Board-level component modeling using VITAL

Russell E. Vreeland
TRW Ground Systems Center
One Space Park
Redondo Beach, CA 90278
russ@vhdl.org

Abstract

The VITAL 95 standard for VHDL libraries was written primarily to solve the problem of developing consistent ASIC libraries with VHDL. This paper discusses how a design methodology for board design with board-level simulation models written in VITAL can be advantageous, and outlines lessons-learned at TRW over the past couple of years in implementing board-level component libraries in VITAL/VHDL.

The most notable advantage of developing board-level simulation models in VHDL is the seamless simulation environment when integrating board-level simulations with both behavioral simulations (systems level simulations) and FPGA and ASIC simulations - assuming they are VHDL-based. There is a considerable advantage from eliminating the twin nuisances of supporting and training on multiple simulation tools, and verifying and maintaining the translations of designs and test vectors or test fixtures between those disparate simulation environments.

In order to arrive at the goal of a seamless simulation environment, some technical issues had to be addressed and solved. Working primarily with high-speed boards, we had to apply VITAL modeling to families of ECL and GaAs components which have some unique modeling considerations - such as differential signals and intermediate logic level (not 0 or 1) inputs. We sought to keep our "schemato-centric" approach to design - since the schematic is the database from which the product (printed circuit board) is designed, and most board design engineers are more comfortable with schematics than with large structural VHDL netlists. We therefore had to integrate our VITAL models with the hierarchical and structured design methodologies of our Cadence and Mentor schematic capture tools.

A long term problem in maintaining component libraries is their sheer size. Total component counts in the tens of thousands of components are not uncommon for a typical design development organization. Since many components use the same simulation model except for the timing data, we sought to reduce the component count by reusing simulation models, separating out the timing completely by utilizing the VITAL standard's SDF backannotation capabilities. We've achieved a working method for technology independent libraries using SGML-based timing files which are parsed by a software program which writes out an SDF file for the design according to the timing selections made by the designer.

Other board-level modeling topics discussed in this paper include: the viability of writing large MSI models to strict VITAL level 1 compliance, and modeling passive components in VHDL (especially bi-directional resistor modeling).

Introduction

In late 1994, I, along with colleagues at work embarked on this adventure to transition board-level component libraries and the associated simulation environment to VITAL/VHDL. Most existing models were written for a proprietary simulator with dubious future support and viability. We were not able to leverage very much model creation work from one design center to another - each vendor had proprietary formats which were used to create models, and those formats were not portable. There was a lot of extra work and trouble debugging the interfaces between the various tools and performing the format translations necessary to go between the different simulators in the stages of the same design.

Also, we had this "vision thing" of a seamless simulation environment where system engineers do behavioral, conceptual analyses and generate simulatable specifications, design engineers plug their increasingly complex RTL and (later) gate-level representations of the same design into the same test benches the system engineers used, and at the end, the whole thing goes "on-the-shelf" and becomes a reusable archive for the next generation design that interfaces or extends the one completed. But we stopped ourselves just in time from launching into a

chorus of “I’d like to teach the world to sing …” and set out to tackle the achievable: produce a board simulation environment entirely in VHDL.

With the advent and maturing of the VITAL standard contemporaneously with our efforts, we utilized VITAL as the simulation model standard for our board-level component models. In so doing we integrated VITAL models into a hierarchical, schematic capture environment doing mainly high-speed design with some FPGA and ASIC work.

As a result of the board-level component modeling work begun at TRW, the Free Model Foundation was founded by 4 (at the time) TRW employees, including the author, with the intention of promulgating and furthering the availability of board-level component models. All the models referred to in this paper are available on the web at

`http://vhdl.org/vi/fmf`

The web site exists as a repository of public domain simulation models. Feel free to donate. TRW and Intel already have. Don’t be left behind.

Single-bit models

Board level components have more complicated relationships between the model and its schematic symbol(s) than ASIC gate-level cells, and have a physical IC package that they represent which ASIC gate-level cells do not. These complications are revealed in the many sizes and shapes of schematic symbols that are often used to represent functionally identical components, or split-up versions of the same component. We found that favoring the creation of single-bit simulation models for many-sized board components is a good approach for reasons of simplicity and manageability.

By “single-bit”, what is meant is that the VHDL ports of the simulation model are scalar, regardless of the actual width of the data using one or more pins on the IC package. When a physical part has “wide” functionality - i.e., written such that a bus is naturally attached to several pins on the part, and the individual bits are independent of each other (except for a common clock, or reset or other global pins), that symbol is equivalent to a schematic representation using individual single-bit symbols for each bit of the bus with the common pins tied together. A VHDL netlist derived from that schematic remains functionally identical to the schematic if it instantiates many single-bit components rather than one with vectored ports. In the resultant VHDL netlist utilizing this model, the single-bit component would be instantiated as many times as necessary to provide functionality for each bit of the data. It is incumbent on the CAD tool’s VHDL netlister to provide a means for doing this.

This strategy provides several advantages. Since VITAL is very verbose with its timing check procedures and path delay procedures, a vectored model increases greatly in size as it gets wider. A model written for an 8-bit wide register with asynchronous set and clear, with a full suite of timing checks and path delay procedures, would be incredibly large. The file would be imposing to read, maintain, and understand.

A single-bit model is small and easy to read. It is written once and reused by all the larger (vectored) parts or schematic symbols that have the same core functionality, whereas vectored models are reused only with difficulty by larger vectored parts, and by smaller parts if the netlister can be coaxed into wiring many of its ports to “OPEN”.

Often, a board-level component is represented on a schematic by a single section of the whole package. For example, a 7400 NAND gate might be shown on a schematic with the usual NAND gate logic symbol, but the actual IC package contains 6 NAND gates, which might be used on other sheets of the schematic if at all. By writing a single-bit model for a NAND gate, that model matches the logic gate symbol perfectly. The other sections of the 7400 are instantiated separately with no package information; the simulation doesn’t need to have package information. I have some comments later about the efficacy of packaging directly from a VHDL netlist.

Trying Cadence’s (Concept) and Mentor Graphics’ schematic capture tools, we found that both of them could support the single-bit methodology well.[1] In general, we found that using the schematic capture tools was beneficial to capture large structure in general - even inside large FPGAs and ASICs - not just as a legacy methodology for diehard EE designers to use. There probably won’t be and shouldn’t be an “emacs” board design anytime soon.

[1] Cadence Concept keyed off the use of a directive in a vhdllink.cmd set-up file called “VHDL Replicate Entities”, Mentor Graphics Design Architect’s vectored symbols were given logic sheets “underneath” them which contained Frames with the single-bit symbols contained therein. It worked for us; your mileage may vary.

Technology-independent models

Utilizing the VITAL SDF timing backannotation methodology, we set out to write technology-independent models, which is synonymous with saying the timing information is outside the model. In that way, we hoped that we could get by with writing a lot less simulation models (and instead have more manageable common timing files).

The format of the timing file, "FTML", is based on SGML (Standard Generalized Markup Language).[1] . For each reusable simulation model, there is one FTML file but that file may contain scores of entries for different electronic parts that utilize that same simulation model. Consider the 7400 NAND gate again. How many different technologies and packages could utilize the same V7400[2] simulation model but each have slightly different timing characteristics? One FTML file contains timing data for all of them.

Inside the <TIMING> section of the FTML file, the timing data for the model is listed in the form of a chunk of an SDF file's "DELAY" entry. The only difference between FTML syntax and the snippet of SDF within the <TIMING> tags, is we allow the parentheses to be omitted, and the SDF keywords TIMING, DELAY, ABSOLUTE, and TIMINGCHECK may be omitted (mk_sdf automatically adds these where they ought to go).

To use a timing file, a designer sets the value of a "TimingModel" generic, which is of type "string", in the VHDL simulation model to declare which timing model is to be looked up in the FTML file. Using Cadence and Mentor tools, placing "Properties" on schematic symbols prods their respective VHDL netlisters to remap an existing generic that has the same name as the property to the value assigned to the property placed on the schematic symbol. Thus, selecting a particular version of a part is a matter of changing a property on a symbol, or selecting a part with a pre-assigned property value from a catalog or table. If the designer places no property on the symbol, the default value is a string indicating no FTML file lookup should be done, and the simulation will proceed with the default timing values - which are generally unit default delay, usually a unit delay.

```
<!DOCTYPE FTML SYSTEM "ftml.dtd">
<FTML><HEAD><TITLE>FMF Timing for
 STD245</TITLE>
<AUTHOR>R. Munden - FMF</AUTHOR>
<DATE>95 DEC 12</DATE></HEAD>
<BODY>
<TIMESCALE>1ns</TIME SPEC>
<MODEL>STD245
<FMFTIME>
SN54ALS245AJ<SOURCE>TI SDAD001A
 1984</SOURCE>
SN54ALS245AN<SOURCE>TI SDAD001A
 1984</SOURCE>
<COMMENT> No comment </COMMENT>
<TIMING>
(IOPATH A B (30:75:150) (30:65:130))
(IOPATH B A (30:75:150) (30:65:130))
(IOPATH ENeg A (::) (::) (40:90:180) (50:125:250)
 (20:60:120) (50:125:250))
(IOPATH ENeg B (::) (::) (40:90:180) (50:125:250)
 (20:60:120) (50:125:250))
</TIMING></FMFTIME>
</BODY></FTML>
```

Figure 1 Sample FTML file

After the VHDL netlist is written out, generic maps in component instantiations indicating timing models to be used, an SDF file is generated for the overall netlist. We developed an in-house tool written in C "mk_sdf" that parses the VHDL netlist looking for "TimingModel" generics in component generic maps, parses the vendor-specific VHDL library location mapping file(s)[3] , parses the FTML timing file, and writes out an SDF file specific to the VHDL netlist - which is used in subsequent simulations.

The mk_sdf program is command line driven. It would be great to put a GUI on it, and make some enhancements like the ability to write out new VHDL configurations with the timing information rather than SDF files - this would make it useful to simulators that aren't fully VITAL compliant, or have bugs. We would like eventually to enhance mk_sdf, port it to Java, and make it available to the public domain through the Free Model Foundation. It might even make a neat applet running from the FMF server (the vhdl.org machine) -

[1] The exact format specification of a derivative of SGML is given by a DTD (Documented Type Description) The FTML DTD is available at: **`ftp://vhdl.org/vi/fmf/fmf_public_models/tools/ftml.dtd`**

[2] Many CAD tools have problems with model names that are not legal VHDL entity names - hence the "V" added to the part name

[3] For Cadence: it is the **`cds.lib`** file; for Mentor: the **`quickvhdl.ini`** file; for Synopsys: the **`.synopsys_vss.setup`** file. For all, there is a hierarchy of files to parse: generally, first the default system installation file, then the home directory file, then the working directory file.

if the Java applet security problems can be overcome. (This is my idle speculation - my colleagues at FMF are probably grimacing now as they read this).

A special case: high speed ECL components with differential I/O

The modeling of ECL and GaAs components introduces some special problems. First, these components often have differential I/O, which means that the logic state of the model is dependent on pairs of inputs. Either the component is modeled with a very complex VITAL-compliant functional description - probably a complex State or Truth table, or the component is broken down into a multi-process description - an input process or processes which convert the state of the differential I/O into a simple non-differential or "single-ended" equivalent, and a "normal" VITAL behavioral process to describe the simplified equivalent. We discovered that models were much easier to construct, less bug-prone, smaller, and more readable using a multi-process methodology.

Differential inputs can be divided into 2 cases: inputs whose values are important only when a clock edge occurs (and during the setup and hold times before and after the clock), and inputs whose values should always be considered carefully. The D and Dneg input pair of a differential D-flip/flop illustrate the first case. The value of the "single-ended equivalent" *Dint* of the D input pair is relevant to the model only during the setup/hold interval around the active clock edge. Whenever the D and Dneg inputs transition ideally (like: D: 1-to-0 and Dneg: 0-to-1 simultaneously), it is easy to calculate Dint (goes 1-to-0). However, when there is skew between the input pairs' transition times, the value 'X' can be imputed to Dint in that time interval since Dint is only important at a clock edge and if a clock edge occurs during a skew interval, the value of Dint is unknown - at least that would be the conservative assumption to make. The function of converting the D input pairs to Dint is accomplished in a table lookup.

The other case can be illustrated by the clock input pairs Clk and ClkNeg of the differential D-flip/flop. Here, skew cannot be ignored so easily; a way must be found to propagate the edge transition to the internal, single-ended equivalent signal *Clkint*. The following VitalStateTable calculates the edge transition based on a "last edge" algorithm - the last differential pair to change causes the internal signal to transition. Of course, if the skew lasts years rather than picoseconds, something is wrong that ought to be detected, but VITAL only provides a generic placeholder *tskew* for a (hopefully) future VitalSkewCheck procedure. One could do something jury-rigged with the existing timing procedures, or add something non-VITAL, but we've decided to not check skew at this time - and there is not a lot of data available from vendors on just what numbers to use for maximum tolerable skew (and high-speed designers are like economists: the number of opinions exceeds the number of experts).

In all cases of modeling differential inputs, the logic level "VBB" unique to ECL and GaAs technologies is found. When one of the differential input pins is pulled to VBB (usually about halfway between logic 0 and 1 - in the case of ECL that is halfway between -0.8 and -1.8 volts or about -1.3 V) the other half of the input pair becomes the controlling pin for the input. If VBB is hooked to the negative input pin, the positive pin becomes a single-ended input pin, if VBB is hooked to

```
-------------------------------------------------------------
-- Table for determining value of a non-clk differential input pair.
-- This table indicates 'X' whenever diff. inputs are the same, which
-- is the most conservative approach.
-- ECLVbbValue should not be '0', 'L', '1', or 'H' (duh)
-------------------------------------------------------------
    CONSTANT ECL_s_or_d_inputs_tab : eclstdlogic_table := (
      '0'   => ('1' => '0', 'H' => '0', ECLVbbValue => '0', OTHERS => 'X'),
      'L'   => ('1' => '0', 'H' => '0', ECLVbbValue => '0', OTHERS => 'X'),
      '1'   => ('0' => '1', 'L' => '1', ECLVbbValue => '1', OTHERS => 'X'),
      'H'   => ('0' => '1', 'L' => '1', ECLVbbValue => '1', OTHERS => 'X'),
      ECLVbbValue => ('0' => '1', 'L' => '1', '1' => '0', 'H' => '0',
                      OTHERS => 'X'),
      OTHERS => (OTHERS => 'X')
```

Figure 2 Table lookup for converting non-sensitive (data) diff. inputs

```
------------------------------------------------------------
--Table for computing a single signal from a differential ECL clock
--pair. Mode is '1' or '0' when the signal is single-ended. The rest of
--the table is self-explanatory :)
------------------------------------------------------------
CONSTANT ECL_clk_tab : VitalStateTableType  := (
----------------------------------------
    ------INPUTS-------|-PREV---|-OUTPUT----
    -- CLK CLKNeg Mode | CLKint | CLKint' --
    -------------------|--------|-----------
      ( '-', 'X', '1', '-', 'X'), -- Single-ended, Vbb on CLK
      ( '-', '0', '1', '-', '1'), -- Single-ended, Vbb on CLK
      ( '-', '1', '1', '-', '0'), -- Single-ended, Vbb on CLK
      ( 'X', '-', '0', '-', 'X'), -- Single-ended, Vbb on CLK_N
      ( '0', '-', '0', '-', '0'), -- Single-ended, Vbb on CLK_N
      ( '1', '-', '0', '-', '1'), -- Single-ended, Vbb on CLK_N
      -- Below are differential input possibilities only
      ( 'X', '-', 'X', '-', 'X'), -- CLK unknown
      ( '-', 'X', 'X', '-', 'X'), -- CLK unknown
      ( '1', '-', 'X', 'X', '1'), -- Recover from 'X'
      ( '0', '-', 'X', 'X', '0'), -- Recover from 'X'
      ( '/', '0', 'X', '0', '1'), -- valid ECL rising edge
      ( '1', '\', 'X', '0', '1'), -- valid ECL rising edge
      ( '\', '1', 'X', '1', '0'), -- valid ECL falling edge
      ( '0', '/', 'X', '1', '0'), -- valid ECL falling edge
      ( '-', '-', '-', '-', 'S')  -- default
    ); -- end of VitalStateTableType definition
```

Figure 3 Table lookup for sensitive (CLK) diff. inputs

the positive pin, the negative pin becomes an inverting single-ended input pin.

Searching for a way to model this using the std_logic_1164 package of 9 logic values, we settled on the use of ‘W’ as an indicator of VBB. ‘W’ is fairly safe to use, since it shouldn’t occur very often from natural causes - driving a resolved net with ‘L’ and ‘H’ simultaneously is the only realistic way for this to happen. Since ECL logic is wired-OR, there are a lot of pull-down resistors, and since the VHDL model of a resistor converts the strong ‘0’ of a GND to a weak ‘L” on the other port, there are a lot of ‘L’ values floating around the simulation schematic. But there should be no ‘H’ values unless there are pull-up resistors in the design (using a Thevenin equivalent DC termination scheme for ECL terminations precludes this from happening: one 50-ohm resistor to VTT replaces the voltage divider). The value ‘-‘ is not suitable for VBB because, when resolved with ‘Z’, it becomes ‘Z’. A digital capacitor model we use drives a net with ‘Z’, so that would ruin the VBB value if (as is good design practice) the differential pin hooked to VBB had a bypass capacitor on its net.

Modeling differential output is trivial - simply use inverting and non-inverting concurrent procedures to drive the output ports from an internal signal. This also allows mapping the drive technology onto the port through the VitalResultMap parameter of the concurrent procedures. For the case of ECL wired-OR logic, the mapping of the ‘0’ output value to the ‘Z’ output value rather than ‘L’ forces the user to pull down the net with a digital resistor. Catching non-terminated nets in ECL designs is a big plus for the simulation.

VITAL compliance

There are 2 levels of VITAL compliance: Level 0 compliance pertains to the entity description and serves to keep the model “backannotatable”. There is really no reason any model should be written without complying to Level 0 - if it’s on the board, it should be able to receive backannotation of timing information, either intrinsic timing using the technology independent method discussed earlier, or wire delay timing.

The other level of compliance is Level 1, which pertains to the architecture. Level 1 compliance is a straightjacket, but a necessary straightjacket, to ensure the model conforms to the standard VITAL methods of handling timing checks, path delays, etc. and to

facilitate the acceleration of VITAL simulations by simulator vendors and thereby make gate-level VHDL simulations competitive as a choice for a method of doing gate-level simulations.

Our experience is that it takes a fairly complex part to have to depart from conforming to VITAL Level 1. An example of a very complex part, modeled Level 1 compliant is the Motorola ECLinPS part eclps445 - a serial/parallel converter. We were able to model it with a structural description of many VITAL state tables hooked together like registers as shown in the logic diagram in the Motorola ECLinPS data book. This part represents probably the upper bound of what can be done strictly Level 1 compliant. Indeed, for board designs, since we're dealing with hundreds of components instead of millions of gates, the need for accelerated simulation is not as great, but it is invigorating to be able to specify:

```
ATTRIBUTE VITAL_Level1 of
vhdl_behavioral: ARCHITECTURE IS TRUE;
```

in the model. Even if Level 1 compliance in the functionality section of an architecture is too much to accomplish (control structures or math are needed to describe the functionality), the timing checks procedures, wire delays procedures, negative timing constraints procedures, and path delay procedures of VITAL should be used to make the model have the look and feel, and thus the readability, portability, and maintainability, of a VITAL model.

Modeling the unglamorous: passives, connectors

To complete the job of getting to an all-VHDL board simulation, the assorted board-related components that one doesn't usually think of in connection with simulation must be modeled. These are the passive components (resistors, capacitors, inductors, potentiometers), active components (LEDs, diodes, the occasional discrete transistor, all linear and analog components), and another class whose components are electrically neutral but are on the board netlist (connectors, jumpers, test points, switches).

Of these, nearly all are trivial models. A digital capacitor model (used as a capacitor to GND, not in series with a net - that would be a different model altogether) should just drive a 'Z' value onto both of its ports.

Likewise, a connector model should drive 'Z', but the port should be declared INOUT rather than OUT, allowing an internal signal and a Vital wire delay generic to be used with a VitalWireDelay procedure so that the wire delay can be backannotated, and the wire delay observed within the connector model.

The most subtly difficult model to write was the bi-directional digital resistor model. We wanted the resistor to perform accurately some functions in the digital frame of reference such as pulling up or pulling down nets where required. If a designer were to leave off a pull-down resistor on an wired-or ECL net, we wanted an error to be generated in the simulation (i.e., the resistor must convert the 'Z' value on the net to 'L' or '0'). We also wanted to allow the resistor model to be used in series with a net, as is done in some termination schemes. It had to be directionally symmetrical; the designer could align it anyway he likes on the schematic with impunity. We did not want to force the designer to put properties on the resistor symbol to tell the simulation model how the resistor was being used - instead we wanted the resistor model to be self-sufficient. We did not want the resistor to perform any analog calculations, even simple voltage dividers, so there is no place in the model for the ohm value of the resistor.

The model we came up with is a two-process model which relies on the signal'TRANSACTION attribute. A signal'TRANSACTION is a bit signal which toggles its value every time there is a transaction on the attributed signal - a transaction occurs every time the signal is assigned a value, regardless whether the value is the same as the previous value or not. Each process in the resistor model waits on the signal'TRANSACTION signal to have an event - which triggers the process every time anything occurs on the signal. An event on the signal only occurs when the new value is different, and in the std_logic_1164 package, changes in strength (i.e. going from a '0' to 'L') do not count as events.

This model converts strong values on one port to weak drivers on the other, propagates the 'X' and 'U' values, and has been written for VITAL Level 0 compliance[1] (wire delays can be backannotated to it). It should be considered only 95% tested; we welcome any feedback on its usage.

Thoughts about all-VHDL board descriptions (utopian musings)

A structural VHDL netlist is a very good way to represent a board. The generic list of a VHDL entity provides an adequate mechanism for describing any

[1] Model is available at the FMF web site. **ftp://vhdl.org/vi/fmf/**

necessary information about an electronic part. For taking the board into a Printed Circuit Board design environment, I think it would be straightforward to devise a standard for specifying physical device information (package type, sections, sizes, pin information, etc.) in the generics of the component models and routing constraints in signal attributes such that the VHDL netlist would be the primary database for both the simulation and PCB environments. The process of packaging would then become a process of parsing and rewriting the netlist, combining user-entered packaging directives and properties attached to symbols and nets with information optimized by the packager.

Packaging would not discard architectures but rather put physical IC package component "wrappers" around the single-bit simulation components' entities. These IC package components would consist of entities which match the actual physical package, and structural architectures that instantiate the single-bit simulation models. The generic lists of the IC package components would contain generics for reference designators, JEDEC types, pin assignments for all ports - all the usual and necessary information that a front-end schematic capture and simulation tool set passes on to the PCB tool after packaging. Additionally, they could contain thermal and signal integrity data or references to such data. In some respects, the EIA-567 standard was headed in the right direction for this paradigm to be advanced; only, it seems it was not coordinated with the evolution of the VITAL standard.

```
ENTITY dip14_7400 IS
  GENERIC (
      refdeg        : string := "U4";
      jedec         : string := "DIP14";
-- and so on
);
  PORT (
      Pin1          : IN std_logic;
      Pin2          : IN std_logic;
      Pin3          : OUT std_logic;
-- and so on
);
END dip14_7400;
ARCHITECTURE packaged OF dip14_7400 IS
  COMPONENT V7400  -- simulation model
      PORT(a,b : IN std_logic;  c : OUT std_logic);
      GENERIC MAP -- to be filled in by student
  END COMPONENT;
BEGIN
  Section_1 : V7400
      generic map ( InstancePath => P93,
            TimingModel => SN74AS00 -- etc.)
      port map (Pin1,Pin2,Pin3);
-- and so on
END packaged;
```

Figure 4 Post-packaging entity/architecture

After packaging, this new VHDL netlist, with its extra level(s) of hierarchy, would still be functionally the same and simulate the same as the netlist before packaging. Of course, if the technology independent method outlined earlier is being used the mk_sdf utility would have to be rerun to make any board-level SDF files generated match the new levels of hierarchy created by the packager (component labels will have changed).

.As the PCB design process produces pin and gate swaps and changes to reference designators, these operations would cause changes and updates to the packaged VHDL netlist . Backannotation of PCB place and route delays would be facilitated because PCB tools would write out SDF files, simulation would be done on the same VHDL database as the PCB tool's, and keeping track of the mapping of component labels, net names, etc. from the simulation database (VHDL netlist) to and from the PCB database would cease to be a problem.

This would introduce a degree of portability to board designs not found now in the CAD/CAE world. The packaged, PCB-ready, VHDL netlist could be taken easily from one CAD environment to another. Even backannotation from one vendor's PCB environment to a different vendor's schematic capture environment would be facilitated (if the original board schematics were done using the schematic capture vendor's tools - I'm not suggesting machine-drawn schematics here). This all assumes that standards for representing packaging information in board-level VHDL netlists would be created and observed, of course.

I don't want to presume to suggest any skeleton of such a future standard, but I hope this meandering, blue-sky essay constitutes a challenge to the CAD vendors to show how much they support standards, portability, and open interoperability of their tools across vendors, by taking up this pursuit.

Another advantage of representing a board as an all-VHDL netlist could be, in addition to the obvious steps towards facilitating hardware/software co-simulation, the possibility of putting the "whole" board into an emulator, as is sometimes done with an ASIC. If both a board and an ASIC to be emulated are VHDL structural netlists that are (largely) VITAL-compliant, the use of an emulator could be extended to emulating the board without too much difficulty. This might be

another way to look at both I&T of hardware, and software development utilizing pre-built (emulated) hardware. The use of existing emulator technology might provide a clean interface between the software and hardware under design that both speeds up hardware/software cosimulation and leverages the user interfaces already provided by emulator vendors.

Summary

Board-level component modeling using VITAL is available now as a solution for design centers seeking a standard, non-proprietary means of developing their libraries. Using VITAL, board-level component models can be done with ASIC sign-off quality timing checks and functionality. Since ASIC and FPGA designs are done (unless they're done in that *other* language) in VHDL, the training, maintenance, and support for simulation is reduced to one language and tool (although portability between various VHDL tools is part of the benefit). For the last 2 years we've researched and put into place an increasingly all-VHDL simulation environment with very good results, and the lessons learned are described, for the most part, in this paper.

Acknowledgments

The work done in board-level component modeling described herein was a team effort, as may be ascertained by the use of the first person plural throughout this paper. Richard Munden (currently with Acuson Inc. and the Free Model Foundation, formerly with TRW), Luis Garcia (currently with IKOS Systems and the FMF, formerly with TRW), Bob Harrison (currently with TRW and the FMF), Gordon DeSmet (TRW) and Raymond Steele (TRW) contributed or just flat out did a lot of the work.

TRW Inc. has been incredibly progressive and generous in donating its board-level VHDL models to the FMF and thereby to the public domain.

References

1. IEEE Standard VHDL Reference Manual
2. IEEE VITAL Standard ASIC Modeling Specification
3. Vreeland, R.E., "Tricks and Techniques for Writing VITAL 3.0 Compliant ECL Models", Cadence International User's Group, Mentor User's Group, both Fall 1995
4. High Performance ECL Data, ECLinPS and ECLinPS Lite, Motorola Inc.
5. McKinney, Michael D. "The Bidirectional Wire Model with Delay", Proceedings of the Fall 1994 VIUF