ASIC Flow Engine for Timing Closure (AFETC) a Makefile Generator to Automate Design Budgeter Methodology.

Thomas D. Tessier, t2design Incorporated
tomt@hdl-design.com

Marvin L. Anderson, StorageTek
marvin_anderson@stortek.com

ABSTRACT

In the DC98.08 tool release Synopsys added the capability for the designer to develop top level constraints and then flow them downward to the lower level blocks in a more uniform method than the previous process of Characterize, Compile and Write (CCW). This Synopsys tool is called the Design Budgeter. In order to flow the constraints the designer must, at a minimum, run four invocations of DC on each module. The first pass generates a GTECH DB. The second run develops a baseline for estimation. Invocation three applies the actual constraints to the blocks. The fourth pass does an incremental cleanup of the design. It would be quite easy in a multiple person team environment for the resulting net list to be incorrect with so many passes to manage. We chose to manage our process flow with a script. Our Perl script was designed to generate makefiles that automate the design flow. The use of the Perl script combined with a solid data management system provides design groups a repeatable and predictable flow from RTL to gates ready for layout.

1.0 Introduction

With today's Deep Sub-Micron Designs (DSM), clock speeds are increasing, the number of gates are multiplying, the time to market pressures are tightening and timing closure is a big problem. Traditionally EDA tools have used a bottom-up compile strategy, an engineer is developing the constraints at every level of hierarchy while the EDA tools compile the designs functional blocks from the bottom to the top. Data flow management is human intensive which results in longer iteration times and higher error possibilities.

The traditional staff breakdown used by ASIC design teams has been to either include a timing engineer who was dedicated to work only on timing aspects of an ASIC or expect every engineer on the team to be responsible for the timing of their own functional block. In the first case, the dedicated engineer had to understand the design very intimately to make good timing judgements. In the second case, when the chip is brought together combining the individual engineer's synthesis plans, "fur starts flying". Both of these approaches use a bottom-up compile and constraint strategy. Design team numbers are increasing because the designs are getting bigger. Neither approach provides a very acceptable solution.

A long time desire of the engineering community was to have the ability to apply top level constraints to their designs and have their EDA tools propagate them intelligently down through the hierarchy. To provide a solution, many of Synopsys competitors tried a Top-Down compile approach, which some refer to as hierarchical compile, applying constraints from the top and compiling the entire design from the top down through the hierarchy. Top level constraints improved the human management problem but as ASICs got larger and larger the Top-Down compile tools could not manage all the data. The tools started to hit their capacity.

Synopsys Design Compiler has had significant increases in its capacity and capabilities over the last several years. In Design Compiler (DC) 98.08 Synopsys added a tool called Design Budgeter. The Design Budgeter allows the designer to develop only top level constraints and then flow them downward to the lower level blocks in a more uniform method than the previous process of Characterize, Compile and Write (CCW). Design Budgeter uses a Bottom-Up compile strategy with a Top-Down constraint tool. While constraint building and flow is significantly improved, the bottom-up compile strategy still has data management problems. In order to flow the constraints with this tool the designer has to, at a minimum, run four invocations of DC on each module. The first pass generates a GTECH DB. The second run develops a baseline for estimation. Invocation three applies the actual constraints to the blocks. The fourth pass does an incremental cleanup of the design.

For our DSM design project we chose a strategy that utilized the Design Budgeter combined with the ASIC Flow Engine for Timing Closure the "AFETC" script to manage our process flow. Our Perl script was designed to generate makefiles that automate the design flow. The use of the AFETC combined with a solid data management system provides design groups with a repeatable and predictable flow from RTL to gates that is ready for layout. Synopsys has since developed a new tool Automated Chip Synthesis (ACS) which does accomplish some of the synthesis management that we developed with our script.

Many would ask why create yet another makefile generator when there are so many available? We needed a tool that could automate more of the flow then normally done in the ASIC environment. We planned to run multiple iterations of our flow. Each time we would provide a net list to our in-house layout group who would intern provide us with timing information. Our design team would make the necessary adjustments given the timing data, using AFETC generated files to produce additional iterations. The plan was to continue this cycle until we reached timing closure. For our purposes timing closure is defined as the point at which the design team believes they have acceptable timing margins to release the chip. More iterations give us a better opportunity to alter the design or timing paths to achieve a successful ASIC. The more iterations we ran meant increasing amounts of data which created even a greater need for a automated data flow management tool. We also needed checkpoints along the flow that were not obviously or easily provided by other makefile tool generators. Finally we wanted to design a flow for synthesis, incorporating timing closure, that could be leveraged and extended by future groups. Developing this scripted system in-house seemed the most logical approach.

1.1 The Goals of the AFETC Tool

We had very simple goals for the Perl script tool:

- Keep the design turn time from RTL to the first Layout net list at approximately 24 hours.
- Use TCL version of DC to provide a basis to expand the tool in the future.
- Build in sufficient hooks that allowed design problems to be resolved without breaking down the base foundation of the tool.
- Focus the tool on 0.25u and below technologies
- Leverage in-house facilities whenever possible. We wanted to be able to run as many iterations of layout as necessary to meet timing closure. Our tool base consisted of: Synopsys Design Compiler, Floorplan Manager, PrimeTime and TetraMax; Avant! Planet and Apollo; and VLSI Foundry tools.

2.0 The Flow to Automate

The flow and the tool were developed with DSM designs in mind. The design of the data flow that we wished to manage with AFETC evolved in parallel with the script itself. From the beginning we knew we would use top level constraints which were easier to develop, reduce the stress on everyone and let the tools handle the details. The flow took into account the iterative nature of EDA tools and synthesis. The iterations of the Design Budgeter was one area of concentration along with the loop between layout and synthesis to drive the timing closure. There are three main divisions of the flow. RTL to GTECH is a linear flow but we allowed the ability to restructure the logical design to better match the physical domain. GTECH to Layout has several iterative loops for the budget process with the addition of floorplanning and location based optimization. Layout timing closure is an iterative loop between place and route and synthesis to drive to timing closure.

2.1 RTL to GTECH

The RTL to GTECH step is important for two reasons; it prepares the input for automated budgeting and it identifies any test issues as early as a possible. The flow we used for the design is shown in Figure 1. on page 5. It can be debated that mapped designs used to drive the initial net list should contain scan chains. In our case we concentrated on getting the core circuitry to pass constraints relying on the Test Compiler to fix timing during the final insertion. Turnaround time for the automated process is reasonable so the test evaluation can become a parallel task.

2.2 GTECH to Layout

This is the heart of methodology as shown in Figure 2. on page 6. Most of the inputs or data used are from the resulting mapped and scanned design from the “pass 2” compile. The important decision made here is whether the overall magnitude of the constraint violations, if any, require another budgeting pass or a pass through the Floorplan Manager Location Based Optimization (LBO). If the magnitude of the violations is large then another budget pass using PrimeTime Budget is in order, otherwise we use Floorplan manager LBO. It is expected that the mapped design resulting from the third pass is ready for layout, if not, then detailed analysis of the failing paths is necessary to arrive at a solution.

2.3 Layout Timing Closure

After Place & Route and Clock Tree Synthesis is completed, equivalency check is run to verify the design transformation. Once the new verilog net list is verified, extraction data is used to create a SDF file with the foundry delay calculator. The timing is checked using PrimeTime. If violations exist, analysis as to the best tool to fix the problem must be determined. Once a good design is achieved, test vectors are created and simulated to make sure there are no problems. After this step the chip is ready for final sign-off. As shown in Figure 3. on page 7 this loop can be quite iterative and very time consuming. The analysis of the data and the decisions should not be taken lightly. Experience and experimentation is the key to success with the timing closure loop.

Figure 1. RTL to GTECH

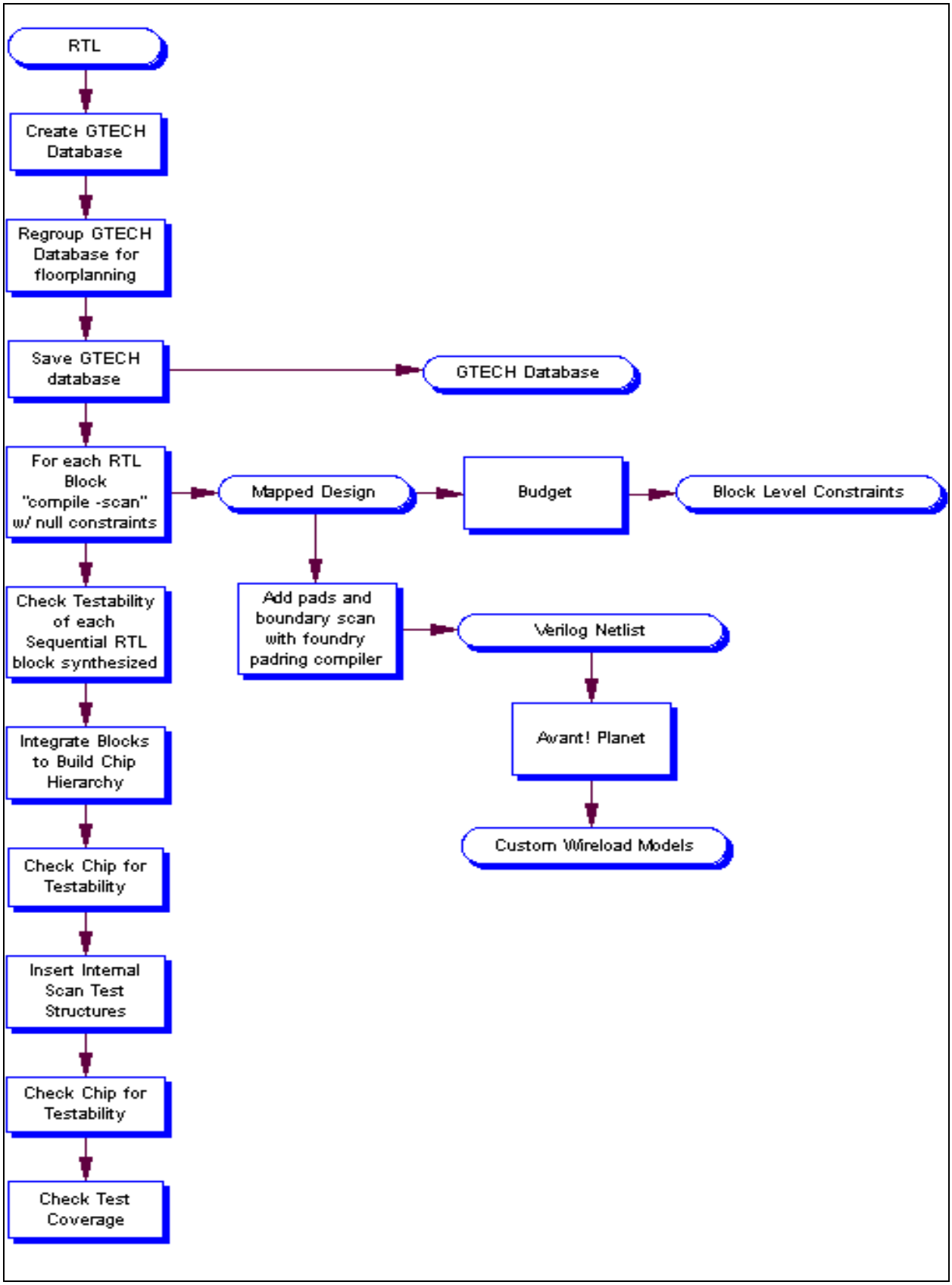


Figure 2. GTECH to Layout

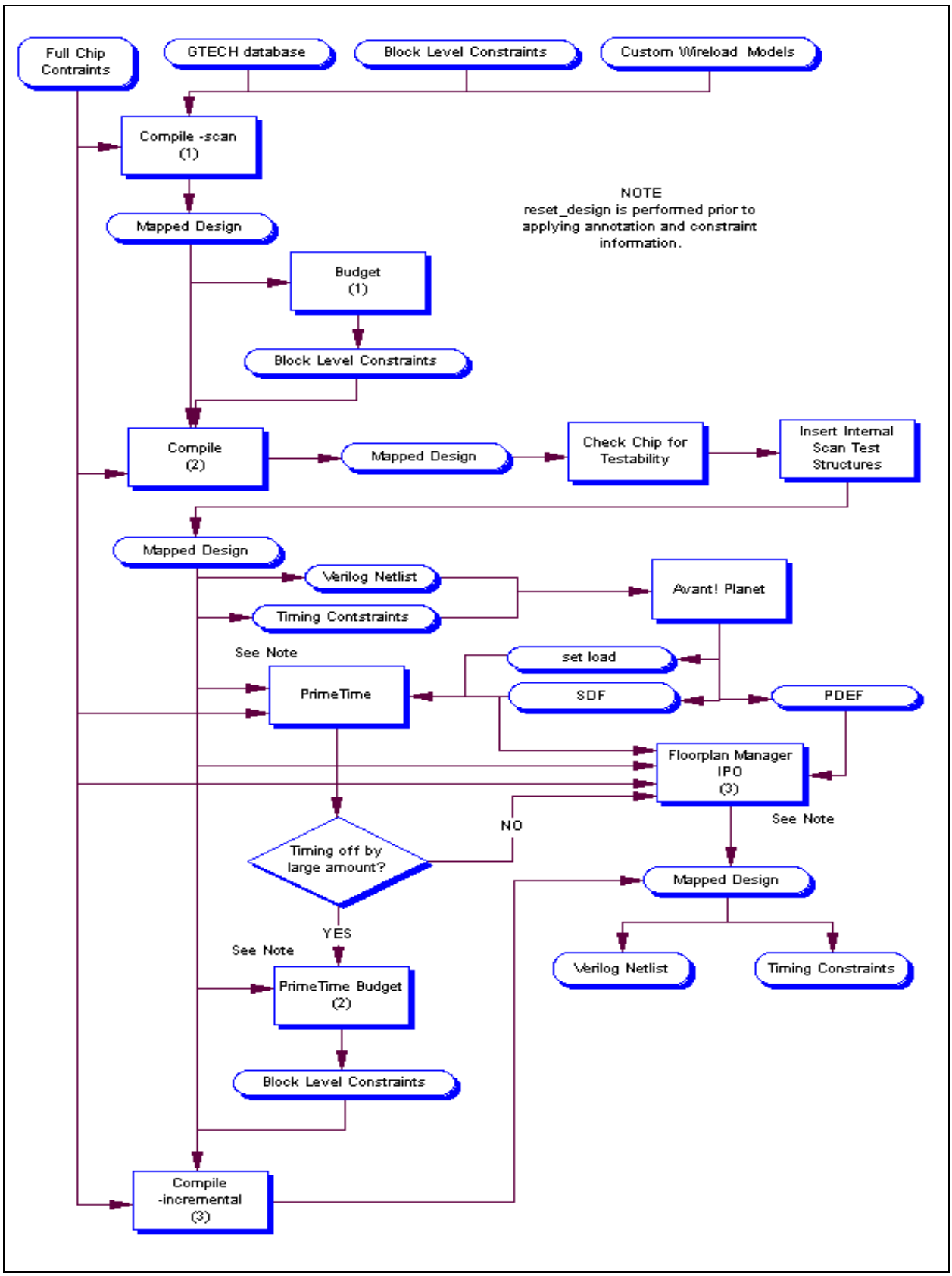
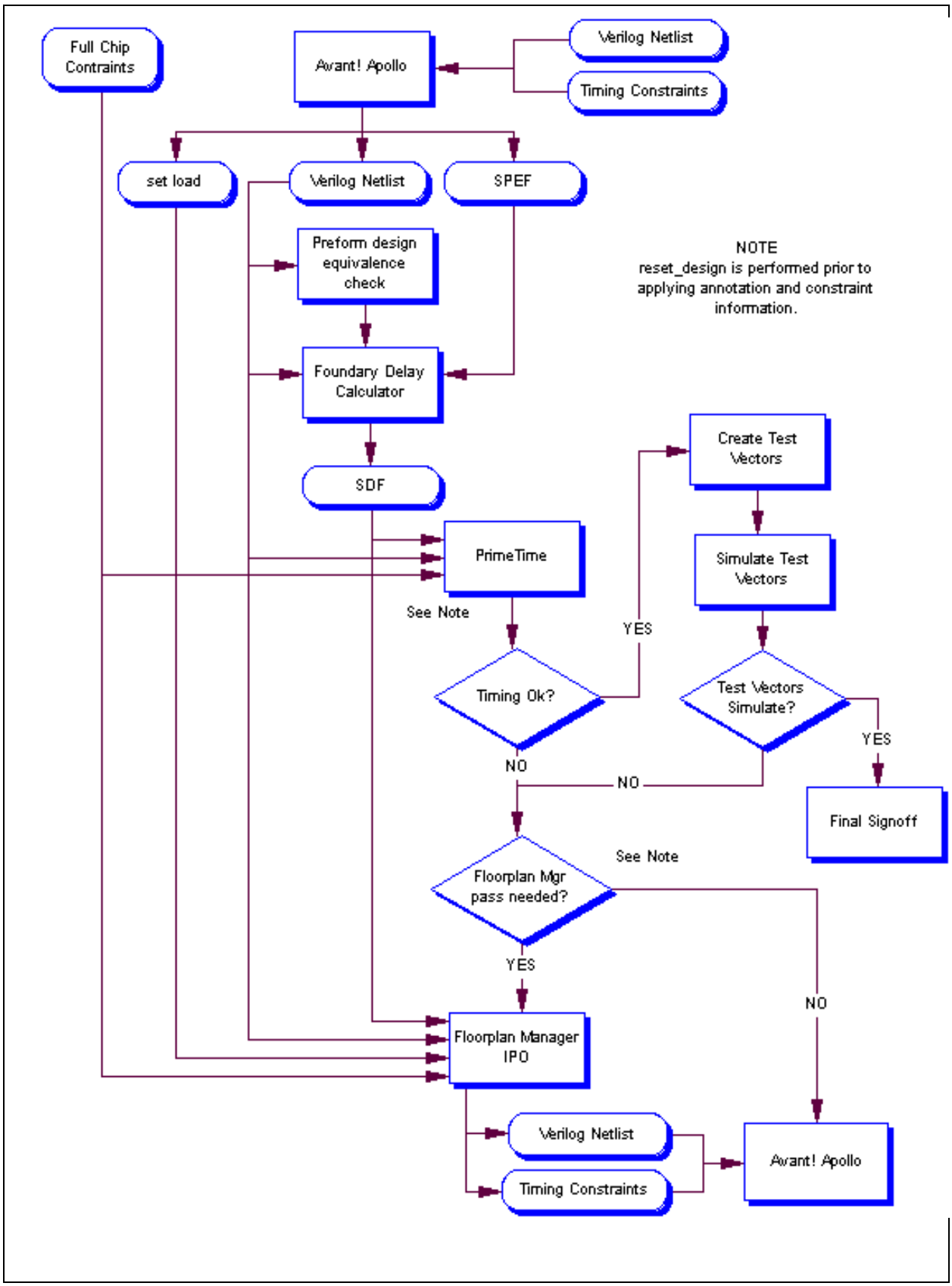


Figure 3. Layout Timing Closure



3.0 How AFTEC works from RTL to Layout.

Our first attempts at automating the budget process were difficult. From reading the Synopsys manual it appeared you needed to make three passes through Design Compiler (DC). However, with our designs we needed to make 5 passes; we needed two passes to modify the GTECH based data and three more passes for the budgeting process. We first scripted the flow up in the normal way; many, many scripts in serial. This process took days to run and did not meet our goal of a complete synthesis turn in approximately 24 hours.

Our local AE indicated that Synopsys was working on ACS (Automated Chip Synthesis) but it would not be available for about 6 months. He also told us that another AE had built a Makefiles system to automate the budget flow. The next day we had that tool (called build_make) and used it as a baseline for our future enhancements. The build_make tool written in Perl, generates Makefiles and support scripts. This tool automated a portion of the GTECH to Layout flow; the portion had only to do with the budgeting process.

We also started with the build_make tool as it had many of the hooks necessary for the budget process. We modified the script to output TCL (dc_shell-t) scripts thus providing a common basis for future expansion. As you might have guessed writing a Perl script which outputs “make” and “TCL” can make ones head spin.

The next modifications were to have the tool output two types of scripts automatically. We identified two types of scripts; global scripts that would not be changed and generic scripts which were templates and would have to be edited by the user. The script output names were chosen to give the user a clue as to what needed to be edited, see Section 6.0 on page 19 for example of the output. A “Makefile.inc” file was generated to allow the user to change the names of the scripts and directories used by the “Makefiles.” Many TCL support scripts are written out to a tcl directory which was accessible to any of the scripts.

We use GNU make to process the Makefiles. GNU make as many are aware from past SNUG papers can run concurrent jobs. We exploited that feature on our dual and quad processor systems to greatly improve run-times. We did not have access to LSF or similar queuing programs, therefore the only compute based optimization attempted was on the same multiple processor machine.

3.1 Budget Flow Improvements.

In this section we detail several improvements we made to the baseline build_make tool, which allowed us to arrive at our own tool; ASIC Flow Engine for Timing Closure (AFETC). Key among these was the addition of the RTL parser, the GTECH restructuring and the generation of all the template files.

There are several shortcomings we discovered in the baseline build_make tool. Most of them had to do with the fact we were automating a complete flow not just the budgeting process. First it didn’t handle RTL code conversion to GTECH database. We added that feature by using the Rough Verilog Parser (a Perl Module), see Section 5.1 on page 18, which would give us our design hierarchy by reading in the Verilog. Using a front end parser like the RVP would allow a

design group the ability to use VHDL as the input language as well as Verilog. The group that developed SMART, see Section 5.1 on page 18, assumed that everything could be compiled in the RTL directory. In our case, we included card and ASIC data in the same rtl directory. Once we had the RTL hierarchy we could easily do incremental re-compiles of the design, or so we thought.

The next obstacle was the fact that our designers were used to working on fine grained RTL, that is RTL which represents less that 5K gate blocks. With the new capacity of DC99.05 we could easily do 20K to 50K gate blocks. To use DC99.05 efficiently we wanted to restructure the GTECH design into larger blocks.

During the design restructuring we realized an added benefit in the physical domain. Our floorplan tool (and Place and Route tool) worked better with bigger blocks. This allowed us freedom to experiment in the physical realm without it impacting the development team. It was very important to keep the logical design team focused on the verification effort. In doing the restructuring we inadvertently broke our ability to map the logical RTL to the physical GTECH. This may look like a drawback but we discovered that our overall run times were approximately a day, so we accepted the inconvenience.

With all the data manipulation going on we quickly had a base budget flow with 5 steps:

- Pass (-2) RTL to GTECH mapping of logical design
- Pass (-1) GTECH to optimized for physical design
- Pass (0) Quick Synthesis with mapped gates to get a baseline budget.
- Pass (1) Real top level constraints with pass 0 budgets to arrive at a workable design.
- Pass (2+) Improvements on top level constraints and incremental improvements of design.

3.2 The Improvements - Links to Layout.

As with any methodology this one kept on growing. Our next hurtle was to tie in the floor-planning tool; we use Avant! Planet. At first we just wanted to use the wireload model from the tool; then we discovered Synopsys Floorplan Manager (FPM) which is used for Location Based Optimization (LBO) before (and after) we pass the design to detail place and route (P&R).

We decided that we needed a more accurate wireload model for our design than the standard foundry provided linear model. Wireload models are very interesting as pointed out in the paper, “Resistance is Futile, building better wireload models,” see Section 5.3 on page 18. We didn’t want to go through the design flow twice to use the wireload model; once with the foundry model and a second time with our placement based model. It was decided to use the output of the Quick Synthesis to build a wireload model. The Quick Synthesis mapped design is very fat, non optimal design with most of the internal interfaces. This is a good starting point for the wireload models. The wireloads are good as estimates and it is a real pain to determine if they are helping or hurting your design.

After running the standard budget process we then went to the floorplan tool again to provide us with placement based SDF and PDEF files. These files were then used within PrimeTime to

determine if this run we needed another iteration. Now we could run the design budgeter from within PrimeTime to generate yet another set of constraints. Later we discovered we could get better optimization based on a design which had just missed timing using FPM.

Our next improvement was to use FPM for LBO. We chose to use it both before and after detail P&R to improve the design. Our basis for using FPM before going into layout was to give our in-house Layout group a optimal design from the very beginning in an effort to reduce the number of ECOs that we processed.

The hook into FPM gave us a clean link to timing closure. Our biggest problems were the block level interconnects. Many of the ESNUG articles over this last year have highlighted the problems with DSM designs and the interconnect. We chose to stick with our detailed routing capabilities with Avant! to handle the top level interconnect. The ability to use FPM to forward annotate placement data was critical to our success.

As in most designs even the best tools still need help. Every global script that we used has hooks into the script to allow the designer to change the constraints and the compile options. These were setup on a block by block basis by using a simple convention. If a file called “my_module.scr” is found in the working directory just before the compile step of the “dc_global.dct” script it is used in place of the baseline compile and write commands. This hook allowed engineers to fiddle with the constraints, change the compile options, add loads and anything else they wanted to do. In fact they could “reset_design” and start over; but the flow has proven sound on our designs.

3.3 The Reports

Our philosophy is that, you can never have too many reports. Well that is not quite true. You need reports that tell you about the design. Throughout every phase of using DC we use the QOR (Quality of Results) report to see if we are closing in on our constraints. It is interesting to note that for every pass through DC the QOR reports for the most part, improved. If they didn’t improve we had setup something wrong. The QOR reports were often used around an optimization to see what improvements the tools made. Therefore we often had “pre_” and “post_” files to review.

Other reports that were of interest to the engineers included:

- Latch and Loop reports which we used to detect possible HDL coding problems.
- Area reports for each block and the chip level; these were converted into a web page to allow easy viewing.
- Timing violators report from PrimeTime for the following cases: “pre_floorplan,” “post_floorplan_pre_layout,” “post_layout” and “post_extraction” were the most common although many others were possible.
- Log files which were parsed with logscan, see Section 5.3 on page 18. To filter the output of each run.

To keep things simple all the log messages were written to one log directory. This allowed easier filtering of the data.

3.4 Directory Structure

The directory structure took a while to develop; it began to gel when we discovered that each transformation of the data was to be considered a pass. One of the biggest changes to the build_make tool was the reorganization of the directory structure in which the db files were stored. The build_make tool had all the DC runs put data in the same directory, which made sense at first because it was using the DC directory search order to find the data. After awhile we discovered that MAKE was re-compiling things it shouldn't have. This had more to do with our use of the write command within the DC scripts then it did on any data dependencies. The solution was to put each resulting pass into it's own directory.

We discovered later as we added the Floorplan tool and the LBO phases that this same approach applied to the floorplan and layout tools as well. In Figure 4. on page 12 is shown what major directories were used and how the data moved through them.

Now this seems like a lot of data to keep around; in reality it is not that much data when you consider that at any point in the flow we could go backwards to what created the data. How many times have you done an incremental DC optimization just to see the results get worse, but you wrote over your current db? Using this organization it was easy to go backwards and start over.

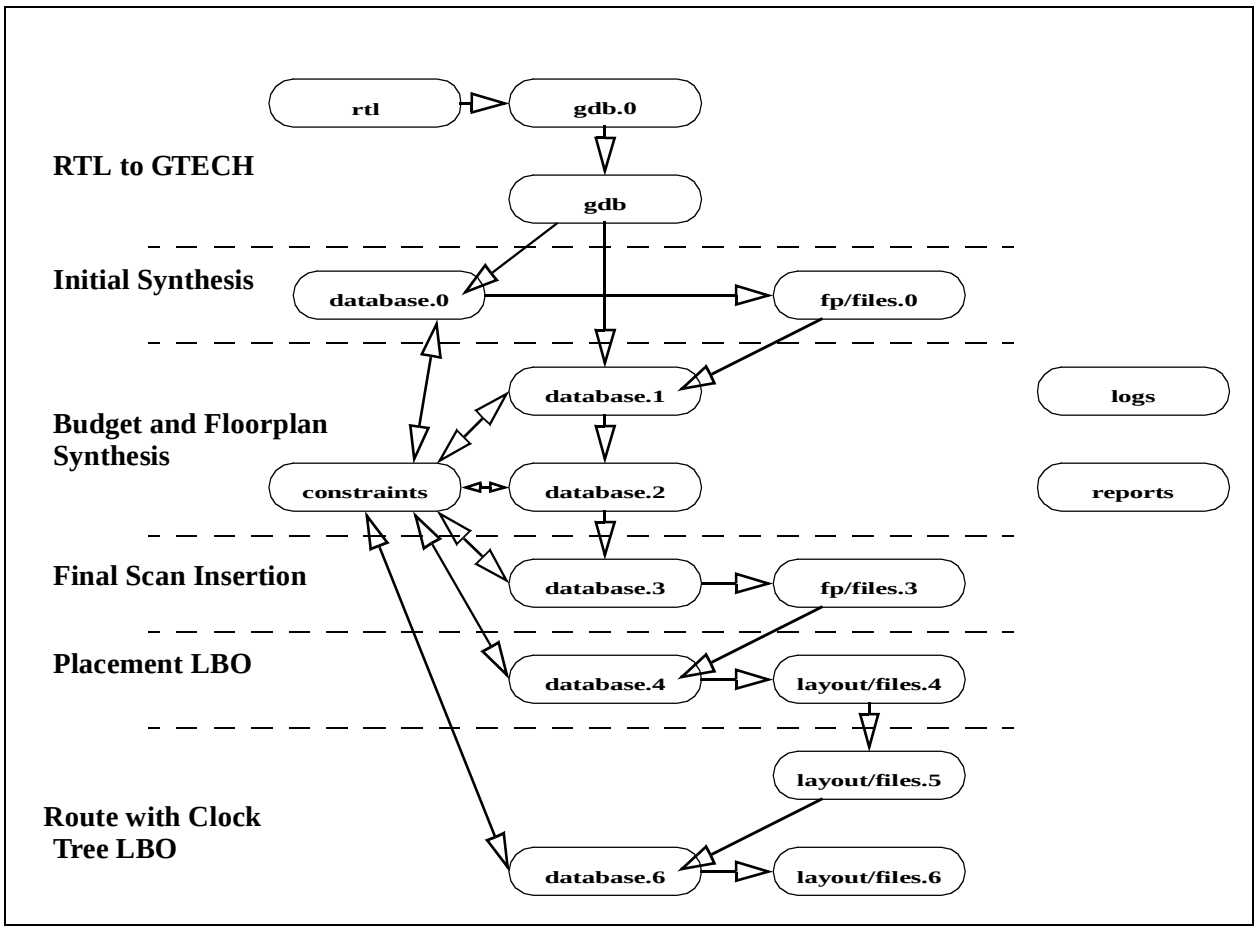
An interesting side effect was the fact that as we transformed the data the pass numbers just keep growing. The pass number was a MAKE variable which was easy to change on the command line. This allowed the same Makefiles to drive multiple iterations. This also allowed the DC scripts to use the pass number to determine what type of optimizations to do.

From Figure 4. on page 12 it became apparent there were several well defined operations the resulting output from AFETC had to address. The operations are broken down into:

- RTL to GTECH - during this operation the RTL code is mapped to GTECH and then converted into a more physical friendly form. The resulting database (gdb) was used as the basis of future optimizations
- Initial Synthesis - took the "gdb" and performed a simple technology mapping. This mapping was used to create the first budget and also the custom wireload models. The data also was used for test validation.
- Budget and Floorplan Synthesis - is the meat of the operations. This takes the budgets from database.0 and the floorplan and starts over with the gdb.
- Final Scan Insertion - is completed just before the net list is re-evaluated by the floorplanner. This later time frame is chosen to optimize the synthesis time spent on functional paths.
- Placement LBO - the placement only data to improve the large inter-block buses.
- Route with Clock Tree LBO - the timing closure loop.

From looking at this figure, it becomes apparent that whenever the net list took on a new form the pass number was incremental. For example database.3 data was used by the floorplanning tool to generate a pre-route SDF and PDEF file. Then the database.3 data was used by either FPM or

Figure 4. Directory Names and Data Flow



PrimeTime to develop database.4 data. The act of floorplanning at this level did not create a new net list. But, in the case of the layout/files.4 data the Apollo P&R tool took the database.4 data and inserted the clock tree (among other things) this represents a change to the net list; which resulted in layout/files.5 directory. All these numbers seem a bit overwhelming at first but after considering net list transformations it becomes easier to think about.

The numbering system for directories was a nice way to organize the data. The log, report and constraint directories have a different method. For each file in these directories the following approach was chosen. The file would be constructed with the form:

<module_name>.<prefix>.<pass>.<type>.

Where:

- <module_name> represents the design which created or was to operate on the data.
- <prefix> was a special code passed into the script to assist in identification of the report or log. For example: “pre_floorplan” or “post_floorplan_pre_layout.” The prefix names were designed to convey meaning to the reader.

- <pass> is the now famous pass variable of the data.
- <type> could be a report (rpt), log file (log) or a constraint (dcc).

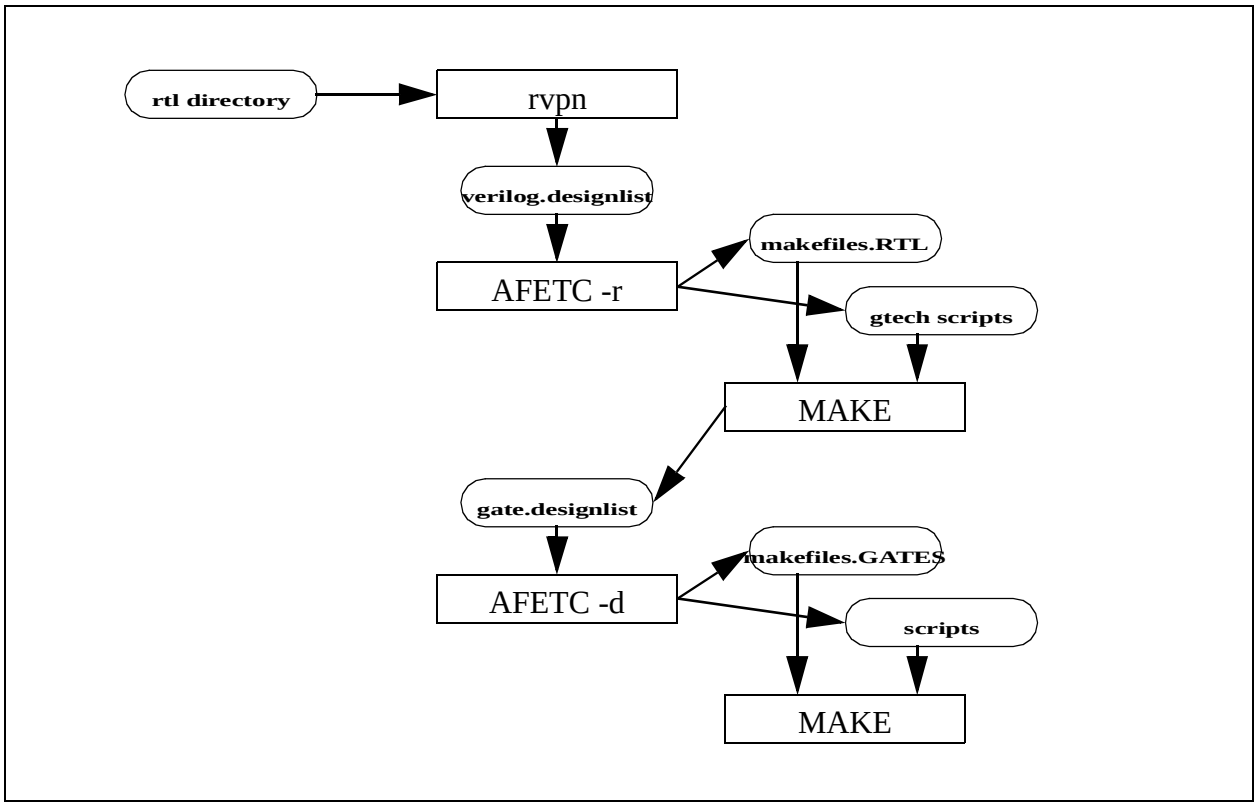
The constraints were quite interesting in that the constraint files written from the design budgeting tool “db_shell” were written in standard DC Shell language. Since we based all of our scripts on “dc_shell-t” mode we needed to convert the results of the budget before using them in our scripts. Make provides a method of implicit rules which make short order of this requirement.

The constraints are generated by the budget shell or by PrimeTime (budget process). During the budget process the constraints are created for the next pass. For example if we are operating on database.1 (PASS=1) during the budget process the constraints are written with PASS=2 in the file name. That is they are targeted to be used by PASS=2 operations. During the next run of the synthesis process with PASS=2 the “dc_global” script is expecting the constraint file to be named properly.

3.5 An Example of Running the Tool.

Running the AFETC tool is quite easy after you have some basic information in place. The AFETC tool is called twice to generate all the supporting scripts and Makefiles. Shown in Figure 5. on page 13 are the inputs used by AFETC and the calling sequence. AFETC is called once after the RTL designlist is generated and again after the GTECH database is built and the gate level designlist is available. From then on MAKE is used to perform the work.

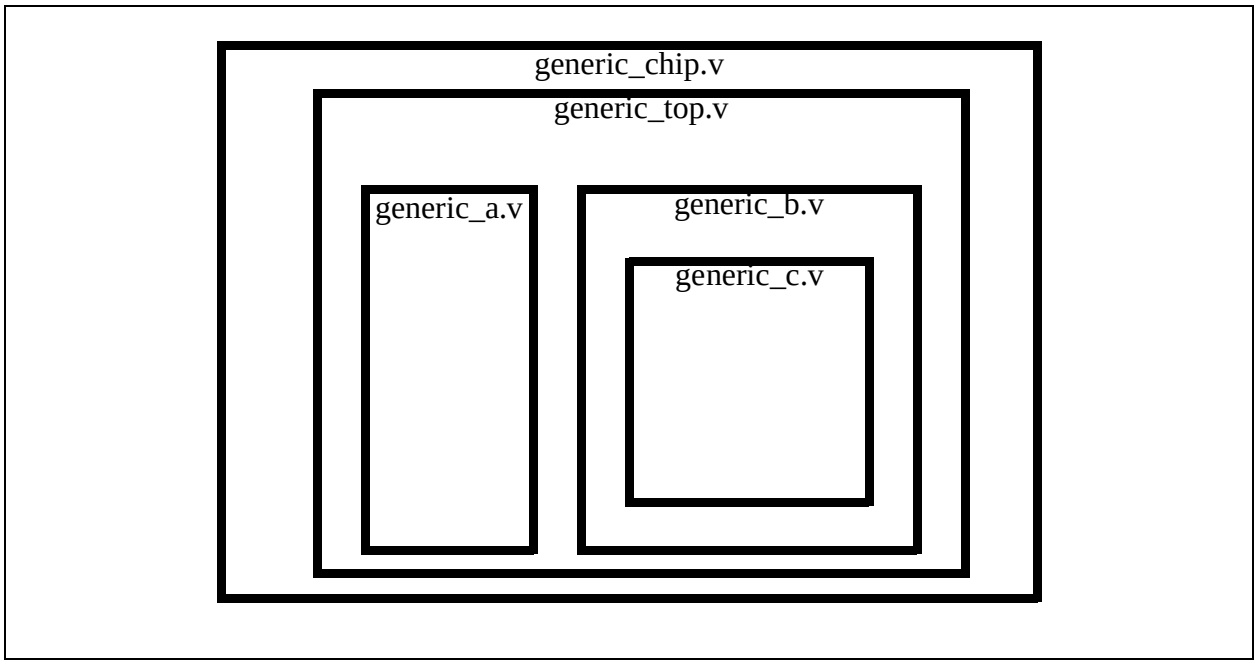
Figure 5. AFETC Calling Sequence



The design of the environment was based on the chip model shown in Figure 6. on page 14. Where each block contains the following types of logic:

- generic_chip.v - contains, pads, JTAG cells and the instantiate the top.
- generic_top.v - contains the real core logic and accepts only input and outputs but no bi-directional signals. This was due to allow better control over the constraints.
- generic_a.v, generic_b.v and gerneic_c.v are lower level modules.

Figure 6. Generic Chip



Once the user had a simulating design and is ready to start the synthesis the first step is to generate the hierarchy file used by the tool by invoking the “rvpn” tool the command line for the generic design is:

```
rvpn -t generic_top../blm/generic_top.v ../blm/generic_a.v ../blm/generic_b.v ../blm/
generic_c.v > generic_top.verilog.designlist
```

A *.v could have replaced all the verilog files in this example. Next run the AFETC tool with the designlist (see Section 6.0 on page 19 for listing of designlist file) to generate baseline scripts and the “Makefile.RTL” using the command:

```
AFETC -r generic_top.verilog.designlist
```

The tool responds with what it is doing, see Figure 7. on page 15, which shows that the designlist was processed to which level of hierarchy it considered important. The level can be changed by using the -l option to the AFETC tool. Next the tool informs the user of what global and generic level scripts it is generating. The listing for many of the scripts can be found, see Section 6.0 on page 19.

Figure 7. Results of running AFECT

```
Using Designlist: generic_top.verilog.designlist
Info: Level 1:
generic_top
Info: Level 2:
..generic_a
..generic_b
Info: Level 3:
....generic_c
Building RTL Specific Makefile: Makefile.RTL
Creating DCT script file gt_global.dct
Creating DCT script file gth_global.dct
Creating DCT script file dc_global.dct
Creating DCH script file dch_global.dct
Creating Initial constraints script file generic_top.initial.dct
Creating DB script file db_global.dct
Creating Layout script file generic_layout.dct
Creating Fp script file generic_fp.dct
Creating PT script file generic_pt.dct
Creating Area script file generic_area.dct
Creating Test Compiler script file generic_tc.dct
Creating Generic Compile Script file generic_compile.scr
Creating Generic Wireload TCL file generic_wireload.tcl
Creating Ungroupd DesignWare TCL file ungroup_dw.tcl
Creating Generic Chip Compile file generic_chip.dct
Creating Generic FPM Compile file generic_fpm.dct
Creating Makefile include file Makefile.inc
This should be edited before running make
```

After you have run AFETC the first time, the tool is designed to not replace the script files it finds in the working directory. This was done to allow the user to edit the files in place while experimenting with the tool. The Makefiles (and two auxiliary files the Makefile.RTL and the Makefile.GATES) are always rebuilt by AFETC, but provides the user the ability to write the Makefiles to another name via the “-m” command line switch.

The tool requires that a top level constraint file be used for the budgeting process. If the user is unsure they can copy the <top>.initial.dct over to <top>.dct and edit it to make the first pass at the initial constraints. From the point that the user has good top level constraints the files should be maintained in a revision control system and placed in the “Makefile.inc.”

The user then runs their first make command: “make setup” which builds several place-holder directories and a few links for the constraint files. The following directories are built at this phase:

- fp - for the floorplan data.
- layout - for the layout data.
- log - for the log messages.
- reports - for the resulting reports.
- database.# - initial created for the first 2 passes
- gdb.# - for the GTECH design.

The user is now ready to run the first synthesis on the design. The user starts the synthesis with "make all_gtech". If they want to run with multiple jobs they use the --job=# switch if they are using GNU Make. Now we have made a big assumption that the user has their environment setup properly from the synopsys point of view. The user must provide a ".synopsys_dc.setup" file which is valid for their design environment, and supports DC TCL mode.

The next operation is to link the design and build the hierarchical level of the database. This is only a requirement if the user intends on changing the hierarchy of the GTECH design. For this design we don't plan on changing the hierarchy so the "gdb.0" and "gdb" directories are the same. This command calls the "output_hier" tcl function to dump a new designlist called "generic_top.designlist." This designlist could be different if the hierarchy was modified at this step. The AFETC tool is then run with this designlist to produce the "Makefile.GATES" file which will be used to compile the gates. The command is "AFETC -d generic_top.designlist."

As noted before the scripts will not be overwritten, but a new note about building the gate level makefile is displayed. We are now ready to map the design with DC using the "Makefile.GATES." For the impatient engineer use the command "make budgets.0 PASS=0," which would compile the design bottom-up and run the initial budget pass. But, if you want to see more of the steps use the "make all_hier PASS=0" and see what happens. You may note that the design compiles from bottom-up and works just fine. If you have GNU Make running multiple jobs it makes a mess of the display but gets the work done quickly. If you ran with the budget.0 command you would have built all the lower level modules, a hierarchal design and completed the first budget pass on this design. The constraint files are located in the "constraint" directory ready for the next pass. In the next pass you would do a "make budgets.1 PASS=1" and the process would continue.

The AFETC tool is invoked twice because we believed that most designs will experience a change from logical to physical hierarchy. The first time the tool is invoked it uses the output from the "rvph" Rough Verilog Parser Hierarchy. The "rvph" parses the RTL code to understand the hierarchy thus the dependencies within, when AFETC is run, produces the Makefiles and the Makefile.RTL. The second time it is invoked it uses a list created after the complete GTECH database is linked. A TCL script, "output_hier," is called when the GTECH hierarchy is stitched together (in it's final form) The TCL script "output_hier" generates the proper output format for AFETC. The second invocation of AFETC writes out the Makefile.GATES. The use of separate Makefiles allow the design team the flexibility with an eye towards design productivity.

Once you have the first net list in database.0 it is a good time to check your test coverage and if your pad-ring will connect to your design. If you intend to floorplan the design this is a good time to test your floorplan methodology. This is when the user will run into scripts that they need to maintain as each tool vendor is different. The use of the option "make -n" is a valuable tool in driving out script dependencies. If the user see's "FIXUP" then the file needs to be edited in the "Makefile.inc" and most likely the generic script needs to be modified. It is best if the generic is given a specific name and the "Makefile.inc" points to that name.

4.0 Conclusions

AFETC as developed has worked on one very large project at the time of this paper.

Our goal from the beginning was to use the extra capacity and capabilities of Design Compiler to drive our ASIC quicker to timing closure. The first requirement was to get the design from RTL to Layout in about 24 hours. This was accomplished by creating blocks that matched the capacity of Design Compiler and exploiting the parallelism of GNU make.

The second requirement was to allow AFETC to be extendable to other design groups. Two ASIC projects are just starting to use the tool. Hooks are in place to make this succeed.

The timing closure loop is very tool intensive with many decision points. AFETC was developed to generate data that would enable the design engineer to make decisions and use the tool to act upon them. We have succeeded in our quest to uncover problems and develop solutions to the timing closure issue. The AFETC tool provides the hooks necessary to allow a variety of EDA tools and approaches to the timing closure issue with a focus on using Design Compiler, PrimeTime and Floorplan Manager.

The key to the success of the tool is the flow that was developed along with the tool which should work for at least another technology generation of foundry silicon.

4.1 Recommendations

A GUI build to setup the tool would be very handy but as of this time is unnecessary since using the “rvpn” tool works from the Verilog source to drive the process.

Build a VHDL parser like “rvpn” to support VHDL.

The Makefiles should recursively call itself to arrive at a desired endpoint. For example if the command “make PASS=2 chip” is issued the Makefiles should be smart enough to know if PASS=0 and PASS=1 have been run in order to generate data needed by PASS=2.

The inclusion of some of the design engineer support tools like the Planet floorplanner should be automated. This has more to do with our understanding of scripting in Planet than any tool limitation. This issue could be solved with a little more time, but who has that?

The use of LSF or similar queuing program needs to be investigated. As at many companies this is a cultural issue more than a technical one.

With all tools this one cannot address everyone’s needs. Specifically it does not address the needs of the FPGA designers. Although they were considered for inclusion in this tool the widespread differences in the tools suites would have caused great difficulty for everyone involved.

5.0 Acknowledgments

I could not have done it without the contribution of my co-author Marvin Anderson who often pointed out rough spots in the flow and forever helped me in understanding the details of the test issues.

The ASIC Technologies group; Walt, Marvin, Roger and Rob for pointing out why what I did broke the downstream tools yet were still willing to find work-arounds.

The processor design team; Craig, Tony, Christina, Galen, Linda, David, Drew, Carlos, John and Roger who endured many fits and starts with the tool as the first users. A special thanks to Maurice who understood what a challenge it was to produce this tool and was forever a sounding board for my ideas.

The layout group; Tony, Dave and Richard whose input into what the Avant! tools needed was invaluable to putting in the timing closure loop.

Finally my business partner (and wife) Terry Tessier for reviewing this dry boring EDA paper more times then she wants to consider.

5.1 Third Party Tools

Costas Calamvokis (v2html@iname.com) the developer of the Rough Verilog Parser and v2html converter which makes short order of determining the hierarchical order of the RTL code so we could add the hooks in for RTL to GTECH.

David C. Black the developer of LOGSCAN for recognizing the shear volume of data that EDA tools generate and arrived at a working solution to filtering the data for us humans.

5.2 Synopsys

Our local Synopsys AE Jeffrey Flieder (flieder@synopsys.com) who provided guidance and assistance to getting the Synopsys to Avant! timing closure flow functioning.

A special thanks to Jon Baldry former Synopsys RTL Synthesis CAE who provided the idea and the baseline tool from which our tool sprang. We have leveraged his designlist concept and perl scripts to parse in numerous ways.

Synopsys for documenting everything, even if it is hard to find, in the on-line manuals. Once you know you need the information you can find it. Too bad you are often looking after an unexpected result:-(

5.3 SNUG's Past

Resistance is Futile!, Building Better Wireload Models, Steve Golson presented at SNUG San Jose 99.

SMART 2.0: The Makefile That Ate My Brain, Rodney Ramsay presented at SNUG San Jose 99.

6.0 AFETC Generated Files of Interest

The AFETC tool generates many files. In the paper we give an example of a simple verilog design which we parsed with AFETC. The following sections provide the files both used as input and output for the AFETC program that are fundamental to its success.

6.1 The Designlist File

The designlist file has a very simple format. For each comma separated line contains the parent followed by all of its children in the design hierarchy. When a parent has no children it is just listed by itself. This file is free form as the AFETC tool has the ability to find the top of the tree. This file was created with the command:

rvpn -t generic_top ../blm/generic_top.v ../blm/generic_a.v ../blm/generic_b.v ../blm/generic_c.v > generic_top.verilog.designlist

Figure 8. The "designlist" file for the generic example

```
# This report was automatically generated by bin/rvpn
#  Creation date: Mon Dec 13 13:15:31 1999
generic_top, generic_a, generic_b
generic_a
generic_b, generic_c
generic_c
```

6.2 Makefiles

This section contains the sample Makefiles created by the AFETC tool on the simple verilog example.

Figure 9. The root Makefile

```
## Makefile generated by bin/AFETC \
## -d\
## generic_top.designlist
##
## This file was auto-generated
## by AFTEC Generated Script - Edit with care
##
## All edits here may be overwritten
## Edit the include file to setup design and system parameters to keep
## I promise to honor them

usage ::
    echo "Usage of this Makefile"; \
   echo "  This Makefile is automaticly generated with the AFETC tool."; \
   echo "    The AFETC tool expects the user to have created a "; \
   echo "    <module>.designlist from the GTECH database using the ??? funciton." ; \
   echo "    A simple <module>.designlist is shown below.  With this example a user ";
\
   echo "    could use the AFETC tool to generate a template for generating "; \
   echo "    the GTECH design which includes the call to the ??? function. "; \
   echo "  Once the user has the <module>.designlist file the AFETC tool is "; \
   echo "  used to generate all the supporting scripts and this Makefile."; \
   echo " "; \
   echo "  The user is expected to supply a top level timing constraint file written
"; \
   echo "  in the dc-Tcl language which is compatible with PrimeTime. The filename is
"; \
   echo "  <module>.con and should be located in the syn directory.";
```

Figure 9. The root Makefile

```
include Makefile.inc

## Now to do some checks for directories and stuff
${RTLPATH} ::
    echo "Information: Cheching for directory $@";
    if [ ! -d $@ ]; then mkdir $@; fi

${GDBPATH}.0 ::
    echo "Information: Cheching for directory $@";
    if [ ! -d $@ ]; then mkdir $@; fi

${GDBPATH} ::
    echo "Information: Cheching for directory $@";
    if [ ! -d $@ ]; then mkdir $@; fi

${DBPATH}.0 ::
    echo "Information: Cheching for directory $@";
    if [ ! -d $@ ]; then mkdir $@; fi

${DBPATH}.1 ::
    echo "Information: Cheching for directory $@";
    if [ ! -d $@ ]; then mkdir $@; fi

${DBPATH}.2 ::
    echo "Information: Cheching for directory $@";
    if [ ! -d $@ ]; then mkdir $@; fi

# if more than two passes are necessary setup those directories.
${DBPATH}.${PASS} ::
    echo "Information: Cheching for directory $@";
    if [ ! -d $@ ]; then mkdir $@; fi

${REPSPATH} ::
    echo "Information: Cheching for directory $@";
    if [ ! -d $@ ]; then mkdir $@; fi

${CONSPATH} ::
    echo "Information: Cheching for directory $@";
    if [ ! -d $@ ]; then mkdir $@; fi
    cd $@; ln -s ../${TOP_DESIGN}.${TCONEXT} .; cd ..

${LOGSPATH} ::
    echo "Information: Cheching for directory $@";
    if [ ! -d $@ ]; then mkdir $@; fi

${LAYOUTPATH} ::
    echo "Information: Cheching for directory $@";
    if [ ! -d $@ ]; then mkdir $@; fi

${FLOORPLANPATH} ::
    echo "Information: Cheching for directory $@";
    if [ ! -d $@ ]; then mkdir $@; fi

## This is a simple rule to get the search_path
## and link_path to db_shell
vlsidiv_dct.setup :: vlsidiv_dc.setup
    ${DCTXSCRIPT} $< $@; mv $@ tmp.tdct; sed "s/^echo/#echo/" tmp.tdct > $@;

## simple rule to ensure that the constraints are modified
##  to something that works for DCT
${CONSPATH}/%.${TCONEXT} :: ${CONSPATH}/%.${CONEXT}
    ${DCTXSCRIPT} $< $@; mv $@ $(@F).tdct; sed "s/^echo/#echo/" $(@F).tdct > $@; rm
$(@F).tdct

.INIT :: \
  ${LOGSPATH} \
  ${LAYOUTPATH} \
  ${FLOORPLANPATH} \
  ${CONSPATH} ${REPSPATH} \
  ${DBPATH}.0 ${DBPATH}.1 ${DBPATH}.2 \
```

Figure 9. The root Makefile

```
  ${GDBPATH}.0 \
  ${GDBPATH} \
  ${INITCONS} ${DCTSCR} ${DBSCR} vlsidiv_dct.setup \
  ${TOP_DESIGN}.${TCONEXT}

setup :: .INIT

clean_gtech ::
    rm ${LOGSPATH}/*_gtech* \
     ${REPSPATH}/_gtech*;

clean ::
    rm ${LOGSPATH}/* \
    ${REPSPATH}/* \
    ${CONSPATH}/*.[0-9]*.${CONEXT} \
    ${CONSPATH}/*.[0-9]*.${TCONEXT} \
    ${CONSPATH}/*.[0-9]*.${CONEXT}.chk \
    ${DBPATH}.0/*.${DBEXT} \
    ${DBPATH}.1/*.${DBEXT} \
    ${DBPATH}.2/*.${DBEXT};

## Now for the serious stuff
all_hier_gtech :: ${GDBPATH}/${TOP_DESIGN}_hier.${GDBEXT}

all_gtech :: ${GDBPATH}.0/${TOP_DESIGN}.${GDBEXT}

all_hier :: ${DBPATH}.${PASS}/${TOP_DESIGN}_hier.${DBEXT}

all :: ${DBPATH}.${PASS}/${TOP_DESIGN}.${DBEXT}

chip :: ${DBPATH}.${PASS}/${CHIP_DESIGN}.${DBEXT}

verilog :: ${CHIP_DESIGN}_gates.${PASS}.v

2layout :: ${LAYOUTPATH}/${CHIP_DESIGN}.${PASS}.v

2fp :: ${FLOORPLANPATH}/files.${PASS}/${CHIP_DESIGN}.v

scan :: ${DBPATH}.${PASS}/${CHIP_DESIGN}_scan.${DBEXT}

## Top design for GTECH
${GDBPATH}/${TOP_DESIGN}_hier.${GDBEXT} :: ${DCGSCR} ${GDBPATH}.0/
${TOP_DESIGN}.${GDBEXT}
    ${DCTSH} -x "set TDT_des  ${TOP_DESIGN};\
       set TDT_gtech_hier ${GTECHIER};\
       set TDT_gdbdir ${GDBPATH};set TDT_gdbext ${GDBEXT};\
       set TDT_repdir ${REPSPATH};"\
       -f ${GTHSCR} | tee ${LOGSPATH}/${TOP_DESIGN}_gtech_hier.${LOGEXT}

## Top design for budgeting
${DBPATH}.${PASS}/${TOP_DESIGN}_hier.${DBEXT} :: ${DBPATH}.${PASS}/
${TOP_DESIGN}.${DBEXT} ${DCHSCR}
    ${DCTSH} -x "set TDT_des  ${TOP_DESIGN};\
       set TDT_dont_use_cells ${XCELL};\
       set TDT_fpdir ${FLOORPLANPATH};\
       set TDT_custom_wireload ${WIRELOADS};\
       set TDT_foundry_lib ${FOUNDRY_LIB};\
       set TDT_pass ${PASS};set TDT_consdir ${CONSPATH};\
       set TDT_dbdir ${DBPATH};set TDT_dbext ${DBEXT};\
       set TDT_gdbdir ${GDBPATH};set TDT_gdbext ${GDBEXT};\
       set TDT_repdir ${REPSPATH};set TDT_conext ${TCONEXT};"\
       -f ${DCHSCR} | tee ${LOGSPATH}/${TOP_DESIGN}_hier.${PASS}.${LOGEXT}


## Top Level Timing Report
timing.${PASS} :: ${DBPATH}.${PASS}/${TOP_DESIGN}_hier.${DBEXT}
    ${DCTSH} -x "set TDT_des  ${TOP_DESIGN};\
       set TDT_pass ${PASS};set TDT_consdir ${CONSPATH};\
       set TDT_dbdir ${DBPATH};set TDT_dbext ${DBEXT};\
       set TDT_repdir ${REPSPATH};set TDT_conext ${TCONEXT}; read_db $<;\
```

Figure 9. The root Makefile

```
      redirect [format "%s%s"  [format "%s%s"  [format "%s%s" [format "%s%s" ${REPS-
PATH} {/}] ${TOP_DESIGN}] {.${PASS}}] {.tim.rpt}] { report_timing -max_path ${TIMING-
PATHS} -nworst 10 -path full -nosplit }; exit; "\
      | tee ${LOGSPATH}/${TOP_DESIGN}_timing.${PASS}.${LOGEXT} && touch tim-
ing.${PASS}


##
## Setup of Chip Specific Data; each chip is different which is why we have this sec-
tion here
##
${DBPATH}.${PASS}/${CHIP_DESIGN}.${DBEXT} :: ${DBPATH}.${PASS}/
${TOP_DESIGN}_hier.${DBEXT} ${RTLPATH}/${CHIP_DESIGN}.v
   ${DCTSH} -x "set TDT_des  ${CHIP_DESIGN};\
      set TDT_top $<;\
      set TDT_dont_use_cells ${XCELL};\
      set TDT_rtl ${RTLPATH};\
      set TDT_fpdir ${FLOORPLANPATH};\
      set TDT_custom_wireload ${WIRELOADS};\
      set TDT_foundry_lib ${FOUNDRY_LIB};\
      set TDT_pass ${PASS};set TDT_consdir ${CONSPATH};\
      set TDT_dbdir ${DBPATH};set TDT_dbext ${DBEXT};\
      set TDT_repdir ${REPSPATH};set TDT_conext ${TCONEXT};"\
      -f ${DCCSCR} | tee ${LOGSPATH}/${CHIP_DESIGN}.${PASS}.${LOGEXT}

## Chip Level Timing Report
chip_timing.${PASS} :: ${DBPATH}.${PASS}/${CHIP_DESIGN}.${DBEXT}
   ${DCTSH} -x "set TDT_des  ${CHIP_DESIGN};\
      set TDT_pass ${PASS};set TDT_consdir ${CONSPATH};\
      set TDT_dbdir ${DBPATH};set TDT_dbext ${DBEXT};\
      set TDT_repdir ${REPSPATH};set TDT_conext ${TCONEXT}; read_db $<;\
      redirect [format "%s%s"  [format "%s%s"  [format "%s%s" [format "%s%s" ${REPS-
PATH} {/}] ${CHIP_DESIGN}] {.${PASS}}] {.tim.rpt}] { report_timing -max_path ${TIM-
INGPATHS} -nworst 10 -path full -nosplit }; exit; "\
      | tee ${LOGSPATH}/${CHIP_DESIGN}_timing.${PASS}.${LOGEXT} && touch
chip_timing.${PASS}


## Chip level area and power
area.${PASS} :: ${DBPATH}.${PASS}/${CHIP_DESIGN}.${DBEXT} ${APSCR}
   ${DCTSH} -x "set TDT_des  ${CHIP_DESIGN};\
      set TDT_pass ${PASS};set TDT_consdir ${CONSPATH};\
      set TDT_dbdir ${DBPATH};set TDT_dbext ${DBEXT};\
      set TDT_repdir ${REPSPATH};set TDT_conext ${TCONEXT};"\
    -f ${APSCR} | tee ${LOGSPATH}/${CHIP_DESIGN}_area.${PASS}.${LOGEXT} && touch
area.${PASS}

## Chip Level Test Compiler Run
tc.${PASS} :: ${DBPATH}.${PASS}/${CHIP_DESIGN}_scan.${DBEXT} ${TCSCR}
   ${DCTSH} -x "set TDT_des  ${CHIP_DESIGN};\
      set TDT_pass ${PASS};set TDT_consdir ${CONSPATH};\
      set TDT_dbdir ${DBPATH};set TDT_dbext ${DBEXT};\
      set TDT_repdir ${REPSPATH};set TDT_conext ${TCONEXT};"\
    -f ${TCSCR} | tee ${LOGSPATH}/${CHIP_DESIGN}_tc.${PASS}.${LOGEXT} && touch
tc.${PASS}

## Write out the Verilog for the chip design.
${CHIP_DESIGN}_gates.${PASS}.v :: ${DBPATH}.${PASS}/${CHIP_DESIGN}.${DBEXT}
   ${DCTSH} -x "set TDT_des  ${CHIP_DESIGN};\
      set TDT_pass ${PASS};set TDT_consdir ${CONSPATH};\
      set TDT_dbdir ${DBPATH};set TDT_dbext ${DBEXT};\
      set TDT_repdir ${REPSPATH};set TDT_conext ${TCONEXT};\
      read_db $<; write -hierarchy -format verilog -output $@ ; exit" | tee ${LOGS-
PATH}/${CHIP_DESIGN}_gates.${PASS}.${LOGEXT}

## Write out the Scanned Verilog for the chip design.
${CHIP_DESIGN}_scan.${PASS}.v :: ${DBPATH}.${PASS}/${CHIP_DESIGN}.${DBEXT}
   ${DCTSH} -x "set TDT_des  ${CHIP_DESIGN};\
```

Figure 9. The root Makefile

```
      set TDT_top ${TOP_DESIGN};\
      set TDT_pass ${PASS};set TDT_consdir ${CONSPATH};\
      set TDT_dbdir ${DBPATH};set TDT_dbext ${DBEXT};\
      set TDT_repdir ${REPSPATH};set TDT_conext ${TCONEXT};"\
      -f ${TCSCR} | tee ${LOGSPATH}/${CHIP_DESIGN}_scan.${PASS}.${LOGEXT}

## Chip Level PrimeTime Run from Layout information.
pt_layout.${PASS} :: ${LAYOUTPATH}/${CHIP_DESIGN}.${PASS}.${DBEXT} ${PTSCR}
   ${PTSH} -x "set TDT_des  ${CHIP_DESIGN};\
      set TDT_pass ${PASS};set TDT_consdir ${CONSPATH};\
      set TDT_sdfdir ${LAYOUTPATH}; set TDT_sdfext SDF;\
      set TDT_dbdir ${LAYOUTPATH};set TDT_dbext ${DBEXT};\
      set TDT_repprefix .pt_layout.;\
      set TDT_repdir ${REPSPATH};set TDT_conext ${TCONEXT};"\
   -f ${PTSCR} | tee ${LOGSPATH}/${CHIP_DESIGN}_pt_layout.${PASS}.${LOGEXT} && touch
pt_layout.${PASS}

## Chip Level Floorplan manager from the layout information.
fpm_layout.${PASS} :: ${LAYOUTPATH}/${CHIP_DESIGN}.${PASS}.${DBEXT} ${FPMSCR}
   ${DCTSH} -x "set TDT_des  ${CHIP_DESIGN};\
      set TDT_pass ${PASS};set TDT_consdir ${CONSPATH};\
      set TDT_sdfdir ${LAYOUTPATH}; set TDT_sdfext SDF;\
      set TDT_dbdir ${LAYOUTPATH};set TDT_dbext ${DBEXT};\
      set TDT_phydir ${LAYOUTPATH};\
      set TDT_repprefix .fpm_layout.;\
      set TDT_repdir ${REPSPATH};set TDT_conext ${TCONEXT};"\
    -f ${FPMSCR} | tee ${LOGSPATH}/${CHIP_DESIGN}_fpm_layout.${PASS}.${LOGEXT} &&
touch fpm_layout.${PASS}

## Chip Level PrimeTime Run from Floorplan information.
pt_fp.${PASS} :: ${FLOORPLANPATH}/files.${PASS}/${CHIP_DESIGN}.${DBEXT} ${PTSCR}
   ${PTSH} -x "set TDT_des  ${CHIP_DESIGN};\
      set TDT_pass ${PASS};set TDT_consdir ${CONSPATH};\
      set TDT_sdfdir ${FLOORPLANPATH}/files.${PASS}; set TDT_sdfext SDF;\
      set TDT_dbdir ${FLOORPLANPATH}/files.${PASS};set TDT_dbext ${DBEXT};\
      set TDT_repprefix .pt_fp.;\
      set TDT_repdir ${REPSPATH};set TDT_conext ${TCONEXT};"\
    -f ${PTSCR} | tee ${LOGSPATH}/${CHIP_DESIGN}_pt_fp.${PASS}.${LOGEXT} && touch
pt_fp.${PASS}

## Chip Level Floorplan manager from the floorplan information.
fpm_fp.${PASS} :: ${FLOORPLANPATH}/files.${PASS}/${CHIP_DESIGN}.${DBEXT} ${FPMSCR}
${FLOORPLANPATH}/files.${PASS}/${CHIP_DESIGN}_setload.${TCONEXT}
   ${DCTSH} -x "set TDT_des  ${CHIP_DESIGN};\
      set TDT_pass ${PASS};set TDT_consdir ${CONSPATH};\
      set TDT_sdfdir ${FLOORPLANPATH}/files.${PASS}; set TDT_sdfext SDF;\
      set TDT_dbdir ${DBPATH};set TDT_dbext ${DBEXT};\
      set TDT_fpdir ${FLOORPLANPATH};\
      set TDT_phydir ${FLOORPLANPATH}/files.${PASS};\
      set TDT_repprefix .fpm_fp.;\
      set TDT_repdir ${REPSPATH};set TDT_conext ${TCONEXT};"\
    -f ${FPMSCR} | tee ${LOGSPATH}/${CHIP_DESIGN}_fpm_fp.${PASS}.${LOGEXT} && touch
fpm_fp.${PASS}

## *** This operation done to the layout directory
## Write out the Verilog for the top of the design.
## since comments are not allowed in these stream files here is what we are doing:
## # 1. read in the database
## # 2. Call script
${LAYOUTPATH}/files.${PASS}/${CHIP_DESIGN}.v :: ${DBPATH}.${PASS}/
${CHIP_DESIGN}.${DBEXT}
   ${DCTSH} -x "set TDT_des  ${CHIP_DESIGN};\
      set TDT_pass ${PASS};set TDT_consdir ${CONSPATH};\
      set TDT_dbdir ${DBPATH};set TDT_dbext ${DBEXT};\
      set TDT_repdir ${REPSPATH};set TDT_conext ${TCONEXT};\
      set TDT_layoutdir ${LAYOUTPATH}/files.${PASS};\
      set TDT_hierarchy ${CHIP_HIERARCHY};\
      read_db $<; " \
      -f ${LAYOUTSCR}  | tee ${LOGSPATH}/${CHIP_DESIGN}_tolayout.${PASS}.${LOGEXT};
```

Figure 9. The root Makefile

```
## *** This operation done to the floorplan director ***
## Write out the Verilog for the top of the design.
## since comments are not allowed in these stream files here is what we are doing:
## # 1. read in the database
## # 6. run a script
${FLOORPLANPATH}/files.${PASS}/${CHIP_DESIGN}.v :: ${DBPATH}.${PASS}/
${CHIP_DESIGN}.${DBEXT}
   ${DCTSH} -x "set TDT_des  ${CHIP_DESIGN};\
      set TDT_pass ${PASS};set TDT_consdir ${CONSPATH};\
      set TDT_dbdir ${DBPATH};set TDT_dbext ${DBEXT};\
      set TDT_repdir ${REPSPATH};set TDT_conext ${TCONEXT};\
      set TDT_fpdir ${FLOORPLANPATH}/files.${PASS};\
      set TDT_hierarchy ${CHIP_HIERARCHY};\
      read_db $<; "\
      -f ${FPSCR} | tee ${LOGSPATH}/${CHIP_DESIGN}_2fp.${PASS}.${LOGEXT}

#
## End of Chip Specific Stuff; beginning of the real make and budget engine on a bot-
tom up basis.
#

## Budget design
budgets.${PASS} :: ${DBPATH}.${PASS}/${TOP_DESIGN}_hier.${DBEXT} .synopsys_pt.setup
   ${DBSH} -f ${DBSCR} -x "\
      set TDT_dbdir ${DBPATH};\
      set TDT_pass ${PASS}; \
      set TDT_custom_wireload ${WIRELOADS};\
      set TDT_foundry_lib ${FOUNDRY_LIB};\
      set TDT_consdir ${CONSPATH}; \
      set TDT_consext ${CONEXT}; \
      set TDT_top_constraints ${CONSPATH}/${TOP_DESIGN}.${TCONEXT}; \
      set TDT_level 3; \
      set TDT_file $(<F);" | tee -a ${LOGSPATH}/allocate_budget.${PASS}.${LOGEXT} &&
touch budgets.${PASS}

## Now to make the actual designs

# they are included to give us some granularity in the design.

include Makefile.RTL
include Makefile.GATES


## end of Makefile
```

Figure 10. The Makefile.inc

```
# AFETC Generated Script - Edit with care
## Variables required by Makefile Makefile
##
## User should modify this script, it will not be over
##  written by another invocation of the AFETC script
##
##
## TOP Design Name (override this to make lower modules
TOP_DESIGN = generic_top

## the chip level name
CHIP_DESIGN = generic_chip

## Directories and file exts
##-------------------------
## RTL is the directory that rtl souce code will be found
RTLPATH = ../blm
VLOGEXT = v
```

Figure 10. The Makefile.inc

```
## GDB is the directory that dc_shell will expect to read GTECH db's from
GDBPATH = gdb
GDBEXT = db

## DB is the directory that dc_shell will write db's to
DBPATH = database
DBEXT = db

## CONSPATH is the directory that budget_shell will write constraints to
## and dc_shell will expect initial and top constraints to exist in
CONSPATH = constraint
CONEXT = dcc

TCONEXT = dct

## initial constraints file
INITCONS = generic_top.initial.dct

## LOGSPATH is the directory that log files will be written to
LOGSPATH = log
LOGEXT = log

## LAYOUTPATH is the directory that to layout files will be written to
LAYOUTPATH = layout

## LAYOUTPATH is the directory that to layout files will be written to
FLOORPLANPATH = fp

## REPSPATH = reports directory for any reports written from dc_shell
REPSPATH = reports

## Executables
##------------
## SYNOPSYS is only used in this Make include to help find
## paths to the executables
SYNOPSYS = /usr/synopsys/${SYNOPSYS_VERSION}/

## ARCH is only used in this Make include file
GETARCH = ${SYNOPSYS}/admin/install/syn/bin/getarch
## HARD CODED BECAUSE IT DOESNT WORK
## ARCH = $(GETARCH:sh)
ARCH = hpux10

## DCSH is the full path to the dc_shell executable
## DCSH = ${SYNOPSYS}/${ARCH}/syn/bin/dc_shell
DCSH = run_synop dc_shell

## DCSH is the full path to the dc_shell executable
## DCTSH = ${SYNOPSYS}/${ARCH}/syn/bin/dc_shell-t
DCTSH = run_synop dc_shell-t

## DBSH is the full path to the budget_shell executable
## DBSH = ${SYNOPSYS}/${ARCH}/syn/bin/budget_shell
DBSH = run_synop budget_shell

## PTSH is the full path to the pt_shell executable
## PTSH = ${SYNOPSYS}/${ARCH}/syn/bin/pt_shell
PTSH = run_synop pt_shell

## TXSCRIPT is the util to translate dc scripts into pt scripts
## TXSCRIPT = ${SYNOPSYS}/${ARCH}/syn/bin/transcript
TXSCRIPT = run_synop transcript

## DCTXSCRIPT is the util to translate dc scripts into dct scripts
## DCTXSCRIPT = ${SYNOPSYS}/${ARCH}/syn/bin/dc-transcript
DCTXSCRIPT = run_synop dc-transcript

## FOUNDRY_SCREEN is the util or script to Screen netlists
FOUNDRY_SCREEN = ./bin/vlsi_di22_screener.csh
```

Figure 10. The Makefile.inc

```
## the Baseline Foundary library you are using, for the wireload models.
FOUNDRY_LIB = vcs1083

## Scripts
##--------
## GTHSCR is the script that should be run to perform
## the compilation on the hierarchy version of the design
## to ensure all the pieces are found and linked together.
## the groupding and ungrouping takes place within this script.
GTHSCR = gth_global.dct

## GTSCR is the script that should be run to perform
## the compilation on the design using dc-t shell to build
## the GTECH design. This script reads in the verilog from
## the designlist and subdesign list, links and rights out
## the GTECH database
GTSCR = gt_global.dct

## GTECHIER is a tcl file that contains variables that only
##  the designer could know how to change.  Such things as
##  grouping and ungrouping and special cells for test.
GTECHIER = generic_top_hierarchy.tcl

## DCSCR is the script that should be run to perform
## the compilation on the design
DCSCR = dc_global.dct

## DCTSCR is the script that should be run to perform
## the compilation on the design using dc-t shell.
DCTSCR = dc_global.dct

## DCTSCR is the script that should be run to perform
## the compilation on the design using dc-t shell.
DCHSCR = dch_global.dct

## DBSCR is the script that should be run to perform
## the budgeting on the design
DBSCR = db_global.dct

## APSCR is the script that should be run to perform
## the Area and Power analysis on the design
APSCR = generic_area.dct

## TCSCR is the script that should be run to perform
## the Test Compiler Run for Scan insertion and stuff.
TCSCR = generic_tc.dct

## DCCSCR is the script that should be run to perform
## The linking of top and chip together
DCCSCR = generic_chip.dct

## LAYOUTSCR is the script that should be run to perform
## The linking of top and chip together
LAYOUTSCR = FIXUP.layout.dct

## FPMSCR is the script that should be run to perform
## Floorplan Manager links between floorplanning and layout.
FPMSCR = FIXUP.fpm.dct

## FPSCR is the script that should be run to perform
## The linking of top and chip together
FPSCR = generic_fp.dct

## XCELL is the script that should be run
## to force the exclusion of cells this is very
## library dependant
XCELL = excluded_cells.dct

## A file which contains a linked list of wireload files
```

Figure 10. The Makefile.inc

```
WIRELOADS = generic_wireload.dct

## Sundry variables
##-----------------
## PASS is 0 for pre-budget >0 for post-budget
PASS = 0

## TIMINGPATHS is 10 for sake of having a number.  This is the number
##  of paths that are checked for timing checker.  The larger the number
##  the more paths validated.
TIMINGPATHS = 10

## End of Make include file
```

Figure 11. The Makefile.RTL

```
##
## This file was auto-generated
## by AFTEC Generated Script - Edit with care
##
## All edits here may be overwritten
## Edit the include file to setup design and system parameters to keep
## I promise to honor them

## Now to make the actual designs
${GDBPATH}.0/generic_a.${GDBEXT} : ${RTLPATH}/generic_a.${VLOGEXT}
    ${DCTSH} -x "set TDT_des  generic_a;\
      set TDT_subs {};\
      set TDT_rtldir ${RTLPATH};\
      set TDT_vlogext ${VLOGEXT};\
      set TDT_gdbdir ${GDBPATH}.0;set TDT_gdbext ${GDBEXT};\
      set TDT_repdir ${REPSPATH};"\
      -f ${GTSCR} | tee ${LOGSPATH}/generic_a.gtech.${LOGEXT}

${GDBPATH}.0/generic_b.${GDBEXT} : ${RTLPATH}/generic_b.${VLOGEXT}\
   ${GDBPATH}.0/generic_c.${GDBEXT}
    ${DCTSH} -x "set TDT_des  generic_b;\
      set TDT_subs {};\
      set TDT_rtldir ${RTLPATH};\
      set TDT_vlogext ${VLOGEXT};\
      set TDT_gdbdir ${GDBPATH}.0;set TDT_gdbext ${GDBEXT};\
      set TDT_repdir ${REPSPATH};"\
      -f ${GTSCR} | tee ${LOGSPATH}/generic_b.gtech.${LOGEXT}

${GDBPATH}.0/generic_c.${GDBEXT} : ${RTLPATH}/generic_c.${VLOGEXT}
    ${DCTSH} -x "set TDT_des  generic_c;\
      set TDT_subs {};\
      set TDT_rtldir ${RTLPATH};\
      set TDT_vlogext ${VLOGEXT};\
      set TDT_gdbdir ${GDBPATH}.0;set TDT_gdbext ${GDBEXT};\
      set TDT_repdir ${REPSPATH};"\
      -f ${GTSCR} | tee ${LOGSPATH}/generic_c.gtech.${LOGEXT}

${GDBPATH}.0/generic_top.${GDBEXT} : ${RTLPATH}/generic_top.${VLOGEXT}\
   ${GDBPATH}.0/generic_a.${GDBEXT}\
   ${GDBPATH}.0/generic_b.${GDBEXT}
    ${DCTSH} -x "set TDT_des  generic_top;\
      set TDT_subs {};\
      set TDT_rtldir ${RTLPATH};\
      set TDT_vlogext ${VLOGEXT};\
      set TDT_gdbdir ${GDBPATH}.0;set TDT_gdbext ${GDBEXT};\
      set TDT_repdir ${REPSPATH};"\
      -f ${GTSCR} | tee ${LOGSPATH}/generic_top.gtech.${LOGEXT}


## end of file
```

Figure 12. The Makefile.GATES

```
##
## This file was auto-generated
## by AFETC Generated Script - Edit with care
##
## All edits here may be overwritten
## Edit the include file to setup design and system parameters to keep
## I promise to honor them

## now the real design
${CONSPATH}/generic_a.2.${TCONEXT} : ${CONSPATH}/generic_a.2.${CONEXT}

${CONSPATH}/generic_a.1.${TCONEXT} : ${CONSPATH}/generic_a.1.${CONEXT}

${CONSPATH}/generic_a.0.${TCONEXT} :
    rm -f $@; ln -s ../${INITCONS} $@;

${DBPATH}.${PASS}/generic_a.${DBEXT} : ${GDBPATH}/generic_a.${GDBEXT}\
   ${CONSPATH}/generic_a.${PASS}.${TCONEXT}
    ${DCTSH} -x "set TDT_des  generic_a;\
      set TDT_dont_use_cells ${XCELL};\
      set TDT_subs {};\
      set TDT_pass ${PASS};set TDT_consdir ${CONSPATH};\
      set TDT_fpdir ${FLOORPLANPATH};\
      set TDT_custom_wireload ${WIRELOADS};\
      set TDT_foundry_lib ${FOUNDRY_LIB};\
      set TDT_initcons ${INITCONS};\
      set TDT_dbdir ${DBPATH};set TDT_dbext ${DBEXT};\
      set TDT_gdbdir ${GDBPATH};set TDT_gdbext ${GDBEXT};\
      set TDT_repdir ${REPSPATH};set TDT_conext ${TCONEXT};"\
      -f ${DCTSCR} | tee ${LOGSPATH}/generic_a.dctsh.${PASS}.${LOGEXT}

${CONSPATH}/generic_b.2.${TCONEXT} : ${CONSPATH}/generic_b.2.${CONEXT}

${CONSPATH}/generic_b.1.${TCONEXT} : ${CONSPATH}/generic_b.1.${CONEXT}

${CONSPATH}/generic_b.0.${TCONEXT} :
    rm -f $@; ln -s ../${INITCONS} $@;

${DBPATH}.${PASS}/generic_b.${DBEXT} : ${GDBPATH}/generic_b.${GDBEXT}\
   ${CONSPATH}/generic_b.${PASS}.${TCONEXT}\
   ${DBPATH}.${PASS}/generic_c.${DBEXT}
    ${DCTSH} -x "set TDT_des  generic_b;\
      set TDT_dont_use_cells ${XCELL};\
      set TDT_subs {};\
      set TDT_pass ${PASS};set TDT_consdir ${CONSPATH};\
      set TDT_fpdir ${FLOORPLANPATH};\
      set TDT_custom_wireload ${WIRELOADS};\
      set TDT_foundry_lib ${FOUNDRY_LIB};\
      set TDT_initcons ${INITCONS};\
      set TDT_dbdir ${DBPATH};set TDT_dbext ${DBEXT};\
      set TDT_gdbdir ${GDBPATH};set TDT_gdbext ${GDBEXT};\
      set TDT_repdir ${REPSPATH};set TDT_conext ${TCONEXT};"\
      -f ${DCTSCR} | tee ${LOGSPATH}/generic_b.dctsh.${PASS}.${LOGEXT}

${CONSPATH}/generic_c.2.${TCONEXT} : ${CONSPATH}/generic_c.2.${CONEXT}

${CONSPATH}/generic_c.1.${TCONEXT} : ${CONSPATH}/generic_c.1.${CONEXT}

${CONSPATH}/generic_c.0.${TCONEXT} :
    rm -f $@; ln -s ../${INITCONS} $@;

${DBPATH}.${PASS}/generic_c.${DBEXT} : ${GDBPATH}/generic_c.${GDBEXT}\
   ${CONSPATH}/generic_c.${PASS}.${TCONEXT}
    ${DCTSH} -x "set TDT_des  generic_c;\
      set TDT_dont_use_cells ${XCELL};\
      set TDT_subs {};\
      set TDT_pass ${PASS};set TDT_consdir ${CONSPATH};\
      set TDT_fpdir ${FLOORPLANPATH};\
```

Figure 12. The Makefile.GATES

```
     set TDT_custom_wireload ${WIRELOADS};\
     set TDT_foundry_lib ${FOUNDRY_LIB};\
     set TDT_initcons ${INITCONS};\
     set TDT_dbdir ${DBPATH};set TDT_dbext ${DBEXT};\
     set TDT_gdbdir ${GDBPATH};set TDT_gdbext ${GDBEXT};\
     set TDT_repdir ${REPSPATH};set TDT_conext ${TCONEXT};"\
     -f ${DCTSCR} | tee ${LOGSPATH}/generic_c.dctsh.${PASS}.${LOGEXT}

${CONSPATH}/generic_top.${PASS}.${TCONEXT} : ${CONSPATH}/generic_top.${TCONEXT}
    rm -f $@; ln -s $(<F) $@;

${DBPATH}.${PASS}/generic_top.${DBEXT} : ${GDBPATH}/generic_top.${GDBEXT}\
   ${CONSPATH}/generic_top.${PASS}.${TCONEXT}\
   ${DBPATH}.${PASS}/generic_a.${DBEXT}\
   ${DBPATH}.${PASS}/generic_b.${DBEXT}
    ${DCTSH} -x "set TDT_des  generic_top;\
     set TDT_dont_use_cells ${XCELL};\
     set TDT_subs {};\
     set TDT_pass ${PASS};set TDT_consdir ${CONSPATH};\
     set TDT_fpdir ${FLOORPLANPATH};\
     set TDT_custom_wireload ${WIRELOADS};\
     set TDT_foundry_lib ${FOUNDRY_LIB};\
     set TDT_initcons ${INITCONS};\
     set TDT_dbdir ${DBPATH};set TDT_dbext ${DBEXT};\
     set TDT_gdbdir ${GDBPATH};set TDT_gdbext ${GDBEXT};\
     set TDT_repdir ${REPSPATH};set TDT_conext ${TCONEXT};"\
     -f ${DCTSCR} | tee ${LOGSPATH}/generic_top.dctsh.${PASS}.${LOGEXT}


## end of file
```

6.3 Global Scripts

This section contains many of the global scripts created by the “build_make” tool.

The gt_global is used by all the RTL to compiled. This is a very simple script.

Figure 13. The gt_global.dct Script

```
# AFTEC Generated Script - Edit with care
# generic compile script for
#
# Creating bottom level GTECH databases from Verilog.
#

if {  $TDT_vlogext == "" } {
   echo {Error: variable TDT_vlogext must be set to the file ext}
  exit 1
}
if {  $TDT_rtldir == "" } {
   echo {Error: variable TDT_rtldir must be set to the directory containing RTL}
  exit 1
}
if {  $TDT_gdbext == "" } {
   echo {Error: variable TDT_gdbext must be set to the GTECH DB file ext}
  exit 1
}
if {  $TDT_gdbdir == "" } {
   echo {Error: variable TDT_gdb must be set to the directory containing GTECH DBs}
  exit 1
}
if {  $TDT_repdir == "" } {
   echo {Warning: TDT_repdir not set, will write reports to cwd}
  set TDT_repdir {.}
}
if {  $TDT_subs == "" } {
   set TDT_subs [list]
}
if {  $TDT_des == "" } {
```

Figure 13. The gt_global.dct Script

```
   echo {Error: design not specified in TDT_des variable}
  exit 1
}

# Modify the search path so that the already optimized childen are found:
set _search_path [format "%s%s"  {./} $TDT_rtldir ]
set search_path [concat  $_search_path $search_path]


# Lots of errors will be had if it cannot find the GTECH databases in the search
engine but
#  this is what we want at this level.

echo {Info: reading all Verilog subdesigns}
foreach TDT_gdbsub $TDT_subs {
 read_verilog [format "%s%s"  [format "%s%s"  [format "%s%s"  [format "%s%s"
$TDT_rtldir {/}] $TDT_gdbsub] {.}] $TDT_vlogext]
 }

read_verilog [format "%s%s"  [format "%s%s"  [format "%s%s"  [format "%s%s"
$TDT_rtldir {/}] $TDT_des] {.}] $TDT_vlogext]

link

# now write out the design so we have access to it for future stages:
# this is necessary to control where it puts the designs for future reference
foreach_in_collection d [get_designs] {
  current_design $d
  write -format db -output $TDT_gdbdir/$current_design.$TDT_gdbext
}
#
exit
# end of file
```

The dc_global.dct is the script used to compile all lower level blocks. Notice the tests code for the pass variable and the extensive use of the “current_design” variable to make it portable. Of special interest is the trap just before compile to include your own “<module>.scr” to override the compile.

Figure 14. The dc_global.dct Script

```
# AFETC Generated Script - Edit with care
# generic compile script for
#  using and creating
#  design budgeting passes
#
# map efforts for compile runs
set TDT_pass0_map_effort {med}
set TDT_pass1_map_effort {med}
set TDT_pass2_map_effort {med}

if {  $TDT_gdbext == "" } {
   echo {Error: variable TDT_gdbext must be set to the GTECH DB file ext}
  exit 1
}
if {  $TDT_dbext == "" } {
   echo {Error: variable TDT_dbext must be set to the mapped DB file ext}
  exit 1
}

if {  $TDT_dbdir == "" } {
   echo {Error: variable TDT_db must be set to the directory containing mapped DBs}
  exit 1
}

set TDT_dbdir_src [format "%s%s" [format "%s%s"  $TDT_dbdir {.} ] [expr $TDT_pass -
1]]
```

Figure 14. The dc_global.dct Script

```
set TDT_dbdir [format "%s%s" [format "%s%s"  $TDT_dbdir {.} ] $TDT_pass ]

if {  $TDT_gdbdir == "" } {
   echo {Error: variable TDT_gdb must be set to the directory containing GTECH DBs}
  exit 1
}
if {  $TDT_repdir == "" } {
   echo {Warning: TDT_repdir not set, will write reports to cwd}
  set TDT_repdir {.}
}
if { $TDT_initcons == "" } {
   echo {Error: initial constraint variable TDT_intcons not set}
  exit 1
}
if {  $TDT_consdir == "" } {
   echo {Error: constraint directory variable TDT_consdir not set}
  exit 1
}
if {  $TDT_subs == "" } {
   set TDT_subs [list]
}
if {  $TDT_des == "" } {
   echo {Error: design not specified in TDT_des variable}
  exit 1
}

if { $TDT_pass >= 1} {
  # Modify the search path so that the already optimized childen are found:
  set _search_path [format "%s%s"  {./} $TDT_dbdir ]
  set search_path [concat  $_search_path $search_path]
  # Modify the search path so that if this is the bottom it will find its source.
  set _search_path [format "%s%s"  {./} $TDT_dbdir_src ]
  set search_path [concat  $_search_path $search_path]
} # else No search path for pass 0 as we want each db file to stand on
  # its own.  This was a real pain to figure out.

if {  $TDT_pass < 2 } {
   echo {Info: reading all GTECH subdesigns}
   foreach TDT_gdbsub $TDT_subs {
   read_file [format "%s%s"  [format "%s%s"  [format "%s%s"  [format "%s%s"
$TDT_gdbdir {/}] $TDT_gdbsub] {.}] $TDT_gdbext]
   set_dont_touch $TDT_gdbsub {false}
  }
  # relys on search path to find the already mapped DB files.
  read_file [format "%s%s"  [format "%s%s"  [format "%s%s"  [format "%s%s"
$TDT_gdbdir {/}] $TDT_des] {.}] $TDT_gdbext]
} else {
   # relys on search path to find subsequent designs
   echo {Info: reading mapped design}
   foreach TDT_dbsub $TDT_subs {
     read_file [format "%s%s"  [format "%s%s"  [format "%s%s"  [format "%s%s"
$TDT_dbdir_src {/}] $TDT_dbsub] {.}] $TDT_dbext]
     set_dont_touch $TDT_dbsub {false}
   }
   read_file [format "%s%s"  [format "%s%s"  [format "%s%s"  [format "%s%s"
$TDT_dbdir_src {/}] $TDT_des] {.}] $TDT_dbext]
}

link

# grab the constraints setup in the .dc_setup file.
set_operating_conditions -max $operating_conditions_max -min
$operating_conditions_min
set_wire_load -selection_group $wire_load_selection_group
set_wire_load -mode $wire_load_mode

if {  $TDT_pass == 0 } {
   echo {Warning: setting dont_use on all rpl implementations in standard.sldb}
   set_dont_use standard.sldb/*/rpl
```

Figure 14. The dc_global.dct Script

```
   # the removal of all rpls was a little extreme as this cause a very interesting
   #  side effect of removing several logic operators.  This puts back the compares
   #  which seems to fix the problem.
   remove_attribute standard.sldb/*cmp*/rpl dont_use
   echo {Info: including initial constraints}
   source $TDT_initcons

} else {
   echo [format "%s%s"  {Info: including design constraints for } $current_design]
   set _constraint_file_in [format "%s%s"  [format "%s%s"  [format "%s%s" [format
"%s%s" [format "%s%s" [format "%s%s" $TDT_consdir {/}] $current_design] {.}]
$TDT_pass] {.}] $TDT_conext]
   source $_constraint_file_in
}

# set _report_file_out [format "%s%s"  [format "%s%s"  [format "%s%s"  $TDT_repdir {/
}] $current_design] {.chk.rpt}]
# redirect $_report_file_out { check_timing }
echo {Uniquifying design to solve non-unique instantiations}
uniquify
# redirect -append $_report_file_out { check_design }

# grab the excluded cells script if necessary
if { $TDT_dont_use_cells == ""} {
    echo {Info: No Excluded Cells}
} else {
    echo [format "%s%s"  {Info: Reading Excluded Cells List: } $TDT_dont_use_cells]
    source $TDT_dont_use_cells
}

# custom wireload files if the side file exists
#
if {  $TDT_pass >= 1 } {

if { [file exists $TDT_custom_wireload] } {
    echo [format "%s%s" {Info: Reading Wireload Tcl File - } $TDT_custom_wireload]
    source $TDT_custom_wireload
    # now interate on the file until we have read in all the wireloads and attached
    # them to the target library
    foreach _my_fp $WireLoadFiles {
      set _wireload_file [format "%s%s" [format "%s%s" $TDT_fpdir {/}] $_my_fp]
      if { [file exists $_wireload_file] } {
        update_lib $TDT_foundry_lib $_wireload_file
        echo [format "%s%s" {Info: Reading in Wireload File } $_wireload_file]
      } else {
       echo [format "%s%s" [format "%s%s" {Warning: Wireload File } $_wireload_file]
{ Not Found.}]
      }
    }
    # now that we have the wireload models we need to attach the correct wireload
model to
    # our design so that we can perform further optimizations

    # first find a matching wireload table that is suppose to be used for the
current_module
    foreach _wl $WireLoads {
        if { [lsearch $designWireLoad($_wl) $current_design] >= 0 } {
            echo [format "%s%s" {Info: Found Wireload } $designWireLoad($_wl)]
            if { [set_wire_load $_wl -mode enclosed] } {
                # remove the auto select if we have our own as it defaults to true
                set auto_wire_load_selection false
                echo [format "%s%s" {Info: Applied DesignWireLoad } $_wl]
                }
            break
         } else {
           echo [format "%s%s" {Info: No DesignWireLoad Found for } $current_design]
         }

    }
```

Figure 14. The dc_global.dct Script

```
} else {
    echo [format "%s%s" [format "%s%s" [format "%s%s" {Info: No Custom Wireload File:
} $TDT_custom_wireload ] { or Pass not greater than 1: } ] $TDT_pass]
}
}



# each design is allowed a custom compile script that has the same format as below
#  but allows the user to control how this module gets compiled.  This is setup to
allow
#  the user flexiblity in the design.
set _compile_script [format "%s%s" $TDT_des {.scr}]
if { [file exists $_compile_script] } {
    echo [format "%s%s" {Info: Reading Compile Script - } $_compile_script]
    source $_compile_script
} else {
  echo [format "%s%s"  {Info: compiling design } $current_design]
  if {  $TDT_pass == 0 } {
     compile -scan -map $TDT_pass0_map_effort
  } else {
     if {  $TDT_pass == 1 } {
       compile -scan -map $TDT_pass1_map_effort
     } else {
       # if we are on the last passes then ungroup the design ware components
       # as synopsys should have made good choices by now and we could get a little
       # more optimization by removing this layer of hierarchy. ## Hard pathed TCL
script.
       source ungroup_dw.tcl
       ungroup_dw -hier
       compile -scan -incr -map $TDT_pass2_map_effort
     }
  }
}

set_dont_touch $current_design {true}
if {  $TDT_subs != [list] } {
   set _file_out [format "%s%s"  [format "%s%s"  [format "%s%s"  [format "%s%s"
$TDT_dbdir {/}] $current_design] {.}] $TDT_dbext]
   write -h -o $_file_out
} else {
   set _file_out [format "%s%s"  [format "%s%s"  [format "%s%s"  [format "%s%s"
$TDT_dbdir {/}] $current_design] {.}] $TDT_dbext]
   write -o $_file_out
}

# this alters comparison with hier compile
set _report_file_out [format "%s%s"  [format "%s%s"  [format "%s%s"  [format "%s%s"
[format "%s%s"  $TDT_repdir {/}] $current_design] {.pass}] $TDT_pass] {.qor.rpt}]
# redirect $_report_file_out { report_qor }
set _report_file_out [format "%s%s"  [format "%s%s"  [format "%s%s"  [format "%s%s"
[format "%s%s"  $TDT_repdir {/}] $current_design] {.pass}] $TDT_pass] {.tim.rpt}]
# redirect $_report_file_out { report_timing -max_path 10 -nworst 10 -path short }

#
exit
# end of file
```

The dch_global.dct was used to build the hierarchy design. Many of the notable snips in this code is the generation of a “my_<foundary_lib>.db” to support the use of custom wireloads.

Figure 15. The dch_global.dct Script

```
# AFETC Generated Script - Edit with care
# Generations of a Hiearchy with Timing Constraints Design Database for
#  use with downstream tools.
```

Figure 15. The dch_global.dct Script

```
#
#
# change the directory path so that the data for this pass is found.
set TDT_dbdir [format "%s%s" [format "%s%s"  $TDT_dbdir {.} ] $TDT_pass ]

# Modify the search path so everyone is found.
set _search_path [format "%s%s"  {./} $TDT_dbdir ]
set search_path [concat  $_search_path $search_path]


# read and link design using search path to build it.
read_file [format "%s%s"  [format "%s%s"  [format "%s%s"  [format "%s%s"  $TDT_dbdir
{/}] $TDT_des] {.}] $TDT_dbext]
link;

# grab the constraints setup in the .dc_setup file.
set_operating_conditions -max $operating_conditions_max -min
$operating_conditions_min
set_wire_load -selection_group $wire_load_selection_group
set_wire_load -mode $wire_load_mode


# read in the top level constraints so they are applied.
set _constraint_file_in [format "%s%s" [format "%s%s" $TDT_des {.}] $TDT_conext]
source $_constraint_file_in

# generate a qor report
set _report_file_out [format "%s%s"  [format "%s%s"  [format "%s%s"  [format "%s%s"
[format "%s%s"  $TDT_repdir {/}] $TDT_des] {.pass}] $TDT_pass] {.qor.rpt}]
redirect $_report_file_out { report_qor }

# modify the attributes before we start working with the design.
foreach_in_collection TDT_cd [get_designs] {
  current_design $TDT_cd
  if {[get_attribute $current_design dont_budget] == "true" } {
     set_dont_touch {true}
     } else {
       remove_attribute $current_design dont_touch
     }
  }

# reset the design to the top

current_design $TDT_des
link
# check_design report
set _report_file_out [format "%s%s"  [format "%s%s"  [format "%s%s"  $TDT_repdir {/}]
$current_design] {_hier.chk.rpt}]
redirect $_report_file_out { check_design }

# Loop Check Report
set _report_file_out [format "%s%s"  [format "%s%s"  [format "%s%s"  $TDT_repdir {/}]
$current_design] {_hier.loop.rpt}]
redirect $_report_file_out { report_timing -loops }

# Latch Checking
set _report_file_out [format "%s%s"  [format "%s%s"  [format "%s%s"  $TDT_repdir {/}]
$current_design] {_hier.latch.rpt}]
redirect $_report_file_out { all_registers -level_sensitive }

# note hard coded name.
set _file_out [format "%s%s"  [format "%s%s"  [format "%s%s"  [format "%s%s" [format
"%s%s"  $TDT_dbdir {/}] $current_design] {_hier}] {.}] $TDT_dbext]
write -h -o $_file_out

# custom wireload files if the side file exists
#
if {  $TDT_pass >= 1 } {
```

Figure 15. The dch_global.dct Script

```
if { [file exists $TDT_custom_wireload] } {
    echo [format "%s%s" {Info: Reading Wireload Tcl File - } $TDT_custom_wireload]
    source $TDT_custom_wireload
    # now interate on the file until we have read in all the wireloads and attached
    # them to the target library
    foreach _my_fp $WireLoadFiles {
      set _wireload_file [format "%s%s" [format "%s%s" $TDT_fpdir {/}] $_my_fp]
      if { [file exists $_wireload_file] } {
        update_lib $TDT_foundry_lib $_wireload_file
        echo [format "%s%s" {Info: Reading in Wireload File } $_wireload_file]
      } else {
       echo [format "%s%s" [format "%s%s" {Warning: Wireload File } $_wireload_file]
{ Not Found.}]
      }
  }
  set _file_out [format "%s%s"  [format "%s%s"  [format "%s%s"  [format "%s%s"
$TDT_dbdir {/my_}] $TDT_foundry_lib] {.}] $TDT_dbext]
  write_lib $TDT_foundry_lib -format db -output $_file_out
} else {
   echo [format "%s%s" [format "%s%s" [format "%s%s" {Info: No Custom Wireload File:
} $TDT_custom_wireload ] { or Pass not greater than 1: } ] $TDT_pass]
}
}

# foreach_in_collection TDT_cd [get_designs] {
#  current_design $TDT_cd
#  set _file_out [format "%s%s"  [format "%s%s"  [format "%s%s"  [format "%s%s"
$TDT_dbdir {/}] $current_design] {.}] $TDT_dbext]
#  write -format db -output $_file_out
# }

exit
# end of file
```

The db_global.dct was used by the Design Budgeter, with budget_shell, to build the module level constraint. Within this script is the increment of the pass variable, and the generation of the constraint files.

Figure 16. The db_global.dct Script

```
# AFETC Generated Script - Edit with care
# Budget Compiler simple script
#
## assume that .synopsys_pt.setup is read already

# set the dbdir up for the proper pass.
set TDT_dbdir [format "%s%s" [format "%s%s"  $TDT_dbdir {.} ] $TDT_pass ]

# need to add dbdir to the path so everything can be found.
set _search_path [format "%s%s"  {./} $TDT_dbdir ]
set search_path [concat  $_search_path $search_path]

# only if we are at pass 1 or better do we bother.
# if we have a custom wireload file then use the my_vendor library file in the db
directory
if { $TDT_pass >= 1} {
    if { [file exists $TDT_custom_wireload] } {
        echo "Info: Referencing Library with Custom Wireload Models"
        set _my_library [format "%s%s"  {my_} $TDT_foundry_lib ]
        set link_library [concat $_my_library $link_library]
        set link_path $link_library
    }
}

read_db $TDT_file
link
```

Figure 16. The db_global.dct Script

```
# assume that the user has build the constraints with both min and max, which
#  is not supported in Budget shell.  We must reset the design first for all timing
#  aspects, then read in the constraint file.  The constraint file must have in it a
#  check of the variable: synopsys_program_name which is equal to "budget_shell"
around
#  all the minimum timings in order for this to work.
reset_design -timing
source $TDT_top_constraints

incr TDT_pass
allocate_budget \
    -no_environment \
    -format dcsh \
    -level $TDT_level \
    -create_context \
    -write_context \
    -file_format_spec "$TDT_consdir/%D.$TDT_pass.$TDT_consext";

exit;
# end of file
```

6.4 The Generic Scripts

Although the tool outputs many generic scripts we will focus on the ones necessary for the design budgeting process.

The wireload tcl script is an associated TCL list of wireload files and wireload table entries. This file is used by the “dc_global” and “dch_global scripts” to read in and apply the wireloads to the design. The associated list allows the user flexibility to choose at what level to apply the wireloads.

Figure 17. The generic_wireload.tcl Script

```
# AFETC Generated Script - Edit with care
#  Generic example of a wireload script for design.
#
# Assume Avant! Planet Tools.
#

# list of wireload files found in the Floorplanning directory
set WireLoadFiles { \
       my_top.wl \
       my_sub.wl \
    }

# List of wireloads in the wireload files.
set WireLoads { \
        GROUP_top_wl \
        my_top_wl \
        GROUP_sub_wl \
        my_sub_wl \
    }

# Associated list of wireloads and the designs they are used on
#   Should be at least one entry per WireLoads from above.
# Every wireload from above must be accounted for here even if
#  there is no design associated with them.
set designWireLoad(GROUP_top_wl) { \
        my_top \
    }

set designWireLoad(my_top_wl) {}
set designWireLoad(GROUP_sub_wl) {}
set designWireLoad(my_sub_wl) {}
# end of file.
```

The initial conditions are provided as a starting point for the types of conditions to setup on at the chip or top level. The template gives the user the types and styles of conditions they should use.

Figure 18. The generic_top.initial.dct Script

```
# AFETC Generated Script - Edit with care
# initial constraints for building design
#   for budgeting
set _period {10}
if ![string compare $_period NEED_TO_SET_PERIOD] {
   echo {Error: Need to specify tightest period in initial cons file}
  exit 1
}
set _half_period [expr $_period / 2]
set _waveform [list 0 $_half_period]
set _multiplier 2
set _input_delay [expr 0.65 * $_period]
set _output_delay [expr 0.35 * $_period]

set _inputs [all_inputs]
set _outputs [all_outputs]
set _register_c [all_registers -no_hier -clock_pin]

if {  $_register_c == [list] } {
   set _clock_pins [list]
} else {
   set _clock_pins [all_fanin -to $_register_c -start]
  create_clock $_clock_pins -name {reg2reg} -period $_period -waveform $_waveform
  set_multicycle_path $_multiplier -from {reg2reg} -to {reg2reg}
}

create_clock -name {simple} -period $_period -waveform $_waveform
set_input_delay $_input_delay -clock {simple} [remove_from_collection $_inputs
$_clock_pins]
if { $_outputs != {} } {
   set_output_delay $_output_delay -clock {simple} $_outputs
}
# end of file.
```

The generic_chip.dct script is used to generate the combined top and chip level. Often, depending upon the type of pad-ring compiler. JTAG is considered also at this time. Also at this time it is good to remove the DesignWare components and other structural manipulations.

Figure 19. The generic_chip.dct Script

```
# AFETC Generated Script - Edit with care
# Generation of the chip level combined db file for use with down
#  stream tools.  This file must have a much of the constraint
#  information as possible to use with the downstream tools.
#
#
# change the directory path so that the data for this pass is found.
set TDT_dbdir [format "%s%s" [format "%s%s"  $TDT_dbdir {.} ] $TDT_pass ]


# The top level design has already been read in at this point we now need to
#  process the top level files.


# Modify the search path so the source is found for the tap controller
set _old_search_path $search_path
set _search_path [format "%s%s"  {./} $TDT_rtl ]
set search_path [concat  $_search_path $search_path]

# set _file_in [format "%s%s" $TDT_des {_tap_state.v}]
# read_verilog $_file_in
# set _file_in [format "%s%s" $TDT_des {_tap.v}]
```

Figure 19. The generic_chip.dct Script

```
# read_verilog $_file_in

# current_design ${TDT_des}_tap
# ungroup -all -flatten
# compile -map_effort med
# dont_touch $current_design
# set _file_out [format "%s%s"  [format "%s%s"  [format "%s%s"  [format "%s%s"
$TDT_dbdir {/}] $current_design] {.}] $TDT_dbext]
# write -o $_file_out


# now get the chip level design with pads.
set _file_in [format "%s%s" $TDT_des {.v}]
read_verilog $_file_in

current_design $TDT_des;

# since we read in the hierarchy file life should be find and no search path
#  changes are necessary.
set search_path $_old_search_path]

# reset the search path so all the db files are found if necessary
set _search_path [format "%s%s"  {./} $TDT_dbdir ]
set search_path [concat  $_search_path $_old_search_path]

link;

# read in the top level constraints so they are applied.
# set _constraint_file_in [format "%s%s" [format "%s%s" $TDT_des {.}] $TDT_conext]
# source $_constraint_file_in

# generate a qor report
# set _report_file_out [format "%s%s"  [format "%s%s"  [format "%s%s"  [format "%s%s"
[format "%s%s"  $TDT_repdir {/}] $TDT_des] {.pass}] $TDT_pass] {.qor.rpt}]
# redirect $_report_file_out { report_qor }

# reset the design to the top
current_design $TDT_des

# check_design report
set _report_file_out [format "%s%s"  [format "%s%s"  [format "%s%s"  $TDT_repdir {/}]
$current_design] {_hier.chk.rpt}]
redirect $_report_file_out { check_design }

# Loop Check Report
set _report_file_out [format "%s%s"  [format "%s%s"  [format "%s%s"  $TDT_repdir {/}]
$current_design] {_hier.loop.rpt}]
redirect $_report_file_out { report_timing -loops }

# Latch Checking
set _report_file_out [format "%s%s"  [format "%s%s"  [format "%s%s"  $TDT_repdir {/}]
$current_design] {_hier.latch.rpt}]
redirect $_report_file_out { all_registers -level_sensitive }

# note hard coded name.
set _file_out [format "%s%s"  [format "%s%s"  [format "%s%s"  [format "%s%s"
$TDT_dbdir {/}] $current_design] {.}] $TDT_dbext]
write -h -o $_file_out

exit
# end of file
```

The floorplan script is shown as an example of moving data from db format to a verilog format that third party tools use. Notice the dump_timing.tcl script which dumps the Avant! specific timing details out to a file for use with the Avant! P&R tools.

Figure 20. The generic_fp.dct Script

```
# AFETC Generated Script - Edit with care
#  Script for dumping data required by layout tools
#

# always generate an error
set _basescript {WROTE_IT}
if ![string compare $_basescript NEED_TO_WRITE] {
   echo {Error: Need to Build Design Specific FP Script}
  exit 1
}

# the the design to the top modules
set module $TDT_des

# read and link design using search path to build it, ASSUME it was read in on the
command line
# link;

# grab a tcl script to report the timing of each level.
# NOTE HARDCODED
source tcl/dump_timing.tcl

# grab file with stuff setup in it.
source $TDT_hierarchy

# modify the attributes before we start working with the design.
foreach TDT_cd $LayoutBlocksHierarchy {
  current_design $TDT_cd
  DumpTiming $current_design $TDT_pass $TDT_fpdir
  }

foreach TDT_cd $LayoutBlocksFlat {
  current_design $TDT_cd
  DumpTiming $current_design $TDT_pass $TDT_fpdir
  }

# reset the design to the top and dump the DB so we have a match.
current_design $TDT_des

set _file_out [format "%s%s"  [format "%s%s"  [format "%s%s"  $TDT_fpdir {/}]
$current_design] {.v}]
write -hierarchy -format verilog -output $_file_out

exit
# end of file
```

7.0 Trademark Information

Verilog is a registered trademark of Cadence Design Systems, Inc.

SDF and SPEF are a trademark of Open Verilog International.

Synopsys, PrimeTime, DesignWare are registered trademarks of Synopsys, Inc.

Design Compiler, Test Compiler, Floorplan Manager, characterized, dont_touch, dont_touch_network and uniquify, are trademarks of Synopsys, Inc.

Avant!, Plant!, Apollo! are trademarks of Avant! Corporation.

All other brand or product names mentioned in this document, are trademarks or registered trademarks of their respective companies or organizations