

Statistics and Heuristics Developed from Systems Engineering with VHDL Models

Chris B. Curtis
Motorola Government and Systems Technology Group
8201 E. McDowell Rd.
Scottsdale, AZ 85252

Abstract

Several design heuristics and methods have grown out of our experiences in using VHDL behavioral models as system design tools. This paper summarizes data gathered during model development and describes design methods developed to increase the utility of such models.

Section 1. Introduction

Over the last three years, we have developed a number of behavioral models using VHDL. The goal of these simulations is to validate the design of subsystems and to verify the design of their interfaces¹.

The study includes four models simulating various types of cryptographic processing subsystems for use in an avionics environment. These models are behavioral in nature and simulate the flow of messages into and out of the subsystem. Each model is comparable in size and complexity with an average model size of approximately 5,000 lines of executable code.

Each model interfaces to a specially developed testbench which in turn interfaces to a scenario development tool (scenarios are lists of messages sent and received by the system during specific operations). This scenario development tool allows simulations to use the same vector set used for testing the real system.

Section 2. Technique

This study examines the development of four models. The technique used to develop these models evolved as each new development unfolded.

The first model development simply used a typical top-down design approach². The methodology applied no special techniques for VHDL or modeling.

Development of the first model (model A) required much more time than expected. In fact, model A took so long to complete that it contributed very little to the development of the system. Without significant development cycle time reductions, this type of behavioral model could not be a useful system design tool.

It became clear during the development of model A that it is possible to develop a standardized architecture for similar systems. While standardized, the architecture must be flexible enough to apply to a wide class of these systems.

The standardized architecture that evolved from model A is shown in Figure 1. The architecture includes a set of concurrent processes for each input interface. A message received at an interface is converted by a set of conversion functions (one overloaded function for each message format) to a standard format.

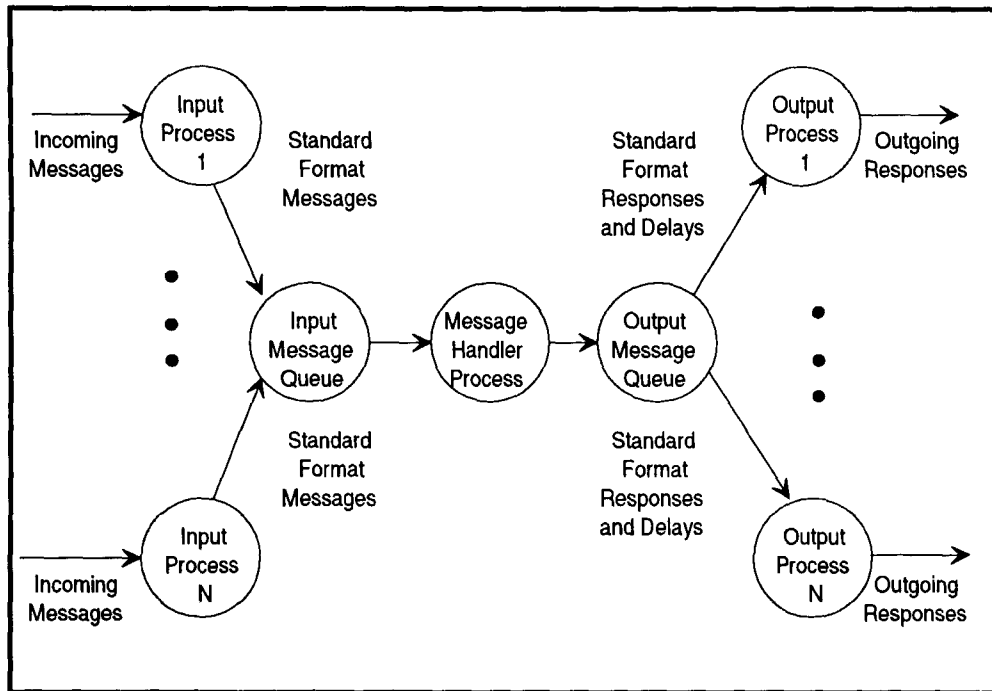


Figure 1. Standardized Model Architecture

Once formatted, the received message is stored in a queue. The messages in this queue are serviced by a central message handling process. The message handler verifies the validity of each message against the simulation's current state and against the contents of the message. One of a set of functions (one for each message type) performs the validation.

The message handler responds to each incoming message. The responses are generated by a set of functions (one for each type of response). The output of the response function is a standardized message and time delay information.

This response data is queued for sending to the correct output interface. A different concurrent process services each output by taking messages off the output queue that are destined for the process' interface. The process waits for the specified delay time and outputs the message.

The second model development (model B) pioneered the use of a rudimentary form of the standardized architecture. This proved to significantly reduce development time and helped to reduce errors in the model.

Even so, model B also took longer to complete than anticipated. However, its development time was short enough for the model to produce some benefits. For example, the model detected errors in test vector sets subsequently used to test the system.

The third model (model C) was a more ambitious effort than prior attempts. A more evolved form of the standardized architecture was available allowing modeling of a more complex system.

Although model C was larger than previous models, total cycle time was significantly reduced. The reduction in cycle time is almost entirely attributable to the use of the standardized architecture.

Even with the reduced cycle time, however, development time was still too great. As the project progressed, the developers began applying a "pin point" approach. This approach isolated high-risk functions that the developers implemented first. The developers reserved more familiar and less complex functionality for implementation at a later time.

This pin pointing technique allowed early implementation and modeling of those portions of the system most at risk. The technique allowed the model to fulfill its role as a design tool for high risk portions of the system.

In developing the fourth model (model D) the designers were able to apply the lessons learned in the first three models. Model D used a highly advanced form of the standardized architecture. In addition, the model contained only high risk system functions.

Implementation of model D was complete prior to the start of the system's preliminary design. Early completion allowed this model to produce benefits that the previous models were unable to deliver.

For example, developers of the system performed several "what if" simulations to verify operational scenarios. Simulations detected a number of errors in the design of system message formats. Early detection of these errors greatly reduced the cost of making corrections in the system design.

Each of the four models described above interfaced to the same test bench software. This software, in turn, interfaced to a set of test generation tools developed to generate test vectors automatically from scenario diagrams contained in system design documentation. These test vectors are used for testing both the model and the system once it has been developed³.

This automated test generation technique has three major benefits. First, test generation cycle time is very short. Second, tests can be generated with fewer errors.

Finally, the test vectors generated for the model's test bench are guaranteed to be the same as the test vectors used to test the system. This third and most important benefit of the automated test generator helps insure that errors detected in the model are system errors and not some artifact of the tests themselves.

Section 3. Statistics

Each model development described in the previous example has an associated metrics database. These databases contain, among other things, the system design errors detected during the modeling process.

This study includes analysis of numbers of errors detected versus time of completion, number of errors detected versus model design phase, and approximate development time versus development approach. These statistics illustrate the utility the different model design approaches.

In the first analysis, the number of system design errors detected by a model is compared to the system design phase in which the model was completed. Figure 2 shows this relationship.

Figure 2 shows how the completion time of a model has a large impact on its utility. For example, model A was not complete until late in the design cycle of the system. Later models required less time to complete and, as the figure shows, were more useful to the system designers.

Figure 3 shows the average number of system design errors detected during model development versus the model development phase. The averages are taken across all four model developments.

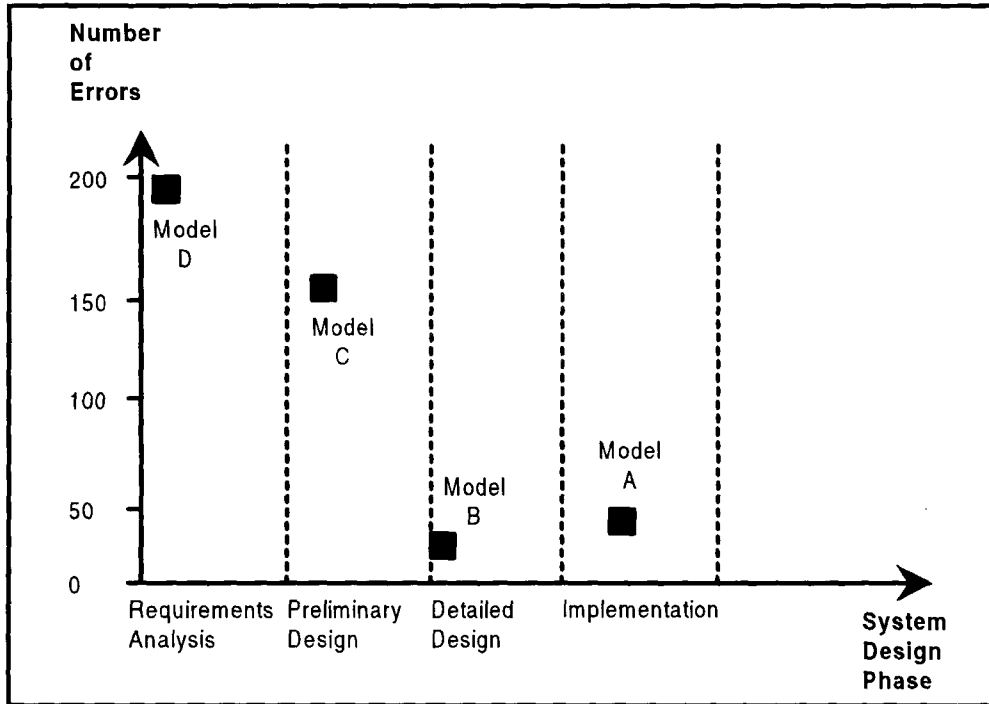


Figure 2. Detected Errors Versus System Design Phase

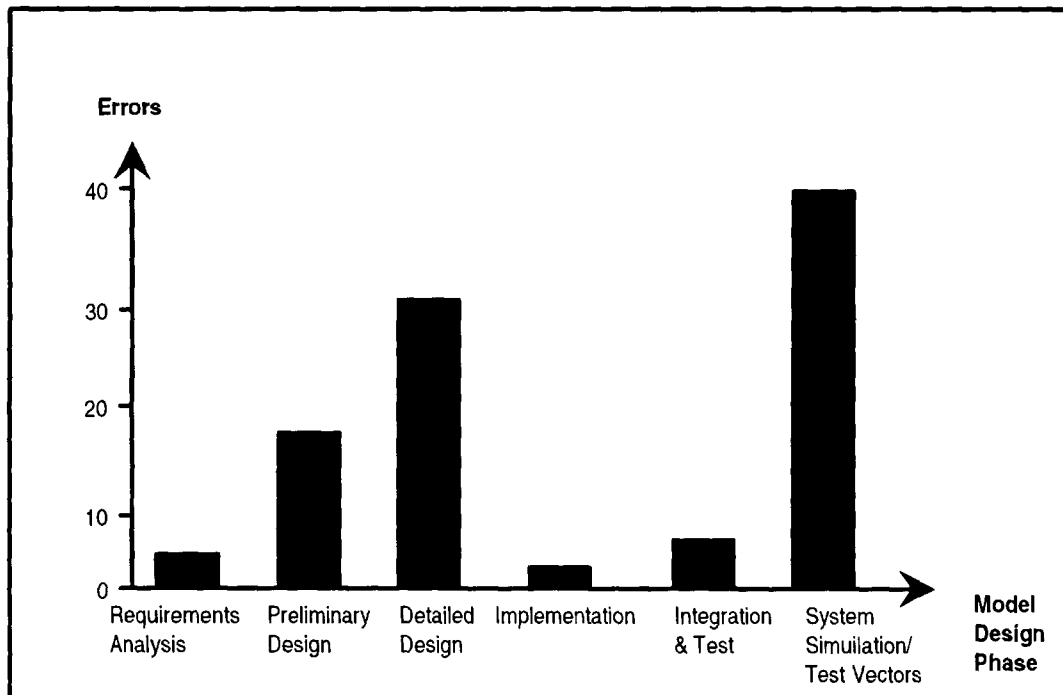


Figure 3. Average Number of Errors Detected Versus Model Design Phase

As the figure shows, it is not necessary to complete the model to realize most of the benefits. There are two peaks of error detection. The first peak is fairly early in the model development, the second peak occurs when the model is complete. During implementation (coding), very little will be learned from a model.

Figure 4 shows total number of hours required to develop each model. Model A required the most total time to develop. Model B, with an early form of the standardized architecture took less time.

Model C required nearly as much time to complete as model A even though the standardized architecture was used and pin point functionality was introduced. Model C, however, is a much more complex than the others.

Model D implemented all the techniques learned in the previous models. Although it was as complex as the other models, it was completed very quickly. Model D represents

a 4 fold decrease in cycle time over model A.

Section 4. Heuristics

The statistics presented in the previous section imply a number of design heuristics. These methods increase the beneficial results of a modeling effort.

The most obvious lesson learned from the models in this study is the need for speed. Models must be complete early in the system's design cycle to be useful. The benefits produced by a model are inversely proportional to the development time of the model.

A qualitative interpretation of the data in Figure 2 is shown in Figure 5. The figure shows that the number of system design errors detected by the modeling process is very dependent upon when the model is completed.

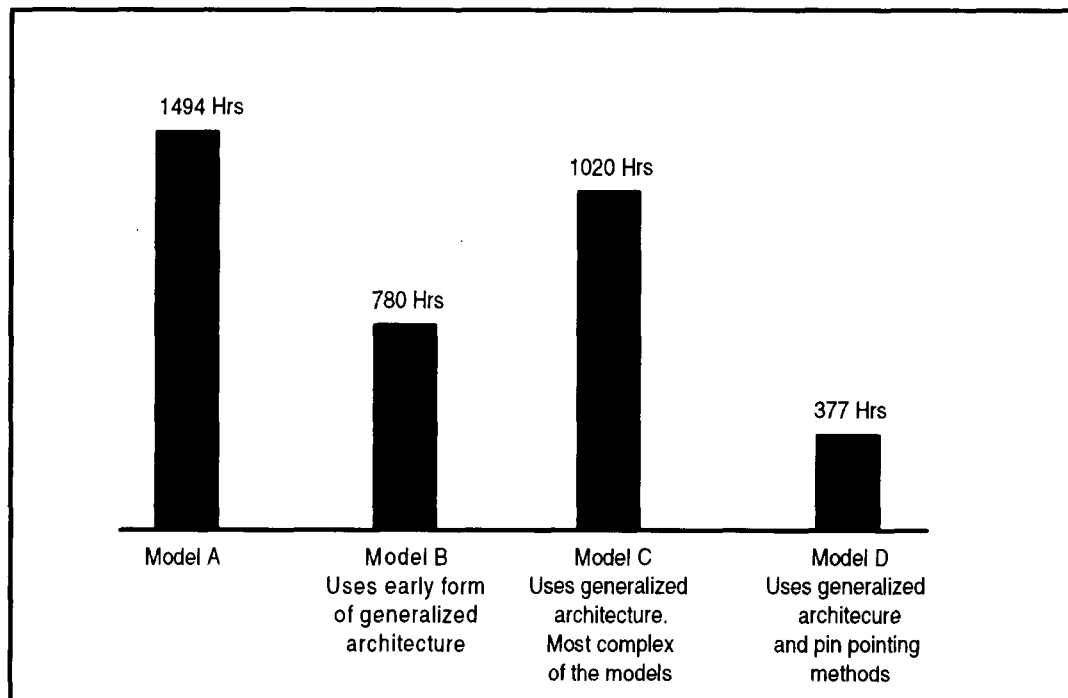


Figure 4. Time Required to Develop Each Model

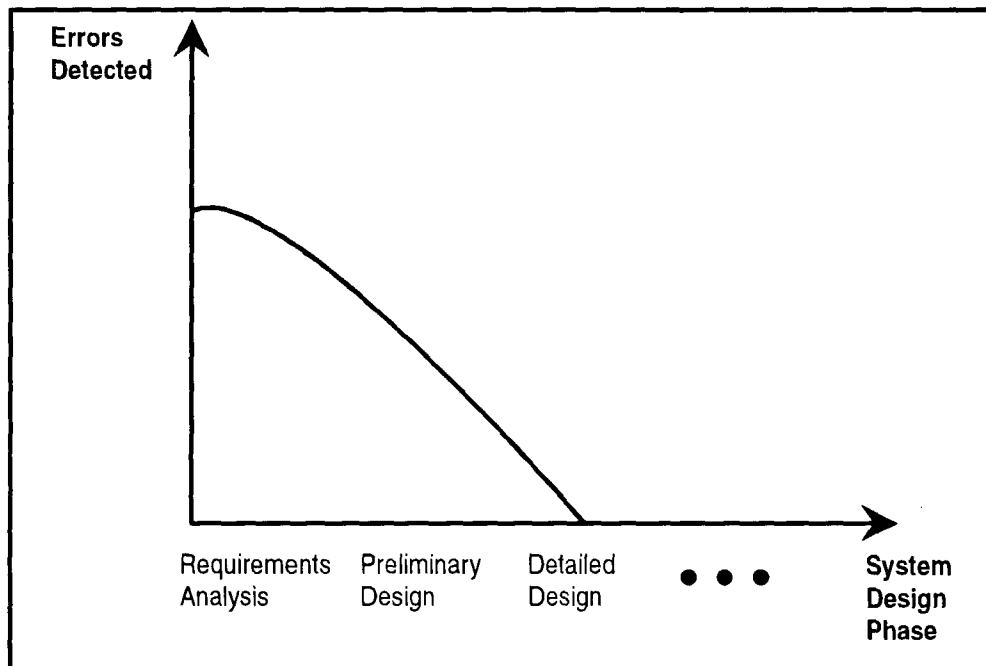


Figure 5. Qualitative Relationship Between Errors Detected and System Design Phase

It is possible to optimize development of a model to take advantage of the fact that most of the benefits of the development are realized prior to implementation of the model. If the model development is complete through detailed design before the system design reaches detailed design, most of the potential benefits of the model can be realized. The implementation phase of the model is essentially a "dead zone" where little system error detection is realized.

However, if the model is completed quickly and some flexible method⁴ is implemented for generation of test data, a second peak of error detection activity occurs. This peak is caused by the model's ability to run "what if" scenarios before the detailed design of the system is complete.

Another important lesson learned from the model developments described here is the need for a standardized architecture. Often, systems can be grouped into classes that share certain characteristics. If a standardized architecture is developed that

implements the characteristics of a class of systems, rapid VHDL prototyping becomes possible. The use of standardized architectures for the class of systems described here produced a four fold decrease in cycle time.

Given time, it may even be possible to automate the use of such an architecture. The standardized architecture designed for the models described here is a candidate for this type of automation. Development of a tool to automatically generate a VHDL model given certain system parameters appears straight forward. Such a tool could produce ten fold decreases in cycle time.

Another method that decreases cycle time in model development is pin pointing of functionality. Design the model so that only high-risk functionality is included. Rigorously eliminate functionality that adds no value to models.

This is especially important for functionality that represents "infra-structure". For

example, implementing a detailed model of a communications bus is counter productive if the bus is well understood and the real goal of the model is to study message sequencing. In this case, the bus architecture is infra-structure rather than functionality and is not important to the system designers.

Section 5. Conclusions

The development of a VHDL behavioral model is a powerful system engineering design tool. The power of the model as a tool, however, depends on the how the model is developed and used.

Developing the model quickly while following good design guidelines serves to detect design errors, reduce design risk, and enhance confidence. Techniques like using standardized architectures speed up model development.

The design heuristics developed in this paper make it possible to maximize the

benefits of a behavioral model. Realizing this goal makes behavioral modeling a useful system design tool.

References

- (1) Schaefer, Bradley R., and Worger, Bill, "Systems Design Methodologies Using VHDL", *Proceedings of the VHDL International User's Forum*, May, 1992.
- (2) Perry, Douglas L., "VHDL", *McGraw-Hill, Inc.*, 1991, 281-285.
- (3) Lindsey, Michael K., "Automated Test Procedure Generation from System Specifications", *Proceedings of the Third International Workshop on Rapid System Prototyping*, June, 1992.
- (4) May, Phil, "A Flexible VHDL Test Bench Architecture", *MILCOM Proceedings*, October, 1992.