Design Advantage™

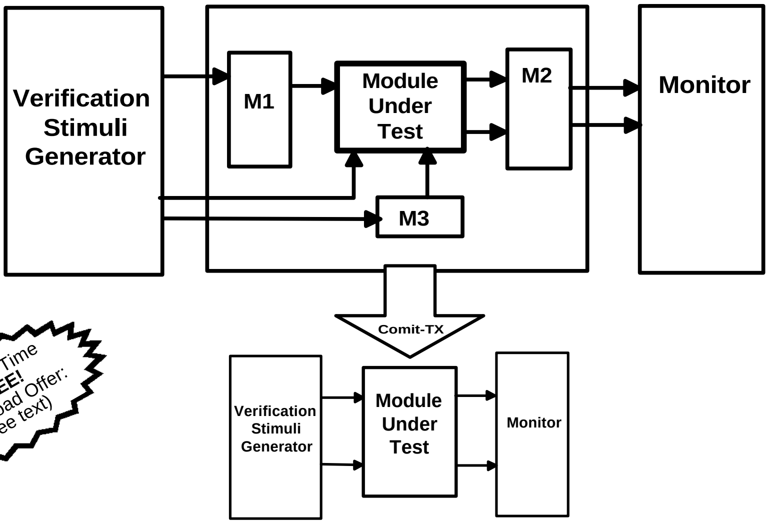
Volume 1, Number 3 The newsletter of Comit Systems, Inc. November 1998

Save Time, Maintain Design Integrity with Innovative Test-Bench Extractor

Venkat Talapaneni

Comit-TX, is an innovative Verilog Test Bench Extractor that speeds up the design process by saving time in module level verification. For a limited time, it is available free of charge through Comit's website.

With increasingly large and complex designs, different modules tend to be represented at different levels of abstraction in the initial phases of the design. They would later be replaced by RTL or structural representation.



Even then, the modules continue to be modified for a variety of reasons, including area or timing optimizations. While modifying the different modules and fitting them back into the full design, it is critical to ensure that the module's functionality and its interface with other modules remain unchanged.

Verifying the different modules in a design by simulating the entire system is time consuming.

Comit-TX extracts a self-checking Verilog testbench of any module inside a design that has a system level testbench. Comit-TX, with the extracted testbench, enables the module's replacement to be verified in a stand-alone basis in an environment identical to its final working environment, without having to simulate the entire system.

During simulation, the extracted testbench applies vectors on the input signals of the module monitors the output signals for expected behavior. As the testbench is self-checking, the verification is automatically done during simulation, and a report is produced indicating any mismatch between the expected and observed behavior of the module.

Save time. Maintain Design Integrity. Verify your designs with Comit-TX. Download FREE for a limited time from www.comit.com■

Clean, Consistent Timing Modeling In VHDL 93

Vijay Nebhrajani

Do you write models for components which need timing information and want a clean, consistent way to do it? Here is a simple and easy method to model the timing specifications.

Let us say that the timings to be modeled are in the form of a table given below. Of course, any number of timings are possible. We model only two here, for the purpose of illustration.

Parameter	Name	Min	Typ	Max
t_AVWH	Address setup	2.0	3.0	4.0
t_WHAX	Address hold	1.0	1.5	2.0

The first step is to declare a type `string_ptr` which is the equivalent of a `char *` in C. This is a pointer that points to a string, thereby allowing the use of strings that are not constrained to a specific array length.

Secondly, we declare a `timing_value_t` type which is an enumerated type that lists out the three type of timings we have to model – min, typ and max.

A type called `time_array_t` is defined as an array of `time` in such a way that we can access the elements of this array with an index of type `timing_value_t`.

After this we declare a record that stores one line of the timing table. We need a fixed length array that stores the six characters of the parameter string, a string pointer that points to the name string, and a time array capable of holding the three values for that parameter.

Lastly, we need a table that stores exactly two such records. This is declared as an array of `timing_record_t`. The actual code would look like the segment

Design Advantage™

The newsletter of:

COMIT®
SYSTEMS

Comit Systems, Inc.
3375 Scott Blvd. Suite 330
Santa Clara, CA 95054

Editor : Rajesh Bawankule.
We welcome your comments and
suggestions. Please email us at
rajesh@comit.com

below:

```
-----------------------------------
-- SAMPLE TIMING MODEL CODE
-----------------------------------
type string_ptr is access string;
type timing_value_t is (minumum,
                        typical,
                        maximum);
type time_array_t is
   array(timing_value_t) of time;
type timing_record_t is record
  parameter    : string(1 to 6);
  name         : string_ptr;
  val_table    : time_array_t;
end record;
type timing_table_t is
  array (0 to 7) of
  timing_record_t;

shared variable tim_table :
  timing_table_t := (
  -------------------------------
  Parameter
  Name
  Min         Typ       Max
  -------------------------------
  "t_AVWH",
  new string'("Address setup"),
  (2000 ps, 3000 ps, 4000 ps)),
    "t_WHAX",
  new string'("Address hold"),
  (1000 ps, 1500 ps, 2000 ps)),
  --------------------------------
-
);
```

The `new` operator is used to create a string which is exactly the same as the name. This table needs to be a shared variable, because VHDL 93 will not allow access types (pointers) as constants and, besides, it will probably be needed by several processes. Care has to be taken that the table is not inadvertently modified, since there is nothing preventing that from happening. Finally, we alias the timings in the following way:

```
alias t_AVWH : time is
tim_table(0).val_table(typical);
alias t_WHAX : time is
tim_table(1).val_table(typical);
```

In place of the "`typical`" we could have "maximum" or, for that matter, a constant of type `timing_value_t` initialized to one of `minimum`, `typical` or `maximum`.

The method described above has other advantages besides code readability, easy maintenance etc. One of them is simple timing error reporting:

```
procedure report_violation
 (index : in integer) is
begin
 report (("Violation :" &
  tim_table(index).name.all)
  & (", " & tim_table
  (index).parameter))
 severity warning;
end report_violation;
```

A coding style such as this lends itself to effective reuse, and even a library of timing check `functions` can be built around it. ■