What Time Is It: Implementing a SystemVerilog Object-Oriented Wrapper for Interacting with the C Library `time`

Eldon Nelson M.S. P.E., Synopsys, Inc.

SYNOPSYS®

Conclusion

A common question within the SystemVerilog community is how to get the wall-time during the simulation. There is no built-in method within SystemVerilog to get wall-time. This paper documents the development and motivations of a SystemVerilog object-oriented wrapper of the C library `time`. The design of this wrapper is based upon the object-oriented solutions from the Python `time` library and the Ruby `Time` library. The solution is released under the GNU GPL license and is available on GitHub.

The `svtime` package provides a non-object-oriented implementation of the wrapper, which is very similar to the Python `time` implementation. This Python style time library uses purely static methods and a struct matching that of the C `time` library. Also provided in the `svtime` package is a Ruby style object-oriented implementation. This uses, in contrast, an object to handle the conversions and functions in an object-oriented style. The author prefers the Ruby style implementation and recommends that version for ease-of-use at a small, but reasonable, overhead of 3% over the Python style. The performance benefit of using a SystemVerilog DPI wrapper is 801 times faster than a common solution using `$system`, as documented on Stack Overflow.

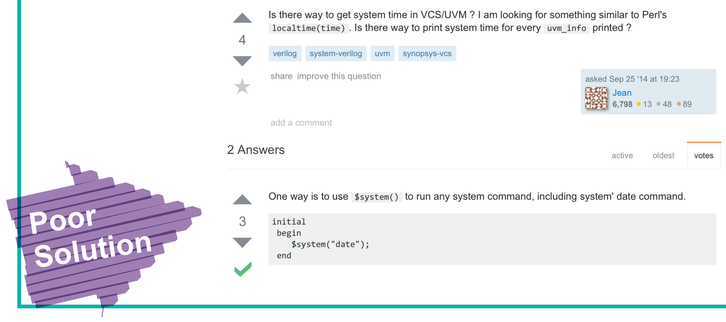
Introduction

Simulation time is the time within the SystemVerilog simulation which can be obtained by running the built-in `$time` function. Wall-time is the time you see on your watch; the actual time in space that we are all experiencing together. SystemVerilog users are attempting to mark when a test started in the simulation log for post processing, determine how long in wall-time a sequence takes to run, or conduct simulator performance experiments. The wall-time query with most other programming languages is common and easily answered. Unfortunately, SystemVerilog does not have access to wall-time, out-of-the-box. This lack of the wall-time query has resulted in less than optimal understanding and solutions to get to the wall-time from the SystemVerilog simulation.

This paper discusses design choices and motivations, and provides the source code for a user-friendly SystemVerilog object-oriented wrapper to interact with the C library `time`. The proposed approach is several hundred times faster than conventional `$system` functions. In addition, an approach to determining key bottlenecks in simulation runs is proposed by plotting the simulation time versus wall-time.

Quote and Motivation: Stack Overflow

"Is there a built in DPI function to get time so that I don't have to write one?"
User Jean on Stack Overflow, Oct 14, 2014



Results

https://github.com/tenthousandfailures/svtime

The product of this paper is a user-friendly SystemVerilog wrapper around the C `time` library. It is published on GitHub under a GNU GPL license. The desire is to make this a common library for SystemVerilog. A well documented `make` file, full examples, and a thoughtful port of the Python/Ruby time libraries are provided for SystemVerilog. Pull requests accepted.

Example `svtime` API Usage

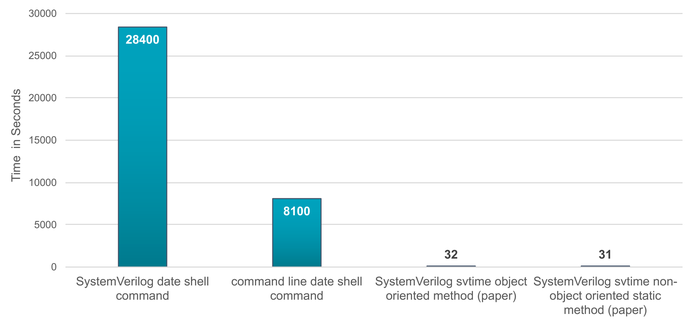
```
svtime_pkg::svtimep svtimep_inst;

...

svtimep_inst = new();
svtimep_inst.now();
$display("svtimep_inst.to_s() = %s", svtimep_inst.to_s());
$display("svtimep_inst.sec() = %0d", svtimep_inst.sec());
$display("svtimep_inst.min() = %0d", svtimep_inst.min());

svtime::sleep(2);
svtimep_inst.now();
$display("svtimep_inst.to_s() = %s", svtimep_inst.to_s());
```

Example `svtime` API Usage Output

```
svtimep_inst.to_s() = 2017-11-16 01:37:07
svtimep_inst.sec() =             7
svtimep_inst.min() =            37
svtimep_inst.to_s() = 2017-11-16 01:37:09
```

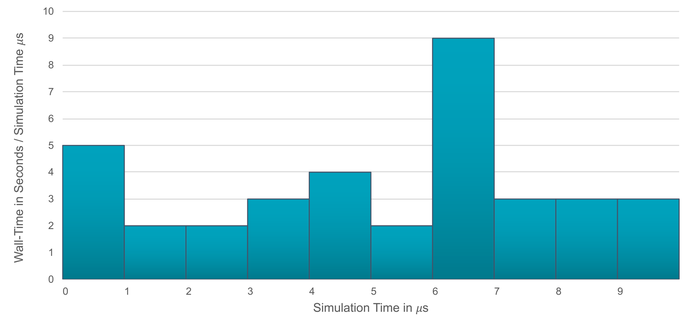
Performance of `svtime` Versus Other Methods

A major benefit of using the SystemVerilog DPI and the provided wrapper is speed compared to other methods. The speed improvement of using this method versus using a `$system` call is over 800 times faster! It is even faster than calling `date` on the Linux command line. It is apparent why this result happens. It is because of the performance difference between system DRAM and current non-volatile memory systems. The graph below compares the performance of different methods of getting wall-time in SystemVerilog over 10 million calls (lower is better).



Application 1: Simulation Performance Monitoring

Plot simulation time versus wall-time per simulation time. This allows for seeing when in a simulation that simulator performance degrades. Knowing this could ultimately help in deploying efforts to speed up a simulation once the simulation time that caused it is known.



Application 2: Wall-Time Aware Nightly Regressions

Make the nightly regressions simulations aware of the wall-time. The regressions can then quiesce its stimulus on a schedule to ensure that nightly regressions can end gracefully before the morning reporting.

```
interface time_alarmclock(input reg clk);
   parameter period = 2;
   parameter alarm_hour = 5;
   parameter alarm_min  = 0;
   parameter prefix = "ta";

   longint   cycles = 0;
   bit       triggered = 0;
   svtime_pkg::svtimep svtimep_inst;

...

   // trigger if wall-time equals alarm_hour and at or above alarm_min
   always @(posedge clk) begin;
      svtimep_inst.now();
      if ((cycles >= period) &&
          (triggered == 0) &&
          (alarm_hour == svtimep_inst.hour()) &&
          (alarm_min <= svtimep_inst.min())
         ) begin
            $display("[%s] time_alarmclock TRIGGERED at simtime \
                      %0t with period %0d at %2d:%2d:%2d",
                     prefix, $time(), period,
                     svtimep_inst.hour(), svtimep_inst.min(), svtimep_inst.sec()
                    );
            triggered = 1;
            cycles = 0;
          end else begin
            cycles++;
          end
   end
endinterface
```

Contact

Eldon Nelson M.S. P.E.
Synopsys Verification Application Engineer
eldon.nelson@synopsys.com
eldon_nelson@ieee.org