Software/Hardware Codesign for Validation and Verification

G. A. Frank and B. E. Clark

Center for Digital Systems Research
Research Triangle Institute
Research Triangle Park, North Carolina 27709

Abstract

Embedded computer applications are requiring more performance, higher reliability, and lower development and maintenance costs. These requirements are being met by fault-tolerant, distributed parallel processors with tightly integrated software and hardware. Software/hardware codesign provides a mechanism for exploring the myriad of alternative system architectures during the early stages of the development of a system, and can be used during the later stages of software and hardware design to make sure that the detailed design decisions do not cause the software and hardware designs to diverge. This paper discusses software/hardware codesign, presents an approach to codesign and a tool set that supports the approach, and discusses an example of using codesign methods and tools for validation and verification.

Introduction

Software/hardware codesign is an essential approach to developing embedded computer systems requiring both custom software and hardware. Embedded computer applications are requiring more performance, higher reliability, and lower development and maintenance costs. These requirements are being met by fault-tolerant, distributed parallel processors with tightly integrated software and hardware. Software/hardware codesign provides a mechanism for exploring the myriad of alternative system architectures during the early stages of the development of a system. Software/hardware codesign methods and tools can also be used during the later stages of software and hardware design to make sure that the detailed design decisions do not cause the software and hardware designs to diverge. It is this later use of codesign methods and tools that is the subject of this paper.

Figure 1 shows the system design process as specified in MIL-STD-2167, a military standard describing software development. This process is typical in that after a system specification has been developed, software and hardware design proceed independently until system integration. This parallel design and implementation approach towards software and hardware has the potential to reduce the time to market, but pose the risk that software and hardware designs will diverge. Diverging software and hardware designs will have problems that are not detected until system integration. Solving these problems during system integration can cause major redesigns, which will increase the development costs and time to market. Thus, a critical issue is reducing this risk of increased system integration time and effort.

Fig. 1 A System Design Process
The risk of diverging software and hardware designs can be mitigated through continuing validation and verification (V&V) of the designs as they evolve. The goal of these V&V efforts is to detect problems as soon as possible in order to reduce the costs of solving them. The thesis of this paper is that codesign methods and tools provide powerful techniques for performing incremental V&V as the software and hardware designs evolve.

Software hardware codesign

Software/hardware codesign is characterized by three interdependent activities:

- Optimizing a system design by trading off software and hardware design features in order to achieve system quality goals. System quality goals may be related to throughput, reliability, and maintainability. Any of these goals may be met by a variety of combinations of software and hardware designs. The system architect doing software/hardware codesign must search this large space of potential solutions to find combinations of software and hardware that provide the greatest system quality at the least cost.

- Verifying that the software and hardware design for a system are consistent. For example, the system software may assume that the hardware memory is large enough to hold a working set of data structures and programs. If the data structures and programs grow too large to fit in the physical hardware and no virtual memory is available, then the software and hardware designs are not consistent.

- Validating that the integrated software and hardware designs meet system requirements. For example, a system requirement may state a deadline for a response to a stimulus. In an avionics system for an aircraft that is in terrain following mode, the avionics system must create signals to the actuators in order to lift the aircraft over an obstacle before the aircraft reaches the obstacle. This system requirement can be met by a combination of parallel processing hardware and software. During codesign, designs of the software and the hardware must be frequently tested in an integrated fashion against the overall system requirements.

The need for software/hardware codesign is increasing as the system requirements become both more severe and more complex. For example, most embedded system designs in the past were driven purely by throughput and response time goals. Now those goals may also include system reliability, maintainability, and security. Furthermore, the potential system designs are becoming more complex. Instead of optimizing the software for a fast uniprocessor, the system architect must deal with developing and partitioning software for distributed, parallel, fault-tolerant computer hardware. Distributed systems require complex interaction between software and hardware to allocate and schedule tasks in a multiprocessor environment, and to route messages between tasks. Fault-tolerant systems require interaction between the hardware fault detection, masking, and containment mechanisms and the software reconfiguration and recovery procedures. Secure systems require interaction between hardware encryption/decryption mechanisms and software implemented security protocols.

Perhaps the best recent example of effective software/hardware codesign has been the development of Reduced Instruction Set Computers (RISC). These computer architectures arose from tradeoffs of capabilities in VLSI hardware design and compiler software design.

Another example of software/hardware codesign is the development of fault-tolerant real-time computers, such as avionics computers. These systems use fault-tolerant multiprocessor hardware and sophisticated operating system software. These systems are developed by trading off the cost of using multiple processors to achieve both high performance and high reliability against the complexity of sophisticated fault detection, isolation, reconfiguration, and recovery techniques.

A codesign approach

Since 1980, the Research Triangle Institute (RTI) has been developing a methodology for software/hardware codesign [1]. This methodology models the system software with hierarchical data flow graphs, based on the extensive work on the use of data flow graphs for structured systems analysis and structured system design pioneered by Tom DeMarco [2]. In order to support software/hardware codesign, hardware is also modeled as a hierarchy of graphs that describe the hardware components and their interconnections. Each node in the hardware graph is a hardware component, such as a processor, bus controller, memory, or I/O device. Each arc in the hardware graph is an interconnect. Each task in the software model is then mapped to a unique component of the hardware model. This mapping is a form of graph embedding, in the sense that each arc in the soft-
ware graph must map to a path in the hardware graph. This mapping constraint guarantees that any communication required by the software is realizable in the hardware. The methodology also states that all tasks mapped to a specific resource have to contend for that resource. This is an important aspect of simulation, and is critical for understanding the performance impact of particular mappings.

The mapped model is evaluated through simulation and analysis. Simulation can provide performance information, such as response times for critical stimuli, system throughput, and hardware module utilization. Analysis can provide information on hardware cost, system reliability, and software complexity. The results of simulations and analyses interact. For example, analysis of system reliability is dependent upon such parameters as software fault detection, isolation, reconfiguration, and recovery time.

Fig. 2 Mapping Software to Hardware

Work at RTI over the last ten years on a methodology for software/hardware codesign has led to the development of the Architecture Design and Assessment System (ADAS) [1, 3]. ADAS is a computer-aided design tool for high-level functional and performance analysis of real-time systems. In keeping with the methodology, ADAS models the system software with hierarchical data flow graphs, and models hardware with hierarchical block diagrams showing hardware components and their interconnections. ADAS extends this model by providing attributes for graphs, nodes, and arcs which describe the performance characteristics of the system. These attributes support the capability to simulate the performance of the system using a form of Petri Nets called marked graphs, a modified form of the marked graph models used by Karp and Miller [4]. The marked graph simulation has built-in models of resource contention based on the mapping of software tasks to hardware resources.

Using a codesign method and tool for validation and verification

RTI developed an ADAS simulation model of the operating system of a distributed avionics system [5]. Figure 3 shows the top level data and control flow for the system. The simulation was developed to describe the startup, shutdown, and reconfiguration times for the avionics system. The goal was to first build a model that matched the test bed hardware and modeled the software that was being developed. However, the model was developed in parallel with the final development of the software and hardware and during integration of the avionics software and hardware. As a result, the modeling effort played a V&V role.

Fig. 3 Top Level Data and Control Flow for an Avionics System

Since the model was developed in parallel with the software and hardware, extensive changes were required in the model to keep it current with changes in the software and hardware design. Over 70% of the model was changed, and some parts were changed several times. If the model is to serve a role in V&V, then the effort to keep the model consistent with the design must be budgeted and managed. A major part of this effort is providing communication between the software and
hardware designers and the modelers. A critical part of this effort was review of the model by the software architects. In turn, the modeling effort raised key questions and issues for the software architects. The software development environment shielded the software developers from issues relating to multiprocessor communication and control. These issues were the primary focus of the modeling team. The primary emphasis of the software developers was maximizing the operating system performance during normal operation mode. The modeling effort emphasized startup, shutdown, and re-configuration modes, which also had to meet critical system performance requirements, particularly the configuration module.

During the development of the model, several anomalies in the proposed system architecture were discovered. These anomalies related to interactions between the software and hardware design. For example, the initial software design provided for direct loading of the operating system kernel into the hardware modules. However, as the operating system design and development proceeded, the kernel size grew and eventually exceeded the Startup ROM address space limitations. This problem was revealed in the modeling effort and fixed, after a failed attempt to load, by creating a bootstrap loader for the kernel. Another anomaly was the lack of buffer space in the BIU. This has resulted in slowing the speed of bus transfers to the speed of the secondary memory. The model indicated that this slowdown of the bus would severely compromise system performance. The hardware developer added buffers. Additional performance problems included assignment of performance parameters such as the token holding time for the block transfer bus, which was set too small for the size of data blocks being transferred. Finally, the modeling effort pointed out that new system states and hardware module modes were needed in the design. These states are transitory in the test bed hardware, but became highly significant in a system of the size required for an actual application. By modeling large systems that cannot be assessed in test beds because of cost and schedule, a codesign model serves as a check and balance to prevent parochial design.

Conclusions

Software/hardware codesign is a critical technology for the development of high-quality embedded computer systems. Formal models and computer-based tools are essential for making codesign practical. Tools exist that support software/hardware codesign, but they need to be further integrated into the system engineering process. Software/hardware codesign tools can be used to support the validation and verification of system designs, particularly when separate software and hardware design efforts are proceeding in parallel.

Acknowledgements

The modeling of the avionics system was done with the support of the TRW AARTS team. The AARTS modeling project was realized through the leadership and guidance of Joe Wilgus at the Air Force Wright Laboratory. F. Gail Gray at Virginia Polytechnic Institute, Tennis White at IBM, and Jim Morrison at RTI developed much of the technology and performed much of the labor required to implement the AARTS model.

References


