

Systems Design Methodologies Using VHDL

Bradley R. Schaefer and Bill Worger
Motorola Inc., Government Electronics Group
Scottsdale, Arizona 85252

Abstract

The use of VHSIC Hardware Description Language (VHDL) as a modeling, simulation, and synthesis tool in an engineering design methodology is presented in this paper. Use of VHDL models can be extended into the verification of specifications, test plans, and test procedures prior to commitment to hardware, fully developed software, and LSI, thereby catching problems earlier in the design cycle. This paper discusses the possible uses of VHDL in various parts of the design cycle, and comments on the viability of using VHDL in design phases.

Introduction

VHDL is a relatively new "standard" language for documenting/defining hardware, and with some extensions, perhaps software designs. The language provides the engineer the capability to develop models of systems that can be simulated prior to development and fabrication, thereby finding integration, design, and even specification problems early in the design cycle. Identifying problems early on will likely save significant time, effort, and rework to the design. This type of modeling also provides a mechanism for validating test plans and procedures prior to real hardware and software integration.

VHDL by itself cannot provide all of these benefits; an organized design methodology must be developed to accommodate modeling activities, along with a design framework in which the system models are created. This framework implies an efficient method of developing test information, a method of incorporating this information into VHDL simulations and modeling activities, and a technique for evaluating simulation results. Figure 1 illustrates a top-down design methodology, emphasizing simulation, VHDL modeling tasks, logic synthesis, integration tasks, and final end-item test activities. This figure demonstrates the ability to link top level system design, to intermediate design tasks, to final integration activities.

This paper discusses some of the advantages and problems in the approaches demonstrated in Figure 1, and discusses a design methodology that can utilize VHDL in various project design phases (requirements, design concept, preliminary, detailed, fabrication/qualification), suggesting possible frameworks that can support full system level modeling.

System Modeling — Top Level Black Box Modeling Requirements Phase

In the requirements phase, a Black box model of a system that is exercised from a message flow standpoint can be very useful in validating the requirements of the system, as well as defining/identifying the internal behavior of the system. This type of modeling is useful for digital systems that have significant data processing. A message flow model allows the transfer of information between the model and the VHDL testbench (which acts as the External System) in a record format, not in a real physical I/O sense. These message records have limited amounts of data, basic enumerated data types initially, and can be sent over representative discrete signals or buses at representative data rates of the actual data transfer.

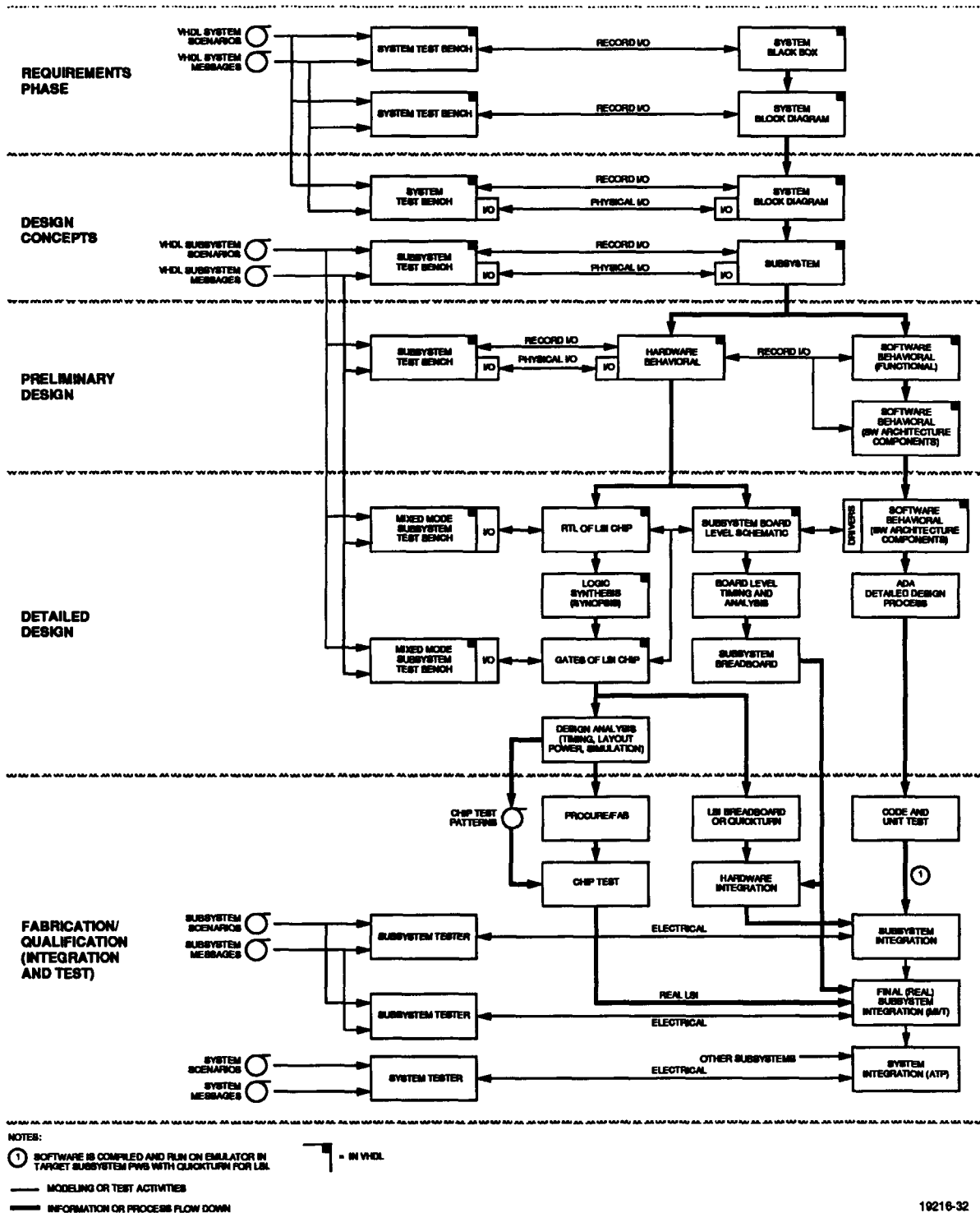


Figure 1. System Design Methodology Using VHDL

19216-32

Having such a Black box model clearly defines the stimulus and responses of the system, specifies timing associated with the stimulus/response and for the internal processing of the system, and provides the designer and the customer with a clearer picture of the overall behavior of the system. The use of enumerated data types in the message records implies that the actual processing in the system model may be reduced or of limited scope until real representative data is used in the model (as in the design concept phase). Use of enumerated types allows the definition of stimulus and response of the system. The

problem with such a model is that in order to be exhaustive in detailed functionality, it may take a significant amount of time to develop, perhaps longer than is available. Therefore, this model is viewed as a preliminary step towards defining the system, and as much detail as possible is rolled in based on time/budget constraints. Figure 2 demonstrates a typical message record structure in VHDL and how it maps to nonvariant records.

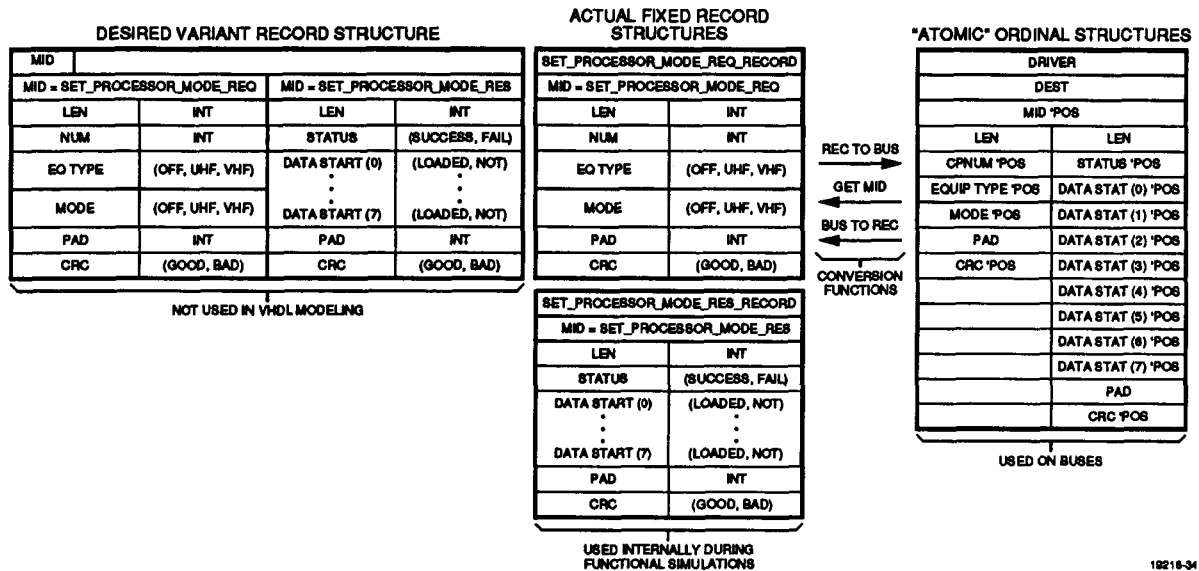


Figure 2. Message Structure Implementation in VHDL Simulation

If more details of the system are known in this phase, and the system is already decomposed into major subsystems, a system block diagram model will be more useful than a pure Black box model. Again, message flow data records are used between the subsystems, and each subsystem needs to be tested with its own VHDL testbench before being integrated into the full VHDL model of the system.

It must be emphasized that all data flow is modeled with enumerated data record messages, simply asserting the coarse functional behavior of the system without detailed bit level interactions. The VHDL code for the subsystems is written at a level that does not reflect any real hardware architecture (not at RTL or structural levels), but has the flavor of a high level programming language, using processes and constructs that are functionally oriented instead of implementation oriented. The details of how test information is injected in the VHDL testbenches is discussed later in this paper.

Top Level Subsystem Modeling — Design Concept Phase

Once a preliminary block diagram model is developed, additional details and refinements are rolled into the model, improving its accuracy and real world details. Instead of having enumerated types in the message data records, real representative data can be inserted in the record I/O, allowing real processing. This can be done on a field-by-field basis within the message record, allowing incremental upgrades to the functionality of the model that acts on real data. Critical interfaces that are not clearly understood or that are risky may require more detailed modeling in order to provide more definition and timing information. These critical interfaces may be behaviorally modeled down to the bit level, establishing the physical interfaces and protocols, forcing the engineers to define this interface and its timing/protocols. Even though the model describes the physical/bit level behavior, it still does not define the structure, implementation, or details of the architecture. It simply provides an executable definition of the critical interface.

At this phase, both data record messages and real physical I/O may interact with the model. The VHDL testbench must be modified to accommodate the physical I/O enhancements to the model. This presents data representation problems in both the testbench and in the model itself. Data may be represented at three levels in the model at this time, or mixed as needed;

- Enumerated Data Record Messages.
- Real Data Record Messages.
- Real Physical I/O Messages.

The model and testbench must be designed to provide reasonable transitions between these various levels of detail, with the most work found in the transition to real bit level physical message. The most difficult problem in doing this kind of message modeling is due to the lack of variant record support in VHDL. This problem can be overcome using a stimulus/response modeling technique that converts a single bus data structure to a form of variant record using overloaded VHDL function calls.

At the end of this phase, sufficient simulation/modeling activities should be completed so that a viable hardware and software partitioning can be performed. The confidence level in this partitioning is high, as the functionality of the system or subsystem has been significantly explored and documented.

Functional Software Modeling — Preliminary Design Phase

In order to create a model that truly represents detailed behavior of the system, a HW/SW partitioned, a mixed level representation of system may become more necessary. Hardware may be readily represented at behavioral, Register Transfer Level (RTL), or at gate/structural levels. Software representations are not so straightforward as many problems exist in modeling software in VHDL. Only two approaches appear readily feasible for modeling software behavior using VHDL:

- 1 - Obtain a gate/RTL/behavioral level model of the microprocessor targeted for software. Compile test or final software down to object code, and run this object code on the processor model.
- 2 - Develop a behavioral model of the software functionality; augment the SW functional model with physical I/O drivers to "emulate" a subset of the microprocessor's physical I/O behavior.

Both approaches have strengths and weakness that must be addressed and evaluated before a specific approach or path is taken. The usefulness of the model must be weighed against the schedule, cost constraints, and risk profiles of the development program.

Microprocessor Models

In case 1 above, a microprocessor model running real code in simulation is the most straightforward approach for modeling, and is easier to do when you actually own the gate or RTL implementation of the processor. Disadvantages in running this kind of simulation:

- The software running is in machine code format only; simulators currently do not support source level debug of high level languages. This makes debugging of Ada source code (or other high level languages) difficult because all you have is machine code (assembly) to work with.
- It is usually difficult to obtain RTL or gate level netlist of the microprocessor designs. Even behavioral models may be difficult to obtain early in the design cycle.
- To run significant amounts of real time software, simulator runtimes become very long - not allowing very significant coverage. Behavioral or RTL processor models must run on the simulator host - very

slow runtimes. Gate level models can run on accelerators, which significantly improves runtimes, but not necessarily enough to do a lot of useful software debugging on the processor model.

- Software coding must be near completion to be useful in the model; this is very late in the design cycle, when the hardware and software design is nearly done. This may be too late; at this point the successful simulation model may only give a measure of confidence to the designer and not give early indications of problems. Prototype or test code may be used in place of actual software, at least verifying the HW/SW interfaces.

The advantage of doing this type of simulation is that the actual software is tested against the actual hardware design, and often this can be done prior to committing LSI to mask fabrication. This can be very useful if the timing of the program allows this schedule flexibility. In lieu of this modeling technique, modeling the software behavior of the microprocessor plus software is the next possible option (as discussed in the next section).

Modeling Software with VHDL Entities and Physical Interfaces

Developing a behavioral model of the software plus the microprocessor is a possible solution to the gate level modeling problems. Software can be modeled in a similar fashion to hardware, that is modules of software become entities, with the method of shared information being signals instead of variables in subroutine calls. This causes several problems that must be resolved prior to the modeling activity:

- Entities force a multi-tasking approach for the software modules, not necessarily reflective of a real software architecture. The real software architecture may be sequential in nature (due to a lack of multi-tasking language support or by choice), and the concurrent processes in VHDL may lead to a false impressions that a multi-tasking software approach can be easily implemented in a sequential software design. Sequence can be forced between entities, but it is not natural to the concurrent nature of VHDL.
- Subroutine calls are not easily supported between entities - all I/O must be done through signals between subroutines or processes, and subroutines in separate entities cannot share global or semi-local variables.
- A lack of variant record support is very problematic, where most high level languages solve this cleanly.

Techniques to overcome such problems are possible. One can structure concurrent processes so that sequential execution of processes can still be enforced. Signals can be used in lieu of variables for subroutine calls, but the number of signals becomes significant - one must be careful about how subroutines and "signa-variables" are chosen. An single "atomic" record structure can be chosen, with routines to map the generic atomic record to more specific records. For more details see the references. In addition, a hardware interface driver must be supplied to the software behavioral model to make the model act as the real microprocessor. This hardware driver will likely implement only a subset of the microprocessor I/O functionality, making the model look like a processor sitting on the processor bus. This driver may be very complex, depending on the amount of low level device control required by the model. Therefore, this needs to be carefully explored prior to full commitment to this type of model. In general, if straightforward memory maps are used by the "processor", and reasonable numbers of hardware devices are to be interfaced, this technique may make sense for simulation.

This approach can be thought of as a software "processor socket replacement" model, and may be the most useful VHDL software model for medium to low complexity designs. This allows for the development and simulation of hardware with real software functionality; RTL or gate level VHDL models of hardware can be run with the "processor socket replacement" software model to ensure correct HW/SW partitioning. This type of modeling can solidify software requirements, and provide validity to a proposed hardware

interface. These software models can be carried on into the detailed design phase, adding additional detail to the models, but the usefulness of the model to HW/SW partitioning and definition should be questioned; will continued modeling provide useful returns? Where do you draw the line? As before, this type of modeling must be balanced between the HW/SW interface design risks, cost, schedule, and customer needs.

LSI/ASIC Modeling Detailed Design — Developing RTL Models

Developing RTL models of hardware provide productivity enhancements as well as flexibility to retarget the essence of the design to other technologies and structural representations. Many Government contracts now require both behavioral (RTL) and structural (gate) level models of ASICs and custom LSI. Some of the benefits of modeling hardware in VHDL at a RTL level are the following;

- RTL models can be treated as a detailed specification for the LSI or ASIC.
- RTL can be used to develop detailed test vectors for the real implementation.
- RTL VHDL can be synthesized to gate or structural levels, which provides the ability to retarget or optimize the implementation for speed, power, space, and new technologies.

RTL designs can be modularly developed, forming a library of complex synthesizable elements that can be re-used for other developments.

RTL and Behavioral "Mixed" Models

The mixing of RTL, gate level, and behavioral "socket replacement" software models verify hardware functionality and interfaces, and to a degree, design correctness. The partitioning of HW/SW can be further validated, providing additional confidence in the design before the LSI or ASICs are sent to mask fabrication. Normally, LSI is only validated by either localized chip level test vectors or by building full functionality breadboards. The mixed mode simulations in effect replace breadboard level testing.

This breadboard replacement by simulation is only valid when a significant amount of the functionality of the software is simulated with the hardware, as discussed in the previous sections. Otherwise, the H/S boundary, split of functionality, and timing cannot be assured. In addition, simulation runtimes should be sufficiently long to obtain a high degree of confidence. This implies that large amounts of real time execution (not simulation time) should be performed to exercise the expected functionality as well as a reasonable amount of exception handling and tests.

Synthesis

The VHDL RTL models can also be synthesized, providing multiple benefits;

- Productivity improvement, by reducing the amount of time required for detailed logic design.
- Re-target designs to new technologies, cell families, and implementations.
- Optimize designs for various criteria - speed, size, power.
- Create re-useable modular VHDL elements, in effect a VHDL library for new system developments.

Problems with synthesis seem to revolve around the problem of not really knowing in detail what logic was synthesized. This impacts breadboard level debug, simulation debug, fault grade, and ASIC/LSI trouble shooting. Full custom designs can be handcrafted to obtain optimal size, power, and speed, but typically

this takes significant effort to achieve, and the advent of low power CMOS and shrinking geometries in LSI put the focus on performance/speed.

Gate or structural VHDL can be validated against the RTL VHDL by simulating a good set of tests against both representations. This forces the test vectors, sets, and testbenches to be well defined at the RTL level before being able to verify functionality at the gate or structural level. It is critical that the test information developed at the RTL level be of sufficient detail to provide good coverage over the structural or gate level model/design.

Problems with Design Changes and Back Annotation

As gate/structural implementations are modeled and simulated, design changes will become necessary as the design evolves (as expected). The problem is whether to put those changes directly into the gate level implementation, or whether to insert those changes at higher levels of abstraction, RTL, or behavioral, and then resynthesize to obtain the 'new' structural design. One may take several positions on corrective actions;

- 1) Changing the structural without changing the higher level RTL model causes a divergence of the two designs. This is in affect abandoning the RTL model. Side effects;
 - One cannot go back and resynthesize from the outdated RTL without losing the new updated detailed design information.
 - This goes against some contractual requirements that require RTL level models of the design.
- 2) Modify both the RTL and structural separately, maintaining design changes at both levels of representation. Side effects;
 - Functionality of two models can easily diverge if changes are not carefully made to both models.
- 3) Modify only RTL and resynthesize only the changed structural representations. Problems;
 - A knowledge base, detailed timing, and other analysis based on the previous structural representation is lost.
 - Time to resynthesize could be significant, although if the design is strongly hierarchical, only the changed components or blocks would have to be resynthesized.

A balance of the use of the three corrective actions is likely to be the best approach, using case 1 late in the design cycle and in cases where RTL is not a deliverable requirement, case 2 late in the design cycle where RTL is a deliverable requirement, and case 3 early in the design cycle where little time is spent in more detailed timing analysis and LSI/ASCI layouts. Back annotating design changes at the gate/structural level into higher levels of abstraction is not well thought out yet, may not be achievable, and at best is still problematic in the synthesis world.

Final Electrical - Acceptance Test — Fabrication / Qualification Phase

Final integration of hardware and software is the "proof of the pudding". VHDL system simulation models play little or no role now, with the major contribution being that the same test information (in the form of scenarios and messages) used to drive the various levels of VHDL system simulation are now also used to drive the integration and acceptance test activities. The test procedures are actually "debugged" prior to this phase, as well as the design concepts. If the VHDL simulations have been carefully and meticulously maintained during the design processes, very few show stoppers should occur in the integration phase.

Specification / Test Procedure Verification — All Design Phases

In order to develop a tight coupling between the written specification and the VHDL model, a process of extracting test information from the specification, and transferring that data into the VHDL simulation is important. Therefore, a toolset can be used to extract scenario and message definition data from top level specifications, and use this information to develop detailed test procedures. These semi-automated test procedures are then used directly for VHDL simulations (loaded into VHDL testbenches), in real subsystem testers, and in system level testers. Figure 3 shows a context diagram, demonstrating how a test procedure compiler is used to obtain test information for both VHDL simulation and electrical testers.

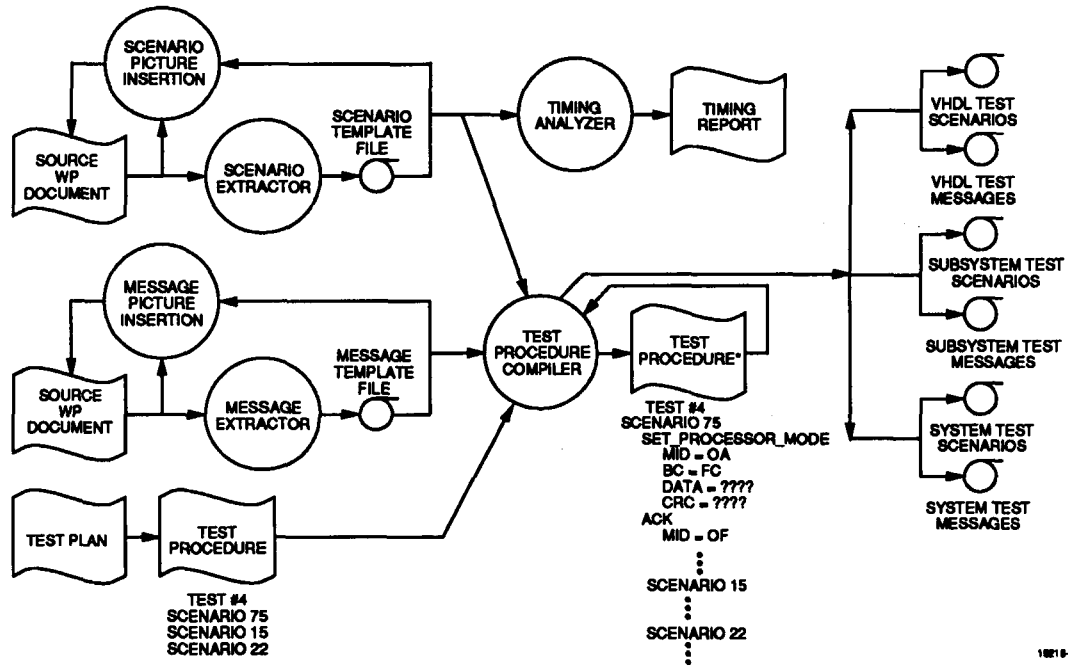


Figure 3. Scenario and Message Extraction Test File Generation

Scenario and Message data is extracted in the form of templates, which can then be used to form test sequences. Test Procedures reference sets of scenarios that can be combined to form the actual test, and scenarios reference sets of messages that form the actual scenarios. The test procedure compiler expands the tests, scenarios, and messages, down to levels that can be used to drive simulations or electrical testers. In some cases, the expansions may go from to data enumerated in record formats, real data in record formats, or finally down to the actual bytes and physical I/O found in real message transactions. Using test information developed in this fashion, the specifications become the "golden standard" for message and scenario definitions, and the test procedures developed for simulation become the same test procedures used for final subsystem and system level test. This linkage minimizes human error and ambiguity in the specification, forcing the specification to be detailed enough to support testing, and providing a mechanism for validating the test procedures early in the design cycle.

The tests developed using scenario references can be "broken" in order to develop tests for exception handling. Each time the test procedure compiler is run, the messages and scenarios in the test procedure are reverified against the message and scenario templates, and if the message or scenario is to be "broken" to test exception handling, the resulting error messages can be overridden. This allows flexibility in the development of test procedures, but still enforces accuracy in the procedures by revalidation of messages and unchanged scenarios by the test compiler.

If message or scenario definitions are updated/modified in the source level specifications, the changes are immediately rolled into the test procedures by a simple recompiling of the original test procedures. This maintains strong configuration control between the specifications, simulation activities, and testing activities. A high percentage of integration problems are due to poorly structured integration tests; VHDL models that are simulated using the same test information as used in integration will uncover many design and documentation problems before actual H/S is involved.

Summary

VHDL can be used for system simulation at levels of pure behavioral, RTL, and mixed levels of representation (gates, RTL, and behavioral). Test procedures can be generated and tested with VHDL simulation early in the design cycle, and these same test procedures can then be used in real electrical design tests. Software can be modeled in VHDL, but one must be careful as to not get caught up in the modeling processes, and make sure the return on investment is viable and worthwhile. Microprocessor "Socket Replacement" models can be more readily used in full system simulation. RTL level VHDL can be used for synthesis providing many benefits, but one must meticulously manage changes inserted into both the gate and RTL models. The use of VHDL in the design cycle directly reduces program risk, enhances productivity, creates concise definitions of functionality, and provides a method of validating test information prior to costly integration activities.