Reuse-Oriented Model Year Architectures for Rapid Prototyping

G. Caracciolo and J. Pridmore
Lockheed Martin Advanced Technology Laboratories
Camden, NJ
gcaracci@atl.ge.com

Abstract

The Rapid Prototyping of Application-Specific Signal Processors (RASSP) program is striving to change the way embedded signal processor design is performed, providing >4X improvements in time-to-market, cost, and design quality. These improvements will be achieved using a methodology that stresses hardware and software reuse in conjunction with Model Year Architectures that facilitate reusability and upgradability through open interface standards. This paper will describe a Model Year Architecture approach for the development of cost-effective signal processors that can be applied to a wide range of military and commercial applications.

1: Introduction

The drivers for RASSP signal processor architecture definition result from the requirements imposed on signal processors to meet changing mission-critical processing needs and military requirements for long-term life cycle support. Additionally, RASSP must address the full spectrum of signal processing applications, from low-cost commercial applications, such as cellular communications and HDTV (1-10 processors), to very large military sensor systems, such as shipboard radar systems (100 - 1000 processors). This range of requirements imposes a formidable challenge in defining an architectural approach that addresses low-cost technology insertion, upgradabil-ity, and extensibility.

The Model Year Architecture (MYA) is being developed to address these issues, promoting design upgrades and reuse via standardized, open interfaces, while leveraging state-of-the-art commercial technology developments. Designs are performed using a concurrent engineering process that facilitates continuous product improvements via iterative virtual prototypes, which can be easily retargeted to support a range of applications[1].

RASSP Model Year architectures must be supported by library models to facilitate trade-offs and optimizations for specific applications. The hardware and software elements within the library are "encapsulated" by functional wrappers, which add a level of abstraction to hide implementation details and facilitate efficient technology insertion. Thus, the notion of Model Year upgrades is embodied in reuse libraries and the methodology for their utilization.

2: Model Year Architecture Framework

The RASSP program supports the design of architectures through a *framework* that provides a structured approach to ensure that designs incorporate the following required Model Year features: scalability, heterogeneity, open interfaces, modular software, life cycle support, testability, and system retrofit. [2]. The basic elements that comprise the MYA are the Functional Architecture, Encapsulated Library Components, and Design Guidelines and Constraints, as shown in Figure 1. Synergism between the MYA framework and the RASSP methodology is required, as all areas of the methodology, including architecture development, hardware/software codesign, reuse library management, hardware synthesis, target software generation, and design for test are impacted by the MYA framework.

The *Functional Architecture* defines the necessary components and the manner in which their interfaces must be defined to ensure that the design is upgradable and facilitates technology insertion. As such, the Functional Architecture is a starting point for developing solutions for an application-specific set of problems, not a detailed instantiation of an architecture. Specifically, the Functional Architecture specifies a high-level starting point for performing application-specific architecture selection; a standard approach for selecting and implementing standard, open interfaces; and guidelines for efficient verification and test. The Functional Architecture DOES NOT specify the topol-

©1995 IEEE. Reprinted with permission, from *Proceedings of the International Confernece on Acoustics, Speech, and Signal Processing*; Detroit, MI; May 9-12, 1995; pp. 2703-2706.

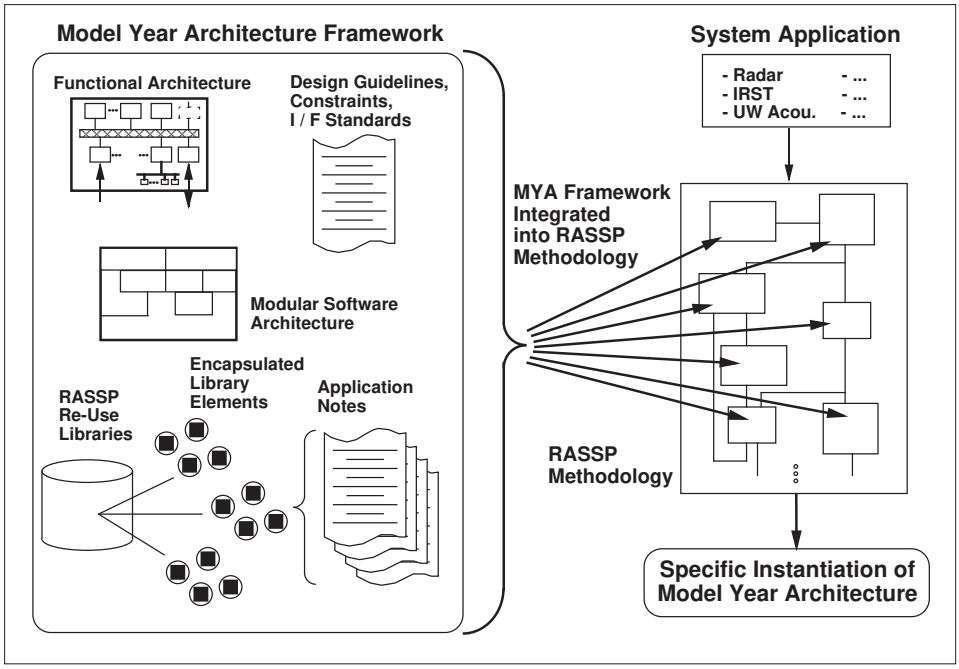

Figure 1. Model Year Architecture framework.

ogy or configuration of the signal processor architecture, specific processor types, or system-level interface standards (external to the signal processor).

The Functional Architecture concept is based on the use of abstract architectural objects and standard functional interfaces at key points within a layered architecture. An important aspect of the Functional Architecture is that application-specific realizations of a signal processor are embodied in the proper definition and use of *Encapsulated Library Elements* within the reuse library. Encapsulation refers to additional structure added to otherwise "raw" library elements to support the Functional Architecture and ensure library element interoperability and technology independence to the maximum extent possible. Incorporated within the reuse libraries are application notes that the designer can use to properly apply and aggregate the individual hardware and software components into a final processor product.

The MYA Framework also provides a set of *Design Guidelines and Constraints* for general architectural development, such as how to properly use the functional architecture framework, general use of encapsulated libraries, and most importantly, procedures and templates to encapsulate new library components. These design guidelines and constraints are incorporated into the RASSP design methodology.

The *Modular Software Architecture*, shown in Figure 2, simplifies developing high-performance, real-time DSP applications — allowing the developers to easily describe, implement, and control signal processing applications for multiprocessor implementations. The architecture supports the Model Year concept by providing a common Application Programming Interface (API) to the underlying real-time operating system services. This allows a new hardware platform with a new microkernel to change for each model year while maintaining the API. Support for the API is through the RASSP Run-Time System (RRTS), which provides the services required for the control and execution of multiple graphs on a multi-processor system. The RRTS and its support for the API forms the essential component of software encapsulation for a processor object.

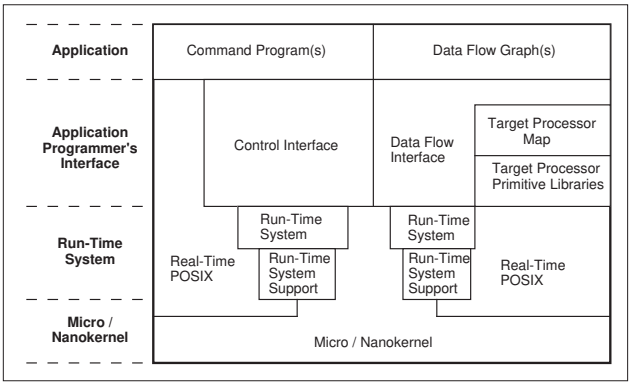

Figure 2. Modular Software Architecture.

The application layer is divided into two parts, similar to the Processing Graph Method (PGM) developed by the Naval Research Lab [3]. The first part of an application is the Command Program, which provides response to external control inputs, starting and stopping data flow graphs, managing I/O devices, monitoring flow graph execution and performance, starting other command programs, and setting flow graph parameters. The Control Interface provides services that implement these operations.

The second part of the application layer is the data flow graphs (DFGs), implemented using a data flow language. Services provided by the DFG interface are largely invis-

ible to the developer and include managing graph queues, interprocessor communication, and scheduling. The RASSP program will support static and dynamic scheduling paradigms. The constructed flow graph will be converted into a HOL such as C or Ada via autocode generation and will contain calls to a standard set of domain primitives. A full suite of tools is being developed on RASSP to support this software architecture. All RASSP tools will be made commercially available.

3: Applying the Model Year Architecture Framework

3.1: Hardware architecture

Verification of a MYA signal processor is iteratively performed throughout the codesign process, requiring the reuse libraries to support models at various levels of hierarchy. Three levels of VHDL modeling hierarchy are currently being developed and used in a series of benchmarking experiments to define reuse library elements for RASSP:

- *Performance/Uninterpreted/Architectural* models provide timing-only behavior for processor nodes, buses/interconnects, etc. to support high-level architectural trade-offs (number and types of processors, type and topology of network).
- *Abstract Behavioral Models* provide full functional behavior at the data output level with (potentially) an abstract level of timing. This level includes both algorithm-level and Instruction Set Architecture (ISA)-level models.
- *Full-Functional and Interface models* provide full functionality at the signal level and timing fidelity at the clock level. This includes Register Transfer Level and logic models.

Through these models, the Functional Architecture constructs are supported. For example, Figure 3 illustrates an application of a functional interface at the hardware level, called a Standard Virtual Interface (SVI), for a construct called a Reconfigurable Network Interface (RNI). The RNI is divided into three logical elements: 1) local interface, 2) external interface, and 3) bridge element. The local and external interfaces implement the specific protocols to the elements being interconnected, in this example a HIgh speed Parallel Port Interface (HIPPI) and VME interface. The bridge element, which typically consists of a buffer memory, and a controller implemented via custom logic (e.g. FPGA, ASIC) or a programmable processor, performs the actual bridging function. The buffer memory facilitates asynchronous coupling and flow control between the two networks, while the controller coordinates data transfers. The three logical elements of the RNI are implemented as encapsulated library elements that serve to isolate changes resulting from upgrades. For example, the VME interface could be replaced by another encapsulated interface, such as the Scalable Coherent Interconnect (SCI), with little or no impact on the HIPPI hardware and software.

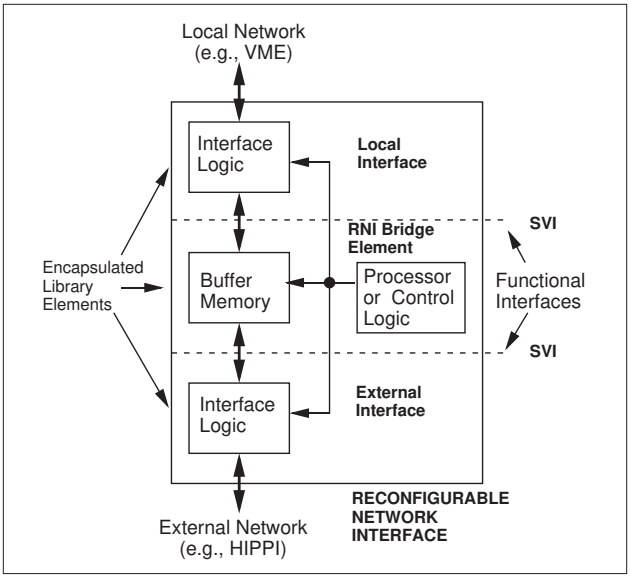

Figure 3. Functional Interface example applied to a Reconfigurable Network Interface.

To refine details of various architectural constructs and to determine their performance impact, a number of experiments are ongoing, primarily through VHDL modeling and simulations. For example, i860 ISA-level models [4] and the Floating Point Application-Specific Processor 5 (FPASP5) vector processor model developed by Rome Laboratory [5] are being used in conjunction with models for Peripheral Component Interconnect (PCI) and HIPPI interfaces. These models are being used as vehicles for refining the functional architecture concept by encapsulating the models and demonstrating a plug-and-play capability among the i860, FPASP5, and the different interface elements. Note that the functional interface at the software level must also be maintained, which will also be verified by executing interface software on the simulation models.

3.2: Software architecture

Software development cannot be discussed without its relationship to the architecture of the signal processor; in fact, it is an important part of the application-specific architecture design process. The representation of architectural elements as objects includes not only hardware representations in the form of VHDL models, but also behavior defined by the software libraries associated with that hardware. The software portion of architectural objects is handled

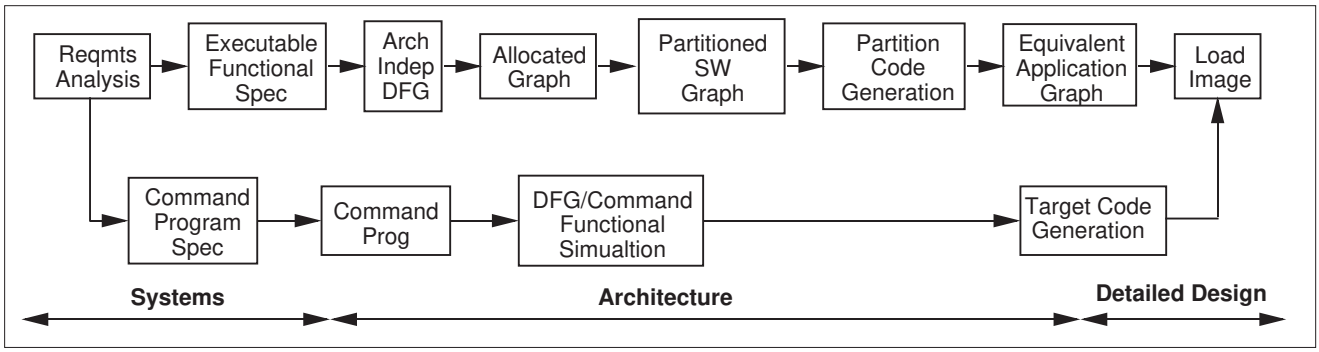

Figure 4. RASSP graph-based software development scenario.

by the process shown in Figure 4. This process depicts the progression of software generation from the requirements to load image, with emphasis on the graph objects involved and the general RASSP process in which they occur. It also shows the parallel development and co-simulation of the command program.

Architecture definition involves the creation and refinement of the data flow graphs that drive both the architecture design and the software generation for the signal processor. The data flow graph(s) of the signal processing are developed and the nodes are allocated to either hardware or software. Automated generation of the software partitions is performed to provide executable threads that are to be run on the DSPs. These autocoded partitions are combined into an application graph which is functionally equivalent to the original. The graphs are co-simulated with the command program to ensure proper interaction.

The final step in the software development, which is the production of the load image, occurs during detailed design. The software load image generation is an automatic build process that is driven by the autocode generation results. The inputs to the process include the architectural description, the detailed DFGs describing the processing, the partitioning and mapping information, the autocode generation results, and the command program. The process is controlled by a software build management function which extracts the necessary information from the library and manages the construction of all the downloadable code as directed by the partitioning and mapping data.

This process is verified through virtual prototyping prior to committing to an actual hardware build and is carried out at several levels of hierarchy including performance level simulations, ISA level simulations of key hardware and software elements, and low-level simulation of hardware interfaces.

4: Conclusions

The RASSP program is applying a Model Year Architecture concept to the rapid prototyping of embedded signal processors. This concept facilitates reusability and regular, low-cost technology upgrades. This is accomplished through the definition of a framework for developing open architecture signal processors, which can be applied to a wide range of military and commercial applications. The framework relies heavily on Object-Oriented concepts to properly encapsulate the architectural reuse library components that are modular and scalable. Ongoing work is refining the concepts of the Model Year Architecture framework, including the definition of architectural object classes, interfaces, and attributes for the various elements. Additionally, benchmarks are being developed to quantify hardware and software overhead through virtual prototype examples to refine the encapsulation concept. The MYA will support an automated reuse-based code generation process for heterogeneous multiprocessors.

References

1. Mark Richards, “The Rapid Prototyping of Application Specific Signal Processors (RASSP) Program: Overview and Accomplishments”, Proceedings of the First Annual RASSP Conference, August 1994.

2. Gerald Caracciolo, “RASSP Model Year Architecture Working Document Version 1.0”, October 28, 1994.

3. Naval Research Laboratory, “Processing Graph Method Specification”, Version 1.0 11 Dec. 1987.

4. V.J. Madisetti, T. Egolf, S. Famorzadeh, L-R Dung, “Virtual Prototyping of Embedded DSP Systems”, to appear in Proceedings of IEEE ICASSP’95.

5. Richard Linderman, Ralph Kohler, “Designing a Wafer-Scale Vector Processor Using VHDL”, GOMAC 1991 Digest of Papers. 1991 pp 65-68.