Using Both VHDL and Verilog for Board-Level Simulation

Acuson Corporation
munden@acuson.com

Free Model Foundation
munden@vhdl.org

Introduction

Acuson designs and manufactures high-end medical ultrasonic imaging machines. Like many companies in Silicon Valley, it has traditionally been a verilog house. Verilog was originally used for ASIC design and verification. More recently, it has been used for FPGA synthesis. It has worked fine in these applications. We intend to continue using it as a front end to synthesis and as a sign-off simulator for ASIC vendors.

Eventually Verilog found its way into board-level simulations. Here difficulties were encountered. The first problem was in netlisting. Because Verilog is used primarily for chip design where a single library is used, the netlister did not handle multiple libraries well and was famous for spewing out error messages that were totally unrelated to the actual errors.

Although the netlister has since improved, other problems still remain with using verilog at the board level. The most critical is model availability. There are very few source level models available for commercial components in verilog. In contrast, there are free, unencrypted models available in VHDL for at least 1900 part numbers. More are being published every week. Also, while Verilog works fine for simulating 1's and 0's, board level simulation needs to work well with signal strengths that are modified by resistors used for terminations and pull-ups or pull-downs. Board level simulation must also handle components that are open-collector or open-emitter, or that have differential inputs.

For these and other reasons, it was decided to migrate Acuson to VHDL for board-level simulation. The Free Model Foundation had already done a great deal of work on techniques of modeling commercial components so, we chose to use their modeling and simulation methodology. However, all those ASICs and FPGAs are stilling going to be done in verilog. So, a mixed HDL simulation environment is required. In this environment most component models are in VHDL while ASIC and FPGA models are usually in verilog. The test bench may be in which ever language the engineer prefers.

It was also recognized that if we stayed in a single language environment, whatever language we chose, some model, critical to our ability to simulate a design, would likely be available only in the other language.

A primary goal of the transition is to make the overall design and simulation cycle easier and faster. We tested different mixed language simulators and determined that a single kernel simulator was required. Non-single kernel tools were biased in favor of the language in which the design was netlisted. They offered limited visibility into that part of the simulation that was taking place in the other language. The only single kernel simulator available at the time was ModelSim from Model Technologies. Since that time, Cadence has released their long awaited Affirma NCsim simulator, also known as "ncsim". This paper will discuss how either simulator can be used with the Cadence schematic capture tool, Concept.

An advantage of HDL simulation is that it is standards based. As other CAE vendors release single kernel simulators, Acuson will have the option of considering them. I anticipate that simulators should be fairly interchangeable in our simulation flow. Minor adjustments to the libraries may be required.

Library Structure

The heart of every CAE process is its libraries. This is particularly true in the board design and simulation process. Acuson uses the Concept schematic capture system from Cadence as a front end to Cadence's Allegro PCB design tool. Acuson's libraries have been optimized for mixed HDL simulation. They are built on an extension of the library structure designed at TRW in 1995 for VHDL (Leapfrog) only simulation.

Technology Independence

A key feature of the Acuson library is the separation of functionality from timing. This allows for technology independence and significantly reduces the total number of parts (and models) in the library. While part types come in multiple speed grades or technologies, and in many packages, they require only a single model. The saving varies with part family - for the 7400 series,

the savings can be huge; for more specialized components, it may be minimal.

Technology independence also allows Acuson to use timing values that are customized to our own needs. We are able to do that without changing or recompiling the models.

Directories for Cadence/NCsim

Let's look at a library set up to use FMF style models in an all Cadence design flow. We will assume the std02 is modeled in verilog.

The directory structure starts out looking very similar to the traditional Valid/Cadence SCALD library as can been seen from figure 1.

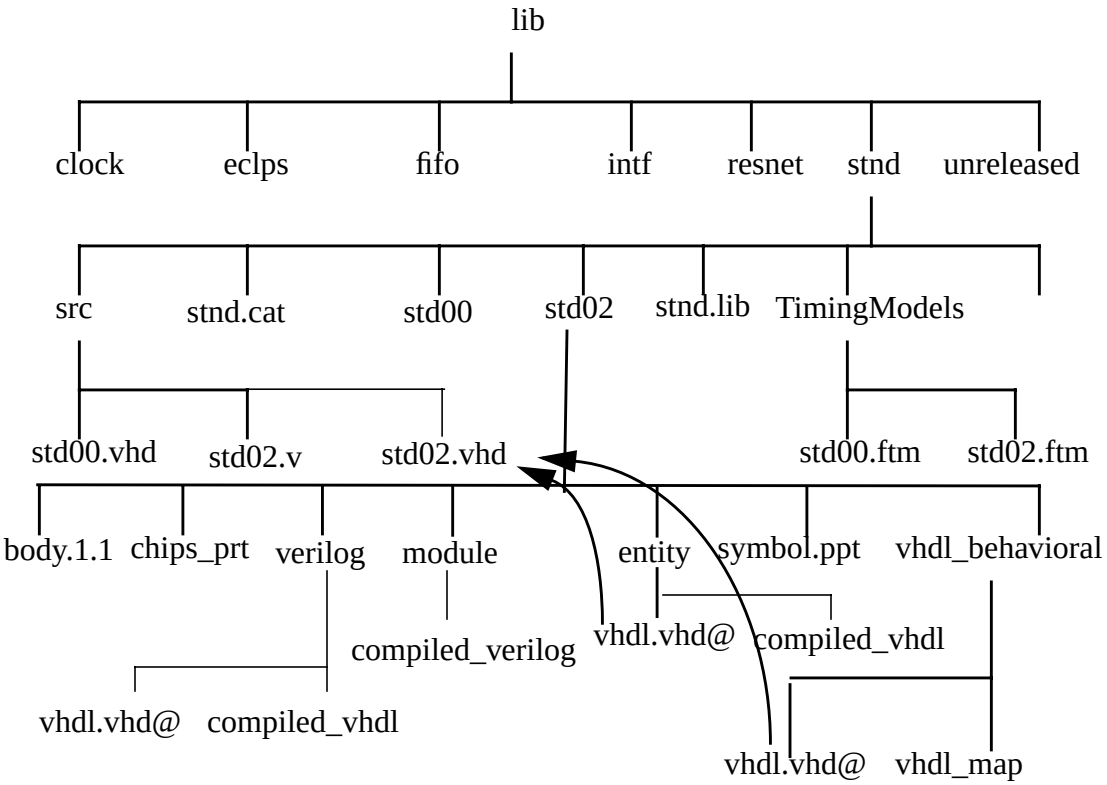

FIGURE 1. Library directory structure 1

Within each component group there are two new directories, *src* and *Timingmodels*. The *src* directory contains the model VHDL or verilog source code. The *Timingmodels* directory has the timing files for each model. Each timing file can hold timing information for any number of variations of a part type.

In figure 1, the std02 uses a verilog model. Within the *entity, vhdl_behavioral* and *verilog* directories there are links named *vhdl.vhd*. They all point back to the model source in the *src* directory. These links are required for the netlister, VHDLLink, to work. The reason the *verilog* directory exists is because the architecture of the VHDL model is named "verilog". A part modeled in VHDL would not have the *verilog* or *module* directories.

The body file is the schematic symbol and the *chips_prt* file maps the symbol to the physical pin numbers of the part. A single *chips_prt* file can handle all the different packages the part can be purchased in.

At Acuson, after the librarian creates the component directory with the body files and *chips_prt* file, a perl script called chp2vh is run. It creates the *entity* and *vhdl_behavioral* directories, a dummy VHDL model, a *vhdl_map* template, and the two links. For parts modeled in VHDL, the dummy model is then replaced with a functional model and the map file edited accordingly.

When the VHDL model is compiled with ncvhdl, the compiled versions of the model go into the *entity* and *vhdl_behavioral* directories. If the part is verilog based, the VHDL architecture will be named "verilog" and a *verilog* directory will be created by the

compiler. This directory will also contain compiled VHDL and a link back to the source code.

If there is a verilog model, it is compiled with ncvlog creating a *module* directory. The compiled verilog is stored here.

The *symbol.ppt* file is what is called a physical part table. It allows various properties to be associated with a symbol when it is placed on the schematic. These properties then determine the foot print that will be used in layout, the timing that will be used in simulation, and other things as determined by the librarian.

Directories for Cadence/Modeltech

In the case of Cadence used with the ModelSim simulator, the directory structure looks similar to the previous Valid/Cadence SCALD library but is somewhat simpler as seen in figure 2.

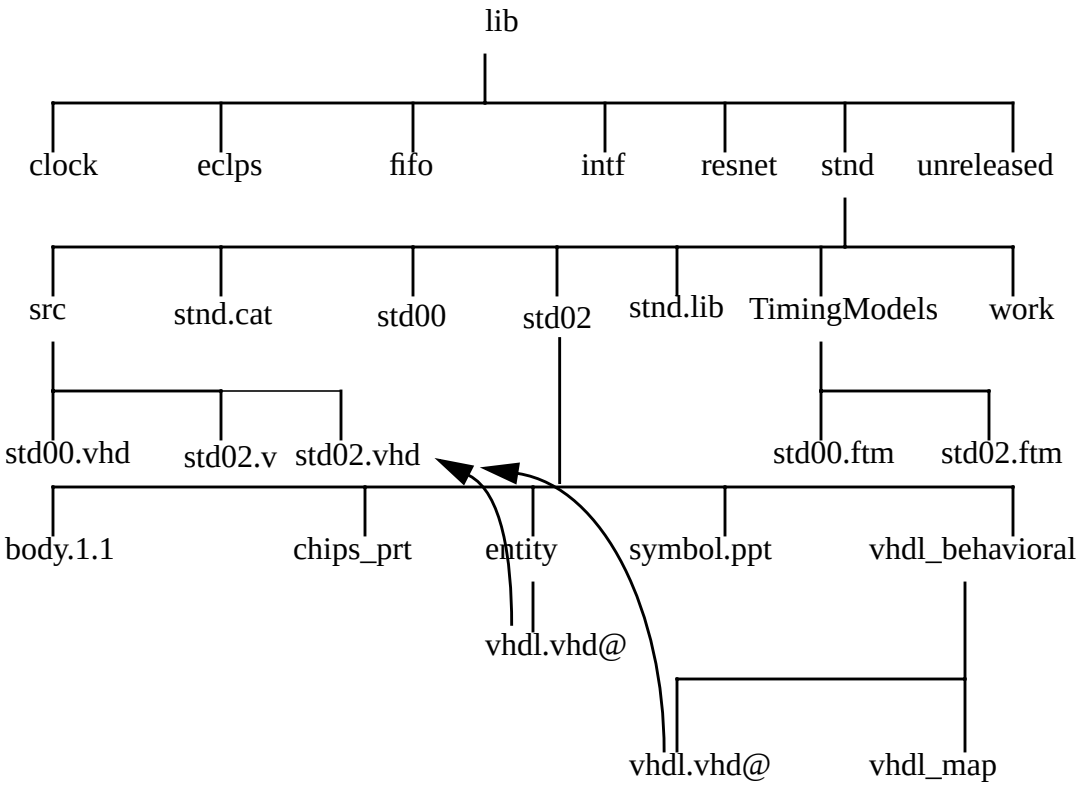

FIGURE 2. Library directory structure 2

There is a new directory named *work*. The *work* directory contains the ModelSim compiled VHDL and verilog models for the entire library. The compiled code is no longer stored in the part directory. The *verilog* and *module* directories are not needed and all parts have the same directory structure whether they use VHDL or verilog models.

Map Files

Map files are required to map pin names from Concept *chips_prt* files to the VHDL port names listed in the entity. Concept allows pin names that are illegal in VHDL and verilog so, the names must be mapped to the

legal names used in the models. A sample *vhdl_map* file is shown in figure 3.

```
FILE_TYPE = VHDL_MAP;
PRIMITIVE 'ACT04_DP','ACT04_SO','F04';
   DEFAULT_MODEL = 'STD04';
   MODEL 'STD04';
      PIN_MAP
         'A'<0> = '(A)';
         '-Y'<0> = '(YNeg)';
      END_PIN;
   END_MODEL;
END_PRIMITIVE;
END.
```

FIGURE 3. sample vhdl_map file

Mapping Verilog Models for NCSim

Including verilog models in the simulation is transparent to the user but adds several additional steps for the librarian. The steps outlined below are those required for Cadence's new Affirma NCsim product. They will be contrasted with the procedures used in a mixed Cadence/ModelTech environment.

Compile Verilog

After adding the verilog model to the *src* directory in the library, the first step is to compile it. The command is "ncvlog src/std02.v". This will compile the verilog model and place the results in std02/module.

The next step is to create the VHDL shell. This will be a dummy VHDL model that has an entity with a port list but an empty architecture. We can create this model by running the ncshell command "ncshell -import verilog -into vhdl -mode event stnd.std02". The shell is required for netlisting the design in VHDL. It will replace any dummy model that may have been created by chp2vh (described below).

Map File

The map file, *vhdl_map* links the VHDL entity to the Concept symbol. A template for the map file is generated by running a perl script named chp2vh. The file must then be manually edited to be sure it matches the VHDL entity created by ncshell. Because we are netlisting in VHDL and using a single kernel simulator, we want to avoid any references to verilog.

Mapping Verilog Models for ModelSim

The steps outlined below are those required for mapping a verilog model using ModelSim. They are similar to the steps needed using ncsim but the commands and order differ.

Map File

This time we start with the map file. We run our perl script, chp2vh, to create a *vhdl_map* template and links to a dummy model.

Compile Verilog

The verilog model gets compiled into the *work* directory for the library it resides in. The ModelSim command is "vlog <filename>". VHDL models, if they exist in the library, may be compiled into the same directory. No new directories are created under the part.

Update Dummy Model

The dummy model must have an interface (entity) that maps directly to the verilog module. ModelSim has a utility called "vgencomp" that reads the compiled verilog model and outputs a corresponding VHDL component declaration. This component declaration is then used to correct the VHDL entity for the dummy model.

The command to run vgencomp is "vgencomp std02".

Update the Map

Since we changed the entity in the dummy model to match the verilog module interface, we now must edit the *vhdl_map* template to match the new entity. This looks like a great opportunity for another perl script. At the moment, it is being done manually.

Caveat

It is important in the ModelSim flow that the dummy VHDL model is either never compiled or always overwritten by compiling the verilog model afterwards. Otherwise, assuming it compiles successfully, the simulation will use it instead of the verilog. Since the dummy model has no behavior, it will simulate as an open circuit but will not necessarily give any error messages.

Tool Flow

The tool flow is similar with either tool set. Here is the general design flow through simulation and analysis. It shows the various tools and the order in which they are used.

Libraries

It is assumed at this point that the libraries are in place and are pre-compiled for the user. This would normally be done by the librarian, however, the user may be responsible for setting up any verilog FPGA or ASIC models in the design as described above. Acuson's component libraries are technology independent and utilize FMF simulation models. This reduces the effort required for library development and maintenance.

Schematics

Schematics are drawn in Concept as they would be for any board design with some caveats to be discussed below. The schematics may be either hierarchical or flat. Because Acuson uses FMF technology independent libraries, components must be added to the schematics using the component browser set to physical mode. Selecting parts this way causes properties to be added to the schematics that are later used to select the correct timing and physical package for each component.

Netlister

When the schematic is sufficiently complete, it is netlisted using VHDLLink. VHDLLink reads a command file that needs to be correctly configured. Note worthy items in the *vhdllink.cmd* file include:

- connect_by_name on;
  - to prevent netlisting errors and make the netlist more readable;
- temp_sig on;
  - forces the netlister to declare local signals for connecting components rather than connecting ports to each other directly;
- verilog_import off;
  - the netlister does not need to know you use verilog;
- vhdl_rep on;
  -required for FMF style, single gate models. Cadence forgets to put this in the setup form in some releases.

Once the command file is correct, VHDLLink may be run from the command line or from an icon in the c2v process manager.

Examine the vhdllink.lst file for errors and warnings. If the libraries have been throughly tested, the only errors are likely to be related to the schematic.

VHDL Compiler

Having produced a VHDL netlist, the next step is to compile it. Using the Cadence simulator the command would be "ncvhdl <filename>". For Acuson, using the ModelTech simulator the command is "vcom <filename>". Before the netlist can be compiled the first time, a work directory must be established to receive the compilation results. In Cadence this is done by editing two files called *cds.lib* and *hdl.var.* For ModelTech it is done with the command "vlib work".

If there are compiler errors, they will be written to stdout.

SDF Generator

Each model has an associated timing file that describes the internal delays of a component with any required timing constraints. Some of the benefits of external timing files are discussed below in the Models section.

SDF generation produces a file in Standard Delay Format that may be used by the simulator to provide accurate timing for the simulation. Initial simulation runs may not require timing and can skip this step. Without SDF annotation, FMF models default to unit delays (1 ns).

The SDF tool is called mk_sdf and may be obtained as a Solaris binary or as C source code from the Free Model Foundation (at no cost). It uses a command file named *mk_sdf.cmd* which should reside in the working directory. Figure 4 shows a sample *mk_sdf.cmd* file.

```
SET sdffile_suffix _bat.sdf
SET use_global_timing_dir false
SET timingfile_dir TimingModels
SET timingfile_suffix .ftm
SET time_scale 1ns
SET local_path .
SET diagnostics off
SET vhdl_file vhdllink.vhd
SET lwb off
```

FIGURE 4. Sample mk_sdf.cmd file

Some of the lines in the command file that may require some explanation are:

- SET sdffile_suffix _bat.sdf
  - The SDF file will have the same name as the VHDL netlist with this suffix appended.

- SET use_global_timing_dir false
  - Set this to true if you will have all of your timing files in one local directory. Otherwise, mk_sdf will look in the Cadence libraries for the timing file directories.
- SET timingfile_dir TimingModels
  - The name of the directory(ies) that contain the timing files.
- SET timingfile_suffix .ftm
  - Timing files use the same name as their associated models but with this extension instead of .vhd.
- SET diagnostics off
  - If you are having difficulties getting mk_sdf to work, turning diagnostics on will result in some additional output files that may (or may not) help locate the problem.
- SET vhdl_file vhdllink.vhd
  - The name of the netlist. This will be either *vhdllink.vhd* or the design_name.vhd depending on a setting in *vhdllink.cmd*. It may also be supplied on the command line.
- SET lwb off
  - mk_sdf understands the Cadence lwb file structure. Turn this on only if you are running Logic Work Bench.

Any line in the command file may be overridden from the command line. Normal execution is simply "mk_sdf" without any arguments.

When using NCsim, the sdf file from mk_sdf must be compiled before it can be used. The command is "ncsdfc <sdf_file_name>". Then a sdf command file must be created to specify sdf related commands to the elaborator. This file may be as simple as a single line naming the compiled sdf file.

Elaboration

The Cadence NCsim simulator has a separate elaboration step. If timing backannotation is being done, the SDF file is read at this time. The command is "ncelab -SDF_CMD_FLIE <cmd file> mydesign". The cmd file specifies the name of a file of SDF annotation commands. SDF backannotation may be omitted by running "ncelab mydesign" instead. This will result in all models being simulated with 1 nanosecond delays.

Simulator

The ModelSim simulator does not have a separate elaboration step. Elaboration is always done when the simulator is started. The simulator is invoked from the command line with arguments for SDF backannotation and name of the SDF file and the design name: "vsim -sdfmax vhdllink_bat.sdf mydesign". If timing is not required, the "-sdfmax vhdllink_bat.sdf" argument may be omitted for unit delay simulation. Or, multiple SDF files may be read to include interconnect delays from Allegro, timing files for ASICs and FPGAs, etc.

For the Cadence tool, the simulator is invoked by entering "ncsim mydesign".

Based on the results of the simulation, there may be a loop back to schematic capture from this point to refine the design.

Layout

After satisfactory simulation results are obtained, the design goes to PCB layout. If timing margins were determined to be tight, anticipated interconnect delays based on manhattan distances between pins may be computed and backannotated through SDF to check if timing constraints are likely to be met by the proposed layout.

Signal Analysis

Subsequent to layout and routing of the PCB, signal integrity analysis may be performed. The same physical parts tables that selected the correct timing file for simulation also provided the name of the signal integrity model to be used in the Cadence signal integrity tool, SigNoise. SigNoise is capable of computing values for various physical effects such as crosstalk and noise margins. It can also compute accurate interconnect delays based on the characteristics of a driver's output buffer, receiver thresholds, propagation delays calculated for the board stackup, and transmission line analysis of the traces.

Timing Backannotation

Accurate interconnect delays, including transmission line effects, can be extracted from Allegro with the command "a2sdf -s <boardname> <outputfilename>". Allegro versions 13.0 and earlier produce a file that can only be read in Logic WorkBench. It should be possible to write a script that reads this file and the namemap files to produce a usable SDF file.

I have been told that Allegro produces a usable SDF file in the PE 13.5 release.

Simulator

Once the interconnect SDF file is generated, a final full timing simulation can be run. It would be more efficient at this point to run a static timing analyzer. Unfortunately there do not seem to be any good candidate

tools on the market. All the data needed appears to be in the SDF files but the few static timing analyzers available read only proprietary model formats. Perhaps this is a market opportunity for someone.

Special Considerations

While the design and simulation process described above should be relatively easy and straight forward, it is possible to make it difficult. Below are some techniques and caveats to keep in mind to avoid unnecessary complications.

Schematic Considerations

Schematics are the primary means of capturing board design intent. Minor details in schematics can have substantial downstream effects.

Net Names

Try to use legal VHDL names for all nets. Names that are not legal VHDL will be changed by VHDLLink into legal, but less readable, names.

Unused Bits

Some people like to run busses into hierarchical blocks but not connect all bits of the bus inside the block. The netlister may have difficulties with this. Either connect to the block only the number of bits actually used or, attach a flag body to each unconnected net inside the block.

Scald Name

The scald file (.wrk) and the top level schematic should not have the same name. Giving them the same name results in: “ERROR(252) Library, package, part name conflict.”

Mixed Busses

Sometimes, an engineer will create a bus for which some bits are inputs to a block and other bits are outputs from the block. Doing this will successfully prevent netlisting. All bits on a bus must go in the same direction.

Flag Bodies

The netlister occasionally has difficulties determining the correct mode of a port on a hierarchical body. Sometimes it assigns INOUT and other times it gives up and assigns UNDEFINED. A port mode of UNDEFINED will prevent compilation of the netlist. Usually port mode can be forced to the correct value by attaching the appropriate version of a flag body to the net inside the block.

Models

Acuson uses FMF style models. This modeling style has several important benefits.

- Technology independent models allow a reduction of the number of elements in the component library.
- New speed grades or technologies of a component can be added without duplicating anything already in the library.
- Model timing can be customized without touching the model.
- Simulations can be run with just unit delays, just component delays, or with component delays and timing checks.
- Interconnect delays can be backannotated from Allegro.
- Models are available.
- They are distributed as source code.
- They use a uniform coding style.

FMF

The Free Model Foundation is a not-for-profit corporation. Its goal is to improve the availability and usability or simulation models of off-the-shelf electronic components. It is working to accomplish this goal by helping the component vendors understand the benefits of providing such models. It provides services to the component vendors to assist them with technical issues. If staffing is a problem for the vendor, FMF can help them find out sourcing arrangements for model creation.

FMF will help users work with their suppliers to provide appropriate models. FMF encourages end users that must create their own simulation models, to share those models with others.

The Free Model Foundation offers all of its models, packages, and tools for download from its website at vhdl.org/fmf.

Conclusions

Board-level simulation is not as widely practised as it should be. The principle deterrents are model

availability and simulator ease of use. The advent of mixed language simulators improves the practicality of board-level simulation by expanding the base of models that may be used.

The Free Model Foundation is working to alleviate the model shortage by helping vendors provide open source HDL models. A large but insufficient number of models are now freely available. With the support of the user community, this can be vastly expanded.

Adequate simulators exist now. Performance is less of an issue for simulating boards than it is for ASICs. Ease of use is more important.

There are some tricks to mixed HDL simulation. However, most of these tricks can be performed behind the scenes by the librarian so the engineers do not have to learn them.

The acceptance of board-level in general, and mixed language simulation in particular, will be dependent on the ability of the tool vendors and the CAE support staffs to hide the underlying complexity. Engineers must have an easy, intuitive path to simulation or they will resist the transition from bread boards and iterative prototypes.