SPEED UP VERILOG SIMULATION BY 10-100X WITHOUT SPENDING A PENNY

Rajesh Bawankule, Hardware Engineer, Cisco Systems, Inc., San Jose, CA

Abstract

Increasing size and complexity of ASICs/FPGAs are causing dramatic shift in verification methodologies. Full chip functional simulation at RTL level still consumes large chunk of engineering months and is a major bottleneck in meeting project schedules. Following paper shares few insights, tips and tricks author discovered in designing chips in past.

Let go, Observe, Act

A successful execution benefits from an upfront investment in proactive planning and not reacting. I was involved recently in the design and verification of two large chips. The first one was around 18-20 million gates ASIC and second was 7 million gates. I followed few simple steps to boost the regression performance. As the title suggests following steps do not require investment in any additional hardware or software resources. Of course it needs a precious resource, your time and interest.

Let go

The first step in speeding up your simulation is to stop boasting about your simulation run times. One should not take pride in telling that his/her simulation takes days and weeks. It simply means that you do not know better ways to run your simulations yet.

It is tough to let go of your attachments to test run times. There is no point in running the same test cases over and over again and show others that you are doing something productive. You should let go of your ego and fear and convince yourselves that job can always be done in better way.

Observe

The second step is to observe what is going on as it is and not as what people say or what you have thought in the past. It is important to remove the mental block that the current way of doing things is the only way.

We will discuss few built in tool features like profiling which will assist you in observing simulation bottlenecks.

Act

Once you have observed the state of simulation it is easy to notice the bottlenecks and remove them to increase simulation speed.

The few lessons I have learnt in past are summarized as cases studies. The title suggests the main concept used in that particular case. The simulation speed increase in these cases is mutually exclusive. You can combine multiple actions to get bigger effective increase.

Profiling

Profiling is one of the simplest and the most effective tools for observing simulation bottlenecks. Almost all the simulators have it in some form or another. I am going to show you with an example of Synopsys VCS but it can be done with any other simulator.

The following small change is required to create a simulation profile for VCS Verilog simulator.

vcs +prof -Mupdate -PP +vcsd -f filelist

Running with this script will create a file called vcs.prof. This file will be different for different tests as tests hammer different portions of the design. A following vcs.prof will reveal some interesting aspects of simulation with a simple RegisterTest.

```
=========================
      TOP LEVEL VIEW
=========================

   TYPE      %Totaltime
----------------------------------------
PLI            0.10
VCD           61.40
KERNEL         1.88
DESIGN        36.62
----------------------------------------
```

This means that 60% time is spent in dumping the VCD file. Someone forgot to turn off the dumping switch! If we are running regressions then we need not keep this dumping on. Just turning off the dumping gave us 2x speed up.

In another part of report,

```
==================================================================
           MODULE VIEW
==================================================================

Module(index)   %Totaltime   No of Instances   Definition
----------------------------------------------------------------------------------------------------
LIB_FF_T (1)        3.94          375        /libpath/LIB_FF_T.v:45.
LIB_FF (2)          3.87          520        /libpath/LIB_FF.v :42
SubtractBlock(3)    2.84          128        ../rtl/SubtractBlock.v:14
SPARES(4)           2.70          375        ../global/SPARES.v:7
MVE16384066084(5)   2.67          16         ../macrocells/MVE16384066084.v:53
IO_CELL(10)         1.17          87         ../rtl/ IO_CELL.v:22
```

13-14% time is spent in 2 flops and the spare gates block. This means around 35% logic simulation time is spent on these gates. ((100/40) * 14). Well, at least for this test. Other tests will hammer some other portion of logic.

Spare Gates as well as IO macro cells contained instances of flops and gates that do not get optimized after synthesis. I created a blank module for the spare gates and wrote a simpler model for the flipflop to reduce run times. Combined with not dumping I could get around 4x speedup.

Optimized compilations

Most of us have switched to compiled simulators from interpreted simulators for performance benefits. There are a few tricks to make even these compiled simulations faster. Synopsys VCS has built-in Radiant technology which can dramatically boost the simulation runs while running long regressions.   These Radiant optimizations improve the simulation performance of all types of designs from behavioral and RTL to gate-level designs.

You specify Radiant technology optimizations at compile-time. Radiant technology has the following compile-time options:

+rad or +rad+2   Specifies Radiant level 2

+rad+1   Specifies Radiant level 1

+optconfigfile   Specifies applying Radiant technology optimizations to part of the design using a configuration file.

Existing compilation Makefile can be modified by creating a separate rule to create rad optimized simulation run file. Optimizations work well when RTL code is written keeping pure synchronous design style in mind. Many times one will be forced to use external IP or existing legacy code which may create simulation mismatches with rad optimizations. In such cases exclude these files from getting optimized by using the +optconfigfile option.

Radiant optimizations do not work with coverage tools or SDF back annotated simulations. Avoid dumping VCD files while using rad optimizations as it causes a performance hit.

In my experience, rad optimizations deliver anywhere between 4x to 10x performance boost.

2 State Simulation

Two state simulation is a high performance option where signals can only have the simulation values 1 and 0. X, Z and strength values are not simulated. Two state simulation takes advantage of the fact that after initialization many digital designs predominantly simulate with only 1 and 0 values.

+2state can be used to run your entire design or part of it in 2state mode instead of the conventional 4state mode. This can give between 20% and 200% speedup.

You need to compile your design with +2state switch. You specify which part of your design works only in 2 state and rest in 4 state by using the configuration file with the +optconfigfile+filename compile-time option.

Initial State Simulation

Many complex tests require initializing configuration registers and memories to a known state. Many times using regular tasks to perform writes and reads to verify operations can consume $10 - 50\%$ of total test time. Initializing registers and memories in bulk can save this time.

One may not be satisfied with this approach as it bypasses the write-read operations, which are central to initialization operation. A special dedicated test can be written to just check this initialization sequence. A subset of test, which goes through detailed initialization sequence, can also be used to get enough confidence on initialization sequence with getting benefit of increased simulation speed.

Efficient Modeling

You will notice that one of the biggest time hogging element is the memory model. There are various ways to speed up memory model.

Using design and simulations wrappers

Full chip simulation normally uses instantiations of IO cells along with core logic. The data buses with the large width are broken and connected to IO cells. In the simulation side these different bits are spliced together and reconstructed to form data buses back.
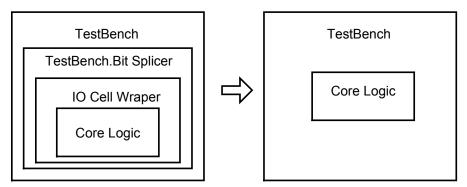

Figure 1: Removal of IO cell and simulation wrappers

Bit blasted buses translate to more number of simulation events and thus slow down the simulation. Carefully removing IO cell wrapper in design and Bit splicer in simulation testbench will give you tremendous boost in simulation performance. I have seen 25-40% saving in run times by doing this. You still need to run few key tests with IO wrappers to have confidence in your setup.

Turning off the switches

A typical memory model supplied by vendor contains lots of constructs to check setup, hold timing and protocol violations. One may not need execution of these checks once memory controller is stable. Often vendor provides a switch to disable these checks in run time. Use these switches to effectively speed up the memory simulation.

Creating simpler models

A simpler memory model can be substituted for a vendor model to meet the basic functionality. Using this model one will get simulation boost sacrificing the complex checks.

If you observe keenly you will notice that vendor has some fixed notation for register array and embedded memory blocks. File names give a fair idea of what may be contained in the memory model. A simple perl script can be written to read these file names and create another directory with simpler models.

Using right switches

Many times one keeps on using legacy makefiles and does not bother to check various simulation switches and their effects. Make sure that you use proper switches while running long regressions.

+nospecify : Suppresses module path delays and timing checks in specify blocks. This option can significantly improve simulation performance. This can be most effective at functional gate level simulations.

+notimingchecks : Tells VCS to ignore timing check system tasks when it compiles your design.

Note that +nospecify is a superset of +notimingchecks.

Avoid any debug switches like +cli, +acc, -line etc. They can significantly slow down simulations.

Using adaptive PLIs

A typical test bench uses one or more of following PLIs.

- Stimulus generators / checkers.
- Interface with vendor memory models
- Test pattern generator for test equipments.
- Dump file converters for 3rd party simulation brwosers.

For a typical test or a test suite, not all PLIs are required. Run a test with the additional VCS switch +vcs+learn+pli This monitors the PLI access that was actually used during the simulation, and creates pli_learn.tab

In next compilation you tell VCS to apply what it has learned in the previous run. Specify the secondary PLI table file, with the +applylearn+filename compile-time option (you can omit +filename from the +applylearn compile-time option and VCS uses the pli_learn.tab secondary PLI table file). Simulate again with a simv executable in which only the ACC capabilities you need are enabled.

Compiling with Vera

Hardware Verification Languages like Vera and e (Verisity) are used along with Verilog for more effective verification. It is better to use latest versions of VCS as well as Vera to get fastest run times. Newer releases also add new switches to make simulations faster.

Use following example to increase your run times. I have seen around 15% increase in performance after using newer switch.

Older Way:
```
vcs router.test_top.v router.v router.vshell \
-P $VERA_HOME/lib/vera_pli.tab \
$VERA_HOME/lib/libSysTask.a
```

Newer better Way:
```
vcs –vera router.test_top.v router.v router.vshell \
```

Measuring runtime

```
time vcs -Mupdate –f filelist   this will measure compile from scratch
time ./simv                     this will measure run-time for particular simulation
```

Similar method may be used when running other UNIX based simulators.

Using Linux Machines

Most often, one will be using mostly either a Sun Solaris machines or the Linux boxes for running simulation jobs or regressions. It is suggested that one should decide on future purchases based on proper research and benchmarking.

Linux vs Sun
I am listing my personal experiences, which may help you in decision-making.

Simulation Speedup:
Under controlled environment Linux gives 2-4x speedup over Sun machine. The processor and memory combination is not comparable. It is impractical to compare based on processor speed criteria. Comparison is based on Linux and Sun machines bought in similar timeframe.

Reliability:
I have seen Sun machines working non-stop for years. Linux machines are relatively unreliable as they are assembled with components designed for average consumer in mind. It is advisable to purchase a reliable brand known for their robustness. I found that most failures were related to power supplies and hard drives.

Degradation:

Linux boxes show dramatic degradation when multiple simulation jobs are executed. On Sun machines two same simulations will increase time taken by each simulation by approximately 2x. On Linux machine same simulation will take 4-8x more time.

It is advisable that care should be taken not to fire multiple simulations on Linux box.

Intelligent use of resources

Selecting computing resource is only one aspect of efficient computing. Faster disk arrays, networking connections, switches play a big role in deciding simulation performance. I have seen one group of Sun servers performing 3x better than other group of similar Sun servers because of different configuration.

Creating a scheme where simulation dumps and log files are saved in local memory/hard disk of computing machine saves network accesses giving significant boost in simulation performance.