Autocoding in Lockheed Martin ATL•Camden RASSP Hardware/Software Codesign

Christopher B. Robbins
Management Communications and Control, Inc.

Abstract *- Autocoding provides the Lockheed Martin Advanced Technology Laboratories•Camden (LM-ATL•Camden) RASSP hardware/software codesign process the means to rapidly realize implementations of the codesign software architectures. Management Communications and Control, Inc. (MCCI) autocoding tools automate translation of software architecture specifications to designs and their implementations for functional and detailed hardware/software cosimulation, unit testable modules, and complete application load image specifications. The tools support an open application programmer's interface to the codesign process. Autocoding tools support the codesign processes objective of providing a seamless translation of applications from math tool level functional algorithm specifications to target architecture load images. Autocoding technology is directed at reducing the labor content of software design and coding, enabling rapid development of the software elements of application specific signal processing systems.*

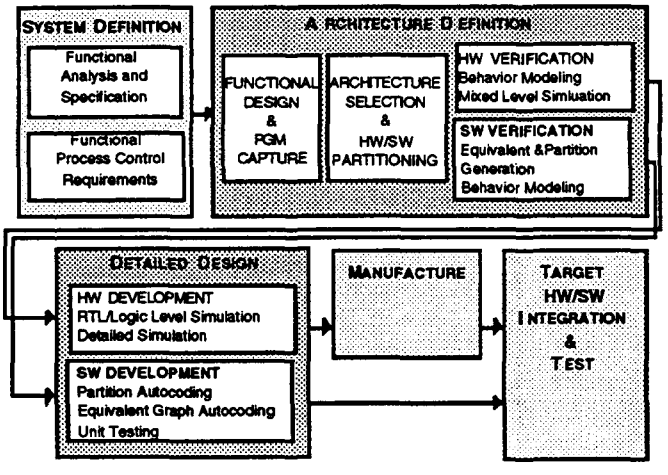

Figure 1 - Lockheed Martin ATL•Camden RASSP HW/SW Codesign Process

Role of Autocoding in RASSP Hardware/Software Codesign Process - The autocoding toolset provides automated assistance for realizing top level software designs from architecture specifications, and it fully automates detailed software design and coding. Figure 1 illustrates the LM-ATL•Camden hardware/software codesign process. The process provides for (1) system processing and control functional definition; (2) architecture definition, partitioning, and hardware and software architecture verification; (3) hardware and software detailed design; (4) hardware manufacture and target architecture hardware; and (5) software integration and test. These steps include domain engineering activity where processing function, control, and architecture specifications are developed and architecture implementation activity at verification, detailed design, and manufacture or coding levels. Autocoding tools are focused on the traditionally labor intense software verification, software detailed, and coding codesign implementation activities. Software architecture specifications are automatically translated into executable partition and equivalent application specifications in response to the domain engineer's partition and control parameter specifications. Performance estimation feedback is nearly instantaneous, supporting rapid iteration over parameter ranges and partitioning decisions within the architecture specification. Verified partition and equivalent application specifications are then automatically translated to source code for the partition executables and data structures from which the run-time system creates executable images of the equivalent application specifications.

The autocode toolset provides the implementing technology for the LM-ATL•Camden RASSP open application programmer's interface (API). Figure 2 illustrates the elements of this open API. The algorithm functionality is captured in PGM data flow graphs. PGM graphs exist in iconic and notational form. PGM supports specification of full system data flow and data flow control. Processing functions are specified by graph nodes. Queues specify data flow between processing nodes. Formal queues and variables may be externally controlled. A command message interface to the graph provides the control interface. Functional process control requirements of the system definition are implemented as sequences of command procedures directed by command messages. Signal processing is specified by domain primitives, target independent functional signal processing primitive specifications. The autocoding toolset implements domain primitives on each type of computational element supported in the model year architecture. Software architectures specified using the open API may be ported among all model year architectures without change to architecture specification or external controls.

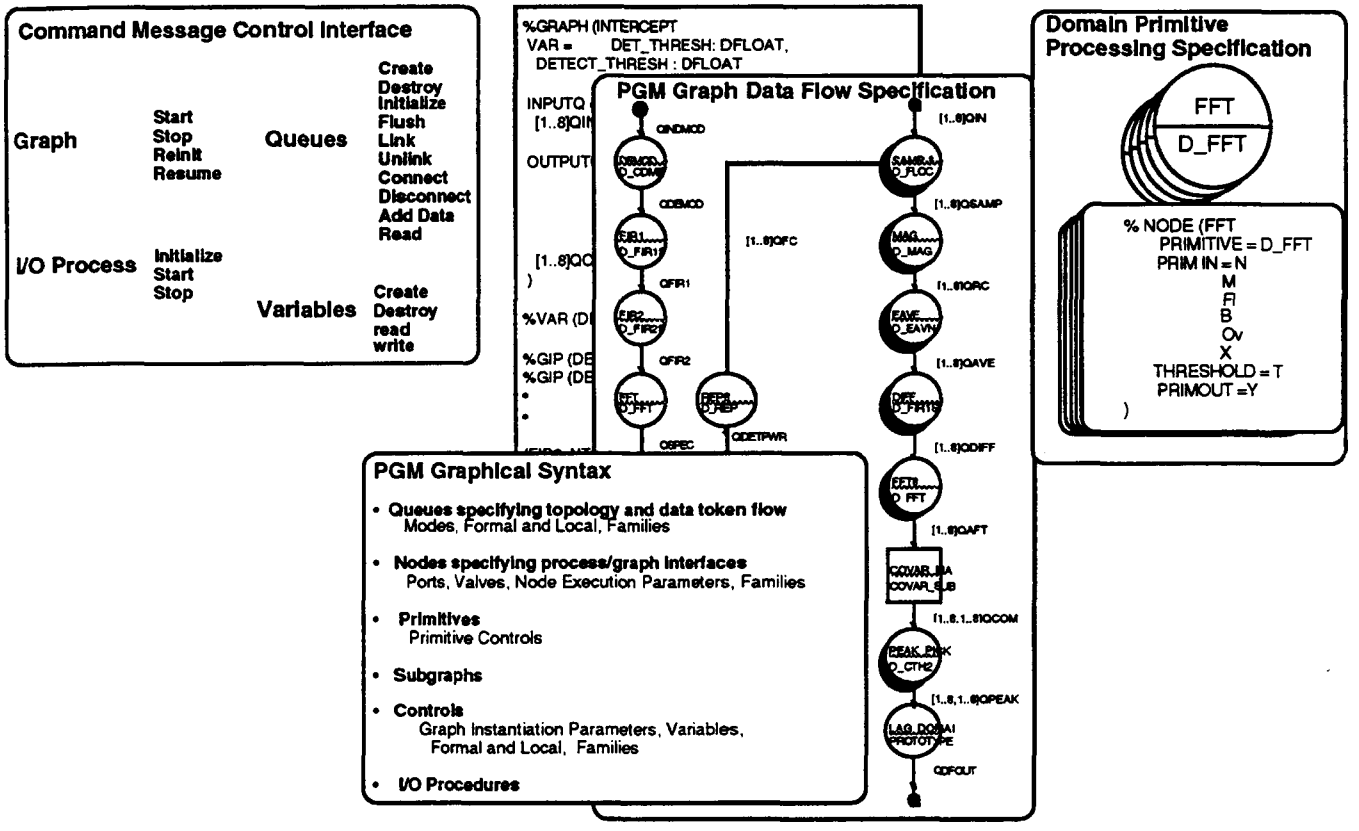

Figure 2 - Open Application Programmer's Interface

Correct by construction software development disciplines are incorporated into the codesign process. Applications are specified and autocoded using elements of an extensively tested domain primitive software library. They are assembled into executable form using mature PGM data flow rules. Validation and/or unit testing activities are incorporated at each stage of codesign to ensure compliance with functional and performance requirements. The autocoding toolset supports the RASSP program goal of uneventful integration and testing.

The autocoding toolset automates system realization of software architecture specifications. It provides automation support to design realization of the software architecture at architecture verification and detailed design levels of the codesign process. This enables rapid realization of software architecture specifications as virtual prototypes, unit testable application modules, and specifications for compilable system load images on targets supported by the model year architecture. Autocoding tools will provide the application domain engineer with the power to expand the number of application/architecture specification variations that can be evaluated to the point of acceptable cost/performance risk confidence levels within real project budgets. Superior systems and reduced development and/or recurring costs must follow.

Autocode tools automate software design realization in software architecture verification - Equivalent and partition graph generation tools automate generation of top level software designs from hardware/software architecture specifications. As illustrated in Figure 3, hardware/software architectures are specified to software verification as PGM application data flow graphs with a candidate architecture file and corresponding partition lists. A top level software design is generated from inputs and designer inputs. The top level design consists of (1) an equivalent application data flow graph which specifies the executable image of the application, and (2) a stand-alone partition graph for each equivalent node specifying executable processing. Both outputs may be used to verify requirements capture and performance of the top level software design. When accepted, these outputs may become inputs to the detailed design level autocode

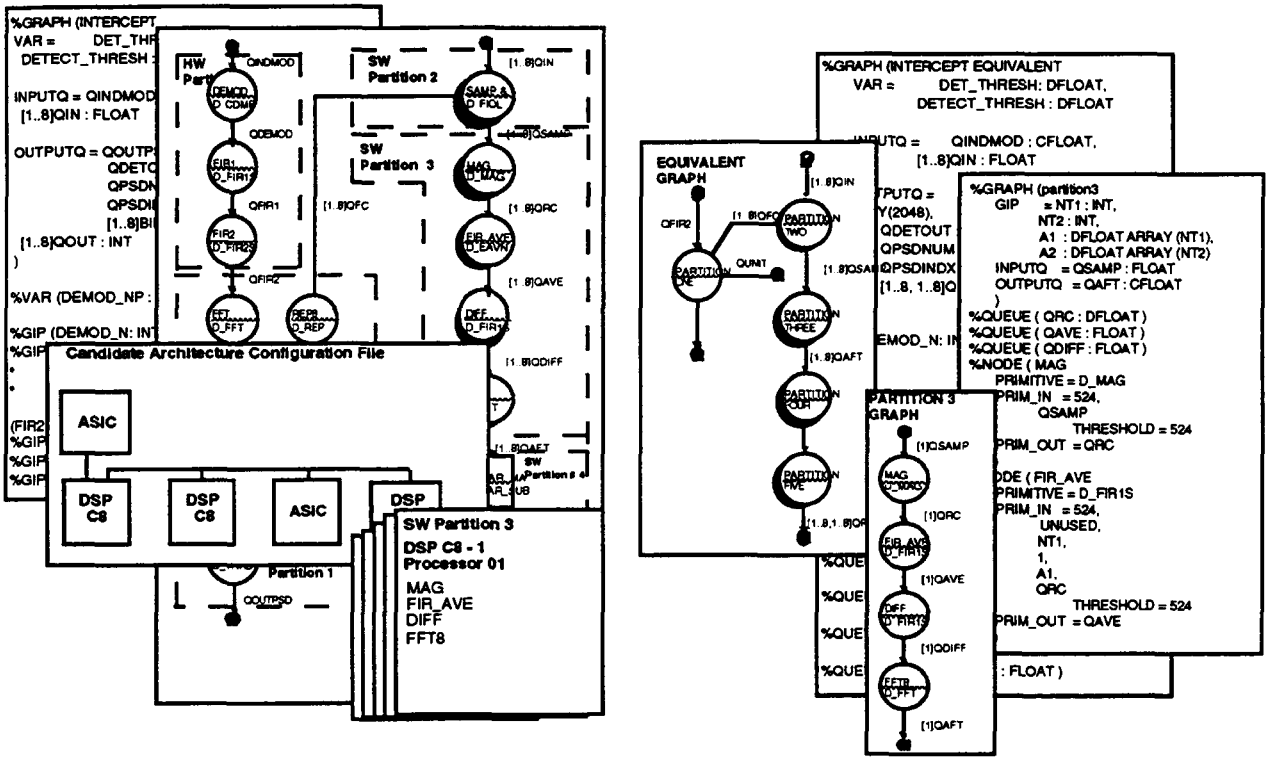

Figure 3 - Equivalent and Partition Graph Generation from Architecture Specifications

tools that generate application and executable DSP program detailed designs.

Using the equivalent graph generator tool, the software designer generates equivalent and partition graphs from the software architecture graph and input software partitions. Partitions may be the partition lists received from architecture selection or a further subdivision of these lists. Performance estimates are generated consisting of execution time estimates for each partition and equivalent graph execution time and data transfer requirements. Architecture specifications that will not translate to efficient run-time images may be quickly rejected. Top level designs may be refined. The software designer may iterate partition subdivision and corresponding data flow control parameterization to obtain software design within architecture specifications best meeting requirements for minimum resource utilization, load balancing, and latency or memory constraints.

Functional and performance simulation of the top level software design will be supported by outputs of the software verification tools. Partition timing estimates of each equivalent node will be passed to a VHDL-based performance simulator for low level hardware/software performance simulation. Functional simulation of the software design will validate requirements capture at the design level. Graph Translation Tool (GrTT) is being developed under a RASSP tech base contract. When accepted by LM-ATL•Camden, GrTT will be used to generate Ada behavior models for each partition; see Figure 4. Behavior models may be used as the executable primitives of the equivalent application graph when that graph is executed on a functional simulator. Processing results may be compared with the executable requirements specification, validating the top level software design's requirements capture. Behavior models may be created for the hardware partitions as well. These may be used to complete the functional simulation of the entire application. They also may be provided to the hardware design verification process as the procedural part of hardware partition VHDL behavior architectures.

At the completion of software verification, top level specifications will exist from which executable code targeted for the candidate architecture may be automatically generated. The software verification level autocoding tools automate the generation of equivalent and partition graph specifications. The error prone, laborious hand coding of top level specifications will be eliminated. Codesigns may be realized at the rate designers can

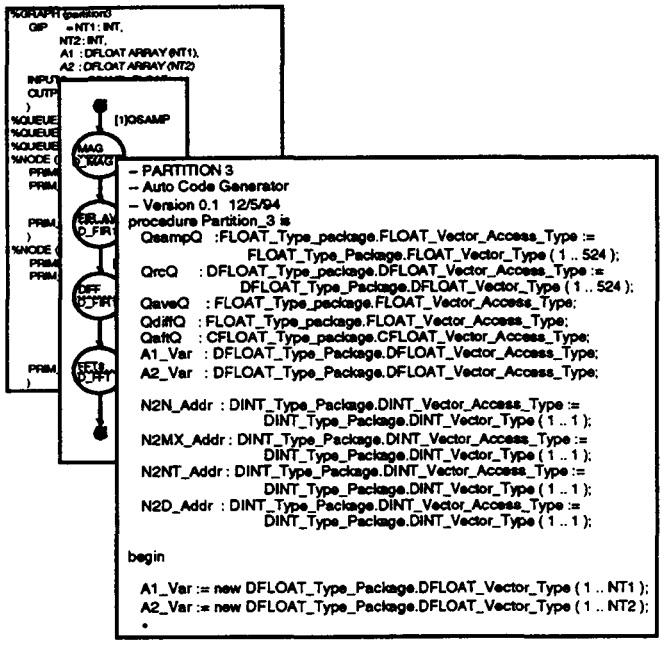

Figure 4 - Behavior Model

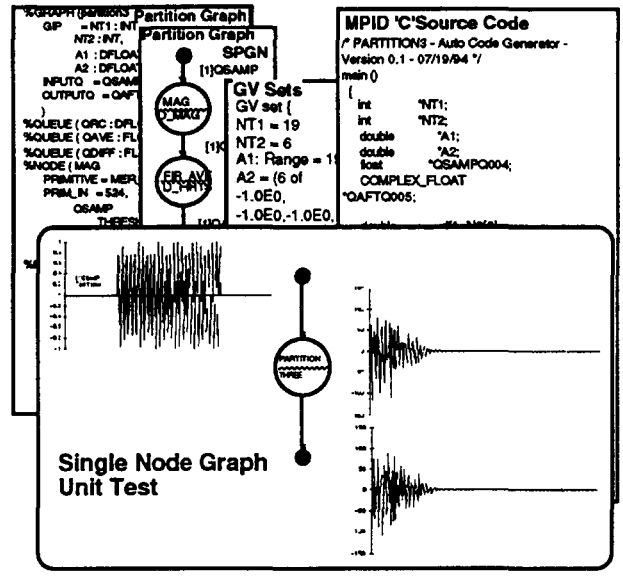

Figure 5 - MPID Generator

make design decisions and evaluate their consequences.

Autocode tools to automate detailed design and coding - Detailed design level autocode tools include the Multi Processor Interface Description (MPID) Generator and the Application Generator. These tools generate compilable images of partition and equivalent graph elements of the top level design. The role these tools play in detailed design is illustrated in Figures 5 and 6.

MPIDs are compilable programs that implement the processing specified by partition graphs. Both transient or start up and cyclic behavior of the partition graphs is preserved in the translation to compilable form as is the partition graph's response to all enumerated values of controls. At its ports, the execution behavior of the compiled MPID will be identical to the functional behavior of its partition graph specification. MPID generator will generate 'C' source code implementing the partition's processing specifications utilizing calls to the target processor's math library. A memory map converting all partition internal queues and variables to static buffers is generated. MPID generator is supported by a domain primitive database which provides constraint, error condition, target specific state machine behavior, and target performance data for each domain primitive. MPID source code will be made as efficient as possible by maximizing in-place execution of target math library calls and minimizing non-library call code to that needed to interface to the equivalent application graph and to respond to external controls.

In addition to source code for the executables, MPIDGen produces detailed performance estimates and single node equivalent graphs specifying the MPID as the primitive. The detailed performance estimates are used to validate software verification performance estimates. The single node graph supports unit testing. Unit test applications are generated using the single node equivalent graph. Test vectors from behavior models are processed to validate partition translations. Because of PGM's determinism, validation of each partition implies validation of the full application.

The Application Generator translates the equivalent application graph with its set of MPID source files into run-time data structures that are used by the run-time system to create an executable image of the application as distributed tasks on the target processors. The run-time data structures incorporate the MPIDs as executable elements of the tasks and provide other memory management and execution control information needed to realize a run-time image of the equivalent application graph.

Reusable run-time support is provided as part of the application. Figure 7 illustrates the organization of the run-time system into user-supplied signal processing and BIT applications, reusable application and load managers, and model year operating system kernels. Application data

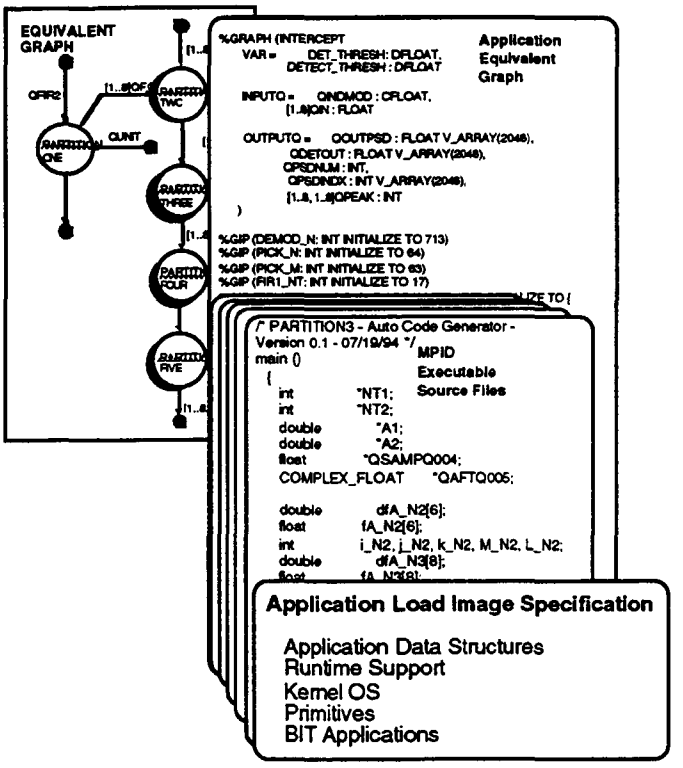

Figure 6 - Application Generator

structures provide the interface to the graph manager. The graph manager also controls the command message port. In response to command messages, the graph manager instantiates applications as tasks, connects them to data sources and sinks, initiates their processing, and applies all external controls to modify their processing. BIT applications will be handled similarly. Interfaces to the model year software are low level, simplifying the port of the run-time support to new model year computational elements.

A make file is generated that specifies the load image to the RASSP Enterprise System at the source and data structure level. This make file specifies the application data structures, source files for MPIDs, run-time support, and BIT applications. Object files for kernel OS and target primitives are also specified. The RASSP Enterprise System generates a complete load image from this specification.

Autocoding tools will contribute significantly to achieving 4x cost reduction goals - The MCCI autocoding tools will reduce labor required to generate top level software designs from architecture specifications and detailed designs implementing them. Manual coding will be eliminated altogether. Automated generation of

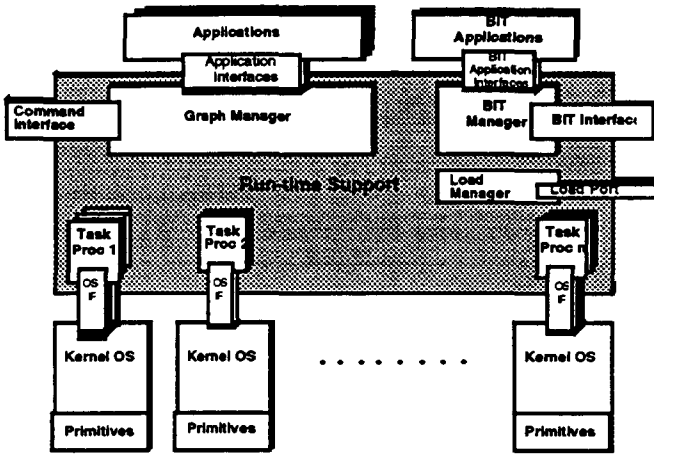

Figure 7 - RASSP Run-time Support

software designs from architecture specifications will allow meaningful evaluation of many alternate designs at a fraction of the time currently required to design signal processing systems. Automated detailed design and code generation will provide testable unit and system implementations of software designs virtually instantly compared to hand coding approaches. Systems representing a thorough design space trade off of alternative application specifications, hardware and software architecture specifications, and lower level partitioning and parameter trades will be produced rapidly. Reusable run-time support avoids an expensive development effort. Reuse of the run-time system provided as part of the reuse library and model year architecture will eliminate the user's need for expensive run-time scheduling and control support development for their software designs. The open architecture supports legacy code capture, model year application porting, and application reuse. As the RASSP Enterprise System gains legacy, application reuse opportunity will increase.

Demonstrations - Use of the autocode tools in software verification and detailed design elements of the hardware/software codesign process are demonstrated. Development of an executable design realization of the LM-ATL•Camden SAR benchmark is demonstrated. The demonstration includes a prototype MPID generator and run-time support with hand simulations of equivalent graph and application generation. Execution and control of the running autocoded SAR application is being demonstrated.