)$$

The next step is to form the partial derivative of  $f_3(x_1, x_2, x_3) = (x_1 \circ_1 x_2) \circ_2 x_3$ .  $x_1 \circ_1 x_2$  is the inner function which is equal to the function  $f_1$  with the result  $y_1$ .

The outer function is given by  $y_1 \circ_2 x_3$  which is equivalent to  $f_2$ . Applying the chain rule 6.12 the partial derivatives  $\frac{\partial f_3}{\partial x_1}$  and  $\frac{\partial f_3}{\partial x_2}$  are formed.

$$\begin{aligned}\frac{\partial f_3}{\partial x_1} &= \frac{\partial f_2}{\partial y_1} \cdot \frac{\partial f_1}{\partial x_1} \\ \frac{\partial f_3}{\partial x_2} &= \frac{\partial f_2}{\partial y_1} \cdot \frac{\partial f_1}{\partial x_2}\end{aligned}\quad (6.8)$$

Inserted in equation 6.7 the following expression is derived.

$$\varepsilon_{f3} = \frac{\partial f_2}{\partial y_1} \cdot \frac{\partial f_1}{\partial x_1} \cdot \frac{x_1}{f_3(x_1, x_2, x_3)} \cdot \varepsilon_1 + \frac{\partial f_2}{\partial y_1} \cdot \frac{\partial f_1}{\partial x_2} \cdot \frac{x_2}{f_3(x_1, x_2, x_3)} \cdot \varepsilon_2 + \frac{\partial f_3}{\partial x_3} \cdot \frac{x_3}{f_3(x_1, x_2, x_3)} \cdot \varepsilon_3 \quad (6.9)$$

Equation 6.5 is now inserted in equation 6.6.

$$\varepsilon_{f2} = \frac{\partial f_2}{\partial y_1} \cdot \frac{y_1}{f_2(y_1, x_3)} \cdot \left( \frac{\partial f_1}{\partial x_1} \cdot \frac{x_1}{f_1(x_1, x_2)} \cdot \varepsilon_1 + \frac{\partial f_1}{\partial x_2} \cdot \frac{x_2}{f_1(x_1, x_2)} \cdot \varepsilon_2 \right) + \frac{\partial f_2}{\partial x_3} \cdot \frac{x_3}{f_2(y_1, x_3)} \cdot \varepsilon_3 \quad (6.10)$$

Canceling this equation yields to the following result:

$$\varepsilon_{f2} = \frac{\partial f_2}{\partial y_1} \cdot \frac{x_1}{f_2(y_1, x_3)} \cdot \frac{\partial f_1}{\partial x_1} \cdot \varepsilon_1 + \frac{\partial f_2}{\partial y_1} \cdot \frac{x_2}{f_2(y_1, x_3)} \cdot \frac{\partial f_1}{\partial x_2} \cdot \varepsilon_2 + \frac{\partial f_2}{\partial x_3} \cdot \frac{x_3}{f_2(y_1, x_3)} \cdot \varepsilon_3 \quad (6.11)$$

Because of the equivalence of  $f_2(y_1, x_3)$  and  $f_3(x_1, x_2, x_3)$  the equivalence of  $\varepsilon_{f2}$  (6.11) and  $\varepsilon_{f3}$  (6.9) is proved.

**Remark 6.1.5.** Chain rule from [BS63]:

$$\begin{aligned}y &= f(u) \\ u &= \varphi(x) \\ \frac{dy}{dx} &= f'(u) \cdot \varphi'(x)\end{aligned}\quad (6.12)$$

### 6.1.4 Algorithm Analysis Process

The algorithm analysis process is based on the idea of using computational graphs to estimate the error propagation of an algorithm [Bau74]. This principle is comparable to an automatic error analysis. It is in the nature of the principle that the complexity of the calculations needed for such analysis is very high. The error propagation depends strongly on the input data. For each different input data set, the calculation has to be done again. The idea is to reduce the complexity of calculations of a single algorithm by precalculation of algorithm parts and result storage in a database. The



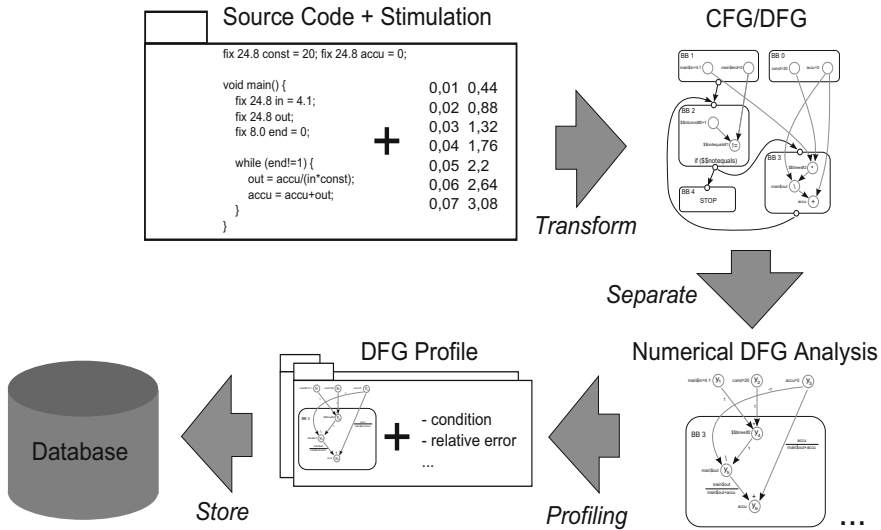


Figure 6.1: Profiling Flow

presented approach transforms the algorithm described in a programming language into a structure similar to a computational graph.

The analysis flow is set up as a bipartite process. The analysis starts with the profiling flow (figure 6.1) which is a training flow to build up a database of analysis results. The input contains the source code of an algorithm and input data sets. This part of the analysis is very costly in computation time.

The classification of algorithms is based on the results of the analysis. A new programme can be analyzed by classifying the graph of the programme and getting the corresponding results out of the database. The classification flow is described later in section 6.2.

#### 6.1.4.1 CFG-DFG Analysis

The computational graph is the starting point for the analysis approach. The application is transformed into a combined CFG-DFG, while the CFG is the primary structure. A DFG is similar to a computational graph used in [Bau74]. The numerical analysis, regarding the condition of the programme algorithms, can be executed on the DFGs. Each vertex of the CFG is called a basis block. A basis block contains a pure DFG, so it contains no control

flow information. The DFG is a set of operations with partial order. A directed acyclic graph can be used to create this order.

**Definition 6.1.6.** A DFG  $dfg = (V, E_i, type, var, cond, err)$  is a directed acyclic graph with a set of vertices  $V$ , the set of arcs  $E_i \subseteq V \times V$ , which are called internal arcs, and the labeling functions  $type : V \rightarrow O$ ,  $var : V \rightarrow \mathbb{R}$ ,  $cond : V \rightarrow \mathbb{R}$  and  $err : V \rightarrow \mathbb{R}$ . The set of arcs is defined as follows:  $E_i$  is a subset of the global set of data flow arcs  $E_D$  ( $E_i \subseteq E_D$ ). The set of operations is defined as:  $O = \{+, -, *, /\}$  is a set of unary and binary operations (see subsection 6.1.3).

The basic block is an extension of a DFG. In addition to the arcs, vertices and labels of the DFG there are external data flow arcs to model the data flow connections to other basic blocks or to the same block<sup>1</sup>.

**Definition 6.1.7.** A basic block  $b = (V, E_i, E_e, type, var, cond, err)$  is a directed graph with a set of vertices  $V$ , the set of arcs  $E_i \subseteq V \times V$  (called internal arcs), an additional set of arcs  $E_e \subseteq V \times V$ , which are called external arcs, and the labelling functions  $type : V \rightarrow O$ ,  $var : V \rightarrow \mathbb{R}$ ,  $cond : V \rightarrow \mathbb{R}$  and  $err : V \rightarrow \mathbb{R}$ . The sets of arcs are defined as follows:  $E_i$  and  $E_e$  are a subset of the global data flow arc set  $E_D$  ( $E_i \subseteq E_D$ ,  $E_e \subseteq E_D$ ). The basic block  $b$  is an element of the global set  $B$  ( $b \in B$ ).

The CFG models the control flow connections between the basic blocks.

**Definition 6.1.8.** A CFG  $cfg = (B, E_c)$  is a directed graph with a set of basic blocks  $B$  and the set of arcs  $E_c \subseteq B \times B$ .  $E_c$  is a subset of the global set  $E_C$ .

Additionally the following definitions are used.

**Definition 6.1.9.** The degree of a vertex (or node) of a graph is the number of edges incident to the vertex.

**Definition 6.1.10.** A source vertex (or node) is a vertex with indegree zero.

**Definition 6.1.11.** A sink vertex (or node) is a vertex with outdegree zero.

**Definition 6.1.12.** A vertex  $V_1$  is called a direct predecessor of a vertex  $V_2$  if  $(V_1, V_2) \in E$ .  $E$  is the set of arcs of a graph. The function  $pred : V \rightarrow \{V\}$  yields the direct predecessors of a vertex. If the vertex  $v$  has no direct predecessor  $pred(V)$  returns  $\emptyset$ .

**Definition 6.1.13.** A vertex  $V_1$  is called a predecessor of a vertex  $V_n$  if a sequence  $(V_1, V_2), \dots, (V_{n-1}, V_n) \in E$  exists.  $E$  is the set of arcs of a graph. The function  $preds : V \rightarrow V$  yields all predecessor of a vertex. If the vertex  $v$  has no predecessor  $suc(V)$  returns  $\emptyset$ .

---

1 Connections of a basic block to itself are cyclic connections.

**Definition 6.1.14.** A vertex  $V_1$  is called a direct successor of a vertex  $V_2$  if  $(V_2, V_1) \in E$ .  $E$  is set of arcs of a graph. The function  $\text{succ} : V \rightarrow \{V\}$  yields the direct successors of a vertex.

The input language of the analysis approach is a reduced C dialect, called C--. The C-- to CFG-DFG converter is based on the frontend of the compiler for the DESCOMP approach, which the author developed as seminar work for the PhD thesis of Mario Schölzel [Sch06a].

The programme (listing 6.1) is transformed to a CFG-DFG structure (figure 6.2). The dotted lines represent the control flow between the basic blocks while the solid lines show the data flow. The start nodes are labelled with the name and the value of the variable they represent. The other nodes are labeled with the internal variable names. The values of the internal variables are set while the calculation process on the graph is executed.

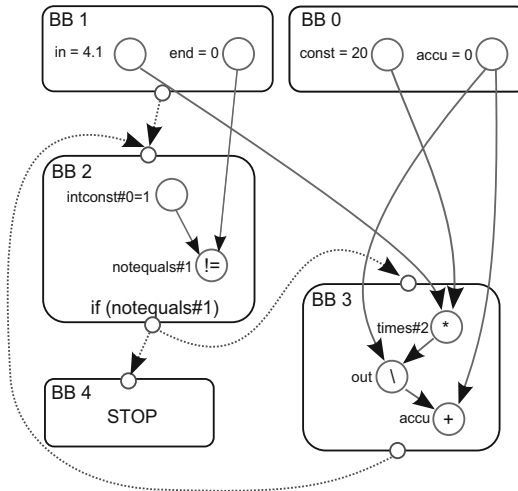


Figure 6.2: Example-CFG-DFG

It is possible to execute the programme flow based on the CFG. All control flows and data dependencies are available. The execution is necessary to calculate the values of the internal variables (figure 6.3). If all internal variables of a DFG are calculated, the condition of each algorithmic part of the programme can be calculated. The arcs are labeled with the condition

```

1 fix 24.8 const = 20; fix 24.8 accu = 0;
2
3 void main() {
4     fix 24.8 in = 4.1;
5     fix 24.8 out;
6     fix 8.0 end = 0;
7
8     while (end!=1) {
9         out = accu/(in*const);
10        accu = accu+out;
11    }
12 }

```

Listing 6.1: Example-C--Code

numbers of the single operations (figure 6.4). The calculation of operation results and condition numbers is an incremental process.

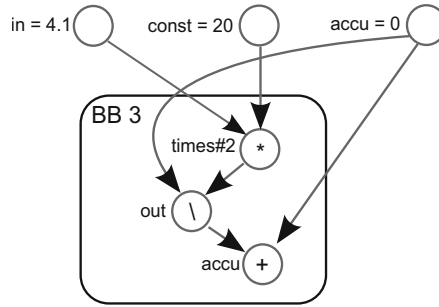


Figure 6.3: Example-DFG

In order to calculate the relative error  $\varepsilon_i$  all previous errors  $\varepsilon_j$  of the predecessor nodes have to be calculated. The calculation is described in lemma 6.1.3. For example the error  $\varepsilon_6$  in figure 6.4 is the result of

$$\varepsilon_6 = \varepsilon_5 \frac{out}{out + accu} + \varepsilon_3 \frac{accu}{out + accu} \quad (6.13)$$

It is clear that the labeling depends on the current state of execution, so the labeling functions *var*, *cond* and *err* are changing. It is necessary to create

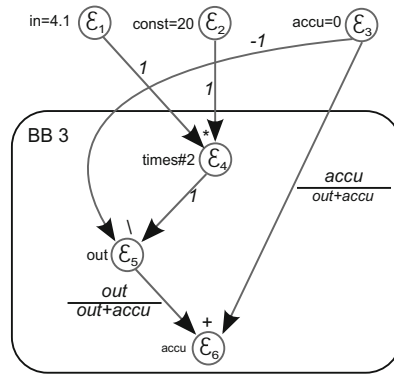


Figure 6.4: Example-DFG-Condition-Labels

a couple of execution scenarios to build the database. These scenarios are called a training set.

#### 6.1.4.2 Training Set

The classification database is filled with a training set of data. The set contains a couple of programs with a spectrum of input data. The previous calculation is done to create a spectrum of relative errors or condition numbers. The training programs are transformed to the CFG. The different execution paths in the CFG are mapped to a couple of DFGs. Every DFG represents one execution path. Afterwards these DFGs are subdivided into DFG subgraphs. Every subgraph has a minimum of three nodes, is connected, acyclic, and has one or more source vertices. A spectrum of input values and (relative) errors is applied to every subgraph. The result of the analysis, the condition number and the calculation results of the sink vertices, is stored into the database. The DFG is used as index into the database to retrieve the results. Each DFG is a binary tree. This is due to the fact that the operations are unary or binary. Based on that knowledge a unique signature can be easily applied to each DFG. Each signature can be inserted into a search tree to create a very efficient retrieval process. In order to build the signature we need first a numbering function for binary trees which adds a unique index to every node.

**Remark 6.1.15.** *The DFG has a natural order in the nodes so it is possible and necessary to differ between the left and the right successor of a node.*

**Definition 6.1.16.** A numbering function  $index : V \rightarrow \mathbb{N}$  is defined as

$$index(V) = \begin{cases} 1 & \text{if } pred(V) = \emptyset \\ 2 * n(pred(V)) & \text{if } V \text{ is the left successor of } pred(V) \\ 2 * n(pred(V)) + 1 & \text{if } V \text{ is the right successor of } pred(V) \end{cases} \quad (6.14)$$

Afterwards the signature function for binary tree DFGs can be defined as follows:

**Definition 6.1.17.** A signature  $sign(df\ g)$  of a binary tree DFG can be defined as a list of operator types  $o$  with  $O = \{+, -, *, /\}$  which is unique for each binary tree DFG. The list is defined as a set of tuples  $(o, n)$  with  $o = type(v)$  as operator type of the DFG node  $v$  and  $n = index(v)$  as index of the node.

$$sign(df\ g) = \{(type(v), index(v)) | v \in V\} \quad (6.15)$$

### 6.1.4.3 Complexity of Subgraph Training Set Calculation

The number of calculations based on one training set with  $n \in \mathbb{N}$  different input grows exponentially. Each binary graph has  $k$ -inputs with  $k \in \mathbb{N}$  and one output. Each of the  $n$  input values is applied to each input. From the combinatorics it is known that the number of variations with repetition in  $k$  places from a set with  $n$  elements is  $n^k$ . So  $n^k$  calculations are caused and  $n^k$  result sets are produced by an input data set with  $n$  values.

## 6.2 Classification of Algorithms

The classification of algorithms depends strongly on the goal to be achieved by the classification. In [scr] the four types of classifications are mentioned, **classification by purpose**, **by implementation**, **by design paradigm** and **by complexity**. The **classification by purpose** forms groups of algorithms depending on the task of the algorithm, for example search algorithms, cryptographic algorithms or compression algorithms. The **classification by implementation** differs between the basic implementation principles of algorithms. These basic principles are for example recursive or iterative, logical or procedural, serial or parallel and deterministic or non-deterministic. The **design paradigm** does not describe the implementation of an algorithm, but the kind of problem solving. Simple design paradigms

are “divide and conquer”, reducing the problem by creating smaller instances, which are easier to solve, linear programming, express the problem as a set of linear inequalities and minimize the inputs, or using graphs. The **classification by complexity** depends on the time to complete the problem depending on the size of the input. The algorithm can be linear in time, exponential in time or above exponential. Another possibility is to **classify algorithms by the structure** of an algorithm. The idea is that similar structured algorithms have similar characteristics. The structure is given by the data flow and control flow.

### 6.2.1 Related Work

Classification of graphs or using graph models to classify documents are actual domains of research. Especially automatic classification technologies become more and more important due to the exponential growth of data <sup>2</sup>. For example data-mining<sup>3</sup> uses classification techniques to transform this data into extracted information. A large amount of literature to this topic can be found.

One of the most import classification algorithms is the k-NN. The k-NN is a method for classifying objects based on closest training examples in the feature space [Wikb]. It is a type of instance-based learning. In *Classification of Web Documents Using a Graph Model* Adam Schenker, Mark Last, Horst Bunke and Abraham Kandel use k-NN algorithms to classify documents which are represented as graphs [SLBK03]. Classical techniques use numeric vectors in the feature space to classify the documents. The presented approach works with graphs as a fundamental data structure. The web documents are converted to graphs containing the most frequently occurring words of the document and the relations between them. A graph-theoretical distance measure is applied to calculate the distance needed by the k-NN method.

Kernel methods are another solution for graph classification. The function giving the inner products, for example a scalar quantity dealing with vectors, is called the kernel. So the kernel reduces a high dimensional problem into a low dimensional one. A learning machine does not access the high dimensional feature vector of a representative, only the low dimensional inner product is accessed. Hisashi Kashima and Akihiro Inokuchi present the method in *Kernels for Graph Classification* [KI02]. The design of

---

2 The amount of data doubling every three years [LV03].

3 Data mining is the process of uncovering and extracting patterns from large data sets.

the feature space and the kernel function is the critical task. Guido Del Vescovo and Antonello Rizzi present an approach called *Automatic Classification of Graphs by Symbolic Histograms* [VR07] in 2007. The graph is converted into a vector signature consisting of so-called symbols. This signature or feature vector is called a symbolic histogram. Each symbol represents a cluster of subgraphs of equal size. This technique can be applied for non-exact matchings.

## 6.2.2 Classification Process

The classification of algorithms is based on the results of the analysis presented in the previous section. A new programme is analyzed by classifying the graph of the program and getting the corresponding results out of the database. For the retrieval process, the graph is converted to a couple of signatures (see definition 6.1.4.2). Unlike to the symbolic histogram in [VR07], the signature is used for exact matchings. Figure 6.5 shows the different steps of the classification flow. The application is transformed into a combined CFG - DFG structure. Afterwards single DFGs with one end node are created representing the different control flows of the application for each result. From the DFGs signatures and sub signatures are generated as a search profile for the database. This graph matching process is explained later. The search profile yields the numerical data from the algorithm analysis process. This data has to be recombined depending on the structure of the original graph.

### 6.2.2.1 Graph Matching

The graph matching maps the programme graph to graphs which are used as database indices. The problem to find a matching can be formulated as follows. A cover of the whole DFG with subgraphs from the database has to be found. Starting at the end node of a single DFG, the largest available binary subgraph is chosen from the database. All covered nodes are removed from the DFG, except for the start nodes of the subgraph. These start nodes are the new end nodes for the next database search (see figure 6.6). If the whole DFG is separated into binary subgraphs the algorithm terminates. A complete profile for the whole DFG is reconstructed by recombining the data of the profiles retrieved from the database by the binary subgraphs.



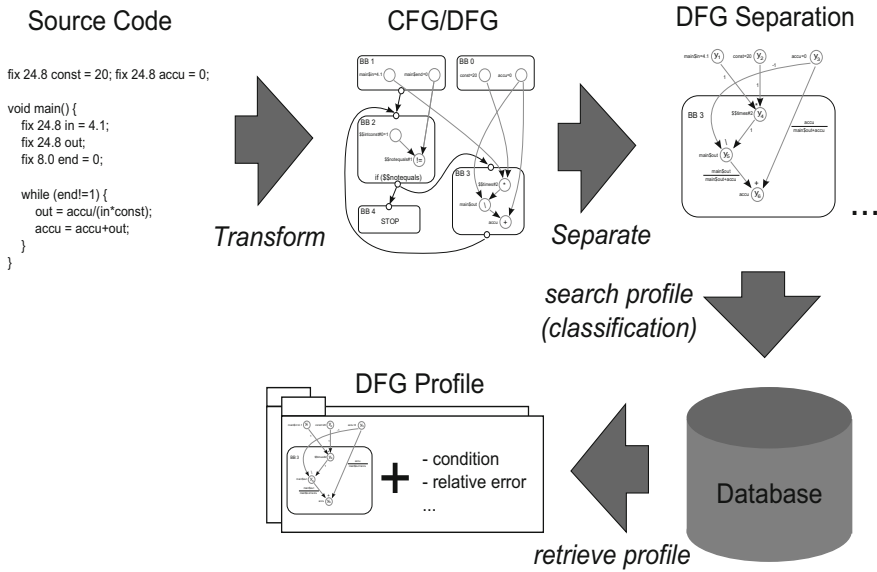


Figure 6.5: Classification Flow

Before the following description of the matching algorithm in pseudocode further down, some additional functions are introduced:

**Definition 6.2.1.**  $\text{subSign}(dfg, v)$  returns the signature of the largest binary subgraph of the DFG  $dfg = (V, E_i, \text{type}, \text{var}, \text{cond}, \text{err})$  starting from the node  $v \in V$ .

**Definition 6.2.2.**  $\text{cutSign}(s)$  returns a sub signature  $s'$  of signature  $s = \{(type(v), index(v)) | v \in V_s\}$ . The sub signature  $s'$  is retrieved by deleting the tuple  $(type(v_i), index(v_i))$  with the highest value of  $index(v_i)$  from  $s$ .

**Definition 6.2.3.**  $\text{getNodes}(s, v, V)$  returns a set  $V' \subseteq V$  of nodes which are corresponding to signature  $s$  starting from node  $v \in V$

**Algorithm 6.2.4.**

$dfg = (V, E_i, \text{type}, \text{var}, \text{cond}, \text{err})$ : the DFG

$DB_{\text{sign}}$ : set of signature of the DFGs stored in the database

$S_{\text{sign}}$ : set of signatures

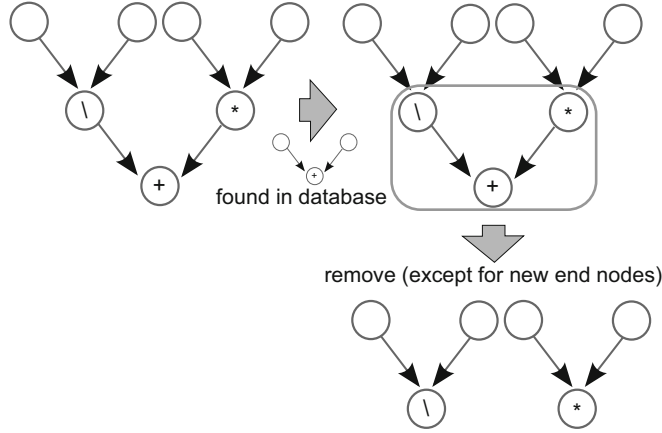


Figure 6.6: Graph Matching – Covering the DFG

```

while  $V \neq \emptyset$ 
  get  $v_e \in V$  with  $\text{succ}(v_e) == \emptyset$ 
   $s_{\text{sub}} = \text{subSign}(\text{dfg}, v_e)$ 
  while  $s_{\text{sub}} \notin \text{DB}_{\text{sign}}$ 
     $s_{\text{sub}} := \text{cutSign}(s_{\text{sub}})$ 
  end
   $V_{\text{clean}} = \text{getNode}(s_{\text{sub}}, v_e, V)$ 
  remove  $\forall v_r \in V_{\text{clean}}$  from  $V$  if  $\forall \text{succ}(v_r) \in V_{\text{clean}}$ 
end

```

**Remark 6.2.5.** The described algorithm will not find the optimal covering of the graph with given subgraphs. This problem is known as NP complete<sup>4</sup>, so the efficient implementations are currently unknown.

#### 6.2.2.2 Recombination

The recombination of the results is needed if the whole programme graph cannot be found within the database. The retrieved results from the sub

<sup>4</sup> NP ("nondeterministic polynomial time") is one of the most fundamental complexity classes. NP is the set of decision problems which are solvable in polynomial time by a non-deterministic Turing machine. NP-complete problems are a subset of the NP class. Currently, there are no polynomial-time algorithms known for NP-complete problems [Wikc].

graph analysis are combined to the result, which is valid for the whole graph. Of course only an approximation of the profile can be produced but the database retrieval and recombination process is much faster than the calculation of the numerical profile. The process of recombination is based on scaling and matching of the results from the database. The produced result is an upper bound of the real condition numbers.

**Proof 6.2.6.** *Graph A and graph B are two graphs to be combined into graph C (figure 6.7). The arcs of the graphs are labeled with the condition numbers  $c_n$  with  $n \in \{1, \dots, 4\}$  depending on the operations  $o_1$  and  $o_2$ . The errors  $\varepsilon_n \in \mathbb{R}^+$  with  $n \in \{1, \dots, 6\}$  are taken from the database while  $\bar{\varepsilon}_e$  is unknown. The preconditions are  $\varepsilon_5 = \varepsilon_4^5$  and  $\varepsilon_3 \geq \varepsilon_4$ .  $\bar{\varepsilon}_e$  can be calculated by multiplication of a factor  $d$  with  $\varepsilon_6$ . It is assumed that  $d = \frac{\varepsilon_3}{\varepsilon_5}$  fulfills this demand.*

$$\bar{\varepsilon}_e = \varepsilon_6 \frac{\varepsilon_3}{\varepsilon_5} \quad (6.16)$$

assuming that

$$\begin{aligned} \bar{\varepsilon}_e &\geq \varepsilon_e \\ \bar{\varepsilon}_e &= \frac{(\varepsilon_4 c_3 + \varepsilon_5 c_4) \varepsilon_3}{\varepsilon_5} \geq \varepsilon_4 c_3 + \varepsilon_3 c_4 \end{aligned}$$

and so

$$\begin{aligned} \varepsilon_3 c_4 + \frac{\varepsilon_4 c_3 \varepsilon_3}{\varepsilon_5} &\geq \varepsilon_4 c_3 + \varepsilon_3 c_4 & (6.17) \\ \frac{\varepsilon_4 c_3 \varepsilon_3}{\varepsilon_5} &\geq \varepsilon_4 c_3 \\ \frac{\varepsilon_3}{\varepsilon_5} &\geq 1 \\ \varepsilon_3 &\geq \varepsilon_5 \end{aligned}$$

So the assumption  $\bar{\varepsilon}_e \geq \varepsilon_e$  is true because  $\varepsilon_5 = \varepsilon_4$  and  $\varepsilon_3 \geq \varepsilon_4$ .

If  $\varepsilon_3 < \varepsilon_4$  the precalculated error  $\varepsilon_6$  is an upper bound of the real error  $\varepsilon_e$ .

### 6.2.2.3 Experimental Results

The actual implementation is very limited regarding the size of the database and the storage results. The input language is a subset of the C-language

---

5 The initial errors of the graphs in the database are identical.

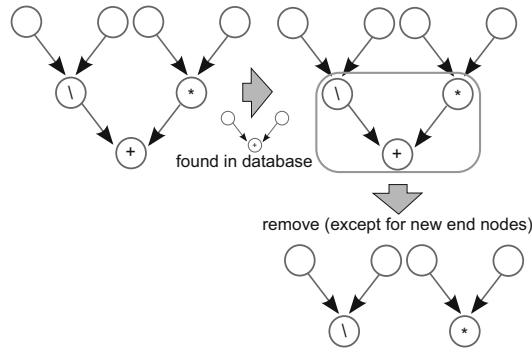


Figure 6.7: Result-Recombination

focusing on the algorithmic parts of programs. This is a proof-of-concept of an analysis framework to be developed. The example implementation is limited to storing of 100 000 data sets per analyzed subgraph, while during the analysis more data sets can be analyzed. An average value of a couple of sets is stored. The database is fed by some initial graphs and an initial analysis spectrum for the input variables. If the spectrum has  $n$ -Values and a graph has  $k$  input values  $n^k$  combinations are analyzed. The example spectrum has 92 values, so  $92^k$  values are analyzed while  $17^k$  result values are stored. Table 6.2 shows some results from the database filling process. An input with four input nodes and overall seven nodes needs about 800 seconds run time. The retrieval of the results (table 6.3) of the same graph takes less than 10 seconds. The run time for analysis increases exponentially to the graph size. The analysis of a graph with eight input nodes would take approximately several hours, while a combination of results for the same graph takes ten seconds (table 6.3, last line). Of course this result is less exact and just an upper bound of an analysis, but it is much faster to compute.

### 6.3 Analysis and Classification of Systems

The analysis and classification of systems is a well studied field. The following contemplations are taken from standard literature of control theory and system theory. Lutz Wendt's book *Taschenbuch der Regelungstechnik* (handbook of control technology) [Wen07b] and the book *Lineare Regelungs- und Steuerungstheorie* (linear control theory) are just two examples of a wide

<b>database size in MB</b>	284
<b>number of graphs in the database</b>	41
<b>result data sets in the database</b>	1 519 273

Table 6.1: Database Size

<b>Result Set Size</b>	<b>Run Time (sec)</b>	<b>Number of Input Nodes</b>	<b>Number Of Nodes</b>
289	0.109	2	3
4913	7.235	3	5
83521	793.277	4	7

Table 6.2: Time for Database Filling

<b>Number of Combinations</b>	<b>Run Time (sec)</b>	<b>Number of Input Nodes</b>	<b>Number of Nodes</b>
0	0.078	2	3
0	0.609	3	5
0	9.25	4	7
4	10.124	8	15

Table 6.3: Time for Database Retrieval

range of literature. Stability criteria of linear and nonlinear time invariant systems are presented. The example of the section 6.4 uses the criteria for the LTI systems defined below.

Figure 6.8 shows a simple example for stable and unstable systems. Only the system on the right hand side will return to the initial rest position if it is disturbed. The bowl shown in the system on the left and in the middle will leave its position if a force is applied.

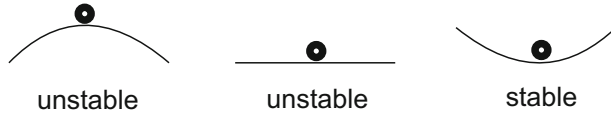


Figure 6.8: Rest Positions of a Mechanical System

### 6.3.1 Stability of Linear Time Invariant Systems

For LTI systems a complete consistent theory of stability exists. This theory can be applied to systems with linear or nearly linear behaviour around the operation point. The following definitions are done for SISO systems, similar considerations can be made for MIMO systems, too. A general definition for the stability of transfer functions of LTI systems can be found in [Rei06a]. A SISO LTI system is called stable if the system response of every limited input signal causes every time a limited output signal. This property is called BIBO-property (bounded input - BI, bounded output - BO). In a more formal way stability can be defined as follows:

**Definition 6.3.1.** *A LTI SISO system is called stable if the transfer function  $g(t)$  (time domain) is bounded for all  $t$  and*

$$h(t) \xrightarrow[t \rightarrow \infty]{} M \quad (6.18)$$

*$M$  has to be a finite limit. Otherwise the system is called instable.*

The next lemma gives criteria to determine the stability. The calculation is based on the system transfer function within the frequency domain.

**Lemma 6.3.2.** *A LTI SISO system with rational transfer function  $h(s) = \frac{y(s)}{x(s)}$  is stable if and only if all poles of the transfer function are located within the open left half plane<sup>6</sup> of complex plane  $\mathbb{C}$ <sup>7</sup>.*

The analysis can be done by calculating the zeros of the denominator polynomial. The calculation of the zeros is expensive for equations of higher order. But it is not necessary to calculate precisely the value of the zeros, it is just of interest if the real part is negative<sup>8</sup>. Standard literature like [Wen07c] differs between **algebraic** and **geometric** criteria. The

<sup>6</sup> The open half plane does not include the separating straight line.

<sup>7</sup> It is given that  $y(s)$  and  $x(s)$  are relatively prime, so their greatest common divisor is one.

<sup>8</sup> Being negativ is equivalent to being situated within the open left half plane of complex plane.

**algebraic** approach determines the stability based on the coefficients of the characteristic equation. The *Routh criterion* and the *Hurwitz criterion* are popular. The **geometric** criteria uses the locus of the characteristic equation, the so-called Nyquist plot, to decide about the stability. This is the *Nyquist* or *Strecker-Nyquist* criterion.

**Definition 6.3.3.** *The characteristic equation of a transfer function*

$$h(s) = \frac{y(s)}{x(s)} = \frac{b_0 + b_1s^1 + \cdots + b_ms^m}{a_0 + a_1s^1 + \cdots + a_ms^m} \quad (6.19)$$

is formed by setting the denominator polynomial to zero.

$$a_0 + a_1s^1 + \cdots + a_ms^m = 0 \quad (6.20)$$

### 6.3.2 Stability of Nonlinear Time Invariant Systems

The linearity principle (see definition 5.3.2) is not valid for nonlinear systems, that means that the superposition property or the homogeneity of the transfer function are not given. Nonlinear elements in control loops appear in different forms. **Analytic differentiable functions** like  $y = \sin(x)$   $y = x^4$   $y = e^x$  are a possible form. Differentiable functions have a Taylor series, so a linearization around the working point is possible. The analysis can be done with techniques for linear systems. **Piecewise linear functions** are not differentiable everywhere. At the transition points of the function pieces the function or the derivative can be discontinuous. The signum function  $\text{sign}(x)$  is a typical piecewise linear function.

$$\text{sign}(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0 \end{cases} \quad (6.21)$$

Linear system analysis can be applied for the different function parts and combined for the whole system. In **ambivalent functions** for every co-domain value multiply values in the function range can exist. This is typical for system elements with memory. The next element state depends on the previous states.

Nonlinear system elements can be classified by the attributes in table 6.4 [Wen07a].

<b>Differentiability</b>	analytic, continuously differentiable	partial differentiable
<b>Function Graph</b>	continues	discontinuous
<b>Function Derivative</b>	continues	discontinuous
<b>Characteristic Curve</b>	unique	ambivalent
<b>Function (Characteristic Curve)</b>	even (symmetric)	odd (skew symmetric)
<b>Time Dependency</b>	static (without energy store)	dynamic (with energy store)

Table 6.4: Attributes of Nonlinear Elements [Wen07a]

For LTI systems a complete consistent theory exists, but this is not the case for nonlinear systems. An exact method would use nonlinear differential equations. In fact it is still a problem to solve nonlinear differential equations, so an analytic solution is often not available [Wen07d]. As alternative **linearization** is used to analyze nonlinear systems. The nonlinear system is approximated by a linear system. This contemplation is limited by boundary conditions and preconditions. The results are not exact. In [Wen07d] different variations of **linearization** are presented.

**Linearization with Elements with Inverse Characteristic Curve** The nonlinear element is compensated by adding another nonlinear element into the path which has an inverse characteristic curve.

**Linearization with Feedback** The output of the nonlinear element is lead back to the input and is compared with the input. The nonlinear behaviour is reduced.

**Linearization within the Working Point by Cutting Higher Order Derivatives of the Taylor Series** The *Taylor Series* of the transfer function is created, but only the linear term of the first derivative is used.

**Linearization by Cutting Higher Order Derivatives of the Fourier Series** The principle of frequency analysis in linear systems is adapted to nonlinear



systems. A harmonic input signal, for example a sinus function, causes a signal response which is described as a *Fourier Series*.

Beyond linearization, that uses finite linear methods for stability analysis, other methods were developed. The methods to acquire information about a nonlinear system differ from system to system and from research criteria to research criteria. In the book *Taschenbuch der Regelungstechnik* [Wen07d] the following methods for stability analysis are explained:

**Phase Space Analysis** A graphical representation can be used to judge the behaviour of II. order systems. The precondition is that the function is not directly time dependent and has the form  $\ddot{x} = f(x, \dot{x}, u)$ . Using state variables  $x_1$  and  $x_2$  the system is transformed into a I. order system  $\dot{x}_1 = x_2$ ,  $\dot{x}_2 = f(x_1, x_2, u)$ . The solution is printed as phase space of  $\dot{x}_2 = f(x_1)$ .

**Ljapunow's direct method** *Ljapunow's direct method* can be applied for linear and nonlinear dynamic systems. It determines the system stability without calculation of the differential equations. The method uses a general energy function and the progression of energy over time. The system is called asymptotic stable if the system energy decreases after a displacement from the rest position. The system energy has to be zero within the rest position.

**Criteria of Popow** With *Popow's Method* not only a single characteristic curve but a class of characteristic curves can be examined regarding stability criteria. The method can use the *Nyquist Plot* of the system function.

## 6.4 General Analysis

A general analysis of the whole system is possible by combining the presented methods. The stability of the whole in-the-loop system is given if the control loop is stable in addition to an error tolerant control algorithm which is executed on the  $\mu C$ .

### 6.4.1 Stability of Control Loops

A stability definition for the complete (closed) control loop can be found in Kurt Reinschke's book *Lineare Regelungs- und Steuerungstheorie* (Theory of Linear Control) [Rei06b].

Assumed that the SISO transfer functions of the control path and the controller are defined as

$$h(s) = \frac{y_h(s)}{x_h(s)} \text{ and } k(s) = \frac{y(s)_k}{x_k(s)} \quad (6.22)$$

the transfer function of the open control loop is

$$h_o(s) = h(s) \cdot k(s) = \frac{y_h(s)y_k(s)}{x_h(s)x_k(s)} \quad (6.23)$$

It is given that  $y_h(s)$  and  $x_h(s)$  are relatively prime, and  $y_k(s)$  and  $x_k(s)$  are relatively prime, too.

**Definition 6.4.1.** *The Closed-Loop Characteristic Polynomial (CLCP) of the SISO control loop*

$$h_o(s) = h(s) \cdot k(s) = \frac{y_h(s)y_k(s)}{x_h(s)x_k(s)} \quad (6.24)$$

is defined as

$$y_h(s)y_k(s) + y_k(s)y_k(s) \quad (6.25)$$

**Lemma 6.4.2.** *The SISO control loop is stable if its CLCP has no zeros within the closed right half plane of the complex plane of  $\mathbb{C}$ .*<sup>9</sup>

## 6.4.2 Example

The example is based on the CHILS-demonstrator from chapter 10 in subsection 10.2.1. The following system parts are analyzed:

- the control path
- the control - Proportional–Integral–Derivative (PID)-Algorithm analytical analysis
- the control loop
- the control - PID-Algorithm numerical analysis

---

<sup>9</sup> The closed half plane includes the separating straight line.

### 6.4.3 Control Path Analysis

For the control path analysis the transfer function in the frequency domain is used:

$$h(s) = \frac{1.345}{s^2 + 1.026s + 0.77} \quad (6.26)$$

Based on lemma 6.3.2 the poles of the transfer function are situated at  $s = -0.5130000000 - 0.7119206416i$  and  $s = -0.5130000000 + 0.7119206416i$  within the open left half plane of complex number plane. So the LTI system is stable.

### 6.4.4 Control Analysis – Analytical

The PID algorithm has the following transfer function:

$$k(s) = \frac{k_i + k_p s + k_d s^2}{s} \text{ with } k_i, k_p, k_d \in \mathbb{R} \quad (6.27)$$

Often an easy rule can be applied for this kind of transfer functions: if the numerator polynomial order  $m$  is bigger than the denominator polynomial order  $n$  the system is instable because at least one pole is located at  $\infty$  (see [Rei06a]).

The PID algorithm from chapter 10 subsection 10.2.1 has the following parameters:

- Integral gain  $k_i = 0.3$
- Proportional gain  $k_p = 0.5$
- Derivative gain  $k_d = 0$

Unfortunately the parameter  $k_d = 0$  is zero, so the rule cannot be applied. But we can calculate the poles of the transfer function. The transfer function has one pole at  $s = 0$ , so it is not stable.

### 6.4.5 Control Loop Analysis

Combining the PID algorithm transfer function  $k(s)$  with the control path transfer function  $h(s)$  we can analyze the whole control loop. According to lemma 6.4.2, the calculation of the zeros of CLCP

$$CLCP(s) = y_h(s)y_k(s) + y_k(s)y_k(s) \quad (6.28)$$

is feasible.

$$CLCP(s) = (s^2 + 1.026s + 0.77) \cdot (s) + (1.345) \cdot (0.3 + 0.5s) \quad (6.29)$$

The zeros of  $CLCP$  are  $s = -0.3465029413 - 1.044829570i$ ,  $s = -0.3465029413 + 1.044829570i$  and  $s = -0.3329941173$ . All of them are situated within the open left half plane of complex number plane, so the control loop is stable.

### 6.4.6 Control Analysis – Numerical

The numerical analysis yields the error influence on the control algorithm implementation. Different implementation variations of the same algorithm can be compared. A spectrum of condition numbers retrieved from the result database is used to show the differences. The following example is based on the PID algorithm implementation used by the CHILS-demonstrator presented in chapter 10. The core algorithm of the PID controller is analyzed.

Listing 6.2 shows the original implementation of the PID core algorithm. Listings 6.3 to 6.5 are variations with small changes in line 18 and 19 of listing 6.2. The most critical operations are addition and subtraction (see subsection 6.1.3). An addition is critical if one operand is negative and the absolute value of both operands is nearly equal. The subtraction has a bad condition if both operands are nearly equal. The effect of small changes in the source code is mostly invisible.

```

1 /* PID-Algorithm Version 1 */
2 fix 24.6 Kp = 0.5;
3 fix 24.6 Ki = 0.3;
4 fix 24.6 Kd = 0;
5 fix 24.6 esum = 0;
6 fix 24.6 Ta = 0.005;
7 fix 24.6 ealt = 0;
8 fix 24.6 w_in;
9 fix 24.6 x_in;
10 fix 24.6 y_out;
11 fix 24.6 e = 0;
12
13 void main() {
14     fix 24.6 run = 1;

```

```

15         while (run) {
16             e = w_in-x_in;
17             esum = esum + e;
18             y_out = Kp*e + Ki*Ta*esum +
19                     (Kd/Ta)*(e-ealt);
20             ealt = e;
21     }}

```

Listing 6.2: PID Algorithm Version 1

Version 2 changes the position of the brackets. This causes an additional multiplication.

```

1  /* PID-Algorithm Version 2 */
2  ...
3  void main() {
4      while (run) {
5          e = w_in-x_in;
6          esum = esum + e;
7          /* line changed */
8          y_out = Kp*e + Ki*Ta*esum +
9                  (Kd*e - Kd*ealt)/Ta;
10         ealt = e;
11     }}

```

Listing 6.3: PID Algorithm Version 2

Version 3 adds an additional division.

```

1  /* PID-Algorithm Version 3 */
2  ...
3  void main() {
4      while (run) {
5          e = w_in-x_in;
6          esum = esum + e;
7          /* line changed */
8          y_out = Kp*e + Ki*Ta*esum +
9                  (Kd*e)/Ta - (Kd*ealt)/Ta;
10         ealt = e;
11     }}

```

Listing 6.4: PID Algorithm Version 3

Version 4 precalculates two constants to reduce the number of operations. This eliminates the division from the code snippet.

```

1 /* PID-Algorithm Version 4 */
2 ...
3 fix      24.6 KdTa = 0;
4 fix      24.6 KiTa = 0.0015;
5
6 void main() {
7     while (run) {
8         e = w_in-x_in;
9         esum = esum + e;
10        /* line changed */
11        y_out = Kp*e + KiTa*esum +
12              (KdTa)*(e-ealt);
13        ealt = e;
14    }}

```

Listing 6.5: PID Algorithm Version 4

The results in figure 6.9 and 6.10<sup>10</sup> show that version 1 and 2 have only a few differences. Version 3 has a fluctuating behaviour while version 4 is more stable. The first two algorithm implementations yield the best stability behaviour within the analyzed data range.

**Remark 6.4.3.** *The basis of analyzed data is very limited at the moment. For a deeper analysis the analysis database has to be extended. Furthermore, it makes sense to find a suitable limit of the input value range depending on the application. Other numerical criteria can be analyzed in future implementations. An option is to use affine arithmetic for algorithm analysis (see subsection 6.1.1 in chapter 6).*

## 6.5 Summary

The over-all analysis of the real system and the system environment consists of the analysis and the classification of algorithms executed on a  $\mu\text{C}$  and of the general system analysis. Algorithm analysis as a part of numerical mathematics estimates the error introduced by the numerical calculation. The used algorithm analysis flow is a bipartite process. It starts with the

<sup>10</sup> Figure 6.10 is an extract of figure 6.9. The highest result values are cut off to show the differences between the PID versions.

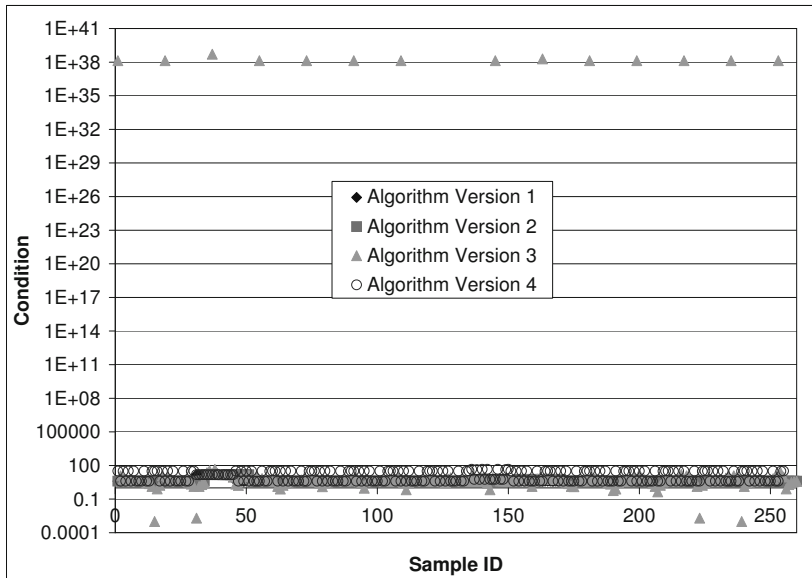


Figure 6.9: Numerical PID Control Analysis Results

profiling flow to build up a database of analysis results. The input contains a set of source code and set of input data. The application is transformed into a kind of computational graph. The graph is a combination of control and data flow graphs. This analysis is very costly in computation time. The classification of algorithms is based on the results of the analysis. A new programme can be analyzed by classifying the graph of the programme and getting the corresponding results from the database. The results of the numerical analysis show the influence of errors on the programme.

The analysis and classification of systems is a well studied field, so the contemplations are taken from standard literature of control theory and system theory. The stability analysis of LTI systems is done by calculating the poles of the transfer function of the system. For nonlinear systems no complete consistent theory exists because an analytic solution of nonlinear differential equations is often not possible. As an alternative linearization is used to approximate the nonlinear system by a linear system.

A general analysis of the whole system is possible by combining the presented methods. The stability of the whole in-the-loop system is given

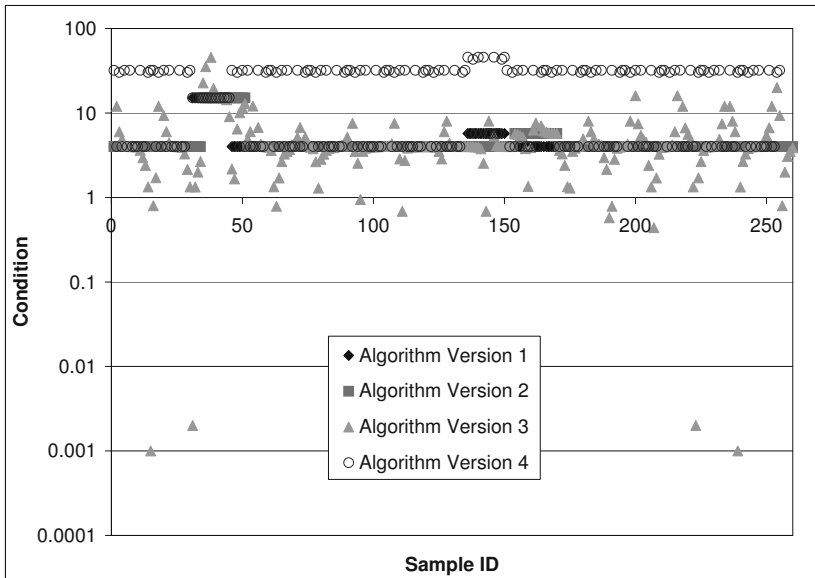


Figure 6.10: Numerical PID Control Analysis Results – Extract

if the control loop is stable in addition to an error tolerant control algorithm which is executed on the  $\mu\text{C}$ .



## 7 Optimization – Runtime Analysis

The runtime analysis, or analysis within the runtime, completes the previously presented analysis approaches. The pre-analysis yields to determine a suitable step size for data exchanges. However, the runtime information is needed to see if this choice is right. The runtime analysis primarily measures the rate of lost events and the delay of distributed events. Different metrics have to be introduced to evaluate this information. The analysis itself is implemented by the CHILS monitor. The CHILS monitor collects the event information and pre-evaluates it. Afterwards the CHILS device yields the evaluated information directly to the user or to the simulation, for example as criterion to adapt the simulation step size.

### 7.1 Metric Basics

Before a metric for runtime analysis can be defined, the target of the measurements have to be defined. We have to take into account what we have to measure and how the measurement is done. The influence of the measurements depends also on the target simulation, so different strategies are necessary if a continuous or a discrete simulation system is used. The following contemplations concern the runtime analysis of  $\mu C$  generated events. The CHILS event exchange mechanism, which has been presented in chapter 3 in section 3.4, will delay and can even result in loss of  $\mu C$  events. Basically, these lost or delayed  $\mu C$  events are measured for the runtime analysis.

#### 7.1.1 Events

In chapter 2 an event is defined as an indicator for a change (see definition 2.2.3.2). That means for our coupling system, that if the output of the simulation or of the  $\mu C$  changes, an event has to occur. Conversely, this means that if no change happens no event occurs. So the coupling is event-driven in terms of simulation. Two coupling quality influencing cases are possible. Events can get lost, so they are not distributed to the receiver,

or events can be delayed. The maximum delay time is limited by the exchange step size, so by the time between two exchanges. The loss of events is determined by the step size and by the interval between multiple events from the same source. Events get lost if this interval is smaller than the step size.

### 7.1.2 Simulation Coupling

Why should metrics for lost events be defined? Is it not always a knock out criterion? No, it is not. For example if the coupling is done with a continuous simulation environment and the input of the simulation is a quasi-continuous signal, like control or measurement values, lost signal changes reduce the accuracy but they normally do not drive the simulation into a false state. A metric for lost events makes sense in that case. On the other hand, if the simulation models a state-machine and a lost event leads to a false state, it is not acceptable to lose events.

## 7.2 Metric Definitions

For the following definitions it is assumed that the execution time of the user application running on the  $\mu C$  is finite.

### 7.2.1 Execution Time Accuracy

The **execution time accuracy** is based on the time the application needs to fulfill a certain task, so it is based on the runtime of a piece of code.

**Definition 7.2.1.** *The **execution time accuracy** is defined as a division of the runtime of a well defined piece of code within a simulated or emulated system and of the runtime of the same piece of code within the real target system.*

$$acc_{ex} = (1 - \frac{\|T_r - T_s\|_2}{T_r + T_s}) * 100\% \quad (7.1)$$

$T_r$  runtime within the real target system

$T_s$  runtime within the simulated/emulated system

$\|...\|_2$  euclidian norm

### 7.2.2 Event Distribution Accuracy

The relative event distribution accuracy is based on the rate of event losses. The highest accuracy is reached if all of the detected events were distributed to the simulation. The **rate of event losses** is defined for each event source  $s$ .

**Definition 7.2.2.** *The **relative rate of event losses** is defined as difference of all detected events and all lost events normalized by the number all events. A lost event is a detected event but it is not distributed to the event sink<sup>1</sup>.*

$$rt_{loss}(s) = \begin{cases} \frac{|E_a(s)| - |E_l(s)|}{|E_a(s)|} & \text{for } |E_a(s)| > 0 \\ 1 & \text{for } |E_a(s)| = 0 \end{cases} \quad (7.2)$$

with

$E_a(s)$  set of all detected events of the event source  $s$

$E_l(s)$  set of all lost events of the event source  $s$

$|\dots|$  cardinality of a set

In order to calculate the event distribution accuracy it is not necessary to store the exact value of an event. It is sufficient to gather the information of event appearance in relation to other events. The event appearances can be recorded in a **vector of changes**.

**Definition 7.2.3.** *A **vector of changes**  $\vec{c}(s)$  of the event source  $s$  is defined as a binary vector of variable length. The indices of the vector correspond to the index of the event detection step. If an event has been detected in step  $n$ , the **vector of changes** holds a one at index  $n$ . Otherwise the corresponding index is holds a zero.*

This definition requires an equidistant step size of the event detection mechanism.

A **measure of changes** of an event source can now be created by a piecewise linear function which corresponds to the binary vector of changes. The vector of changes is interpreted beginning from the lowest index. Each index is equivalent to a function piece of the length of one. If the vector holds a zero at the index  $n$ , the function is zero in the range of  $n + \epsilon$  to  $n + 1$  ( $\epsilon$  is the smallest number which is greater than zero). Otherwise if the vector holds a one, the function is continued with the function value at the position  $n$  and a slope of one in the range of  $n + \epsilon$  to  $n + 1$ .

---

1  $rt_{loss}(s) == 0$  means that all detected events are lost.

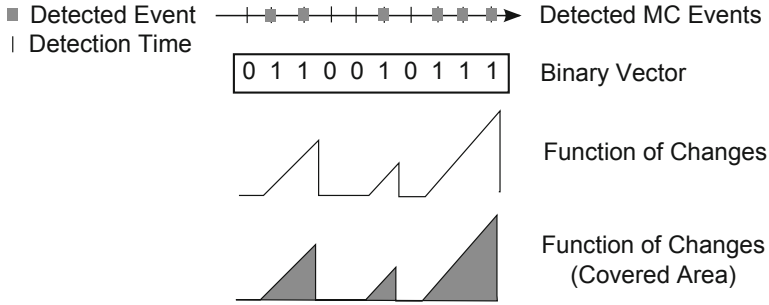


Figure 7.1: Measure of Changes of an Event Source

**Definition 7.2.4.** The *function of changes* is defined as follows:

$$fu_{ch}(i, \vec{ch}(s)) = \begin{cases} 0 & \text{for } i \leq 0 \\ 1 + fu_{ch}(i-1, \vec{ch}(s)) & \text{for } \vec{c}_i(s) = 1 \wedge |\vec{ch}(s)| \geq i > 0 \\ 0 & \text{for } \vec{c}_i(s) = 0 \wedge |\vec{ch}(s)| \geq i > 0 \\ 0 & \text{for } i > |\vec{ch}(s)| \end{cases} \quad (7.3)$$

with

$\vec{ch}(s)$  vector of changes of the event source  $s$

$fu_{ch}(i, \vec{ch}(s))$  function of changes  $s$

**Definition 7.2.5.** The *measure of changes* is now defined by the area covered by the *function of changes*.

$$mea_{ch}(\vec{ch}(s)) = \int_0^{|\vec{ch}(s)|} fu_{ch}(i, \vec{ch}(s)) di \quad (7.4)$$

$mea_{ch}(\vec{ch}(s))$  measure of changes of the event source  $s$

The advantage of this measure is that events which are detected directly one after another get a higher value than separately detected events. The **measure of changes** is now defined as an absolute measure. For further considerations a relative measure is needed to be independent from the steps of event detection. This measure is called **relative order of changes**.

**Definition 7.2.6.** The **relative grade of changes** is defined as a division of the measure of changes of the event source  $s$  and the highest possible value for the measure of changes of this source limited by the number of event detection runs.

$fu_{maxch}(i, \vec{ch}(s))$  function of the maximum of changes of the event source  $s$

$mea_{maxch}(s)$  measure of changes for the maximum of changes of the event source  $s$

$gd_{ch}(\vec{ch}(s))$  relative order of changes of the event source  $s$

$$fu_{maxch}(i, \vec{ch}(s)) = \begin{cases} 0 & \text{for } i \leq 0 \\ 1 + fu_{maxch}(i-1, \vec{ch}(s)) & \text{for } |\vec{ch}(s)| \geq i > 0 \\ 0 & \text{for } i > |\vec{ch}(s)| \end{cases} \quad (7.5)$$

$$mea_{maxch}(\vec{ch}(s)) = \int_0^{|\vec{ch}(s)|} fu_{maxch}(i, \vec{ch}(s)) di \quad (7.6)$$

$$gd_{ch}(\vec{ch}(s)) = mea_{ch}(\vec{ch}(s)) / mea_{maxch}(\vec{ch}(s)) \quad (7.7)$$

The **relative order of changes** reflects the quantity of the difference of a single event source between two exchange cycles and includes also the time component in the difference.

The **event distribution accuracy** can now be defined as **relative rate of event losses** (see definition 7.2.2) weighted by the **relative order of changes**.

**Definition 7.2.7.** The **event distribution accuracy** is defined as follows:

$$acc_{dis}(s, \vec{ch}(s)) = rt_{loss}(s) * (1 - gd_{ch}(\vec{ch}(s))) * 100\% \quad (7.8)$$

### 7.2.3 Event Occurrence Accuracy

The **event occurrence accuracy** is based on the time between the event detection by the monitor and the event distribution by the monitor. The highest **event occurrence accuracy** is given if the distribution happens right after the detection. The measured time between detection and distribution is generally not the exact delay of the event, because the CHILS monitor detects events only passively. The measured time is here called the **measured event delay**. The discussion about different event exchange mechanisms can be found in chapter 3 in section 3.3 *Exchange of Events between Microcontroller and Simulation* and in section 3.4 *CHILS Event Exchange Mechanism*.

**Definition 7.2.8.** The *measured event delay* is defined as the time between the detection of an event  $e$  and the distribution of this event.

$$d_{evt}(e) = t_{dis}(e) - t_{det}(e) \quad (7.9)$$

with

$t_{dis}(e)$  time of the distribution of the event  $e$

$t_{det}(e)$  time of the detection of the event  $e$

**Definition 7.2.9.** The *event occurrence accuracy* is defined by the difference between the sum of all event delays and the time between two exchange cycles.

$$acc_{occ} = \begin{cases} \left( \frac{|E_{dis}(s)| * t_r - \sum_{e \in E_{dis}(s)} (d_{evt}(e))}{|E_{dis}(s)| * t_r} \right) * 100\% & \text{for } |E_{dis}(s)| > 0 \\ 100\% & \text{for } |E_{dis}(s)| = 0 \end{cases} \quad (7.10)$$

$t_r$  time between two exchanges cycles of  $\mu C$  and simulation

$E_{dis}(s)$  number of all distributed events of the event source  $s$

## 7.3 Example

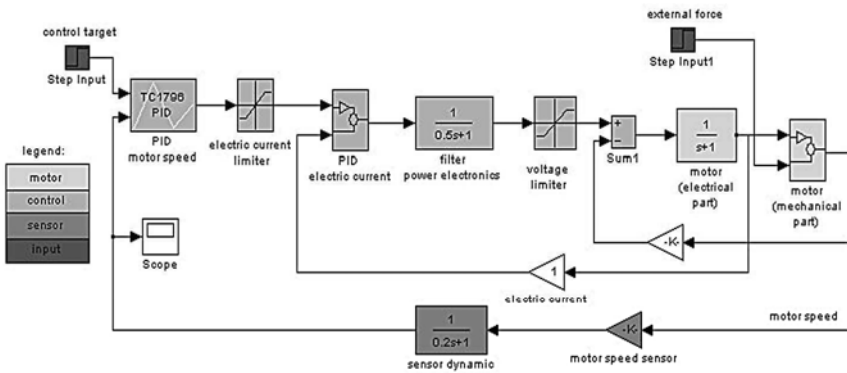


Figure 7.2: Electric Motor Control via PID Controller

An electric motor control has been modelled as a demonstration for runtime analysis and measurement. Figure 7.2 presents the MATLAB®/Simulink® model of an electric motor<sup>2</sup> which is controlled by an application on the TC1796ED  $\mu$ C. The application implements a PID control algorithm. The TC1796ED controls the motor speed depending on a step input, the setpoint of the system, and the actual motor speed measured by a speed sensor. An external force simulates a ramp which is set after 10s. The control output of the application, in this example the electric current, is a quasi-continuous signal.

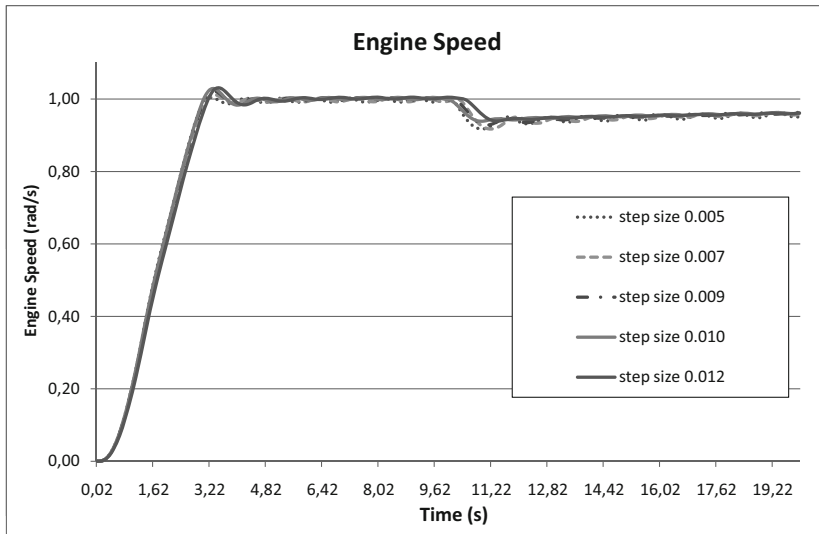


Figure 7.3: Results Electric Motor Control

In figure 7.3, the measured motor speed over a period of 20s is presented. Within 2s the motor speeds up to  $1\text{rad/s}$ , approximately 8s later the motor speed slows down to  $0.9\text{rad/s}$  caused by the external force. After that the motor speeds up again. The system was simulated with a simulation step size of 0.005s, 0.007s, 0.009s, 0.010s and 0.012s. The software PID controller

<sup>2</sup> This model is taken from [Hof98].

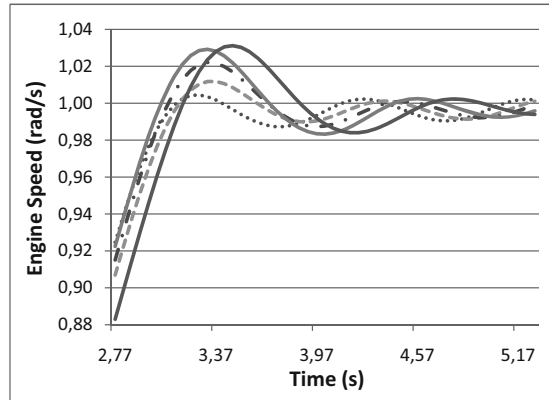


Figure 7.4: Results Electric Motor Control (Detailed Section)

on the TC1796ED has a cycle time of 0.005s. The event detection is set to a sampling rate of ten<sup>3</sup>.

Figure 7.3 shows that the overall difference between the step sizes is small. Only the enlarged section in figure 7.4 makes the differences visible. The maximal variance is about 5%.

Table 7.1 prints the **event distribution accuracy** and other measurement values from the different simulation step sizes. Only at the step size of 0.005s none of the detected events is lost. This corresponds to the expectations because of the cycle time of 0.005s of the PID algorithm. The **event distribution accuracy** decreases from 99.8% to 38.5%. This is caused by the high rate of lost events, which exceeds 60% at the step size of 0.012s. On the other hand, the **grade of change** is very small. This means that the value changes of the event source are very small between two exchange cycles, so the influence of the lost events is small. The similar results in figure 7.4 support this conclusion. The presented measurement values help the modeller to decide between performance and accuracy of the simulation.

## 7.4 Summary

The runtime analysis yields the rate of lost events and the delay of the distributed events to determine if a chosen step size for the data exchange

<sup>3</sup> If the step size is for example 0.005s, the monitor is activated every 0.0005s to detect changes.



Step Size	Grade of Change	Event Distribution Accuracy	Relative Rate of Event Losses	Detected Events per Exchange	Lost Events per Exchange
0.005	0.002	99.795 %	1	6.5	0.0
0.007	0.003	82.517 %	0.827	11.0	2.8
0.009	0.005	55.800 %	0.560	14.8	7.4
0.010	0.005	40.607 %	0.408	16.7	10.2
0.012	0.006	38.529 %	0.387	18.1	11.4

Table 7.1: Event Distribution Accuracy (Mean Values)

between the hardware and the simulation is correct. Lost events are not always critical for the hardware to simulation coupling. Depending on the signal type and simulation scenario, only the accuracy is reduced but the simulation is not driven into a false state. Especially if the signal is quasi-continuous, like a control or a measurement value, lost signal changes can be tolerated. The event detection is realized by oversampling. The CHILS monitor becomes active between the data exchanges of the simulation and the  $\mu C$ . In addition to the basic functionalities, the CHILS monitor captures the changes of the  $\mu C$  outputs and calculates the event distribution accuracy and the relative rate of event losses.

## 8 CHILS Framework – Concept

The current chapter introduces the overall CHILS framework. Implementation issues, the range of applications and the features of the framework are presented. The implementation realizes the previously discussed concepts of hardware to simulation coupling, interface implementation, and coupling analysis. The CHILS framework currently embeds a 32Bit Infineon TriCore®  $\mu$ C in the simulation environments SystemC and MATLAB®/Simulink®. The implementation supports the TC1796ED, TC1766ED, TC1767ED and the TC1797ED  $\mu$ C. The so-called emulation devices combine the unchanged standard  $\mu$ C and an additional emulation extension located on the same silicon die.

### 8.1 Functionality – CHILS Basics

The CHILS framework implementation uses the features of the emulation devices to implement the coupling system. The CHILS monitor is executed from the extended memory of the emulation extension, so no memory resources are lost for the user application except for a small part for the CHILS monitor call stack. The time stepping is done by the internal debug resources of the emulation devices. The Multi Core Debug System (MCDS) on the emulation devices provides a counter and trigger systems which extends the normal  $\mu$ C debug resources. The counters are used to start the CHILS monitor and to suspend the peripherals, while the monitor is active. Figure 8.1 shows a so-called EasyKit which offers a  $\mu$ C evaluation board combined with an onboard Wiggler to connect the board directly to the PC via Universal Serial Bus (USB).

#### 8.1.1 Data Exchange and Synchronization

The coupling between the  $\mu$ C and different simulation environments is the major topic of this work. A low hardware effort for coupling is one of the objectives of CHILS. The complete data exchange between the simulation and the  $\mu$ C is realized via the standard debugger interface, so the hardware

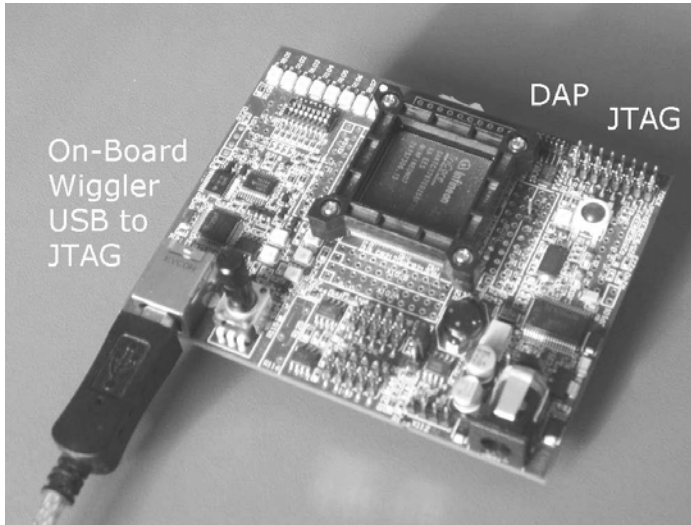


Figure 8.1: TC1767ED  $\mu$ C-EasyKit-Board

effort for the coupling is minimal in comparison to other HIL simulation approaches where every I/O pin itself has to be connected to the interface of the simulation computer.

The simulation and the device run in parallel as explained in chapter 3 in section 3.4. The data exchange is executed after a predefined time which is configured by the simulation and can be changed at every synchronization step. The  $\mu$ C and the simulation are waiting for each other for synchronization. Two applications are executed by the  $\mu$ C, first of all the user application, for example a control algorithm, and the CHILS monitor. As mentioned before the CHILS monitor controls the data exchange and the synchronization with the PC side (the CHILS device). It is non-intrusive to the user application so the coupling is nearly transparent to the user application. The CHILS monitor is a device specific application while the CHILS device is the same application for all supported devices.

The device is physically connected to the host computer via a Joint Test Action Group (JTAG) or a Device Access Port (DAP) interface (see figure 8.2 label 1). The Device Access Server (DAS) Server (2) manages the connection on the host computer (3). DAS abstracts the physical connection, for example a JTAG Wiggler connected via USB, to the  $\mu$ C. The CHILS device (4) is implemented as DAS client within a single Dynamic Link Library

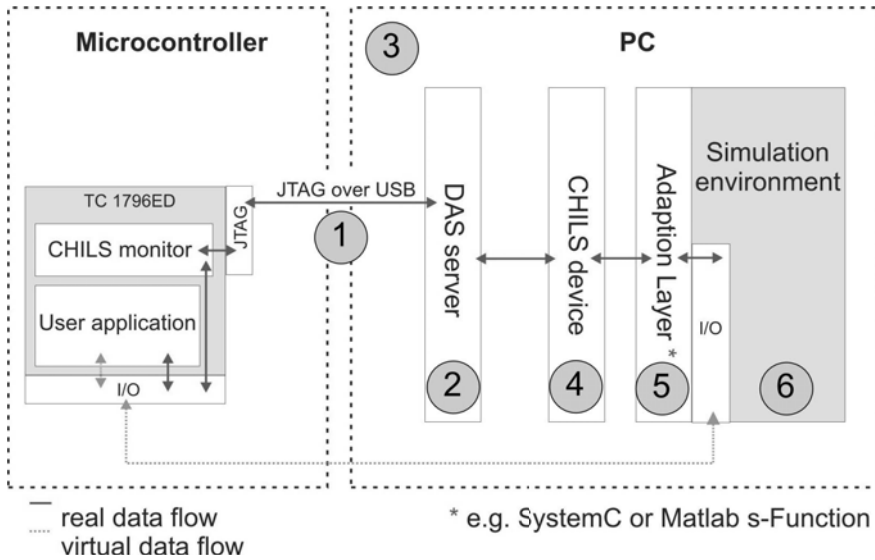


Figure 8.2: CHILS Coupling Implementation between  $\mu C$  and Simulation

(DLL). A thin adaption layer (5) realizes the connection to the simulation environment (6), for example MATLAB®/Simulink® or SystemC. CHILS offers a C/C++ interface that can be included into most environments. The interface is independent from the specific  $\mu C$ .

### 8.1.2 Interface Abstraction

The data exchange is realized on different levels of abstraction, which were defined in chapter 4 in section 4.2 *Interface modelling and Abstraction*. Four levels of interface abstraction are defined: the Message-Level for non-fixed size generic data types (for example CAN), the Byte-Level for primitive fixed size data types (for example ASC), Digital-Level for fixed bit vector data types (for example GPIO) and the Analogue-Level for analogue input values. The CHILS device interfaces are implemented on these different levels of abstraction, depending on the interface type. This approach reduces the amount of data shared between the  $\mu C$  and the simulation environment.

## 8.2 Microcontroller Requirements

The implementation of the CHILS approach has specific  $\mu\text{C}$  hardware demands. The most important demand is the possibility to run the CHILS monitor independently from the user application on the same  $\mu\text{C}$  hardware. On the TriCore® devices this is realized by the implemented **debug monitor** mechanism. A dedicated trap, the breakpoint trap, is designed to be used to enter a debug monitor even if the interrupts are disabled. This trap is raised on trigger conditions by the debug system. The multiple counters of the MCDS can be used as triggers, so an automatic context switch to the CHILS monitor after a predefined number of CPU cycles is possible. Such a counter system is necessary to generate the time steps between the exchanges.

A configurable suspend of the peripherals of the  $\mu\text{C}$  is also very important for the context switch between the monitor and the user application. The peripherals are supposed to be inactive while the CHILS monitor is active. The previously mentioned **Delayed Suspend** allows a soft suspend of the peripherals, so no read or write operations to the peripherals are lost.

A useful feature is an additional memory region. An available additional memory avoids the reduction of resources for the user application. The CHILS monitor has a size of about 50 to 75KByte. The set of data exchange structures and virtual registers, which were introduced in chapter 4 in section 4.2, require approximately 50KByte of memory. The emulation devices TC1766ED, TC1796ED, TC1767ED and TC1767ED have an extra memory of 256KByte to 512KByte which is used for debug traces or as overlay memory for software optimization. This **emulation memory** is used to host the CHILS monitor in this case.

### 8.2.1 Microcontroller Adaptations for Future Versions

The currently used TriCore®  $\mu\text{Cs}$  supports the most important features needed for the CHILS approach. Nevertheless further adaptations will allow to extend the support.

#### 8.2.1.1 Microcontroller Peripherals

The first idea concerns the  $\mu\text{C}$  peripherals. As explained in chapter 4 in section 4.2, two options exist to make the  $\mu\text{C}$  interfaces available for the user application via the CHILS approach. Option one is to use the **real**

**hardware registers.** This is currently only possible for the GPIO and for the GPTA which uses the GPIO ports. Option two is to use sets of **virtual registers** which are located in the emulation memory. Virtual registers are necessary if the real hardware registers are not accessible (especially the writable ones) from the CHILS monitor. The user application has to be adapted on driver level. The drivers of the application have to be changed if the application runs in a real scenario or in a simulated scenario via CHILS. For the normal application development supported by CHILS this is only a small drawback. But for driver development this poses problems. The requirement is to make all input and output registers of the peripherals readable and writable for the CHILS monitor. In addition, the reaction of a peripheral to write actions of the monitor has to be equivalent to the normal reaction on a new external input value. For example if a new value is written to the ADC peripheral capture register by the CHILS monitor, the ADC has to generate a trigger condition to write the value via Direct Memory Access (DMA) directly to the memory if this is configured by the user.

The implementation of **shadow registers** is an alternative for these input values. These special registers are only writable for the monitor and they emulate an external input. It is important that these registers are writable if the peripheral is suspended, in difference to the normal peripheral registers.

The generated peripheral output also needs to be transferred to the monitor. This is possible by **shadow registers** or by a DMA transfer to a special memory region. The peripheral has to write its output data, for example a CAN message, to the **shadow register** set or to a defined memory region, where the CHILS monitor can read it.

### 8.2.1.2 Monitor Context

The next improvement concerns the CHILS monitor context and the programme context. Currently only the most important registers are stored by the breakpoint trap mechanism. The so called lower and upper context, the general purpose register of the TriCore® and some other core registers, are saved manually. An automatic context save of all registers would be a desirable feature. Even more interesting is the range of the context save of the user application and the monitor, and the position of the heap and stack memory. This memory areas should also be located in the range of the additional memory.

## 8.3 Range of Applications

The CHILS approach is manifoldly applicable in the development of complex  $\mu$ C based systems. The essential advantage is the possibility to start the development of the software for the final system in a very early stage of development. If the system is only available as a high-level model on functional level, CHILS already allows the development of the control software on the real  $\mu$ C hardware. CHILS is positioned as advanced alternative to PIL solutions. In difference to PIL solutions, CHILS enables the developer to include the peripherals into the software development (see chapter 2 section 2.3 *Possibilities of Hardware-Simulation-Coupling*).

### 8.3.1 CHILS and the V-Model

The V-model is a project-management structure often used for the development of complex system. The origin of the V-model is located in guidelines for the IT software development project-management of the German government [V-M]. The V-model was established in 1986. At present, it is also used in other industrial sectors, for example for the hardware and the system development in the automotive, the communication or the IT industry. So different adapted versions of the V-model exist in parallel to the actual software development version V-Modell XT 1.3 [Rau09].

The advantage of the V-model, in contrast to models like the waterfall model, is that the phases of the design and the testing are connected. The knowledge of the design is used to generate test scenarios on the same level of detail. The classical V-model allows iterations in the vertical direction, so between different levels of abstraction. CHILS can be applied for system design and system testing, architecture design and architecture testing and for module design and unit testing (figure 8.3).

### 8.3.2 Rapid Control Prototyping

RCP is a methodology especially for control applications. The method combines and integrates older methods, for example the V-model, to reduce their disadvantages (see *Rapid Control Prototyping*, Univ.-Prof. Dr.-Ing. D. Abel [Abe03d]). The V-model does not allow horizontal iterations. RCP offers the possibility to run the system testing directly followed by the system design and to iterate between these points. RCP requires a closed tool chain to enable the iterations in vertical and horizontal direction. The

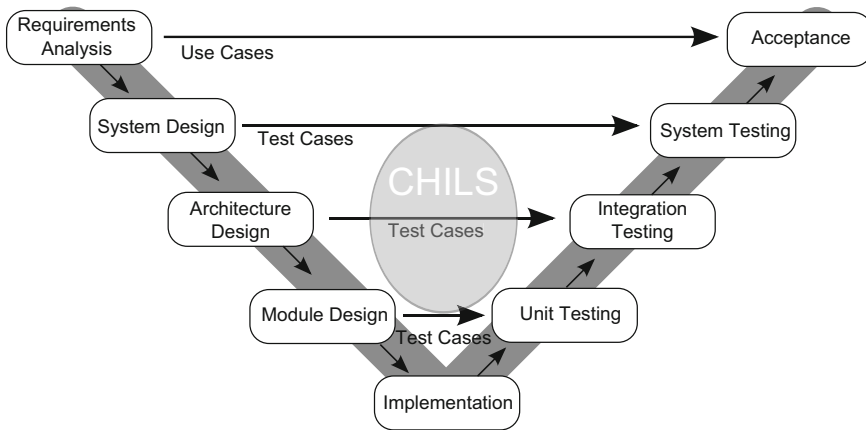


Figure 8.3: CHILS Application in the V-Model

CHILS approach can be integrated in such a tool chain to support RCP processes. An advantage of control software development with the help of CHILS was mentioned before. The software for the final system and the software for the system model can be nearly similar. So from control software side a change between the real  $\mu\text{C}$  environment and the virtual  $\mu\text{C}$  environment is easy to realize.

M. Deppe's, M. Robrecht's, M. Zanella's paper [DRZH01] introduces a prototyping environment for a complex mechatronic system. The so-called X-mobile is an autonomous experimental vehicle. The pure model representation was stepwise transformed to a real world system by replacing the simulated components with mechanical, electrical and hydraulic components. A similar, but smaller setup, was used to evaluate the CHILS approach. The results can be found in chapter 10 in subsection 10.2.1 *CHILS-Demonstrator*.

### 8.3.3 Test Applications

Testing is an important area of application for HiL systems as mentioned in chapter 2 in section 2.3 *Possibilities of Hardware-Simulation-Coupling*. CHILS can be used for automatical software tests, tests of critical scenarios or complex conditions. The virtual environment can be modified easily, so a real control can be tested in different control loops. A current disadvantage is that the tested software is not 100 percent identical to the final software



on the target system. For example the real drivers for some peripherals cannot be tested because CHILS needs some adaptations (see chapter 4 section 4.2 *Interface modelling and Abstraction*).

### 8.3.4 Estimation of the Real-Time Capability of Software with CHILS

CHILS can be used for early estimations of the real-time capability of software. The application can be profiled in an early stage of system development. For this purpose the environmental simulation does not need to be real-time capable. The test or profiling bench can be implemented on the PC side, for example in SystemC, while the  $\mu$ C is connected via CHILS.

## 8.4 Features

The CHILS framework offers a couple of features for integration into existing tooling environments and to support the user. The framework covers coupling capabilities with different simulation environments, different  $\mu$ C devices, debugger support and tools for simulation to hardware coupling optimization.

### 8.4.1 Simulation Coupling – CHILS-API

The CHILS-Application Programming Interface (API) encapsulates the CHILS device implementation. The interface can be used by any C/C++ application in a Microsoft Windows environment<sup>1</sup>. CHILS wrappers can be easily implemented for different simulation environments which support the integration of own C or C++ code. In appendix C *Listing* in listing C.1 the interface is printed. The following code snippets describe the start-up procedure.

1. Include the header-file

```
1 #include "das_ifx_devices_info.h"  
2 #include "dhil_device.h"
```

2. Create and init a CHILS device instance (here a TC1766ED instance)

---

<sup>1</sup> Tests were made under Windows 2000 and Windows XP.

```

1 FILE *logFile = stdout;
2 // Create CHILS device
3 CDhilDevice *dev = dhilCreateDevice(
4     DAS_DID0_IFX_JTAG_TC1766ED_B, logFile);
5 unsigned error = DHIL_ERR_NO_ERROR;
6 // Init CHILS device
7 error |= dev->setupInit();

```

### 3. Set application start address

```

1 // Set MC programme start address
2 dev->setApplStartAddress(0xD4000000);

```

### 4. Set used interfaces (for example ASC1)

```

1 // Setup ASC1
2 error |= dev->setupASCIn(1);
3 error |= dev->setupASCOOut(1);

```

### 5. Set-up device (the parameter is the number of TriCore cycles between to data exchanges)

```

1 error |= dev->setupDevice(cyclesPerExchange);

```

Now the device is ready for data exchange.

### 6. Set values

```

1 // send new value if previous value
2 // has been received
3 error |= dev->setASCInValue(1,&valueASCIn,
4     &valueReceived);

```

### 8. Exchange data between PC and MC

```

1 error |= dev->exchangeWithDevice();

```

### 9. Get values

```

1 // Get ASC1 value
2 error |= dev->getASCOOutValue(1,&valueASCOOut,
3     &valueTransmitted);

```

The time-steps are generated after each call of the function **exchangeWithDevice()**. The *get* and *set* methods of all defined  $\mu$ C interfaces are used to read or write the data to the device.

### 8.4.2 Device Coupling – DAS Architecture

The coupling to the physical device is realized via the Infineon DAS architecture (see also figure 8.2). The DAS architecture is a TCP/IP based client server architecture [DAS07]. The DAS server abstracts the physical connection to the  $\mu$ C. The DAS client can exchange data with the device over a TCP/IP connection to the DAS server. The CHILS device, as a DAS client, can connect to the different supported TriCore®  $\mu$ Cs. Multiple instances of the CHILS device can run in parallel to use more than one  $\mu$ C in a simulation.

### 8.4.3 Debugger Support – MCD-API

The Multi-Core Debug (MCD) API is a new debug and analysis interface [MCD09]. The API addresses the debugging of multi-core platforms. Adopting the interface on both tool- and target-side, debug and analysis solutions can be attached easily to the latest available SoC prototype, no matter if it is a virtual prototype or a real silicon. The DAS based Infineon MCD-API implementation was adopted to support CHILS. The abstraction of the debugger interface allows to hide the CHILS monitor while the debugger is active. So the user debugs the user programme and not the CHILS monitor programme. This is implemented by inhibiting the  $\mu$ C to stop within the monitor program range. The simulation and the debugger are connected in parallel to same device via DAS.

### 8.4.4 Tools for Coupling Optimization

Chapters 5 to 7 present different techniques for coupling optimization between hardware and simulation. The goal of the HIL simulation system analysis is an optimized setup for the data exchange between hardware and simulation.

#### 8.4.4.1 Coupling System Analysis

The coupling system analysis is based on a model of the coupling system as LTI system. MATLAB® scripts were written for the comparison of different coupling systems in section 5.5 in chapter 5. These scripts can be also adapted for other systems, if the system parameters are known.

### 8.4.4.2 Algorithm Analysis

Chapter 6 presents an algorithm analysis technique to determine the stability of algorithms executed on the  $\mu$ C. A command line tool, which is called Programme Graph Analysis (PGA), was implemented to realize the analysis. The current implementation is limited to small algorithm sizes and simplified structures of analyzable programs but it shows the value of such tools. The algorithm input has to be written in a simplified C dialect, called C--. The PGA uses an object oriented database with the capability to store some hundreds of thousand of analysis entries.

```

1 fix 8.0 x_1_in = 0;
2 fix 8.0 x_2_in = 0;
3 fix 8.0 y_out = 0;
4
5 void main() {
6     y_out = x_1_in + x_2_in;
7 }
```

Listing 8.1: C-- Example Listing

The short programme in listing 8.1 produces the following output. The analysis results are stored as tables in csv-files.

```

analyzing: syn_test_1_add.c--
parsing source file... done
building symbol tables... done
checking semantic... done
creating intermediate code... done
creating target code... done
// $$globalConstantInit -> BasicBlock 0
// main -> BasicBlock 1
BasicBlock 0 {
    node 0: (0) const ((bit 8:0, x_1_in))
    node 1: (0) const ((bit 8:0, x_2_in))
    node 2: (0) const ((bit 8:0, y_out))
    stop
}
BasicBlock 1 {
    node 0: ((0:0:0, bit 8:0, x_1_in),(0:1:0, bit 8:0, x_2_in)) +
        ((bit 8:0, y_out))
    stop
}
```

```

Open Database from G:/Park/Private_Park/CK/DB40/
... storage object retrieved from db ...
Retrieve Analysis Results...
Decomposition started...
... Decomposition finished
[+][const][const]
number of combinations: 0

```

```

running time for result retrieval: 157

```

```

running time: 219 result list size: 144 completed
with 0 errors and 0 warnings

```

#### 8.4.4.3 CHILS Runtime Profiling

The CHILS runtime profiling provides information about lost events and the event distribution accuracy which is explained in chapter 7. The data is stored in a log file. In addition to this, it can be retrieved from a CHILS-API function to implement an automatic step size control. A short extract from the log file is printed below. The log file also includes information about the runtime of the exchange process, the CHILS monitor idle time and the CHILS monitor status.

```

DHIL: Connecting over DAS
      Host computer address: localhost
      Device class ID: 0x100E2083, instance 0
JTAG frequency set to 10.000 MHz

DHIL: Device connected
DHIL: DHIL device monitor hex file loaded TC1796/main.hex
DHIL: DHIL device programme hex file loaded TC1796/prog.hex
DHIL: Device configured
DHIL: run cycles 20000
DHIL: mDeviceState->waitCycles = 1000
DHIL: mDeviceMonitorRunCycles = 245000
DHIL: ExchangeTime 0.001043
DHIL: mDeviceState->instr_id = 65534
DHIL: dhilLastInstrId = 65534
DHIL: mDeviceState->instruction = 94

```

```

DHIL: User Configuration Read Time 0.001670
DHIL: DetectedEvents 0
DHIL: LostEvents 0
DHIL: grade of change 0.000000
DHIL: event distribution accuracy 100.000000
DHIL: relative rate of event losses 0.000000
DHIL: measure of changes 0

```

#### 8.4.4.4 CHILS Simulation Step Size Calculation

The CHILS Step Size Calculator (CSSC) is a tool to calculate a suitable exchange step size based on the  $\mu$ C-CPU frequency, the frequency of the signal to sample, the sample frequency and the over sampling rate. It also predicts the simulation performance and the simulation accuracy based on previously taken measurements.

CPU freq (kHz)	step freq (kHz)	step size (cycles)	step size (ms)
150000	10	15000	0.1
signal freq to sample (kHz)		over sampling rate	
1		10	
sample freq changed to: 1 kHz			
step size calculation: 15000 cycles			
sample freq calculation: 10.0 kHz			
sample time calculation: 0.1 ms			
STM time accuracy appr.: 99.77334 %			
GPTA time accuracy appr.: 99.68 %			
sim perf appr.: 16071.429 kHz			

Figure 8.4: CHILS Step Size Calculator

## 8.5 Optimization Flow

The whole optimization flow contains the techniques which are presented in chapter 5, chapter 6 and in chapter 7. Figure 8.5 shows the three major steps of the flow. In step one the hardware to simulation coupling system is designed. The coupling system analysis approach in chapter 5 helps to

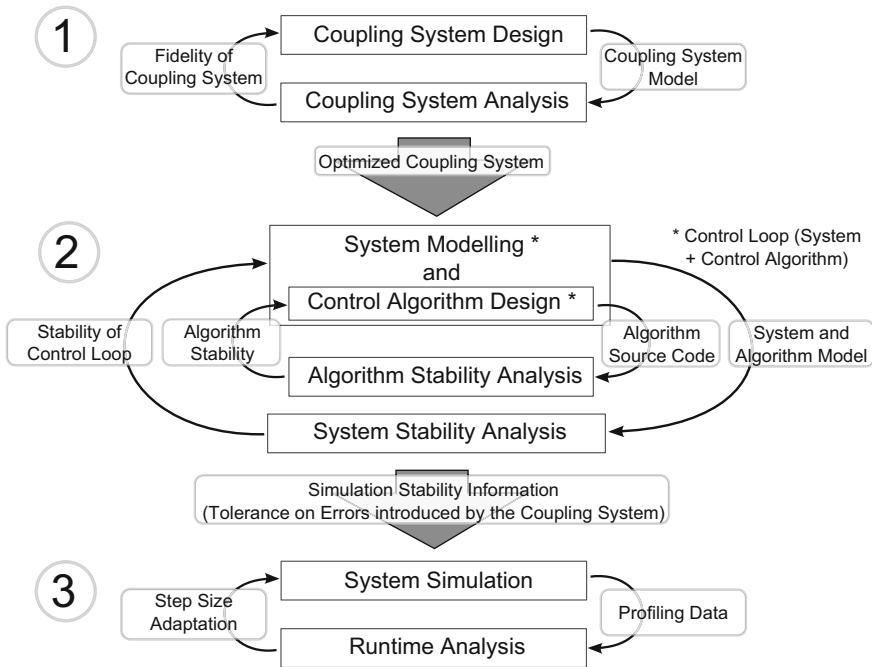


Figure 8.5: Optimization-Flow

find an optimized setup. The model of the coupling system is analyzed and as the result the fidelity value for the system is calculated.

Step two covers the system modelling and the algorithm design for the final hardware-software system. The system and the control algorithm form the control loop of the whole system. The algorithm analysis approach in chapter 6 is used to determine the stability of the algorithm implementation. In parallel, the stability of the whole control loop can be analyzed. System models and algorithm models are the basis for this analysis. As a result information regarding the stability of the whole system simulation are retrieved. These information show the tolerance on errors which are introduced by the coupling if the coupling is not ideal. The knowledge about the error tolerance can be used to speed up the simulation.

Step three presents the runtime analysis approach in chapter 7. The step size of the simulation can be adapted on the basis of the profiling information from previous simulation loops.

## 8.6 Summary

The CHILS framework embeds diverse TriCore®  $\mu$ Cs into different simulation environments. It covers coupling capabilities with different simulation environments, different  $\mu$ C devices, debugger support and tools for simulation to hardware coupling optimization. A device specific monitor application, the CHILS monitor, runs in parallel with the normal user application. The monitor execution is nearly transparent to the user application. The generic CHILS device is the counterpart of the CHILS monitor. It provides a C/C++ interface that can be used to program wrappers for every simulation environment which supports C or C++ modules. The physical data exchange is realized over the standard debug interface of the  $\mu$ C devices by the use of the DAS API.

The CHILS approach requires some specific features to be realized for a specific  $\mu$ C. The most important demands are an automatic switch between user application and monitor context, a non-destructive peripheral suspend feature, and additional memory to host the monitor and virtual register sets.

The CHILS approach is applicable in the development of complex  $\mu$ C based systems manifoldly. Applications are for example RCP and early software development. CHILS can be applied for system design and system testing, architecture design and architecture testing and for module design and unit testing in the context of a V-model based project-management.



## 9 CHILS Framework – Classification

The classification of the CHILS framework compares the framework with  $\mu$ C software models of different abstraction levels and hardware replacements. In this chapter the advantages and disadvantages of the different techniques are covered to underline the value of CHILS for the design process of complex hardware/software systems.

### 9.1 Software Models on Different Levels of Abstraction

For more than 20 years, software models of SoCs and  $\mu$ Cs are an evolving field of research. Different levels of abstraction are introduced, mostly to speedup the models. As mentioned in chapter 2, the available performance grows slower than the needs, caused by the rising complexity of such systems.

In [GJ00] Lovic Gauthier and Ahmed Amine Jerraya mention the following ‘classic’ methods to model a  $\mu$ C in software.

*Software simulations based on hardware description languages:* here we have for instance VHDL and Verilog. These languages allow to describe circuits from the behavioural level down to the gate level. These simulations have the advantage of being very precise and offering opportunity of visualizing the behaviour details of the circuit. However, these precise simulations of complex circuits such as microcontrollers are very slow (typically 5 to 10 instructions per second for a VHDL simulator).

*Software simulations based on instruction-set simulators:* ISS are programs that simulate the execution of the instructions of a processor. There are instruction-set simulators at different specification levels but most of the time they act at the instruction level. These simulators are much faster than the ones mentioned above (most of the time, they are compiled programs

and no longer interpreted descriptions). However, they are less precise than the first two simulations. Moreover, by definition they do not really simulate the circuit (they only simulate the instruction set). Electronic factors are thus not taken into account: input/output gates are not simulated, neither are the microcontroller peripherals. Even some cycletrue behaviour such as accesses to external buses may be difficult to handle. ([GJ00])

In their paper *Cycle-true simulation of the ST10 microcontroller including the core and the peripherals* [GJ00], they introduce a cycle accurate “C-model” of a  $\mu\text{C}$ . During the last years, additional methods occur. Especially the transactional level simulation becomes even more important as state-of-the-art technique in research and in industry (see for example [BBB<sup>+</sup>03], [Mar03], [FFP04] and [Ghe05]).

Abstraction Level	Typical Programming Languages	Explicit Concepts
<i>System Level</i>	MPI, Simulink	All Functional
<i>Virtual Architecture</i>	Untimed SystemC	+Abstract Resources
<i>Transaction Accurate</i>	TLM SystemC	+Resource Sharing and Control Strategies
<i>Virtual Prototype</i>	Cosimulation with ISS	+ISA and Detailed I/O Interrupts
<i>RTL</i>	HDL	+CPU Implementation and Reset Sequences

Table 9.1: Abstraction Levels [JBP06]

A general overview of different abstraction levels can be found in [JBP06]. The authors define five levels, **RTL**, **Virtual Prototype**, **Transaction Accurate**, **Virtual Architecture** and the **System Level** (see table 9.1).

Table 9.2 shows a more differentiated view on abstraction levels and modelling techniques which is chosen for the following comparison. The presented order does not necessarily imply a higher abstraction level.

Abstraction Level / Modelling Technique	Modelling Language or Environment (Example)
<i>Functional Model</i>	MATLAB®/Simulink®
<i>Instruction Accurate Model</i>	Instruction Set Simulator (ISS) as proprietary C/C++ programme
<i>Transactional Level without Time (Programmer's View without Time)</i>	un-timed SystemC
<i>Transactional Level with Time (Programmer's View with Time)</i>	timed SystemC
<i>Cycle Accurate Model</i>	'C model'
<i>Register Transfer Level (RTL)</i>	HDL like VHDL
<i>Physical Level</i>	SPICE

Table 9.2:  $\mu$ C Modelling Abstraction Levels and Techniques

### 9.1.1 Functional Simulation

A **Functional Model** describes the high-level functionality of a system. The separation of hardware and software is ignored on this level. **Functional Models** are often modelled in environments like MATLAB®/Simulink® or with languages like the Unified Modelling Language (UML).

### 9.1.2 Instruction Accurate Simulation

**Instruction Accurate Models** focus on the accurate simulation of the functional behaviour of the source code execution of  $\mu$ Cs. Especially an ISS primarily simulates the instruction set and skips the additional  $\mu$ C modules like bus systems and peripherals. An ISS is a simulation model which mimics the behaviour of a processor by 'reading' instructions and maintaining internal variables which represent the processor's registers. That means an ISS is a purely functional processor simulator. Complex processor internal behaviour like pipelining and caching is mostly abstracted. Infineon provides the TSIM[Tri02b] as ISS for the TriCore® architecture. **Instruction Accurate Models** and **TLM** models are not necessarily on different abstraction levels. For example an ISS is sometimes coupled with models of peripherals.

### 9.1.3 Transactional Level with/without Time

**TLM** is a high-level modelling approach where details of communication between modules are separated from the details of the implementation of functional units. **TLM** does not denote a single level of abstraction. It is a modelling technique. Moreover, we can differentiate between **Transactional Level with Time** and **Transactional Level without Time** depending on whether timing information is included in modelling or not.

In a report of CoWare [CoW06] four levels of abstraction for TLM modelling are mentioned. The **Functional View** represents an executable specification of the whole application. The **Architects View** targets at architectural exploration. The separation of the hardware and the software part of a system is often done on that level. The **Programmer's View** is a use case for embedded software design. The model shall be used as virtual prototype. The **Verification View** is used for cycle accurate system validation and HW/SW-co-verification. In this comparison the focus is on **Programmer's View** models.

**TLM** is strongly connected with SystemC. SystemC is not a dedicated programming language, it is a C++ based modelling library <sup>1</sup>. C/C++ allows different integrations of the software part. The software can be executed from a simulated memory region and interpreted by a simulated instruction set, or it can be executed as a native programme which is coupled to a hardware model.

### 9.1.4 Cycle Accurate Models

**Cycle Accurate Models** set the level of abstraction to the simulation of single cycles. They are often programmed as proprietary “C models”. **Cycle Accurate Models** and **RTL** allow an execution of  $\mu$ C software in the model.

### 9.1.5 Register Transfer Level

The **RTL** is the currently used level of Integrated Circuit (IC) design, before the mapping to the physical domain <sup>2</sup> is done. Description languages like VHDL or Verilog are used for the digital design. Language extension, for

---

<sup>1</sup> SystemC is also not coupled to a concrete level of abstraction.

<sup>2</sup> The mapping includes IC layout, routing and so on.

example VHDL-AMS, are used to model analog-mixed-signal designs. IC descriptions down to the gate level are possible.

### 9.1.6 Physical Level

The lowest level of abstraction is the **Physical Level**. The IC is modelled as a system of differential equations to cover physical effects. Attempts to integrate the simulation of quantum mechanics and molecular dynamics into a normal IC design flow are a field of research (see VHDL-AMS extension in [LCBF08]).

## 9.2 Embedding of Hardware in Simulations

Emulation and the established HIL simulation are described as concepts for embedding hardware in simulations in chapter 2 of this thesis. This section presents further details on the specific solutions, while a formal comparison of the CHILS coupling system and other HIL coupling systems is presented in chapter 5 in section 5.5 *Comparison of Different Coupling Systems*.

### 9.2.1 Original Microcontroller

Embedding a real  $\mu\text{C}$  into simulation environments can be implemented in two different ways. The first solution connects every  $\mu\text{C}$  pin via specialized hardware to the simulation computer. The other possibility is to use only a connection via a dedicated communication interface.

#### 9.2.1.1 Pin-Based Connection

The **SimPOD**-solution connects each I/O pin of the  $\mu\text{C}$  by the use of the DeskPOD™ hardware box. The company addresses the market of early silicon validation and HW/SW-Co-verification. The specialized hardware has more than 1200 user-programmable I/O [Des] and is able to exchange data between the  $\mu\text{C}$  and the simulation at each clock cycle. This is possible by controlling the  $\mu\text{C}$  clock by the external hardware box. SimPOD places the solution as provider of cycle-accurate 'models' of  $\mu\text{Cs}$ , which can be coupled to logic simulators, software debuggers, or other applications [Simb].

SimPOD announces a model performance of 250k cycles/sec. The simplest way to interface with the DeskPOD™ is through the ‘simulator’, the PC software controlling the DeskPOD™. In that case it behaves like a module and has a generated Verilog or VHDL shell. It can also interface directly through the Program Language Interface (PLI) functions or the software can link to an API.

SimPOD describes the functionality of the DeskPOD™ solution as follows. The DeskPOD™ has two modes of operation: the engaged and dis-engaged one. In the engaged mode the DeskPOD™ is under full control of the simulator and in that case the limiting factors are simulator speed, network round-time and processing time of the DeskPOD™ unit. In this mode processing speed is limited to few thousands of cycles per-second and probably no more than 7000. In the dis-engaged mode situation is more complex. In this case, DeskPOD™ is executing a cycle and checks for engage conditions or an engage request from the simulator. If conditions are met it sends the data to the simulator, otherwise it executes another cycle. DeskPOD™ processing speed can be as low as 15000MIPS and as high as 250000 MIPS depending on the way it is set up.

The next possibility is to use evaluation boards and connect the board I/O via DAQ cards to the simulation computer. In chapter 2 in subsection 2.3.2 *Commercial Hardware-in-the-Loop Solutions* several HIL solutions are mentioned, for example from dSPACE, ETAS, Visual Solutions and National Instruments. The solutions are very similar, so the dSPACE offers are used as an example. dSPACE [dsp] offers a wide variance of DAQ cards with digital and analogue inputs and outputs, which are used within their simulation computer racks. These solutions are mostly used to couple a complete ECU with a simulated environment. In this case the simulation and the simulator have to be real-time capable because the complete ECU cannot be slowed down. dSPACE provides classical HIL solutions.

The systems of a typical passenger car require sampling times of 1ms. A real-time simulation of a 6-cylinder gasoline engine including I/O with the *dSPACE Automotive Simulation Models Engine Simulation Package* can achieve cycle time of 0.15ms [dSp09]. The table 9.3 shows measured performance data of the dSPACE engine simulation package without the time for I/O operations. A simulation including the inner pressure of a cylinder can only be performed by the more powerful DS1006 system. The simulation demands a cycle time less than 1ms.

The dSPACE systems are primarily designed for HIL testing purposes and not for debugging of the control software. dSPACE does not deliver

	<b>DS1005 - 1GHz</b>	<b>DS1006 - 2.6GHz</b>
simulation cycle time		
<i>Engine Gasoline</i>	173 $\mu$ s	46 $\mu$ s
<i>Engine Gasoline + Turbo Charger</i>	212 $\mu$ s	52 $\mu$ s
<i>Engine Diesel</i>	250 $\mu$ s	64 $\mu$ s
<i>Engine Diesel + Turbo Charger + Diesel Exhaust</i>	772 $\mu$ s	194 $\mu$ s
<i>Internal Cylinder Pressure Diesel</i>	1516 $\mu$ s	360 $\mu$ s
<i>Internal Cylinder Pressure Diesel + Turbo Charger</i>	1543 $\mu$ s	381 $\mu$ s
<i>Internal Cylinder Pressure Diesel + Turbo Charger + Diesel Exhaust</i>	1551 $\mu$ s	389 $\mu$ s
<i>Internal Cylinder Pressure Gasoline</i>	1549 $\mu$ s	390 $\mu$ s
<i>Internal Cylinder Pressure Gasoline + Turbo Charger</i>	1562 $\mu$ s	393 $\mu$ s

Table 9.3: Performance of 6 Cylinder Engine Simulation with dSPACE

debugging tools but it is possible to start and stop remotely the simulation environment.

### 9.2.1.2 Embedding via Communication Interface

**PIL** solutions are using communication interfaces like the standard serial interface or debugger interfaces like JTAG. In [JLD<sup>+</sup>04] an approach is presented using the JTAG interface to connect an evaluation board with the Virtual Test Bed (VTB) simulation environment. PIL solutions embed just the computational capabilities of the  $\mu$ C, the CPU core, into the simulation. The peripherals of the device are not included, unlike CHILS. Commercial solution often provide an automatic code generation from control algo-

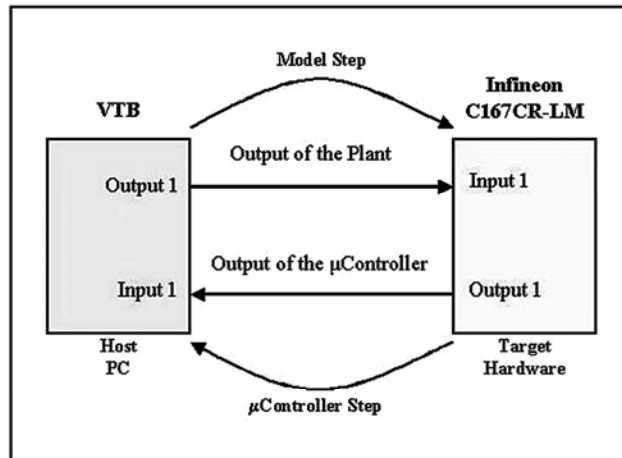


Figure 9.1: VTB-PIL Coupling from [Len04]

rithms in high level simulation environments like MATLAB®/Simulink® as mentioned in chapter 2 in subsection 2.3.2.

PIL solutions use special I/O functions. The simulation and the target device run alternately. The control algorithm on the target device, the  $\mu$ C on an evaluation board, is executed for one sample cycle. Afterwards the calculated output is sent to the simulation. Now the simulation steps forward and sends its data back to the target device.

In [Len04] Santiago Lentijo describes the PIL approach of the VTB as follows (see also figure 9.1).

During PIL simulation, the VTB solver manages the simulation of the plant dynamics by using a variable-time step approach as reported in. The sampling action of the control is managed as an event. The PIL model identifies when the event is going to happen and forces the VTB solver to move the simulation to the exact event time. When this occurs, VTB exports the output data (output of the plant) to the control system under test via a serial communications link. When the target processor receives signals from the plant model, it executes the controller code for one sample step. The controller returns its output signals (output of the controller) computed during this step to VTB, via the same communications link. At this point one sample cycle of the simulation is complete and the plant model proceeds to



the next sample interval. The process repeats and the simulation progresses. When the controller code is not executed the simulation is performed assuming a zero order hold action on the control output. ([Len04])

### 9.2.2 Emulation

Emulation is defined in chapter 2 in section 2.4 as a duplication of the functions of one system using a different system, so that the second system behaves like the first system. The focus is on the exact reproduction of external behaviour, so the unchanged binary code of the original system can be executed on the emulated system. Three version of emulators are mentioned. The classical emulator is a **processor based emulator** that contains between tens of thousands to hundreds of thousands of ALUs and registers [Tur05]. Caused by the enormous hardware effort, this kind of emulators are not popular any more.

**FPGA based emulators** map the SoC design to the configurable logic blocks of an FPGA. FPGAs are a flexible solution as long as the complete design can be mapped to one FPGA device. Otherwise the separation of the design and the mapping to more than one FPGA are a very complex tasks. FPGA boards are available as external stand-alone boards or as plug-in cards for a workstations or a PC. Plug-in cards have the advantage of a fast communication over the system bus to the simulation on the PC. Examples can be found in *Integrierte Simulation und Emulation eingebetteter Hardware/Software-Systeme*[Sch05], *Run-Time Reconfigurable Co-Emulation for Hardware-Assisted Functional Verification and Test of FPGA Designs* [Sir06] and *Accelerated Logic Simulation by Using Prototype Boards* [HSBG98].

**VLIW based emulators** use the computation power of VLIW processors for emulation tasks. VLIW processors provide a highly parallel architecture with multiple ALUs. RTL descriptions are translated into a VLIW programme [Hau01]. Also combinations of VLIWs and FPGAs can be found in literature. In his thesis *Zyklengenaue Binärkodeübersetzung für die schnelle Prototypenentwicklung von System-on-a-Chip* [Sch06b] Jürgen Schnerr presents such a system.

### 9.3 Comparison

The comparison of CHILS and the different hardware and software solutions covers diverse properties of the solutions (see table 9.4 and table 9.5). Absolute values for performance and accuracy are not mentioned due to the strong dependence on the simulated or emulated  $\mu\text{C}$  architecture. Some performance approximations are presented in chapter 2. Functional models of  $\mu\text{C}$  architecture can reach realtime performance or more than realtime performance. Figure 2.1 presents some performance data for instruction accurate and cycle accurate models from *Target Compiler Technologies* [ReT]. A cycle accurate ISS has a performance of about one MIPS. This is very fast for cycle accurate models, but this model is only an ISS without peripheral units and bus systems simulated. Less accurate models can achieve about 100 MIPS for example if the focus is set to instruction accuracy and not additional to timing accuracy. A complete  $\mu\text{C}$  modelled on that abstraction level can have a performance which is the tenth part of the ISS performance. RTL models run normally at a speed of less than thousand instructions per second. The Physical Level is skipped in the table, because it is not realistic to simulate complete  $\mu\text{C}$  architectures on that level.

As the comparison shows, the CHILS approach combines high performance and high accuracy with a low effort for simulation coupling in difference to other ‘in-the-loop’ solutions. The primary difference to PIL solutions is the availability of the peripherals for the user application on the  $\mu\text{C}$ . Software models are either fast or accurate but they provide the highest observability. Software hooks for debugging and tracing can be easily implemented in difference to hardware hooks.

Table 9.4: Comparison of CHILS with Different Hardware Solutions

Solution Properties	Hardware (Original)				Hardware (Emulation)	
	CHILS	PIL ( $\mu$ C Evaluation Board)	ECU-in-the-Loop	SimPOD $\mu$ C-in-the-Loop	FPGA	VLIW
Hardware Effort for Coupling With Simulation Environment	less <sup>3</sup>	less <sup>4</sup>	very high	very high <sup>5</sup>	less <sup>6</sup>	less <sup>7</sup>
Hardware Costs <sup>8</sup>	approx. 500Euro	approx. 500Euro	approx. 30,000Euro	approx. 70,000Euro	approx. 200 to 2000Euro	approx. 400Euro
Synchronization Frequency	every 'n' CPU cycles	after each algorithm evaluation	every 'n' CPU cycles	after each CPU cycle	after each CPU cycle	after each CPU cycle
Data Exchange Mechanism ( $\mu$ C Software View)	real or virtual HW-I/O Register	special IO-Functions	HW-I/O Register	HW-I/O Register	HW-I/O Register	HW-I/O Register
CPU + Memories Included	yes	yes	yes	yes	yes	yes
Peripherals Included	yes	no	yes	yes <sup>9</sup>	yes <sup>10</sup>	no
Coupling Transparent to the Application	yes	no	yes	yes	yes	yes

3 The standard debugger interface is used for the physical connection.  
4 A standard serial interface is used for the connection.  
5 Special hardware is needed to connected each of the  $\mu$ C pins to the simulation computer.  
6 A FPGA board as a plug-in device for the simulation computer uses standard interfaces like the PCI bus.  
7 The effort is less if the VLIW processor board is a plug-in device for the simulation computer.  
8 The hardware cost do not include the simulation computer.  
9 The connection of analogue peripherals causes problems (see subsection 5.5.4 in chapter 5).  
10 Peripherals can be included as components in the FPGA based emulation.

Table 9.4: Comparison of CHILS with Different Hardware Solutions (continued)

Solution Properties	Hardware (Original)				Hardware (Emulation)	
	CHILS	PIL ( $\mu$ C Evaluation Board)	ECU-in-the-Loop	SimPOD $\mu$ C-in-the-Loop	FPGA	VLIW
Application Reuse in Real System	complete	only algorithmic application part	complete	complete	complete	partly
Peripherals Usable in Application	yes	no	yes	yes	yes	no
Type of Application	complete application	just algorithmic part	complete application	complete application	complete application	application without peripheral usage
Simulation Performance	mid high <sup>11</sup>	high <sup>12</sup>	high <sup>13</sup>	low	high	high
All $\mu$ C Resources Available for Application	yes	no	yes	yes	yes	no
Time Accuracy	high	N.A.	very high	high	high	mid to high
Application Execution is Independent from Synchronization with the Simulation Environment	yes	no	yes	yes	yes	yes
Data Abstraction of the $\mu$ C Interfaces	yes	yes	no	no	yes	yes

<sup>11</sup> The performance depends on the exchange frequency between the hardware and the simulation.

<sup>12</sup> Only the algorithmic part of the software is included in the simulation setup.

<sup>13</sup> The simulation and the hardware run in real time.

Table 9.4: Comparison of CHILS with Different Hardware Solutions (continued)

Solution Properties	CHILS	Hardware (Original)			Hardware (Emulation)	
		PIL <i>Evaluation Board</i>	$\mu$ C <i>ECU-in-the-Loop</i>	SimPOD $\mu$ C- <i>in-the-Loop</i>	FPGA	VLIW
<i>Demands Environment Simulation in Real Time</i>	no	no	yes	no	no	no
<i>Observability</i>	mid <sup>14</sup>	low	low	mid <sup>15</sup>	mid <sup>16</sup>	low

<sup>14</sup> The connection of a debugger is supported.  
<sup>15</sup> The connection of a debugger is supported.  
<sup>16</sup> The connection of a debugger is supported.

Table 9.5: Comparison of CHILS with Different Software Solutions

Solution Properties	Software Models						Instruction Set Simulator	Functional Level
	RTL Model	Cycle Accurate Model	TLM Model Program-mer's View with Time (for example in SystemC)	TLM Model Program-mer's View without Time (for example in SystemC)	Instruction Set Simulator	Functional Level		
Hardware Effort for Coupling with Simulation Environment	no HW needed	no HW needed	no HW needed	no HW needed	no HW needed	no HW needed	no HW needed	no HW needed
Simulation Model Costs	potentially costs of simulator and model	potentially costs of simulator and model <sup>17</sup>	potentially costs of simulator and model <sup>18</sup>	potentially costs of simulator and model <sup>19</sup>	potentially costs of simulator and model	potentially costs of simulator and model	potentially costs of simulator and model	potentially costs of simulator and model
Synchronization Frequency	after each Signal change	after each CPU cycle	after each transaction	after each transaction	after each algorithm evaluation	after each algorithm evaluation	after each algorithm evaluation	after each algorithm evaluation
Data Exchange Mechanism ( $\mu$ C SW view)	simulated registers	simulated registers	simulated registers	simulated registers	special IO-Functions	special IO-Functions	special IO-Functions	N.A.
CPU + Memories Included	yes	yes	yes	yes	yes	yes	yes	no
Peripherals Included	yes	yes	yes	yes	no	no	no	no
Coupling Transparent to the Application	yes	yes	yes	yes	no	no	no	no

---

<sup>17</sup> If available at all.

<sup>18</sup> If available at all.

<sup>19</sup> If available at all.

Table 9.5: Comparison of CHILS with Different Software Solutions (continued)

Solution Properties	Software Models					
	RTL Model	Cycle Accu- rate Model	TLM Model Program- mer's View with Time (for example in SystemC)	TLM Model Program- mer's View without Time (for example in SystemC)	Instruction Set Simulator	Functional Level
<i>Application Reuse in Real System</i>	complete	complete	complete	complete	partly	only algo- rithm, no HW/SW partitioning
<i>Peripherals Usable in Application</i>	yes, if mod- elled	yes, if mod- elled	yes (if mod- elled)	yes (if mod- elled)	no	N.A.
<i>Type of Application</i>	complete ap- plication	complete ap- plication	complete ap- plication	complete ap- plication	application without peripheral usage	N.A.
<i>Simulation Performance</i>	very low	very low to low	low to mid	low to mid	high (only algorithmic part of sw is used)	high to very high
<i>All <math>\mu</math>C Resources Avail- able for Application</i>	yes, if mod- elled	yes (if mod- elled)	yes (if mod- elled)	yes (if mod- elled)	no	no
<i>Time Accuracy</i>	very high	high	mid to high	N.A.	low	N.A.
<i>Application Execu- tion is Independent from Synchronization with the Simulation Environment</i>	yes	yes	yes	yes	no	N.A.

Table 9.5: Comparison of CHILS with Different Software Solutions (continued)

Solution Properties		Software Models					Instruction Set Simulator	Functional Level
		RTL Model	Cycle Accu- rate Model	TLM Model Program- mer's View with Time (for example in SystemC)	TLM Model Program- mer's View without Time (for example in SystemC)			
Data Abstraction of the μC Interfaces		yes	yes	yes	yes	yes	yes	N.A.
Demands Environment Simulation in Real Time		no	no	no	no	no	no	no
Observability		high	high	high	high	high	mid	N.A.



## 9.4 CHILS Performance and Accuracy

### 9.4.1 CHILS Accuracy

The execution time accuracy is defined in definition 7.2.1 as division of the runtime of a well defined piece of code within a simulated or emulated system and of the runtime of the same piece of code within the real target system. Based on the measurements in chapter 3 in section 3.5 *Effects caused by Coupling*, the **execution time accuracy** is calculated. Table 9.6 shows measurements of the STM for the runtime of the *Sieve of Eratosthenes* application and the *NOPS* application from section 3.5 *Effects Caused by Coupling*. The execution time accuracy increases with rising step size as consequence of the switching between the CHILS monitor and the CHILS device. Table 9.7 shows measurements based on a signal which is generated by the GPTA.

### 9.4.2 CHILS Performance

All measurements are done on a Pentium 4 PC with 2.4 GHz and 1 GB RAM and a TC1796ED evaluation board. A low cost DAP MiniWiggler is used to connect the  $\mu$ C board and the PC. The TC1796ED is configured to run with a CPU frequency of 150 MHz and a system frequency of 75 MHz. The reference for the simulation performance is a CPU frequency of 150 MHz. Table 9.8 presents the simulation performance values of two applications. Application one is the PID controlled air-screw rocker model which is presented in chapter 10 in subsection 10.2.1. The measurement values for activated and deactivated oversampling for event detection are listed. The application one does not need high exchange rates between the MATLAB®/Simulink® model and the  $\mu$ C control programme. Step sizes between 10ms and 1ms are chosen.

Application two is a PWM signal generation and measurement application. The GPTA of the TC1796ED generates a 1 KHz PWM signal while the MATLAB®/Simulink® model captures and measures the signal. This application is used in chapter 3 to determine the influence of the CHILS monitor to the user application. A sampling rate of 10KHz or higher is needed to capture and evaluate the 1 KHz PWM signal with a duty cycle of 20%. So the step size is chosen between 0.1ms and 0.01ms.

Figure 9.2 shows the simulation performance values which cover the full range of reasonable step sizes for CHILS applications (from 10ms up to

Application	Step Size (CPU Cycles)	STM Timing Accuracy
<i>Monitor - Non Cached</i>		
NOPS - Flash	20000	99.91%
	10000	99.81%
	5000	99.62%
NOPS - SRAM	20000	99.91%
	10000	99.82%
	5000	99.65%
Sieve - Flash	20000	99.92%
	10000	99.86%
	5000	99.66%
Sieve - SRAM	20000	99.91%
	10000	99.82%
	5000	99.65%
<i>Monitor - Cached</i>		
NOPS - Flash	20000	99.93%
	10000	99.86%
	5000	99.73%
NOPS - SRAM	20000	99.93%
	10000	99.86%
	5000	99.73%
Sieve - Flash	20000	99.94%
	10000	99.89%
	5000	99.75%
Sieve - SRAM	20000	99.93%
	10000	99.86%
	5000	99.73%

Table 9.6: STM Time Accuracy Measurements

0.003ms). Two synthetic test applications are used to show the difference between a setup with more or less I/O lines but without influence of a simulation. In appendix B more detailed tables can be found in section B.3. The overhead of the runtime analysis mechanism, the event detection and event evaluation, are strongly influenced by the amount of data to be captured and evaluated, as can be seen in the synthetic test with all

Mon. Step Size (CPU Cyc)	PWM Freq. (Hz)	GPTA Timing Acc.
<i>Monitor - NonCached</i>		
4040	3750.00	99.37%
440	3750.00	94.38%
300	3750.00	91.63%
15000	1000.00	99.81%
1500	1000.00	98.26%
<i>Monitor - Cached</i>		
4040	3750.00	99.46%
440	3750.00	95.16%
300	3750.00	92.76%
15000	1000.00	99.85%
1500	1000.00	98.49%

Table 9.7: GPTA Time Accuracy Measurements

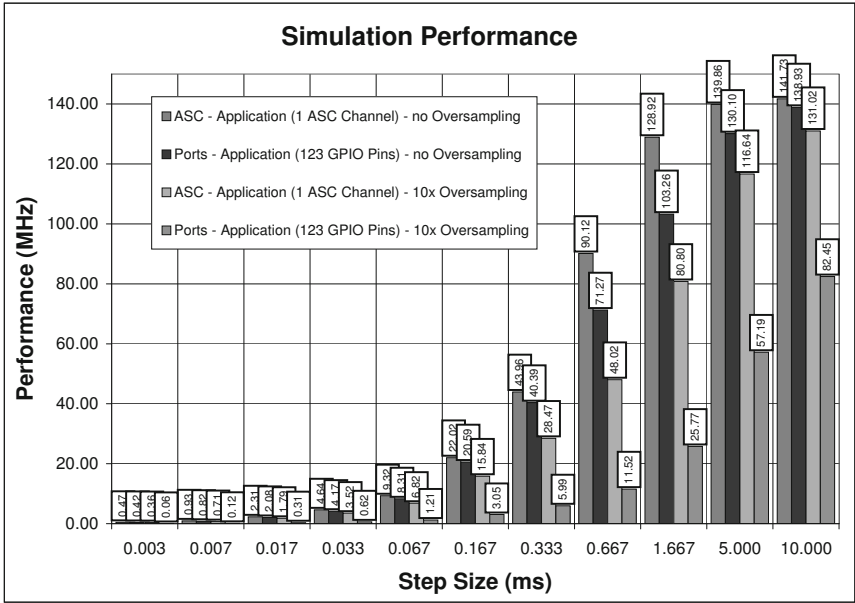


Figure 9.2: CHILS Synthetic Performance Values (Full Range of Application)

configured GPIO pins. In the normal application, this influence is reduced by two facts: a limited number of configured pins and the runtime of the simulation. It can be assumed that the simulation on the PC is the limiting factor for the overall performance in real setups.

GPIO pins	ADC channels	Step Size (ms)	Step Size (CPU Cyc.)	Sim. Time (sec)	Sim. Runtime (sec)	Sim. Perf. (MHz)
<i>PID Contr.+Air-Scr. Rocker Model, No Oversampling f. Runtime Analysis</i>						
32	3	1	150000	60	242	37.19
32	3	2	300000	60	139	64.75
32	3	5	750000	60	73	123.29
32	3	10	1500000	60	63	142.86
<i>PID Contr.+Air-Scr. Rocker Model, 10x Oversampling f. Runtime Analysis</i>						
32	3	1 <sup>20</sup>	150000	60	246	36.59
32	3	2	300000	60	147	61.22
32	3	5	750000	60	85	105.88
32	3	10	1500000	60	78	115.38
<i>PWM Generation (1000Hz), No Oversampling for Runtime Analysis</i>						
65	0	0.01	1500	1	426	0.35
65	0	0.02	3000	1	225	0.67
65	0	0.05	7500	1	101	1.49
65	0	0.1	15000	1	61	2.46

Table 9.8: CHILS-MATLAB/Simulink Sample Application Performance

## 9.5 Summary

The CHILS approach combines high performance and high accuracy with a low effort for simulation coupling in contrast to other ‘in-the-loop’ solutions. The primary difference to PIL solutions is the availability of the peripherals for the user application on the  $\mu$ C. Software models are either fast or accurate, but they provide the highest observability.

<sup>20</sup> The CHILS monitor is activated every 0.1ms if the step size is 1 ms and 10 times oversampling is configured.

## 10 CHILS Framework – Applications

The current chapter presents some CHILS applications as case studies. Wrapper modules for SystemC and MATLAB®/Simulink® are programmed as part of the thesis. The CHILS PC interface is realized as a DLL as it is explained in chapter 8 in section 8.1. Wrappers can be programmed in any language which can access the C-interface of the DLL.

### 10.1 SystemC-Coupling

The SystemC wrapper consists of different parts. Listing C.3 in appendix C presents the generic wrapper for CHILS. This wrapper is independent from the specific  $\mu$ C. It provides the possibility to configure the input and output pins of the device and to connect them with SystemC channels to the environment. The time stepping is configured at the beginning of the simulation. The CHILS device wrapper will generate SystemC events after each data exchange with the CHILS monitor.

Listing 10.1 shows a simple TC1796 SystemC CHILS model with two input channels and three output channels. This model can be easily instantiated.

```
1 #include "chils_sc_module.h"
2 /* TC1796 CHILS model */
3 class tc1796_p0: public chils_module {
4     public:
5         /* Input/Output pins */
6         sc_in<bool>      reset_n_i;
7         sc_in<bool>      p0p0_i;
8         sc_in<bool>      p0p1_i;
9         sc_out<bool>      p0p2_o;
10        sc_out<bool>      p0p3_o;
11        sc_out<bool>      p0p4_o;
12
13        tc1796_p0(sc_module_name name,
14                 double          cpuFrequency,
15                 double          chilsPeriod,
16                 sc_time_unit     chilsTimeUnit,
```

```

17             FILE                *chilsLogFile)
18
19     : chils_module(name, IFX_JTAG_ID_TC1796ED_BC
20                   , chilsLogFile)
21     , p0p0_i("tc1796_p0p0_i")
22     , p0p1_i("tc1796_p0p1_i")
23     , p0p2_o("tc1796_p0p2_o")
24     , p0p3_o("tc1796_p0p3_o")
25     , p0p4_o("tc1796_p0p4_o")
26     {
27     /* Setup Input/Output Pins and connect lines */
28     chils_setupPortPin(p0p0_i, 0, 0);
29     chils_setupPortPin(p0p1_i, 0, 1);
30     chils_setupPortPin(p0p2_o, 0, 2);
31     chils_setupPortPin(p0p3_o, 0, 3);
32     chils_setupPortPin(p0p4_o, 0, 4);
33
34     chils_setupResetPin(reset_n_i);
35     chils_setupDevice(cpuFrequency, chilsPeriod
36                       , chilsTimeUnit);
37     }
38     void sc_trace(sc_trace_file* trf)
39                 { chils_trace(trf); }
40 };

```

Listing 10.1: TC1796 SystemC CHILS Model

Figure 10.1 shows the output of the simulation. The programme on the TC1796ED reads the input of *p0p1* (Port 0 , Pin 1) and copies the value to *p0p2*. In addition, the programme generates pseudo random values for pins *p0p3* and *p0p4*.

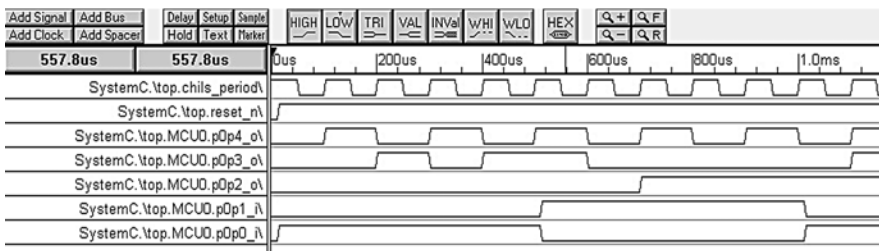


Figure 10.1: SystemC Output

CHILS also offers the possibility to have more than one  $\mu$ C model. The number of models is limited to the number of  $\mu$ C devices which are connected via DAS to the simulation computer. Listing 10.2 presents a SystemC model consisting of a cross-connected TC1766 and TC1796 device.

```

1 #include "chils_sc_toplevel_TC1766_TC1796.h"
2 /* Coupled TC1796 and TC1766 CHILS model */
3 toplevel_TC1766_TC1796 :: toplevel_TC1766_TC1796 (
4                                     sc_module_name name,
5                                     double chilsPeriod ,
6                                     sc_time_unit chilsTimeUnit ,
7                                     FILE *chilsLogFile)
8 : sc_module(name)
9
10 , chils_period("chils_period" , chilsPeriod
11                                     , chilsTimeUnit)
12 , reset_n("reset_n")
13
14 /* Cross-Connection between TC1796 and TC1766 */
15 , mcu0_p0p2_to_mcu1_p0p1("mcu0_p0p2_to_mcu1_p0p1")
16 , mcu0_p0p3_to_mcu1_p0p0("mcu0_p0p3_to_mcu1_p0p0")
17
18 , mcu0("MCU0" , 150.0E6 , chilsPeriod , chilsTimeUnit
19                                     , chilsLogFile)
20 , mcu1("MCU1" , 150.0E6 , chilsPeriod , chilsTimeUnit
21                                     , chilsLogFile)
22 {
23   mcu0.reset_n_i(reset_n);
24   mcu1.reset_n_i(reset_n);
25   /* Connect Channels to TC1796 */
26   mcu0.p0p2_o(mcu0_p0p2_to_mcu1_p0p1);
27   mcu0.p0p3_o(mcu0_p0p3_to_mcu1_p0p0);
28   /* Connect Channels to TC1766 */
29   mcu1.p0p1_i(mcu0_p0p2_to_mcu1_p0p1);
30   mcu1.p0p0_i(mcu0_p0p3_to_mcu1_p0p0);
31 }

```

Listing 10.2: Coupled TC1796 and TC1766 SystemC CHILS Model

### 10.1.1 Virtual Peripheral Extension

The virtual peripheral extension is a demonstration of the CHILS capabilities to extend an existing  $\mu\text{C}$  by new peripheral units which are not implemented within the current version (see figure 10.2). The real  $\mu\text{C}$  can be used as replacement of a future  $\mu\text{C}$  for early software development. The virtual peripheral can be accessed by the  $\mu\text{C}$  software through a virtual register set (listing 10.3). Synchronous and asynchronous communication modes are possible.

```

1 typedef struct ext_sync_t {
2     volatile unsigned int OUT;
3     volatile unsigned int IN;
4     volatile unsigned int OUT_ACK;
5     volatile unsigned int IN_ACK;
6     volatile unsigned int OUT_REQ;
7     volatile unsigned int IN_REQ;
8     volatile unsigned int IS_SYNCHRONIZED;
9 } ext_sync_t;

```

Listing 10.3: Virtual Registers for Virtual Peripheral Extension

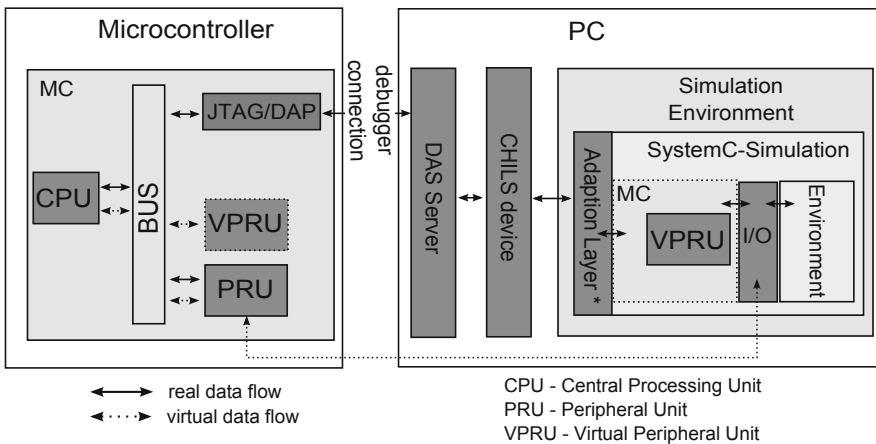


Figure 10.2: CHILS Virtual Peripheral

Listing 10.4 and listing 10.5 show extracts from the header files of the SystemC modules. The TC1796 module is implemented as bus master



device while the virtual peripheral is connected as slave to the simulated bus system.

```

1 #include <systemc.h>
2 #include "chils_sc_tc1796_master_p0.h"
3 #include "SimpleCHILSSlave.h"
4
5 /* TC1796 Wrapper is Bus Master */
6 class toplevel_tc1796_master: public sc_module {
7 public:
8     sc_clock          chils_period;
9     sc_signal<bool>    reset_n;
10    // Input to MC for stimulation
11    sc_signal<bool>    p0p0_sin;
12    sc_signal<bool>    p0p1_sin;
13
14    // Output of MC
15    sc_signal<bool>    p0p2_sout;
16    sc_signal<bool>    p0p3_sout;
17    sc_signal<bool>    p0p4_sout;
18    sc_signal<sc_time> clk;
19    sc_signal<bool>    irq;
20    sc_signal<bool>    iLineIRQ;
21
22    /* Input/Output Lines of Virtual Peripheral */
23    sc_signal<char>    simpleSlaveIn;
24    sc_signal<char>    simpleSlaveOut;
25
26    tc1796_master_p0    mcu0;
27    /* Include of the Virtual Peripheral als Slave */
28    SimpleCHILSSlave    slave;
29
30    toplevel_tc1796_master(sc_module_name name,
31                          double          chilsPeriod,
32                          sc_time_unit    chilsTimeUnit,
33                          FILE             *chilsLogFile);
34    void sc_trace( sc_trace_file* trf);
35 };

```

Listing 10.4: TC1796 SystemC with Virtual Peripheral - *toplevel\_tc1796\_master*

```

1 #include <systemc.h>
2 #include "ifx_basics.h"

```

```

3 #include "ifx_standards.h"
4
5 using namespace SC_BSX;
6 using namespace SC_STD;
7 /* The Virtual Peripheral is a Bus Slave */
8 class SimpleCHILSSlave : public vp_bus_slave {
9     public:
10         sc_in<sc_time> iClk;
11         sc_out<bool> oIRQ;
12         sc_out<bool> oIRQiLine;
13         sc_in<bool> iReset;
14         /* Input/Output Lines of Virtual Peripheral */
15         sc_in<char> iLine;
16         sc_out<char> oLine;
17         SimpleCHILSSlave( sc_module_name mn,
18             uint32 base, uint32 size);
19         SC_HAS_PROCESS(SimpleCHILSSlave);
20     protected:
21         // iLine Processing
22         void iLineCB();
23         void clockCB();
24         // Reset
25         void resetCB();
26 };

```

Listing 10.5: Virtual Peripheral *SimpleCHILSSlave*

## 10.2 Matlab/Simulink-Coupling

MATLAB®/Simulink® offers the possibility to include C or C++ code in a simulation via a mechanism which is called S-function. This code is compiled and executed within the MATLAB®/Simulink® simulation context. The C-interface of a standard Windows DLL can be directly used within an S-Function.

An extract of the CHILS MATLAB®/Simulink® wrapper can be found in appendix C in section C.2. S-Functions have a predefined structure and predefined C or C++ functions, which are called by the simulation environment. The most important functions are *mdlStart* to initialize the model parameters and *mdlOutputs* to execute the next simulation step (see listing 10.6). These functions use the CHILS interface functions directly.



```

Toplevel: simpleSlaveIn write = u
Toplevel: simpleSlaveOut read = z
INFO [570100 ns] in topIC1796.SimpleCHILSSlave : iLine received: u
INFO [570100 ns] in topIC1796.SimpleCHILSSlave : Sending iLine IRQ...
INFO [600 us] in topIC1796.MCU0 : period toggle received...
INFO [600 us] in topIC1796.MCU0 : iLineIn register read...u
INFO [600 us] in topIC1796.SimpleCHILSSlave : oLineOut register written: z
INFO [600 us] in topIC1796.MCU0 : oLineOut register written...z
INFO [700 us] in topIC1796.MCU0 : period toggle received...
INFO [700 us] in topIC1796.MCU0 : iLineIn register read...u
INFO [700 us] in topIC1796.SimpleCHILSSlave : oLineOut register written: z
INFO [700 us] in topIC1796.MCU0 : oLineOut register written...z
INFO [800 us] in topIC1796.MCU0 : period toggle received...
INFO [800 us] in topIC1796.MCU0 : iLineIn register read...u
INFO [800 us] in topIC1796.SimpleCHILSSlave : oLineOut register written: u
INFO [800 us] in topIC1796.MCU0 : oLineOut register written...u
INFO [900 us] in topIC1796.MCU0 : period toggle received...
INFO [900 us] in topIC1796.MCU0 : iLineIn register read...u
INFO [900 us] in topIC1796.SimpleCHILSSlave : oLineOut register written: u
INFO [900 us] in topIC1796.MCU0 : oLineOut register written...u
INFO [1 ms] in topIC1796.MCU0 : period toggle received...
INFO [1 ms] in topIC1796.MCU0 : iLineIn register read...u
INFO [1 ms] in topIC1796.SimpleCHILSSlave : oLineOut register written: u
INFO [1 ms] in topIC1796.MCU0 : oLineOut register written...u

```

Figure 10.3: CHILS Virtual Peripheral Simulation Output

```

1 static void mdlStart(SimStruct *S){
2     ...
3 dev = dhilCreateDevice(DAS_DID0_IFX_JTAG_TC1796ED_BC
4                        ,logFile)
5     ...
6 /* Setup init */
7 error |= dev->setupInit();
8
9 /* Setup Input Ports */
10 for (i = 0; i < mChilsMaxPortPinIn ; i++){
11     error|=
12         dev->setupPortPinIn (mChilsPortPinIn [i].portIndex ,
13                             mChilsPortPinIn [i].pinIndex );
14 }
15     ...
16 error |= dev->setupDevice(systemRunCycles);
17     ...
18 }
19 static void mdlOutputs(SimStruct *S, int_T tid){
20     ...
21 /* Set Input Values */
22 for(i = 0; i < mChilsMaxPortIn; i++){
23     InputBooleanPtrsType uPtr =
24         (InputBooleanPtrsType)ssGetInputPortSignalPtrs(S,i);
25     for(j = 0; j < mChilsPortIn[i].pins; j++){

```

```

26     bool b = (*uPtr[j] != 0);
27     error |=
28         dev->setPortPinInValue(mChilsPortPinIn[c].portIndex,
29                               mChilsPortPinIn[c].pinIndex, b);
30     c++;
31 }
32 }
33 /* Exchange Data with MC */
34 error |= dev->exchangeWithDevice();
35 ...
36 c = 0;
37 /* Get Output Values for Simulation */
38 for(i = 0; i < mChilsMaxPortOut; i++){
39     boolean_T *pY = (boolean_T *)ssGetOutputPortSignal(S, i);
40     for(j = 0; j < mChilsPortOut[i].pins; j++){
41         error |=
42             dev->getPortPinOutValue(mChilsPortPinOut[c].portIndex,
43                                     mChilsPortPinOut[c].pinIndex, &valueInOut,
44                                     &valueChanged);
45         pY[j] = valueInOut;
46         c++;
47     }
48 }
49 ...
50 }

```

Listing 10.6: Virtual Registers for Virtual Peripheral Extension

A Simulink® block is created for the graphical modelling environment. The block allows the configuration of the  $\mu$ C model and generates the necessary interconnections dynamically. Figure 10.4 shows the block and the configuration mask.

## 10.2.1 CHILS-Demonstration Platform

Heide Schenk's master thesis contains the development of a demonstration platform for the CHILS approach [Sch09]. The goal of the thesis was to evaluate the value of CHILS for a model-driven development process of an electromechanical system. A complete development process from the model to the final system is demonstrated.

The process started with the concept of the electromechanical system. A rocker with an attached air-screw was chosen. The task for the controller

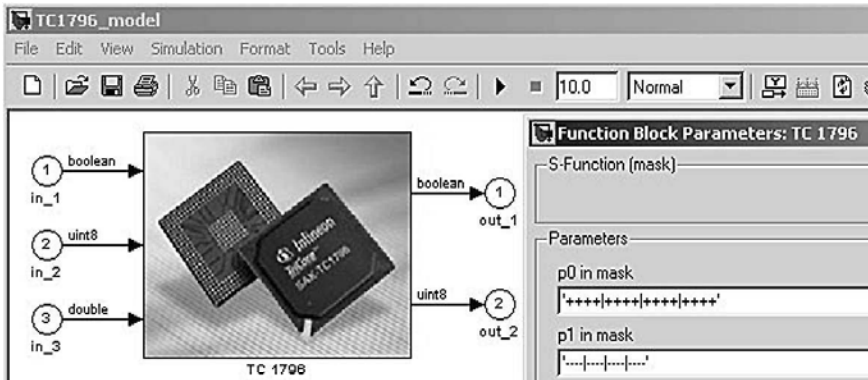


Figure 10.4: TC1796 Simulink® Block

component, a TC1796  $\mu\text{C}$ , was to keep the balance of the rocker. Figure 10.5 shows the design of the system.

The next step was to model the system with MATLAB®/Simulink®. Controller and control path were simulated.

Afterwards the controller was replaced by the real TC1796  $\mu\text{C}$  with the control software. CHILS enables this replacement. A fully functional version of the control software was developed at this stage of development. The simulated environment was replaced by the real rocker hardware in the next step. Only slight adoptions were implemented for the final version of the control software.

### 10.2.1.1 Results

Results from the final system and the simulated system, the CHILS based  $\mu\text{C}$  'model', are shown in figure 10.6. The control software used in the simulation and in the real system is nearly identical. All parameters of the control are the same. Only the virtual registers were replaced by the real registers. The blue curve represents the values from the real system. The noise of the sensor signal is clearly visible. The red curve shows the measurements from the simulation. Ignoring the noise, which is not modelled in the simulation, both curves are similar in their progression. The simulation is more dynamic while the damping in the real system is stronger. This effect is primarily caused by the model of the control path, which is derived from a system prototype.

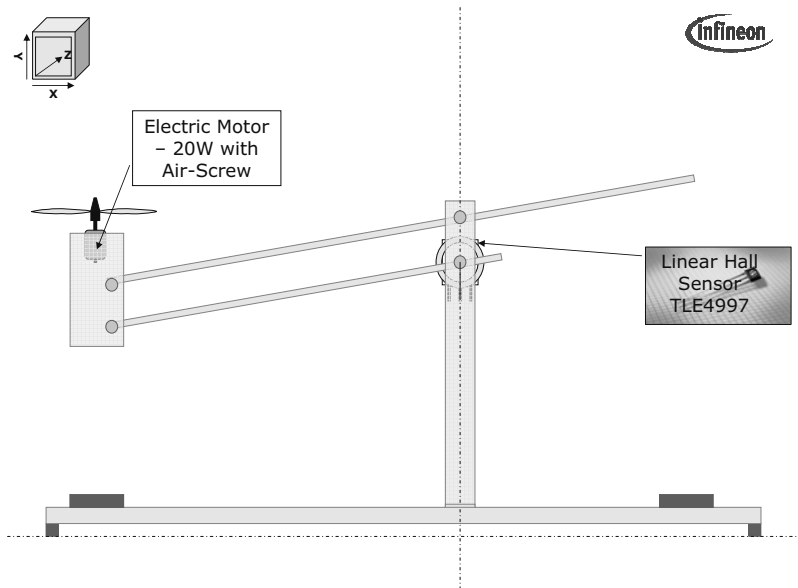


Figure 10.5: Rocker with Air-Screw [Sch09]

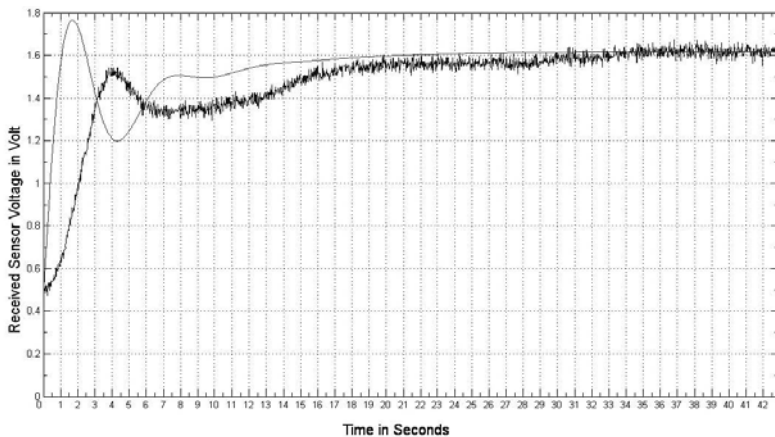


Figure 10.6: CHILS-Based Simulation and Real Control Path Measurements [Sch09]

The conclusion from [Sch09] can be summarized as follows. The CHILS approach is suitable for support and acceleration of the development of  $\mu$ C based system in an early stage of development as well as in the test phase. Within the V-model it can be applied for system design and system testing, architecture design and architecture testing and for module design and unit testing. Short iterations between the CHILS based simulation and the final system are possible to speed up the software development. CHILS can be used in addition in the development of mission critical software. Mistakes regarding exceeded boundaries, control algorithms or data type conversions can be checked and found within the simulated environment. Source code corrections can be easily ported to the final system by the easy transferring supported by CHILS.

# 11 Summary and Outlook

The raising of the abstraction level of models primarily addresses the biggest problem of current SoC and  $\mu$ C modelling. The available performance grows slower than the needs, caused by the rising complexity of such systems, so one can say: “Simulation Can’t Keep Pace with Design Size” [DMMN03]. Another important part is the verification of the correctness of the model. The model is useless if the functional correctness is not verified. Special model test benches are required for models of a high abstraction level because the silicon test benches of the real SoC and  $\mu$ C require a cycle accurate behaviour. This situation leads to the idea of using the real  $\mu$ C as a replacement for a  $\mu$ C model inside of a system simulation. The  $\mu$ C is already verified and the maximum performance is much higher than any simulation model. The overall system performance is primarily limited by the coupling system between the real hardware and the simulation.

## 11.1 Summary

The sub problems of the simulation to  $\mu$ C coupling can be divided into three main areas: the **connection between hardware and simulation**, the **interface abstraction**, and the **event exchange optimization**. Furthermore the implementation and the combination of the main approaches are of interest.

### 11.1.1 Connection between Hardware and Simulation

First off all the **connection between hardware and simulation**, so the exchange of information (events), has to be realized. This concerns the exchange modes, synchronization aspects and the hardware needed for the coupling. It also includes possibilities to connect the real hardware with different simulation environments. The hardware to simulation coupling concept used in this work is based on concepts for simulator coupling in Co-Simulation environments. The approach covers connections to continuous



and discrete simulations which differ in interpretation of time, communication and activation of simulation modules. The most practical way with the highest performance is to run the simulation and the  $\mu\text{C}$  in parallel. Events that can be seen as changes of a signal are exchanged after a predefined time between simulation and hardware. A fixed exchange step size or a variable step size is possible. The coupling between the simulated environment and the  $\mu\text{C}$  is nearly transparent to the executed software on the  $\mu\text{C}$ . Furthermore, there is no difference between a real and a simulated  $\mu\text{C}$  for the simulation.

### 11.1.2 Interface Abstraction

The **interface abstraction**, especially the level of abstraction, is a problem that is independent from the exchange of events itself. The level of abstraction only defines what an event is but it does not define how it is published. Four levels of interface abstraction are defined (see table 4.3) with respect to the available  $\mu\text{C}$  interfaces: Message-Level, Byte-Level, Digital-Level and Analogue Level. The highest suitable level of abstraction for an interface is chosen to reduce the amount of data to be shared between the  $\mu\text{C}$  and the simulation environment. This reduction accelerates the simulation- $\mu\text{C}$  coupling. Each interface module consists of two parts. One part is located on the  $\mu\text{C}$ ; the other part is located on the simulation computer. The simulation computer part of each interface module is able to convert data between the different levels of abstraction. The  $\mu\text{C}$  part of an interface consists of real or virtual registers which are readable and writable by the user application. Normally a driver layer hides the implementation, so the user software does not need to be changed to run on the final system if it is developed within a CHILS based simulation.

### 11.1.3 Event Exchange Optimization

The **event exchange optimization** has the goal to find an optimized setup for the data exchange between hardware and simulation by analyzing the system. Two techniques can be used: an **analysis of the running system** and a **pre-analysis of the overall system** and the **coupling system analysis**. The **coupling system analysis** calculates the fidelity of HIL simulation coupling systems in a formal way. The calculation is based on the transfer function in the frequency domain of the coupling system. SISO and MIMO systems are covered by this approach. The approach can be used to compare

different HIL simulation coupling systems. An optimization process, based on the fidelity value, can be executed to find the best possible configuration of a coupling system.

The overall **analysis of the real system and the system environment** consists of an analysis and a classification of algorithms executed on a  $\mu\text{C}$  and of a general system analysis. Algorithm analysis as a part of numerical mathematics. It estimates the error introduced by the numerical calculation. The used algorithm analysis flow is a bipartite process. It starts with the profiling flow to build up a database of analysis results. The input contains a set of source code and a set of input data. The application is transformed into a kind of computational graph. The graph is a combination of control flow and data flow graphs. This analysis is very costly in computation time. The classification of algorithms is based on the results of the analysis. A new programme can be analyzed by classifying the graph of the program and retrieving the corresponding results from the database. The results of the numerical analysis show the influence of errors on the programme.

The **analysis and classification of systems** is a well studied field, so the contemplations are taken from standard literature of control theory and system theory. The stability analysis of LTI systems is done by calculating the poles of the transfer function of the system. For nonlinear systems no complete consistent theory exists because an analytic solution of nonlinear differential equations is usually not available. As alternative, linearization is used to approximate the nonlinear system by a linear system.

A general analysis of the whole system is possible by combining the presented methods. The stability of the whole in-the-loop system is given if the control loop is stable in addition to an error tolerant control algorithm which is executed on the  $\mu\text{C}$ .

The **runtime analysis** yields the rate of lost events and the delay of distributed events to determine if a chosen step size for the data exchange between hardware and simulation is correct. Lost events are not always critical for the hardware to simulation coupling. Depending on the signal type, only the accuracy is reduced but the simulation is not driven into a false state. Especially if the signal is quasi-continuous, like a control or a measurement value, lost signal changes can be tolerated.

#### 11.1.4 CHILS Framework

The **CHILS framework** embeds diverse TriCore®  $\mu\text{Cs}$  into different simulation environments. It covers coupling capabilities with different simulation

environments, different  $\mu$ C devices, debugger support and tools for simulation to hardware coupling optimization. The framework provides a C/C++ interface that can be used to programme wrappers for every simulation environment which supports C or C++ modules. The physical data exchange is realized over the standard debugger interface of the  $\mu$ C devices.

The CHILS approach is applicable in the development of manifold complex  $\mu$ C based systems. Applications are for example RCP and early software development. CHILS can be applied for system design and system testing, architecture design and architecture testing and for module design and unit testing in the context of a V-model based project.

### 11.1.5 Conclusion

Adopting the CHILS concept for system simulation opens new possibilities in modelling complex hardware/software systems with existing  $\mu$ Cs. The solution is able to speed up the simulation, improves the accuracy, and allows to develop and to execute the software on real  $\mu$ C hardware at an early stage of system development. The required hardware is cost effective because the standard debugger interface of the  $\mu$ C is used. The comparison between the CHILS hardware to simulation coupling system and other HIL solutions, the dSPACE system, a classical HIL solution, and the DeskPOD™ from the company SimPOD, a special solution for  $\mu$ C to simulation coupling, shows that the CHILS approach has a higher coupling system fidelity and quality in a typical simulation scenario of a passenger car.

## 11.2 Outlook and Future Work

Outlook and future work for the CHILS approach can be divided into different sections: driver support, hardware support,  $\mu$ C extensions, system analysis and integration into commercial solutions.

### 11.2.1 Driver Support

A common abstraction like AUTOSAR supports the application of the CHILS approach. The driver adaptations can be implemented within this layer so CHILS becomes absolutely transparent to the user software. The

development of AUTOSAR driver modifications for the TC1766 has already started and an early prototype is available.

### 11.2.2 Hardware Support

CHILS currently supports four devices of the TriCore family: TC1766ED, TC1796ED, TC1767ED and TC1797ED. Adaptations for other TriCore based ED devices can be realized with small effort. CHILS implementations for other  $\mu$ C families might be more complex. The  $\mu$ Cs need a breakpoint trap mechanism and internal counters, which can be configured as a trap source, to allow an automatic switch between the user application and the CHILS monitor. Additional memory and peripheral suspend capabilities are also needed. Otherwise the restrictions for a CHILS setup are too high.

### 11.2.3 Microcontroller Extensions

A current disadvantage of CHILS is that the tested software is not 100 percent identical to the final software on the target system. For example the real drivers for some peripherals cannot be tested because CHILS needs some adaptations (see chapter 4 section 4.2). Adaptations for future versions are already described in chapter 8 in subsection 8.2.1. The most important adaptations concern the input and output registers of the  $\mu$ C peripherals. Virtual registers are currently necessary for some peripherals because the real hardware registers are not accessible by the CHILS monitor. Mechanisms like shadow registers can be implemented to overcome this drawback.

Another improvement regards the CHILS monitor context and the programme context. Only the most important registers are stored by the breakpoint trap mechanism of the actual TriCore  $\mu$ Cs. An automatic context save of all registers would be an improvement.

In some application the usage of virtual registers can be an advantage, because the input and output values of a system have to be bypassed. Classical HIL does not work in all situations. In very high integrated systems, like sensor elements and calculation elements which are implemented on the same silicon die, the simulated sensor values have to be replayed to the system, because the effort for the generation of real sensor input is too high. A good example is an integrated radar sensors system.

### **11.2.4 Integration into Commercial Solutions**

The integration of CHILS into commercial solutions, for example auto-code generators, is of interest because a larger group of users can be addressed. An integrated tool suite is necessary to allow a fast progress of prototyping, especially for RCP applications. Currently CHILS supports execution of a debugging environment in parallel to the simulation. The debugging environment has to be based on the MCD API.

### **11.2.5 Coupling System Analysis**

The future scope of activities will be to find optimal coupling strategies regarding the simulation environment and the simulated system. The basics for the coupling system analysis were introduced in this work.

### **11.2.6 Algorithm Analysis**

The completion of the analysis framework would be an important task. The current implementation is very limited regarding code sizes and programming language expressions. The database supports a relatively small number of some hundreds of thousands of data sets. The algorithm analysis approach is currently defined for the calculation of condition numbers of algorithms. This is an important criterion in numerical calculation but not the only one. Other possibilities include algorithm analysis using affine arithmetic as an extension of interval analysis [FCR03]. The calculation directly includes the fuzziness of the variables.

# A Formulas

## A.1 Chapter 5

### A.1.1 Fidelity Functions of System Comparison

**Weight function digital part**

$$w_i = \begin{cases} 0 & \text{for } i \leq 15000 \\ \min W(i) & \text{for } i \leq 75000 \\ \text{mid} W(i) & \text{for } i \leq 150000 \\ \max W(i) & \text{for } i > 150000 \end{cases} \quad (\text{A.1})$$

$$\min W(i) = (i - 15000) * 5 * 10^{-10} \quad (\text{A.2})$$

$$\text{mid} W(i) = (i - 75000) * 5 * 10^{-9} + \min W(75000) \quad (\text{A.3})$$

$$\max W(i) = (i - 150000) * 5 * 10^{-6} + \text{mid} W(150000) \quad (\text{A.4})$$

**Weight function analogue part**

$$w_i = \begin{cases} 10 & \text{for } i < 1 \\ 100000 * i * w_{i-1} & \text{for } i \geq 1 \end{cases} \quad (\text{A.5})$$

## B Tables

### B.1 Chapter 3

#### B.1.1 Time Difference Measurement

Appl.	Monitor Step Size (CPU Cycles)	Sampl. Rate	PWM Fre- quency	Mea- sured PWM Fre- quency	Period Differ- ence	CPU Cycles Differ- ence per Ex- change
<i>Monitor – Cached</i>						
PWM - Main- Prog.	1500	100.00	1000.00	1019.75	1.94%	29.04
	150	1000.00	1000.00	1268.52	21.17%	31.75
PWM - Interrupt	1500	100.00	1000.00	1016.46	1.62%	24.25
	150	1000.00	1000.00	1265.82	21.00%	31.50

Table B.1: STM Time Difference Measurements – PWM Generation

### B.1.2 Time Difference Measurement – *dsync/isync*

Monitor Step Size (CPU Cycles)	Sampl. Rate	PWM Fre- quency (Hz)	Measured PWM Fre- quency (Hz)	Period Length Differ- ence	CPU Cycles Differ- ence per Exchange
<i>Monitor – Non Cached</i>					
Sieve - SRAM	20000	33406093	33445130	3338	35.39
	10000	33406093	33465151	6671	35.38
	5000	33406093	33641483	13323	35.34
<i>Monitor – Non Cached (no isync / no dsync)</i>					
Sieve - SRAM	20000	33406093	33445130	3338	23.39
	10000	33406093	33484111	6671	23.39
	5000	33406093	33561507	13323	23.33
<i>Monitor – Cached</i>					
Sieve - SRAM	20000	33406093	33451799	3338	27.38
	10000	33406093	33497451	6671	27.39
	5000	33406093	33588188	13323	27.34
<i>Monitor – Cached (no isync / no dsync)</i>					
Sieve - SRAM	20000	33406093	33446806	3338	24.39
	10000	33406093	33487316	6671	24.35
	5000	33406093	33568757	13323	24.42

Table B.2: STM Time Difference Measurements (*isync/dsync* Difference)



## B.2 Chapter 5

### B.2.1 Result Tables of System Comparison

#### Scenario I

Digital  $\mu C$  Input Lines 16, 128

Digital  $\mu C$  Output Lines 16, 128

Analogue  $\mu C$  Input Lines 0, 0

Step Size/ Cycle Time (ms)	CHILS fidelity Simulation to Hardware	CHILS fidelity Hardware to Simulation	Approx. CHILS per- formance (kHz)
0,10	1,00	1,00	16071
0,25	0,96	0,96	34615
0,40	0,94	0,94	48648
0,50	0,94	0,94	56250
0,75	0,84	0,84	71052
1,00	0,80	0,80	81818
1,20	0,17	0,17	88524

Step Size/ Cycle Time (ms)	dSPACE fi- delity Sim- ulation to Hardware	dSPACE fi- delity Hard- ware to Simu- lation	dSPACE per- formance (kHz)
0,10	x	x	x
0,25	0,96	0,96	150000
0,40	0,94	0,94	150000
0,50	0,93	0,93	150000
0,75	0,84	0,84	150000
1,00	0,70	0,70	150000
1,20	0,17	0,17	150000

Table B.3: System Comparison – Scenario I

**Scenario II****Digital MC Input Lines** 16, 128**Digital MC Output Lines** 16, 128**Analogue MC Input Lines** 8, 32

<b>Step Size/ Cycle Time (ms)</b>	<b>CHILS fidelity Simulation to Hardware</b>	<b>CHILS fidelity Simulation to Hardware</b>	<b>Approx. CHILS per- formance (kHz)</b>
0,10	1,00	1,00	16071
0,25	0,89	0,89	34615
0,40	0,85	0,85	48648
0,50	0,84	0,84	56250
0,75	0,65	0,65	71052
1,00	0,58	0,58	81818
1,20	0,07	0,07	88524

<b>Step Size/ Cycle Time (ms)</b>	<b>dSPACE fi- delity Sim- ulation to Hardware</b>	<b>dSPACE fi- delity Sim- ulation to Hardware</b>	<b>dSPACE per- formance (kHz)</b>
0,10	x	x	x
0,25	0,89	0,89	150000
0,40	0,85	0,85	150000
0,50	0,83	0,83	150000
0,75	0,65	0,65	150000
1,00	0,45	0,45	150000
1,20	0,07	0,07	150000

Table B.4: System Comparison – Scenario II

## B.3 Chapter 9

### B.3.1 Synthetic Simulation Performance

ASC Channels	Step Size (ms)	Exchanges	Step Size (CPU Cycles)	Sim. Time (sec)	Sim. Runtime (sec)	Sim. Performance (MHz)
<i>TC1796 ASC0 Application – No Oversampling for Runtime Analysis</i>						
1	0.003	72000	500	0.24	76.73	0.47
1	0.007	72000	1000	0.48	77.19	0.93
1	0.017	72000	2500	1.2	77.93	2.31
1	0.033	72000	5000	2.4	77.62	4.64
1	0.067	72000	10000	4.8	77.28	9.32
1	0.167	72000	25000	12	81.74	22.02
1	0.333	72000	50000	24	81.89	43.96
1	0.667	72000	100000	48	79.89	90.12
1	1.667	36000	250000	60	69.81	128.92
1	5.000	12000	750000	60	64.35	139.86
GPIO Pins	Step Size (ms)	Exchanges	Step Size (CPU Cycles)	Sim. Time (sec)	Sim. Runtime (sec)	Sim. Performance (MHz)
<i>TC1796 Ports Application – No Oversampling for Runtime Analysis</i>						
123	0.003	72000	500	0.24	85.79	0.42
123	0.007	72000	1000	0.48	87.41	0.82
123	0.017	72000	2500	1.2	86.43	2.08
123	0.033	72000	5000	2.4	86.31	4.17
123	0.067	72000	10000	4.8	86.64	8.31
123	0.167	72000	25000	12	87.41	20.59
123	0.333	72000	50000	24	89.12	40.39
123	0.667	72000	100000	48	101.03	71.27
123	1.667	36000	250000	60	87.16	103.26
123	5.000	12000	750000	60	69.18	130.10

Table B.5: Synthetic Performance Values (no Oversampling for Event Detection)

<b>ASC Channels</b>	<b>Step Size (ms)</b>	<b>Exchanges</b>	<b>Step Size (CPU Cycles)</b>	<b>Sim. Time (sec)</b>	<b>Sim. Runtime (sec)</b>	<b>Sim. Performance (MHz)</b>
<i>TC1796 ASC0 Application – 10x Oversampling for Runtime Analysis</i>						
1	0.003	72000	500	0.24	100.57	0.36
1	0.007	72000	1000	0.48	100.94	0.71
1	0.017	72000	2500	1.2	100.83	1.79
1	0.033	72000	5000	2.4	102.15	3.52
1	0.067	72000	10000	4.8	105.62	6.82
1	0.167	72000	25000	12	113.66	15.84
1	0.333	72000	50000	24	126.45	28.47
1	0.667	72000	100000	48	149.93	48.02
1	1.667	36000	250000	60	111.39	80.80
1	5.0	12000	750000	60	77.16	116.64
1	10.0	6000	1500000	60	68.69	131.02
<b>GPIO Pins</b>	<b>Step Size (sec)</b>	<b>Exchanges</b>	<b>Step Size (CPU Cycles)</b>	<b>Sim. Time (ms)</b>	<b>Sim. Runtime (sec)</b>	<b>Sim. Performance (MHz)</b>
<i>TC1796 Ports Application – 10x Oversampling for Runtime Analysis</i>						
123	0.003	72000	500	0.24	577.48	0.06
123	0.007	72000	1000	0.48	579	0.12
123	0.017	72000	2500	1.2	578.56	0.31
123	0.033	72000	5000	2.4	581.86	0.62
123	0.067	72000	10000	4.8	593.02	1.21
123	0.167	72000	25000	12	590.31	3.05
123	0.333	72000	50000	24	601.03	5.99
123	0.667	72000	100000	48	625.1	11.52
123	1.667	36000	250000	60	349.27	25.77
123	5.0	12000	750000	60	157.37	57.19
123	10.0	6000	1500000	60	109.16	82.45

Table B.6: Synthetic Performance Values (10x Oversampling for Event Detection)

# C Listings

## C.1 Chapter 8

### C.1.1 CHILS-API

```
1  /* *****
2  *
3  * DESCRIPTION: DAS Hardware in the Loop (HIL) interface for devices
4  *
5  * *****/
6  #ifndef __dhil_device_h
7  #define __dhil_device_h
8  #include <stdio.h>
9  #include "das_api.h"
10 #include "das_dad.h"
11 //-----
12 typedef enum {
13     DHIL_ERR_NO_ERROR          = 0,
14     DHIL_ERR_WRONG_STATE,
15     DHIL_ERR_DEVICE, // DHIL monitor in device not working properly
16     DHIL_ERR_DEVICE_PORT_INDEX_TOO_HIGH,
17     DHIL_ERR_DEVICE_PIN_ALREADY_SETUP,
18     DHIL_ERR_DEVICE_PIN_NOT_EXISTING,
19     DHIL_ERR_DEVICE_PIN_NOT_SETUP, // Pin or whole port not setup
20     DHIL_ERR_DEVICE_ASC_INTERFACE_INDEX_TOO_HIGH,
21     DHIL_ERR_DEVICE_ASC_INTERFACE_ALREADY_SETUP,
22     DHIL_ERR_DEVICE_ASC_INTERFACE_NOT_SETUP,
23     DHIL_ERR_DEVICE_SSC_INTERFACE_INDEX_TOO_HIGH,
24     DHIL_ERR_DEVICE_SSC_INTERFACE_ALREADY_SETUP,
25     DHIL_ERR_DEVICE_SSC_INTERFACE_NOT_SETUP,
26     DHIL_ERR_DEVICE_ADC_INTERFACE_INDEX_TOO_HIGH,
27     DHIL_ERR_DEVICE_ADC_INTERFACE_ALREADY_SETUP,
28     DHIL_ERR_DEVICE_ADC_INTERFACE_NOT_SETUP,
29     DHIL_ERR_DEVICE_FADC_INTERFACE_INDEX_TOO_HIGH,
30     DHIL_ERR_DEVICE_FADC_INTERFACE_ALREADY_SETUP,
31     DHIL_ERR_DEVICE_FADC_INTERFACE_NOT_SETUP,
32     DHIL_ERR_DEVICE_EXTENSION_REGISTER_INDEX_TOO_HIGH,
33     DHIL_ERR_DEVICE_EXTENSION_REGISTER_ALREADY_SETUP,
34     DHIL_ERR_DEVICE_EXTENSION_REGISTER_NOT_SETUP,
35     DHIL_ERR_DAS, // DAS connection to device
36     DHIL_ERR_DEVICE_START_FAILED,
37 } eDhilError;
38 //-----
39 typedef enum {
40     CHILS_TRACE_MODE_NONE      = 0,
41     CHILS_TRACE_MODE_MIN,
42     CHILS_TRACE_MODE_MAX,
43 } CHILSTraceMode;
```

```

44 //-----
45 class CDhilDevice {
46     public:
47         // 1. Setup
48         virtual eDhilError setupInit() = 0;
49         // Has to be called first in setup phase. It will reset all
50         // pin/port settings etc.
51
52         // Pin direction is device point of view
53         virtual eDhilError setupPortIn (
54             unsigned portIndex,
55             unsigned short pinMask) = 0;
56
57         eDhilError setupPortPinIn (
58             unsigned portIndex, unsigned pinIndex)
59         {
60             return setupPortIn(portIndex, 1<<pinIndex);
61         }
62
63         virtual eDhilError setupPortOut (
64             unsigned portIndex,
65             unsigned short pinMask) = 0;
66         eDhilError setupPortPinOut(
67             unsigned portIndex, unsigned pinIndex)
68         {
69             return setupPortOut(portIndex, 1<<pinIndex);
70         }
71         // ASC
72         virtual eDhilError setupASCIn(unsigned ascIndex) = 0;
73         virtual eDhilError setupASCOut(unsigned ascIndex) = 0;
74
75         // SSC
76         virtual eDhilError setupSSCIn(unsigned ascIndex) = 0;
77         virtual eDhilError setupSSCOut(unsigned ascIndex) = 0;
78
79         // ADC
80         virtual eDhilError setupADCIn(unsigned adcIndex) = 0;
81
82         // Extension Module
83         // slave input register -> application reads -> slave writes
84         virtual eDhilError setupExtModuleRegIn(unsigned index) = 0;
85
86         // slave output register -> application writes -> slave reads
87         virtual eDhilError setupExtModuleRegOut(unsigned index) = 0;
88
89         // Has to be called last in setup phase.
90         // It will exchange CHILS settings with target
91         virtual eDhilError setupDevice(
92             unsigned long systemRunCycles) = 0;
93
94         // functionality like setupDevice(...),
95         // but no monitor/program reload
96         virtual eDhilError changeDeviceSetup(
97             unsigned long systemRunCycles) = 0;
98
99         // method to change runcycles while device is running
100        virtual eDhilError changeSystemRunCycles(
101            unsigned long systemRunCycles) = 0;

```



```

160
161 // Event Detection
162 virtual unsigned int getDetectedEvents() = 0;
163
164 virtual unsigned int getLostEvents() = 0;
165
166 // set application start address
167 virtual void setApplStartAddress(unsigned int startAddr) = 0;
168
169 virtual void exit() = 0;
170 };
171
172 extern class CDhilDevice *dhilCreateDevice(
173             unsigned long deviceClassId,
174             FILE *logFile,
175             unsigned deviceInstanceIndex = 0,
176             const char *serverHostAddress = 0);
177
178 /* The standard way to use the DAS API, is to define a
179 * global pointer dhildevice,
180 * and initialize it with dhilCreateDevice()
181 */
182 extern CDhilDevice *dhildevice;
183 #endif // __dhil_device_h

```

Listing C.1: DLL Header File

## C.2 Chapter 10

### C.2.1 MATLAB®/Simulink® s-function

```

1 #include "das_ifx_devices_info.h"
2 #include "dhil_device.h"
3 #include <stdio.h>
4 #include <assert.h>
5 #include <iostream>
6
7 #ifdef __cplusplus
8 extern "C" { // use the C fcn-call standard for all functions
9 #endif // defined within this scope
10
11 #define S_FUNCTION_NAME sfun_tc1796
12 #define S_FUNCTION_LEVEL 2
13
14 #include "simstruc.h"
15 #include <assert.h>
16
17 /*=====
18 * Defines *
19 *=====*/
20
21 #define TRUE 1
22 #define FALSE 0
23
24 #define PORTINPUTS 10
25 #define PORTOUTPUTS 10

```



```

26
27 #define ASCINPUTS 1
28 #define ASCOUTPUTS 1
29
30 #define ADCINPUTS 3
31
32 #define NINPUTS PORTINPUTS + ASCINPUTS + ADCINPUTS
33 #define NOUTPUTS PORTOUTPUTS + ASCOUTPUTS
34
35 #define NKPARAMS NINPUTS+NOUTPUTS
36
37 // Port pin masks for TC1796
38 const unsigned short CHILSPortPinInMaskTC1796[16] =
39 { 0xFFFF, 0xFFFF, 0xFFFC, 0xFFFF, 0xFFFF, 0x00FF,
40   0xFF0, 0x00FF, 0x00FF, 0x01FF, 0, 0, 0, 0, 0, 0 };
41 const unsigned short CHILSPortPinOutMaskTC1796[16] =
42 { 0xFFFF, 0xFFFF, 0xFFFC, 0xFFFF, 0xFFFF, 0x00FF,
43   0xFF0, 0x00FF, 0x00FF, 0x01FF, 0, 0, 0, 0, 0, 0 };
44
45 // DHIL device configuration single asc interface struct
46
47 struct chils_asc_interface {
48     unsigned char ascIndex;
49 };
50
51 struct chils_adc_pin {
52     unsigned char adcIndex;
53 };
54
55 struct chils_portPin {
56     unsigned short portIndex;
57     unsigned short pinIndex;
58 };
59
60 struct chils_port {
61     unsigned short portIndex;
62     unsigned short mask;
63     unsigned short pins;
64 };
65
66 const unsigned CHILS_MAX_PORT_PINS = 256;
67 const unsigned CHILS_MAX_PORTS = 16;
68 const unsigned CHILS_MAX_ASC_INTERFACES = 4;
69 const unsigned CHILS_MAX_ADC_PINS = 44;
70 const unsigned CHILS_MAX_PORT_SIZE = 16;
71
72 unsigned mChilsMaxPortPinIn = 0;
73 unsigned mChilsMaxPortPinOut = 0;
74 unsigned mChilsMaxPortIn = 0;
75 unsigned mChilsMaxPortOut = 0;
76 unsigned mChilsMaxASCIn = 0;
77 unsigned mChilsMaxASCOut = 0;
78
79 unsigned mChilsMaxADCIn = 0;
80 double systemFrequency = 150.0E6;
81
82 FILE *logFile;
83

```

```

84  /*=====*
85  *  Globals  *
86  *=====*/
87
88  unsigned short pXo[NOOUTPUTS];
89  unsigned short pXi[NINPUTS];
90
91  chils_portPin  mChilsPortPinIn [CHILS_MAX_PORT_PINS];
92  chils_portPin  mChilsPortPinOut[CHILS_MAX_PORT_PINS];
93
94  chils_port  mChilsPortIn[CHILS_MAX_PORTS];
95  chils_port  mChilsPortOut[CHILS_MAX_PORTS];
96
97  chils_asc_interface  mChilsASCIn[CHILS_MAX_ASC_INTERFACES];
98  chils_asc_interface  mChilsASCOut[CHILS_MAX_ASC_INTERFACES];
99
100 chils_adc_pin  mChilsADCIn[CHILS_MAX_ADC_PINS];
101
102 ...
103
104 CDhilDevice  *dev;
105
106 /*=====*
107 *  S-function methods  *
108 *=====*/
109
110 ...
111
112 #define MDL_START /* Change to #undef to remove function */
113 #if defined(MDL_START)
114 /* Function: mdlStart =====
115 * Abstract:
116 * This function is called once at start of model execution. If you
117 * have states that should be initialized once, this is the place
118 * to do it.
119 */
120 static void mdlStart(SimStruct *S)
121 {
122     int i = 0;
123     unsigned error = 0;
124
125     logFile = fopen("logfile1796.txt", "w");
126
127     if (dev==0){
128         dev = dhilCreateDevice(DAS_DID0_IFX_JTAG_TC1796ED_BC, logFile);
129         assert(dev != 0);
130     }
131
132     if (dev!=0){
133         fprintf(logFile, "CHILS.STATUS: _CHILS_device_created\n");
134         fflush( logFile );
135     } else {
136         fprintf(logFile, "CHILS.ERROR: _CHILS_device_creation_failed\n");
137         fflush( logFile );
138     }
139
140     // Setup init
141     error |= dev->setupInit();

```

```

142
143 // Setup in port
144 for (i = 0; i < mChilsMaxPortPinIn ; i++){
145     error |= dev->setupPortPinIn(mChilsPortPinIn[i].portIndex ,
146         mChilsPortPinIn[i].pinIndex);
147 }
148 // Setup out port
149 for (i = 0; i < mChilsMaxPortPinOut ; i++){
150     error |= dev->setupPortPinOut(mChilsPortPinOut[i].portIndex ,
151         mChilsPortPinOut[i].pinIndex);
152 }
153
154 for (i = 0; i < mChilsMaxASCIn ; i++){
155     error |= dev->setupASCIn(mChilsASCIn[i].ascIndex);
156 }
157
158 for (i = 0; i < mChilsMaxADCIn ; i++){
159     error |= dev->setupADCIn(mChilsADCIn[i].adcIndex);
160 }
161 for (i = 0; i < mChilsMaxASCOut ; i++){
162     error |= dev->setupASCOut(mChilsASCOut[i].ascIndex);
163 }
164
165 double chilsPeriod = ssGetSampleTime(S,0);
166 unsigned systemRunCycles =
167     (unsigned)(systemFrequency * chilsPeriod);
168
169 // Setup device
170 error |= dev->setupDevice(systemRunCycles);
171 assert(error == 0);
172
173 // init values
174 for (i = 0; i < mChilsMaxPortPinIn ; i++)
175 {
176     error |= dev->setPortPinInValue(mChilsPortPinIn[i].portIndex ,
177         mChilsPortPinIn[i].pinIndex ,0);
178 }
179
180 error |= dev->exchangeWithDevice();
181 }
182 #endif /* MDL_START */
183
184
185
186 /* Function: mdlOutputs =====
187 * Abstract:
188 * In this function, you compute the outputs of your S-function
189 * block. Generally outputs are placed in the output vector ,
190 * ssGetY(S).
191 */
192 static void mdlOutputs(SimStruct *S, int_T tid)
193 {
194     unsigned error = 0;
195     int i = 0;
196     int j = 0;
197     int c = 0;
198     bool valueInOut = 0;
199     bool valueChanged = 0;

```

```

200  bool valueTransmitted = 0;
201  bool valueReceived = 0;
202  char valueASCOut;
203
204  // set input values
205  for(i = 0; i < mChilsMaxPortIn; i++){
206      InputBooleanPtrsType uPtr =
207          (InputBooleanPtrsType)ssGetInputPortSignalPtrs(S,i);
208      for(j = 0; j < mChilsPortIn[i].pins; j++){
209          bool b = (*uPtr[j] != 0);
210          error |= dev->setPortPinInValue(mChilsPortPinIn[c].portIndex ,
211              mChilsPortPinIn[c].pinIndex , b);
212          c++;
213      }
214  }
215  c = 0;
216  for(i = mChilsMaxPortIn; i < mChilsMaxASCIIn + mChilsMaxPortIn; i++){
217      InputUInt8PtrsType uPtr =
218          (InputUInt8PtrsType)ssGetInputPortSignalPtrs(S,i);
219      error |= dev->setASCIInValue(mChilsASCIIn[c].ascIndex ,
220          (char*)uPtr[0] , &valueReceived);
221      c++;
222  }
223
224  c = 0;
225  for(i = mChilsMaxASCIIn + mChilsMaxPortIn;
226      i < mChilsMaxASCIIn + mChilsMaxPortIn + mChilsMaxADCIn; i++){
227      InputRealPtrsType uPtr =
228          (InputRealPtrsType)ssGetInputPortSignalPtrs(S,i);
229      error |= dev->setADCInValue(mChilsADCIn[c].adcIndex ,
230          (double*)uPtr[0]);
231      c++;
232  }
233
234  double chilsPeriod = ssGetSampleTime(S,0);
235  unsigned systemRunCycles = (unsigned)(systemFrequency * chilsPeriod);
236      // change run cycles
237  error |= dev->changeSystemRunCycles(systemRunCycles);
238  error |= dev->exchangeWithDevice();
239  assert(error==0);
240
241  // get output values
242  c = 0;
243  for(i = 0; i < mChilsMaxPortOut; i++){
244      boolean_T *pY = (boolean_T *)ssGetOutputPortSignal(S,i);
245      for(j = 0; j < mChilsPortOut[i].pins; j++){
246          error |= dev->getPortPinOutValue(mChilsPortPinOut[c].portIndex ,
247              mChilsPortPinOut[c].pinIndex , &valueInOut , &valueChanged);
248          pY[j] = valueInOut;
249          c++;
250      }
251  }
252
253  for(i=mChilsMaxPortOut; i<mChilsMaxASCOut+mChilsMaxPortOut; i++){
254      uint8_T *pY = (uint8_T *)ssGetOutputPortSignal(S,i);
255      // XXX test code!
256      error |= dev->getASCOutValue(mChilsASCOut[c].ascIndex,&valueASCOut,
257          &valueTransmitted);

```

```

258         C++;
259         pY[0] = (uint8_T)valueASCOOut;
260     }
261 }
262 ...
263
264 } // end of extern "C" scope
265 #endif

```

Listing C.2: MATLAB/Simulink 1796ED s-Function

## C.2.2 SystemC

```

1  #include "chils_sc_module.h"
2  #include "das_ifx_devices_info.h"
3  //-----
4  chils_module::chils_module(sc_module_name name,
5                           unsigned long chilsDeviceClassId,
6                           FILE *chilsLogFile)
7  : sc_module(name)
8  {
9      mChilsMaxPortPinIn = mChilsMaxPortPinOut = 0;
10     mChilsPeriod = 0;
11     mChilsMyDeviceInstanceIndex = mChilsMaxDeviceInstanceIndex;
12     mChilsMaxDeviceInstanceIndex++;
13
14     mDhilDevice = dhilCreateDevice(chilsDeviceClassId, chilsLogFile);
15
16     sc_assert(mDhilDevice != NULL);
17     eDhilError dhilErr = mDhilDevice->setupInit();
18     sc_assert(dhilErr == DHIL_ERR_NO_ERROR);
19     SC_THREAD(chils_exchangeWithDeviceScThread);
20 }
21 //-----
22 chils_module::~chils_module()
23 {
24     mDhilDevice->exit();
25 }
26 //-----
27 void chils_module::chils_exchangeWithDeviceScThread()
28 {
29     sc_assert(mChilsPeriod != 0);
30     eDhilError dhilErr;
31     unsigned i;
32     while (true) {
33         wait(mChilsPeriod, mChilsTimeUnit);
34         // low active reset
35         if ((mChilsResetPin->read()) == false) {
36             dhilErr = mDhilDevice->reset();
37         }
38         for (i = 0; i < mChilsMaxPortPinIn; i++) {
39             dhilErr = mDhilDevice->setPortPinInValue(
40                 mChilsPortPinIn[i].portIndex,
41                 mChilsPortPinIn[i].pinIndex,
42                 mChilsPortPinIn[i].scPinIn->read());
43             sc_assert(dhilErr == DHIL_ERR_NO_ERROR);
44         }

```

```

45
46     dhilErr = mDhilDevice->exchangeWithDevice();
47     sc_assert(dhilErr == DHIL_ERR_NO_ERROR);
48
49     for (i = 0; i < mChilsMaxPortPinOut; i++) {
50         bool value, valueChanged;
51         dhilErr = mDhilDevice->getPortPinOutValue(
52             mChilsPortPinOut[i].portIndex,
53             mChilsPortPinOut[i].pinIndex,
54             &value, &valueChanged);
55         sc_assert(dhilErr == DHIL_ERR_NO_ERROR);
56         if (valueChanged) {
57             mChilsPortPinOut[i].scPinOut->write(value);
58         }
59     }
60 }
61 }
62 //-----
63 void chils_module::chils_setupDevice(
64     double          cpuFrequency,
65     double          chilsPeriod,
66     sc_time_unit    chilsTimeUnit)
67 {
68     mChilsPeriod = chilsPeriod;
69     mChilsTimeUnit = chilsTimeUnit;
70
71     sc_time scPeriod(chilsPeriod, chilsTimeUnit);
72
73     for(int i=0;i<mChilsMaxPortPinIn;i++){
74         mDhilDevice->setupPortPinIn(
75             mChilsPortPinIn[i].portIndex,
76             mChilsPortPinIn[i].pinIndex);
77     }
78
79     for(int i=0;i<mChilsMaxPortPinOut;i++){
80         mDhilDevice->setupPortPinOut(
81             mChilsPortPinOut[i].portIndex,
82             mChilsPortPinOut[i].pinIndex);
83     }
84
85     unsigned systemRunCycles =
86         (unsigned)(cpuFrequency * scPeriod.to_seconds());
87
88     eDhilError dhilErr =
89         mDhilDevice->setupDevice(systemRunCycles);
90     sc_assert(dhilErr == DHIL_ERR_NO_ERROR);
91 }
92 //-----
93 void chils_module::chils_setupPortPin(
94     sc_in<bool>& pinIn,
95     unsigned portIndex,
96     unsigned pinIndex)
97 {
98     unsigned i = mChilsMaxPortPinIn++;
99     sc_assert(mChilsMaxPortPinIn < CHILS_MAX_PORT_PINS);
100
101     mChilsPortPinIn[i].portIndex = portIndex;
102     mChilsPortPinIn[i].pinIndex = pinIndex;

```

```

103     mChilsPortPinIn[i].scPinIn    = &pinIn;
104 }
105 //-----
106 void chils_module::chils_setupPortPin(
107     sc_out<bool>& pinOut,
108     unsigned portIndex,
109     unsigned pinIndex)
110 {
111     unsigned i = mChilsMaxPortPinOut++;
112     sc_assert(mChilsMaxPortPinOut < CHILS_MAX_PORT_PINS);
113     mChilsPortPinOut[i].portIndex = portIndex;
114     mChilsPortPinOut[i].pinIndex  = pinIndex;
115     mChilsPortPinOut[i].scPinOut  = &pinOut;
116 }
117 //-----
118 void chils_module::chils_setupResetPin(sc_in<bool>& pinIn)
119 {
120     scResetPin = &pinIn;
121 }
122 //-----
123 void chils_module::chils_trace(sc_trace_file* trf)
124 {
125     unsigned i;
126     for (i = 0; i < mChilsMaxPortPinIn; i++) {
127         mChilsPortPinIn[i].scPinIn->add_trace(trf
128             , mChilsPortPinIn[i].scPinIn->name());
129     }
130     for (i = 0; i < mChilsMaxPortPinOut; i++) {
131         mChilsPortPinOut[i].scPinOut->add_trace(trf
132             , mChilsPortPinOut[i].scPinOut->name());
133     }
134     scResetPin->add_trace(trf, scResetPin->name());
135 }

```

Listing C.3: SystemC CHILS Generic Wrapper

```

1  /* *****
2  *
3  *      Copyright (c) 2007, Infineon Technologies AG
4  *      Infineon Confidential Proprietary
5  *
6  * *****
7  * MODULE:
8  * $Id: chils_sc_tc1796_p0.h,v 1.2 2008/01/11 13:41:28 koehlerc Exp $
9  *
10 * VERSION:
11 * $Revision: 1.2 $
12 *
13 * $Date: 2008/01/11 13:41:28 $
14 *
15 * $Author: koehlerc $
16 *
17 * *****
18 * DESCRIPTION:
19 *
20 * ***** */
21 #ifndef __chils_sc_tc1796_p0_h

```

```

22 #define __chils_sc_tc1796_p0_h
23 #include "chils_sc_module.h"
24 //-----
25 const unsigned IFX_JTAG_ID_TC1796ED_BC = 0x100E2083;
26 //-----
27 // TC1796 CHILS model just with port 0
28 class tc1796_p0: public chils_module {
29     public:
30         sc_in<bool>      reset_n_i;
31         sc_in<bool>      p0p0_i;
32         sc_in<bool>      p0p1_i;
33         sc_out<bool>     p0p2_o;
34         sc_out<bool>     p0p3_o;
35         sc_out<bool>     p0p4_o;
36
37         tc1796_p0(sc_module_name name,
38                 double          cpuFrequency,
39                 double          chilsPeriod,
40                 sc_time_unit    chilsTimeUnit,
41                 FILE            *chilsLogFile)
42             : chils_module(name, IFX_JTAG_ID_TC1796ED_BC, chilsLogFile)
43             , p0p0_i("tc1796_p0p0_i")
44             , p0p1_i("tc1796_p0p1_i")
45             , p0p2_o("tc1796_p0p2_o")
46             , p0p3_o("tc1796_p0p3_o")
47             , p0p4_o("tc1796_p0p4_o")
48         {
49             chils_setupPortPin(p0p0_i, 0, 0);
50             chils_setupPortPin(p0p1_i, 0, 1);
51             chils_setupPortPin(p0p2_o, 0, 2);
52             chils_setupPortPin(p0p3_o, 0, 3);
53             chils_setupPortPin(p0p4_o, 0, 4);
54             chils_setupResetPin(reset_n_i);
55             chils_setupDevice(cpuFrequency, chilsPeriod, chilsTimeUnit);
56         }
57         void sc_trace(sc_trace_file* trf) { chils_trace(trf); }
58     };
59 #endif // __chils_sc_tc1796_p0_h

```

Listing C.4: MATLAB/SystemC 1796ED Module (with Port0 Pins)



# D Datasheets

## D.1 Chapter 2

Block diagram of the TC1796  $\mu\text{C}$

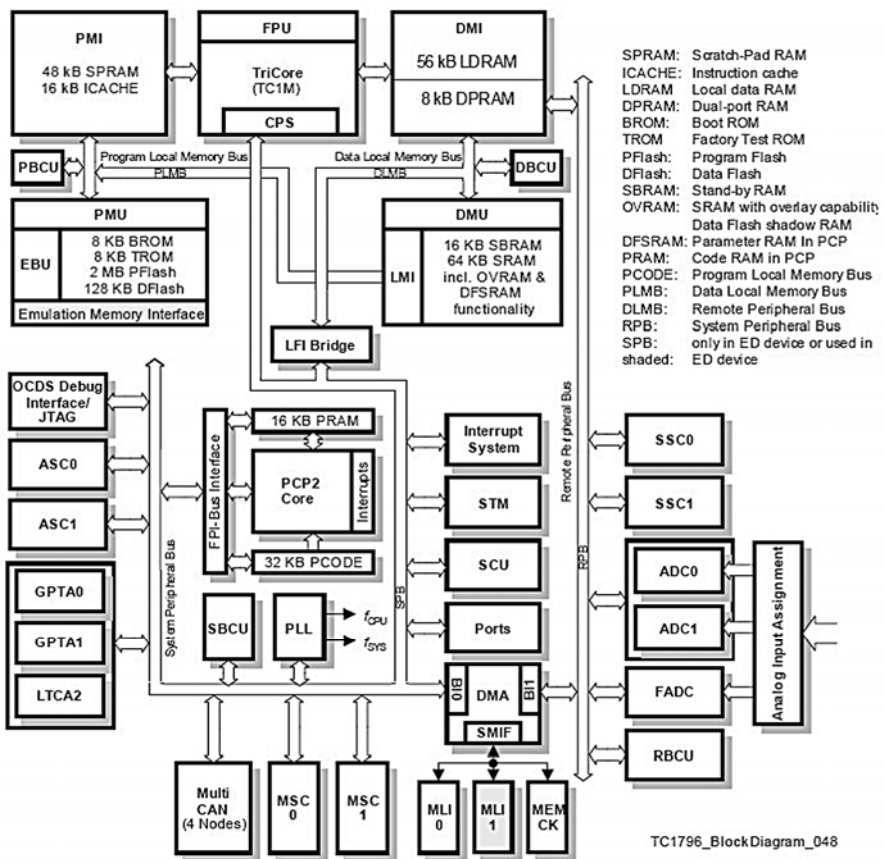


Figure D.1: Block Diagram TC1796

## D.2 Chapter 5

### D.2.1 DeskPOD™ Datasheet

Simulator Interface	10/100 BaseT Ethernet
Simulation Performance (Engaged Mode)	5000-10000 cycles per second
Simulation Performance (Disengaged Mode)	15000-250000 cycles per second
Digital I/O Lines (Bidirectional)	608
Other I/O Lines	user specific I/O lines possible (for example D/A , A/D)

Table D.1: DeskPOD™ Hardware Datasheets

### D.2.2 dSpace Datasheets

Datasheets for the following dSpace Hardware:

- 3x DS4003 Digital I/O Board
- 1x DS2103 Multi-Channel D/A Board
- 1x DS1006 Processor Board
- 1x DS2211 HIL I/O Board

DS2103 Multi-Channel D/A Board	
D/A Channels	32 parallel D/A channels 14-bit resolution +/- 5 V or +/-10 V output voltage range 0,010 ms settling time +/- 1mV offset error +/- 0,2% gain error (of FSR) +/- 1 LSB Differential linearity error

Table D.2: dSPACE HW Datasheets – DS2103 Multi-Channel D/A Board

<b>DS2211 HIL I/O Board</b>	
CAN Bus	2 CAN channels max. 1Mbaud
Serial Interface	single UART (universal asynchronous receiver and transmitter)
Digital Inputs	16 digital inputs (shared with PWM inputs) 24 PWM measurment inputs
Digital Out-puts	16 digital ouputs (shared with PWM ouputs) 9 PWM outputs
D/A Channels	20 channels 0-10V output voltage 12Bit resolution 0,020ms settling time +/- 0,5% gain error (of FSR) +/- 5mV offset error
A/D Channels	16 channels 14 Bit resolution 0,0011ms conversion time +/- 0,5% gain error +/- 10mV offset error 0-60V input voltage

Table D.3: dSPACE HW Datasheets – DS2211 HIL I/O Board

<b>DS2211 HIL I/O Board</b>	
Digital I/O	96 bidirectional digital I/O lines arranged in three ports

Table D.4: dSPACE HW Datasheets – DS2211 HIL I/O Board

<b>PHS Bus</b>	
Transfer Rate	20 Mbyte/s
Master Clients	1 16
Bit Width	32

Table D.5: dSPACE HW Datasheets – PHS Bus

# Bibliography

- [20S] 20SIM. <http://www.20sim.com/>, last checked: 13.08.2010
- [Abe03a] ABEL, Univ.-Prof. Dr.-Ing. D.: *Rapid Control Prototyping*. Aachener Forschungsgesellschaft Regelungstechnik e.V. (AFR), 2003. – 12–13 S.
- [Abe03b] ABEL, Univ.-Prof. Dr.-Ing. D.: *Rapid Control Prototyping*. Aachener Forschungsgesellschaft Regelungstechnik e.V. (AFR), 2003. – 352–353 S.
- [Abe03c] ABEL, Univ.-Prof. Dr.-Ing. D.: *Rapid Control Prototyping*. Aachener Forschungsgesellschaft Regelungstechnik e.V. (AFR), 2003. – 25–26 S.
- [Abe03d] ABEL, Univ.-Prof. Dr.-Ing. D.: *Rapid Control Prototyping*. Aachener Forschungsgesellschaft Regelungstechnik e.V. (AFR), 2003. – 8–10 S.
- [Anh06] ANHALT, Christopher ; ETAS (ed.): *Pre-configured HiL System for Power-train Control Units - PT-LABCAR: Scalable system offers cost savings and flexibility*. ETAS, 2006
- [ASML92] ALLES, S. ; SWICK, C. ; MAHMUD, S. ; LIN, F.: Real time hardware-in-the-loop vehicle simulation. In: *Proc. th IEEE Instrumentation and Measurement Technology Conference IMTC '92*, 1992, 159–164
- [AUT] AUTOSAR. <http://www.autosar.org/>, last checked: 13.08.2010
- [Bac05] BACIC, Marko: On hardware-in-the-loop simulation. In: *Proc. and 2005 European Control Conference Decision and Control CDC-ECC '05. 44th IEEE Conference on*, 2005, S. 3194–3198
- [Bac07] BACIC, Marko: Two-port network modelling for hardware-in-the-loop simulators. In: *Proc. American Control Conference ACC '07*, 2007. – ISSN 0743–1619, 3029–3034
- [Bau74] BAUER, F.L.: Computational Graphs and Rounding Errors. In: *SIAM Journal of Numerical Analysis* 11 (1974), March, Nr. 1, S. 87–96
- [BBB<sup>+</sup>03] BENINI, L. ; BERTOZZI, D. ; BRUNI, D. ; DRAGO, N. ; FUMMI, F. ; PONCINO, M.: SystemC cosimulation and emulation of multiprocessor SoC designs. In: *Computer* 36 (2003), Nr. 4, 53–59. <http://dx.doi.org/10.1109/MC.2003.1193229>. – ISSN 0018–9162
- [BBN<sup>+</sup>06] BOUCHHIMA, F. ; BRIERE, M. ; NICOLESCUL, G. ; ABID, M. ; ABOUL-HAMID, E. M.: A SystemC/Simulink Co-Simulation Framework for Continuous/Discrete-Events Simulation. In: *Proceedings of the 2006 IEEE International Behavioral Modeling and Simulation Conference*, 2006

- [BCT94] BRIGGS, Preston ; COOPER, Keith D. ; TORCZON, Linda: Improvements to Graph Coloring Register Allocation. In: *ACM Transactions on Programming Languages and Systems* 16 (1994), S. 428–455
- [Bec05] BECKER, Hans: *Differentialgleichungen mit MATHCAD und MATLAB*. Springer Berlin Heidelberg, 2005. – 17–22 S.
- [Bie07] BIESER, Carsten: *Konzept einer bibliotheksbasiert konfigurierbaren Hardware-Testeinrichtung für eingebettete elektronische Systeme*, Universität Karlsruhe, doctoral thesis, 2007
- [BMG05] BIESER, C. ; MULLER-GLASER, K.-D.: COMPASS - a novel concept of a reconfigurable platform for automotive system development and test. In: *Proc. 16th IEEE International Workshop on Rapid System Prototyping (RSP 2005)*, 2005, 135–140
- [BMRJ01] BARACOS, Paul ; MURERE, Guillaume ; RABBATH, C.A. ; JIN, Wensi: Enabling PC-Based HIL Simulation for Automotive Applications. In: *Proceedings of the IEEE conference on Electric Machines and Drives Conference 2001 (IEMDC 2001)*, 2001, 721 -729
- [BNAA05] BOUCHHIMA, F. ; NICOLESCU, G. ; ABID, M. ; ABOULHAMID, E. M.: Discrete–Continuous Simulation Model for Accurate Validation in Component-Based Heterogeneous SoC Design. In: *The 16th IEEE International Workshop on Rapid System Prototyping 2005 (RSP 2005)*, 2005, 181 - 187
- [Bra06] BRABAND, Clemens: *Synchronisation von Simulatoren unter Berücksichtigung des Konzepts polymorpher Signale*, Johann Wolfgang Goethe Universität, diploma thesis, September 2006. – Diplomarbeit
- [Bri92] BRIGGS, Preston: *Register Allocation via Graph Coloring*. Houston, TX, USA, Rice University, doctoral thesis, 1992
- [BS63] BRONSTEIN, I.N. ; SEMENDJAJEW, K.A.: *Taschenbuch der Mathematik*. 1963
- [CGL<sup>+</sup>00] CESÁRIO, W.O. ; GAUTHIER, L. ; LYONNARD, D. ; NICOLESCU, G. ; JERRAYA, A.A.: An XML-based Meta-model for the Design of Multiprocessor Embedded Systems. In: *VIUF'00* (2000)
- [CoW06] *TLM Peripheral Modeling for Platform-Driven ESL Design - Using the SystemC Modeling Library*. Version: 2006. <http://www.coware.com/PDF/TLMPeripheralModeling.PDF>, last checked: 02.09.2009
- [DAS07] *DAS Product Brief*. Version: 2007. <http://www.infineon.com/cms/en/product/promopages/das/index.html>, last checked: 13.08.2010
- [Des] *DeskPOD II - Adaptable Affordable Acceleration*. <http://www.simpod.com/products-HWSWCoverification.html>, last checked: 16.08.2010
- [DIN] DIN19226. Deutsches Institut für Normung

- [DMMN03] DENEULT, Damian ; McANDREW, Rich ; MILLER, Charlie ; NEWELL, Richard: *Retargetable Transaction-Based System Level Verification*. Version: 2003. [http://www.patni.com/thought-papers/pre-retargetable\\_transaction.aspx](http://www.patni.com/thought-papers/pre-retargetable_transaction.aspx), last checked: 04.08.2009
- [DR04] DENEULT, Damian ; RIZZATTI, Lauro: *Evaluating and improving emulator performance*. Version: 2004. <http://www.eetimes.com/electronics-news/4154631/Evaluating-and-improving-emulator-performance>, last checked: 16.09.2010
- [DRZH01] DEPPE, M. ; ROBRECHT, M. ; ZANELLA, M. ; HARDT, Dr. W.: Rapid Prototyping of Real-Time Control Laws for Complex Mechatronic Systems. In: *Proceedings of the 12th International Workshop on Rapid System Prototyping (RSP'01)* (2001)
- [dsp] dSpace. <http://www.dspace.de/>, last checked: 13.08.2010
- [dSp09] dSPACE Catalog 2009. dSPACE GmbH, 2009
- [dym] Dymola. <http://www.dynasim.se/dymola.htm>, last checked: 13.08.2010
- [EF97] EPELMAN, Marina ; FREUND, Robert M.: Condition number complexity of an elementary algorithm for resolving a conic linear system. In: *Working papers Massachusetts Institute of Technology (MIT), Sloan School of Management* (1997). – WP 3942-97-MSA
- [Eil06] EILERS, Stefan: *Zeitgenaue Simulation gemischt virtuell-realer Prototypen*, Universität Hannover, doctoral thesis, 2006. – 30–32 S.
- [EMR87] ENGELN-MÜLLGES, Gisela ; REUTER, Fritz: *Numerische Mathematik für Ingenieure*. 5. Bibliographisches Institut Wissenschaftsverlag, 1987. – 17 S.
- [Ern07] ERNST, Oliver: *Numerische Mathematik für Ingenieure - Computer-Arithmetik und numerische Stabilität*. Version: 2007. <http://www.mathe.tu-freiberg.de/~ernst/Lehre/Techniker/Folien/nt07Kapitel0.pdf>, last checked: 31.01.2009
- [ETA] <http://www.etas.com/de/index.php>, last checked: 13.08.2010
- [FBP<sup>+</sup>07] FRANCIS, G. ; BURGOS, R. ; P.RODRIGUEZ ; F.WANG ; BOROYEVICH, D. ; LIU, R. ; MONTI, A.: Virtual Prototyping of Universal Control Architecture Systems by means of Processor in the Loop Technology. In: *Twenty Second Annual IEEE Conference on Applied Power Electronics (APEC 2007)*, 2007, 21 -27
- [FCR03] FANG, Claire F. ; CHEN, Tsuhan ; RUTENBAR, Rob A.: FLOATING-POINT ERROR ANALYSIS BASED ON AFFINE ARITHMETIC. In: *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing* (2003)

- [FFP04] FORMAGGIO, Luca ; FUMMI, Franco ; PRAVADELLI, Graziano: A Timing-Accurate HW/SW Co-simulation of an ISS with SystemC. In: *Proceedings of the CODES+ISSS'04*, 2004, S. 152–157
- [GBNB06] GHEORGHE, L. ; BOUCHHIMA, F. ; NICOLESCU, G. ; BOUCHENEB, H.: Formal Definitions of Simulation Interfaces in a Continuous/Discrete Co-Simulation Tool. In: *Proc. Seventeenth IEEE International Workshop on Rapid System Prototyping*, 2006. – ISSN 1074–6005, 186–192
- [Güh05] GÜHMANN, Clemens: Model-Based Testing of Automotive Electronic Control Units. In: *3rd International Conference on Materials Testing (Test 2005)*, 2005
- [Ghe05] GHENASSIA, Frank: *Transaction-Level Modeling with SystemC: TLM Concepts and Applications for Embedded Systems*. Springer, 2005
- [GJ00] GAUTHIER, Lovic ; JERRAYA, Ahmed A.: Cycle-true simulation of the ST10 microcontroller including the core and the peripherals. In: *Proceedings of the 11th International Workshop on Rapid System Prototyping, 2000 (RSP 2000)*, 2000, 60–65
- [GNB08] GHEORGHE, Luiza ; NICOLESCU, Gabriela ; BOUCHENEB, Hanifa: Semantics for Rollback-Based Continuous/Discrete Simulation. In: *Proceedings of the 2008 IEEE International Behavioral Modeling and Simulation Conference*, 2008
- [Goo08] GOOSSENS, Gert: *Fast and Accurate System-Level Modeling for Heterogeneous Multi-Processor SoCs*. Version: 2008. [www.retarget.com](http://www.retarget.com), last checked: 13.08.2010
- [GRS07] GIROD, Bernd ; RABENSTEIN, Rudolf ; STENGER, Alexander: *Einführung in die Systemtheorie*. Teubner Verlag, 2007. – 3–6 S.
- [GVNG94] GAJSKI, Daniel D. ; VAHID, Frank ; NARAYAN, Sanjiv ; GONG, Jie: *Specification and Design of Embedded Systems*. Bd. 12. Upper Saddle River, NJ, USA : Prentice-Hall, Inc., 1994. – 53–67 S.
- [Hau01] HAUG, Gunter: *Emulation synthetisierter Verhaltensbeschreibungen mit VLIW-Prozessoren*. Logos Verlag Berlin, 2001
- [Hig02a] HIGHAM, Nicholas J.: *Accuracy and Stability of Numerical Algorithms*. SIAM, 2002. – 471–487 S.
- [Hig02b] HIGHAM, Nicholas J.: *Accuracy and Stability of Numerical Algorithms*. SIAM, 2002. – 24–25 S.
- [HKPS05] HASSOUN, S. ; KUDLUGI, M. ; PRYOR, D. ; SELVIDGE, C.: A transaction-based unified architecture for simulation and emulation. 13 (2005), Nr. 2, 278–287. <http://dx.doi.org/10.1109/TVLSI.2004.840763>. – ISSN 1063–8210

- [Hof98] HOFFMANN, Josef: *MATLAB und SIMULINK*. Addison Wesley Longman Verlag GmbH, 1998. – 351 S.
- [HP02] HEAP, Danny ; PITT, Francois: *Algorithms and Data Structures - Time-driven simulation*. Version: 2002. <http://www.cs.toronto.edu/~heap/270F02/node51.html>, last checked: 13.08.2010
- [HSBG98] HAUFE, Jürgen ; SCHWARZ, Peter ; BERNDT, Thomas ; GROSSE, Jens: Accelerated Logic Simulation by Using Prototype Boards. In: *Design, Automation and Test in Europe DATE 1998, Paris* (1998), S. 183–189
- [HTW<sup>+</sup>08] HE, Min ; TSAI, Ming-Che ; WU, Xiaolong ; WANG, Fei ; NASR, R.: Hardware/Software Codesign Pedagogy for the Industry. In: *Proc. Fifth International Conference on Information Technology: New Generations ITNG 2008*, 2008, 279–284
- [Huß08] HUSS, Jan: *Praktikum Grundlagen Regelungstechnik*. Version: 2008. <http://www.etch.fh-hamburg.de/labore/labor07/ueb.htm>, last checked: 23.11.2009
- [HZ07] HAN, Dianwei ; ZHANG, Jun: A Comparison of Two Algorithms for Predicting the Condition Number. In: *Sixth International Conference on Machine Learning and Applications*, 2007
- [Jac03] JACOBI, G.: Software ist im Auto ein Knackpunkt. In: *VDI Nachrichten* (2003). [www.vdi-nachrichten.com/vdi-nachrichten/archiv/vdi-nachrichten.asp](http://www.vdi-nachrichten.com/vdi-nachrichten/archiv/vdi-nachrichten.asp)
- [JBP06] JERRAYA, A.A. ; BOUCHHIMA, A. ; PETROT, F.: Programming models and HW-SW interfaces abstraction for multi-processor SoC. In: *Proc. 43rd ACM/IEEE Design Automation Conference*, 2006, 280–285
- [JCL99] JAE-CHEON LEE, Myung-Won S.: Hardware-in-the Loop Simulator for ABS/TCS. In: *International Conference on Control Applications* (1999)
- [JLD<sup>+</sup>04] JIANG, Zhenhua ; LEONARD, Rodrigo ; DOUGAL, Roger ; FIGUEROA, Herman ; MONTI, Antonello: Processor-in-the-Loop Simulation, Real-Time Hardware-in-the-Loop Testing, and Hardware Validation of a Digitally-Controlled, Fuel-Cell Powered Battery-Charging Station. In: *2004 35th Annual IEEE Power Electronics Specialists Conference* (2004), S. 2251–2257
- [Kar04] KARRENBERG, Ulrich: *Signale, Prozesse, Systeme*. Springer Verlag, 2004
- [KI02] KASHIMA, Hisashi ; INOKUCHI, Akihiro: Kernels for Graph Classification. (2002). [http://www.geocities.jp/kashi\\_pong/publication/am02.pdf](http://www.geocities.jp/kashi_pong/publication/am02.pdf)
- [Kie98] KIENCKE, Uwe: *Signale und Systeme*. R. Oldenbourg Verlag München Wien, 1998. – 5–6 S.



- [KM06] KLANDZEVSKI, Vase ; MAYER, Albrecht: Novel Gateway for Embedding Stand-Alone Devices in Hardware-in-the-Loop Simulation. In: *Fourth EDAA Ph.D. Forum at DATE 2006*, 2006
- [KMH08a] KOEHLER, Christian ; MAYER, Albrecht ; HERKERSDORF, Andreas: Chip Hardware-in-the-loop Simulation (CHILS) - Embedding Microcontroller Hardware in Simulation. In: WAMKEUE, R. (ed.) ; IASTED (Veranst.): *Proceedings of the 19th IASTED International Conference on Modeling and Simulation* IASTED, 2008, S. 297–302. – 978-0-88986-741-3
- [KMH08b] KOEHLER, Christian ; MAYER, Albrecht ; HERKERSDORF, Andreas: Determining the Fidelity of Hardware-In-the-Loop Simulation Coupling Systems. In: *Proceedings of the 2008 IEEE International Behavioral Modeling and Simulation Conference*, 2008
- [KMH09] KOEHLER, Christian ; MAYER, Albrecht ; HERKERSDORF, Andreas: Chip Hardware-in-the-Loop Simulation (CHILS) Coupling Optimization through new Algorithm Analysis Technique. In: *Proceedings IEEE 16th International Conference Mixed Design of Integrated Circuits and Systems*, 2009
- [KV05] K.N, Vikram ; VASUDEVAN, V.: Hardware-Software Co-simulation of Bus-based Reconfigurable Systems. In: *Microprocessors and Microsystems - Elsevier Science* 29 (2005), Nr. 4, 133-144. <http://dx.doi.org/10.1016/j.micpro.2004.07.004>
- [Lab09] NI LabVIEW für Rapid Control Prototyping und Hardware-in-the-Loop Simulation. Version: 2009. <http://zone.ni.com/devzone/cda/tut/p/id/6309>, last checked: 13.08.2010
- [Lan06] LANDENBACH, Richard: *Schnittstellenmaske für die Kopplung unterschiedlicher Simulatoren über polymorphe Signale*, Johann Wolfgang Goethe Universität, diploma thesis, 2006
- [Lap01] LAPINSKII, Viktor S.: *Algorithms for Compiler-Assisted Design-Space-Exploration of Clustered VLIW ASIP Datapaths*, University of Texas at Austin, doctoral thesis, 2001
- [LCBF08] LOMBO-CARRASQUILLA, Andres ; BECERRA-FORIGUA, Gloria E.: Integrated Modeling Methodology for Nanoscale Electronic Devices. In: *Proceedings of the 19th IASTED International Conference on Modeling and Simulation*, 2008
- [Lüd03a] LÜDECKE, André: *Simulationsgestützte Verfahren für den Top-Down-Entwurf heterogener Systeme*, Universität Duisburg-Essen, doctoral thesis, 2003. – page 8-10
- [Lüd03b] LÜDECKE, André: *Simulationsgestützte Verfahren für den Top-Down-Entwurf heterogener Systeme*, Universität Duisburg-Essen, doctoral thesis, 2003. – page 14-17

- [Len04] LENTIJO, Santiago: *Multitasking Extension of Processor in the Loop (PIL) Simulation*. Version: 2004. <http://vtb.ee.sc.edu/review/2004/AnnualConference/Presentations>, last checked: 09.03.2009
- [Lin09] *Link for TASKING - Build, test and verify embedded code using TASKING*. Version: 2009. <http://www.MathWorks.com/products/tasking/>, last checked: 22.07.2009
- [LV03] LYMAN, Peter ; VARIAN, Hal R.: *How Much Information? / University of California at Berkeley*. Version: 2003. [http://www2.sims.berkeley.edu/research/projects/how-much-info-2003/printable\\_report.pdf](http://www2.sims.berkeley.edu/research/projects/how-much-info-2003/printable_report.pdf). 2003. – technical report
- [Mar03] MARTIN, G.: *SystemC and the future of design languages: opportunities for users and research*. In: *Proc. 16th Symposium on Integrated Circuits and Systems Design SBCCI 2003*, 2003, S. 61–62
- [Mat] *MathWork*. <http://www.MathWorks.com/>, last checked: 13.08.2010
- [MCD09] *Multi-Core Debug API solves connection problems*. Version: 2009. <http://www.infineon.com/dgdl?folderId=db3a304412b407950112b409ae660342&fileId=db3a304320d39d5901215947cc1116a2>, last checked: 13.08.2010
- [Men] *Mentor Graphics*. <http://www.mentor.com/>, last checked: 13.08.2010
- [MM89] MATTERN, F. ; MEHL, H.: In: *Informatik Spektrum* 12 (1989), 198–210 S.
- [MMB07] MACDIARMID, Monte ; MACDIARMID, Monte ; BACIC, Marko: *Quantifying the Accuracy of Hardware-in-the-Loop Simulations*. In: BACIC, Marko (ed.): *Proc. American Control Conference ACC '07*, 2007. – ISSN 0743–1619, 5147–5152
- [mod] *Modelica*. <http://www.modelica.org/>, last checked: 13.08.2010
- [OVP] *OVP*. [http://www.ovpworld.org/technology\\_TLM2.0.php](http://www.ovpworld.org/technology_TLM2.0.php), last checked: 13.08.2010
- [PAB<sup>+</sup>05] PRYOR, Duaine ; ANDREWS, Jason ; BAILEY, Brian ; STICKLEY, John ; PROWSE-FOSSLER, Linda ; MAS, Gerard ; COLLEY, John ; JOHNSON, Jan ; ELIOPOULOS, Andy ; ACCELLERA (ed.): *Standard Co-Emulation Modeling Interface (SCE-MI) Reference Manual - Version 1.1.0*. Accellera, January 2005. [www.eda.org/itc](http://www.eda.org/itc)
- [PLG02] PERL, Jürgen ; LAMES, Martin ; GLITSCH, Ulrich: *Modellbildung in der Sportwissenschaft*. Hofmann, Schorndorf Verlag, 2002
- [PSF04] PLÖGER, M. ; SCHÜTTE, H. ; FERRARA, F.: *Automatisierter HIL-Test im Entwicklungsprozess vernetzter, automotiver Elektroniksysteme*. In: *AUTOREG (VDI/VDE)* (2004)

- [PVM] *Parallel Virtual Machine*. <http://www.epm.ornl.gov/pvm>, last checked: 13.08.2010
- [Rau09] RAUSCH, Prof. Dr. A. ; TECHNISCHE UNIVERSITÄT CLAUSTHAL (ed.): *V-Modell XT 1.3 Dokumentation*. Technische Universität Clausthal, 2009. <http://ftp.tu-clausthal.de/pub/institute/informatik/v-modell-xt/Releases/1.3/V-Modell-XT-Gesamt.pdf>
- [Rei06a] REINSCHKE, Kurt: *Lineare Regelungs- und Steuerungstheorie*. Springer-Verlag, 2006. – 103–110 S.
- [Rei06b] REINSCHKE, Kurt: *Lineare Regelungs- und Steuerungstheorie*. Springer-Verlag, 2006. – 173–196 S.
- [ReT] *ReTarget*. <http://www.retarget.com/>, last checked: 13.08.2010
- [Rüm97] RÜMEKASTEN, Markus: *Hybride, tolerante Synchronisation für die verteilte und parallele Simulation gekoppelter Rechnernetze*, University of Bonn, doctoral thesis, 1997
- [Rot04] ROTHMUND, Tobias: *Bedeutung von Hardware in the Loop in der Automobilindustrie*. Version: April 2004. <http://www.mechatronics-net.de/gerworkgroup/sitzung8/8ak-rothmund.pdf>, last checked: 13.08.2010
- [San84] SANDBERG, Irwin W.: A perspective on system theory. In: *IEEE Transactions on Circuits and Systems* 31 (1984), Nr. 1, S. 88–103. – ISSN 0098–4094
- [San02] SANVIDO, Marco Aurelio A.: *Hardware-in-the-loop Simulation Framework*, ETH Zürich, doctoral thesis, 2002
- [Sch04] SCHOLZ, Christian: *Zur Simulatorkopplung für mechatronische Systeme*. VDI Verlag, 2004
- [Sch05] SCHMITT, Stephen: *Integrierte Simulation und Emulation eingebetteter Hardware/Software-Systeme*, Eberhard-Karls-Universität Tübingen, doctoral thesis, 2005
- [Sch06a] SCHÖLZEL, Mario: *Automatisierter Entwurf anwendungsspezifischer VLIW-Prozessoren*, Brandenburgischen Technischen Universität Cottbus, doctoral thesis, 2006
- [Sch06b] SCHNERR, Jürgen: *Zyklengenaue Binärkodeübersetzung für die schnelle Prototypenentwicklung von System-on-a-Chip*, Eberhard-Karls-Universität Tübingen, doctoral thesis, 2006
- [Sch09] SCHENK, Heide: *Entwicklung einer Demonstrationsplattform zur Evaluierung des CHILS-Konzepts*, University of Applied Science in Regensburg, diploma thesis, 2009
- [scr] <http://www.scriptol.com/programming/list-algorithms.php>, last checked: 07.07.2009

- [SCS02] SANVIDO, M. ; CECHTICKY, V. ; SCHAUFELBERGER, W.: Testing Embedded Control Systems Using Hardware-in-the-Loop Simulation and Temporal Logic. In: *IFAC World Congress*. Barcelona, Spain, 2002
- [Sea] <http://www.mentor.com/products/fv/seamless/>, last checked: 2010/08/13
- [SGW04] SCHROLL, Rüdiger ; GRIMM, Christoph ; WALDSCHMIDT, Klaus: HEAVEN: A Framework for the Refinement of Heterogeneous Systems. In: *FDL, ECSI*, 2004, 44-56
- [Sima] <http://www.simpod.com/>, last checked: 13.08.2010
- [Simb] *HW/SW Coverification Backgrounder*. <http://www.simpod.com/products-datasheets.html>, last checked: 16.08.2010
- [Sir06] SIRIPOKARPIROM, Rawat: *Run-Time Reconfigurable Co-Emulation for Hardware-Assisted Functional Verification and Test of FPGA Designs*, Technische Universität Hamburg-Harburg, doctoral thesis, 2006
- [SLBK03] SCHENKER, Adam ; LAST, Mark ; BUNKE, Horst ; KANDEL, Abraham: Classification of Web Documents Using a Graph Model. In: *Proceedings of the Seventh International Conference on Document Analysis and Recognition (ICDAR 2003)*, 2003
- [SML04] SIRIPOKARPIROM, R. ; MAYER-LINDENBERG, F.: Hardware-assisted simulation and evaluation of IP cores using FPGA-based rapid prototyping boards. In: *Proc. 15th IEEE International Workshop on Rapid System Prototyping*, 2004. – ISSN 1074–6005, 96–102
- [SPR] <http://www.ecsi-association.org/sprint/v1/main.asp?fn=def>, last checked: 22.09.2009
- [SR04] SCHMITT, S. ; ROSENSTIEL, W.: Verification of a microcontroller IP core for system-on-a-chip designs using low-cost prototyping environments. In: *Proc. Design, Automation and Test in Europe Conference and Exhibition Bd. 3*, 2004. – ISSN 1530–1591, 96–101 Vol.3
- [SW93] SCHABACK, Robert ; WERNER, Helmut: *Numerische Mathematik*. Springer-Verlag, 1993. – 9–10 S.
- [Sys] *SystemC*. <http://www.systemc.org>, last checked: 13.08.2010
- [TAS] *Tasking*. <http://www.tasking.com/>, last checked: 2010/08/13
- [Tex99] TEXAS INSTRUMENTS INCORPORATED (ed.): *Understanding Data Converters - Application Report*. Texas Instruments Incorporated, 1999. <http://focus.ti.com/lit/an/slaa013/slaa013.pdf>
- [Tri02a] *TriCore 1 32-bit Unified Processor Core v1.3 Architecture*. V 1.3.3. Infineon Technologies AG, 2002

- [Tri02b] *TriCore 1 32-bit Unified Processor Core v1.3.3 Architecture*. V 1.3.3. Infineon Technologies AG, 2002. – 44–45 S.
- [TS90] TÖRNIG, Willi ; SPELLUCCI, Peter: *Numerische Mathematik für Ingenieure und Physiker - Band 2*. Springer Verlag, 1990. – 145–229 S.
- [Tur05] TURNER, Ray: Learn how processor-based emulation works. In: *EE Times India* (2005). [http://www.eetindia.co.in/ART\\_8800382900\\_1800000\\_TA\\_a0bdef11.HTM](http://www.eetindia.co.in/ART_8800382900_1800000_TA_a0bdef11.HTM)
- [UML] UML. <http://www.uml.org/>, last checked: 13.08.2010
- [V-M] V-Modell. <http://de.wikipedia.org/wiki/V-Modell>, last checked: 13.08.2010
- [VGB04a] VISSER, Peter M. ; GROOTHUIS, Marcel A. ; BROENINK, Jan F.: FPGAs as versatile configurable I/O devices in Hardware-in-the-Loop Simulation. In: *RTSS 2004 Work-In-Progress Proceedings* (2004), S. 41–45
- [VGB04b] VISSER, Peter M. ; GROOTHUIS, Marcel A. ; BROENINK, Jan F.: Multi-Disciplinary Design Support using Hardware-in-the-Loop Simulation. In: *Proceedings of the 5th Progress Symposium on Embedded Systems*, 2004
- [VJR04] VIRZONIS, D. ; JUKNA, T. ; RAMUNAS, D.: Design of the Embedded Software Using Flexible Hardware-In-the-Loop Simulation Scheme. In: *IEEE MELECON 2004* (2004), S. 351–354
- [VR07] VESCOVO, Guido D. ; RIZZI, Antonello: Automatic Classification of Graphs by Symbolic Histograms. In: *Proceedings of the 2007 IEEE International Conference on Granular Computing*, 2007
- [vtb] *Virtual Test Bed*. <http://vtb.engr.sc.edu/>, last checked: 13.08.2010
- [Wen07a] WENDT, Lutz: *Taschenbuch der Regelungstechnik*. Verlag Harri Deutsch, 2007. – 765 S.
- [Wen07b] WENDT, Lutz: *Taschenbuch der Regelungstechnik*. Verlag Harri Deutsch, 2007
- [Wen07c] WENDT, Lutz: *Taschenbuch der Regelungstechnik*. Verlag Harri Deutsch, 2007. – 207–257 S.
- [Wen07d] WENDT, Lutz: *Taschenbuch der Regelungstechnik*. Verlag Harri Deutsch, 2007. – 750–763 S.
- [WG06a] WINKLER, Dietmar ; GÜHMANN, Clemens: Hardware-in-the-Loop simulation of a hybrid electric vehicle using Modelica/Dymola. In: *Proceedings of the 22nd International Battery, Hybrid and Fuel Cell Electric Vehicle Symposium and Exposition*, Japan Automobile Research Institute, Yokohama, Japan, 2006, S. 1054–1063

- [WG06b] WINKLER, Dietmar ; GÜHMANN, Clemens: Synchronizing a Modelica Real-Time Simulation Model with a Highly Dynamic Engine Test-Bench System. In: *Proceedings of the 5th International Modelica Conference* (2006)
- [Wika] *Emulator*. <http://en.wikipedia.org/wiki/Emulator>, last checked: 13.08.2010
- [Wikb] *KNN*. [http://en.wikipedia.org/wiki/K-nearest\\_neighbor\\_algorithm](http://en.wikipedia.org/wiki/K-nearest_neighbor_algorithm), last checked: 13.08.2010
- [Wikc] *Nondeterministic Polynomial Time*. [http://en.wikipedia.org/wiki/Nondeterministic\\_polynomial\\_time](http://en.wikipedia.org/wiki/Nondeterministic_polynomial_time), last checked: 13.08.2010
- [WMH04] WAELTERMANN, Peter ; MICHALSKY, Thomas ; HELD, Johannes: Hardware-in-the-Loop Testing in Racing Applications. In: *SAE Motor Sports Engineering Conference & Exhibition* (2004), January
- [xPC09] *xPC Target 4*. Version: 2009. <http://www.mathworks.com/products/xpctarget/>, last checked: 13.08.2010
- [YMC00] YANG, Zan ; MIN, Byeong ; CHOI, Gwan: Si-emulation: system verification using simulation and emulation. In: *Proc. International Test Conference*, 2000, 160–169
- [ZPK00] ZEIGLER, Bernard P. ; PRAEHOFER, Herbert ; KIM, Tag G.: *Theory of modeling and simulation - integrating discrete event and continuous complex dynamic systems*. Academic Press, 2000

# Glossary

## A

**Absolute Accuracy Error** The absolute accuracy or total error of an ADC is the maximum value of the difference between an analogue value and the ideal midstep value. It includes offset, gain, integral linearity errors and also the quantization error in the case of an ADC. 80

**ACSL** The Advanced Continuous Simulation Language (ACSL) is a computer language designed for modelling continuous systems described by time-dependent, nonlinear differential equations. xvii, 20

**ALU** An Arithmetic Logic Unit (ALU) is a digital circuit performing arithmetic and logical operations. One or more ALUs are central blocks of the central processing unit (CPU) of a microprocessor or microcontroller. xvii, 25, 151

**AUTOSAR** AUTOSAR (Automotive Open System Architecture) is an open and standardized automotive software architecture, jointly developed by automobile manufacturers, suppliers and tool developers [AUT]. The partnership of automotive OEMs, suppliers and tool vendors aims to create and establish open standards for automotive electrics and electronics architectures for all application domains. 178, 179

## C

**CFG** A Control Flow Graph (CFG) represents all paths that might be traversed through a programme during its execution as a directed graph. Each node of a CFG is called a basic block. A basic block is piece of code without any control flow instructions. The edges of the graph represent the control flow of the programme. xvii, 89, 93–95, 97, 100

**CHILS** Chip-Hardware-in-the-Loop Simulation (CHILS) is a technique that is used in the development and test of complex embedded systems which are build of a existing microcontrollers. The CHILS approach has been developed as part of this thesis. The current CHILS implementation embeds a high-end Infineon TriCore® microcontroller into different simulation environments like MATLAB/Simulink® and SystemC as a replacement for a model of the microcontroller. xviii, 2, 3, 11, 13, 15, 18, 19, 22, 27, 42–48, 50, 51, 53, 59–63, 67, 77, 83–85, 110, 112, 117, 121, 125, 127–134, 136, 138, 141, 143, 147, 149, 152, 159, 162, 163, 165, 166, 168, 170, 171, 173, 176–180

**Conversion Rate** The conversion rate is maximum number of conversion per second which an ADC can achieve. 80

**CSSC** The CHILS Step Size Calculator (CSSC) is a tool of the developed CHILS framework whose objective it is to find valid and suitable step sizes for the data exchange between hardware and simulation. xviii, 139

**Cycle Accurate** Cycle Accurate models set the level of abstraction to the simulation of single cycles. They are often programmed as proprietary “C models”. 5

## D

**DAS** The Infineon Device Access Server (DAS) provides an abstraction of the physical (debugger) interface of chip for debugging, tracing, calibration and measurement applications. During operation the physical connection (for example JTAG for real device or directly for C-models) is fully transparent to the tool. The tool interface is on software level (DAS API) and implemented in a generic DLL [DAS07]. xviii, 128, 136, 141, 165

**DESCOMP** The basic idea of Design Exploration by Compilation (DESCOMP) approach is the usage of an compiler to generate an optimized processor architecture for a set of predefined algorithms. The DESCOMP approach is described in the PhD thesis of Mario Schölzel [Sch06a]. 95

**DFG** A Data Flow Graph (DFG) is a representation of a programme which describes the data dependencies of program instructions as a directed graph. Each node represents an instruction while the edges are data dependencies between instructions. xviii, 89, 93–95, 97, 98, 100, 101

**Differential Linearity** The differential nonlinearity error (sometimes seen as simply differential linearity) is the difference between an actual step width (for an ADC) or step height (for a DAC) and the ideal value of one LSB. Therefore if the step width or height is exactly one LSB, then the differential nonlinearity error is zero. 80

## E

**ECU** Electronic Control Unit (ECU) is an embedded system that controls one or more electrical systems or subsystems in a motor vehicle. xix, 1, 16–18, 20, 24, 67, 82, 148

## F

**FS** The Full Scale (FS) defines the upper limit of convertible values of an ADC. xix, 79

**FSR** The Full Scale Range (FSR) defines the range of analogue values that can be generated or measured. xix, 79



## G

**Gain Error** The gain error is defined as the difference between the nominal and the actual gain points on the transfer function after the offset error has been corrected to zero. For an ADC, the gain point is the midstep value when the digital output is full scale, and for a DAC it is the step value when the digital input is full scale. 79

## H

**HIL** Hardware-in-the-loop (HIL) simulation is a technique that is used in the development and test of complex embedded systems. In a HIL simulation the represented system consists of the simulated part and a real part, the “hardware-in-the-loop.” The real hardware interacts with the simulation. xx, 2, 15–21, 23–27, 65, 67, 68, 74, 77, 85, 128, 133, 136, 147, 148, 176–179

## I

**Instruction Accurate** Instruction Accurate models focus on the accurate simulation of the functional behaviour of the source code execution of  $\mu C$ . Timing aspects are mostly ignored. 6

**Integral Linearity** The integral nonlinearity error (sometimes seen as simply linearity error) is the deviation of the values on the actual transfer function from a straight line. For an ADC the deviations are measured at the transitions from one step to the next, and for the DAC they are measured at each step. 79

**ISS** An instruction set simulator (ISS) is a simulation model which mimics the behaviour of a processor by “reading” instructions and maintaining internal variables which represent the processor’s registers. xx, 5, 18, 23, 143, 145, 152

## K

**kNN** The k-NN is a method for classifying objects based on closest training examples in the feature space. It is a type of instance-based learning. xx, 88, 99

## L

**LSB** For ADC the width of one conversion step is defined as one LSB (one least significant bit),  $1LSB = \frac{FS}{2^n - 1}$ . For a DAC, one LSB corresponds to the height of a step between successive analogue outputs. The LSB is a measure of the resolution of the converter since it defines the number of units of the full analogue range. xx, 78–80, 219

**LTI** A Linear Time Invariant (LTI) system is signal processing systems that fulfills the linearity principle (the relationship between the input and the output of the system is a linear map) and that is independent from time (the output does not depend on the particular time the input is applied). The primarily mathematical representations of LTI are linear differential equations. 66, 69, 74, 81, 83, 105, 106, 108, 111, 115, 136, 177

## M

**MCD** Multi-Core Debug (MCD) API is a generic debug and analysis interface developed as part of the Open SoC Design Platform for Reuse and Integration of IPs (SPRINT) Project [SPR]. The MCD API is a C-interface providing the necessary means in order to perform efficient application debugging for multi-core SoCs. xx, 136, 180

**MCDS** The Infineon Multi Core Debug System (MCDS) is a configurable and scalable trigger, trace qualification, and trace compression logic block. The MCDS can record the trace of one or several cores in parallel with scalable time-stamping, conserving the order down to cycle level. This allows accurate tracing of concurrency related bugs, including shared variable-access problems for the developer's viewing. The MCDS is part of the emulation extension which consists also of a large RAM for overlay and tracing. xx, 127, 130

**MIL** Model-in-the-Loop (MIL) or System-Simulation has the target to map a real or a to-be-built system into a simulation. The system is mapped to a representation in form of equations (continuous simulation) or event sources (event driven simulation). All components of the simulation are executed as simulation models (the object of control and the control) on a developing system. Often Co-Simulation techniques are used to simulate complete system by coupling different simulation environments. xx, 23, 26

**MIMO** Multiple Input Multiple Output (MIMO) refers in in control engineering a complex control system with more than one input and/or more than one output. 70–72, 85, 106, 176

**MIPS** Instructions per second (IPS) is a measure of a computer's processor speed. MIPS stands for Million Instructions per Second. IPS values are often peak execution rates of artificial instruction sequences. 5, 6, 148, 152

## O

**Offset Error** The offset error is defined as the difference between the nominal and actual offset points. For an ADC, the offset point is the midstep value when the digital output is zero, and for a DAC it is the step value then the digital input is zero. 79

**P**

**PGA** Programme Graph Analysis (PGA) is command line tool for the numerical analysis of algorithms written in a C dialect called C—. The tool has been developed as part of this thesis. xxi, 137

**Phase-Locked Loop** A phase-locked loop (PLL) is a signal generating system that generates a signal that has a fixed relation to the phase of a reference signal. PLL are often used to clock timing pulses in digital logic circuits. 81, 82, 84

**PID** A Proportional–Integral–Derivative (PID) controller is a control loop feedback mechanism involving three separate parameters: the proportional value, the integral value and derivative value. The proportional value defines the reaction to the current error between control variable and measuring variable, the integral value defines the reaction based on the sum of recent errors, and the derivative value determines the reaction based on the error change rate. xxi, 110–112, 114, 123, 124, 159

**PIL** Processor-in-the-Loop (PIL) simulation is special kind of a HIL or a SIL simulation for complex embedded systems which consist of a plant part and a controller part. The control algorithm interacts with the model of the plant. It can be executed either on the real Microcontroller ( $\mu$ C) or on an Instruction Set Simulator (ISS). In difference to the CHILS approach, PIL does not cover the peripherals of a  $\mu$ C. xxi, 18, 22, 35, 132, 149, 150, 152, 162

**R**

**RCP** Rapid Control Prototyping (RCP) is a methodology especially for control applications. The real vehicle is controlled by a model of the software which can be rapidly adapted and optimized. The method combines and integrates older methods, for example the V-model, to reduce their disadvantages. xxi, 25, 132, 133, 141, 178, 180

**Realtime Simulation** It is called realtime simulation if the simulated time and the runtime of the simulation are identical. 12

**Register Transfer Level** The Register Transfer Level (RTL) is the currently used level of integrated circuit (IC) design, before the mapping to the physical domain (IC layout, routing and so on) is done. Description languages like VHDL or Verilog are used for the digital design. IC descriptions down to gate level are possible. 25, 45

**S**

**Settling Time** The settling time is the maximal time which a DAC needs to reach an output value of a defined accuracy after the digital input value changed. 80

**SIL** Software-in-the-Loop (SIL) simulation is a technique that is used in the development and test of complex embedded systems which consist of a plant part and a controller part. Instead of a model of the (control) software is the real control algorithm is used. The control algorithm interacts with the model of the plant. xxii, 18, 23

**Simulated Time** See Simulation Time. 12

**Simulation Runtime** The runtime of the simulation is the time which is consumed by the simulation from an external observer point of view. 12

**Simulation Time** The state change of a model during a simulation can be seen as simulated time or simulation time. The simulated time is the internal time base of the simulation. It is independent from the runtime of the simulation. 12, 14, 15, 31, 38, 40, 52

**Simulink®** MathWorks Simulink® is a commercial tool for modeling, simulating and design of complex dynamic systems. Systems can be designed on a high level by modelling the interactions between functional blocks. Simulink® is tightly coupled with MATLAB®. The MATLAB® engine is used to evaluate the equations of the Simulink® blocks. 13, 17–20, 31, 33, 46, 123, 127, 129, 145, 150, 159, 163, 168, 170, 171

**SISO** Single-Input Single-Output (SISO) usually describes in control engineering a simple control system with one input and one output. xxii, 69–71, 74, 85, 106, 110, 176

**SoC** System-on-Chip (SoC) refers to the integration of all components of a computer or electronic system into a single integrated circuit. SoC describes typically powerful processors with more than one processing elements in difference to a microcontroller. 9

**SPICE** Simulation Programme with Integrated Circuit Emphasis (SPICE) is a general-purpose analogue electronic circuit simulator which has been originally developed at the Electronics Research Laboratory of the University of California, Berkeley. The integrated circuit (IC) is described elements like transistors, resistors, capacitors and their connections. The elements are modelled by nonlinear differential algebraic equations. xxii, 20, 145

**SVM** Support Vector Machines (SVM) are a class of supervised learning methods. A support vector machine constructs a hyperplane or set of hyperplanes in a high-dimensional space. The hyperplanes can be used for classification and regression. xxii, 88

**SystemC** SystemC is a library of C++ classes and macros which provides an event-driven simulation kernel in C++. SystemC is defined and promoted by OSCI (Open SystemC Initiative) [Sys]. SystemC is often associated with Transaction-level modeling (TLM) of digital electronic systems. 6, 127, 134, 145, 146

## T

**Target Code Generation** A target code generator is able to transform the an high-level description of an algorithm into source code (for example C-code) for a target system (normally a  $\mu\text{C}$ ). 18

**TLM** Transactional Level Modelling (TLM) is a high-level modelling approach where details of communication between modules are separated from the details of the implementation of functional units. TLM denotes not a single level of abstraction. TLM is strongly connected with the C++ modelling library SystemC. xxii, 6, 145, 146

## U

**UML** Unified Modelling Language (UML) is a standardized general-purpose modeling language for the design and development of software systems created by the Object Management Group (OMG) [UML]. UML is also used for architectural design of hardware or hardware/software systems. xxii, 145

## V

**V-model** The V-model is an often used project-management structure for the development of complex system. The advantage of the V-model, in difference to models like the waterfall model, is that the phases of detail in design and testing are connected. The knowledge of the design is used to generate test scenarios on the same level of detail. 132, 141, 173, 178

**Verilog** Verilog is a hardware description language (HDL) for the design, verification, and implementation of digital electronic systems at the Register Transfer Level (RTL). 143, 146, 148

**VHDL** Very High Speed Integrated Circuit Hardware Description Language (VHDL) is a hardware description language (HDL) for the design, verification, and implementation of digital electronic systems at the Register transfer level (RTL). 143, 145, 146, 148

**VHDL-AMS** VHDL-AMS VHDL-AMS is a derivative of VHDL which includes an analogue and mixed-signal extension (AMS). VHDL-AMS enables the modeller to define the behaviour of analogue and mixed-signal systems. 147

**VLIW** Very Large Instruction Word (VLIW processors are highly parallel architectures containing multiple ALUs. The parallelization of code is realized on compiler level.) xxii, 25, 151, 153

**VTB** The Virtual Test Bed (VTB) [vtb], developed by Electrical Engineering department of the University of South Carolina, is a software for prototyping of large-scale, multi-technical dynamic systems. The VTB embeds models from different simulation environment into a unified simulation environment. xxii, 20, 149, 150

**W**

**Wiggler** A Wiggler is an interface used in the debug, design and programming of  $\mu$ Cs and SoCs based embedded systems. An Wiggler Box provides the physical connection between the On Chip Debug System (OCDS) of a  $\mu$ C or a SoC and a PC with development tools. 127, 128