Maximizing HDL Simulation Performance

How do you know what is happening during simulation? Here's one way, using the new ModelSim SE Performance Analyzer from MTI.

by Darron May, Technical Marketing Engineer, Model Technology Inc, darronm@model.com

Simulation can be time consuming, and often there is no visibility into what is actually happening. That's why Profiling or Performance Analysis can very valuable. Performance Analysis shows where the simulator is spending time, and can be extremely useful at all levels of the design abstraction from behavioral to gate level. The new ModelSim SE simulator (Special Edition) has a built-in Performance Analyzer, available in the 5.3 version of VSIM, with the potential to save many hours of regression test time.

Performance Analyzer

The Performance Analyzer provides a graphical representation of where ModelSim is spending run time. The graphical representation is useful in quickly and easily determining what is impacting your design environment's simulation performance—it is rare that a simulation environment is fully tuned for performance.

At any moment the simulator will be executing a specific action, which can be categorized into a number of groups. This action could be attributed to a line of code or an internal housekeeping event. Profile samples are accumulated for each group and line of code, and then presented as the amount of execution time with respect to the total run time.

The benefit of using this type of statistical analysis is that you typically don't have to run an entire simulation to get enough information to analyze the environment.

Improving RTL Performance

The following example shows how the Performance Analyzer was used to improve simulation run time at the RTL level; it takes a customer design and shows how the simulation run time was improved. This VHDL design was from a data communications application, used as a buffer between two different data stream rates; the concepts hold true for an equivalent model in Verilog. The Performance analyzer was used to take around 3000 samples. After simulation it could be seen that 98% of the simulation time was spent executing two lines of code as shown in Figure 1, each using a similar style but in separate parts of the design. These lines of code were VHDL loop statements. The run time for this example was 20 minutes 34 seconds (Code Fragment 1).

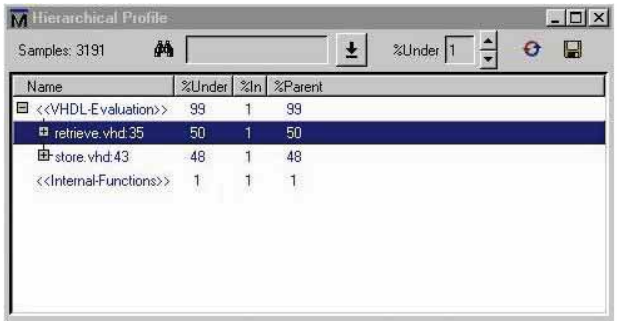

Figure 1 - Profiler Report Window

Code Fragment 1 - ORIGINAL

```
for i in 0 to (buffer_size - 1) loop
IF (i = ramadrs((counter_size - 1) downto 0)
THEN
rd0a <= buffers(i);
END IF;
end loop ;
```

A loop was used to make an assignment to each of the buffer locations in the design if the input counter address matched the currently indexed location. It was possible to add an "exit" command into both of these loops which would break the loop when the match occurred. After the code modifications, the simulation was re-run and the run time was reduced to 10 minutes and 3 seconds, a 2X run time improvement (Code Fragment 2).

```
Code Fragment 2 - EXIT
for i in 0 to (buffer_size - 1) loop
 IF (i = ramadrs((counter_size - 1) downto 0)
 THEN
 rd0a <= buffers(i);
 exit;
 END IF;
end loop ;
```

However, the loop was still consuming the majority of the simulation time as detailed by the profiler output. As the loop is being used to compare an address pointer to a location it was possible to re-write this model to use an indexed array to achieve the same function. When the simulation was re-run with the new array version of the model the run time reduced to 1 minute 2 seconds, which was 20X faster than the original model. Code fragments for each change are shown.

The profile then showed a better distribution, however there was now another part of the design consuming the majority of the time. This was the control block and was centered around a large counter that was modeled using a std_logic_vector. In VHDL integers take less memory than std_logic_vectors and can be manipulated as part of the language. Therefore if this model could be re-written to use integer math, then time could be saved in conversion, addition, and comparison functions (Code Fragment 3).

```
Code Fragment 3 - ARRAY
address := conv_integer(ramadrs((counter_size-1) downto 0));
rd0a <= buffers(address);
```

The model was again re-written to reduce the number of events by using less processes, integer math, and variables for connecting functions instead of signals. Signals cause events when they change whereas variables do not. The simulation was re-run with the new model and the simulation time was further reduced to 27 seconds.

Due to the profiler showing where time was spent in the simulation, the use of better modeling techniques allowed an overall speed improvement of over 40X from the original model.

Gate Level Performance

The Performance Analyzer can also be used at the gate level. ModelSim automatically recognizes that a VITAL function is being referenced from the IEEE library and generates code to call hand-optimized built-in routines. All of the functions within the VITAL_primitives and VITAL_timing packages are accelerated in this way for both ModelSim PE and ModelSim EE. There is an extra level of optimization that occurs in the EE product. To qualify for this global acceleration, the VITAL cell has to be VITAL Level 1 compliant. The model is then transformed into a state machine that does the equivalent behaviour more efficiently. This can have a significant effect on the simulation performance. The Performance Analyzer can be used to see the effects of an non-accelerated model as the execution of accelerated code is shown in a separate category.

Conclusion

The Performance Analyzer has already been used by a number of customers at all stages of the design cycle to save many hours of regression test time, or to allow more verification in the same amount of time. With the growing time-to-market pressures, allowing more verification in the same time period will increase product quality.

For more information see Model Technology at: www.model.com.