ASSESSING AND IMPROVING CURRENT PRACTICE IN THE DESIGN OF APPLICATION-SPECIFIC SIGNAL PROCESSORS

G. A. Shaw and J. C. Anderson

MIT Lincoln Laboratory
244 Wood Street
Lexington, MA 02173-9108
shaw@ll.mit.edu

V. K. Madisetti

Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, GA 30332-0250
vkm@eedsp.gatech.edu

ABSTRACT

The Department of Defense ARPA program for Rapid Prototyping of Application Specific Signal Processors (RASSP) exists to significantly improve the process by which embedded digital signal processors are developed (prototyped) and supported (maintained and upgraded). As used in the RASSP program, the term prototype signifies a system that is a precursor to a deployed system, but still meets all of the essential performance goals and is designed to facilitate maintainability and upgradability. In this paper, current practice in the design of embedded digital signal processors, as exemplified in the traditional waterfall design methodology, is examined and shortfalls in the design methodology and supporting tools are identified. Opportunities for improving the traditional design practice are then identified and evaluated in terms of potential benefits, as well as impediments, to implementation and adoption by the community.

1. INTRODUCTION

Within the past 10 years, high-end signal processing applications have grown from millions of operations per second, implemented in hardwired or uniprocessor architectures, to billions of operations per second implemented on arrays of programmable multiprocessors. At the same time, the functionality that was once implemented at the board level with large-scale integrated circuits has been subsumed at the chip level in very large scale integrated circuits containing millions of transistors and hundreds of pins, employing clock rates approaching 100 MHz and complex software development environments. The introduction of more complicated building blocks, higher clock rates, and tightly-coupled hardware and software environments has opened a gap between traditional design and verification methods and the complexity and supportability required for contemporary digital signal processors. Furthermore, with new DSP chip technology being introduced annually, traditional methods of optimizing a design for a given application and associated processing engine are no longer cost-effective or appropriate in terms of supporting an upgrade path.

Applications requiring embedded signal processors are as numerous and diverse as the methodologies employed to design and build them. Consequently, there is no unique model of "current practice" as it applies to the design of application specific signal processors. Nevertheless, a representative model of current practice is essential to the RASSP program in order to assess where improvements are needed, and also as a basis for measuring progress of the RASSP program toward the goals of reduced design cycle time, reduced cost, and improved quality.

This work was sponsored by the Advanced Research Projects Agency, Electronic Systems Technology Office.

Table 1: Current Practice Views

VIEW	EXAMPLES
Process	Design Flow, Productivity, Test, Reviews
Resource	Development Time, Cost, Tools, Libraries
Product	Form, Performance, Defects, "...ilities"

The focus of the RASSP program is on high-performance form-factor constrained signal processors consisting of anywhere from a few to hundreds of processing engines. The models described here are an attempt to characterize, at least in an average sense, the current practice in developing such state-of-the-art embedded signal processors at the inception of the RASSP program, circa 1993.

2. CURRENT PRACTICE MODEL VIEWS

In developing a representative or "industry standard" model of current practice, there are at least three views of interest to the RASSP program: The *process* view emphasizes issues such as the steps or methodology followed, the degree of concurrent activity, and the productivity achieved. The *resource* view might also be termed generalized cost, and emphasizes the people, tools, time, etc. required to develop a prototype signal processor. The *product* view emphasizes issues related to the soundness and performance of the product, as well as adherence to requirements. The three views are clearly not independent. For example, methodology affects development cost (resource) and quality (product).

2.1. Process View

The traditional design methodology, which is embodied in military process standards such as DOD-STD-2167A for software development, is a waterfall design process, as illustrated in Figure 1. The underlying concept behind the

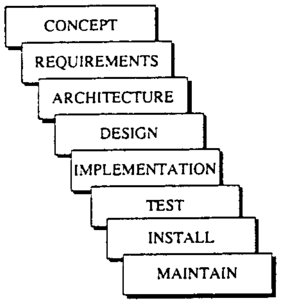

Figure 1: Waterfall development methodology.

waterfall design process is a progression through various levels of abstraction, or phases, with the intent of fully characterizing each level of abstraction before moving to the next level, and providing a comprehensive work package at each phase. Strict adherence to the waterfall design methodology is impractical, in part because the requirements for an embedded signal processor are often vague at the beginning of a project and the processor is often subject to significant design changes. For example, in a radar system, waveforms, processing algorithms, and subsystem interfaces may all be modified during the course of signal processor development. Nonetheless, this methodology is characteristic of current practice, particularly in the defense industry. While following the waterfall design methodology does not preclude attaining the RASSP goals of rapid design cycle time, low life cycle cost and model year upgrade capability, the waterfall design methodology does tend to foster a number of bad design practices including:

1. Low exploitation of concurrent engineering
2. Emphasis on wrong problems early in the design phase
3. Inflexibility late in the design phase
4. Low level of customer interaction and subsequent satisfaction
5. Significant rework and cost resulting from not discovering design flaws until the integration phase.

Figure 2 is a simplified representation of a waterfall design methodology for embedded signal processor design. Perhaps the most significant deficiency in the methodology is that hardware subsystems and application software are not integrated until late in the process, and significant design flaws may go undetected until the integration step.

2.2. Resource View

Figure 3 is an estimate of the relative distribution of cost associated with the development of the synthetic aperture radar (SAR) image formation processor described in [1]. A number of activities such as program management and

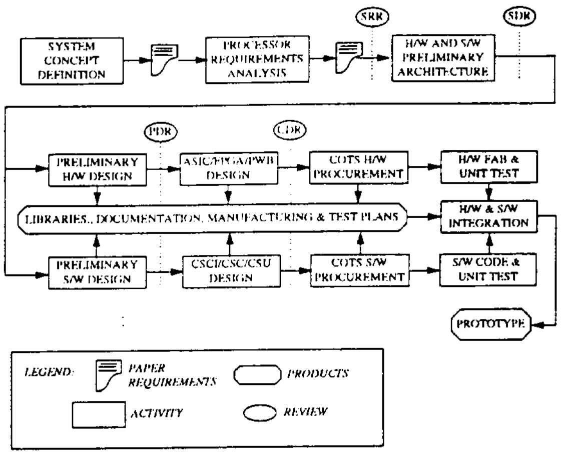

Figure 2: Simplified current practice design flow.

reporting have been omitted from Figure-3 for simplicity. The performance requirements for the processor are summarized in Table 2. In Figure 3, requirements and architec-

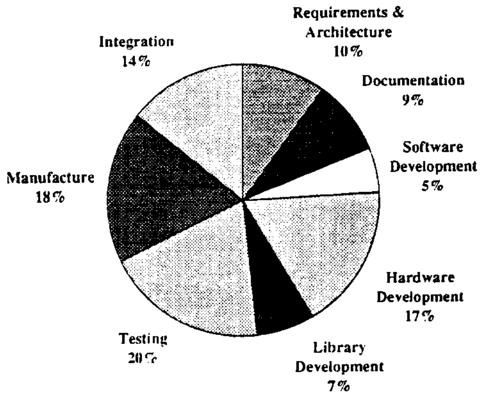

Figure 3: Relative dollar cost associated with various development tasks.

ture development together represent only 10% of the total. However, once an architecture is selected, much of the development and life cycle cost of the system, as well as achievable performance, are determined. Also note that testing and integration consume a larger percentage of cost than the combined software and hardware development (34% versus 22%). The software development in this example consists mainly of well-defined algorithms such as FFTs. State-of-the-art electronic design automation (EDA) tools to support the design flow of Figure 2 may cost as much as \$80K-\$100K per single-user license, and a collection of these tools, spanning the end-to-end process, could cost in excess of \$1M to purchase, and \$150K or more a year to maintain, excluding training costs. Despite the cost of the tools, interoperability across tools is not assured, particu-

Table 2: SAR Processor Requirements

Item	Requirement
Max. Volume	2.2 cuf
Max. Power	500 W
Max. Weight	60 lbs
I/O Rate	18/27 MB/s
Interface	Fiber
Polarizations	3
Frame Size	2048 × 512 pixels
Dynamic Range	> 103 dB

larly in the case of commonly used high-level system design tools. Computer-aided design (CAD) and computer-aided software engineering (CASE) tools are available to support either the hardware or software design, but there is little to support the co-design and simulation of hardware and software. In particular, there are few libraries and models to support co-simulation of hardware and software, and there are not standards for interoperability of models at various levels of design abstraction.

3. OPPORTUNITIES FOR IMPROVEMENT

Incremental improvements in design practice occur more or less continually, but significant improvements are almost always due to a revolutionary change in the resources or processes employed [2]. The shortfalls in traditional design methodology suggest areas which might be targeted for revolutionary change.

3.1. Process Improvements

3.1.1. Executable Requirements

Figure 2 emphasizes the fact that current practice is to provide processor requirements in written form, often hundreds of pages of requirements which must be interpreted and captured in a traceabilty tool. Provision of requirements in machine readable and executable form has the potential to significantly reduce the ambiguity in written requirements. The SAR benchmark described in [1] includes an executable requirement written in VHDL which is intended to serve as the basis for test bench generation during detailed design and verification.

3.1.2. Virtual Prototyping

A virtual prototype [3], consisting of a software model of the hardware executing a representation of the application code, has the potential to uncover design flaws before the costly step of hardware fabrication and test fixture generation. In the case of integrated circuit design, virtual prototyping has already been proven to be an enabling technology for first-pass correct design. The same concepts can be applied to board level design provided suitable models and simulation tools are available. A virtual prototype also has the potential to support early customer evaluation and exploration of performance trades.

3.1.3. Successive Refinement

Unlike the waterfall development methodology, which emphasizes complete descriptions of the signal processor at each level of abstraction in the design process, successive refinement emphasizes rapid development of a less than full function prototype to uncover potential problems early and to influence the design through hands-on experience. The terms *successive refinement, spiral design, incremental development, risk-driven design*, are all used somewhat interchangeably to describe this basic approach. Spiral design was pioneered for software development [4], but the concept can be applied to hardware as well. Benefits of successive refinement include the ability to involve the end user in evaluating early prototypes, early discovery of problems in the design concept, and improved estimates of the cost and schedule to produce a fully-functional prototype. However, successive refinement can be costly when hardware fabrication is in the loop, and virtual prototyping represents a potentially cost-effective methodology for supporting successive refinement.

3.1.4. Co-development Methodologies

Co-development, or hardware-software co-design, refers to the ability to begin with an implementation-independent representation of the requirements for a signal processor and evolve these requirements to a hardware and software implementation that is optimum, or nearly so, in some sense. Virtual prototyping supports hardware-software co-design by facilitating the transfer of functionality between hardware and software, enabling performance analysis and trade-offs prior to the existence of the hardware. Current practice predominantly relies on the experience of designers to allocate functionality to either hardware or software early in the design. Once the allocation is made, the hardware and software development tends to proceed along relatively independent paths with few opportunities created for improvement through trade-off analyses.

3.1.5. Parametric Cost Estimation

Parametric cost estimators (PCEs) have been shown to give engineering managers a competitive edge by accurately predicting project costs. PCE tools, available now in stand-alone form, can be integrated with front-end design tools to provide a more quantitative and traceable basis for architecture selection. In the absence of a well-defined cost estimation methodology, critical items, such as testability, are often overlooked in determining cost, schedule and risk associated with candidate architectures. Representing each candidate architecture by a set of cost breakdown structures and applying the appropriate PCE tools helps ensure that all relevant aspects are considered. PCE tools also enable assignment of numerical values for cost, schedule and risk associated with each candidate architecture, and provide documentation of the basis for architecture selection.

The ability to identify and quantify life cycle cost issues is an important capability afforded by PCE tools. Hardware life cycle costs are a function of maintenance concept (e.g. throw away vs. fix a failed module), and a specific maintenance concept must be supported by the appropriate built-in test features and external test equipment. PCE tools provide a means for rapidly evaluating a large number of maintenance concepts, and results of the PCE life cycle analysis have a direct impact on test requirements and architecture selection.

3.2. Resource Improvements

3.2.1. Standard Hardware Interfaces

As DSP chips continue to gain in complexity and functionality, the major effort in embedded processor design has shifted to specialized hardware for systolic processing and the communication and control interfaces for multiprocessor architectures. The VME bus is a familiar example of a standard interface which facilitates rapid development of application hardware by promoting reuse and standard protocols for communication. However, bus architectures do not scale well, and interfaces with substantially higher bandwidth and latency are required for many applications. Designing for upgradability demands the use of standard, scalable interfaces and memory architectures. Standard interfaces are essential in promoting widespread software and hardware reuse.

3.2.2. Reuse Libraries

DSP chip developers currently provide C-language instruction set simulators and highly optimized FFT and other software modules with a new chip. The provision of these tools and libraries promotes reuse on a wide scale. However, virtual prototyping and hardware-software co-development methodologies require many additional models at at various levels of abstraction. Currently, such models are not widely available, and concerns exist over the intellectual property embodied in such models. Modeling standards including, for example, the appropriate levels of abstraction, are needed to support wide-spread reuse.

In the case of application software, substantial reuse has been shown to yield gains of 4× or more in productivity for uniprocessor development [5]. However, substantial reuse of software in embedded signal processing is hampered by the lack of standard communication and control interfaces, and the highly parallel hardware. Reuse can be facilitated by the adoption of standards and the model-year architecture concepts described below.

3.3. Product Improvements

3.3.1. Model-Year Architectures

Programmable processing chips tend to double in performance approximately every 18 months, and with multiple vendors developing new chips, improved technology is available on even shorter cycles. In order to field signal processing systems with the latest available processor technology, the hardware and software architectures must be sufficiently "portable" or standardized to support late binding of the processor chips to the software and board level communication fabric. In the case of software, this flexibility is achieved through high-level language implementation for the control and standardized library calls for DSP number crunching, such as FFTs. In the case of the hardware, the equivalent of a high-order language is a high-level description of the custom designs which can be synthesized into a preferred technology, such as FPGAs. The equivalent of the optimized DSP library is standardized interfaces and associated communication protocols, as described in Section 3.2.1.

3.3.2. Executable Specifications

In the same way that executable requirements facilitate the initial development of a signal processor, the final design can also be documented in a machine readable and executable form. Executable specifications have long been the norm for application software written in a high-level, portable language. The existence of standards for hardware design languages affords the opportunity to document hardware in a similar fashion, facilitating upgrades and reducing life-cycle support costs.

4. CONCLUSION

Historically, significant improvements in the required design cycle time and cost to produce embedded digital signal processors have been brought about by revolutionary changes in the design process or resources comprising the design environment. Presently, the process from schematic entry to printed wiring board, or from CASE tool to application code, is fairly mature. However, substantial improvements are feasible in the front-end processes relating to requirements capture, functional modeling, partioning into hardware and software, and designing for easy upgradability and supportability.

A Mosaic server has been established on the World Wide Web as a source of additional information and publications. The Lincoln Laboratory RASSP home page is accessible via the uniform resource locator (URL) http://www.ll.mit.edu/~llrassp/rassp_home.html.

5. REFERENCES

[1] B. W. Zuerndorfer, et al, "RASSP Benchmark-1 Technical Description," *MIT Lincoln Laboratory Project Report RASSP-1,* 13 December 1994.

[2] K. A. Radtke, "The AT&T Hardware Design Environment: A Large System's Hardware Design Process," *31^{st} ACM/IEEE Design Automation Conference,* 1994.

[3] V. K. Madisetti, et al, "Virtual Prototyping of Embedded DSP Systems," *Proceedings IEEE International Conference Acoustics, Speech, and Signal Processing,* 1995.

[4] B. W. Boehm, "A Spiral Model of Software Development and Enhancement," *ACM Software Engineering Notes,* August 1986.

[5] R. B. Grady *Practical Software Metrics for Project Management and Process Improvement,* Prentice Hall, NJ, 1992.