Design Concepts for a Virtualizable Embedded MPSoC Architecture

Alexander Biedermann

Design Concepts for a Virtualizable Embedded MPSoC Architecture

Enabling Virtualization in Embedded Multi-Processor Systems

Springer Vieweg

Alexander Biedermann
Darmstadt, Germany

Technische Universität Darmstadt, 2014

Darmstädter Dissertation, D17

ISBN 978-3-658-08046-4 ISBN 978-3-658-08047-1 (eBook)
DOI 10.1007/978-3-658-08047-1

Library of Congress Control Number: 2014954234

Springer Vieweg
© Springer Fachmedien Wiesbaden 2014
This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.
The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.
The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made.

Printed on acid-free paper

Springer Vieweg is a brand of Springer Fachmedien Wiesbaden
Springer Fachmedien Wiesbaden is part of Springer Science+Business Media
(www.springer.com)

für meine Eltern

Abstract

In the present work, a generic hardware-based virtualization architecture is introduced. The proposed virtualization concept transforms an array of off-the-shelf embedded processors into a system with high execution dynamism. At any point in time, tasks may be shifted among the processor array transparently. Neither the software of tasks nor the processor cores have to be modified in order to enable this feature. The work details task-processor interconnection, virtualized task communication as well as means for energy awareness and fault tolerance in a virtualizable MPSoC.

Based on this architecture, concepts for the design of embedded multi-processor systems with the ability for a dynamic reshaping of their initial configuration are highlighted. This includes energy aware systems, fault tolerant systems as well as parallelized systems. For the latter, the so-called Agile Processing scheme is introduced, which enables a seamless transition between sequential and parallel execution schemes of tasks. The design of virtualizable systems is furthermore aided by a dedicated design framework, which integrates into existing, commercial design flows.

Application examples taken from different domains in embedded system design feature specific optimization goals and demonstrate the feasibility of the proposed design concepts and of the virtualization architecture. For this purpose, prototyped virtualizable multi-processor designs have been successfully synthesized for FPGAs.

Zusammenfassung

Die vorliegende Arbeit stellt ein generisches hardware-basiertes Virtualisierungskonzept vor, das aus einem Array aus eingebetteten Standardprozessoren ein System mit hoher Ausführungsdynamik schafft. Tasks können zur Laufzeit zu jedem Zeitpunkt transparent zwischen den Prozessoren im Array verschoben werden. Weder existierende Software der Tasks, noch die Prozessoren bedürfen hierbei einer speziellen Anpassung. Die Arbeit detailliert die skalierbare Task-Prozessoranbindung, virtualisierte Task-Kommunikation, sowie Maßnahmen zur Energieverwaltung und Fehlererkennung bzw. -maskierung in virtualisierbaren Systemen.

Auf Basis des vorgestellten Virtualisierungskonzepts können eingebettete Multi-Prozessorsysteme unterschiedlicher Ausprägung designt werden. Dies beinhaltet energie-bewusste Systeme, fehlertolerante Systeme oder parallel verarbeitende Systeme. Bei letzterem wird das sog. Agile Processing-Schema als nahtlose Transition zwischen sequentieller und paralleler Ausführung von Tasks eingeführt. Das Design virtualisierbarer Systeme wird durch ein eigens bereitgestelltes Design-Framework unterstützt, das eine nahtlose Anbindung an existierende, kommerzielle Design Flows bietet.

Das Entwurfskonzept wird durch Anwendungsbeispiele aus eingebetteten Systemen mit jeweils unterschiedlichen Optimierungszielen demonstriert. Hierfür wurden entsprechende Designs prototypisch für FPGAs synthetisiert.

Acknowledgments

This thesis would not exist without the help, advice, and guidance by a lot of people.

First of all, I would like to thank Prof. Sorin A. Huss for supervising my thesis. For more than half a decade, Prof. Huss guided me on the journey towards this thesis by giving advice when I was stuck, by pointing out links I had not seen and by sharing experience I was lacking. I am grateful for having had the opportunity to research and work at the Integrated and Circuits Lab where I could pursue and evolve my ideas.

Second, I would like to thank Prof. Jörg Henkel for taking the time to be second assessor of this thesis.

At the Integrated Circuits and Systems Lab, I especially thank Maria Tiedemann, who always provided kind and experienced support in all matters I still consider being mysterious and inscrutable, such as travel expense accounting.

Ideas evolve both in phases of secluded thinking as well as in lively debates. For the latter, I thank my colleagues Tom Aßmuth, Attila Jaeger, Felix Madlener, Gregor Molter, and Qizhi Tian for delightful hours of fervid discussions about Life, the Universe and Everything. I also thank my other colleagues at the Integrated Circuits and Systems Lab and at the Center for Advanced Security Research Darmstadt (CASED), in particular Tolga Arul, Carsten Büttner, Thomas Feller, Annelie Heuser, Adeel Israr, Zheng Lu, Sunil Malipatlolla, André Seffrin, Abdulhadi Shoufan, Marc Stöttinger, Hagen Stübing, and Michael Zohner.

Tens of thousands lines of implementation work by a lot of students have enabled that this work is not solely a construct of ideas, but is based on working prototypes, which allowed to test and to demonstrate the opportunities arising from the proposed virtualization concept. In this connection, I especially thank Boris Dreyer for discussing ideas and providing exceptional work over the last three years. I furthermore thank Clemens Bergmann, Antonio Gavino Casu, Hieu Ha Chi, Quoc Hien Dang, Johannes Decher, Binh Vu Duc, Nicolas Eicke, Steffen Fleckenstein, Sebastian Funke, Peter Glöckner, Maik Görtz, Christopher Huth, Inac Kadir, Michael Koch, Thorsten Jacobi, Dan Le, Randolph Lieding, Wei Lin, Kevin Luck, Mischa Lundberg, Amir Naseri, Joel Njeukam, Jan Post, Andreas Rjasanow, Lucas Rothamel, Tobias Rückelt, Gregor Rynkowski, Daniel Schneider, Kai Schwierczek, Omid Pahlevan Sharif, Johannes Simon, Niels Ströher, Markus Tasch, Do Thanh Tung, Manuel Weiel, and Matthias Zöllner.

Finally, I thank my beloved family for never demanding too much detail about my research about "something with computers" at the coffee table.

Alexander Biedermann

Contents

1 **The "Nulticore" Dilemma** 1

2 **Virtualizable Architecture for embedded MPSoC** 5
- 2.1 The Term Virtualization . . . 5
- 2.2 Virtualization for Embedded Multi-Processor Architectures . . . 7
  - 2.2.1 Characteristics of Embedded Processors . . . 7
  - 2.2.2 Purpose of Virtualization in Embedded Multi-Processor Arrays . 9
  - 2.2.3 Requirements for Embedded Virtualization . . . 12
  - 2.2.4 Prerequisites to enable Virtualization Features . . . 13
  - 2.2.5 Virtualization Procedure . . . 17
  - 2.2.6 Analysis and Characteristics of the Virtualization Procedure . . . 23
  - 2.2.7 Intermediate Conclusions . . . 28
- 2.3 Dynamically Reconfigurable Task-to-Processor Interconnection Network 30
  - 2.3.1 Interconnection Requirements for a virtualizable MPSoC . . . 30
  - 2.3.2 Architectural Alternatives . . . 32
  - 2.3.3 Permutation Networks . . . 34
  - 2.3.4 A Sorting Permutation Network as Task-Processor Interconnect 39
  - 2.3.5 Optimization . . . 47
  - 2.3.6 Intermediate Conclusions . . . 57
- 2.4 Task Scheduling via Interconnection Network . . . 59
  - 2.4.1 Binding Definitions of Task Groups and Task Budgets . . . 59
  - 2.4.2 Scheduling an n-to-1 Relation . . . 61
  - 2.4.3 Scheduling several n-to-1 Relations . . . 63
  - 2.4.4 Self-Scheduling of Tasks . . . 64
  - 2.4.5 Intermediate Conclusions . . . 75
- 2.5 Virtualized Task-to-Task Communication . . . 76
  - 2.5.1 Communication Alternatives . . . 76
  - 2.5.2 Conventional Point-to-Point Transfer in non-Virtualized Systems 79
  - 2.5.3 Architectural Enhancements and Code Changes . . . 81
  - 2.5.4 Basic Communication Scheme . . . 83
  - 2.5.5 Enhanced Communication Scheme . . . 89
  - 2.5.6 Properties and Advanced Features . . . 93
- 2.6 Reliability Features of the Virtualization Layer . . . 95
- 2.7 Features for Energy Management . . . 101

2.8 Intermediate Conclusions 104

3 The Virtualizable MPSoC: Requirements, Concepts, and Design Flows 107

3.1 Requirements for Designing on Top of the virtualizable Architecture . . 109

3.1.1 Weaknesses of Existing Design Tools 109

3.1.2 The FripGa Design Framework 110

3.2 Design Concepts for Reliable, Self-Healing Systems 116

3.2.1 Dynamic Binding Strategies 117

3.2.2 Self-Healing Strategies in the virtualizable MPSoC 119

3.3 Design Flow for Parallelized Execution 122

3.3.1 Problem Motivation 122

3.3.2 Parallelization Topologies 123

3.3.3 Data Dependencies 125

3.3.4 Design Flow 126

3.3.5 Discussion 129

3.4 Design Flow for Agile Processing 135

3.4.1 Motivation and Challenge 135

3.4.2 Prerequisites 136

3.4.3 Design Flow Modification 136

3.4.4 Scheduling Schemes for Agile Processing Designs 141

3.4.5 Discussion 146

3.5 Conclusions 147

4 Application Scenarios 149

4.1 Reshaping in an Automotive Assistance System 149

4.1.1 System Architecture 150

4.1.2 Driver Assistance Tasks 150

4.1.3 Road Types 151

4.1.4 Functional Profiles 152

4.1.5 Timing Constraints 152

4.1.6 Binding Derivation 154

4.1.7 Task Scheduling 157

4.1.8 Setup of the MPSoC and Binding Storage 158

4.1.9 Functional Reshaping 160

4.1.10 Reliability Reshaping 162

4.1.11 Synthesis Evaluation 168

4.1.12 Outcome and Outlook 170

4.2 Reshaping in an Energy-Constrained Quadrocopter System 172

4.2.1 Envisaged Scenario 173

4.2.2 Sensors and Control Systems of the Quadrocopter 174

4.2.3 Example Mission Sequence 176

4.2.4 Role of the virtualizable MPSoC 178

4.2.5 Analysis of Runtime Behavior 179
4.2.6 Synthesis Evaluation 181
4.2.7 Conclusion 182

5 Conclusion and Outlook 185

List of Figures

1.1 The Design Gap in System Design. 2

2.1 A MicroBlaze Processor System. 9
2.2 Task Management vs. dedicated Processor Resources. 10
2.3 Introduction of a Virtualization Layer. 11
2.4 Dynamic Context of a Task over Time. 14
2.5 MicroBlaze Processor with Virtualization Layer. 16
2.6 Code Injection Logic and Task Context Memory. 17
2.7 The Virtbridge. 21
2.8 A Multi-Processor System featuring the Virtualization Layer. 22
2.9 Sharing of a Processor Resource via Virtualization. 24
2.10 Timing of the Virtualization Procedure. 25
2.11 Floorplan of a virtualizable Multi-Processor System. 29
2.12 Virtualizable System with Control Processor. 32
2.13 A Crossbar Switch. 34
2.14 Static Interconnect of two Multistage Networks. 35
2.15 Blockade within a Multistage Network. 35
2.16 A Beneš Network. 36
2.17 A Max-Min Sorting Network. 37
2.18 Resource Consumption of Permutation Networks. 38
2.19 Properties of Permutation Network Types. 40
2.20 Structure of the Task-Processor Network. 41
2.21 Routing Algorithm of the Interconnection Network. 43
2.22 Generic Structure of the Network. 44
2.23 Correlation between Network Size and Frequency. 45
2.24 Clustering to improve Performance. 47
2.25 A close Cluster of two virtualizable Systems. 47
2.26 A loose Cluster of two virtualizable Systems. 48
2.27 Reshaping of a Max-Min Network. 50
2.28 Reshaping of a Max-Min Network. (cont.) 51
2.29 Resource Saving by Reshaping. 52
2.30 Increasing Frequency by Reshaping. 52
2.31 An asymmetrically shaped Network. 53
2.32 Partial Reconfiguration. 54
2.33 Task Replacement by Partial Reconfiguration. 55

2.34 Floorplan of the virtualized MPSoC. 58
2.35 Binding BV_1 over Time. 59
2.36 Binding BV_2 over Time. 60
2.37 Binding BV_3 over Time. 60
2.38 Binding BV_4 with a Task Execution Sequence. 61
2.39 Task Group Scheduling Event. 64
2.40 Task Dependencies. 65
2.41 Different Scheduling Techniques for a Task Graph. 66
2.42 Instruction Stream Monitor. 68
2.43 Nap and Sleep Instructions. 69
2.44 Self-Scheduling Code Sequence. 70
2.45 ASAP Scheduling vs. Self-Scheduling. 72
2.46 Example Task Graph for Self-Scheduling. 74
2.47 Communication Alternatives. 77
2.48 Conventional Point-to-Point Interfaces. 78
2.49 Task Data Matrix. 83
2.50 Message Memory. 84
2.51 Virtualization Layer with Message Hub. 84
2.52 Runtime Code Manipulation for sending Data. 86
2.53 Runtime Code Manipulation for receiving Data. 88
2.54 Identifying the Instruction following a Code Manipulation. 91
2.55 Replacement of the Task Memory Content. 92
2.56 Dual Modular Redundancy. 96
2.57 Triple Modular Redundancy. 96
2.58 Virtualization Layer with VMR Modules. 97
2.59 An example Task Graph. 98
2.60 Task Graph of Figure 2.59 with Redundancy Scheme. 98
2.61 Runtime Code Manipulation by Redundancy Schemes. 100
2.62 Disabling Processor Resources to reduce Energy Consumption. 102
2.63 Reducing Power Consumption by disabling Processors. 103
2.64 Frequency Stepping. 103
2.65 Correlation between Frequency and Power Consumption. 104
2.66 Resource Sharing by Virtualization. 104

3.1 Design Goals enabled by Virtualization. 108
3.2 Screenshot of Xilinx EDK. 110
3.3 Screenshot of FripGa. 111
3.4 Automated Architecture Setup in FripGa. 112
3.5 Design Flow with FripGa. 113
3.6 Virtualization Element in FripGa. 115
3.7 Workflow for virtualizable Designs in Fripga. 115
3.8 Screenshot of Hard/Soft Error CUDD. 117

3.9 Example Task Graph and Architecture Graph. 118
3.10 Error Handling by Application of optimized Bindings. 119
3.11 Shift of Processor Assignment after TMR Error Detection. 121
3.12 Code Parallelization Topologies. 124
3.13 Data Dependencies. 125
3.14 Design Flow for Parallelizing Software. 126
3.15 Split of Software by Pragma Processing. 128
3.16 Architecture resulting from Pragma Processing. 129
3.17 Timing Aspects of Parallelization. 130
3.18 Execution Sequence for a parallelized Task. 131
3.19 Execution Sequence for a parallelized Task. (cont.) 132
3.20 Execution Sequence for a parallelized Task. (cont.) 133
3.21 Structure of Agile Processing Scheme. 136
3.22 Design Flow for Agile Processing Systems. 137
3.23 Data Dependency Resolution. 139
3.24 Communication Protocol for Agile Processing. 140
3.25 Result Aggregation by ResultMux Module. 141
3.26 Communication Partner based on Execution Scheme. 141
3.27 Communication Masking by ResultMux. 142
3.28 Task Graph with Agile Processing Task. 142
3.29 Scheduling of Agile Processing Task. 143
3.30 Enhanced Scheduling of Agile Processing Task. 144

4.1 Layers of the Proreta Architecture. 150
4.2 Freeway, rural Road, and Town Profile. 151
4.3 Execution Times of Driver Assistance Tasks. 153
4.4 Task Graph and Architecture Graph. 154
4.5 Pareto Frontier of Bindings. 155
4.6 Iterative Design Flow for Binding Derivation. 156
4.7 Task Group Scheduling and Timing Constraints. 157
4.8 Timing Validation of a Binding. 159
4.9 Enabling of Self-Scheduling. 159
4.10 Design Setup in FripGa. 160
4.11 Architecture of the Automotive MPSoC. 161
4.12 Functional Reshaping for six Processors. 162
4.13 Functional Reshaping for four Processors. 163
4.14 Reliability Reshaping. 165
4.15 Reliability Reshaping. (cont.) 166
4.16 TMR Execution Scheme. 168
4.17 DMR Execution Scheme. 169
4.18 Floorplan of the virtualized MPSoC. 171
4.19 Device Logic Occupation. 172

4.20 A Quadrocopter with onboard Camera. 173
4.21 Dividing a Picture into Sections. 176
4.22 Mission Sequence for the Quadrocopter. 177
4.23 A virtualizable MPSoC with twelve Processors. 178
4.24 Execution Behavior of Flight Phases I to III. 180
4.25 Execution Behavior of Flight Phases III to VII. 181
4.26 Battery Drain in the Quadrocopter Application Example. 182
4.27 Floorplan of the virtualized MPSoC. 183
4.28 Device Logic Occupation. 184

List of Tables

2.1 Resource Overhead for Virtualization. 28
2.2 Evaluation of different Permutation Network Types. 39
2.3 Latency of virtualized Communication. 93

3.1 Comparison between FripGa and Xilinx EDK. 114
3.2 Example Binding Set optimized for Reliability. 119

4.1 Functional Profiles. 152
4.2 Tasks present in the Quadrocopter System. 174
4.3 Battery Level Thresholds. 175

List of Algorithms

1 Phase 1a of Virtualization Procedure. 19
2 Phase 1b of Virtualization Procedure. 20
3 Phase 2 of Virtualization Procedure. 22
4 Phase 3 of Virtualization Procedure. 23
5 Sorting Property of a Crossbar Switch. 37
6 Mapping of a Binding Vector to a Network Configuration. 42
7 Computation of the Depth of a Max-Min Network. 45
8 Asymmetric Shaping of a Max-Min Network. 49
9 Time Division Scheduling. 61
10 Enhanced Time Division Scheduling. 62
11 Network Configuration for Binding Vector with a Task Group. 62
12 Task Group Scheduling. 63
13 Network Configuration for Binding Vector with Task Groups. 65
14 Self-Scheduling Code Routine. 71
15 Sending with static Addressing. 87
16 Receiving with static Addressing. 88
17 Applying Redundancy Schemes in a virtualizable MPSoC. 99

18 System Reshaping by Virtualization. 108
19 Scheduling of an Agile Processing Task. 143
20 Enhanced Scheduling of an Agile Processing Task. 145

21 Timing Validation for conventional Scheduling Scheme. 157
22 Timing Validation for enhanced Scheduling Scheme. 158
23 Identifying faulty Processors in a DMR Scenario. 169

List of Abbreviations

AP	Auto Pilot
API	Application Programming Interface
APT	Air Pressure Tracking
AR	Audio Response
ASAP	As Soon As Possible
AST	Abstract Syntax Tree
AXI	Advanced eXtensible Interface Bus
BRAM	Block Random Access Memory
BS	Blind Spot Detection
BV	Binding Vector
C2T	CO_2 Level Tracking
CAN	Controller Area Network
CD	Collision Detection
CIL	Code Injection Logic
CISC	Complex Instruction Set Computer
CORDIS	Community Research and Development Information Service
DMR	Dual Modular Redundancy
ECC	Error-Correcting Code
ESL	Electronic System Level Design
FF	Flip-Flop
FLA	Fog Light Assistant
FPGA	Field-Programmable Gate Array
FSL	Fast Simplex Link
GC	Ground Communication
GHD	Ground Heat Detection & Analysis
GPS	Global Positioning System
HBA	High Beam Assistance
HDL	Hardware Description Language
ILP	Integer Linear Programming
IP	Intellectual Property
ISM	Instruction Stream Monitor
LCS	Lane Change Support
LKS	Lane Keeping Support
LUT	Look-up Table
MIC	Microphone

MPSoC	Multi-Processor System-on-Chip
MSR	Machine Status Register
NMR	n Modular Redundancy
NoC	Network-on-Chip
NUMA	Non-Uniform Memory Access
ODP	Organic Design Pattern
PA	Parking Assistant
PC	Program Counter (Chapter 2)/Position Control (Chapter 4)
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
RT	Radiation Level Tracking
SCCC	Single-Chip Cloud Computer
SI	System Initialization
SIMD	Single Instruction, Multiple Data
SoC	System on Chip
SOL	Sources of Light
SPE	Synergistic Processor Element
Tcl	Tool Command Language
TCM	Task Context Memory
TDM	Task Data Matrix
TG	Task Group
TMR	Triple Modular Redundancy
TSD	Traffic Sign Detection
TT	Temperature Tracking
VBR	Visual Body Recognition
VHDL	Very High Speed Integrated Circuit Hardware Description Language
VLIW	Very Long Instruction Word
VMR	Virtualized Modular Redundancy