SAE International™

Embedded Software - Issues and Challenges

2010-01-0669
Published
04/12/2010

Sheetal Patil
Delphi

Laxman Kapaleshwari
Delphi

ABSTRACT

Embedded software is a software system that permanently resides in a device whose operations it controls. Typically, embedded systems are housed on flash memory or ROM chip and may be found in systems like cellular phones, household and office appliances having digital interfaces, medical equipment, automotive components, avionics etc. The fundamental problem facing the design of embedded systems is heterogeneity. Multiple styles of algorithms (e.g. signal processing, communications, and controls) are implemented using a variety of technologies (e.g., digital signal processors, microcontrollers, field-programmable gate arrays, application-specific integrated circuits, and real-time operating systems).Other challenges in automotive development are increasing requirements and therefore increasing size and complexity of the code, development and management of offshore / muti-site software development to reduce costs. Embedded software developers need embedded software development tools to develop embedded software. An embedded software development environment is an integrated collection of software development tools that manage the entire embedded software development process: analyzing, designing, documenting, writing, compiling, debugging, testing, optimizing, and verifying software. The choice of an embedded software development environment is the most important determinant of the productivity and effectiveness of developers. Requirement management systems can be used for capturing, linking, analyzing, and managing changes to requirements and their traceability and hence ensure conformance to requirements and compliance with regulations and standards. Using optimizing compilers with multi software development environment, developers can consistently generate smaller and faster code. A debugger helps developers find more bugs in their software more quickly. Getting the bugs out of the software is the most expensive, time consuming, and high-risk aspect of electronic product development. Testing of applications at unit level allows finding bugs early in the development process and increases the confidence level for system test, because the system is now based on well-tested units. The usage of appropriate test tools enables the automation of unit testing and is the base for regression testing, thus optimizing the development process and providing the base for future product feature enhancements.

THE CHALLENGES

Embedded software developers face many challenges in their work. Addressing the seven challenges discussed in this work can prepare developers to better deal with time constraints, reduce costs and enhance software quality.

1. RELIABILITY AND ROBUSTNESS

Reliability expectations will place greater responsibility on embedded systems developers to eliminate bugs and to design software to tolerate errors and unexpected situations. Failure detection and recovery is vital and the software must be designed so that it can run continuously, sometimes with controlled resets or sometimes without the need for a reboot. Error handling mechanisms should be capable of reporting the exact location of a software bug, even after the system has been deployed. Memory management techniques need to ensure that that a memory leak or stack overflow will not lurk like a time bomb in a long running system. Programmers need to be aware of real-time pitfalls when using interrupts and an RTOS.

An asymmetric multi-core platform, known as FIDES, which enhances robustness of such embedded systems. The FIDES

platform is based on multi-core technology and features high performance with low power consumption. Moreover, the platform helps improve system reliability by separately executing basic functions (e.g., pre-installed applications) and additional functions (e.g., downloaded applications) on different processors.

2. PERFORMANCE

To achieve goals of low-cost and small footprint, resources like processor power, memory size, and power draw must be kept as small as possible. These resources must satisfy the device's specific application and cannot be wasted by clumsy implementations, or by inefficient integration into the larger systems to which they often belong. Software engineers can be very effective at optimizing their code for their device's specific function, but usually do not or cannot optimize the overall system integration.

It is often the aspects of the final systems integration that cause applications to exceed their allocated resources, forcing costly and difficult decisions to be made in optimizing and modifying the application. Since this usually happens at the end of the development lifecycle, many embedded devices miss their committed delivery dates or are even scrapped altogether.

Model-driven development overcomes many of these issues by making successful systems development the primary effort and component design a resultant function.

The latest generation of model-based development languages provides a way to manage complexity issues of this type. The standards-based UML 2.0 and SysML languages allow companies to apply the principles of Model-Driven Development to focus development efforts while expanding overall vision and creativity

3. COMPLEXITY OF THE SYSTEM

Complexity is the outcome of several trends. It arises because of the combination of more and more functionality onto a single system and increasingly complex standards. Furthermore, as networking capabilities are becoming persuasive in embedded systems, a design becomes a system of systems, adding yet another layer of complexity.

Cars and other vehicles have become increasingly dependent on software to realize novel and innovative functions. The time when mechanics and electronics accounted for the majority of value growth in the automotive industry is long gone - today the majority of inventions and value growth comes from functions realized with software.

This sudden shift in technology from mechanics/electronics to software has created a whole new set of problems for the automotive industry. A modern high-end car can contain more than 2.000.000 lines of code, distributed over 80 nodes, using 5 different networks. Development of a system of this magnitude is a daunting task.

This problem can be tackled by standardizing software architecture and by reusing and sharing software. AUTOSAR (AUTomotive Open System ARchitecture) is one such open and standardized automotive software architecture, jointly developed by automobile manufacturers, suppliers and tool developers.

4. REQUIREMENT MANAGEMENT

Though innovative features are key potentials to competitive advantage, their merit will be limited, if quality, cost and time-to-market constraints are not met. Customers' wishes and requirements and the systems' constraints set up the entirety of functional and non-functional requirements that have to be met as a prerequisite for successful products. To be able to meet these requirements one has to be sure to know them in their entirety, without ambiguities, incorrectness or contradictions, in an up-to-date status. The requirements model evolves during product development, i.e. requirements continuously change, are updated, deleted or new ones are created. To be able to control the “magic triangle” of quality, cost and time constraints reliably, these changes have to be integrated into the current requirements model. Their impact has to be analyzed and evaluated, trade-offs have to be carried out, and decisions have to be made and documented.

Requirement management systems such as Teleology's DOORs can be used for capturing, linking, analyzing, and managing changes to requirements and their traceability; hence, ensuring conformance to requirements and compliance with regulations and standards.

5. OPTIMIZATION

Optimization is essentially a cycle of design, evaluation, and design again. It becomes a difficult issue because of the length of time taken to build a functioning system model or prototype to evaluate. An embedded system contains both hardware and software elements that interact in complex ways with one another which can make selecting the ‘best’ combination difficult.

One challenge of embedded automotive systems development involves correctly implementing requirements and specifications created by the system designer. These specifications are often delivered to embedded systems and software engineers in the form of written documents, flow charts, and sample code that must then be manually implemented in the embedded system.

With **Model-Based Design** such as Enterprise architect, everyone works from the same models, avoiding time-consuming, manual coding tasks and reducing the potential for errors. This approach helps produce better products faster and at lower cost by enabling run real-time simulations with hardware-in-the-loop systems and generating production C code from models.

6. PROJECT MANAGEMENT WITH GLOBAL SITES

Nowadays embedded software engineering is no longer a one-company or one-site activity, but more a team-based activity between several companies and different sites across the globe. Reasons for this are tight time-to-market requirements, economic constraints, tough competition and complexity of the systems. In order to acquire the required expertise, efficiency and desired lead-time, embedded systems need to be developed globally in collaboration with subcontractors, third party developers and in-house development. A project manager must be dedicated to coordinate project development and schedules.

When global companies are developing products in collaboration, it usually means that developing teams are dispersed. The collaborative development process between dispersed teams makes demands to the communication, team work and working methods.

Actually, the global team-based development process differs significantly from local development process. For example, different time zones and distances make communication more difficult than in local development. Configuration tools such as CM synergy can be used to manage documents and code.

Tool support in this case is seen as follows. Because development teams are geographically dispersed, team working requires technical assistance. As mentioned in several surveys, communication is the most important issue in this kind of activity, because no team works without communication. Thereby, information sharing is one of the most important requirements for all tools, so all participants can be up- to-date. Other common criteria for all tools are multi-platform support, tool integration, web access, access control and simultaneous usage. In order for tools to benefit the development process, they must be easy to use, simple and customizable for the usage of many companies.

7. VERIFICATION AND TOOLS

An embedded software development environment is an integrated collection of software development tools that manage the entire embedded software development process: analyzing, designing, documenting, writing, compiling, debugging, testing, optimizing, and verifying software. The choice of an embedded software development environment is the most important determinant of the productivity and effectiveness of developers.

Using optimizing compilers within a multi-software development environment, developers can consistently generate smaller and faster code. A debugger helps developers find more bugs in their software more quickly. Getting the bugs out of the software is the most expensive, time consuming, and high-risk aspect of electronic product development. Testing of applications at unit level finds bugs early in the development process and increases the confidence level for system test, because the system is now based on well-tested units. The usage of appropriate test tools enables the automation of unit testing and is the base for regression testing, thus optimizing the development process and providing the base for future product feature enhancements.

CONCLUSION

Embedded software development is likely to remain a challenging domain over the foreseeable future; however, there are many technological and methodological developments which may yield substantial benefits in the years ahead.

REFERENCES

1. Woodward Michael V., Member, IEEE, Mosterman Pieter J., Member, IEEE, **Challenges for Embedded Software Development**

2. Seminar Software Engineering for Automotive Systems - Florian Leitner, **Automotive Open System Architecture**

3. Schmitt Wolfgang, Manager, Hitex, **Automated Unit Testing of Embedded ARM Applications**

4. Heinonen Samuli, Kääriäinen Jukka, Soininen Sanna, Takalo Juha, VTT Technical Research Centre of Finland, **Analysing Tool Support for Embedded System Engineering in Collaborative Development Environment**

5. Hansson Hans, Nolin Mikael and Nolte Thomas, Mälardalen Real-Time Research Centre, Västerås, SWEDEN, **Beating the Automotive Code Complexity Challenge Components, Models and Tools**

6. Hiroaki INOUE, Naoki SATO, **FIDES: A Multi-Core Platform to Enhance Robustness of Embedded Systems**

7. McKorkle Scott, Telelogic, **Manage Embedded Systems Complexity with SysML**

The Engineering Meetings Board has approved this paper for publication. It has successfully completed SAE's peer review process under the supervision of the session organizer. This process requires a minimum of three (3) reviews by industry experts.

ISSN 0148-7191

doi:10.4271/2010-01-0669

SAE Customer Service:
Tel: 877-606-7323 (inside USA and Canada)
Tel: 724-776-4970 (outside USA)
Fax: 724-776-0790
Email: CustomerService@sae.org
SAE Web Address: http://www.sae.org
Printed in USA