Embedded Software Development Challenges in the Digital Signal Processing Era

John Linn
Director, Systems & Software Laboratory
Texas Instruments Inc., Dallas, Texas

linn@ti.com

Abstract

The rapid growth of products that incorporate programmable Digital Signal Processor is a driving-force for improving the embedded software development process. DSP-based product development presents some difficult challenges for manufacturers. Several embedded software development trends can be anticipated as the industry seeks to overcome these difficulties.

1.0 Introduction

The programmable Digital Signal Processor (DSP) has become a highly desirable computing engine for executing embedded software. Both the number and the complexity of DSP-enabled products are growing exponentially. Software development cost and time-to-market are critical issues for the industry, mirroring the issues of enterprise and desktop/server software development.

Most DSP software is embedded software. Manufacturers typically install product-specific software prior to sale. Most users do not realize they are executing software or even using a DSP when they use the product.

The embedded DSP software developer faces several unique challenges beyond those of classical software development. The dominant role and unique challenges of embedded DSPs will force the industry to evolve better ways of addressing these unique challenges.

This position statement discusses some of the challenges for embedded DSP software development and projects some opportunities to address these challenges.

2. The Era of Digital Signal Processing

Digital signal processing is now commonplace in our everyday lives. In 1997, the booming mobile handset industry sold more mobile wireless telephone handsets than the PC industry sold personal computers. In his COMDEX/Fall 2000 keynote address, Ericsson’s Kurt Hellstrom projected that by 2003, more people will access the net via cell phone than via PC.

Inside each handset are a number of integrated circuits dedicated to the complex tasks of converting back and forth between physical radio signals and speech. In a modern wireless handset, much of this signal processing is done digitally, translating the analog electrical signals into digital representations and performing mathematical signal-processing operations on them using digital arithmetic and Boolean logic. Most modern handsets incorporate at least one programmable DSP, and usually other micro-controllers and custom digital signal processing logic in a heterogeneous architecture.

Programmability allows the manufacturer faster time to market. The manufacturer can design more complex and sophisticated signal processing functions into the device in a shorter amount of time than he could using Application Specific Integrated Circuits (ASIC). Programmability allows for re-programmability, which in turn means quicker bug fixes and quicker incorporation of new functional improvements. The programmable DSP allows the industry to implement new and better communications standards and protocols very quickly, and is one of the reasons for this industry’s rapid growth.

Today’s wireless handsets contain on the order of 100-300K lines-of-code, including code for all embedded programmable devices. By 2005, 3rd Generation (3G) handsets may contain as many as 2-4M lines of code. For comparison, the Air Force F22 Advanced Tactical Fighter is estimated to contain around 3M lines-of-code.

Many other DSP applications are seeing similar growth and complexity increase. Just some examples with similar embedded software complexity are Broadband residential and small-office gateways, commercial and consumer digital cameras, and portable digital audio players.

3. The Embedded Software Challenge

Most new applications of programmable DSPs will contain embedded software. In most ways, the software development challenge for embedded software mirrors that of desktop/server development. However, in several ways, embedded software development is more challenging.

Software development productivity must increase to satisfy the demand for more complex functionality with lower recurring cost and shorter time-to-market. Today,
much of the software written for embedded DSPs is created from scratch, or is based on an ad-hoc “source-code-sharing” re-use strategy. Productivity improvement through software re-use is required.

However, DSP-enabled products have much more hardware diversity and configuration complexity compared to desktop/servers. Often, these products contain multi-processor hybrid architectures, incorporating RISC micro-controllers and special-purpose accelerators in addition to DSPs. Low cost and small size are critical, leaving no room for excess. Embedded software development methodology and tools for these devices must support this diversity, complexity and heterogeneous multi-processor development and debug environments with very little increase in code size or code efficiency over a custom software approach.

As with all software, overall system reliability must improve, in spite of the increasing complexity. Users of today’s DSP-enabled products expect dependability like that of a telephone and better than that of a desktop computer. Development methodology and tools for embedded software must build in system dependability.

Finally, real-time constraints are critical factors in the embedded DSP software development process. Memory, processing speed, and power consumption will remain highly constrained by the high demand for performance, the low manufacturing cost targets and portability requirements. Development methodology and tools for embedded software must deal with real-time performance and the asynchronous interaction between the software and the outside world.

4. The Opportunity

The embedded software challenges bring opportunities. These opportunities should come through improvements in the software development tools, methodologies and industry infrastructure to address the challenges head-on. Some of the trends that we might predict over the next few years might be forecast from the opportunities presented.

Software re-use strategies will evolve to improve embedded software development productivity along two thrusts: 1) component-based software standards, and 2) application-specific frameworks. Component-based software standards provide rules for developing reusable components, including rules for exposing interfaces and using resources. Such standards make it easier to re-use software components in new designs. TI’s “TMS320 DSP Algorithm Standard” is one example of an embryonic standard being adopted by many 3rd party developers.

An application-specific framework is a software architecture that provides the generic structure, dataflow and interfaces required for software development in a specific product domain. The framework accepts plug-in software components to implement the desired functionality. Blue Wave’s FACT™ subsystem management product for telecommunications applications is one example of an emerging DSP framework. Media player encode/decode frameworks are on the horizon.

Frameworks must support heterogeneous multi-processor environments and simplify the difficult tasks of programming for multiple processors. Frameworks can significantly improve heterogeneous multi-processor software development and re-use through abstraction of dataflow and control flow between components executing on different processing resources.

Embedded software components must be much more customizable than allowed by current approaches, without resorting to manual source code modifications. Manufacturers with high volume, low cost products probably won’t accept the memory size or performance degradation usually included in today’s components. However, the technologies to extract only the necessary interfaces and functions from a reusable component and customize its configuration are currently not available. Such new technologies need significant capability to trade performance, code size and memory usage. System-level design and optimization should drive the extraction, configuration and optimization process.

Embedded software component methodologies, frameworks and tools must encourage 3rd party development. Standards, de-facto or otherwise, must evolve. The methodologies must protect the intellectual property of 3rd party developers while providing the customizability they require. Likely, this means avoiding source code delivery, not really practical today.

Real-time performances, deadline scheduling, time synchronization and quality-of-service guarantees, are critical to operational performance. Components and frameworks need to incorporate real-time performance features. Components should include performance predictability or performance reflectivity, with appropriate interfaces to access their performance via the framework. Performance of components must be predictable and characterizable over the total range of environments and customizations.

Overall system reliability is critical to customer satisfaction and reliability/recovery must be designed into the components and the frameworks. As with performance, reliability of components must be predictable and characterizable over the total range of environments and customizations.

5. Conclusion

The challenges of embedded software development are creating a very entrepreneurial environment for solutions. Many new innovations in software development will rise in the next few years to meet these challenges.