VHDL-based System Simulation and Performance Measurement

Patrick A. McCabe
Honeywell Inc.
Space Systems
13350 US. Hwy. 19 North
Clearwater, FL 34624
(813) 539-4047
pmccabe@space.honeywell.com

Abstract

This paper describes the techniques employed in the development of a VHDL-based system simulation of the Honeywell RH32 Prototype Development Unit (PDU). The architecture of the PDU is briefly described, and a design process is presented. The system simulation approach is discussed, including simulation model types, their development and validation, and testbench structure. Additionally, techniques for performance measurement and analysis of the simulation results are discussed, which can be used to improve the system design at the chip and board levels. Finally, conclusions drawn from this effort are presented, along with indications of future work to be performed.

1. Introduction

The Honeywell RH32 processor is a 3-chip type radiation-hardened 32-bit RISC processor with floating-point coprocessor and up to 8 Kbytes each of instruction and data caches, and is capable of operating at a 25-MHz clock rate [1]. The RH32 was developed under contract from the Rome Laboratory, Rome, NY. RH32 is intended for spaceborne processing applications, including mission data processing and spacecraft control processing. The RH32 CPU is a reduced instruction set integer processor with a 5-stage pipeline, and is capable of executing most instructions in one clock cycle, therefore, the RH32 has a maximum performance of 25 MIPS at a 25-MHz clock rate. The floating-point processor (FPP) is an IEEE-754 compliant coprocessor, and handles all floating point operations. The 8-Kbyte cache memory consists of a cache controller and on-chip data and tag rams, and operates as a write-through cache. The same cache chip type is used to implement the separate instruction and data caches. Up to four caches of each type may be used in an expanded cache system.

The Honeywell RH32 Prototype Development Unit (PDU) [2] is a software/hardware development environment that provides:

- rapid prototyping and concept development with commercial cards
- immediate software development on a functionally equivalent target
- integrated hardware debug and software development tools

The PDU backplane can be configured to support either VME bus, Advanced Spacecraft Computer Module (ASCM) module bus, or the RH32 memory bus (MBUS). The core of the PDU is an RH32-based CPU card, which consists of an RH32 processor (CPU, FPP, I-cache and D-cache), 4-Mbytes of local memory, 512-Kbytes of start-up ROM, 96-Kbyte NVRAM trace memory, and peripheral chips which provide timers, interrupts, dual RS-232 interfaces, and VME or module bus interfaces. An IEEE 1149.1 test and maintenance bus is also provided on the card. A block diagram of the RH32 PDU CPU card is shown in Figure 1. The PDU supports mature C and Ada software development tools (VxWorks, VADS Works, Intermetrics).

The RH32 chipset and the peripheral ASICs on the processor card were developed using VHDL. A system simulation of the PDU design was created based on the board schematics. The ASIC VHDL models and commercially available hardware and software models were used to emulate the functions of the components on the board. Interconnections between models in the testbench were created to duplicate the routing of signals in the board schematic.

A number of testbenches evolved over time with varying levels of completeness, capability and model fidelity based on the needs at the time. The final testbench for the system simulation modeled the PDU with sufficient accuracy to allow execution of actual RH32 software on the testbench so that interactions of the processor with the memory and peripheral devices
could be observed, and also allowed hardware and software modifications to be made before the PDU was powered-up for the first time in the lab. This proved to be a highly effective approach, and the number of problems found in simulation clearly demonstrated the value of creating and using the system simulation approach as an integrated part of the design process.

![RH32 PDU CPU Card (P-32)](image)

**Figure 1: RH32 PDU CPU Card (P-32)**

2. Design process

Most of Honeywell Space Systems' ASIC designs are intended for space-borne applications and are implemented using Honeywell's RICMOS radiation-hardened CMOS technology. Several new ASICs were required on the RH32 Prototype Development Unit, and environmental factors allowed the use of commercial quality field-programmable gate arrays (FPGAs). The design process used for the RICMOS ASICs had to be modified to support the development of the Actel FPGAs [3] that were used on the PDU. The design process used for implementing and verifying an FPGA design is shown below in Figure 2.

Briefly, our location uses HP/Apollo workstations, with either Vantage or Synopsys used for VHDL simulation, and Synopsys used for logic synthesis. Mentor software is used for schematic capture and gate-level simulation. For the Actel FPGAs, the Actel toolset was used for timing analysis, place and route.

Logic synthesis for FPGAs is highly challenging, due to the relatively large gate delays inherent in FPGA technology. In order to maximize performance of an FPGA, the VIIDL for the design often tries to model the desired structural primitives of the FPGA library so that Synopsys will create the most efficient implementation of the logic [4].

The RH32 PDU system simulations used the Vantage Spreadsheet for VHDL simulation [5]. In addition to simulation capabilities, Vantage also includes a set of support tools that allow the generation of gate-level simulation stimulus files directly from the VHDL simulation run. A complementary tool also exists which performs the opposite conversion - converting gate-level simulation results into Vantage's results database (RDB) format. A third tool exists which can compare the two RDB's, and generates an output file showing the differences. By sampling the two RDB's at a time late in the clock cycle (to filter out propagation delays), the data obtained from the two simulations should be identical. This technique is heavily used in ASIC design at Honeywell.
3. Design verification

The purpose of simulation is design verification, which is accomplished by exercising the design using a number of stressing test cases. Before beginning a simulation activity, it is important to identify the goals of the simulation – what will be tested, and how will it be tested? A verification plan can be created to assist in this process. The verification plan identifies what tests are required to test the functions implemented in the design, and is cross-referenced with the requirements specification in order to ensure that all requirements are met. By definition, when all verification plan tests have been completed, the design has been adequately tested to assure correct operation in the intended application. The verification tests encompass the following major areas of interest:

Interfaces: These may be chip-to-chip, board-to-board, or subsystem-level interfaces. It is important to develop a solid interface specification early in the design cycle, since multiple designers will be dependent on the same interface. Early development of the interface logic will allow early simulation of that interface, which in turn will provide high confidence in the interface design. An inadequate interface definition, if discovered late in the design cycle, can have drastic consequences. When faced with an interface for which there is no existing model to run simulations against, a behavioral interface model can be created to verify the interface, as described in a later section.

Functionality: Verifying the functionality of a design includes tests of components of a design as well as tests for the design as a whole. It is important to test each component as it is created, and not to assume its correct operation. As components are brought together to form a new level of hierarchy, interdependencies between components may also cause unexpected behavior, and so simulations should also be performed at this level. Finally, the entire design is brought together, and testing of the design as a whole may begin, with the knowledge that most of the errors should have been found by testing of the lower levels.

A design can be created without first testing the components in the lower levels of hierarchy, but when the simulation does not work the first time, the designer is faced with a massive number of errors to debug, with no clear indication of where to start. Both methods have been tried at Honeywell, and we have found that testing of the lower levels, although more time consuming initially, saves time when the major blocks of the design are integrated and tested as a whole.

Software and software development tools: The simulation can not only debug the hardware design - it
can also be used to debug the software design, as well as the software development tools. In the development of a new processor design, the ability to run simulations of real software executing on the hardware is invaluable. Such an exercise will uncover not only hardware problems, but software problems, and can be used to validate the software development tools (assembler/compiler/linker). In our case, a C compiler was developed by another company, and by simulating code produced by the compiler on the system simulation, we were able to uncover problems in the code generator and linker, and also uncovered hardware problems for which we found software work-arounds. This proved to be very beneficial, as these problems were found after the processor chips had gone to fab, and before we received them from the foundry. The compiler vendor was able to modify the compiler to include the software work-arounds, and our system simulation allowed us to verify the changes.

Execution of software on the detailed system simulation also allowed the performance of the system to be measured. Performance profiles of benchmark algorithms were produced from simulation outputs, and were used to evaluate the processor performance. This technique is described in a later section.

4. Simulation models

A system simulation consists of a number of different types of models that emulate the behavior of the system being simulated. It is important to provide the necessary modal fidelity in order to produce an accurate, meaningful simulation. The level of fidelity of a model may change over time to accommodate changing requirements for accuracy in the simulation.

A high-fidelity model provides accurate timing information, implements the necessary interface protocols, and some of the internal behavior of the part being emulated, and checks for violations. However, high-fidelity models are difficult to debug and slow down the simulation. If misapplied, a high-fidelity model may provide more information than can be effectively used in the simulation, and may not be worth the performance penalty. In such a case, little or no value is added by using a high-fidelity model, and a lower-fidelity model should probably be used instead.

On the other hand, a low-fidelity simulation may not be accurate enough, and may cause some design problems to be missed or masked, and may actually create false “problems”. Low-fidelity models are more suitable for simple interfaces.

The system simulations used a combination of several different types of models:

- VHDL behavioral models
- VHDL register transfer level (RTL) models
- hardware and software models from Logic Modeling Corporation (LMC)

These models and how they were used in the system simulations will now be described.

4.1 VHDL behavioral models

VHDL can be used to implement a wide variety of behavioral models, from simplistic models which only emulate gross bus timing on a cycle-by-cycle basis, to high-fidelity detailed models, which include component timing, correct data, and error checking and reporting. The behavioral models interface to the design under test to exercise features of the design.

4.1.1 Bus functional models: The RH32 processor consists of an integer CPU, an IEEE-754 compliant floating-point processor, and separate 8K-byte instruction and data caches. The RH32 caches interface to the rest of the system (memory and I/O) by way of a memory bus (MBUS), consisting of 64-bits (two 32-bit words) of data plus 14-bits of error detect detection and correction (EDAC), a 29-bit double-word oriented address bus, plus miscellaneous control signals. Both instruction and data caches are supported, and multiple cache chips of each type may be used in an expanded cache system. Basic MBUS operations originated by the caches consist of 32- or 64-bit memory operations with EDAC, and 32- or 64-bit I/O operations with parity. The RH32 was synthesized from several dozen VHDL RTL models.

A VHDL-based bus functional model (BFM) was created which emulated the MBUS operations of the RH32 processor in order to avoid using the detailed VHDL design of the RH32 itself. Use of a bus functional model allowed faster simulations of the RH32/system interface and avoided the use of the high level of fidelity that the VHDL RTL code of the RH32 would provide, while still providing the necessary fidelity. The BFM implements all types of operational modes supported by the MBUS, and also supports configuration options that affect subsequent behavior of the model.

The BFM reads an ASCII text file from disk via VHDL TEXTIOT functions, and interprets the file line-by-line to determine what operation to perform next. The text file, also referred to as a macro file, contains pseudo-assembly language instructions which the user creates to direct the flow of initialization and testing performed by the system simulation. The simulation can be automatically halted when the end-of-file is reached in
4.1.2 Other behavioral models: Other types of VHDL behavioral models created for the system simulation included RAM models and I/O models. These models were written using high-level behavioral VHDL and implemented the basic bus timing and protocol of the device being modeled. The RAM models also provided an array of storage locations to provide a small RAM-like memory space, with modulo-N addressing. I/O interface models provided internal storage for the simulation of the internal registers of the devices being modeled. Error checking was also performed by these models in the sense of timing checks (setup and hold times), using assertion statements to report the error.

4.2 VHDL. RTI models

New ASICs designed at Honeywell Space Systems are created using a register transfer level (RTL) style of VHDL coding. After the VHDL is complete, the design can be incorporated into a system simulation, which is a model of the intended application of the ASIC. The VHDL code of the ASIC essentially becomes a unit under test (UUT), and the devices that interface to it are modeled with the level of fidelity required to apply the appropriate stimulus/response to the design by feeding test vectors into its external interfaces. By carefully designing the ASIC and its test cases, nearly 100% of the design can be tested by functional vectors. Honeywell also uses serial scan techniques for non-functional tests, which achieves nearly 100% coverage of stuck-at faults.

The same VHDL code that is used in simulation is passed through the Synopsys Design Compiler for logic synthesis. Writing VHDL for synthesis can be quite different than writing VHDL for simulation-only. Although some behavioral VHDL code will synthesize well, it is often better to start with a good paper design showing the data path and control logic to be implemented by the design, and generate VHDL from the paper design. Storage elements such as latches can be inadvertently created from behavioral code, and may produce undesirable logic structures.

It is useful to develop a suite of standard “components” that are commonly used in an ASIC design, such as registers (loadable and transparent, reset/preset, etc.), multiplexers of various sizes and widths, counters (carry look-ahead, ripple-carry, etc.), parity generators, pulse detectors/generators, and other types of components. VHDL implementations can be created of these components, and they can be simulated/tested, documented and synthesized separately until the desired effects (behavior, size, timing) are achieved. These components can then be used as building blocks for the ASIC design, and should be incorporated into the design block diagrams wherever possible. It is then a very straightforward process to develop synthesizable VHDL from this highly structured paper design.

4.3 LMC SmartModels

We have found SmartModels from Logic Modeling Corporation to be indispensable in our system simulation effort [6]. SmartModels are compiled C-language behavioral models that interface to our simulators. SmartModels can interface to both the Vantage VHDL simulator and the Mentor Quicksim simulator [7]. Hundreds of SmartModels exist of commercial components, including microprocessors capable of executing object code, RAM, ROM, buffers, latches, UARTS, etc.

The SmartModel interface to Vantage is very straightforward. The models use a 48-state logic system, and a shell must be created to convert to IEEE 9-state or other logic system of choice. Memory models can be initialized to emulate programmed PROMS to allow simulated microprocessor execution of object code, which is a powerful feature. The memory models are initialized when the simulation is started by reading text files which contain the memory data.

4.4 LMC hardware models

LMC also provides the capability to integrate hardware into a Vantage or Quicksim simulation. Using the LM1000 or LM1200 hardware modelers, actual ASICs can be interfaced to the simulation to provide highly accurate behavior and timing [8].

ASICs are mounted onto an adapter board, which in turn is placed into the hardware modeler chassis. Special hardware and software exist to create the interface to the simulation from this chassis. As the simulation progresses, simulated stimulus to the hardware model is saved, and these vectors are presented to the actual ASIC in the hardware modeler. The response of the ASIC is captured by the hardware modeler, and these results are in turn passed to the simulation. In this way, VHDL and gate-level designs can be integrated early in the design
cycle with actual hardware to test the correctness of the design. This results in a very high confidence level that when the design is integrated into the target hardware, it will work properly the first time.

LMC sells hardware models of many components already mounted on adapter boards, and also sells blank adapter boards, on which users can mount their own components. At Honeywell, we place our ASICs that have just come back from fabrication into the hardware modeler, and play functional test vectors on them to verify that they operate as expected. This is performed in addition to the manufacturing tests performed at the foundry. We have also used these custom adapter boards to incorporate chips into our simulations that are not available from LMC.

4.5 Testbench structure

The VHDL testbench effectively models the board hardware design by the use of the simulation models described in the previous sections. Many different types of testbench were created depending on the type of testing to be performed.

Each peripheral (I/O device, memory controller) that was designed for the board was created using VHDL, and had one or more testbenches associated with it for testing purposes. The generic structure of a testbench used for this purpose is shown in Figure 3. The peripheral models initially used a bus functional model (BFM) to emulate the RH32 processor bus interface to the peripheral (although the VHDL RTL description of the RH32 chipset was used at a later date). The BFM behavior was controlled by a script, or “macro” file, which contained an assembly language-like set of commands which described the type of bus operations that could be performed by the model. The macro file was designed so that the appropriate initialization and set up would be performed, followed by test cases consisting of different types and modes of data transfer. The response of the peripheral device model was monitored by the BFM, and unexpected results were reported by the BFM during simulation runtime. Once the designers were satisfied with the basic operation of their designs, a copy of the design library would be frozen, and passed on to a system simulation group for more rigorous testing.

The RH32 chipset underwent significant testing by the designers, and the VHDL representation of the released design was passed on to the system simulation group. The PDU testbench was developed by substituting the VHDL RTL designs for the behavioral models in the simulation. The RH32 VHDL description then became the basis for VHDL-based system simulation of the RH32 PDU. A block diagram of the RH32 PDU testbench, along with the types of simulation models used, is shown in Figure 4.

In addition to the RH32 chipset, each ASIC was designed using an RTL-style of VHDL. The simulation of these ASICs was performed using the VHDL RTL source for the devices.

The SRAM and ROM models used in the simulation were LMC SmartModels. The ROM models were initialized using text files which correspond to the startup ROM object code, which initializes the processor and peripherals, performs memory tests, and implements a debugging monitor by way of an RS-232 interface through the HI to a terminal. The start-up ROM code was simulated using the testbench before the board was powered-up for the first time to ensure that the processor would be able to properly perform its initialization and issue a prompt to the terminal.

The LM1200 and LM1000 hardware modelers were used to model the DUART, VIC and VAC/VMEbus interface, and the RH32 chipset. The versatility of the hardware modeler allows for multiple instantiations of each physical component in the modeler, providing a virtually unlimited number of these components to be modeled. This feature was extensively used in the system simulation.

VHDL behavioral models were also created for the other miscellaneous logic in the system, including the data buffers, IEEE-488 interface, and the real-time clock/watchdog timer chip. The behavioral models emulated the timing and interface protocols of the real devices, but except for the buffers, there was very little modeling of the internal functions of these parts. The behavioral models that were created were carefully designed according to the data book spec sheet of the part, and the model behavior and timing was verified for all the possible modes of the part by creating special simulation testbenches for the models.

Multi-processor testbenches were also created to test the bus arbitration features of the RH32 chipset. The single processor testbench shown in Figure 4 was essentially made into a component with a bus interface, and was instantiated multiple times in a higher-level testbench. The multi-processor testbench allowed very complex interactions between the CPU's to be simulated, which also complicated the debug and analysis efforts. Debug was approached in the same way as with the single-processor testbench, but special analysis techniques were used to allow performance measurement of the multi-processor system. These techniques will be described in a later section.
5. Model validation

As each ASIC was developed, testbenches were used to evaluate the functional correctness of the design. This involved a combination of visual inspection of the timing waveforms, and error checking performed by the bus functional models. An ASIC verification matrix was developed which identified the tests to be performed and cross-referenced the requirement in the design specification.

At some point, the simulation run of one or more modes of operation of the model is considered “golden”, and the results database (RDB) of the run was saved to disk. When future modifications were made to the design, the simulation run was performed again, and using Vantage’s comparerdb utility, the new simulation run could easily be compared with the golden run, and the differences highlighted.

Vantage also provides tools for comparing a VHDL design with the synthesized gate-level implementation. A simulation run of the model being tested is performed, and the RDB is saved to disk. The design then goes through logic synthesis and a schematic of the ASIC is created and back annotated with the delay information. The RDB created from the VHDL simulation can then be converted to Mentor Quicksim logfile format [7] by using the rdltxt utility. The input signals used to stimulate the VHDL design are captured and converted to the logfile format. The log file is then played on the gate-level implementation in Quicksim, and the output signal response is captured to a second logfile. This file is then converted by the Vantage log2txt utility, which converts the Quicksim logfile into a Vantage RDB. Using comparerdb, the two RDB’s - one created from VHDL simulation, and the other created from the gate-level simulation - can easily be compared, and the differences highlighted. The comparison is directed to compare the value of signals shortly before the rising edge of the clock, minus the desired setup time margin for the
device. Ideally, there should be no difference between the VHDL simulation and the gate-level simulation when this comparison is performed, and if a difference is found, it must be investigated for a potential problem. In this way, the VHDL model can be verified to be functionally equivalent to the gate level implementation.

6. Performance measurement

Analysis of the simulation runs is not limited to the "logic analyzer" view of the timing waveforms. Sophisticated post-processing can be performed based on data obtained from simulation run to produce useful information that could be used to optimize the design.

In order to perform this analysis, you must know what you want to analyze and how the necessary information can be acquired and processed. As an example, the throughput of a processor executing an algorithm can be obtained from the simulation by capturing the trace of the program counter (PC). This information can be post-processed to produce a throughput profile (MIPS), which could be used to optimize the processor hardware as well as the software development tools, such as the compiler optimization techniques. As another example, the bus control signals in the multiprocessing testbenches were used to analyze bus utilization in our system.

Performance measurement and analysis using VHDL is easily accomplished, and should be a part of the ASIC design process. These simple techniques can produce insightful information that could be used to improve the design before manufacture.

6.1 Simulation post-analysis

One technique for performing performance measurement of a design is to utilize the built-in support for results reporting provided by the VHDL simulator. For example, Vantage provides the capability to produce a text output table of the waveforms being viewed in the ResultsDisplay window. The data can be displayed based on a sample strobe, or on signal events. A table can be generated listing the values of the signals of interest, which is displayed along with the time at which that signal changed.

Our technique involved downloading the table to a Macintosh computer, and loading the table into Microsoft Excel. Once in the spreadsheet, the data can be post-processed to produce the desired information. The post-processed information is then plotted on a chart to graphically show the various performance items being measured. Subsequent analysis of the data can be directly traced back to events in the simulation run, potentially revealing areas of the design that could be modified to achieve higher performance.

As an example, a bubble sort algorithm was written in C, compiled to RH32 object code, and loaded into the memory models in the system simulation. The simulation was executed, and the value of the program counter (PC) was captured in the results display. After the simulation completed, a table was generated by Vantage from the waveform data. The data showed the time at which the PC changed state, and the corresponding PC value. The data was post-processed on the Macintosh to show the amount of time that elapsed during the execution of each instruction. From this data, a throughput value in MIPS could be generated, and this data was plotted to show a throughput profile of the bubble sort execution on the processor. A sample chart produced using this technique is shown in Figure 5.

![Bubble Sort Throughput Profile](image)

*Figure 5: Throughput Profile Example*
For this example, which was created from compiled C code executing on the RH32 processor simulation, the performance was initially low while the instructions and data to be sorted were loaded into the caches. After the caches were loaded, the throughput rose to a much higher average value due to the increase in the cache hit ratio. "Peaks" of very high throughput can be seen occasionally, which indicate that the processor did not experience any pipeline interlocks, and had data readily available to process. Since compiler writers are always looking for ways to optimize code execution in the target processor, a compiler developer could use this data, traced back to the simulation database, to further optimize the code based on analysis of the "valleys" in the curve and what the processor was doing at the time.

6.2 Performance measurement entities

Another technique that can be used for performance measurement which involves the creation of a special Honeywell-developed VHDL entity known as a performance measurement entity (PME). A PME acts as a passive monitor of events in the system simulation, and does not affect the behavior of the simulation in any way. A PME is used to monitor a set of signals in the simulation, and performs some processing on the data that may be difficult to accomplish in a spreadsheet application. The PME uses VHDL TEXTIO functions to write the data it gathers out to a text file for later processing. This data is then in a format suitable for processing with a spreadsheet.

For example, in our multiprocessing testbenches we wished to obtain statistical information regarding the performance of the multiprocessing bus arbitration occurring on the Module Bus. For this example, four processors and a memory subsystem shared a common bus, and each processor performed reads and writes to the shared memory. The processors arbitrated for access to the common bus, and were able to perform the remote read or write operation after the bus was granted. The processors that did not get granted each had to wait their turn according to a round-robin arbitration scheme.

A PME was created for this testbench which monitored the Module Bus arbitration by way of the bus request and grant signals. Data measured by the PME and written to an output file were such items as request rising-edge time, grant rising-edge time, request-to-grant delay, grant duration, number of words transferred, etc. Using a subset of this data, a chart was generated showing the module bus contention for each processor, in the form of request-to-grant delay, as shown in Figure 6.

Post-processing the data without a PME would have been more difficult. The spreadsheet would have had to search the raw data to look for 0-to-1 transitions of the request and grant lines, and a complex calculation would have been required to estimate the number of words transferred. However, such tasks were easily accomplished in a VHDL-based PME.

![Figure 6: PME-measured Bus Contention](image-url)
7. Conclusions

This paper describes several proven techniques for performing VHDL-based system simulation. The system simulations uncovered several design problems with the peripheral ASICs that were not discovered by the designers own testbenches. The problems that were found were corrected and the fixes verified before the chips went to fab.

When the hardware was brought to the lab for debugging, the problems encountered were non-simulated problems such as signal noise, board defects, software errors in code that was not exercised in the simulation, and NFT manufacturing test logic that was not part of the VHDL description. Very few problems were found in the logic that was simulated prior to build, and those problems were found in simulation due to faulty or inadequate test scenarios. The problems that were uncovered in the lab were verified by system simulation, and the simulation was then used to test solutions to the problem.

This paper also discusses techniques for performance measurement using VHDL. Performance measurement can be performed by post-processing the simulation results data, and more complex analysis can be performed by using a performance measurement entity (PME), which performs some reduction of the data that would be difficult to do in a spreadsheet. Statistical info can be obtained regarding system performance, and can be traced back to the simulation database. Analysis of the performance data can be used to improve the system performance before the design has been committed to silicon.

The results of the activities described in this paper suggested a number of areas deserving further development. The power of the performance measurement techniques can be applied much earlier in the design cycle. The systems engineers could generate high-level behavioral models of the main components of the system, and PME’s and post-processing could be used to analyze the system performance. Additionally, the behavioral models created by the systems engineers could be passed on to the hardware group as “executable specifications” of the ASIC behavior. When the detailed RTL is available for inclusion in the system simulation, the behavioral model of the ASIC would be removed, and the detailed RTL model put in its place. The RTL model should exhibit nearly the same behavior as the behavioral model, and should be able to interact with the other behavioral models in the system.

Another capability that was not fully exploited on this project was the automatic generation of testbenches directly from the schematic database. Vantage’s Import/Export application can be used for this purpose. The application can import the Mentor Neted schematic database, and convert the database and export it as a VHDL netlist. Execution of this tool on the top-level board schematics effectively produces a testbench which could be used for system simulation. The VHDL netlist requires some manual editing, and a shell must be created for each board-level component which maps to the simulatable VIDL entity that corresponds to it. Use of the Vantage Import/Export tool would reduce the potential for human errors that might be introduced by the manual generation of a testbench from the board schematics.

8. Acknowledgments

The system simulation activities described in this paper were performed by members of the Advanced Systems Engineering department at Honeywell Space Systems, Clearwater, FL. The multi-processing testbenches and their complex test scenarios were developed and simulated by Ron Wilcox, who also developed the RH32 MBUS bus functional model, which proved to be an indispensable tool during early simulations. The concept of the PME was jointly developed by Karen de Lucca and the author.

9. References