Integrating FMF Models with Concept Libraries

Richard Munden
Acuson Corporation
P.O. Box 7393
Mountain View, CA 94039-7393
munden@acuson.com

1.0 Introduction

A new source of component simulation models has become available. The models come as source code VHDL and they are FREE! The name of the source is the Free Model Foundation.

This paper will explain the details of how TRW has integrated these models into its Concept/Allegro environment.

1.1 History

TRW has probably done business with every major EDA vendor that has ever existed and many minor players as well. We have always sought vendors we knew would be around for a long time. We picked companies we knew could survive the hard times because they were big and had deep pockets like HP and Tektronix. And we picked companies we knew would be around because EDA was their sole business and they were profitable like Cadnetix and Valid. It is a dynamic industry and longevity is not its most notable feature.

After going through a number of EDA suppliers and changing simulators each time, we decided it was time to switch to an industry standard modeling language that was supported by a majority of vendors. Until recently there was no such standard but (standards being gregarious things) now there are two. The two are Verilog and VHDL. Although either of these simulation languages would have worked we only needed one and since VHDL is widely known for its verbosity, complexity, and government involvement, it was the obvious choice. Its only real short coming was that there were no models available for off-the-shelf components. This is what the aerospace community refers to as an insurmountable opportunity.

1.2 About the Free Model Foundation

Simulation models are usually distributed in an encrypted form in order to protect the intellectual property of the modeler. However, this means the user is unable to look inside the model to determine the cause of simulation errors.

Being able to look inside a model means having the source code. To distribute source code is to give up control over usage, copying and redistribution. For business reasons, most modeling companies are unwilling to distribute source code.

Fortunately, the Free Model Foundation, otherwise known as FMF adopted a different business model. The purpose of FMF is to bring about a fundamental change in the way people think about simulation models. Instead of considering models to be intellectual property to be sold or protected, FMF is encouraging the view that models are documentation on how to use a component in the design of a higher level product. Models are extensions of databooks.

To that end, FMF distributes source code models produced by component vendors and produces and distributes its own models. These models are licensed under the same agreement used by the Free Software Foundation and are freely available to users. To see what models are available and download them, point your web browser to http:/vhdl.org/vi/fmf.

2.0 The Nature of FMF Models

FMF models are available for download on the internet and may be copied or modified. If you do not have access to the internet or would like to contribute to the operations of FMF, distributions on media are purchasable. If you find an error in an FMF model, please email a description of the error and the fix to the Free Model Foundation so that the fix can be incorporated into the distribution.

A style guide can be downloaded to help you write your own models compatible with those from FMF.

2.1 The Code

The VHDL models produced by FMF are written in standard VHDL. They use the VITAL (VHDL Initiative Toward ASIC Libraries) packages to the extent practical to make them more generic and allow the backannotation of interconnect delays. They also make use of some FMF provided packages.

These models have their timing parameters stored externally, making them technology independent. Thus several parts having identical function but different timing can share a single model. This greatly reduces development and maintenance efforts.

Wherever it makes sense, models are written as single functions matching the style of the version 1 bodies found in the Concept schematic libraries from Cadence. An important implication of this is the “replicate bodies” flag must be on in vhdllink when creating a VHDL netlist of the schematic.

2.2 The Timing Files

FMF models default to unit delay timing. Actual part number specific timing is in a file separate from the model. The timing file is in a Standard Generalized Markup Language (SGML) format. The Document Type Definition (DTD) is posted on the web. Each

file is divided into a head and a body (figure 1). Each of these have several sections of tagged data.

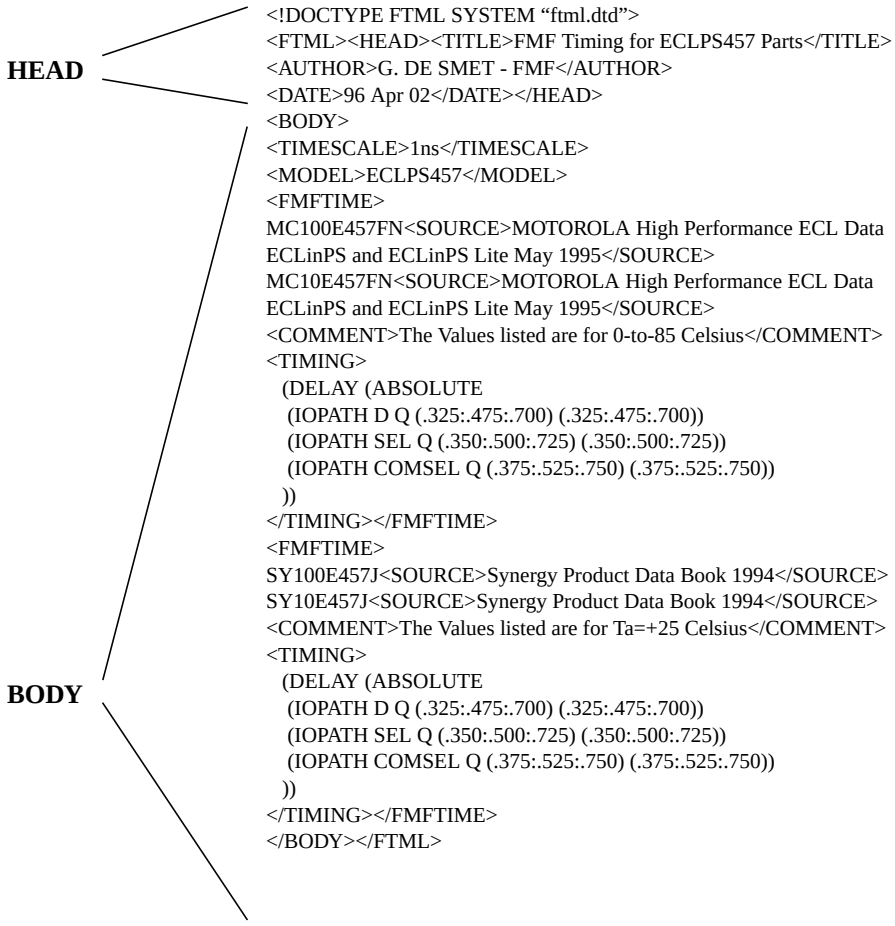

FIGURE 1. Sample timing file

The head contains title, author and date information. All of these tags are mandatory.

The body contains several mandatory and optional tags. Their names and descriptions follow:

- *TIMESCALE* Describes the time units used in the timing sections. There must be one and only one *timescale* tag per file.
- *MODEL.* This names the model with which the timing file is associated.
- *FMFTIME.* This is a section that describes the timing for one or more actual parts. There may be one or more *fmftime* sections per timing file. Each begins with a list of manufacturer’s part numbers for which the timing is valid.
- *SOURCE* is a optional tag. It is recommended that each part number listed have a *source* element indicating the origin of the timing parameters.

- *TIMING* is the tag that contains the actual timing data. The data within the *timing* tags, if selected, will be copied into the SDF file by the SDF generator.
- *COMMENT*s are optional and may be place anywhere within the timing file.

It is the intent of the Free Model Foundation to offer its timing file format for standardization. It is inevitable that some changes will occur in the process.

2.3 The *mk_sdf* program

A program called mk_sdf has been written to read the timing files and generate a SDF file. The SDF file is read into leapfrog during elaboration and provides the timing information for the simulation.

3.0 How it Works

To use FMF models in simulation, a process must be followed during the design phase. This process has been optimized to also make the schematic-to-layout transition as smooth as possible.

3.1 Drawing Schematics

Components are added to the schematic by using the parts browser. The browser is always set to physical mode. When a component is selected a menu appears offering a list of physical types (figure 2). The desired physical type is selected and another menu appears displaying the applicable portion of the physical part table for that generic part type. From that menu, an exact part number can be selected (figure 3). A symbol is then placed on the schematic and a group of properties are automatically attached. The properties added are part_name, pack_type and TimingModel.

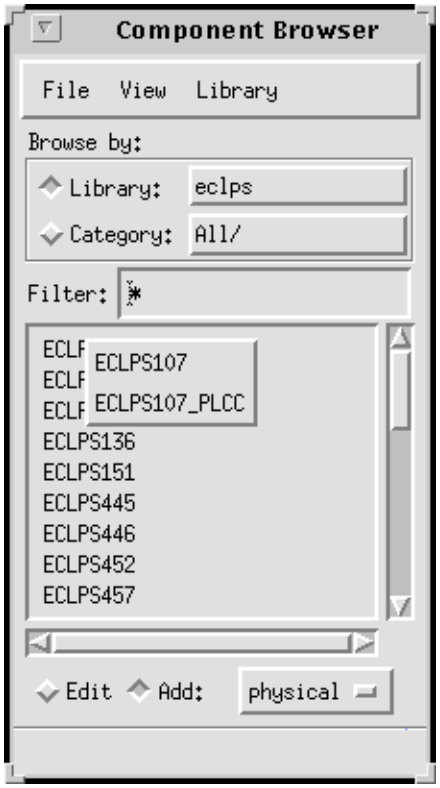

FIGURE 2.

3.2 Adding Timing

When the schematic is ready to be simulated, it is compiled into a VHDL netlist using vhdllink or Logic Workbench. The value of the TimingModel property replaces the default value of the TimingModel generic for each instance in the netlist.

Next the mk_sdf program is run. It reads the netlist and picks out the TimingModel generic value for each instance. It then searches the timing files associated with the VHDL models and finds the correct timing sections. Finally it builds the SDF file. The SDF file can then be read by

Leapfrog (or any other VITAL simulator) at elaboration time. The timing from the SDF file overwrites the default timing (unit delays) in the models.

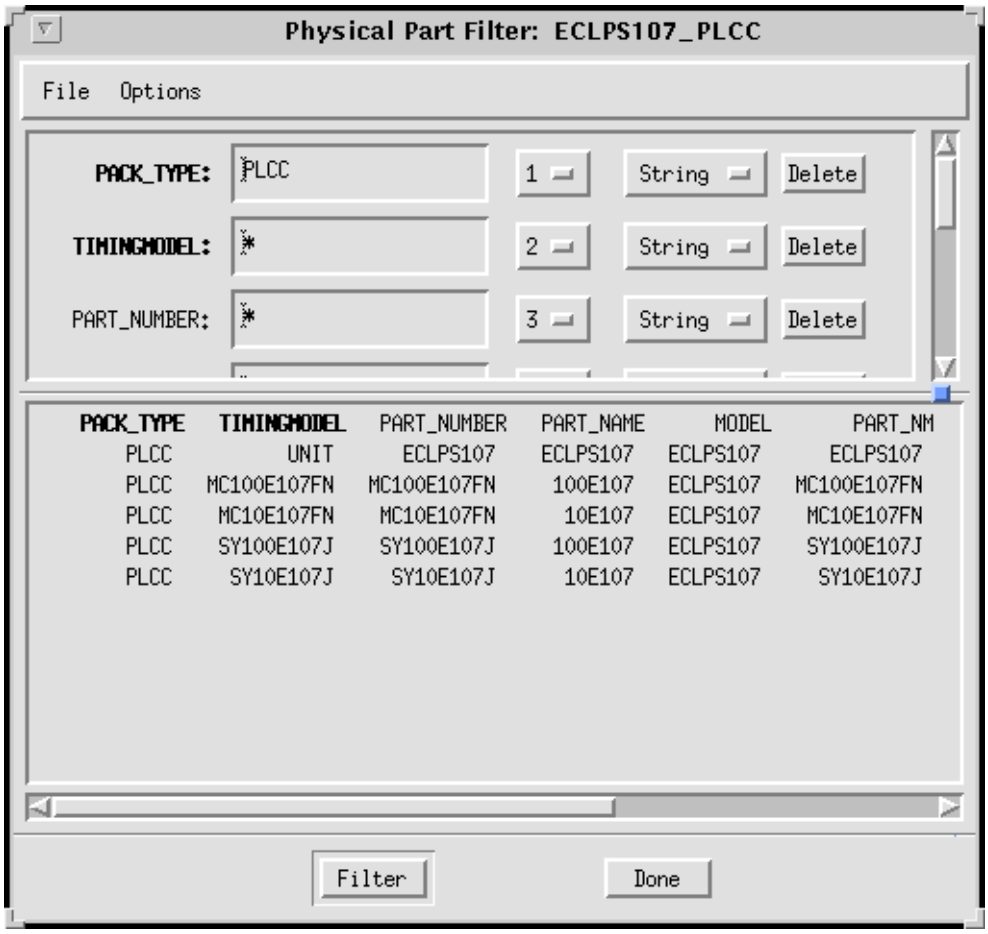

FIGURE 3.

4.0 Library Structure

The library is structured to take advantage of the VHDL models and the cell-based physical part tables. This adds a few more files to the library hierarchy but reduces the number of libraries required.

4.1 A Generic VHDL Structure

Since many parts are available in a variety of technologies or speed grades, the FMF modeling strategy is to cover several parts with a single model. This is possible because the timing is external to the model. Each model has a timing file with timing parameters for every part number for which the model is applicable. The model and the correct timing parameters are linked by the TimingModel property placed on the schematic when the part is instantiated from the PPT.

4.2 The Technology Independence Difference

The Concept libraries distributed by Cadence are technology dependent. They are so because it would add to much additional complexity to support Rapidsim models in a technology independent library. By abandoning Rapidsim, we have been able to move to technology independence, sharply reducing the number of libraries. A search of TRW's libraries for 244s recently found 40 different 244 parts (LS244, ALS244, etc.), each with its own symbols and models. This method should be able to cut that number to 1.

4.3 File Structure

The new library file structure is very similar to the traditional Concept library with a few additions. This is shown in figure 4. The *src* directory contains the VHDL source

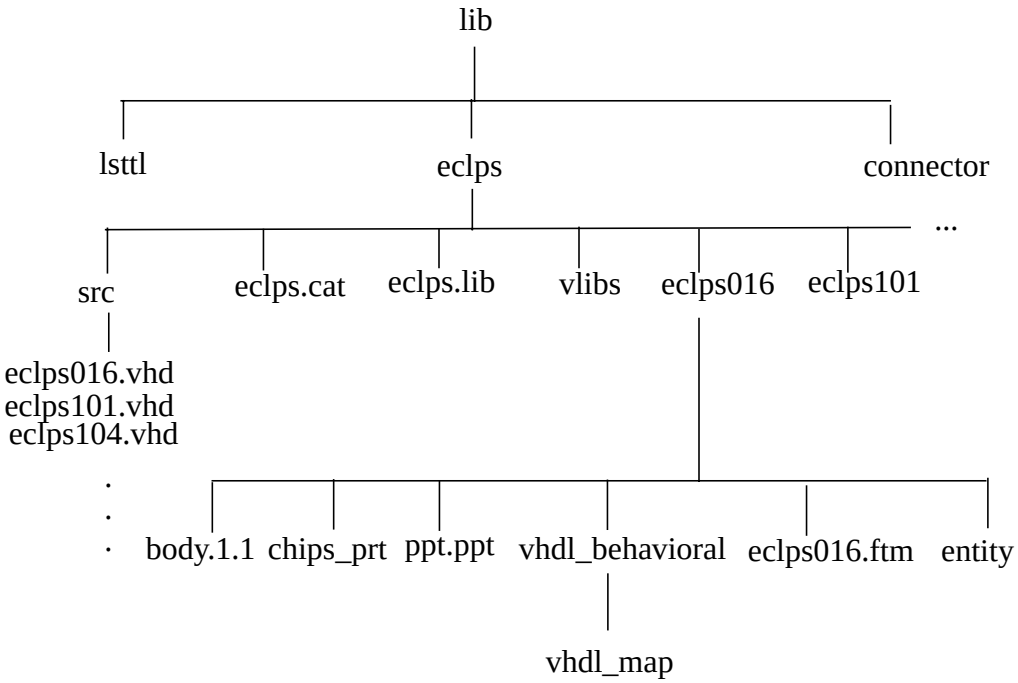

FIGURE 4.

files for the models. The entity and architecture are in the same file. The entity and architecture directories are created automatically when the models are compiled. They contain a number of binary files (the compiled models), and links back to the source code. Note that the models must be recompiled for each platform and each version of Leapfrog you will be using. Other files requiring special attention are discussed in the next section.

5.0 A Look at the Files

This library configuration contains most of the files usually found in a Concept library. However, some files now have some added sections or properties and a few new files have been added.

5.1 Bodies

Bodies are still made to our traditional graphic standard. However, the note describing the part name has become a property because the body will now be used to represent a number of parts with the same function.

5.2 Source Files

The *.vhd files in the *src* directory are the FMF VHDL models. Storing all the models for a library in one directory makes compilation a one command process. The names of the models must match the body names.

5.3 Vlibs

The *vlibs* file is used by the VHDL compiler to determine the name and location of the work directory. This is a 2 line ascii file. In 9504 it became 2 files, *cds.lib* and *hdl.var.*

5.4 Cell-Based PPTs

The cell-based physical part tables are an essential element in the library architecture and the key enabler of technology independence. In addition to the usual physical parameters listed in the PPT for consumption by the packager, these PPTs contain properties that name the timing files, part name, part number, and signal model. They also have extra data used for parts list documentation. The kinds of properties needed in the PPTs vary with the part type. Connectors are quite different from ICs. Figure 5 is a sample PPT for an ECL IC.

```
FILE_TYPE = MULTI_PHYS_TABLE;

PART '100E116'
COMP_DESC = 'QUINT DIFF LINE RECEIVER'
TH_DEV_TYPE = 10E116
: PART_NAME, PACK_TYPE, TIMINGMODEL = PART_NUMBER, MODEL, PART_NM, VENDOR, COMP_CAGE_CODE,
SIGNAL_MODEL;
100E116, PLCC, MC100E116FN (!) = MC100E116FN, 100E116, MC100E116FN, MOTOROLA, 04713, MC100E116FN
100E116, PLCC, SY100E116J (!) = SY100E116J, 100E116, SY100E116J, SYNERGY, 0ZN38, SY100E116J
END_PART

PART '10E116'
COMP_DESC = 'QUINT DIFF LINE RECEIVER'
TH_DEV_TYPE = 10E116
: PART_NAME, PACK_TYPE, TIMINGMODEL = PART_NUMBER, MODEL, PART_NM, VENDOR, COMP_CAGE_CODE,
SIGNAL_MODEL;
10E116, PLCC, MC10E116FN (!) = MC10E116FN, 10E116, MC10E116FN, MOTOROLA, 04713, MC10E116FN
10E116, PLCC, SY10E116J (!) = SY10E116J, 10E116, SY10E116J, SYNERGY, 0ZN38, SY10E116J
END_PART

END.
```

FIGURE 5.

5.5 Chips_prt

The *chips_prt* file is standard fare with one or two exceptions. There may be more than the usual number of entries. Technology independence requires separate entries for each technology type to account for differences in pin loading. Also, particular care is required in adding values to the *primitive* and *part_name* lines. A sample *chips_prt* file is

shown in figure 6. It was difficult to find the right combination of primitive names,

```
FILE_TYPE=LIBRARY_PARTS;
TIME=' COMPILATION ON MON NOV 28 11:49:22 1994  ';
primitive 'ECLPSL11','100EL11_SOIC','ECLPSL11_SOIC';
  pin
    'D':
      PIN_NUMBER='(7)';
      INPUT_LOAD='(0.0005,0.15)';
'Y0':
      PIN_NUMBER='(1)';
      OUTPUT_LOAD='(-7.6,-19.5)';
      OUTPUT_TYPE='(OE,OR)';
.
.
.
end_pin;
  body
    JEDEC_TYPE='SOIC8';
    CLASS='IC';
    FAMILY='ECL100E';
    TECH='100E';
    PART_NAME='100EL11';
    BODY_NAME='ECLPSL11';
    POWER_PINS='(VEE:5;GND:8)';
  end_body;
end_primitive;
primitive '10EL11_SOIC';
  pin
    'D':
      PIN_NUMBER='(7)';
      INPUT_LOAD='(0.0005,0.15)';
'Y0':
      PIN_NUMBER='(1)';
      OUTPUT_LOAD='(-7.4,-20.4)';
      OUTPUT_TYPE='(OE,OR)';
.
.
.
end_pin;
  body
    JEDEC_TYPE='SOIC8';
    CLASS='IC';
    FAMILY='ECL10E';
    TECH='10E';
    PART_NAME='10EL11';
    BODY_NAME='ECLPSL11';
    POWER_PINS='(VEE:5;GND:8)';
  end_body;
end_primitive;
END.
```

FIGURE 6.

part_names, and schematic properties to work with both packager-xl and vhdllink.

5.6 Vhdl_map

The *vhdl_map* file maps the pin names from the *chips_prt* to the port names in the VHDL model. You might not expect vhdllink to read the *chips_prt* file, but it does. For that reason, multiple entries are required on the *primitive* line of the *vhdl_map* file. A sam-

ple *vhdl_map* file is shown in figure 7. All the various *part_name*s and *body_name*s from

```
FILE_TYPE = VHDL_MAP;
PRIMITIVE 'ECLPSL11','100EL11','10EL11';
   DEFAULT_MODEL = 'ECLPSL11';
   MODEL 'ECLPSL11';
      PIN_MAP
         'D' = '(A)';
         '-D' = '(ANeg)';
         'Y0' = '(Y1)';
         '-Y0' = '(Y1Neg)';
         'Y1' = '(Y2)';
         '-Y1' = '(Y2Neg)';
      END_PIN;
   END_MODEL;
END_PRIMITIVE;
END.
```

FIGURE 7.

the *chips_prt* must be listed as primitives.

5.7 Timing Files

Timing files provide the part number specific timing for simulation. A single VHDL model is used for multiple part numbers. Each part number must have a section in the timing file. After the design is compiled by vhdllink and a VHDL netlist is produced, the mk_sdf program is run. It reads the netlist to determine the body_name and part_number for each instance in the design. It then finds the timing file for each model and extracts the timing parameters for each part_number used. Finally, it writes an SDF file that can be used to annotate the component timing into the simulation during elaboration. A separate SDF file can also be read into the simulator to add interconnect delays. A sample *vhdl_map* file is shown in figure 1. A document type definition is available from the FMF web site. However, the format is simple enough that timing files can be created with your favorite text editor.

6.0 Conclusion

By re-architecting our Concept libraries around the Free Model Foundation simulation models, TRW has been able to produce libraries that are more compact and maintainable. Although this is still a work in progress and we have not yet incorporated every type of component into this architecture, we have already seen benefits. These include reduced development times, unencrypted simulation models, more accurate simulations, and easier transitions from design to layout.