Software Engineering For Embedded Systems Chapter 11 Optimizing Embedded Software For Performance

Design Patterns for Embedded Systems in C
Chapter 21. Agile Development for Embedded Systems
Chapter 17. Multicore Software Development for Embedded Systems: This Chapter draws on Material from the Multicore Programming Practices Guide (MPP) from the Multicore Association
Simulation Engineering
Chapter 18. Safety-Critical Software Development
The POLIS Approach
Embedded Systems Architecture
Software Engineering for Embedded Systems
Unleash the Power of Arduino!
Chapter 2. Embedded Systems Hardware/Software Co-Development
A Cyber-Physical Systems Approach
The Open-Source Approach
Chapter 20. Managing Embedded Software Development
Making Embedded Systems
Software Engineering for Embedded Systems
Introduction to Embedded Systems
Chapter 7. Embedded Software Programming and Implementation Guidelines
Chapter 14. Human Factors and User Interface Design for Embedded Systems
Build Better Embedded Systems Faster
Embedded Software Development with C
Chapter 9. Software Reuse By Design in Embedded Systems
Software Engineering for Embedded Systems
Real-Time Embedded Systems
Chapter 13. Optimizing Embedded Software for Power
Chapter 24. Embedded Software for Networking Applications
Software Engineering for Embedded Systems, 2nd Edition
Software Engineering for Embedded Systems
Software Engineering for Embedded Systems
Hardware-Software Co-Design of Embedded Systems
Chapter 6. Hardware's Interface to Embedded Software
Software Engineering for Embedded Systems
Software Engineering for Embedded Systems
Real-Time Embedded Systems
Software Engineering for Embedded Systems
A Comprehensive Guide for Engineers and Programmers
Embedded Systems Hardware for Software Engineers
With C and GNU Development Tools
Software Engineering for Embedded Systems
Software Engineering for Embedded Systems

SHEPPARD JAMARI

Design Patterns for Embedded Systems in C CRC Press
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
Chapter 21. Agile Development for Embedded Systems Elsevier Inc. Chapters
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.

Chapter 17. Multicore Software Development for Embedded Systems: This Chapter draws on Material from the Multicore Programming Practices Guide (MPP) from the Multicore Association Newnes

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

Simulation Engineering Elsevier Inc. Chapters

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.

Chapter 18. Safety-Critical Software Development Springer Science & Business Media

Embedded Software Development With C offers both an effectual reference for professionals and researchers, and a valuable learning tool for students by laying the groundwork for a solid foundation in the hardware and software aspects of embedded systems development. Key features include a resource for the fundamentals of embedded systems design and development with an emphasis on software, an exploration of the 8051 microcontroller as it pertains to embedded systems, comprehensive tutorial materials for instructors to provide students with labs of varying lengths and levels of difficulty, and supporting website including all sample codes, software tools and links to additional online references.

Elsevier Inc. Chapters

Agile software development is a set of software development techniques based on iterative development. Requirements and software systems evolve through collaboration between self-organizing, cross-functional teams. Agile development supports adaptive planning, evolutionary development and delivery, and a time-boxed iterative approach. The goal of agile is rapid and flexible response to change. Agile is a conceptual framework which promotes interactions throughout the development cycle. Applying agile to embedded software projects introduces some unique challenges, such as more difficulty effectively testing evolving software features, because the corresponding hardware may not be available in time, less freedom to make changes, due to the fact that the corresponding hardware change may have an unacceptably high cost, and less ability for "learn as you go" approaches, considering the hardware construction may demand a more upfront style of planning and design. This chapter will introduce agile software development and show how to apply these techniques to an embedded system.

The POLIS Approach Springer Nature

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

Embedded Systems Architecture MIT Press

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.

Software Engineering for Embedded Systems Elsevier Inc. Chapters

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.

Unleash the Power of Arduino! Software Engineering for Embedded SystemsMethods, Practical Techniques, and Applications

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.

Chapter 2. Embedded Systems Hardware/Software Co-Development Mcgraw-hill

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.

A Cyber-Physical Systems Approach Elsevier Inc. Chapters

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 Open-Source Approach Newnes

Code optimization is a critical step in the development process as it directly impacts the ability of the system to do its intended job. Code that executes faster means more channels, more work performed and competitive advantage. Code that executes in less memory enables more application features to fit into the cell phone. Code that executes with less overall power consumption increases battery life or reduces money spent on powering a base station. This chapter is intended to help programmers write the most efficient code possible, whether that is measured in processor cycles, memory, or power. It starts with an introduction to using the tool chain, covers the importance of knowing the embedded architecture before optimization, then moves on to cover a wide range of optimization techniques. Techniques are presented which are valid on all programmable architectures – C-language optimization techniques and general loop transformations. Real-world examples are presented throughout.

Chapter 20. Managing Embedded Software Development Elsevier Inc. Chapters

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.

Making Embedded Systems Elsevier Inc. Chapters

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.

Software Engineering for Embedded Systems "O'Reilly Media, Inc."

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

Introduction to Embedded Systems Elsevier Inc. Chapters

Embedded systems are informally defined as a collection of programmable parts surrounded by ASICs and other standard components, that interact continuously with an environment through sensors and actuators. The programmable parts include micro-controllers and Digital Signal Processors (DSPs). Embedded systems are often used in life-critical situations, where reliability and safety are more important criteria than performance. Today, embedded systems are designed with an ad hoc approach that is heavily based on earlier experience with similar products and on manual design. Use of higher-level languages such as C helps structure the design somewhat, but with increasing complexity it is not sufficient. Formal verification and automatic synthesis of implementations are the surest ways to guarantee safety. Thus, the POLIS system which is a co-design environment for embedded systems is based on a formal model of computation. POLIS was initiated in 1988 as a research project at the University of California at Berkeley and, over the years, grew into a full design methodology with a software system supporting it. Hardware-Software Co-Design of Embedded Systems: The POLIS Approach is intended to give a complete overview of the POLIS system including its formal and algorithmic aspects. Hardware-Software Co-Design of Embedded Systems: The POLIS Approach will be of interest to embedded system designers (automotive electronics, consumer electronics and telecommunications), micro-controller designers, CAD developers and students.

Chapter 7. Embedded Software Programming and Implementation Guidelines Elsevier Inc. Chapters

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.

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

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.

Build Better Embedded Systems Faster Cambridge University Press

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.