

Behavioral Testbenches for Telecommunications Chipset Development

E. Parrella and M. Tota
TranSwitch Corporation
8 Progress Drive
Shelton, CT 06484

Abstract

A chipset has been developed for Synchronous Optical Network [SONET] applications. The chipset consists of a four channel T1 framer, a seven channel T1/SONET mapper and a SONET overhead termination device. One of the key features of the VHDL based development methodology used is the behavioral VHDL testbench. The testbench enabled the team to check and re-check the design for compliance to telecommunications standards throughout the development cycle. The testbench cut design time by eliminating the need for vector creation. Furthermore, through the use of a mixed-level gate/VHDL simulation environment, regression testing of the chips at the behavioral, RTL and gate-level phases was made possible. This paper will describe the general development methodology used on the chipset, and specific aspects of the testbenches used.

Design Environment

TranSwitch recently developed a next generation chipset for Synchronous Optical Network [SONET] applications. This chipset was developed in cooperation with a key customer, who provided engineers to assist in both the specification and design of the chips. Customer engineers were largely new to telecom design techniques and VHDL synthesis. TranSwitch's

engineers had a telecom and synthesis experience base; however, there was minimal experience with VHDL synthesis.

An important goal was to ensure functionality of each of the devices, as well as the entire chipset, in the customer's system. However, these chips are not just ASICs - they are required to not only meet the specifics for our customer's system, but must be general and flexible enough to behave as TranSwitch standard products. The telecom design environment is in a constant state of flux, with new standards and requirements emerging constantly. Because of this, a highly flexible methodology was required for both the design and architecture of these chips. The design process had to be responsive to specification change; the silicon had to be easily reprogrammed to perform in line with expected standards changes or differing customer requirements. Both requirements indicated a large verification effort that was beyond the reach of any designer's capability using traditional simulation methods. A robust, configurable behavioral testbench was required.

Project Description

Four major functions were implemented in three chips; the chips ranged in complexity from 40K to 160K gates. The system configuration is

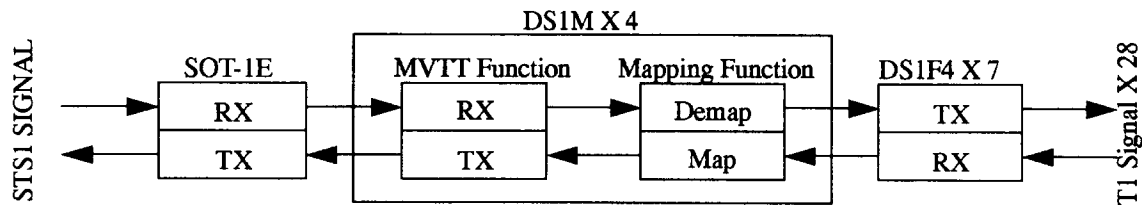


Figure 1. System Configuration

shown in Figure 1.

The T1 framer is a four channel device which can be connected to four T1 lines. Each line carries independent receive and transmit signals, which are known as DS1s. Each DS1 transmits at a 1.544MBit per second rate, which allows the T1 to carry 24 8-bit channels, plus one bit of overhead, in a 125 microsecond frame. These 8 bit channels, which are suitable for voice and data transmission, are known as DS0s. Each DS0 can support a single telephone call. In order to locate the DS0s for demultiplexing, the overhead bit, also known as the frame bit, must be located in the serial stream. This is achieved by synchronizing to the fixed frame pattern this bit position takes on.

Once frame is established, DS1s can be directly mapped into the SONET byte-synchronous T1 mapping. This task is performed by the T1/SONET mapper chip; the mapper takes seven DS1 signals and maps them into the SONET VT1.5 structure. Since 28 VTs map into the SONET STS-1 signal, four mappers are required for a complete SONET interface. In the demap direction, a digital phase locked loop is incorporated to reduce jitter on the tributary (DS1) output. To further reduce the effect of SONET pointer movements on jitter, all pointer movements are absorbed by the pointer leak buffer

and leaked into the digital phased locked loop one bit at a time over an extend period of time. In the map direction, a threshold modulator is employed to reduce the DS1 low frequency jitter carried in the VT by controlling the SONET stuffing mechanism. The mapper also incorporates a Multiple Virtual Tributary Termination (MVTT) function, which was also supplied as a soft cell to the customer; the MVTT supplies pointer and overhead processing for SONET VT1.5.

The Enhanced SONET Overhead Terminator for STS-1 (SOT-1E) provides an enhanced feature set of an existing TranSwitch product. In the receive direction, the SOT-1E will accept a 51 MBit STS-1 signal and extract all Transport and/or Path overhead bytes from the SONET payload while examining them for alarm and error conditions. The payload is also re-timed to the local system timing. In the transmit direction, the SOT-1E will reconstruct a valid 51 MBit STS-1 signal. Alarms are sent either through CPU control or can be automatically triggered by the receive path.

Development Methodology

TranSwitch's design flow is shown in Figure 2. It is the classic top-down design methodology which has been discussed in numerous previous

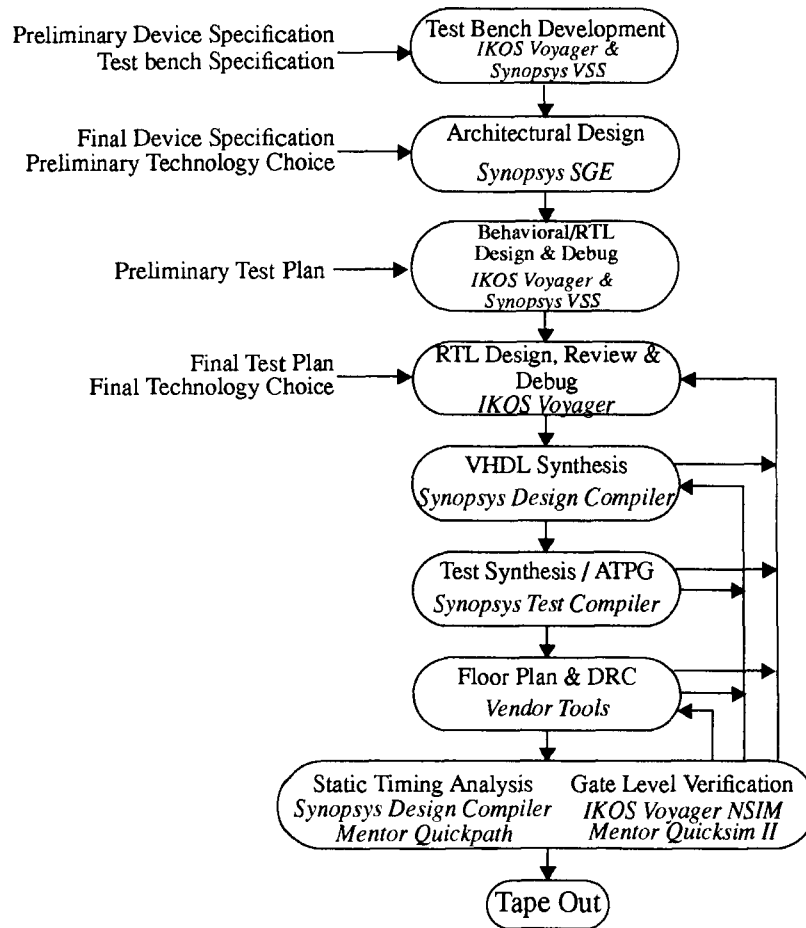


Figure 2: Design Flow

papers and vendor marketing literature, so we will not reprint a detailed discussion of it here. However, some key observations can be made.

- Architectural design phase cannot be ignored. With a synthesis methodology, the designer's value-added is no longer at the gate-crunching or polygon-pushing level. Instead, the designer should emphasize performance/cost improvement through architectural and/or algorithmic trade-offs.

- Behavioral models perform few useful functions for designers; this is mainly due to the fact

that so much of VHDL is synthesizable. We prefer moving directly into RTL except where a simpler model might prove useful for a detailed trade-off analysis (for example, in algorithm performance analysis).

- VHDL should be largely self-documenting; however, like any language it can be abused. Code should be kept as clear and as readable as possible, with descriptive signal, variable and process names. Comments should explain what the code is doing and why; calling out specification paragraphs and standards can be useful. Software-style coding reviews can reveal many

problems which may require long simulations to uncover.

In conclusion, behavioral testbench techniques are a must for a reasonable development schedule.

Advantages of the Behavioral Testbench

Self-checking behavioral testbenches provide a more advanced form of chip verification than static test-vector based analysis. Here we will discuss some of the features which result in a superior design verification process.

Behavioral test benches are more dynamic than vectors. A path of actions can be automatically triggered based upon an erroneous or unexpected result. For instance, an interrupt line can trigger a read of a status register. If the status register reports an expected alarm or exception condition, the test bench can move on the next test. If it reports additional problems, the test bench can be triggered to query other registers - the results of this inquiry may be reported to the user through assertion statements. This speeds debug and aids in regression testing - secondary runs of a test (after design changes) should result in no difference in the assertion messages put out by the simulation.

Tests using test benches can be written more quickly than test vectors. Re-use of existing code and procedures between test suites is commonplace; functions and constants can be placed in a package visible to the test suite. Furthermore, many cycles of testing can be accomplished through the use of the *wait...until* and the *for...loop* constructs. One time-saver providing code re-use is the VHDL procedure. A single procedure can be applied to many different situations. For example, a procedure *TestMemory* can take as arguments a start address and a stop address; there is also an argument *test_type* which is an enumerated type. The enumerated type has elements *checkerboard*, *inverse_checkerboard*, *counting*, *pseudorandom*, *all_zeros* and *all_ones*. Depending upon the test type selected, the memory can be tested with various different

data generation schemes. It is fairly easy to see how testing multiple RAM devices on an ASIC would be greatly simplified with such an approach.

Test bench development can be decoupled from chip development. This allows test benches to be separately validated and certified, and, where useful, placed in a library to be used for multiple projects. With a careful verification procedure, a testbench can be shown to be compliant to a specific standard (for instance, DS1 or SONENT) in much the same way that benchtop test equipment is certified. This approach allows the verification effort to start right away. It also provides a company-wide sanity check on the standards - two designers would be hard pressed to produce different levels of compliance to a standard if they both used the same test bench.

With a test bench, high level specification of verification is possible, provided the features required by tests specified are available in the test bench. In fact, if test benches have similar capabilities to benchtop equipment, qualification and validation tests which will be later applied to the system may be tried earlier. Systems engineers with little or no knowledge of ASIC or VHDL development can come up with a complete verification specification based upon knowledge of the test bench's functions and configurability. This is a powerful technique because it shifts the responsibility for the test plan to those who have the best knowledge of how the chip will be used and how the standards are met.

Test benches are even useful in creating test vectors for ASIC sign-off and production test; these are critical when a "golden" simulator must be run which does not accept VHDL models. A process or component may be instantiated in the top level test suite. This block sits between the component under test and the test generating code or components. The capability to synchronize the test bench to a master clock must be present; the master clock is used to sample all of the inputs, bidirectionals and bidirectional con-

trol signals. The process can simply dump all the inputs and bidirectionals to a file, or it can do more complex processing. In our case, Mentor force file formats were required; we produced these directly out of the VHDL test suite. Furthermore, because disk space is an omnipresent issue, on-the-fly compression by recognition of same input state vectors was performed.

Disadvantages of the Behavioral Testbench

Overall we were pleased with the behavioral testbench methodology. However, there were a few lessons learned which bear listing here.

The first stumbling block was simulation speed; a behavioral testbench takes longer to simulate than raw vectors. We overcame this by bringing on a more powerful VHDL simulator (IKOS) which gave us improved raw speed. Another problem has been in specifying too limited a set of functionality; the SONET testbench, for instance, was extremely useful on DS1M verification, but required changes for the SOT-1E chip. Furthermore, our methodology (see Figure 2) specifies test bench development prior to test plan development. This is to give systems engineers a clear picture of what the test bench can do. Unfortunately, tests are sometimes specified which require changes to the testbench. This illustrates that a mature testbench is desirable (just as mature software is preferable). Finally, vector extraction requires synchronization of the test bench; this feature is readily done in the early phase of test bench design, but is more involved when it must be added long after the test bench is completed.

Architecture of the Testbench

The testbench can be structurally decomposed into analyzer (A) blocks and generator (G) blocks. Generators create valid telecom signals, which consist of single rail (NRZ) or dual rail data, plus clock and in some cases synchronization signals. Figure 3 illustrates three configurations of the test bench showing its various components.

The first component, the DS1 generator (DS1G), creates a valid T1 signal. The data within the signal could be sourced from a user specified pattern or pseudo random generator. Within the T1 is a Facility Data Link (FDL) channel which transmits High Level Data Link Control (HDLC) messages, which report on the health of the T1 link. The output of DS1G can be configured to feed into the DS1 mapper or the DS1 Framers devices or it can be used as a data source for the SONET Generator.

The SONET generator (SONG) creates a valid SONET signal. The output of SONG can be either a 51 MHz serial data stream or a parallel data stream of 6 or 19 MHz. The data within the data stream can be sourced from either a pseudo random generator or a built in VT structure generator. When using the VT structure, each of the 28 VTs can independently source data from a fixed pattern generator or from a DS1G. The output of SONG can be configured for the SOT-1, MVTT, or the DS1 mapper.

The SONET Analyzer (SONA) analyzes a SONET signal for validity. The input to SONA can be either a 51 MHz serial data stream or a parallel data stream of 6 or 19 MHz. The data within the SONET signal can be analyzed by a pseudo random analyzer or can be passed into a built in VT structure. Once the data is passed into the VT structure, each VT can independently pass data to the DS1 Analyzer. The input to SONA can be configured for the SOT-1, MVTT, or the DS1 mapper.

The DS1 Analyzer (DS1A) analyzes a T1 signal for validity. Any data configuration that can be specified within the DS1A can be analyzed in the DS1G. HDLC messages are demultiplexed off the FDL channel and buffered. The input to DS1A can be configured for the DS1 mapper or the DS1 Framers devices or the SONET analyzer.

A CPU model was also created to perform memory cycles with the device under test and to monitor the interrupt pin. The CPU model provides a convenient method of device configuration and status monitoring.

Because the VHDL test bench is so flexible, many

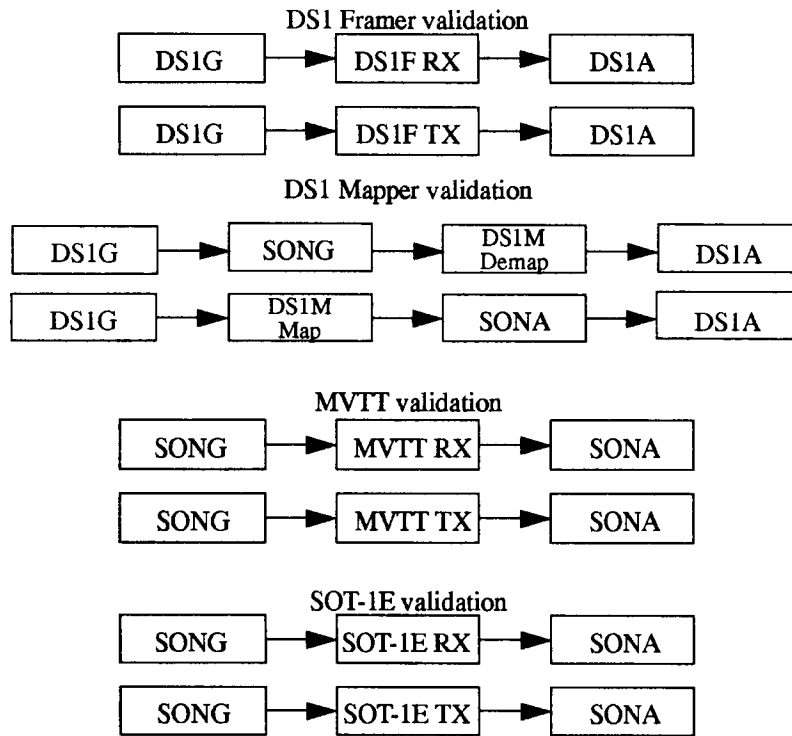


Figure 3: Test Bench Configurations

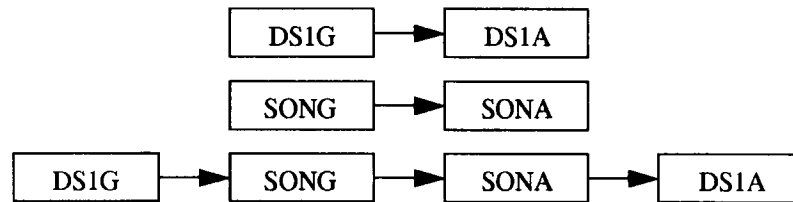


Figure 4: Test Bench Validation Configuration

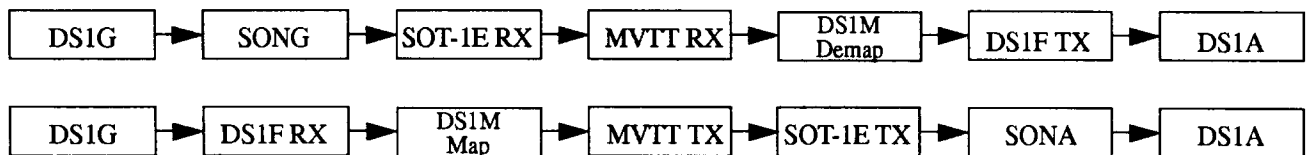


Figure 5: System Test Configuration

different configurations are possible. Configurations without a device under test, as shown in Figure 4, are used to facilitate test bench validation. Figure 5 illustrates a system test configuration. This was little used because we were successful in decoupling chip development efforts through the use of the individual test-bench configurations (Figure 3).

VHDL Structure of the Testbench

There are several ways in which testbenches may be structured. A simple testbench might be implemented in a single entity-architecture pair, with a few processes stimulating a single component. As Figures 3-5 show, we broke up each individual test signal into an analyzer and generator; this naturally broke up into multiple VHDL components instantiated structurally. Because each signal consists of multiple signals multiplexed together, the internal structure of both the generator and analyzer is also a structural instantiation of sub-rate generators and analyzers.

At the top level, device under test, generators, analyzers and CPU model must be hooked up to form the test environment. This is most easily done schematically. We have one more level above this however, which instantiates the schematic with DUT + testbench, and controls the entire testbench with one large sequential process. This process is the test program or test suite. It triggers a series of tests and communicates directly with the user through assertion statements.

A trade-off decision in test bench development was the decision of how to implement a control interface to the test bench. Several possibilities existed:

- Implement a command style interface. Parse commands which are placed in a file. Advantage is that there is no requirement to re-analyze test program. Disadvantage is that this is more complex and adding commands might be difficult.

- Pass generics to the test bench. This sets up the test bench quite easily but may make it difficult to set up for each individual test. It is also not as straightforward to expand the set of generics.

- Direct signal control. This is the simplest to implement and easiest to control from the test suite. Adding modes, however, implies adding ports to each test bench component. Since we used a schematic tool to generate the structural instantiation of the testbench, we wanted to avoid having to change the ports on each symbol. Furthermore, any changes to top level inputs would necessitate changing every test suite instantiating the top level test environment.

- Global signal control. This has all the advantages of direct signal control, but is simple to modify. This method was selected.

We implemented the global signal control as follows. Each component of the testbench has a package associated with it. This package has all the control signals associated with the component defined in it. The test suite has visibility into all components' packages, as does the testbench component's behavioral description. This implements a direct signal connection from the test suite into the test bench and allows dynamic control of the test bench, and on-the-fly monitoring of analyzer outputs.

Conclusions

Behavioral testbenches are a powerful technique for insuring specification and standards compliance of complex VLSI chips. The capability to thoroughly check out a design, particularly a highly configurable architecture, is greatly enhanced. Test vector generation for functional verification is no longer required, and production vectors are readily generated from the test bench. Because the testbench decouples chip development from system development, more design can go on in parallel and system simulation is less critical.