

An Application of VHDL-Based Hardware/Software Codesign

David Black
TRW Space and Electronics Group
Avionics Systems Division, San Diego, CA 92128 USA
black@rc.trw.com

ABSTRACT

This paper addresses the applicability and use of processor synthesis tools and VHDL to support a codesign methodology. It suggests the features required for a viable tool to support high engineering productivity of digital processing systems. It reports on their application to communications and signal analysis digital hardware designs and involves pulse detection and waveform processing of high rate or complex communications and radar signals. This use is appropriate to the Consumer Electronics, Telecommunications, and Aerospace industries and the Military where requirements for minimum size, weight, and power prohibit the exclusive use of Commercial Off-The-Shelf (COTS) processor solutions. A particular processor synthesis tool is used as an example in this discussion.

INTRODUCTION

Shorter time-to-market requirements, leaner development budgets, and the continuing rapid advance of technology all place pressure on the engineering development environment to increase productivity and to support continuing product improvements with a minimum of redesign. Indications are that minimum productivity improvements of one hundred percent are needed. Rapid virtual prototyping with a hardware/software codesign process contributes to increased productivity for complex embedded digital systems. For custom embedded digital processors, a capability to specify high performance, hardware-independent system functional behaviors can contribute greatly to system performance analysis, design verification,

and product reuse when coupled with an efficient hardware synthesis capability.

The key to solving this emerging design productivity problem is the use of codesign tools within a concurrent engineering environment. High level programming languages or hardware description languages alone will not solve this problem. Instead, the designer must be afforded the ability to efficiently trade between hardware and software elements in the design and to evaluate and verify the results through accurate simulation. It is also desirable that both a rigorous hardware description and the embedded software/firmware code be automatically produced. In this paper, hardware/software codesign is applied to digital signal processing. The application of codesign is defined, for purposes of this paper, as the joint design of a processor architecture and a behavioral specification to be executed upon that architecture. Hardware and software engineering disciplines are associated in a tightly coupled, concurrent development environment with each influencing the other.

A variety of advanced computer aided engineering (CAE) tools are being developed which support this codesign process. Various methods are being researched to address this problem. For example, work of the "Hardware/Software Codesign Project" of the University of Queensland, Australia; for example, language translators [1] and hardware/software profiling [2]. Another is the COMET program for Cosynthesis at Board and MCM Levels for Digital Signal Processors at the University of Cincinnati; reference a description of the COMET environment [3] and a distributed high-level synthesis system [4]. Much work is using the Ptolemy framework for simulation,

prototyping, and software synthesis for heterogeneous systems; for example, references [5] and [6]. ProcSyn [7] is an example of a synthesis tool for creating custom, high speed programmable processors. This tool uses a different approach to hardware/software codesign than that of the prior references and is further explored as an example of such codesign in this discussion.

ProcSyn generates VHDL code based upon an executable specification of the desired application behavior. This specification is written in a high level language (HLL) such as Ada or C. It is analyzed by the tool and a processor is synthesized to optimize execution of the HLL program's behavior.

Specific discussions in the use of the ProcSyn tool are based upon TRW's work under the ARPA DICE program, contract MDA-972-92-C-0021. DICE is the (D)ARPA Initiative in Concurrent Engineering. The Advanced Research Projects Agency, ARPA, is an agency of the U. S. Government. ProcSyn is a digital processor synthesis codesign tool developed by JRS Research Laboratories Inc. of Orange, California.

PROCESSOR SYNTHESIS FEATURES

The benefits of processor synthesis in a hardware/software codesign environment should include application behavioral requirements to be input as a high-level language executable specification, automatic generation of a hardware control architecture, and very high utilization of silicon resources for the targeted application. When the executable specification becomes the program to execute on the synthesized processor, then the system test vectors can be carried to the silicon level which enables efficient verification of requirements.

Compiler retargeting technology is the core of this type of CAE tool. To support maintainability and future product enhancements to existing hardware, processor synthesis tools should output a compiler targeted to the synthesized processor. It should generate microcode from other programs coded in the HLL, not merely from the original executable specification. It is critical that the application program does not describe the processor

hardware, but rather it describes the desired application behavior. An interactive development process with re-design interactions cycling on the order of hours or days is needed to support rapid prototyping and increased productivity.

A significant feature in a processor synthesis tool includes user customization of hardware elements to add processing functions at key points of the processor design. Yet, involvement by the user at the microcode level should not be required. Without pinpoint inclusion of user defined components automatically integrated into the processor architecture, the maintainability and ease of programming in an HLL would not satisfy the millions or billions of operations per second required of some applications.

A feature important for practical synthesis is the inclusion of options to perform tradeoffs of hardware (silicon) to performance. In order to minimize cost, a facility is suggested for the user to constrain the architecture by reducing the number of processing resources; for example, adders, shifters, multipliers, and register files. Another option is that to minimize microword width and reduce architectural connectivity. An accurate mapping to various silicon technologies is important to support design tradeoffs. Back annotation of technology timing data is necessary to support compiler synthesis and thus performance analysis. Hardware/software performance reports must give the user clear visibility into processor operation in order to expose performance bottlenecks. Such visibility can lead to algorithm improvement of the executable specification, pinpoint hardware optimizations, and architectural rule modifications.

Architectural rule modification may or may not be a critical factor in processor synthesis. High resource interconnectivity often improves performance, but at the cost of increased silicon. Also, processor sequencer interrupt vector tables to permit fast, non-context, event branches may or may not be required for some applications to meet performance requirements. Other features that are useful in processor synthesis include automatic reduction of lightly used resources, an integrated software development environment, and power, size, and reliability reports.

THE PROCSYN CODESIGN TOOL

ProcSyn is a VLIW (Very Long Instruction Word) processor synthesis tool. It utilizes the concept of hardware and software codesign to develop solutions to implement digital systems based upon a single-chip embedded custom digital processor. See the References section of this document for a selection of papers relevant to the development of this tool.

The input to the tool is an executable specification of the application system. This specification is described in the Ada programming language and encompasses the application's behavioral requirements. The tool generates a single processor implementation to efficiently process the behavior described in the Ada specification. ProcSyn outputs a VHDL description of the processor compatible for gate level synthesis. ProcSyn also generates and outputs microcode of the Ada specification which directly executes upon the processor VHDL description. Also output, is a code generator for converting Ada to processor microcode and a simulator for the custom processor.

Designs utilizing a ProcSyn processor typically yield different architectures than traditional designs. A ProcSyn architecture is based upon a modified crossbar switch capable of producing high utilization and concurrent utilization of many resources. Control and resource management is handled by ProcSyn. ("Resources" in this context are the transformation and storage elements which make up a ProcSyn processor; e.g., adders, shifters, multipliers, register files.) The user only writes code necessary to describe the behavior of the system functions. The user does not compile the code by hand nor generally attempt to control the specific resources of the processor. Otherwise, this would be much too complex and would present a significant maintainability problem.

ProcSyn has the capability for using custom, built-in-functions (BIF) to enhance overall processor performance. A BIF is used when a combination of basic resources are insufficient to achieve performance requirements. The user designs the BIF which typically represents a critical segment of the behavioral coding. For instance, the early ProcSyn reports of the Demonstration system algorithms indicated

insufficient performance. However, analysis of the reports indicated specific areas of the Ada code where BIF could be used to improve performance. A segment of the code was then replaced by a single call to the BIF. With some design iterations, performance was met.

DEMONSTRATION

The purpose of the DICE Demonstration is to show whether or not the JRS Research codesign processor synthesis tool, ProcSyn, can be used to greatly reduce the engineering development costs to produce digital hardware systems. We used an existing signal processing system as the baseline metric with which to compare the ProcSyn based development process to the more traditional process used to produce the existing system. The demonstration chosen is taken from a system used for high performance signal detection and analysis. An eight-module section from the system is the DICE Demonstration baseline.

To implement the selected demonstration application, ProcSyn required a different architectural approach than that of the original development. The DICE Demonstration task did not use any development work from the original effort other than Acceptance Test vectors and test equipment software required to validate the ProcSyn based development effort. The system and signal environment requirements were the input to the DICE demonstration effort.

Scoping the Problem. An initial system engineering judgment is made to identify the most appropriate part of the system to model for ProcSyn based development. An understanding of system requirements is primary to producing algorithms necessary to implement the application using ProcSyn. Since ProcSyn is a synthesizer of a digital processor, the application needs to be described as a program. The program is written in the high level language of Ada. Here, we apply ProcSyn to problems that cannot be solved with a COTS processor, but we use programming in the same way we would if using a COTS processor.

While ProcSyn is limited to producing a single chip processor design, it is best to scope the problem as large as practical, without prejudice. The codesign of hardware and software afforded

by ProcSyn yields an engineering style and system architecture that is often more efficient in terms of hardware than that which traditional design methodologies produce. It is our experience that it is better to start with a system segment that may be too large for ProcSyn rather than too small. That is, the segment may not realize sufficient performance or the synthesis may result in a design which would be too large for fabrication in current technology. However for an initial baseline, we have found it is better to back off from a larger functionality than to add to it. The other advantage of ProcSyn is that it can rapidly (on the order of days or less) determine feasibility. So, it is best to scope the system segment as large as possible to take advantage of the tool's potential to produce hardware with a high utilization.

Certain aspects of a design need not be part of the ProcSyn based model. Obviously, analog parts necessary to meet particular system requirements are inappropriate. Large memories are typically impractical for residence within a ProcSyn processor due to fabrication limitations.

Executable Specification. Once a system is selected, the system behavioral requirements are described as an executable Ada program. Since ProcSyn does not contain any user accessible development tools for the Ada code, the program must be developed by other means. ProcSyn compiles the program Ada to an intermediate language which is similar to assembly language. The user may read this but is not expected to program in the intermediate language; all programming is in the HLL. Many compile-time options are available to aid the user in optimizing code execution and code size. For example, loop unrolling, common subexpression elimination, and invariant code motion control. Not all Ada constructs are supported by the ProcSyn environment. These unsupported features are documented in JRS Research Laboratories documentation.

A particular programming style is appropriate to Ada development for ProcSyn. Attention to algorithm structure is particularly important when programming for high performance, real-time systems. A style to minimize data dependencies and complex data addressing

structures helps to maximize processor performance.

Machine Synthesis. This phase of development using the ProcSyn tool requires significant user involvement. This is the most productive phase of development under ProcSyn. ProcSyn produces reports of performance of the specification and the machine generated. The interactive cycle of virtual machine build, machine analysis, and algorithm or hardware modification can be very fast; on the order of minutes or hours.

An approximation of software performance as executed on the virtual machine hardware permits the rapid cycle mentioned above. Once the user is satisfied with the estimated performance, a more detailed synthesis is performed. ProcSyn produces a code generator for the actual processor hardware to be built. Then, microcode of the user's program is produced using the code generator. These two steps may take hours or a day.

A final analysis is performed on the application software as executed on the synthesized hardware. This is not an approximation, but yields actual performance metrics. Numerous, detailed reports are available to the user. These reports indicate accurate resource utilization, resource conflicts, memory utilization, and other details needed to locate performance bottlenecks or low utilization of resources.

Design Export. The final output of ProcSyn is the (gate-level) synthesizable VHDL (IEEE Std. 1076-1987) of the processor hardware and the machine code of the user's program. A code generator is produced and may be used to generate code for other programs to run on the target processor. A simulator for the processor is also output by ProcSyn.

DEMONSTRATION RESULTS

The baseline design is an eight-module unit from a high performance signal detection and analysis system. Figure 1 is a functional block diagram of the system. Note that "CCA" refers to a circuit card assembly, (a circuit board with parts assembled onto it).

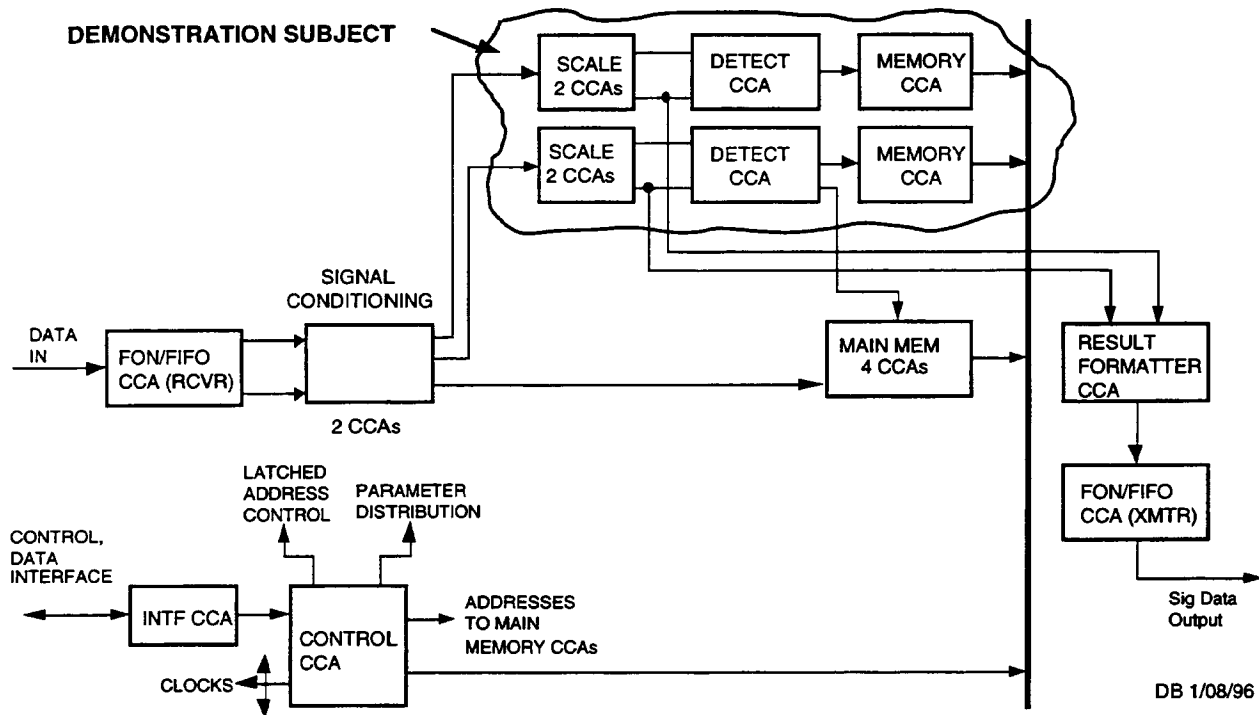
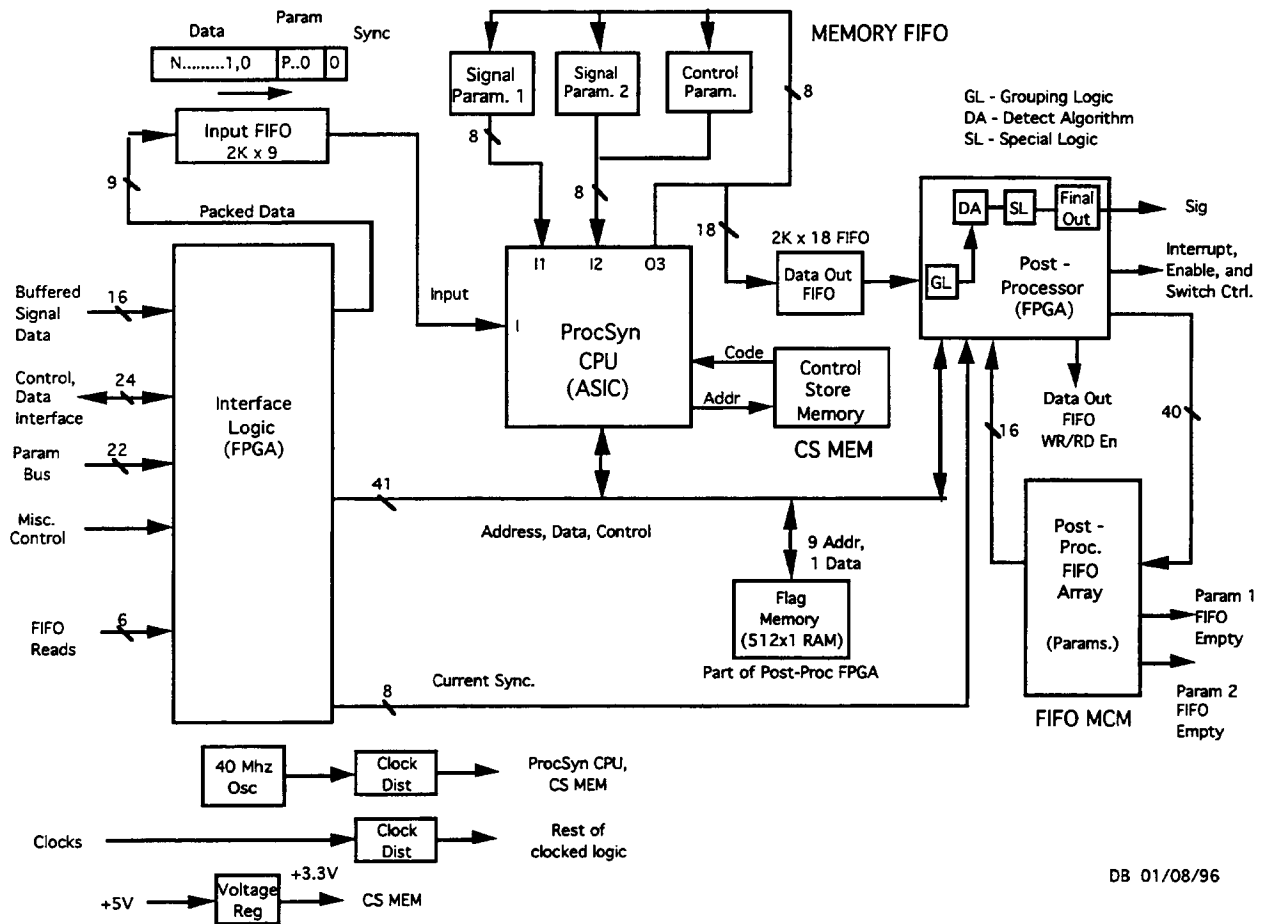


Figure 1. Baseline System Block Diagram

The Demonstration subject, chosen from an existing system, was selected based upon its perceived applicability to a general processing architecture. The functionality was clearly defined and suitable to algorithmic description in a digital domain. The eight-module section chosen for demonstration was originally implemented in a straightforward architecture with functional partitioning yielding specialized hardware modules. A simple control architecture was chosen to minimize functional risk. The trade-off is that more hardware is required for the implementation. For the Demonstration project it was expected that by assuming the availability of custom processor synthesis, a new algorithm could be implemented with the same functionality and performance of the original design. Furthermore, it was expected that a better utilization of hardware could be effected. Since the processor synthesis tool would handle the more complex hardware control requirements, functional risk was low. This turned out to be the case.

The Demonstration hardware was designed and can be implemented in one Standard Enclosed Module, size E (SEM-E). The design consists of the ProcSyn generated ASIC (Application Specific Integrated Circuit), an MCM (Multi-Chip Module), two FPGA (Field Programmable Gate Array), four synchronous memory components, memory buffers, lower density logic, and discrete components. It operates from a positive five volt DC power supply and at clock frequencies up to 40 MHz. It is designed for laboratory conditions. Figure 2 is a functional block diagram of the Demonstration hardware.

The ASIC is the ProcSyn processor, or CPU (Central Processing Unit). This component implements the bulk of the Demonstration baseline functionality. It operates based upon a 40 MHz clock and yields a basic performance of approximately 110 MOPS. The CPU is a 16-bit machine based upon the ProcSyn data crossbar architecture. It is designed for CMOS, submicron technology and consists of approximately 170 thousand gates. Figure 3 is a functional block diagram of the ProcSyn CPU ASIC. In addition to



DB 01/08/96

Figure 2. Demonstration Block Diagram

program sequencer control, clock phasing, diagnostic logic, and the crossbar switch the primary components of the CPU are:

- 1 Address Generator with a multi-port 32-word register file
- 4 Boolean, Arithmetic, Logic Units each with a multi-port 32-word register file
- 1 Integer Multiplier
- 1 Barrel Shifter
- 1 64-word Internal Memory
- 4 Special I/O Built-In-Functions (BIF), the input and output FIFO functions
- 1 Special Internal BIF, the detection function

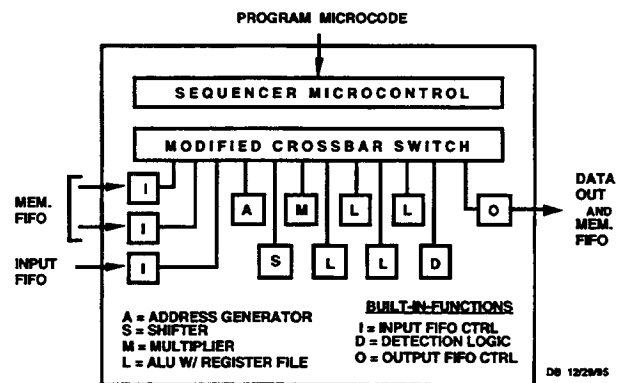


Figure 3. ProcSyn CPU Block Diagram

CONCLUSION

Use of the ProcSyn tool indicates significant reductions in development and recurring costs as compared to those of the baseline system. A reduction from eight modules to one module can be realized with the system based upon a ProcSyn architecture. The ProcSyn tool produced synthesizable VHDL of the embedded signal processor. And, from the TRW application Ada code, the tool produced the microcode to execute on the processor. All component parts of the ASIC were synthesized to the gate level and with timing constraints to meet a 40 MHz design. System simulations were executed for the target application and component level hardware simulations executed for all FPGA and MCM of the module design.

We found that by requiring executable specifications as input, ProcSyn aided greatly in system engineering aspects of the design. ProcSyn indicates performance of a highly interconnected, multiple resource architecture to maximize utilization of parallelism in a given algorithm. It indicates the size of a design and thus its feasibility to build. It is not necessary to perform gate-level synthesis or detailed simulations beyond the ProcSyn environment to determine a design's feasibility. So whether or not a design produced by ProcSyn may be built, we have found its rapid modeling and system analysis power to be constructive in product development. With hardware/software cosimulation and with analysis reports that indicate hardware and software performance details to a high fidelity, the designer may rapidly optimize the design to the particular application.

The ProcSyn tool is indicated for use in implementing system functions which include parallel and independent functions with minimal data dependencies. Clearly this is appropriate as the ProcSyn architecture is based upon a crossbar multiplexor to interconnect multiple resources. Additionally, the user's ability to constraint hardware, permits very efficient silicon utilization. For example, if a set of algorithms do not need floating point operations, then the floating point unit need not be included in the custom processor architecture. Earlier benchmarks indicated ProcSyn generated CPU yielding four times the processing throughput as compared to a major COTS DSP (Digital Signal

Processor) CPU. System functions requiring moderate latency are appropriate. Systems with very low latency requirements may not be applicable. However, the Demonstration architecture included four high speed, low latency external memory buffer interfaces which integrated into the ProcSyn CPU. These were implemented using the tool's capability to integrate user created built-in-functions.

Finally, ProcSyn is a tool currently in development. It is not yet in a state for general, commercial use. Throughout the DICE program, the tool was repeatedly stressed to the breaking point. Those hurdles were surmounted and most deficiencies were corrected within the tool; in some cases a workaround was required. Some items remain outstanding. We found the tool to significantly reduce our engineering development costs and technical risk. Sometimes, expert support from the vendor was required to operate the tool.

FUTURE OF PROCESSOR SYNTHESIS CODESIGN

The concept of high level behavioral specification is not new. Yet processor synthesis tools such as ProcSyn are rare. The potential to advance the development and maintainability of high performance signal processing applications in a highly constrained environment is great. Continued development of the ProSyn tool is indicated. More effort is required to realize the commercial use of these types of tools. The key here is the integration of compiler retargeting technology applied to a flexible, user adjustable hardware architecture.

ACKNOWLEDGMENTS

We gratefully acknowledge the Advanced Research Projects Agency for supporting the work for which this paper is based. The related work was performed on the DICE program, contract MDA-972-92-C-0021, at the Avionics Systems Division of the TRW Space and Electronics Group. ProcSyn is a CAE tool developed by JRS Research Laboratories Inc. of Orange, California.

REFERENCES

- [1] Matthew F. Parkinson, Paul M. Taylor and Sri Parameswaran, "C to VHDL Converter in a Codesign Environment", Proceedings of the VHDL International Users Forum, May 1994.
- [2] Matthew F. Parkinson and Sri Parameswaran, "Profiling in the ASP Codesign Environment", 8th International Symposium on System Synthesis, September 1995.
- [3] Ranga Vemuri, Hal Carter and Perry Alexander, "Board and MCM Level Synthesis for Embedded Systems: The COMET Cosynthesis Environment", a version presented at the First Annual RASSP Conference, August 1994.
- [4] Jayanta Roy, Nand Kumar, Rajiv Dutta and Ranga Vemuri, "DSS: A Distributed High-Level Synthesis System", IEEE Design & Test of Computers, June 1992.
- [5] Asawaree Kalavade and Edward A. Lee, "A Hardware/Software Codesign Methodology for DSP Applications", IEEE Design and Test of Computers", September 1993.
- [6] Asawaree Kalavade and Edward A. Lee, "Manifestations of Heterogeneity in Hardware/Software Codesign", Proceedings of the Design Automation Conference, June 1994.
- [7] E. Warshawsky, "ASIC Technology and the Selection of Processors for DoD Systems", Proceedings of the IEEE 1990 ASIC Seminar and Exhibit, Rochester, New York, September 1990.

A Selection of ProcSyn Related Documents

-- TRW Space and Electronics Group, "ProcSyn Report", delivered under contract to ARPA, February 1996.

-- JRS Research Laboratories Inc., "Software User's Manual for the IDAS DICE Ada-to-Microcode Compiler System".

-- D. Runner and E. Warshawsky, "Synthesizing Ada's Ideal Machine Mate", VLSI Systems Design, December 1988.

-- Robert Sheraga and John Gieser, "Experiments in Automatic Microcode Generation", IEEE Transactions on Computers, June 1983.

-- John Gieser, "On Horizontally Microprogrammed Microarchitecture Description Techniques," IEEE Transactions on Software Engineering, September 1982.