Software Engineering For Embedded Systems Methods Practical Techniques And Applications Expert Guide

This is the first edition of 'The Engineering of Reliable Embedded Systems': it is released here largely for historical reasons. (Please consider purchasing 'ERES2' instead.) [The second edition will be available for purchase here from June 2017.]

This chapter provides some guidelines that are commonly used in embedded software development. It starts with principles of programming, including readability, testability, and maintainability. The chapter then proceeds with discussing how to start an embedded software project, including considerations for hardware, file organization, and development guidelines. The focus then shifts to programming guidelines that are important to any software development project, which includes the importance of a syntax coding standard. The chapter concludes with descriptions of variables and definitions and how they are typically used in an embedded software project.

Optimization metrics for compiled code are not always measured in resulting execution clock cycles on the target architecture. Consider a modern cellular telephone or wireless device which may download executables over a wireless network connection or backhaul infrastructure. In such cases, it is often advantageous for the compiler to reduce the size of the compiled code which must be downloaded to the wireless device. By reducing the size of the code needed to be downloaded, savings are achieved in terms of bandwidth required for each wireless point of download. Optimization metrics such as the memory system performance of compiled code are other metrics which are often important to developers. These are metrics correlated to the dynamic run-time behavior of not only the compiled code on the target processor, but also the underlying memory system, caches, DRAM and buses, etc. By efficiently arranging the data within the application or, more specifically, the order in which data and corresponding data structures are accessed by the application dynamically at run-time, significant performance improvements can be gained at the memory-system level. In addition, vectorizing compilers can also improve performance due to spatial locality of data when SIMD instruction sets are present and varying memory-system alignment conditions are met.

Nowadays embedded and real-time systems contain complex software. The complexity of embedded systems is increasing, and the amount and variety of software in the embedded products are growing. This creates a big challenge for embedded and real-time software development processes and there is a need to develop separate metrics and benchmarks. “Embedded and Real Time System Development: A Software Engineering Perspective: Concepts, Methods and Principles” presents practical as well as conceptual knowledge of the latest tools, techniques and methodologies of embedded software engineering and real-time systems. Each chapter includes an in-depth investigation regarding the actual or potential role of software engineering tools in the context of the embedded system and real-time system. The book presents state-of-the art and future perspectives with industry experts, researchers, and academicians sharing ideas and experiences including surrounding frontier technologies, breakthroughs, innovative solutions and applications. The book is organized into four parts “Embedded Software Development Process”, “Design Patterns and Development Methodology”, “Modelling Framework” and “Performance Analysis, Power Management and Deployment” with altogether 12 chapters. The book is aiming at (i) undergraduate students and postgraduate students conducting research in the areas of embedded software engineering and real-time systems; (ii) researchers at universities and other institutions working in these fields; and (iii) practitioners in the R&D departments of embedded system. It can be used as an advanced reference for a course taught at the postgraduate level in embedded software engineering and real-time systems.

Extensions of the SPES 2020 Methodology

A Comprehensive Guide for Engineers and Programmers

Embedded Systems – A Hardware-Software Co-Design Approach

Introduction to Embedded Systems

Component-Based Software Development for Embedded Systems

The Open-Source Approach

Offering comprehensive coverage of the convergence of real-time embedded systems scheduling, resource access control, software design and development, and high-level system modeling, analysis and verification Following an introductory overview, Dr. Wang delves into the specifics of hardware components, including processors, memory, I/O devices and architectures, communication structures, peripherals, and characteristics of real-time operating systems. Later chapters are dedicated to real-time task scheduling algorithms and resource access control policies, as well as priority-inversion control and deadlock avoidance. Concurrent system programming and POSIX programming for real-time systems are covered, as are finite state machines and Time Petri nets. Of special interest to software engineers will be the chapter devoted to model checking, in which the author discusses temporal logic and the NuSMV model checking tool, as well as a chapter treating real-time software design with UML. The final portion of the book explores practical issues of software reliability, aging, rejuvenation, security, safety, and power management. In addition, the book: Explains real-time embedded software modeling and design with finite state machines, Petri nets, and UML, and real-time constraints verification with the model checking tool, NuSMV Features real-world examples in finite state machines, model checking, real-time system design with UML, and more Covers embedded computer programing, designing for reliability, and designing for safety Explains how to make engineering trade-offs of power use and performance Investigates practical issues concerning software reliability, aging, rejuvenation, security, and power management Real-Time Embedded Systems is a valuable resource for those responsible for real-time and embedded software design, development, and management. It is also an excellent textbook for graduate courses in computer engineering, computer science, information technology, and software engineering on embedded and real-time software systems, and for undergraduate computer and software engineering courses.

When designing an embedded system, special care must be taken when you design the user interface. For simple devices, simple text, command buttons, and LEDs are adequate. For more complex systems, full graphical user interfaces and touch panels are required. User interface design focuses on the following key areas: (a) the design of interfaces between different software components, (b) the design of interfaces between the software and other nonhuman producers and consumers of information, and (c) the design of the interface between a human and the computer. This chapter will focus on the process, guidelines, human factors and techniques required to design an effective user interface.

An introduction to the engineering principles of embedded systems, with a focus on modeling, design, and analysis of cyber-physical systems. The most visible use of computers and software is processing information for human consumption. The vast majority of computers in use, however, are much less visible. They run the engine, brakes, seatbelts, airbag, and audio system in your car. They digitally encode your voice and construct a radio signal to send it from your cell phone to a base station. They command robots on a factory floor, power generation in a power plant, processes in a chemical plant, and traffic lights in a city. These less visible computers are called embedded systems, and the software they run is called embedded software. The principal challenges in designing and analyzing embedded systems stem from their interaction with physical processes. This book takes a cyber-physical approach to embedded systems, introducing the engineering concepts underlying embedded systems as a technology and as a subject of study. The focus is on modeling, design, and analysis of cyber-physical systems, which integrate computation, networking, and physical processes. The second edition offers two new chapters, several new exercises, and other improvements. The book can be used as a textbook at the advanced undergraduate or introductory graduate level and as a professional reference for practicing engineers and computer scientists. Readers should have some familiarity with machine structures, computer programming, basic discrete mathematics and algorithms, and signals and systems.

A PRACTICAL GUIDE TO HARDWARE FUNDAMENTALS Embedded Systems Hardware for Software Engineers describes the electrical and electronic circuits that are used in embedded systems, their functions, and how they can be interfaced to other devices. Basic computer architecture topics, memory, address decoding techniques, ROM, RAM, DRAM, DDR, cache memory, and memory hierarchy are discussed. The book covers key architectural features of widely used microcontrollers and microprocessors, including Microchip's PIC32, ATMEL's AVR32, and Freescale's MC68000. Interfacing to an embedded system is then described. Data acquisition system level design considerations and a design example are presented with real-world parameters and characteristics. Serial interfaces such as RS-232, RS-485, PC, and USB are addressed and printed circuit boards and high-speed signal propagation over transmission lines are covered with a minimum of math. A brief survey of logic families of integrated circuits and programmable logic devices is also contained in this in-depth resource. COVERAGE INCLUDES: Architecture examples Memory Memory address decoding Read-only memory and other related devices Input and output ports Analog-to-digital and digital-to-analog converters Interfacing to external devices Transmission lines Logic families of integrated circuits and their signaling characteristics The printed circuit board Programmable logic devices Test equipment: oscilloscopes and logic analyzers

The SPES 2020 Methodology

Software Engineering for Embedded Systems, 2nd Edition

Embedded Systems Hardware for Software Engineers

Chapter 20. Managing Embedded Software Development

Chapter 6. Hardware’s Interface to Embedded Software

This chapter explores the unique challenges that limit reuse in embedded systems, and strategies to overcome them. It explores what limits reuse, and traditional approaches to overcome the limitations such as a hardware abstraction layer or an RTOS porting layer. It does not stop there. The shortcomings of layered software drive a desire for highly optimized reusable software components. This chapter introduces the component factory concept: a mechanism that creates reconfigurable and reusable hardware- and RTOS-agnostic components generated by an expert system.

This book provides a comprehensive introduction into the SPES XT modeling framework. Moreover, it shows the applicability of the framework for the development of embedded systems in different industry domains and reports on the lessons learned. It also describes how the SPES XT modeling framework can be tailored to meet domain and project-specific needs. The book is structured into four parts: Part I “Starting Situation” discusses the status quo of the development of embedded systems with specific focus on model-based engineering and summarizes key challenges emerging from industrial practice. Part II “Modeling Theory” introduces the SPES XT modeling framework and explains the core underlying principles. Part III “Application of the SPES XT Framework” describes the application of the SPES XT modeling framework and how it addresses major industrial challenges. Part IV “Evaluation and Technology Transfer” assess the impact of the SPES XT modeling framework and includes various exemplary applications from automation, automotive, and avionics. Overall, the SPES XT modeling framework offers a seamless model-based engineering approach. It addresses core challenges faced during the engineering of embedded systems. Among others, it offers aligned and integrated techniques for the early validation of engineering artefacts (including requirements and functional and technical designs), the management of product variants and their variability, modular safety assurance and deployment of embedded software.

Embedded Software Development: The Open-Source Approach delivers a practical introduction to embedded software development, with a focus on open-source components. This programmer-centric book is written in a way that enables even novice practitioners to grasp the development process as a whole. Incorporating real code fragments and explicit, real-world open-source operating system references (in particular, FreeRTOS) throughout, the text: Defines the role and purpose of embedded systems, describing their internal structure and interfacing with software development tools Examines the inner workings of the GNU compiler collection (GCC)-based software development system or, in other words, toolchain Presents software execution models that can be adopted profitably to model and express concurrency Addresses the basic nomenclature, models, and concepts related to task-based scheduling algorithms Shows how an open-source protocol stack can be integrated in an embedded system and interfaced with other software components Analyzes the main components of the FreeRTOS Application Programming Interface (API), detailing the implementation of key operating system concepts Discusses advanced topics such as formal verification, model checking, runtime checks, memory corruption, security, and dependability Embedded Software Development: The Open-Source Approach capitalizes on the authors’ extensive research on real-time operating systems and communications used in embedded applications, often carried out in strict cooperation with industry. Thus, the book serves as a springboard for further research.

Build complex embedded systems faster and with lower costs by: * Knowing when and how much simulation testing is appropriate * Applying engineering methods to simulation design and development * Using the best tools available to develop simulations. * Va

Chapter 12. Optimizing Embedded Software for Memory

A Cyber-Physical Systems Approach

Chapter 2. Embedded Systems Hardware/Software Co-Development

Embedded Systems Architecture

Advanced Model-Based Engineering of Embedded Systems

Software Engineering for Embedded Systems

When planning the development of modern embedded systems, hardware and software cannot be considered independently. Over the last two decades chip and system complexity has seen an enormous amount of growth, while more and more system functionality has moved from dedicated hardware implementation into software executing on general-purposed embedded processors. By 2010 the development effort for software had outgrown the development efforts for hardware, and the complexity trend continues in favor of software. Traditional design techniques such as independent hardware and software design are being challenged due to heterogeneous models and applications being integrated to create a complex system on chip. Using proper techniques of hardware-software codesign, designers consider the trade-offs in the way hardware and software components of a system work together to exhibit a specified behavior, given a set of performance goals and technology. This chapter will cover these topics.

This tutorial reference takes the reader from use cases to complete architectures for real-time embedded systems using SysML, UML, and MARTE and shows how to apply the COMET/RTE design method to real-world problems. The author covers key topics such as architectural patterns for distributed and hierarchical real-time control and other real-time software architectures, performance analysis of real-time designs using real-time scheduling, and timing analysis on single and multiple processor systems. Complete case studies illustrating design issues include a light rail control system, a microwave oven control system, and an automated highway toll system. Organized as an introduction followed by several self-contained chapters, the book is perfect for experienced software engineers wanting a quick reference at each stage of the analysis, design, and development of large-scale real-time embedded systems, as well as for advanced undergraduate or graduate courses in software engineering, computer engineering, and *software design.*

Authored by two of the leading authorities in the field, this guide offers readers the knowledge and skills needed to achieve proficiency with embedded software.

Embedded systems often have one or more real-time requirements. The complexity of modern embedded software systems requires a systematic approach for achieving these performance targets. An ad hoc process can lead to missed deadlines, poorly performing systems and cancelled projects. There is a maturity required to define, manage, and deliver on multiple real-time performance requirements. Software performance engineering (SPE) is a discipline within the broader systems engineering area that can improve the maturity of the performance engineering process. SPE is a systematic, quantitative approach to constructing software systems that meet performance objectives. SPE is a software-oriented approach; it focuses on architecture, design, and implementation choices. It focuses on the activities, techniques, and deliverables that are applied at every phase of the embedded software development life-cycle, especially responsiveness and scalability, to ensure software is being architected and implemented to meet the performance-related requirements of the system.

Extensions of the SPES Methodology

Embedded and Real Time System Development: A Software Engineering Perspective

Chapter 24. Embedded Software for Networking Applications

An Embedded Software Engineering Toolkit

Real-Time Software Design for Embedded Systems

Model-Based Engineering of Embedded Real-Time Systems

One of the most important considerations in the product life-cycle of an embedded project is to understand and optimize the power consumption of the device. Power consumption is highly visible for hand-held devices which require battery power to be able to guarantee certain minimum usage/idle times between recharging. Other main embedded applications, such as medical equipment, test, measurement, media, and wireless base stations, are very sensitive to power as well – due to the need to manage the heat dissipation of increasingly powerful processors, power supply cost, and energy consumption cost – so the fact is that power consumption cannot be overlooked. The responsibility for setting and keeping power requirements often falls on the shoulders of hardware designers, but the software programmer has the ability to provide a large contribution to power optimization. Often, the impact that the software engineer has on the power consumption of a device is overlooked or underestimated. The goal of this chapter is to discuss how software can be used to optimize power consumption, starting with the basics of what power consumption consists of, how to properly measure power consumption, and then moving on to techniques for minimizing power consumption in software at the algorithmic level, hardware level, and data-flow level. This will include demonstrations of the various techniques and explanations of both how and why certain methods are effective at reducing power so the reader can take and apply this work to their application immediately.

This Expert Guide gives you the techniques and technologies in software engineering to optimally design and implement your embedded system. Written by experts with a solutions focus, this encyclopedic reference gives you an indispensable aid to tackling the day-to-day problems when using software engineering methods to develop your embedded systems. With this book you will learn : The principles of good architecture for an embedded system Design practices to help make your embedded project successful Details on principles that are often a part of embedded systems, including digital signal processing, safety-critical principles, and development processes Techniques for setting up a performance engineering strategy for your embedded system software How to develop user interfaces for embedded systems Strategies for testing and deploying your embedded system, and ensuring quality development processes Practical techniques for optimizing embedded software for performance, memory, and power Advanced guidelines for developing multicore software for embedded systems How to develop embedded software for networking, storage, and automotive segments How to manage the embedded development process Includes contributions from: Frank Schirrmeister, Shelly Gretlein, Bruce Douglass, Erich Styger, Gary Stringham, Jean Labrosse, Jim Trudeau, Mike Brogioli, Mark Pitchford, Catalin Dan Udma, Markus Levy, Pete Wilson, Whit Waldo, Inga Harris, Xinxin Yang, Srinivasa Addepalli, Andrew McKay, Mark Kraeling and Robert Oshana. Road map of key problems/issues and references to their solution in the text Review of core methods in the context of how to apply them Examples demonstrating timeless implementation details Short and to- the-point case studies show how key ideas can be implemented, the rationale for choices made, and design guidelines and trade-offs.

This Expert Guide gives you the techniques and technologies in software engineering to optimally design and implement your embedded system. Written by experts with a solutions focus, this encyclopedic reference gives you an indispensable aid to tackling the day-to-day problems when using software engineering methods to develop your embedded systems. With this book you will learn: The principles of good architecture for an embedded system Design practices to help make your embedded project successful Details on principles that are often a part of embedded systems, including digital signal processing, safety-critical principles, and development processes Techniques for setting up a performance engineering strategy for your embedded system software How to develop user interfaces for embedded systems Strategies for testing and deploying your embedded system, and ensuring quality development processes Practical techniques for optimizing embedded software for performance, memory, and power Advanced guidelines for developing multicore software for embedded systems How to develop embedded software for networking, storage, and automotive segments How to manage the embedded development process Includes contributions from: Frank Schirrmeister, Shelly Gretlein, Bruce Douglass, Erich Styger, Gary Stringham, Jean Labrosse, Jim Trudeau, Mike Brogioli, Mark Pitchford, Catalin Dan Udma, Markus Levy, Pete Wilson, Whit Waldo, Inga Harris, Xinxin Yang, Srinivasa Addepalli, Andrew McKay, Mark Kraeling and Robert Oshana. Road map of key problems/issues

and references to their solution in the text Review of core methods in the context of how to apply them Examples demonstrating timeless implementation details Short and to- the-point case studies show how key ideas can be implemented, the rationale for choices made, and design guidelines and trade-offs

Embedded Systems Architecture is a practical and technical guide to understanding the components that make up an embedded system's architecture. This book is perfect for those starting out as technical professionals such as engineers, programmers and designers of embedded systems; and also for students of computer science, computer engineering and electrical engineering. It gives a much-needed 'big picture' for recently graduated engineers grappling with understanding the design of real-world systems for the first time, and provides professionals with a systems-level picture of the key elements that can go into an embedded design, providing a firm foundation on which to build their skills. Real-world approach to the fundamentals, as well as the design and architecture process, makes this book a popular reference for the daunted or the inexperienced: if in doubt, the answer is in here! Fully updated with new coverage of FPGAs, testing, middleware and the latest programming techniques in C, plus complete source code and sample code, reference designs and tools online make this the complete package Visit the companion web site at http://booksite.elsevier.com/9780123821966/ for source code, design examples, data sheets and more A true introductory book, provides a comprehensive get up and running reference for those new to the field, and updating skills: assumes no prior knowledge beyond undergrad level electrical engineering Addresses the needs of practicing engineers, enabling it to get to the point more directly, and cover more ground. Covers hardware, software and middleware in a single volume Includes a library of design examples and design tools, plus a complete set of source code and embedded systems design tutorial materials from companion website

Model-Based Engineering of Collaborative Embedded Systems

Design Principles and Engineering Practices

Chapter 22. Embedded Software for Automotive Applications

International Dagstuhl Workshop, Dagstuhl Castle, Germany, November 4-9, 2007. Revised Selected Papers

Chapter 1. Software Engineering of Embedded and Real-Time Systems

Chapter 14. Human Factors and User Interface Design for Embedded Systems

A recent survey stated that 52% of embedded projects are late by 4-5 months. This book can help get those projects in on-time with design patterns. The author carefully takes into account the special concerns found in designing and developing embedded applications specifically concurrency, communication, speed, and memory usage. Patterns are given in UML (Unified Modeling Language) with examples including ANSI C for direct and practical application to C code. A basic C knowledge is a prerequisite for the book while UML notation and terminology is included. General C programming books do not include discussion of the contraints found within embedded system design. The practical examples give the reader an understanding of the use of UML and OO (Object Oriented) designs in a resource-limited environment. Also included are two chapters on state machines. The beauty of this book is that it can help you today. . Design Patterns within these pages are immediately applicable to your project Addresses embedded system design concerns such as concurrency, communication, and memory usage Examples contain ANSI C for ease of use with C programming code

This chapter introduces the automotive system, which is unlike any other, characterized by its rigorous planning, architecting, development, testing, validation and verification. The physical task of writing embedded software for automotive applications versus other application areas is not significantly different from other embedded systems, but the key differences are the quality standards which must be followed for any development and test project. To write automotive software the engineer needs to understand how and why the systems have evolved into the complex environment it is today. They must be aware of the differences and commonalties between the automotive submarkets. They must be familiar with the applicable quality standards and why such strict quality controls exist, along with how quality is tested and measured, all of which are described in this chapter with examples of the most common practices. This chapter introduces various processes to help software engineers write high-quality, fault-tolerant, interoperable code such as modeling, autocoding and advanced trace and debug assisted by the emergence of the latest AUTOSAR and ISO26262 standards, as well as more traditional standards such as AEC, OBD-II and MISRA.

Thetopicof"Model-BasedEngineeringofReal-TimeEmbeddedSystems"brings together a challenging problem domain (real-time embedded systems) and a - lution domain (model-based engineering). It is also at the forefrontof integrated software and systems engineering, as software in this problem domain is an essential tool for system implementation and integration. Today, real-time - bedded software plays a crucial role in most advanced technical systems such as airplanes, mobile phones, and cars, and has become the main driver and - cilitator for innovation. Development, evolution, veri?cation, con?guration, and maintenance of embedded and distributed software nowadays are often serious challenges as drastic increases in complexity can be observed in practice. Model-based engineering in general, and model-based software development in particular, advocates the notion of using models throughout the development and life-cycle of an engineered system. Model-based software engineering re- forces this notion by promoting models not only as the tool of abstraction, but also as the tool for veri?cation, implementation, testing, and maintenance. The application of such model-based engineering techniques to embedded real-time systems appears to be a good candidate to tackle some of the problems arising in the problem domain.

This textbook introduces the concept of embedded systems with exercises using Arduino Uno. It is intended for advanced undergraduate and graduate students in computer science, computer engineering, and electrical engineering programs. It contains a balanced discussion on both hardware and software related to embedded systems, with a focus on co-design aspects. Embedded systems have applications in Internet-of-Things (IoT), wearables, self-driving cars, smart devices, cyberphysical systems, drones, and robotics. The hardware chapter discusses various microcontrollers (including popular microcontroller hardware examples), sensors, amplifiers, filters, actuators, wired and wireless communication topologies, schematic and PCB designs, and much more. The software chapter describes OS-less programming, bitmath, polling, interrupt, timer, sleep modes, direct memory access, shared memory, mutex, and smart algorithms, with lots of C-code examples for Arduino Uno. Other topics discussed are prototyping, testing, verification, reliability, optimization, and regulations. Appropriate for courses on embedded systems, microcontrollers, and instrumentation, this textbook teaches budding embedded system programmers practical skills with fun projects to prepare them for industry products. Introduces embedded systems for wearables, Internet-of-Things (IoT), robotics, and other smart devices; Offers a balanced focus on both hardware and software co-design of embedded systems; Includes exercises, tutorials, and assignments.

Making Embedded Systems

Simulation Engineering

Concepts, Methods and Principles

Chapter 3. Software Modeling for Embedded Systems

Chapter 15. Embedded Software Quality, Integration and Testing Techniques

Design Patterns for Embedded Systems in C

This book integrates new ideas and topics from real time systems, embedded systems, and software engineering to give a complete picture of the whole process of developing software for real-time embedded applications. You will not only gain a thorough understanding of concepts related to microprocessors, interrupts, and system boot process, appreciating the importance of real-time modeling and scheduling, but you will also learn software engineering practices such as model documentation, model analysis, design patterns, and standard conformance. This book is split into four parts to help you learn the key concept of embedded systems; Part one introduces the development process, and includes two chapters on microprocessors and interrupts---fundamental topics for software engineers; Part two is dedicated to modeling techniques for real-time systems; Part three looks at the design of software architectures and Part four covers software implementations, with a focus on POSIX-compliant operating systems. With this book you will learn: The pros and cons of different architectures for embedded systems POSIX real-time extensions, and how to develop POSIX-compliant real time applications How to use real-time UML to document system designs with timing constraints The challenges and concepts related to cross-development Multitasking design and inter-task communication techniques (shared memory objects, message queues, pipes, signals) How to use kernel objects (e.g. Semaphores, Mutex, Condition variables) to address resource sharing issues in RTOS applications The philosophy underpinning the notion of "resource manager" and how to implement a virtual file system using a resource manager The key principles of real-time scheduling and several key algorithms Coverage of the latest UML standard (UML 2.4) Over 20 design patterns which represent the best practices for reuse in a wide range of real-time embedded systems Example codes which have been tested in QNX---a real-time operating system widely adopted in industry

Embedded systems have long become essential in application areas in which human control is impossible or infeasible. The development of modern embedded systems is becoming increasingly difficult and challenging because of their overall system complexity, their tighter and cross-functional integration, the increasing requirements concerning safety and real-time behavior, and the need to reduce development and operation costs. This book provides a comprehensive overview of the Software Platform Embedded Systems (SPES) modeling framework and demonstrates its applicability in embedded system development in various industry domains such as automation, automotive, avionics, energy, and healthcare. In SPES 2020, twenty-one partners from academia and industry have joined forces in order to develop and evaluate in different industrial domains a modeling framework that reflects the current state of the art in embedded systems engineering. The content of this book is structured in four parts. Part I “Starting Point” discusses the status quo of embedded systems development and model-based engineering, and summarizes the key requirements faced when developing embedded systems in different application domains. Part II “The SPES Modeling Framework” describes the SPES modeling framework. Part III “Application and Evaluation of the SPES Modeling Framework” reports on the validation steps taken to ensure that the framework met the requirements discussed in Part I. Finally, Part IV “Impact of the SPES Modeling Framework” summarizes the results achieved and provides an outlook on future work. The book is mainly aimed at professionals and practitioners who deal with the development of embedded systems on a daily basis. Researchers in academia and industry may use it as a compendium for the requirements and state-of-the-art solution concepts for embedded systems development.

Interested in developing embedded systems? Since they don’t tolerate inefficiency, these systems require a disciplined approach to programming. This easy-to-read guide helps you cultivate a host of good development practices, based on classic software design patterns and new patterns unique to embedded programming. Learn how to build system architecture for processors, not operating systems, and discover specific techniques for dealing with hardware difficulties and manufacturing requirements. Written by an expert who’s created embedded systems ranging from urban surveillance and DNA scanners to children’s toys, this book is ideal for intermediate and experienced programmers, no matter what platform you use. Optimize your system to reduce cost and increase performance Develop an architecture that makes your software robust in resource-constrained environments Explore sensors, motors, and other I/O devices Do more with less: reduce RAM consumption, code space, processor cycles, and power consumption Learn how to update embedded code directly in the processor Discover how to implement complex mathematics on small processors Understand what interviewers look for when you apply for an embedded systems job "Making Embedded Systems is the book for a C programmer who wants to enter the fun (and lucrative) world of embedded systems. It’s very well written—entertaining, even—and filled with clear illustrations." —Jack Ganssle, author and embedded system expert.

An embedded system is a computer system designed for a specific function within a larger system, and often has one or more real-time computing constraints. It is embedded as part of a larger device which can include hardware and mechanical parts. This is in stark contrast to a general-purpose computer, which is designed to be flexible and meet a wide range of end-user needs. The methods, techniques, and tools for developing software systems that were successfully applied to general purpose computing are not as readily applicable to embedded computing. Software systems running on networks of mobile, embedded devices must exhibit properties that are not always required of more traditional systems such as near-optimal performance, robustness, distribution, dynamism, and mobility. This chapter will examine the key properties of software systems in the embedded, resource-constrained, mobile, and highly distributed world. The applicability of mainstream software engineering methods is assessed and techniques (e.g., software design, component-based development, software architecture, system integration and test) are also discussed in the context of this domain. This chapter will overview embedded and real-time systems.

Build Better Embedded Systems Faster

An Overview of Current Research Trends
Real-Time Embedded Systems
Chapter 9. Software Reuse By Design in Embedded Systems
Chapter 13. Optimizing Embedded Software for Power
Chapter 7. Embedded Software Programming and Implementation Guidelines

Software Engineering for Embedded Systems: Methods, Practical Techniques, and Applications, Second Edition provides the techniques and technologies in software engineering to optimally design and implement an embedded system. Written by experts with a solution focus, this encyclopedic reference gives an indispensable aid on how to tackle the day-to-day problems encountered when using software engineering methods to develop embedded systems. New sections cover peripheral programming, Internet of things, security and cryptography, networking and packet processing, and hands on labs. Users will learn about the principles of good architecture for an embedded system, design practices, details on principles, and much more. Provides a roadmap of key problems/issues and references to their solution in the text Reviews core methods and how to apply them Contains examples that demonstrate timeless implementation details Users case studies to show how key ideas can be implemented, the rationale for choices made, and design guidelines and trade-offs.

The software architecture of embedded computing systems is a depiction of the system as a set of structures that aids in the reasoning and understanding of how the system will behave. Software architecture acts as the blueprint for the system as well as the project developing it. The architecture is the primary framework of important embedded system qualities such as performance, modifiability, and security, none of which can be achieved without a unifying architectural vision. Architecture is an artifact for early analysis to ensure that a design approach will lead to an acceptable system. This chapter will discuss the details of these aspects of embedded software architectures.

Software Engineering for Embedded Systems: Methods, Practical Techniques, and Applications, Second Edition provides the techniques and technologies in software engineering to optimally design and implement an embedded system. Written by experts with a solution focus, this encyclopedic reference gives an indispensable aid on how to tackle the day-to-day problems encountered when using software engineering methods to develop embedded systems. New sections cover peripheral programming, Internet of things, security and cryptography, networking and packet processing, and hands on labs. Users will learn about the principles of good architecture for an embedded system, design practices, details on principles, and much more. Provides a roadmap of key problems/issues and references to their solution in the text Reviews core methods and how to apply them Contains examples that demonstrate timeless implementation details Users case studies to show how key ideas can be implemented, the rationale for choices made, and design guidelines and trade-offs

State of the art techniques and best practices in the development of embedded software apply not only to high-integrity devices (such as those for safety-critical applications like aircraft flight controllers, car braking systems or medical devices), but also to lesser-integrity applications when the need to optimize the effectiveness of the available test time and budget demands that pragmatic decisions should be made. To complement this multitude of software test techniques there is a similar plethora of test tools available to automate them. These tools are commonplace in the development of safety-critical applications, but elsewhere not everyone has the budget to buy all, or indeed any, of them. Of course, the providers of these tools would advocate the purchase of each and every one of them, so how can a limited budget best be allocated? And where no budget exists, how can similar principles be applied to provide confidence that the finished item is of adequate quality? In addressing these issues not only are the concepts behind the techniques presented, but also some “case study” software code examples to drill a little deeper and illustrate how some of them are implemented in practice.

Design Patterns for Great Software
Embedded Systems and Software Validation
Unleash the Power of Arduino!
The Engineering of Reliable Embedded Systems (LPC1769)
Methods, Practical Techniques, and Applications
Model-Based Engineering of Embedded Systems
Software Engineering for Embedded SystemsMethods, Practical Techniques, and ApplicationsNewnes

This book provides a good opportunity for software engineering practitioners and researchers to get in sync with the current state-of-the-art and future trends in component-based embedded software research. The book is based on a selective compilation of papers that cover the complete component-based embedded software spectrum, ranging from methodology to tools. Methodology aspects covered by the book include functional and non-functional specification, validation, verification, and component architecture. As tools are a critical success factor in the transfer from academia-generated knowledge to industry-ready technology, an important part of the book is devoted to tools. This state-of-the-art survey contains 16 carefully selected papers organised in topical sections on specification and verification, component compatibility, component architectures, implementation and tool support, as well as non-functional properties.

This chapter provides information to successfully organize and manage any embedded software project or program. It introduces quality systems, the OSI model of architecting software into stacks, several software development models and ways in which teams may be organized, and overviews communications. Managing the constraints of scope, schedule, costs including resources, quality, and customer satisfaction fully addresses all the work and activities of any project or program. The natural progression of software development from its concept through its life-cycle until release is discussed. Tools are presented for successful planning and execution of resource management, risk management, problem solving, and the traceability of work extending from requirements to respective engineering responses to testing against those software specifications.

Embedded networking applications are changing and evolving quickly. Embedded multicore technology, for example, is appearing not only in high-end networking applications, but even in mid- and low-end networking applications. Achieving networking performance is only possible if software takes advantage of multiple cores. Multicore programming is not as simple as single-core programming. A new mindset is required, from architecting, designing to coding. Networking application development in multicore SoCs should not only concentrate on achieving scalable performance, but should also ease development and result in software that is maintainable for a long time. Some of the programming techniques listed in this chapter should help in achieving this goal.

Chapter 10. Software Performance Engineering for Embedded Systems

Programming Embedded Systems

With C and GNU Development Tools

Embedded Software Development

Chapter 4. Software Design Architecture and Patterns for Embedded Systems

Creating a model for your embedded system provides a time- and cost-effective approach to the development of simple or incredibly complex dynamic control systems, all based on a single model maintained in a tightly integrated software suite. Using modern modeling software tools you can design and perform initial validation in off-line simulation. These models then form the basis for all subsequent development stages. Creating models for your embedded design provides numerous advantages over the traditional design approach. Using this approach – combined with hardware prototyping – you reduce the risk of mistakes and shorten the development cycle by performing verification and validation testing throughout the development instead of only during the final testing stage. Design evaluations and predictions can be made much more quickly and reliably with a system model as a basis. This iterative approach results in improved designs, in terms of both performance and reliability. The cost of resources is reduced, because of reusability of models between design teams, design stages, and various projects and the reduced dependency on physical prototypes. Development errors and overhead can be reduced through the use of automatic code-generation techniques. These advantages translate to more accurate and robust control designs, shorter time to market, and reduced design cost.

This chapter discusses the interface that hardware provides for the embedded software. It discusses the registers and interrupts that provide that interface. But there is more; there are the human aspects of getting the hardware team and the embedded software team to collaborate on the project. Collaboration is needed during the design phase, the co-development phase, the integration phase, and the debugging phase and this chapter discusses those concepts. Several hardware design aspects are discussed that improve the quality of the product and software design aspects are discussed to help support hardware versions.

Modern embedded systems require high performance, low cost and low power consumption. Such systems typically consist of a heterogeneous collection of processors, specialized memory subsystems, and partially programmable or fixed-function components. This heterogeneity, coupled with issues such as hardware/software partitioning, mapping, scheduling, etc., leads to a large number of design possibilities, making performance debugging and validation of such systems a difficult problem. Embedded systems are used to control safety critical applications such as flight control, automotive electronics and healthcare monitoring. Clearly, developing reliable software/systems for such applications is of utmost importance. This book describes a host of debugging and verification methods which can help to achieve this goal. Covers the major abstraction levels of embedded systems design, starting from software analysis and micro-architectural modeling, to modeling of resource sharing and communication at the system level Integrates formal techniques of validation for hardware/software with debugging and validation of embedded system design flows Includes practical case studies to answer the questions: does a design meet its requirements, if not, then which parts of the system are responsible for the violation, and once they are identified, then how should the design be suitably modified?

This Open Access book presents the results of the "Collaborative Embedded Systems" (CrESt) project, aimed at adapting and complementing the methodology underlying modeling techniques developed to cope with the challenges of the dynamic structures of collaborative embedded systems (CESs) based on the SPES development methodology. In order to manage the high complexity of the individual systems and the dynamically formed interaction structures at runtime, advanced and powerful development methods are

required that extend the current state of the art in the development of embedded systems and cyber-physical systems. The methodological contributions of the project support the effective and efficient development of CESs in dynamic and uncertain contexts, with special emphasis on the reliability and variability of individual systems and the creation of networks of such systems at runtime. The project was funded by the German Federal Ministry of Education and Research (BMBF), and the case studies are therefore selected from areas that are highly relevant for Germany’s economy (automotive, industrial production, power generation, and robotics). It also supports the digitalization of complex and transformable industrial plants in the context of the German government's "Industry 4.0" initiative, and the project results provide a solid foundation for implementing the German government's high-tech strategy "Innovations for Germany" in the coming years.