

Reengineering Legacy Embedded Systems

VIJAY K. MADISETTI
YONG-KYU JUNG
MOINUL H. KHAN
JEONGWOOK KIM
VP Technologies, Inc.

THEODORE FINNESSY
US Air Force Research
Laboratory

Systems that require reengineering are a fact of life in both the commercial and military arenas. The authors have developed a systematic methodology based on virtual prototyping, with accompanying tools and libraries, for rapidly upgrading electronic systems.

LEGACY SYSTEMS are hardware and/or software systems currently performing useful tasks but requiring reengineering or upgrading for various reasons. The most pressing reasons are parts obsolescence and system needs such as greater functionality, increased processing and interface scalability, better form (size, weight, power, volume), and decreased maintenance and life-cycle support costs. Another reason is the availability of superior algorithms, architectures, and technologies that meet or exceed the system's specifications, often at a lower cost.

Legacy systems abound in both the military and commercial electronics arenas. Indeed, commercial electronics systems, such as PCs and cellular phones, are often obsolete in a matter of months, and time-to-market pressure has institutionalized product reengineering. In the military, the long lifetimes of deployed systems—decades in the case of radar systems—make the problem of reengineering legacy systems inevitable. It is no exaggeration to say that the question is not whether these systems will require engineering, but when and how it should be done.

One can reengineer a legacy system in an evolutionary manner by upgrading portions (usually hardware) of the system, or in a revolutionary manner by redesigning the entire system. A legacy system usually represents a high value and a high cost investment: years of accumulated knowledge and a large

base of users who rely on its operation. At a minimum, the reengineered replacement system must preserve the legacy system's functionality, performance, and interfaces.

At VP Technologies we have developed a successful reengineering approach based on virtual prototyping. Our approach relies on tools and extensive libraries of models that are simulatable and synthesizable at multiple abstraction levels, from specification to detailed implementation. The underlying technologies of virtual prototyping, automated library model generators, flexible processor architectures, and advanced optimization algorithms facilitate increased reengineering capabilities at a reduced cost.

Overview

Existing reengineering approaches focus on hardware upgrades, often at the single-chip level. This one-component-at-a-time approach does not extend to board- or subsystem-level reengineering and an accompanying software upgrade. For some legacy systems, a single, form-efficient IC might replace many old chips; thus, a board-level approach would be of much value. Often, we lack design specifications for the legacy system, and the original designers are no longer available to provide information. In such cases, we need methods for extracting the legacy system's design intent.

Maintaining legacy systems is expensive. For example, it costs \$5 million a year to

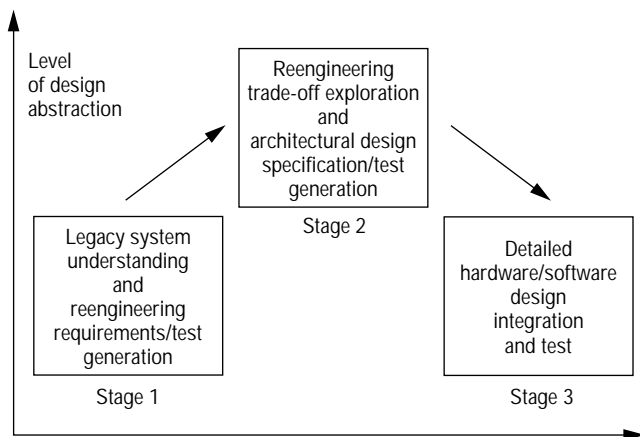


Figure 1. Legacy system reengineering process.

maintain 20-year old functionality in a radar system. Thus, current research efforts focus on developing reverse-engineering, reengineering, and manufacturing tools necessary to design and manufacture cost-effective replacements for electronics products affected by parts obsolescence. This research includes developing methods for extracting useful and accurate reverse-engineering information from legacy data sources. It also includes developing automation tools and an infrastructure to reduce the costs of reengineering obsolete components and systems. We also expect future system designs to allow quick, cost-effective upgrades through advances such as virtual machines, open-system architectures, and software middle layers.

Reengineering process

Figure 1 shows our three-stage reengineering process. In the first stage, we recover design requirements and generate test vectors from the legacy system, thus attempting to understand the system. We use existing manufacturing data, netlists, specifications, test vectors, code dumps, and listings to develop an executable virtual prototype of the legacy system in a language such as VHDL (VHSIC Hardware Description Language) or UML (Unified Modeling Language). In the second stage, we feed the generated executable requirements along with cost and form factor constraints into trade-off advisors to decide on the right architectural and test specifications for the reengineered system. The third stage completes the detailed hardware-software design as well as system integration and testing. Thus, stage 1 focuses on design intent abstraction (or reverse engineering), while stages 2 and 3 are similar to conventional top-down design synthesis methodologies based on virtual prototyping. Supporting each stage are extensive libraries of cost, performance, and clock-accurate, fully functional simulation and synthesis models.

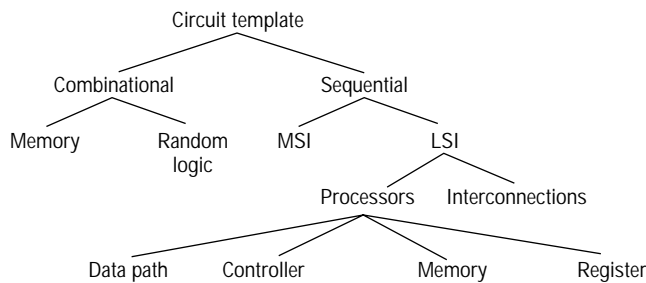


Figure 2. Candidate templates for systems, subsystems, and components stored in template library. Designers can match a legacy ASIC to a circuit template.

Legacy system understanding. The first step in reengineering a legacy circuit card assembly (CCA) is to abstract its design intent and generate executable requirements. In most cases, information about the legacy component or its interface is incomplete or missing. Automated tools can assist in the discovery of the component's underlying structure and behavior. Although system understanding and design intent abstraction is not yet a mature science, it benefits significantly from automation. In addition to design libraries of components obtained from organizations involved in designing the legacy system, automation relies heavily on template-matching algorithms (see Chisholm et al., page 26).

Let's assume that a structural transistor-level netlist of each CCA component can be extracted through etching and circuit geometry abstraction (see box, page 34). We then convert the transistor-level netlist to a gate-level netlist, using a library-based approach that includes describing the gates' transistor implementation. Next we use iterative, search-based pattern-matching procedures to extract the circuit's class (or its canonical template) and structure at multiple levels of abstraction in the template library (see Figure 2). After identifying key components (or their sub-components) in the legacy CCA, we can select suitable replacements for them from the candidate component library. The design tools invite designer input to direct this semiautomated behavioral abstraction process.

Figure 3 (next page) diagrams this process. The quality of its results depends on the breadth, accuracy, and depth of the associated libraries and the effectiveness of the search algorithms that recover the legacy component's design intent. (These libraries and algorithms represent intellectual property and have been the subject of considerable recent research.) Partial or complete documentation of the legacy system's specifications can help the designer intervene at key decision points to bootstrap the process of system understanding.

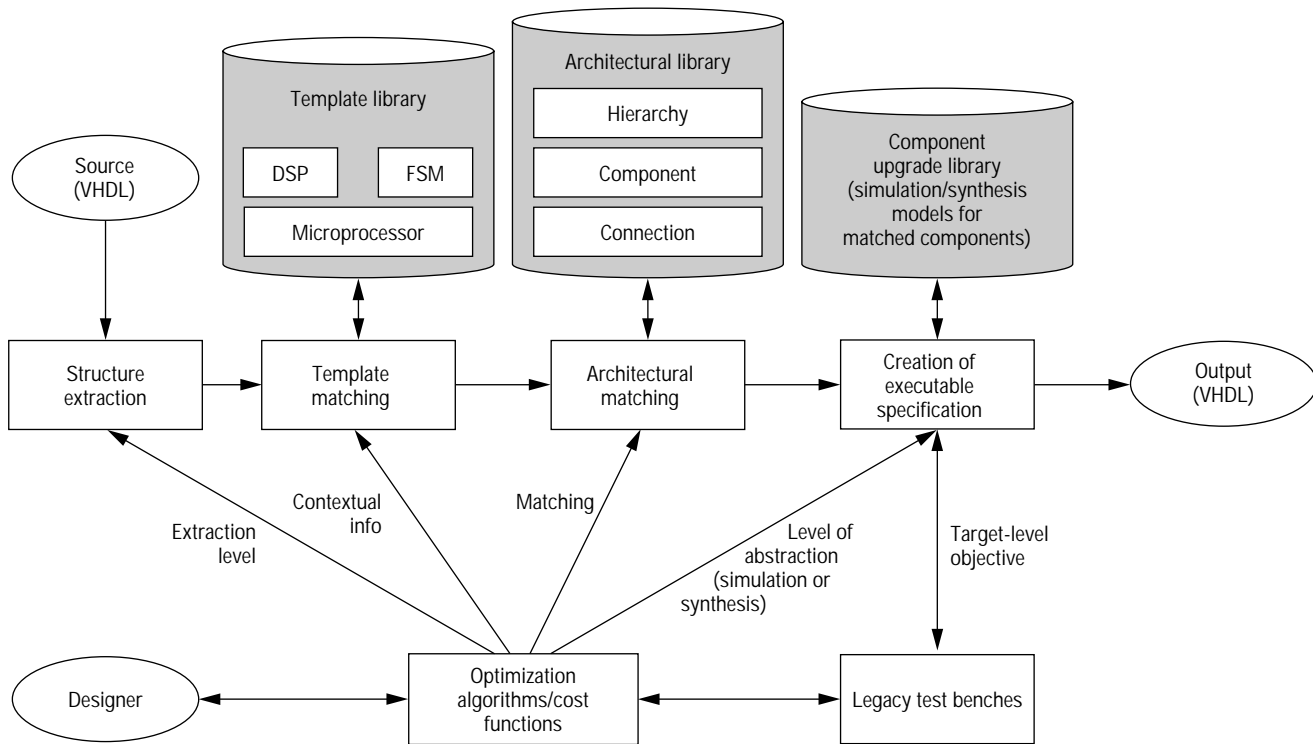


Figure 3. Behavioral abstraction process.

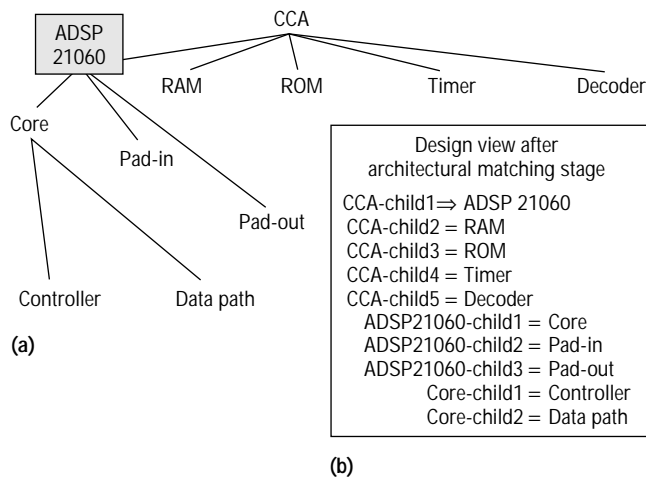


Figure 4. An ADSP21060-based CCA after architectural matching stage: graphical interpretation (a); recovered netlist (b). Both functionality and interconnection information are recovered.

Figure 4 shows an example of the abstracted design view after the architectural matching stage of the process. The abstraction tool recovers the system design through bottom-up/top-down identification of its components and their in-

terconnections, and further decomposes each component into its constituent parts. The designer then navigates through the various component models at multiple levels of abstraction, selecting the components that need reengineering. In the following stage, the designer replaces these components with simulation or synthesis models obtained from the component upgrade library. Together, all component models (and associated software) form the executable requirements for the legacy CCA and its test benches.

Reengineering trade-off exploration. The second stage in the reengineering process evaluates the architectural changes necessary to upgrade the legacy system and their effect on the reengineered system’s cost and quality. This stage reveals architectural errors, reduces reengineering time, and links cost issues with the detailed hardware design and software synthesis. To perform architectural evaluation, we must capture the system’s execution characteristics even before detailed hardware and software are available. Effective virtual prototyping of the system at a high level of abstraction allows easy evaluation of architectural trade-offs.

Figure 5 illustrates the reengineering trade-off exploration stage. After the system understanding stage, we draw performance-level components from the reuse library to model an executable virtual prototype that adequately captures

the legacy system's communication and computation characteristics. We generate and preselect a set of software and hardware design architecture candidates, with the help of system reengineering requirements and cost-model-based decision tools. Virtual exploration consists of executing the system's performance model to reflect various reengineering options, performing hardware-software cosimulation, and evaluating the results. Design evaluation compares various reengineering options through analysis. This analysis also reveals architectural bottlenecks and errors that result if the system's performance requirements (such as real-time guarantees, error/fault performance) are not met.

After several iterations of the exploration steps, the design converges to a feasible solution. Then, the code generation step allows the designer to link the high-level software model with target-specific code generation tools, which reduce software-reengineering effort. Detailed hardware/software integration links the performance model to the fully functional model of the system, allowing detailed reengineering to take place. The link between the fully functional and performance models allows back-annotation if, during detailed design, we detect an error propagated from this stage.

VP's methodology for architectural trade-off evaluation and exploration involves modeling system hardware (processor type, interconnection architecture, memory architecture, and so on) and software (task partitioning and allocation, communication and computation schedules). Our tool suite is supported by a set of hardware component libraries including processors (such as the Mil-Std-1750, the ADSP Sharc, and the TI-62) and interconnections (the Mil-Std-1553, the Mil-Std-1773, the SCI, and the VME64). The models are available at the fully functional, hybrid, and performance levels (see Madiseti and Egolf¹ for a description of the taxonomy of virtual-prototyping model libraries and their abstraction levels).

Similarly, our software-modeling toolkit uses generic in-

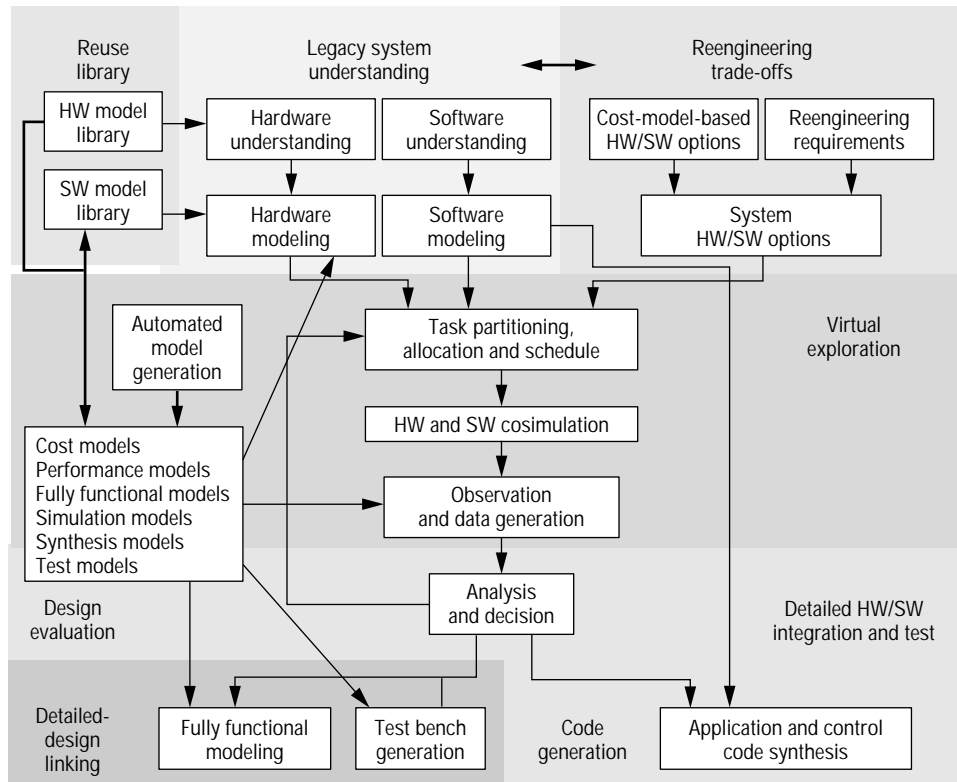


Figure 5. Virtual-prototype-based reengineering trade-off exploration.

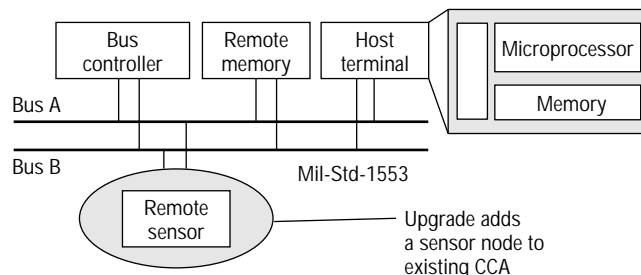


Figure 6. A typical avionics architecture based on the Mil-Std-1553 serial multiplex bus. Here, we add the remote sensor node to the legacy single-board computer.

struction modeling supported by rich libraries of computation, communication, and control primitives. These primitives are linked with the target-specific code libraries to generate system software through autocoding. The software-modeling toolkit allows abstraction of existing software. The designer can evaluate the design and addition of new software before developing the actual software.

Figure 6 shows a commonly used avionics architecture that uses the Mil-Std-1553 serial multiplex data bus for communication between various subsystems. The system con-

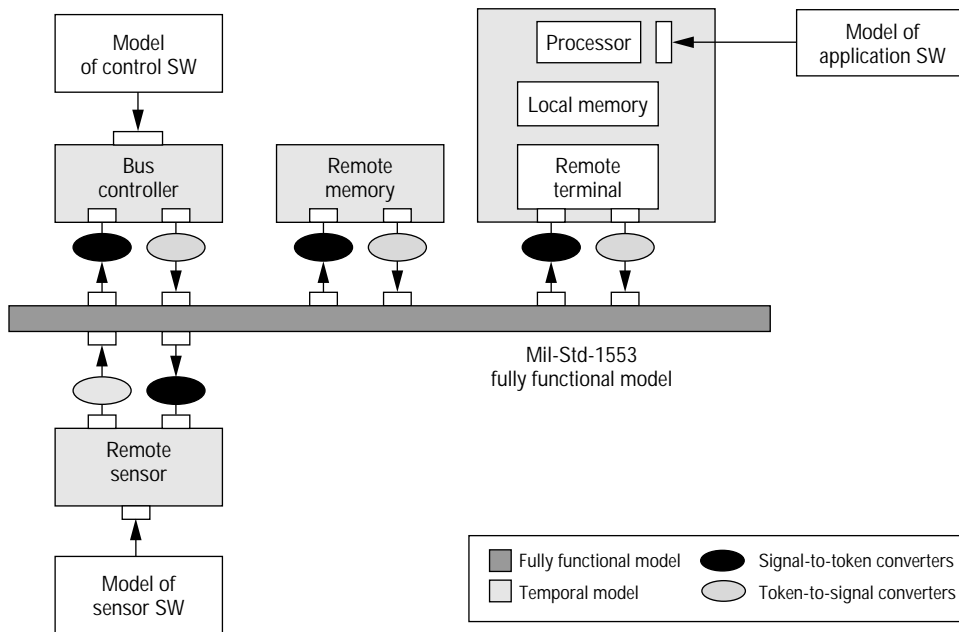


Figure 7. We created this executable virtual prototype of the Mil-Std-1553-based architecture shown in Figure 6 for architectural trade-off exploration and evaluation. It includes both hardware, interconnections, and application/control software executing on the system's various nodes. A fully functional model captures both functional and timing (clock-level) behavior, while temporal models (or performance models) capture only timing behavior, requiring the use of token-to-signal translators for a hybrid model.

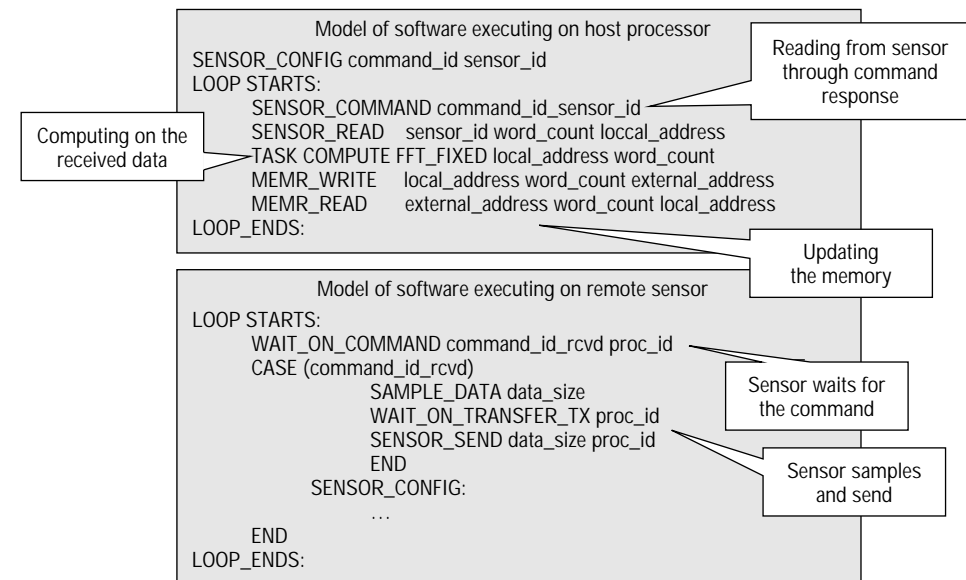


Figure 8. High-level software models executing on various nodes prior to detailed software and hardware design allow rapid development of the virtual prototype. We then use this pseudolanguage to generate control software for the actual hardware/software system, cutting design time further.

sists of a number of subsystems: fire controller, fire control navigation, air data unit, target identification set, radar display, head-up display, and others. Each subsystem performs its tasks under real-time constraints. Computational and communications tasks are often periodic; for example, the navigation unit updates the display unit in real time. But some system tasks are aperiodic—for example, tracking a target object and controlling a weapon.

To provide for all the system's real-time communication needs, the bus controller follows a predefined static or cyclostatic schedule to arbitrate access to the bus. The communication schedule affects each node's communication latency and overhead, and thus potentially affects real-time subsystem performance, especially when errors and retransmission are taken into consideration. In that case, even adding a simple function may change the communication schedule, which may in turn affect other components' real-time performance. Some changes may even render the current architecture unsuitable. Thus, a systematic approach to evaluation of reengineering alternatives is crucial.

In this example, we considered adding a sensor node to an existing 1553-based system. This extension added communications requirements and placed real-time constraints on the system's performance. To ensure that communications between the sensor and relevant

nodes do not exceed real-time constraints, changes in the bus controller's control software and the processors' computation software were necessary. A temporal model of the system (Figure 7) helped us design the bus controller's communication schedule. We developed the temporal hardware model from library components and captured the existing application software of different nodes and the bus controller's control software through the generic instructions, as shown in Figure 8. The new communication schedule will meet the increased communication requirements of the new node and function. In contrast to this approach, current practice relies on simple, back-of-envelope analysis, physical prototypes, or detailed emulation (requiring detailed software), extending over a period of several staff months.

After designing a satisfactory schedule, we generated the control software by linking the generic instruction representation of the control software to the schedule with the code generation tools. Because the Mil-Std-1553 model is fully functional and clock accurate, we could conduct accurate data transactions among the processor, sensor, and other nodes. This allowed finer design tuning (at a higher modeling density) and comprehensive timing and function testing of the component interface at the board level, as shown in Figure 9. We monitored clock edge accuracy in actual bus-level transactions. In contrast, current practice requires the availability of a hardware testing platform, probes and monitoring scopes, as well as actual software.

Figure 10 shows the result of a typical performance evaluation experiment with the Mil-Std-1553 virtual prototype. In this experiment, we varied the transaction rate and data buffer size of the sensor. These parameters affect the maximum rate of the sensor's operating frequency without any overwriting and data loss. Although one can calculate the sensor's operating frequency on paper, it differs in real life due to errors in the channel, data retransmission, buffer sizes, and nondeterministic behavior. Similarly, the virtual prototype allowed us to simulate clock rate drifts and discrepancies and to observe tolerable frequency mismatch. Since no physical hardware was built and assembled and no detailed software coding was necessary, the savings were significant.

Detailed hardware-software design, integration, test.

Virtual prototyping benefits detailed hardware-software design, integration, and testing by greatly decreasing errors

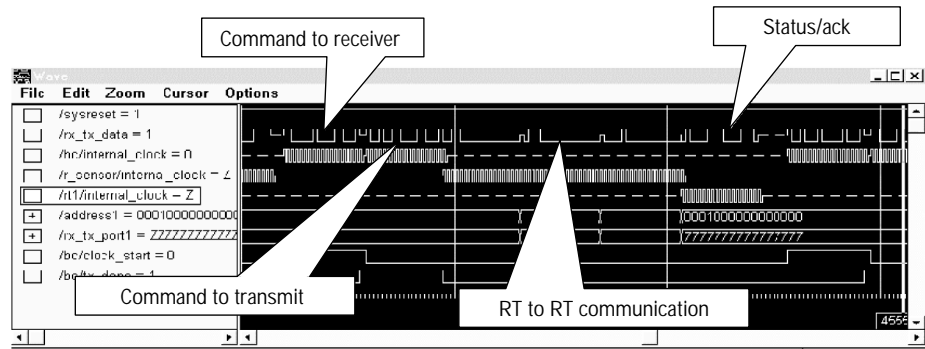


Figure 9. Using the virtual prototype of the Mil-Std-1553-based avionics system enabled rapid evaluation of the performance and functionality of the proposed reengineered architecture.

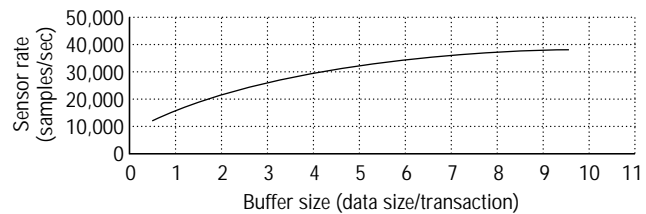


Figure 10. This typical result from the architectural evaluation shows the trade-off between buffer size and the maximum sensor rate. The result shows that a buffer size of 8 is optimal for this application.

through detailed cosimulation prior to system fabrication. We use clock-accurate and function-accurate models of the hardware components to assemble a detailed virtual prototype of the desired reengineered system. By designing, implementing, and testing actual software on this accurate hardware representation prior to the physical fabrication of the board, we save time and money.

We have discussed virtual prototyping at the detailed system integration level elsewhere,^{1,2} so we do not cover this topic in detail here. Suffice it to say that the industry has adopted virtual prototyping to a great degree, and a number of vendors and system integrators are developing supporting tools. Missing, however, are accurate libraries of components such as processors and application-specific ICs (ASICs). Without extensive, verified libraries, a virtual-prototyping environment cannot be useful; thus, we have devoted much of our recent work to populating libraries of fully functional, clock-accurate models.

Model creation and verification is a painfully long and expensive process. For simple models such as a PCI bus, the process requires about six staff months; complex models such as the Intel Pentium II require several staff years. Therefore, VP Technologies has focused on automatically

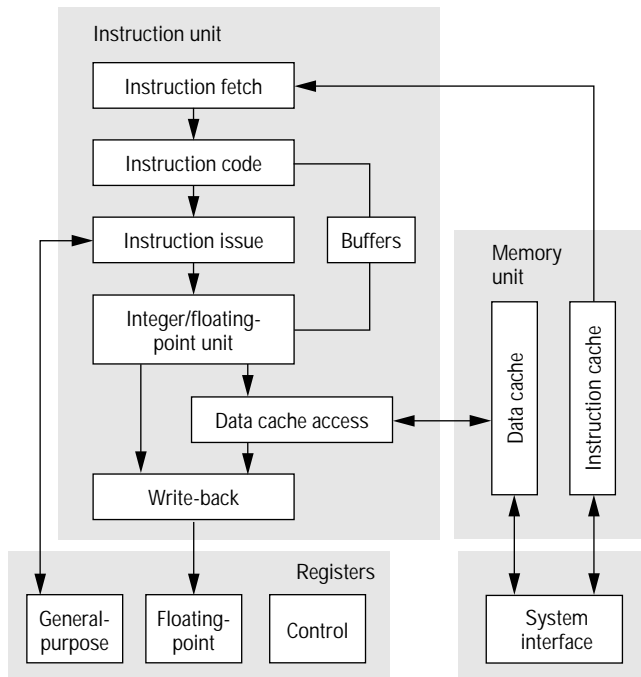


Figure 11. The VPRISC core implements the open Mil-Std-1750 instruction set architecture on a 5/6-stage pipelined architecture.

generated models of complex processors and components based on published data. We have modeled processors as complex as the PowerPC in weeks to months of staff time.

In upgrading obsolete processors and ASICs in legacy systems, we have developed an approach that largely cuts loose the binding between the instruction set architecture and the underlying processor architecture. For example, the Mil-Std-1750 instruction set architecture and architecture specification are the backbone of a number of Air Force avionics platforms. Early processors built to this standard are now technologically obsolete and need upgrading to modern technologies and architectures. However, an extensive amount of code that executes on the Mil-Std-1750 has already been written and represents hundreds of millions of investment dollars. One proposed approach is to redesign the Mil-Std-1750 processor using custom IC technology and thus improve its performance and clock speed. Another proposal is to emulate the 1750 processor on a faster processor such as the PowerPC. Both approaches are expensive.

The approach we propose is to change the processor's underlying architecture but execute the same instruction set. Thus, we retain the legacy code's executability (in function), and we also gain the capability of optimizing performance. The penalty is that we must retarget the original C or Ada compiler to include the underlying changes in the

Table 1. Estimated benchmark performance of legacy implementation of Mil-Std-1750. Parenthesized numbers indicate the clock cycle at which the instruction completed.

Comment	Instruction name	Instruction address	Program instr. order	Fetch order (cycle)	Decode order (cycle)	Issue order (cycle)	Execute order (cycle)	Write-back order (cycle)
Reset	AISP, R1, 1	1	1	1 (5)	1 (6)	1 (7)	1 (8)	1 (9)
	BR, 20h	2	2	2 (10)	2 (11)	2 (12)	2 (13)	2 (14)
First part of main body	AIM, R10, ff03	34	4	4 (22)	4 (23)	3 (24)	3 (25)	3 (26)
	AR, R4, R1	35	5	5 (27)	5 (28)	4 (29)	4 (30)	4 (31)
	ST, R10, R0, fff0	37	7	7 (39)	7 (40)	5 (41)	5 (42)	5 (51)
	AR, R4, R1	38	8	8 (52)	8 (53)	6 (54)	6 (55)	6 (56)
Call subroutine	SR, R10, R4	39	9	9 (57)	9 (58)	7 (59)	7 (60)	7 (61)
	SJS, R10, R0, 60h	41	11	11 (69)	11 (70)	8 (71)	8 (72)	8 (81)
Subroutine	AISP, R5, 15	96	12	12 (87)	12 (88)	9 (89)	9 (90)	9 (91)
	AIM, R6, 16	98	14	14 (94)	14 (95)	10 (96)	10 (97)	10 (98)
	SR, R6, R5	99	15	15 (99)	15 (100)	11 (101)	11 (102)	11 (103)
Return main body	L, R7, R0, fff0	101	17	17 (111)	17 (112)	12 (113)	13 (116)	12 (118)
	AR, R8, R7	102	18	18 (119)	18 (120)	13 (121)	14 (122)	13 (123)

processor's implementation. We can do this by changing the back end of a conventional compiler while keeping the front end common. For a typical RISC processor compiler, the amount of source code that must be rewritten is about 1,000 to 2,000 lines, and for a reusable, language-specific front end, about 10,000 lines.

Figure 11 diagrams the architecture of our VPRISC core. The VPRISC implements the Mil-Std-1750 instruction set architecture with an optimized, pipelined architecture, unlike earlier versions, such as the Fairchild 9450, which were non-pipelined. Our objective was a customizable core that can be optimized for performance or form factor while complying with the standard instruction set. The VPRISC executes Mil-Std-1750 code on processor architectures that can be optimized in performance, power consumption, and functional capability. We plan to support the VPRISC core processors with optimizing compilers and detailed VHDL simulation/synthesis models for our customers.

Tables 1 and 2 compare the performance of the upgraded, pipelined (VPRISC) and legacy (Fairchild 9450) versions of the Mil-Std-1750 architecture on a sample testbench. The reengineered system achieves a performance improvement of 100% at the same clock speed (10 MHz). Note that instruction AR, R8, R7 completes in 60 cycles in the reengineered architecture as opposed to 123 cycles in the legacy architecture.

Greater payoffs would be possible if today's systems were designed in a manner facilitating their rapid upgrading.

Migrating software

Software written for a certain system is not easily portable to upgraded systems without extensive rewriting of its application, control, diagnostic, test, and driver/interface portions. Since control, diagnostic, test, and driver software code is about 20 times the amount of application code, this is an expensive task. Newer technologies, such as middle-layer-based, common operating environments are finding increasing favor in the embedded-systems community.

Figure 12 (next page) shows our proposed unified embedded-software design environment. The mission and application composition layer, with its application programming interface (API), will propose, direct the development of, and validate the latest technologies available for composing a candidate application for the subsystem. This layer will draw from technologies such as the Unified

Table 2. Estimated benchmark performance of reengineered, pipelined VPRISC implementation of Mil-Std-1750.

Comment	Instruction name	Instruction address	Program instr. order	Fetch order (cycle)	Decode order (cycle)	Issue order (cycle)	Execute order (cycle)	Write-back order (cycle)
Reset	AISP, R1, 1	1	1	1 (5)	1 (6)	1 (7)	1 (8)	1 (9)
	BR, 20h	2	2	2 (6)	2 (7)	2 (8)	2 (9)	2 (10)
First part of main body	AIM, R10, ff03	34	6	6 (15)	5 (16)	3 (17)	3 (18)	3 (19)
	AR, R4, R1	35	7	7 (16)	6 (17)	4 (18)	4 (19)	4 (20)
	ST, R10, R0, fff0	37	9	9 (23)	8 (24)	5 (25)	5 (26)	5 (35)
	AR, R4, R1	38	10	10 (24)	9 (25)	6 (26)	6 (35)	6 (36)
Call subroutine	SR, R10, R4	39	11	11 (25)	10 (26)	7 (35)	7 (36)	7 (37)
	SJS, R10, R0, 60h	41	13	13 (35)	12 (36)	8 (37)	8 (38)	8 (47)
Subroutine	AISP, R5, 15	96	16	16 (43)	14 (44)	9 (45)	9 (47)	9 (48)
	AIM, R6, 16	98	18	18 (45)	16 (46)	10 (47)	10 (48)	10 (49)
	SR, R6, R5	99	19	19 (46)	17 (47)	11 (48)	11 (49)	11 (50)
Return main body	L, R7, R0, fff0	101	21	21 (53)	19 (54)	12 (55)	13 (57)	12 (59)
	AR, R8, R7	102	22	22 (54)	20 (55)	13 (57)	14 (59)	13 (60)
	URS, R10	103	23	23 (55)	21 (57)	14 (59)	16 (62)	14 (64)

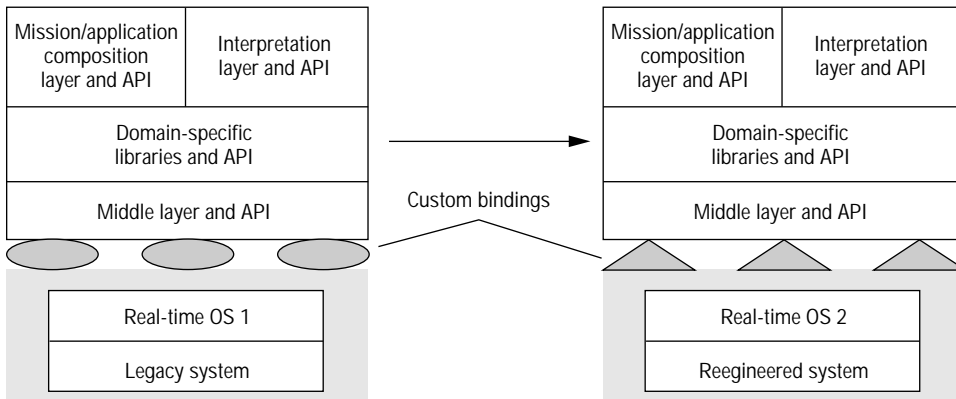


Figure 12. In VP's proposed embedded-software design environment, the existing legacy application software and middle layer migrate transparently to the upgraded hardware system and new real-time operating system.

Modeling Language (UML), heterogeneous computation models, discrete-event- and reactive-system-modeling methodologies, and new graphics and visualization developments. It will provide an easy-to-use, powerful environment with which an application developer can capture the requirements and specifications of the target application.

The interpretation layer will provide an environment for interpreting the results of the embedded-design process from concept to final product development. This layer will also integrate the results of tools and intermediate design data in a form allowing interpretation of the quality and correctness of the prototyping process at various abstraction levels. This form will be stored along with the executable specification for future upgrades.

The domain-specific library layer will provide a set of application libraries consisting of intellectual property or domain-specific knowledge to assist the application developer at all levels of design abstraction. These libraries can be at multiple levels of abstraction, ranging from timing models to detailed, gate-level, hard VHDL cores for a particular technology.

The middle layer (execution and tools) will provide a portable canvas on top of a virtual machine that will not change from one product to another. The middle layer will have APIs to several candidate architectures and provide a link to portable product development. An analogy to this layer is the Windows 95 middle layer and operating system. They hide the underlying hardware, I/O, and driver details from the PC user and provide a canvas (or desktop) on which various libraries and applications are visible as icons. Our middle layer will use the latest technology in virtual machines, multiprocessor architectures, real-time OS development, and technology-specific constraints to provide a seamless pathway to various implementation (prototype or final) platforms.

Rather than rewriting all the layers within the cycle of a

legacy upgrade program, the new approach will require changing the bindings to the new hardware and real-time operating systems. This approach is likely to require far less effort than current approaches, while ensuring that the application can quickly migrate to advanced platforms with ease. The challenges appear to lie in improving the efficiency of the implementation, which is primarily software driven, in terms of speed, real-time capabilities, and memory use.

ALTHOUGH we can reengineer existing legacy systems using the methodology proposed here, greater payoffs would be possible if today's systems were designed in a manner facilitating their rapid upgrading. This design-for-upgrade paradigm requires a shift to the development and deployment of architectures based on open, standard interfaces that allow new technology and features to be phased in with minimal impact on legacy functionality. We are designing our methods, tools, and model libraries to facilitate this transition. 

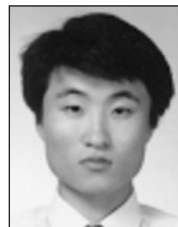
Acknowledgments

This work was supported in part by the US Air Force ACME/Parts Obsolescence Program at the Air Force Research Laboratories, Wright Patterson AFB, under contract F33615-98-C-5130.

References

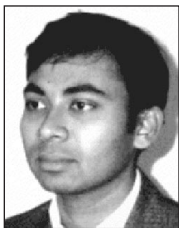
1. V. Madisetti and T. Egolf, "Virtual Prototyping of Embedded Microcontroller-Based DSP Systems," *IEEE Micro*, Vol. 15, No. 5, Oct. 1995, pp. 9-21.
2. V.K. Madisetti, "Rapid Digital System Prototyping: Current Practice, Future Challenges," *IEEE Design & Test of Computers*, Vol. 13, No. 3, Fall 1996, pp. 12-22.

Vijay K. Madisetti's biography and photo appear on page 16.



Yong-Kyu Jung is a technology specialist at VP Technologies, responsible for research and development of DSP cores. Previously, he was a senior research engineer at LG Electronics. His research interests include model generation, microprocessor and DSP architecture, VHDL modeling for simulation and synthesis, and hardware-software codesign. Jung received a BSEE from Ko-

rea University, Seoul, and an MSEE from the Georgia Institute of Technology, Atlanta.



Moinul H. Khan is a technology specialist at VP Technologies. He is responsible for research and development of architectural upgrade and reengineering design environments and control software synthesis. His interests are legacy systems reengineering, hardware-software codesign, compiler design, virtual prototyping, and telecommunications systems modeling. Khan received his BTech from the Indian Institute of Technology and his MSEE from the Georgia Institute of Technology.



Jeongwook Kim is a technology specialist at VP Technologies, where he leads efforts in legacy system understanding and upgrade environments. His research interests include software synthesis, legacy system reengineering, compiler design, and applications. Kim received the BS and MS in electronics from

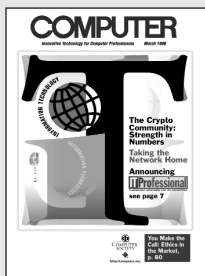
Yonsei University, Seoul.



Theodore Finnessy is a program manager for the Manufacturing and Engineering Systems branch of the US Air Force Manufacturing Technology Directorate at Wright-Patterson AFB. He currently manages the Joint Strike Force Manufacturing Capability Assessment and Toolset program, which is developing a methodology and toolkit to identify and prioritize areas of manufacturing risk in product design. He also manages the Collaborative Optimization Environment, which is developing a software architecture that will use existing engineering analysis tools to optimize a product's design. Finnessy has BS degrees in physics and electrical engineering.

Send questions and comments about this article to Vijay K. Madiseti, VP Technologies, PO Box 680190, Marietta, GA 30068-0004; vkm@vptinc.com.

What's hot in...



We've got it covered.



Subscribe to a set of 18 Computer Society magazines and journals online for only \$50.*

<http://computer.org/subscribe>

* Regular price \$99. Offer expires 15 August 1999.