

Building an Environment for Mixed VHDL/ Verilog Board-Level Simulation

Richard Munden
Acuson Corporation
1220 Charleston Road
Mountain View, CA 94039-7393
munden@acuson.com

1.0 Introduction

Acuson 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 expect VerilogXL to remain the sign-off simulator for ASIC vendors.

Eventually Verilog found its way into Concept based, board-level simulations. Here difficulties were encountered. The first problem was in netlisting with vloglink. This tool was extremely painful to use initially. It did not handle multiple libraries well and was famous for spewing out error messages that were totally unrelated to the actual errors.

Over the last three or four years, vloglink has been greatly improved. It will usually be successful at producing a usable netlist. The error messages are still questionable but are correct often enough that they can no longer be completely discounted.

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 1700 part numbers. More are being published every week.

For these and other reasons, it was decided to migrate Acuson to VHDL for board-level simulation. However, all those ASICs and FPGAs are still going to be done in verilog. So, a mixed HDL simulation environment is required. In this environment most component models will be in VHDL while ASIC and FPGA models, and possibly a few component models, will be in verilog. The test bench may be in which ever language the engineer prefers.

The goal of the transition is to make to overall design-simulation cycle easier and faster. We tested different simulators and determined that a single kernel simulator was required. The only single kernel simulator available at the time was ModelSim from Model Technologies.

An advantage of HDL simulation is that it is standards based. When Cadence (or any other vendor) releases a single kernel simulator, we will have the option of considering it. I anticipate that simulators should be fairly interchangeable in our simulation flow.

2.0 Library Structure

The heart of every CAE process is its libraries. This is particularly true in the board design-simulation process. While Acuson's libraries have been optimized for mixed HDL simulation, they are only slightly changed from the library structure designed at TRW for VHDL (Leapfrog) only simulation.

2.1 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. The saving varies with part family - for the 7400 series, the savings can be huge; for more specialized components, it may be minimal.

2.2 Directories

The directory structure starts out looking very similar to the traditional Valid/Cadence library. Within each component group there are two new directories, "src" and

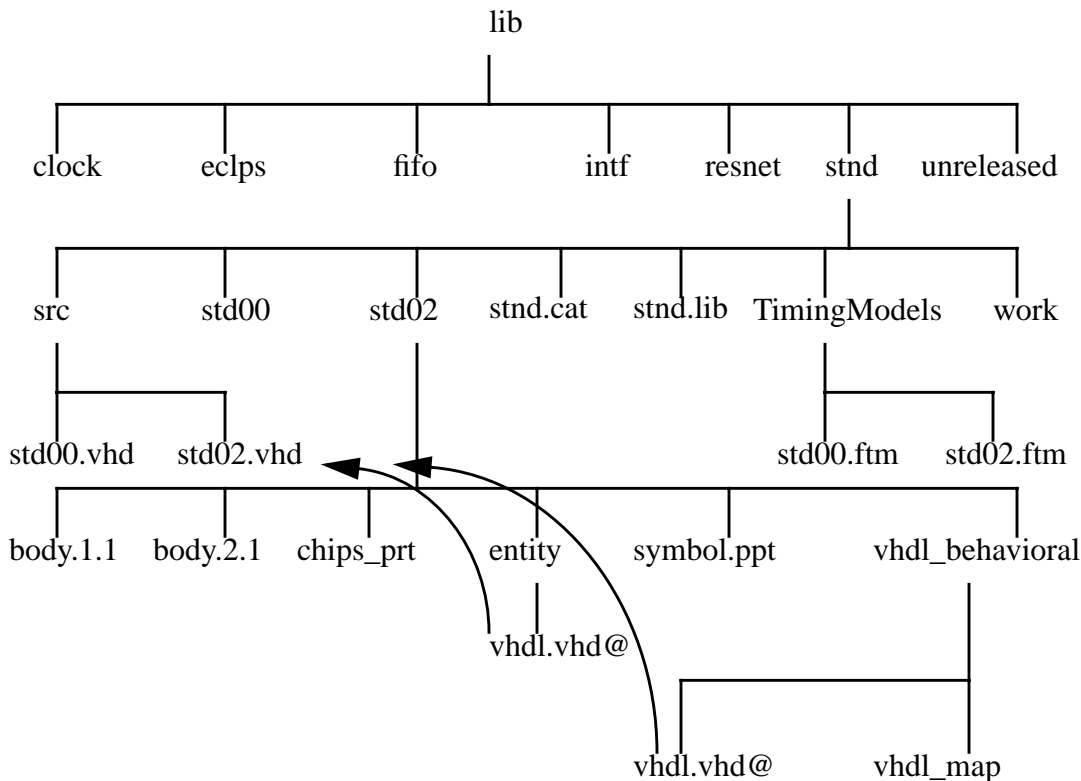


FIGURE 1. Library directory structure

“Timingmodels”. These contain the model HDL source code and timing files respectively.

There is also a directory named “work”. The “work” directory contains the Model-Sim compiled VHDL and verilog models for the library. This is a departure from the Leapfrog based structure I presented at this conference two years ago. Under that structure the compiled models resided under each component directory in the “vhdl_behavioral” directory. It is unclear which way will be best for the new Cadence tool when it is released.

Within the *entity* and *vhdl_behavioral* directories there are links named *vhdl.vhd*. They point back to the model source in the *src* directory. These links are required for the netlister, VHDLLink, to work.

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 model, a *vhdl_map* template, and the two links. The dummy model is then usually replaced with a functional model and the map file edited accordingly.

2.3 Map Files

Map files are required to map pin names from Concept *chips_prt* files to the VHDL port names listed in the entity. On a good day, they perform a simple function simply. On a bad day, it can take a lot of trial and error to figure out how they work.

```
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 2. sample vhdl_map file

One of the more interesting things to figure out was what to put in the primitive line of the map file. If a section of the *chips_prt* has only one entry on its primitive line, then the *part_name* from that section should be entered on the primitive line of the map file. However, if the section of the *chips_prt* has multiple entries on its primitive line, then each of those entries must be duplicated on the primitive line of the map file.

3.0 Mapping in Verilog Models

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 ModelSim. Based on experience with the beta version, I am confident they will be very similar for Cadence's new Affirma Co-Ex product.

3.1 Map File

Because we are netlisting in VHDL and using a single kernel simulator, we want to avoid any references to verilog. We need to fool the netlister into thinking there is a VHDL model even when the simulator will be using verilog.

We run our perl script, `chp2vh`, to create a `vhdl_map` template and links to a dummy model.

3.2 Compile Verilog

The verilog model gets compiled into the work directory for the library it resides in. The ModelSim command is “`vlog <filename>`”.

3.3 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 as part of the VHDL entity for the dummy model.

3.4 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.

3.5 Caveat

It is important 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.

4.0 Tool Flow

Here is the general design flow through simulation and analysis. It shows the various tools and the order in which they are used.

4.1 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.

4.2 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 for each component.

4.3 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 port to each other directly;
- `verilog_import off;`
- the netlister does not need to know you use verilog;
- `vhdl_rep on;`
-required for 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 thoroughly tested, the only errors are likely to be port mismatches. These are described later under special considerations.

4.4 VHDL Compiler

Having produced a VHDL netlist, the next step is to compile it. For us, the compiler is called *vcom* and 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. This done with the command “*vlib work*”.

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

4.5 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 is 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.

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.

```
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 vhdlldlink.vhd
SET lwb off
```

FIGURE 3. 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 vhdlldlink.vhd`
 - The name of the netlist. This will be either `vhdlldlink.vhd` or the `design_name.vhd` depending on a setting in `vhdlldlink.cmd`. It may also be supplied on the command line.
- `SET lwb off`
 - `mk_sdf` understands the 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.

4.6 Simulator

The ModelSim simulator is invoked from the command line with arguments for SDF backannotation and name of the SDF file and the design name: “`vsim -sdfmax vhdlldlink_bat.sdf mydesign`”. If timing is not required, the “`-sdfmax vhdlldlink_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.

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

4.7 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.

4.8 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 provide the name of the signal integrity model to be used in 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.

4.9 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 LWB. 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.

4.10 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.

5.0 Special Considerations

While the design-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.

5.1 Schematic Considerations

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

5.1.1 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.

5.1.2 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.

5.1.3 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."

5.1.4 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.

5.1.5 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.

5.2 VHDLLink

VHDLLink is a fairly robust program. However, it is not without some quirks and at least one flaw.

5.2.1 Understanding the Error Messages

VHDLLink error messages are not particularly cryptic but you have to be able to understand what they mean rather than what they say.

The most common error message is "ERROR(200) Entity port not described in part". If the entity port named in the message is the first one in the model, the real problem is probably:

1. The vhdl_map file does not exist.
2. The primitive line in the map file does not match up with either the part_name or the primitive line of the chips_prt file.
3. Either the chips_prt pin name or the entity port name is not correct in the vhdl_map file.

After ERROR(200) usually comes “ERROR(270) Entity parsing failed.” If this message refers to the same model mentioned in ERROR(200), it should be ignored. There is nothing wrong with the entity, the problem is in the map file.

6.0 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.

7.0 FMF

The Free Model Foundation is a not-for-profit corporation. Its goal is to improve the availability and usability of 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 encourages end users that have to create their simulation models to share those models with others.

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