Synthesis of Embedded Software

Sandeep K. Shukla • Jean-Pierre Talpin
Editors

Synthesis of Embedded Software

Frameworks and Methodologies for Correctness by Construction

Springer

Editors
Dr. Sandeep K. Shukla
Virginia Tech
Bradley Dept. Electrical
& Computer Engineering
Whittemore Hall 302
24061 Blacksburg Virgin Islands
USA
shukla@vt.edu

Dr. Jean-Pierre Talpin
INRIA Rennes-Bretagne
Atlantique
35042 Rennes CX
Campus de Beaulieu
France
Jean-Pierre.Talpin@inria.fr

ISBN 978-1-4419-6399-4 e-ISBN 978-1-4419-6400-7
DOI 10.1007/978-1-4419-6400-7
Springer New York Dordrecht Heidelberg London

Library of Congress Control Number: 2010930045

Introduction

Sandeep Shukla and Jean-Pierre Talpin

Embedded Systems are ubiquitous. In applications ranging from avionics, automotive, and industrial process control all the way to the handheld PDAs, cell phones, and bio-medical prosthetic devices, we find embedded computing devices running embedded software systems. Growth in embedded systems is exponential, and there are orders of magnitude more embedded processors and embedded applications today in deployment than other forms of computing. Some of the applications running on such embedded computing platforms are also safety-critical, real-time, and require absolute guarantees of correctness, timeliness, and dependability.

Design of such safety-critical applications require utmost care. These applications must be verified for functional correctness; satisfaction of real-time constraints must be ensured; and must be properly endowed with reliability/dependability properties. These requirements pose hard challenges to system designers. A large body of research done over the last few decades exists in the field of designing safety-critical hardware and software systems. A number of standards have evolved, specifications have been published, architecture analysis and design languages have been proposed, techniques for optimal mapping of software components to architectural elements have been studied, modular platform architectures have been standardized, etc. Certification by various authorities and strict requirements for satisfying certification goals have been developed. Today's avionics, automotive, process control, and many other safety-critical systems are usually developed based on such a large body of knowledge and technology.

However, there is still no guarantee that a system will not crash, or an automotive throttle by wire system malfunction will not cause problems, or a process control system will fail to work properly. All the model based development process

S. Shukla
Fermat Laboratory, Virginia Tech, Blacksburg, VA, USA
e-mail: Sandeep.Shukla@vt.edu

J.-P. Talpin
INRIA, Centre de Recherche Rennes – Bretagne Atlantique, Campus de Beaulieu,
Rennes, France
e-mail: Jean-Pierre.Talpin@inria.fr

specification, and strict certification are meant to eliminate the possibilities of such eventualities, but cannot eliminate them completely. Formal verification could provide more certainty than test based certification, but even formal verification is as good as the abstract models and properties that one verifies on such models.

Even though, 100% guarantee is not possible, we believe that a rigorous formal methods based approach which transforms abstract specifications via refinement steps will be the methodology of choice for such systems, if it is already not so. While the refinement steps will add further complexity to satisfy various constraints, only refinements to be used are those that could be mathematically proven to preserve the correctness already proven at higher abstraction levels.

Towards achieving the goal of designing safety-critical systems, especially embedded software in this way, a number of disciplines have been developed, and are being researched at the moment. These research areas consist of programming models, abstractions, semantics preserving refinements, real-time and other non-functional constraint based model elaboration, automated synthesis of code, formal verification, etc.

In Europe, a number of programming models and languages based on such programming models have evolved for this purpose. Synchronous and polychronous programming models constitute important classes among them. Stream based programming models, Petri Nets, Process algebra and various semantically enriched UML models also belong to these. In the USA, actor based modeling paradigms, tuple-space based modeling, and guarded command languages are the main ones, other than automata based composition of processes.

Defining a programming model, its semantics, and translation schemes based on operational semantics into execution languages such as C is only one part of the task. The other part is to create an entire methodology around the language, programming model, and tools for creating executable software from it.

In 2009, we developed a one day tutorial at the ACM/IEEE Conference on Design, Analysis and Test in Europe (DATE 2009) where a number of speakers spoke about the various languages, models, and tools for correct-by-construction embedded software design. Following up on this successful tutorial the present book introduces approaches to the high-level specification of embedded software that are supported with correct-by-construction, automated synthesis and code generation techniques. It focuses on approaches based on synchronous models of computation and the many programmatic paradigms it can be associated with to provide integrated, system-level, design methodologies.

Chapter 1, on the “Compilation of Polychronous Data-Flow Equations”, gives a thorough presentation of the program analysis and code generation techniques present in Polychrony, a compiler for the data-flow synchronous language Signal.

Introduced in the late 1980s, Signal and its polychronous model of computation stand among the most developed concepts of synchronous programming. It allows to model concurrent embedded software architectures using high-level multi-clocked synchronous data-flow equations.

The chapter defines the formal methodology consisting of all required program analysis and transformation techniques to automatically generate the sequential or concurrent code suiting the target architecture (embedded or distributed) or compilation goals (modularity or performance).

Chapter 2, on "Formal Modeling of Embedded Systems with Explicit Schedules and Routes", investigates data-flow process networks as a formal framework to model data-flow signal processing algorithms (e.g., multimedia) for networks of processors (e.g., multi-cores).

While not readily a programming model, a process network provides a suitable model of computation to reason on the issues posed by the efficient implementation of data-intensive processing algorithms on novel multi-processor architectures. Process networks provide a bridge between the appealing scheduling and routing issues posed by multi-core architectures and the still very conventional programming language concepts that are commonly used to devise algorithms.

In this aim, process networks act as a foundational principle to possibly define domain-specific yet high-level programming and design concepts to match the main concern of data-processing algorithms, which is to minimize communication latencies (with external memory or between computation units) as early as possible during system design.

Chapter 3, on "Synoptic: A Domain-Specific Modeling Language for Space Onboard Application Software", is a perfect illustration of a related effort to design and structure high-level programming concepts from the specific requirements to an application domain. Synoptic is a modeling language developed in the frame of the French ANR project SPaCIFY. Its aim is to provide a programming environment for real-time embedded software on-board space application, and especially control and command software.

Synoptic consists of an Eclipse-based modeling environment that covers the many aspects of aerospace software design: control-dominated modules are described using imperative synchronous programs, commands using mode automata, data-processing modules using data-flow diagrams, and also partitioning, timing and mapping of these modules onto satellite architectures using AADL-like diagrams.

As such, it is a domain-specific framework relying on the standard modeling notations such as Simulink/Stateflow and AADL to provide the engineer with a unified modeling environment to handle all heterogeneous analysis, design, implementation and verification tasks, as defined in collaboration with the industrial end users of the project.

Chapter 4, on "Compiling SHIM", provides a complete survey to another domain-specific language, whose aim is to help engineer software interfaced with asynchronous hardware: "the shim under the leg of a table". In the design of Shim, emphasis is put on so-called scheduling-independence as a principle guaranteeing correctness by construction while assuming minimum programmatic concepts.

Shim is a concurrent imperative programming language in which races and non-determinism are avoided at compile-time by enforcing simple analysis rules ensuring a deterministic input–output behavior regardless of way program threads are scheduled: scheduling independence. The chapter gives a complete survey of Shim, the programming principles it is build upon, the program analysis techniques it uses, its code generation strategies for both shared-memory and message-passing architectures.

Chapter 5, on "A Module Language for Typing SIGNAL Programs by Contracts", brings up the polychronous model of computation again to present a means to modularly and compositionally support assumption-guarantee reasoning in that framework. Contract-based design has become a popular reasoning concept in which contracts are used to negotiate the correctness of assumptions made on the definition of a component at the point where it is used and provides guarantees to its environment.

The chapter first elaborates formal foundations by defining a Boolean algebra of contracts in a trace-theoretical framework. Based on that contracts algebra, a general-purpose module language is then specified. The algebra and module system are instantiated to the framework of the synchronous data-flow language Signal. This presentation is illustrated with the specification of a protocol for Loosely Time-Triggered Architectures (LTTA).

The aim of Chap. 6, on "MRICDF: A Polychronous Model for Embedded Software Synthesis", is to define a simple modeling language and framework to reason about synchronous multi-clocked actors, as an alternative to the formal methods in the Polychrony toolbox for use by a larger community of software engineers.

As the leitmotiv of MRICDF can be summarized as "polychrony for the mass", a systematic emphasis is put on presenting a modeling language that uses programming concepts as simple as possible to implement a complete synchronous design workbench.

The chapter gives a thorough presentation of the design modeling language and on the issues encountered in implementing MRICDF. The use of this formalism together with its visual editor EmCodeSyn is illustrated through case studies to design safety-critical applications.

Chapter 7, on "The Time Model of Logical Clocks Available in the OMG MARTE Profile", gives a detailed presentation of CCSL, the clock constraints specification language defined in the OMG profile MARTE for specifying timing relations in UML. The aim of CCSL is to capture some of the essence of logical time and encapsulate it into a dedicated syntax that can be used to annotate UML diagrams in order to refine them by formalizing this critical design aspect.

Multi-form or polychronous logical time, introduced and made popular through its central role in the theory of synchronous languages, is already present in many formalisms pertaining to embedded system design, sometimes in a hidden fashion. Logical time considers time bases that can be generated from any sort of sequences

of events, not necessarily equally spaced in physical time. CCSL provides ways to express loose or strict constraints between distinct logical clocks. Solving such clock constraints amounts to relating clocks to a common reference, which then can be thought of as closer to the physical clock.

Finally, Chap. 8, entitled "From Synchronous Specifications to Statically Scheduled Hard Real-Time Implementations", addresses the issue of transforming high-level data-flow specifications expressed in a synchronous model of computation down to sequential or distributed expressed in a real-time model of computation.

Hard real-time embedded systems are often designed as automatic control systems that can include both continuous and discrete parts. The functional specification of such systems is usually done using data-flow formalisms such as Simulink or Scade. These formalisms are either quasi-synchronous or synchronous, but go beyond the classical data-flow model, by introducing means of conditional execution, hence allowing the description of hierarchical execution modes.

Specific real-time implementation approaches have been proposed for such formalisms, which exploit the hierarchical conditions to improve the generated code. This last chapter present one such approach. It takes data-flow synchronous specifications as input and uses static scheduling heuristics to automatically produce efficient distributed real-time code. The chapter insists on improving the analysis of the hierarchical conditions in order to obtain more efficient code.

This book is intended to provide readers with a sample on advanced topics and state of the art in various fields related to safety-critical software synthesis. Hence, it is obviously not an exhaustive handbook. Our hope is that the reader will find the area of research covered in this book important and interesting, and find other interesting research being done in related topics from various other sources.

We wish to thank all the contributors of this book for participating in preparing this volume, and for demonstrating their patience with the entire publication process. We also thank Charles Glaser from Springer for his support with this project. Finally, we thank the National Science Foundation, the US Air Force Office of Scientific Research, INRIA, and the ARTIST network of excellence for their support and funding which enabled us to work on this project.

Contents

Contributors

Charles André Laboratoire I3S, UMR 6070 CNRS, Université de Nice-Sophia Antipolis, 06903 Sophia Antipolis Cedex, France, INRIA, Centre de Recherche de Sophia-Antipolis Méditerranée, 06902 Sophia Antipolis, France, Charles.Andre@sophia.inria.fr

Loïc Besnard CNRS/IRISA, Campus de Beaulieu, Rennes, France, Loic.Besnard@irisa.fr

Jean-Paul Bodeveix IRIT-ACADIE, Université de Toulouse, site Paul Sabatier, 118 Route de Narbonne, 31062 Toulouse Cedex 9, France, bodeveix@irit.fr

Julien Boucaron INRIA, Centre de Recherche de Sophia-Antipolis Méditerranée, 06902 Sophia Antipolis, France, Julien.Boucaron@sophia.inria.fr

Jeremy Buisson VALORIA, Écoles de St-Cyr Cotquidan, Université Européenne de Bretagne, 56381 Guer Cedex, France, jeremy.buisson@st-cyr.terre-net.defense.gouv.fr

Anthony Coadou INRIA, Centre de Recherche de Sophia-Antipolis Méditerranée, 06902 Sophia Antipolis, France, Anthony.Coadou@sophia.inria.fr

Alexandre Cortier IRIT-ACADIE, Université de Toulouse, site Paul Sabatier, 118 Route de Narbonne, 31062 Toulouse Cedex 9, France, Cortier@irit.fr

Fabien Dagnat Institut Télécom – Télécom Bretagne, Université Européenne de Bretagne, Technopole Brest Iroise, CS83818, 29238 Brest Cedex 3, France, fabien.dagnat@telecom-bretagne.eu

Julien DeAntoni INRIA, Centre de Recherche de Sophia-Antipolis Méditerranée, 06902 Sophia Antipolis, France
and
Laboratoire I3S, UMR 6070 CNRS, Université de Nice-Sophia Antipolis, 06903 Sophia Antipolis Cedex, France, Julien.DeAntoni@sophia.inria.fr

Robert de Simone INRIA, Centre de Recherche de Sophia-Antipolis, Sophia Antipolis Cedex, France, rs@sophia.inria.fr

Stephen A. Edwards Columbia University, New York, NY, USA, sedwards@cs.columbia.edu

Mamoun Filali IRIT-ACADIE, Université de Toulouse, site Paul Sabatier, 118 Route de Narbonne, 31062 Toulouse Cedex 9, France, filali@irit.fr

Gerald Garcia Thales Alenia Space, 100 Boulevard Midi, 06150 Cannes, France, gerald.garcia@thalesaleniaspace.com

Thierry Gautier INRIA, Centre de Recherche de Rennes, Campus de Beaulieu, Rennes, France, Thierry.Gautier@inria.fr

Yann Glouche INRIA, Centre de Recherche de Rennes, Campus de Beaulieu, Rennes, France,
Yann.Glouche@inria.fr

Bijoy A. Jose FERMAT Lab, Bradley Department of Electrical and Computer Engineering, Virginia Polytechnic Institute and State University, Blacksburg, VA, USA, bijoy@vt.edu

Paul Le Guernic INRIA, Centre de Recherche de Rennes, Campus de Beaulieu, Rennes, France, Paul.LeGuernic@inria.fr

Frédéric Mallet INRIA, Centre de Recherche de Sophia-Antipolis Méditerranée, 06902 Sophia Antipolis, France
and
Laboratoire I3S, UMR 6070 CNRS, Université de Nice-Sophia Antipolis, 06903 Sophia Antipolis Cedex, France, Frederic.Mallet@sophia.inria.fr

Julien Ouy IRISA-ESPRESSO, Campus de Beaulieu., 35042 Rennes Cedex, France, julien.ouy@irisa.fr

Marc Pantel IRIT-ACADIE, Université de Toulouse, site Paul Sabatier, 118 Route de Narbonne, 31062 Toulouse Cedex 9, France, pantel@irit.fr

Dumitru Potop-Butucaru INRIA, Centre de Recherche de Paris-Rocquencourt, Le Chesnay Cedex, France, dumitru.potop@inria.fr

Ana Rugina EADS Astrium, 31 rue des Cosmonautes Z.I. du Palays, 31402 Toulouse Cedex 4, France, Ana-Elena.RUGINA@astrium.eads.net

Sandeep K. Shukla FERMAT Lab, Bradley Department of Electrical and Computer Engineering, Virginia Polytechnic Institute and State University, Blacksburg, VA, USA, shukla@vt.edu

Yves Sorel INRIA, Centre de Recherche de Paris-Rocquencourt, Le Chesnay Cedex, France, yves.sorel@inria.fr

Martin Strecker IRIT-ACADIE, Université de Toulouse, site Paul Sabatier, 118 Route de Narbonne, 31062 Toulouse Cedex 9, France, strecker@irit.fr

Jean-Pierre Talpin INRIA, Centre de Recherche de Rennes, Campus de Beaulieu, Rennes, France, Jean-Pierre.Talpin@inria.fr

Nalini Vasudevan Columbia University, New York, USA,
naliniv@cs.columbia.edu

Acronyms

IFFT	Inverse Discrete Fourier Transform
DMA	Direct Memory Access
FSM	Finite State Machine
RTL	Register Transfer Level
HLS	High Level Synthesis
EDA	Electronic Design Automation
HDL	Hardware Description Language
CAOS	Concurrent Action Oriented Specifications
CDFG	Control Data-Flow Graph
HTG	Hierarchical Task Graph
GCD	Greatest Common Divisor
BSC	Bluespec Compiler
BSV	Bluespec System Verilog
LPM	Longest Prefix Match
VM	Vending Machine
LTL	Linear-time Temporal Logic
TLA	Temporal Logic of Actions
MCS	Maximal Concurrent Schedule
ACS	Alternative Concurrent Schedule
MNS	Maximum Non-conflicting Subset
MIS	Maximum Independent Set
MLS	Minimum Length Schedule
FFD	First Fit Decreasing
PTAS	Polynomial Time Approximation Scheme
AES	Advanced Encryption Standard
UC	Upsize Converter