

Processes, Metrics and VHDL

by Mike McCollough and Wing Lee

Hughes Aircraft Company, Sensors and Communications Systems Segment

mike@tcville.es.hac.com, wlee@igate1.hac.com

(310) 616-5822, (310) 662-6263

Abstract

We have worked on VHDL design processes for the past 5 years on a variety of projects, from large satellite signal processing payloads and avionics image processing systems to performance level modeling systems. During this time we have continued to refine the components of the design process.

In this paper, we show how we applied our design process to the 1996 VIUF VHDL Design Contest. We credit this process to our entry winning the industry competition. We will describe our process, the metrics we collected and the information we learned from the metric data.

Introduction

We use VHDL to design a variety of digital electronics, from integrated circuits to complete systems. With such a broad range of applications for VHDL, what constitutes a good process for using VHDL? How do we know that a process is the best one to use?

One way to determine how well a process works is to collect metrics. Metrics come in various flavors, from defects on a ASIC or PCB, to design cycle time and hours spent on each task. Metrics accomplish two functions. One, metrics demonstrate that you are following the process, since you are collecting metrics at each step. Two, metrics assist you in grading a process. Metrics determine the areas where the process

works well and the areas where it does not. With this information, the process owners can focus their attention on the problem areas of the design process, and work to improve them

Our company, Hughes Aircraft, has spent the past several years standardizing, refining and documenting the processes used throughout the company. The goal of this effort is to have common processes to build missiles, radars and night vision systems. This allows engineers to easily work the product lines without having to learn a new way of designing. In addition, standardizing on processes, allows us to standardize on tools and save on both software and training costs.

The process that we have been using for VHDL based designs has evolved over the past five years. It has been successfully used on a variety of programs from a multiple ASIC space based surveillance system, to a FPGA based video processing subsystem.

Process Goals

A process is a sequence of steps that if followed will result in a product of known quality. A process should be repeatable and not be dependent upon the unique characteristics of the design team members. The key metrics used to grade a process are design cycle time and design cost. We want to reduce the design cycle time and minimize the design cost.

Design cycle time reduction is difficult to achieve directly. Consequently, we created 4 sub-goals that we used to drive our process to meeting this goal. These

are design reusability, top-down design, real world test benches and containment of design errors.

Design Reusability: We want to maximize the reuse of VHDL. However, in order for VHDL code to be reusable, it must be designed with reuse in mind. This requires that the VHDL be thoroughly commented, that signals be given meaningful names, and that the VHDL be thoroughly tested. In addition, it requires dividing the design into common design elements so that duplication of effort is minimized.

Top Down Design: The design should flow naturally from the requirements and decompose into hierarchical sub-units. This ensures that the design meets all of the requirements and has traceability. In addition it provides a means to identify the impact of requirement changes.

Real World Test Benches: We want to simulate the design as close to the real operating environment as possible given time and resource constraints. A well-known axiom in hardware design is that the part that fails is typically the part that was never tested or tested incorrectly. Consequently, we try to duplicate the real-world environment as much as possible. Obviously, we cannot duplicate the physical conditions that the design will face: ground bounce, clock skew, mismatch impedances, etc. However, we can try to replicate the interface logic that the design must interact with. We achieve this by creating test benches that incorporate functional models of all of the components with which the system communicates.

In cases where it is not feasible to completely duplicate the operating environment, we selectively choose which interfaces to simulate against. For example, we could choose not to simulate an interface that is relatively easy to modify in the laboratory.

Containment of Design Errors: We want to minimize the quantity and types of errors that escape into the next level of the design. The earlier we find errors the cheaper they are to fix. It is much less expensive in both time and money to fix a design problem when only a single person is involved, than it is when you have a team of engineers involved as during integration.

Process Features

In order to help meet the above sub-goals, our process incorporates several key features.

Executable Specifications: We use Executable Specifications to document the requirements. An executable specifications is a functional or algorithmic active model of the system. We compare this model's behavior with the results of the synthesized design to verify the original requirements are being met. This eliminates requirements interpretation errors.

Subunit Testbenches: The process requires that all lower level units be thoroughly tested before they are integrated into higher level units. Thus, if an ASIC consists of five functional units, each functional unit must have a VHDL testbench associated with it. Before we can begin integrating the five functions to form the top-level ASIC, we must ensure that each functional unit has passed all of its tests. This enables us to minimize the type and number of design errors that escape into the next level of the design.

Email: Our process encourages engineers to send email messages to other team members whenever important design decisions or assumptions are made. This not only allows team members to be informed of design changes, but also provides a means of capturing design tradeoffs and rationale as they are made. It is very difficult to reconstruct this information at the end of a project.

Electronic Co-Location: Often the engineers on a project are not located at the same location. Our process allows engineers to communicate their progress electronically, via the use of FTP, video conferencing, email messages, and telephone conferencing. Each team member is able to utilize the work generated by another team member at a different site. This includes sharing design libraries and writing portable VHDL so that team members can be working with different sets of tools (e.g. team members using workstations and team members using PCs).

Configuration Management: Our process encourages the design team to utilize a configuration management system to maintain source control of the VHDL. This not only allows for checkpointing and rollback of designs, but also ensures that team members use “released” versions of VHDL modules. The process allows team members to keep working copies of their VHDL files in their own directories, but requires that they periodically “check-in” working versions of their code so that other team members can incorporate their code into other design modules. It is the job of the Librarian to perform regression testing to ensure that all newly checked-in code passes all functional tests before being promoted into the public area.

Error Logging: We believe that logging of the types of errors and when they occurred provides valuable insight into the development of the design. The types of errors can show whether the design subunits are being tested to an adequate level. The timing of when errors are found can be used to determine the maturity of a design. If errors are being found every hour then the design is not ready to be released. On the other hand, if errors are being found only once every 3 days, then the design may be almost done.

Metrics Collection: Metrics are required by the process to allow us to identify classes of errors for future process improvement and provide estimate the time needed for similar tasks. If much of the time (and money) is spent during integration, then we can focus our attention on determining ways to reduce the integration time. In addition, if it takes 100 hours to build a function of type X and we have to do a similar function of type Y for another program, we can use the 100 hours as a basis for estimating the work involved in making function Y.

VIUF Design Contest

The VIUF design contest problem was to demonstrate using VHDL the performance of the algorithm shown in Figure 1 on various permutations of the architecture shown in Figure 2. This architecture could contain up to five processors and the communication network could support up to three simultaneous channels. The solution to this problem had to include a simple means of describing the

algorithm, it’s assignment to the architecture, and a means for changing the number of processors and communication channels.

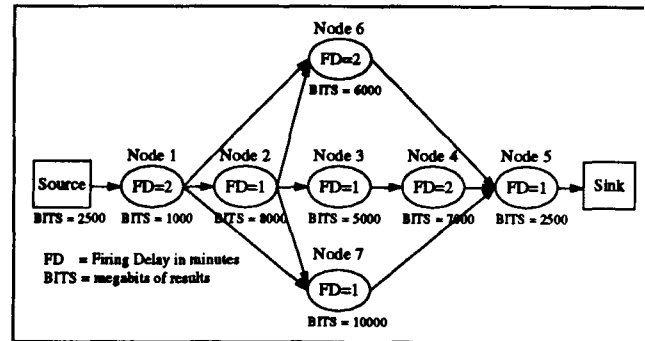


Figure 1 Application Algorithm

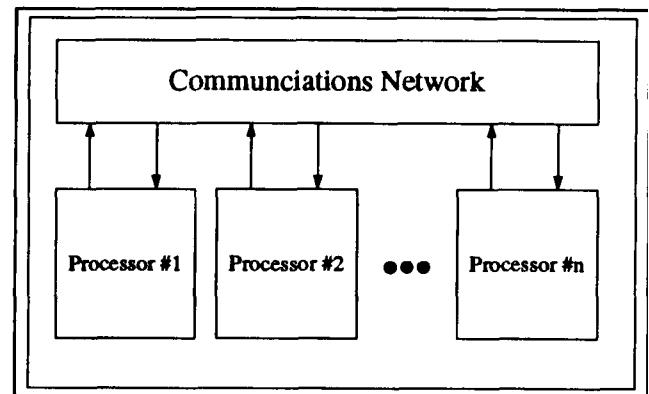


Figure 2 Processing Architecture

Since this problem is simpler than the projects we typically work on, we tailored the process to the problem. When applying common processes, tailoring is used to optimize the process by adding or removing process steps. In this case, we removed several process steps and simplified the remainder.

A block diagram of the tailored process we used to solve the VIUF Design Contest is shown in Figure 3. The process is broken down into three tasks; requirements analysis, functional architecture development, and design synthesis.

Requirements analysis is the procedure of documenting the characteristics that the design must

exhibit. This includes defining the system inputs and outputs and defining the environmental conditions in which the system must reside. In addition all assumptions that are not directly specified by the customer must be documented.

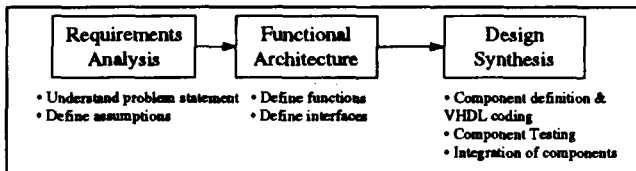


Figure 3 VIUF Design Process Steps

The functional architecture task develops a solution to the problem that meets the requirements. This is where we perform all of the design trade-offs to find the optimal solution to the problem.

The design synthesis task implements the solution into a form that can be delivered to the customer. For the VIUF Design Contest, we further defined the design synthesis into 4 sub-parts. These were VHDL coding, VHDL Component Testing, VHDL Integration and Results Analysis and Design Enhancement. This refinement is shown in Figure 4.

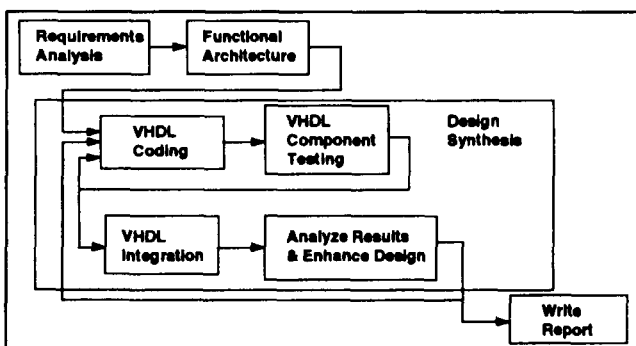


Figure 4 Design Synthesis in Detail

Metrics

We captured metrics to measure how we performed in solving the design contest problem. We used pencil and paper, along with e-mail and log files to collect this information. We divided the metrics into two groups;

process and errors. The process metrics allowed us to learn where we spent our time in the process and what we produced. The error metrics allowed us to learn the types of errors we uncovered and how long it took to correct them.

Process Metrics

The design effort metric shown in Figure 5 allowed us to determine where we spent our time. It took 128 hours to create a solution to the design problem. We spent 3% of our time in requirements analysis. We spent 13% of our time designing the functional architecture. We spent most of our time, 62%, synthesizing, coding and testing the components. The result of spending so much time in coding and testing was that our integration effort only taking 12% of the overall effort.

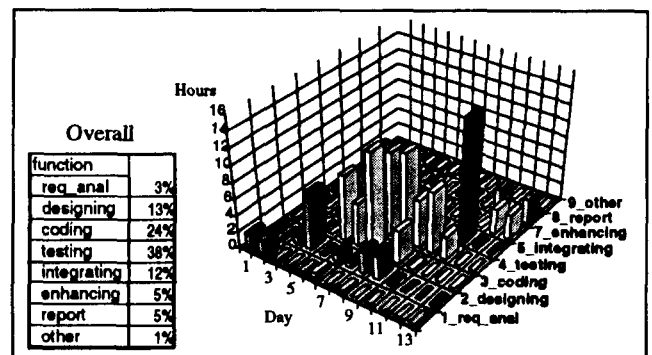


Figure 5 Design Effort

Error Metrics

The error metrics provide a means for understanding the types of errors created in the design process. Any system being designed will have some number of errors. We use the term error to mean a deviation from the desired behavior of a function. What we would like to do is discover and correct errors as early as possible in the design process. By collecting metrics on when and where we detect errors we can then formulate, implement and test methodologies to reduce the errors

We have categorized the errors into 7 types.

1. *Function Behavior*: The logic the function implements is incorrect. We divide this into three sub categories:
 - General: incorrect logic
 - Array Indexing: the array size or value locations in array are wrong
 - Polarity: Signal/Variable has incorrect polarity
2. *Side Effects*: An error introduced by the process of correcting another error.
3. *Simulator Initialization*: These are errors caused by the startup condition of a signal or a variable.
4. *Chip to Chip Interface*: Mismatch in the interconnection between two components.
5. *Miscellaneous*: all other errors
6. *VHDL Nuances*: Errors caused by misunderstanding of the nuance of VHDL.
7. *Test Benches and Artifact errors*: Errors that are found in the test benches used to test components or are introduced by the test method.

While generating our solution to the VIUF contest we discovered and corrected 48 errors. Figure 6 shows the types of errors. Of the 32 errors found in implementing the processing element (PE) processor, we attribute the 5 in the categories Misc. and VHDL nuances to our inexperience in using certain VHDL constructs.

	PE	Comm	Integration	Total
Function Behavior	15	6	1	22
General	6	6	1	
Array Indexing	3			
Polarity	4			
Side Effects	6	2		8
Simulator Initialization	5	2	1	8
Chip to Chip I/F			3	3
Misc (File I/O & Spec Interpretation)	3			3
VHDL Nuances	2			2
Test Bench	1	1		2
Total	32	11	5	48

Figure 6 Types of Errors

The sequence the errors occurred is shown in Figure 7. Early in the development cycle we found behavior errors. Solving these errors introduced some side effects. After we resolved those we found more

behavioral errors. As we resolved out the simple errors we started uncovering more subtle types of errors; ambiguous definitions, simulation initialization and VHDL nuances. Finally when we started testing the processing element with the system we uncovered test bench and interface errors.

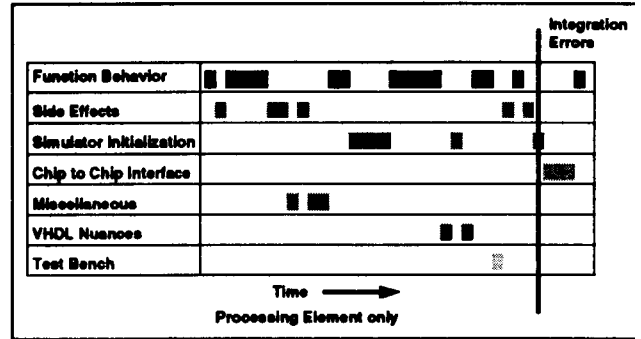


Figure 7 Error Detection Sequence

We then determined how long it took to correct the errors. Figure 8 shows how long each error found in the processing element took to be detected and corrected. We found that most of the errors found during development take very little time to be corrected, whereas during integration they could take five times as long. We discovered most of the function behavior errors early in the development cycle as a result of our process requirement of testing components prior to integration.

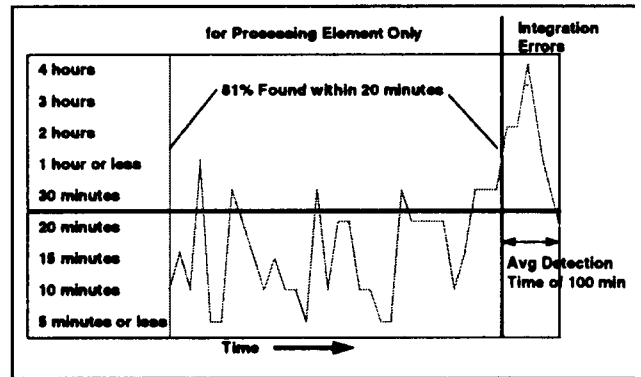


Figure 8 Error Detections and Correction Time

Conclusions

We have described the process we used to solve the VIUF Fall 1996 design contest and our measurements of the implementation of the process. We have shown the types of errors we uncovered in implementing our solution and how long it took to correct them. We spent very little of our time integrating the parts of our solution together because we used subunit testing to remove many of the errors early on.

Acknowledgments

The authors would like to acknowledge the support of Hughes Aircraft Company, especially the Signal Processing Center management: Kevin Moore, Mike Vahey and Dennis Wong for allowing the authors time to complete the project.