

The Five-Day Verification Plan

Peet James, Principal Engineer

Qualis Design Corporation

www.qualis.com

peet@qualis.com

ABSTRACT.

To succeed, today's ASICs must have a comprehensive verification plan. A detailed document that spells out how to attack the overall task of verification is necessary. This paper details how to jump-start the creation and writing of a verification plan that is owned by the entire ASIC team.

1.0 Purpose & History

1.1 Purpose

The purpose of this paper is to outline the creation process and the content of a concise, usable verification plan. A two-track approach is presented:

- The main text describes the generation of the verification plan within five days
- The figures show an example of parts of the verification plan itself

Tips for minimizing impact on the engineering team, obtaining their full “buy in” and implementing part of the plan during the five days are also included.

1.2 History of CVP Evolution

The five-day approach and the resulting verification plan have evolved over about 15 projects. As an engineering consultant who comes in as a “hired gun” to jump-start a verification project, I am exposed to many diverse groups of engineers and to many diverse projects.

The approach spelled out here has been proven in the trenches, and has evolved with each use. Almost without exception, the engineering talent was a little skeptical at first, but by the time they completed and followed the plan, they would not do another ASIC or FPGA without a verification plan. This plan can be adapted for any level of logic: ASIC, FPGA, card, etc., and generally any size (I have used it on several multi-million gate designs). The doc should only be about 10-30 pages in length. Any longer and it becomes too much work to maintain.

For the purpose of the paper and the included example plan, I assume that the verification team is fairly new to the whole verification world. In some cases, mature verification engineers will be available who can jump-start this whole process. In that case, some of the preliminaries will be unnecessary.

2.0 Greasing the Skids

To prepare for creating the verification plan, follow this approach:

- **Read white papers:** Hand out a few white papers on verification a couple of weeks before arrival. (For example, use this paper and my *A Recipe for Multi-Million Gate ASIC Verification* paper presented at SNUG Boston 1999.) Are there any existing verification documents? Hand them out to help the engineering team's brains start thinking about verification.
- **Cheat:** Don't emphasize that we are coming together to make a verification plan document. This news may scare some engineers away. A much better approach is to emphasize that you are gathering the full team together to obtain everyone's input on the verification of the part. This is what we are really doing anyway. The verification plan is just a clear and concise way to document everyone's ideas. These meetings are the first step toward having universal buy-in from the engineering team.
- **Prepare for meetings:** Reserve a conference room for each day of the week starting at the same time and lasting for about 2 hours (10am to 12pm is good). Monday - Friday works great. The first meeting is typically a little longer (3+ hours).
- **Identify leadership:** Identify one verification facilitator who will run the meetings. He or she is responsible for keeping the meetings on track and for writing up the plan each day so it is ready for review during the next meeting. As a consultant, this typically has been my job. However, I still identify a person so that I can prepare them to own the plan after I move on.

3.0 Day One: The Overall Approach

Implementation of the CVP

Day One is all about coming up with an overall approach to verifying the part. It can start with a full “dream team” anything-goes approach at the start, then funneling down to a more practical approach. The goal is to obtain an overview of how the verification infrastructure will be setup and how it will work.

Sometimes several approaches (Vera, E or C with file-driven testcase as well as straight-HDL testbenches) become prevalent. This is OK. A verification infrastructure that can seamlessly handle multiple approaches needs to be developed. Now is the time to have all this out in the open.

Typically, a block-level diagram is drawn up during the course of discussion that represents the general verification approach and its necessary elements. The initial ideas of file structure, scripting, environmental setup and file format are also generated.

Figure 1 on page 4 illustrates a sample block diagram of an ASIC named CoolChip. Figure 2 on page 5 shows a sample of a harness block diagram.

Figure 1.
Sample block diagram of the
CoolChip ASIC

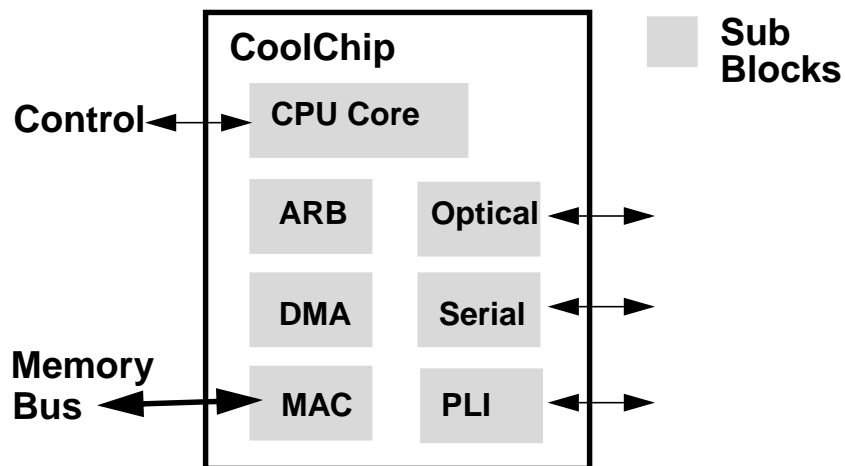
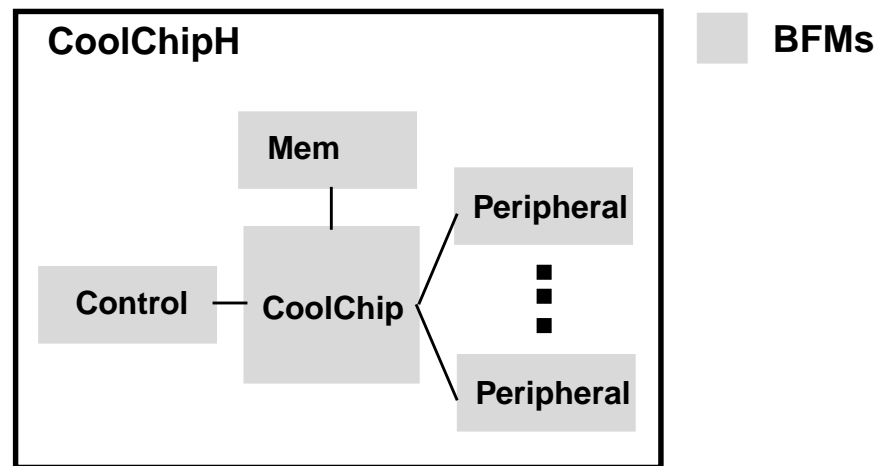


Figure 2.
Sample harness block
diagram of the CoolChip
ASIC



3.1 To Do List

Follow this list of tasks on Day One:

- Discuss the general throughput of the part and the major interfaces. How will it be used in the real world? What are the key things it does? What are the main modes?
- Discuss all the interfaces. What bus functional models (BFMs) are needed for each interface? Buy or build the BFMs? File input and output with these BFMs or auto data generation? C or HDL or tool (Chronology)? Let each person talk about their part of the logic, and give input.
- Is there any verification infrastructure available already? Verification is a place where real reuse can happen. Are there existing scripts, C code, BFMs, utilities, etc.? Steal and reuse anything that is useful. Leverage any existing verification infrastructure.
- Discuss behavioral modeling of the part. How to compare golden with actual? C or HDL or tool (SPW)?
- Determine high-level verification language use (Vera, Specman, System C, RAVE)?
- What sort of harness will be used? Client-Server approach?
- Determine whether any random-based verification architecture/approach is necessary? Are you going to use explicit testcases with simple random background noise? Or are you going to write a random testcase and then constrain/direct it to hit the testcases your want?

- Review general verification practices. A quick pep talk on good verification practices is helpful here. Focus on basic client-server harness and testbench structure information. (“*Writing Testbenches Functional Verification of HDL Models*” by Janick Bergeron and “*VHDL FAQ*” by Ben Cohen are good texts. Both are published by Kluwer Academic Publisher.)
- Discuss scripting. What simulation run scripts are needed?
- Review directory structure. Where will stuff go to facilitate flexibility and ease of use? Revision control? Bug reporting?

Using a regression script, which runs one or more simulation scripts, is a great timesaver. Figure 3 on page 7 explains the usage of a sample regression script.

Figure 3. Sample regression file usage

Simulation Environment

Each simulation is categorized by functions and gives a letter identifier and a number (such as bs_rx_tb1). The numbered simulation directory contains the verify.scr driver file and all the other files necessary to drive the simulation.

First, verify.scr can perform any pre-simulation file generation. Next, it can invoke VHDL and run the simulation. The script can also perform post-processing. This pre- and post-processing might be simple file generation, or it might be complex calls to C or Perl routines to generate or manipulate the necessary files. In this way, a simple VHDL driver (bs_rx_tb1.vhd) can stimulate and check the design, or a series of more complex tests driven by C or Perl can drive the simulation using a universal launch.vhd file. Simulations should be run from the verification directory.

Syntax: `regress [-id <sim-run-dir>] [-f <tbs-file>] [-d <global-dir>]`

Examples:

Single sim, default log run dir	<code>regress bs_reg_tb1</code>
Multiple sims, default log run dir	<code>regress bs_reg_tb1 int_tx_tb1</code>
Single sim, use simrun1 for run dir	<code>regress -id simrun1 bs_rx_tb2</code>
Run all sims listed in passing file	<code>regress -f passing</code>
Run all sims listed in all file, dump logs in gold dir	<code>regress -d gold -f all</code>
Run all sims in mysims file, use sim5 for run dir, dump logs in sim061000	<code>regress -id sim5 -f mysims -g sim061000</code>

Options:

<code>regress</code>	Can be run with a command-line list of testbenches. For example, <code>regress bs_rx_tb1</code> or <code>regress bs_rx_tb1 int_tx_tb1</code> .
<code>-id <sim-run-dir></code>	Declares the simulation run directory that is created under the numbered testbench directory and holds the simulation results. If you don't specify a directory, the script creates a directory called "log" under your simulation directory (<code>top_tb/bs_rx_tb1/log</code>) and runs the script from there.
<code>-f <tbs_file></code>	Points to a file listing the testbenches to run. Alternatively, you can specify the testbenches to run individually by separately them with a space (<code>regress bs_rx_tb1 int_tx_tb1</code>).
<code>-d <global-dir></code>	Specifies the global directory created under the simulation directory for the log files created after each testbench is run.

Figure 4 on page 8 illustrates a sample directory structure for the Cool-Chip verification infrastructure.

Figure 4. Sample file structure for the CoolChip verification infrastructure	\$WORKPATH	Top of revision control hierarchy
	/rtl	RTL code directory
	/verif	Verification infrastructure main directory
	/doc	cvp and other documentation
	/bfm	Bus functional models
	/harness	Harness and shared code
	Coolchiph.vhd	Harness
	syslog_pb.vhd	Universal output logging code
	init_pb.vhd	Universal initialization code
	/bs_rx.tbl	Individual testbench directory
	verify.scr	Testbench script
	bs_rx_tbl.vhd	VDHL-specific testbench code
	/int_tx_tbl	Test files
	verify.scr	Testbench script
	launch.vhd	Universal C-based, file I/O testbench code
	int_tx_tbl.c	C code
<u>NAMING CONVENTIONS</u>		
*.vhd	for VHDL files	
_tb#.	for testbench- related files & directories	
*.scr	for simulation- launch scripts	
*.c	for C code files	

3.2 Gotchas

- Discussion splits into tangents about RTL guts.
- One guy goes dictator and declares, “This is how it shall be written, thus shall it be done.”
- Lack of quorum needed for useful brainstorming and decision making. People are no shows, come in late, leave early, come and go.
- Renegade maverick engineer who won't play nice with others. Won't use scripts, file formats, or directory structures.
- Getting stuck on one aspect (dir structure, script, BFM, etc.) of the verification infrastructure at the expense of the others. The goal here is to get a start on each of all the necessary verification ingredients. The team can go round two and three on particular areas later in the week. Just get the main idea down in a succinct format. Any area that bottlenecks should be back burned until Day 5. Agree on a basic direction and then settle the details on Day 5.

3.3 Assignment

Verification team leader writes up the first sections of the Verification plan and has copies ready for the next meeting.

At the end of the Day One session some specific tasks should be clear enough to assign the following:

- Harness person.
- BFM person(s), at least start with one of the simpler BFMs.
- Directory structure person.
- Scripting person. This person can start writing certain parts and then can report back each day as the week progresses.

These people can start developing their part of the verification infrastructure during the rest of the week. Their goal can be putting together a usable example (harness, fake chip, BFM and a sample testbench) by a certain date. It is a good idea to prioritize the various parts of the verification infrastructure, so that things come together in a timely manner.

4.0 Day Two: Basic Sanity

Feature Description of the CVP

Start Day Two with a review of the verification plan document. Mark up any changes. Do not get side tracked. If issues come up, take them offline or discuss at the end of meeting.

Day Two begins a three-day effort of extracting from the team a list of the features that need to be verified. I recommend dividing this task up into several categories with a certain focus for each category. For this paper we will use three categories: basic sanity, intentional and stress testing. The content and approach of these three categories is explained on the following pages, but any breakdown of useful categories (2 to 5 total categories) will do. Here is a suggested list of other breakdowns:

- Mustard, relish, ketchup
- Grunt, real, what-if, random
- Larry, Moe, Curly
- Simple, directed, illegal

4.1 Yellow-Sticky Method

I use this oriental method (has some fancy name that I forgot) that I modified and renamed as the Yellow-Sticky method. The goal is to extract the maximum information in the shortest amount of time. Each engineer needs one sticky pad, preferably sized 3x3 and in a unique color. If all the engineers have the same color, have each person write their initials in the corner of each sticky.

Here is the procedure:

1. Spend about 15-45 minutes having everyone just write their features to test. At the start emphasize the category that you are writing features for.
2. For the Basic Sanity category emphasize these ideas:
 - These are short tests that run in a couple of minutes.
 - These tests will run when total breakdown occurs and you need to return to a known state. These tests may require running before anyone checks in changes to the database.
 - Focus on simple and basic, verifying grass-root things: register loading, resets, adders add, etc.. To keep people on track, you may have to reiterate these concepts.
3. Write one idea per yellow sticky.
4. Write the idea, not the whole test. For example, write, “Ensure both hard and soft reset bring the chip back to a known state.” The features will be combined later into actual testbenches. When the testbench is eventually assigned to an engineer, he or she can make an outline detailing how the test will be done. There is danger of putting in too much detail here and making the CVP difficult to write and maintain. Limit the paper to 10-30 pages. Do not bog down in the details. Enough information to direct the test is all that is needed.
5. Have the general specification for the part and any other document nearby. Engineers can grab them and use them to prompt ideas for various features to test. Sometimes, you can break up the task into smaller areas of interest as directed by some of the existing documentation.
6. Bulleted lists are great here: For example:
 - Ensure proper branch instruction set
 - + branch-nops
 - + back-to-back branch
7. Once everyone is done, have each person stick their notes up on a wall, white board or paper easel sheets.

8. Start putting like-stuff together. Sub-categories will naturally emerge. Mark duplicates. Combine similar tests into lists.
9. Identify any list that does not fit the category and move it. For example, if the test is too complex, move it to the intentional or stress categories.
10. Label the sub-categories that emerge. Sample sub-categories are: initial, regs, CPU interface, data interface, modes, etc..
11. It is a good idea to identify the person who is the principle owner of the logic verified under each sub-category. Put their initials next to the sub-category. They will be responsible to review the features listed in their sub-category area and keep them up to date. They will also be the contact person for questions and the one who will sign-off on the outline that the testcase writer will write up later.

The figure below shows some sample feature basic sanity tests for the CoolChip ASIC. Some of the details (like spec refs & testcase grouping and naming) would not be added till later. The far right column is a marker for where the test will be run (TH is for the standard test harness, might have BLK for being done at the block level, etc.). The focus here is to list and group the features.

Figure 5.
Sample Basic Sanity tests for the CoolChip ASIC

CoolChip Register and Reset		
1. Ensure that registers with access modes of RO, RW, WP and Mixed are compliant with CoolChip GS (Section 4.1 - Table 1). Bits that are of the toggle type (ROL) should be tested as if they are RO type registers.	BS_REG_TB1	TH
2. Ensure that upon Hard Reset all registers return to their reset values as specified in the CoolChip GS (Section 4.1 - Table 1).	BS_REG_TB1	TH
3. Ensure that upon Soft Reset that those registers outlined in CoolChip GS (Section 4.2) reset to their reset values as specified in the CoolChip GS (Section 4.1 - Table 1), and that the remaining registers retain their current values.	BS_REG_TB1	TH
4. Ensure all unused register addresses return 0's.	BS_REG_TB1	TH
5. Ensure multiple Chip Select lines give proper interrupt (CoolChip GS: Section 5.2) and that the register sub-assembly is not corrupted.	BS_REG_TB2	TH

4.2 Gotchas

- People write everything on one sticky.
- People give too much detail.
- People's handwriting is unreadable.
- People get stuck and need prodding to think outside the box (not just their logic, but other's logic and the chip as a whole).
- One guy writes all the lists, the rest of the team has blank stares. Tip: Have the team jump-start themselves by pulling out the spec and flipping through it. This task helps engage their brains.

4.3 Assignment

Verification lead takes yellow stickies and sub-categories and enters them into the CVP doc for the next day's review. Can delegate this to a technical writer type person, if one is available.

Start thinking of sub-categories, features, etc. for the other categories (intentional and stress).

Continue working on your verification infrastructure assignments from the previous day.

5.0 Day Three: Getting Intentional

Feature Description of the CVP

Start Day Three with a review of Day Two verification plan additions.

5.1 Yellow-Sticky Method

Repeat yesterday's yellow-sticky exercise, but this time focus on the intentional features. This is typically the largest of the three sections. At the start of this intentional feature-listing session, emphasize these ideas:

- Intentional tests run longer, even very long.
- Test the normal features, regular operations, normal “day-in-the-life-of-the-part” stuff.
- Think of normal paths that data takes.
- Do not use a convoluted or devious tests. That is stress testing.
- These tests can be more exhaustive, basic-sanity tests.
- The test might be iterative. Run with this, then run with that, etc..

5.2 Gotchas

Same as Day Two, but typically this goes smoother because the bumps were smoothed out in the basic sanity section.

The list becomes huge. Re-emphasize what intentional means. Use bullets or lists or anything else that will compartmentalize the feature list into a usable form. Leave details until later. This tends to be the largest of the three sections.

5.3 Assignment

Verification lead or a technical writer takes yellow stickies and sub-categories and enters them into the CVP doc for the next day's review.

Have each person who is the owner of each basic sanity sub-category review the feature lists and clean them up. They can also start grouping the features into actual testbenches and input the information into the testbench list.

Continue working on your verification infrastructure assignments from the previous day.

6.0 Day Four: Stressed Out

Feature Description of the CVP

Start Day Four with a review of Day Three verification plan additions.

6.1 Yellow-Sticky Method

Repeat yesterday's yellow-sticky exercise, but this time focus on the stress features. Emphasize these ideas:

- Stress tests run longer, even very long.
- Create “lets-break-it” tests. Create “what-if-this-happens-in-the-middle-of-that” tests.
- Test advanced or questionable features.
- Use convoluted or devious tests.
- Use randomization.
- These tests can be more exhaustive, basic-sanity or intentional tests, or redo's with a twist.
- These tests might be self-directing to run differently each time they are run.

6.2 Gotchas

Same as Day Two and Three, but again this typically goes smoother because the bumps were smoothed out in the earlier sections. At this point, the team can generally do this task themselves.

Some engineers have trouble getting rude on their designs here. They are so used to not even being able to verify normal functions, that they need prodding to think deviously about actually breaking their designs.

6.3 Assignment

Verification lead or technical writer takes yellow stickies and sub-categories and enters them into the CVP doc for the next day's review.

Have each person who is owner of each intentional sub-category review the feature list and clean them up. They can also start grouping the features into actual testbenches and input the info into the testbench list.

Continue working on your verification infrastructure assignments from the previous day.

7.0 Day Five: Divide and Conquer

Testbenches of the CVP

Day Five is a catch all. Here is a list of things that can be done:

- Rehash any verification infrastructure area(s) that were deferred on Day 1. Typically, if a consensus (such as dir structure) was not reached on Monday, it is back-burnered until today (Day 5). Often, due to how far the CVP has come, the issue(s) will have solved themselves or at least become more clear. Hammer them out now.
- List/prioritize the verification infrastructure work that remains to be done. Schedule and assign this work.
- Assign/reassign ownership of the key elements of the verification infrastructure (harness, BFM, scripts, etc.).
- Finalize at least the basic sanity section of the testbench list and assign the first few tests.
- Assign an owner of the verification plan document.
- Set a goal. For example:
 - Having first BFM with a little test working in the verification infrastructure (with the scripts) by a certain date
 - Having all first passes of the BFMs done and hooked up in the harness by a certain date
 - Having the BM of the part done by a certain date

Sample testbenches for the CoolChip ASIC are presented below.

Figure 6.
Sample testbenches for the
CoolChip ASIC

Testbenches						
Bench Name	Effort	Owner	Priority	Cross Ref	Harness	Status
BS_RST_TB1	3	Ed	Phase 1	A.5.2	H1/V	Done
BS_REG_TB1	4	Sam	Phase 2	A.1.1-4	H1/V	Next 2/27/98
BS_BUS_TB1	9	Ralph	Phase 1	A.2.5, B.1.6	H1/C	Debug
BS_INT_TB1	5	Qualis- Peet	Phase 3	A.4.3	H2/CV	Future 3/3/98
INT_TX_TB2	8	Sam	Done at block	B.6.6	BLK/V	Done
Legend:			C = C-driven testbench			
H1 = CoolChip1 <-- harness #1			V = VHDL-driven testbench			
H2 = CoolChip2 <-- harness #2			CV = Combo of C- and VHDL-driven test- bench			

7.1 Gotchas

- No Ownership: Not enough people to take ownership of the verification infrastructure parts. RTLr's plates are too full doing the RTL input. Shameless Plug: Hire Qualis Engineering or other consultants to do some of the work.
- Manager type tries to assign all testbenches, priorities and effort at one time. Best to just do the first few, see how it goes and then extrapolate the rest of the information later.

7.2 Assignment

Verification lead finalizes Revision 1 of the CVP, checks it in to some sort of revision control, and distributes it to all involved parties. Owner of CVP keeps document up to date.

Have the person who is the owner of each stress group review the feature lists and clean them up. They can also start grouping the features into actual testbenches and input the information into the testbench list.

Set up weekly verification meetings to mark status and progress.

8.0 Conclusion

Writing a verification plan can be quite a breakthrough accomplishment for a design team. With initial expectations of “this will be a waste of time” to a complete turn around is a cool thing to experience. Engineers become excited about doing verification. They are “converted” so that they do not attempt another project without a comprehensive verification plan and the resulting verification environment. Sometimes veteran RTLrs who are kind of bored with RTL code, get real excited about expanding into the world of verification code. At any rate, it is a very sound investment for any design team to take the time up front to create and own a verification plan.

The keys to a good verification plan?

- A format that is concise and useful. Keep it on a diet.
- A quick gathering and deployment of the document itself.
- An approach where each of the design team members gives their input.
- Ownership by all parties involved.