A Virtual Prototype VHDL Development Methodology

Michael Vahey, Decker Bushman, Steve Dabell, Ted Ennis, Paul Kalutkiewicz, Wing Lee, Mike McCollough, Brian Nelson, Kathryn Russell, and Howard Wanke

Abstract

Virtual prototyping is a VHDL technique for validating a hardware and software design before hardware is developed. A virtual prototype was used with great success in the Rapid Prototyping of Application Specific Signal Processors (RASSP) program for the development of an Infrared Search and Track (IRST) processing system. This paper describes the RASSP virtual prototype and test results achieved by the demonstration team.

1. Introduction

The ARPA Rapid Prototyping of Application Specific Signal Processors (RASSP) program is developing design processes with the intent of reducing design cycle time by a factor of four. This paper presents some of the demonstration team’s experiences and conclusions after testing the RASSP design process for the development of an Infrared Search and Track (IRST) system.

1.1 Role of the Virtual Prototype

The virtual prototype is a key element of the RASSP design process, allowing us to evaluate the end product before prototype hardware is built. The goals of virtual prototyping are (1) first-time laboratory integration without design errors, (2)

![Diagram](image)

**Figure 1.** Role of virtual prototype. The virtual prototype provides the means for connecting all hardware and software tradeoffs throughout the design cycle.
rapid design changes for model year upgradeability through design reuse and early checkout, and (3) assessment of hardware and software interaction early in the design cycle, thus avoiding costly integration changes in the laboratory. Also, such complete electronic descriptions of the hardware and software help support the military's need for a 20-year life cycle. Thus, before the system is built, the virtual prototype evaluates (1) whether or not the customer's product specifications will meet his needs and (2) if engineering's design will operate successfully.

1.2 Project Description

We created a complete virtual prototype of processing hardware and Ada software before the design was fabricated. The rapid laboratory checkout is a good success indicator that virtual prototyping has value.

The RASSP IRST project was performed by a virtually co-located team. Team members were from Hughes in El Segundo, California; Motorola in Scottsdale, Arizona; and Lockheed Sanders in Nashua, New Hampshire. Using the Internet for file sharing, e-mail, and video teleconferencing, the entire hardware and software design was performed without requiring travel for design reviews or coordination. VHDL descriptions done at each company were integrated to form the virtual prototype. The virtual prototype facilitated distributed design checkout since each designer could check his portion of the design from his office.

2. IRST Processing System

The virtual prototype developed for RASSP models a complex multiprocessor system composed of commercial off-the-shelf (COTS) processing modules, custom interface modules, and Ada code. This section describes the IRST processing system elements in order to aid in understanding the scope of the virtual prototype. Infrared Search and Track processing detects unresolved (sub-pixel) moving objects in an infrared image. Performance is limited by scene clutter (clouds and terrestrial background). Algorithms are applied to register multiple frames of data, filter out clutter, boost target signatures, and detect and track targets. The detected targets are displayed with graphic symbology overlays on top of the original scene data. An RS-422 interface provides target position and velocity.

![Figure 2. IRST processor. A processor design scales to 192 processing elements.](image-url)
Table 1. Design complexity. The virtual prototype simulates a fairly complex design, as shown by the number of entities and lines of code.

<table>
<thead>
<tr>
<th>Module</th>
<th>Number of Entities</th>
<th>Number of Lines of VHDL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data input/distribution</td>
<td>56</td>
<td>4,624</td>
</tr>
<tr>
<td>RS-170 in</td>
<td>7</td>
<td>453</td>
</tr>
<tr>
<td>Digital sensor in</td>
<td>23</td>
<td>749</td>
</tr>
<tr>
<td>Video output</td>
<td>23</td>
<td>1,935</td>
</tr>
<tr>
<td>MCV9 with ISA</td>
<td>189</td>
<td>20,153</td>
</tr>
</tbody>
</table>

reports to the aircraft mission computer.

The Infrared Search and Track processing system is shown in Figure 2. The system design can input either a standard RS-170 video input stream or a custom 135 Mbyte/second digital data sensor stream. The RS-170 video output displays sensor imagery with graphic symbology overlays. Four custom modules were designed for interfaces. COTS modules were used for the signal processing and host controller.

The IRST processing system virtual prototype totals over 39,000 lines of VHDL. An additional 18,000 lines of software implement the algorithms and control software. Table 1 lists the VHDL magnitude by module type.

2.1 Data Input/Distribution Module

The Data Input/Distribution module is a VME card with 35 ICs, including 4 FPAGAs. We used two daughter cards to adapt to sensor-specific electrical and timing interfaces. A VME A24D32 interface is used for control and status. A Mercury RACEWAY interface permits high-speed (110 Mbyte/second) video transfer to the processor modules. An RS-170 video output displays one of the two sensor inputs with symbology. A video crossbar connects the two daughter cards, the RS-170 output, and the RACEWAY interfaces. Subwindow logic routines pass only those columns and rows selected by software. Routing logic under software control will pass the image subwindows to selected processing elements in the multiprocessor system.

2.2 RS-170 Video Input Daughter Card

The RS-170 input daughter card plugs into the Data Input/Distribution module and provides an EIA RS-170 interface. The module provides programmable gain and offset control before analog-to-digital conversion. The module can generate test pattern data for module- and system-level built-in test. Software reads and writes control registers via the VME interface. The module has 11 analog ICs, 3 digital ICs, and 1 gate VHDL synthesized FPGA.

2.3 High-Speed Digital Input Daughter Card

The digital sensor interface also plugs into the Data Input/Distribution module. It accepts data at 135 Mbyte/second, and provides an additional level of software-controlled subwindowing logic and FIFO buffering. It also generates test pattern data for module- and system-level built-in test. Software reads and writes control registers via the VME interface. The module has one gate VHDL synthesized FPGA and 20 other ICs.
2.4 RS-170 Video Output Card

The video output module provides an EIA RS-170 video output with symbology overlay. Most of the logic on the module duplicates logic on the Data Input/Distribution module, but with a different RACEWAY interface. This module accepts high-speed video data from the RACEWAY, and provides frame memory and symbology generation. The module has two VIIDL synthesized FPGAs, one analog IC, and 23 digital ICs. Software reads and writes control registers via the VME interface. Symbology memory is write-only via the VME interface.

2.5 IRST Software

The IRST processing system software has 18,000 lines of code. Only 50% of this code is for the core signal processing algorithms. The rest provides sensor interfaces, display generation, built-in test, and multiprocessor control. Only 677 lines of the code were run on the virtual prototype. This code is used for built-in test and interface control. As such, it “touches the hardware in a noticeable way.” Much of the other code was tested on workstations. The code that depends on the operating system, the Ada runtime system, and the actual hardware interface could not be checked on the virtual prototype because of virtual prototype limitations. This is a limitation that needs to be fixed in the future.

3. Virtual Prototyping—The Error Sieve

The RASSP virtual prototype was developed using a top-down VHDL design methodology with progressive addition of more hardware details. The process supports hardware/software co-design, with the initial phase of the virtual prototype serving merely as a performance model of the end system that shows buses, major computing elements, and I/O. We modeled software and sensor workload as tokens to evaluate processing element and bus loading. In subsequent design refinement, we developed behavioral models of processing elements, interface circuits, and buses. These, in turn, were refined to register transfer level descriptions that supported design synthesis of programmable logic. Instruction set level modeling of the processing elements allowed execution of control

![Figure 3. ISA block.](image-url)
and built-in test Ada software within the VHDL model. By completing each level before beginning the next lower level of detail, we caught design errors early in the design cycle.

The remainder of this paper will focus on the final phase of the virtual prototype simulation, which combines the detailed hardware design models with the processor instruction set architecture model running Ada software. This phase was the most thorough level of VHDL simulation. It proved that the design was correct before the hardware was fabricated. (Note that gate level simulation was done to validate timing. These simulations found timing errors and synthesis tool bugs that required VHDL revisions.)

3.1 Instruction Set Architecture Model

An Instruction Set Architecture (ISA) model of the Intel i860 microprocessor was developed by the Georgia Institute of Technology. The ISA model implements all registers, instructions, and status logic visible to a programmer. It allows i860 object code to run as it would on real hardware. Figure 3 shows the model components. The ISA core consists of the instruction decode block, pipeline processing, registers, and procedural calls. All instructions match the functionality of the i860 hardware description manual and programmer’s reference manual.

The test bench for the model consists of memory, memory controller, and clock and reset generators. The memory model uses a dynamic allocation scheme. It creates memory files as 4096 x 4-byte elements and stores them as integers. When an address outside the range of the current segment is presented to the model, a new segment is allocated and the data is loaded from the appropriate memory file. The location placed on the address pins is used to form the file name required for loading.

A Bus Interface Model (BIM) emulates the

![Figure 4](image-url) Figure 4. Mercury MCV9. A VHDL model of the MCV9 processor module contains sufficient detail to run Ada software and exercise backplane buses.

3.21
Figure 5. System level virtual prototype. The virtual prototype connects all modules and runs the Ada software.

hardware functions of the i860, including memory and I/O access, burst modes, interrupt capabilities, reset, etc.

The i860 Bus Interface Module was inserted into an MCV9 model representation of the Mercury VME 9U RACE product. The MCV9 consists of 16 processing nodes, 7 crossbars for inter-processor communications, a VME interface, and a RACEWAY port. Each processing element consists of an i860 processor, a configurable bank of DRAM, and a Compute Element ASIC.

Mercury created the original design using Viewlogic schematic capture tools. A VHDL structural representation of the MCV9 was created by translating Viewlogic schematics via the use of Viewlogic Export 1076. The ASICs were represented as a combination of behavioral and gate level models, including LSI 100K models for particular boundary cells and macros. To enhance simulation performance, we stripped out the LSI primitives and replaced them with behavioral models. All components of the final subsystem contained behavioral level descriptions. The block diagram of the MCV9 (Figure 4) shows how the i860 is integrated into the processor element.

3.2 System Level Virtual Prototype

The system level virtual prototype combines all system hardware and software elements (Figure 5). The simulation checks for consistency of protocols across interfaces: board to board and hardware to software. The simulator also aids development of low-level diagnostic tests debugged on the virtual prototype and then executed on the real hardware. All tests were written in Ada and run on the MCV9 module.
3.3 Test Results and Problems

The following paragraphs describe the system level tests and any problems we encountered during testing.

1. Software reset. This test ensures that all boards can be reset via software. We applied reset to the boards by setting and clearing a bit in a control register via the VME bus. Once reset was complete, the contents of all registers were read and compared to expected values by software to make sure that they were configured to the default value.

The only problems we found were programming the correct “route word” through the crossbars to the VME interface. Normally the operating system does this, but we had to do it manually because the operating system does not run on this model. Once this problem was corrected, the tests executed with no errors.

2. VME register test. This software writes and reads all registers on the data distribution and video output cards via the VME bus. If all tests pass, software writes a known pattern (A5A5A5A5) to the same location in local memory. The test repeats continuously. If the tests fail, software spins executing NOP instructions. These basic loops make it fast and easy to determine if the test passed or failed. We discovered errors in the VME handshake for A32 D16 mode. The problem was that the software was putting the i860 into big endian mode.

3. RAM test. This test writes and reads portions of the RAM on the video output card. A block of 128 locations is written and then read back. Again, simple loops are used to determine if the tests passed or failed. This test failed, which uncovered a significant design flaw with the VME logic on both the data distribution card and the video output card. The design engineer misinterpreted the VME specification and did not use A1 and A2 as part of the address decode. Therefore, the MCV9 could access only the 32-bit locations and not the 16-bit locations. To fix this error, we changed the hardware design to relocate the address of all VME control registers. Finding this problem before the hardware was fabricated not only saved debug and rework time, but also allowed us to change the software. Had it been found on the real hardware, the software would have been changed to “fix the hardware error.”

4. Floating point RAM test. This test is the same as the RAM test mentioned above, but it tests a different section of RAM.

No errors were found with this test.

5. Interrupt tests. Several tests force hardware and software interrupt conditions to make sure that the hardware and software respond correctly. The tests are FIFO overflow interrupt, data overflow interrupt, beginning of frame interrupt, and end of frame interrupt.

These tests uncovered both modeling and design errors. The modeling error was that the MCV9 signals relating to the interrupt logic were disconnected. The design error was that the data distribution card did not handle the interrupt acknowledge cycle correctly.

These errors were fixed. Subsequent testing of the interrupts revealed software errors, including incorrect parameters set for the video output and data distribution cards.

3.4 Timing Results

The simulation time for a large complex processing system such as the IRST system can be very long. During November, we initiated 169 simulations, of which 135 were completed. They simulated 897 milliseconds and used 402 hours of wall clock time running on a Sparc 10. By running Ada code on the virtual prototype, we discovered three hardware errors and eight software errors. However, because of the long simulation times, we carefully selected the portion of the code to be run. Anything that used the operating system or Ada run time system was bypassed. As such, the laboratory experience and the virtual prototype were somewhat different.

Checking out the design on the virtual prototype helped us discover numerous design errors;
however, the rate of discovering and fixing errors was slow. Figure 6 shows the steps and run times required to find and fix errors on the virtual prototype. Slow virtual prototype checkout only allows checkout of items that occur in the first few hundred milliseconds of system operation. On the other hand, checkout on the virtual prototype did not damage any parts via probing as did the real laboratory checkout.

4. Conclusions

The effectiveness of the virtual prototype can only be determined after the real hardware and software are integrated in the laboratory. Having completed hardware checkout and software integration, we can now draw the following conclusions.

4.1 Benefits

All hardware designs checked in the virtual prototype worked in the laboratory the first time.

Some analog portions could not be checked, and these required minor tuning.

The virtual prototype provided a forum for hardware and software engineers to discuss details sooner and change early system concepts when performance simulation or early VHDL modeling showed timeline problems.

Many software errors were fixed before the laboratory checkout.

Running the actual software on the ISA model identified hardware design problems not discovered in standard VHDL simulation.

4.2 Limitations of Virtual Prototype

Simulation run times were so slow that we could explore only those activities near the beginning of the hardware initialization cycle (the first 150 milliseconds).

We could test only a limited portion of the software because of the slow simulation times and because the operating system could not be run on
the virtual prototype; only software that did not use operating system calls or the run time system could be executed.

4.3 Extensions and Changes to the Virtual Prototype Process

There is a clear need for better simulation run times. Executing the operating system and run time software is essential to fully check out the software and refine the hardware/software design.

4.4 Other Observations

The virtual prototype changed the development schedule such that although the design stage lasted longer than in conventional development, the hardware checkout and system integration went much faster.

We expect that reuse of the virtual prototype elements will allow more rapid design and upgrade of future processing systems. We hope this will be proven in subsequent RASSP model year designs.

6. References


5. Acknowledgments

This work was performed as part of the Lockheed RASSP team. The Lockheed RASSP team is under contract to the Naval Research Laboratory, 4555 Overlook Ave., SW, Washington, DC 20375-5326. The Sponsoring Agency is Advanced Research Projects Agency, Electronic System Technology Office, 3701 North Fairfax Drive, Arlington, Virginia 22203-1714. The Lockheed RASSP team consists of Lockheed Sanders, Inc.; Motorola; Hughes; and ISX.