An Introduction to Embedded Systems

Florian Lechner, Daniel Walter
csad5478@uibk.ac.at, csae8958@uibk.ac.at

November 8, 2006

Abstract

This text provides a brief introduction to the wide field of embedded systems. It covers the history and the main aspects of hard- and software design for embedded systems. The basic concepts of synthesis and automated verification are introduced and a short overview of well-known metrics, which are used to describe the economical and technical attributes of a system, is provided. Additionally the differences between commonly used operating systems are discussed.

1 Introduction

Embedded systems are computers which are part of special-purpose devices. Due to the limited duties this systems can be highly optimized to the particular needs. Traditionally most of this systems are used for control and process measurement, as a side-effect of higher integration of integrated circuits more complex applications can be solved by embedded systems. To be able to solve this problems embedded systems are commonly equipped with various kinds of peripherals.

Early applications of embedded devices include the guidance computer of the Minuteman I missiles and the Apollo guidance computer. The Minuteman I & II missiles are intercontinental ballistic nuclear warheads, produced by Boeing in the 1960's. Due to the large quantities of ICs used in the guidance system of Minuteman II missiles, prices for ICs fell from 1000$ each to 3$ each. This lead to wide adoption of embedded systems in consumer-electronics in the 1980's.

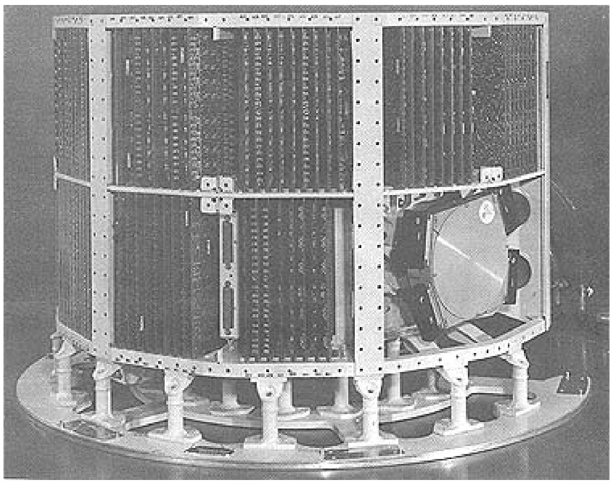
Figure 1: Minuteman I Guidance System

Nowadays embedded systems can be found in devices from digital watches to traffic-control systems. The broad range of applications with totally different requirements lead to various implementation approaches. The range of hardware used in embedded systems reaches from FPGAs to full blown desktop CPUs which are accompanied by special-purpose ICs such as DSPs.

On the software side, depending on the needs, everything, from logic fully implemented in hardware, to systems with own operating system and different applications running on it, can be found.

2 Metrics for Embedded Systems

In order to be able to compare different designs and approaches, there need to be pre-defined, system independent, metrics. These can either be technical specifications or economical criteria.

2.1 Technical Metrics

Technical metrics are mainly used to compare technical designs and specifications of embedded devices or to determine if technical requirements have been fulfilled.

Performance describes the execution time or throughput of the system.

Energy Efficiency is an indicator for the amount of power consumed by the device.

Size as a metric is used if there are constraints for physical size (eg: pacemaker)

Flexibility is a metric for ease of reconfiguration and reusability.

2.2 Economical Metrics

Economical metrics are mostly used to determine which COTS should be use or if the systems will be brought into the market.

Unit Cost describes the monetary cost if manufacturing each copy of the system, excluding NRE cost.

Non-Recurring Engineering (NRE) are the one-time monetary cost of designing the system.

Flexibility in a economical sense describes the ability to change the functionality of the system without incurring heavy NRE cost.

Time to Market indicates the amount of time to develop a system to the point that it can be released and sold to customers.

3 Hardware Platforms

Based on the metrics, introduced in the above section, processors for embedded systems can be distinguished by the grade of customization they grant and the performance they achieve.

3.1 Standard General Purpose Processors

Standard general purpose processors (SGPP) are carefully designed and offer a maximum of flexibility to the designer. Programming SGPPs can be done in nearly every high-level language or assembly language and requires very little knowledge of the system architecture.

As SGPPs are manufactured to high numbers, NRE is spread upon many units. Nevertheless SGPPs are more expensive then other solutions like FPGAs or single purpose processors, when used in products with a large number of selling units.

As they are produced to work in a broad range of environments they are not designed to be energy efficient nor high-performance for specific applications.

Examples for standard general purpose processors are:

- Motorola ARM
- Atmel AVR
- Microchip PIC
- Intel Pentium-(I/II/III/IV)-Series
- AMD Athlon (or other)
- VIA EDEN

3.2 Standard Single Purpose Processors (SSPP)

Standard single purpose processors, sometimes called peripherals, are "off-the-shelf" pre-designed processors, optimized for a single task, such as digital signal processing, analog to digital conversion, timing, etc.

SSPPs are manufactured in high quantities, so NRE is spread upon many units. The total costs per SSPP unit are lower than for custom single purpose processors.

3.3 Custom Single Purpose Processors (CSPP)

As the name suggests, custom single purpose processors are designed for a very specific task. This implies, less flexibility, longer time-to-market and high costs. On the other hand CSPP can be designed to be very small, fast and power-efficient.

Examples for such CSPP are FPGAs or more general PLDs.

3.4 Application Specific Instruction-Set Processors (ASIP)

ASIPs are basically standard general purpose processors which are extended by domain-specific instructions. This allows domain-relevant tasks to be performed highly optimized, while keeping the flexibility of general purpose processors.

4 Specification/Design of Embedded Systems

When designing an embedded system, usually, the first step is to specify the intended or required functionality. This is mostly be done using natural language, after the functionality is specified it is formalized in some sort of definition language such as VHDL or Verilog. Subsequently the resulting design is converted into hardware or software components which are then implemented.

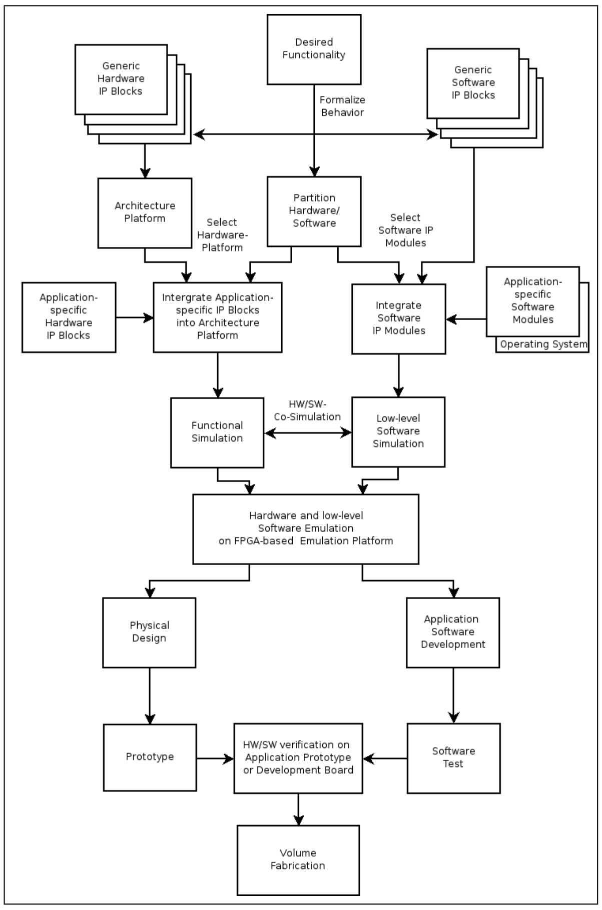
Desired Functionality
Generic Hardware IP Blocks
Formalize Behavior
Generic Software IP Blocks
Architecture Platform
Select Hardware-Platform
Partition Hardware/ Software
Select Software IP Modules
Application-specific Hardware IP Blocks
Intergrate Application-specific IP Blocks into Architecture Platform
Integrate Software IP Modules
Application-specific Software Modules
Operating System
Functional Simulation
HW/SW-Co-Simulation
Low-level Software Simulation
Hardware and low-level Software Emulation on FPGA-based Emulation Platform
Physical Design
Application Software Development
Prototype
HW/SW verification on Application Prototype or Development Board
Software Test
Volume Fabrication

Figure 2: Designprocess for a embedded system

Abstraction

The Gajski-Kuhn Y-chart (Figure 3) visualizes the different layers of abstraction, which can be used in the design process. This abstraction allows the designer to focus on different aspects of the behavioral, structural and geometrical design of the embedded system.

Automation

Nowadays, there is software which can automate some of the steps mentioned above. Especially the process of transforming a formalized description to an actual implementation in hard- or software is commonly automated. This proofs to be challenging due to it's inherently high complexity. Furthermore optimization of the given design metrics must be taken into account when implementing formal descriptions into actual hard- or software. To counter this problems, the whole system is divided into smaller parts, which can be transformed using algorithms which are based on complex heuristics.

As the quality of the implementation heavily depends on the quality of the algorithms used to produce it, software for automated implementation of formalized designs usually costs in the range from 100$ up to several 10000$.

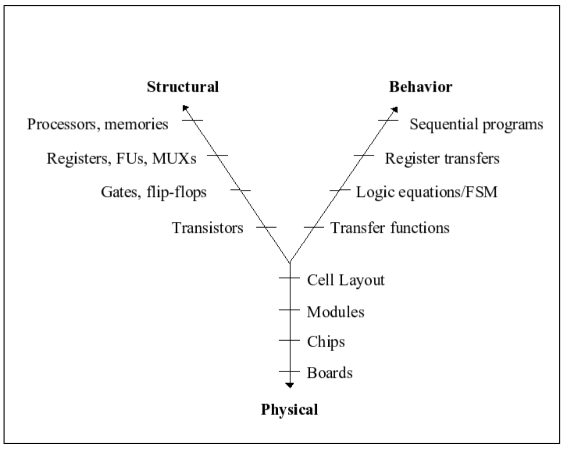

Figure 3: Gajski-Kuhn Y-Chart

Verification

In the verification process, correctness and completeness of a design are checked. Therefor definitions for correctness and completeness need to be defined.

Correctness is the right implementation of the given hardware specifications.

Completeness defines that each input signal results in an appropriate output signal.

For small designs or to verify only certain key features, formal verification is a possibility. For larger or complex systems it is very hard or results in higher costs. So usually such systems are checked by simulation instead of formal verification. Simulations provide extended control of the design and allow the designer to observe the processes which are usually running within a chip. Therefor simulations can be used to increase the confidence in the correctness and completeness of a design but cannot prove either. It is also a commonly used tool to debug the systems behavior.

Reuse

To reduce design-costs and time-to-market commercial off-the-shelf components (COTS) are used wherever possible. This COTS also reduce debugging time, because they are sold as predesigned, prepackaged components or as intellectual property (as definitions in any hardware description language like VHDL).

Summary

As circuit capacities are constantly growing industry seeks improve the productivity of designers. To accomplish this, tools for synthesis (automated conversion of formalized descriptions of system behavior) and verification are developed. Besides these tools there exists the possibility to license complete chip architectures as intellectual property. This allows for the use of tested components which can be customized or extended to the actual needs, thereby reducing the required testing and time-to-market.

Increased IC capacity means that traditionally hardware/software components can coexist on on chip and allow the designer to implement speed relevant functions as hardware or realized complex hardware in software to reduce the design costs.

5 Programming Embedded Systems

Unlike personal computers, embedded systems usually aren't programmed on the platform the program is intended to run. This requires special toolchains with cross-compilers and emulators to test the code before deploying it to the target platform.

Depending on the applications needs, there are different approaches to implement the software. One possibility is to directly control the hardware out of the program. This is the approach with the highest performance, but requires more knowledge about the used architecture and peripherals than using an operating system. On the other hand operating systems provide functionalities for multiprocessing and allow the designer to develop mostly independent from the underlying architecture.

5.1 Operating Systems for Embedded Devices

Operating Systems (for embedded systems) are classified into Real-Time Operating Systems and Non-Real-Time OS.

Real Time Operating Systems are operating systems which guarantee responses to each event within a defined amount of time. This type of operating system is mainly used by time-critical applications such as measurement and control systems. Some commonly used RTOS for embedded systems are: VxWorks, OS-9, Symbian, RTLinux.

Non-Real Time Operating Systems do not guarantee defined response times. Those systems are mostly used if multiple applications are needed. Windows CE and PalmOS are examples for such embedded operating systems.

5.2 Applications for Embedded Systems

Depending on the intended task of the device an embedded system is used in, applications can range from simple measurement duties to full-blown word-processors.

6 Conclusion

As integrated circuits get continuously cheaper, more capable and power efficient, complexity of chip designs are constantly growing. This is illustrated by new design approaches such as NoC and multi-core technologies. To handle this increased complexity, existing tools offer functionality for various level of simulation and basic capabilities for synthesizing design given as formal descriptions. As there is no foreseeable end to this development, higher abstractions for the design process need to be invented.

References

[1] Frank Vahid, Tony Givargis: *Embedded System Design: A Unified Hardware/Software Introduction.* John Wiley & Sons; ISBN 0471386782, 2002

[2] http://www.windriver.com/vxworks/index.html

[3] http://www.symbian.com

[4] http://www.rtlinuxfree.com

[5] http://www.microsoft.com/embedded/

[6] http://www.radisys.com/products/Microware%20OS-9.cfm

List of Figures

Appendix A: Acronyms and Abbreviations

AC ADC	Alternating Current Analog-to-Digital Converter
ALU	Arithmetic Logic Unit
ASIC	Application Specific Integrated Circuit
CAN	Controller Area Network
CPU	Central Processing Unit
CPLD	Complex Programmable Logic Device
DAC	Digital-to-Analog Converter
Demux	Demultiplexer
EDA	Electronic Design Automation
EPROM	Erasable Programmable Read Only Memory
HDL	Hardware Description Language
HLL	High-level Language
IC	Integrated Circuit
ICE	In-Circuit Emulator
I/O	Input/Output
LA	Logic Analyzer
MSI	Medium Scale Integration
OCD	On Chip Debugging
PAL	Programmable Array Logic, Phase Alternating Line
PC	Personal Computer
PDA	Personal Digital Assistant
SoC	System-On-Chip
SOIC	Small Outline Integrated Circuit
SSI	Small Scale Integration
UML	Undefined Markup Language
VHDL	Very High Speed Integrated Circuit Hardware Design Language