2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Emulation Testbench Optimizations for better Hardware Software Co-Validation

Vijayakrishnan Rousseau
Validation Lead, Intel Corporation
1900 Prairie City Road, Folsom, CA - 95630
vijayakrishnan.rousseau@intel.com

Suresh Balasubramanian
Emulation Lead, Intel Corporation
1900 Prairie City Road, Folsom, CA - 95630
suresh.balasubramanian@intel.com

Srikanth Reddy Rolla
Emulation Engineer, Intel Corporation
1900 Prairie City Road, Folsom, CA - 95630
srikanth.reddy.rolla@intel.com

Mohamed Saheel Nandikotkur Hussainsaheb
Emulation Engineer, Intel Corporation
1900 Prairie City Road, Folsom, CA – 95630
mohamed.saheel.nandikotkur.hussainsaheb@intel.com

***Abstract*— With tighter time to market schedules it is not enough if the silicon is bug free, it is also required for the software stack to be ready in order to launch the product. The need to have both software and hardware ready at the time of first silicon platform has become the norm. This necessitates the use of virtual platforms (software model that mimic the final hardware platform) for software (device drivers, firmware etc.) development during pre-silicon phase. Though the virtual platform suffices most of the needs for software testing, there are still certain gaps in the testing/validation. Both the hardware and software interact with each other by adhering to certain hardware software interaction specification as defined by a platform or system architect. In order to validate this aspect, it is mandatory that certain level of software validation is done on the actual hardware before silicon which is achieved by the use of an emulator or FPGA (Field-programmable gate array) prototyping system. This paper discusses about the inefficiencies in an emulation testbench which slow down the hardware software validation and ideas to improve them.**

Keywords—Emulation, FPGA, Simulation Acceleration, Hardware-Software Co-Validation.

I. EMULATOR BASED SYSTEM

Which is the right platform for the hardware software co-validation? Is it FPGA or Emulator? FPGA prototyping systems are faster and less expensive, but the debug capabilities are limited. On the other hand, emulators are more expensive and run at lower clock frequencies but have better debug capabilities some of which are comparable to that of simulators. If the Hardware IP (Intellectual Property) that is being validated is well validated in simulation already or the IP is from a previous product with few incremental changes then FPGA prototyping system could be the preferred system of choice for hardware software co-validation to save cost. Whereas if the IP being validated is a completely new generation IP or the hardware validation is not completely done yet, IP validation and Hardware-Software co-validation happen in parallel making an emulator based system the preferred choice to have better debug turnaround time. The limitations in the Testbench discussed in this paper may be applicable to FPGA

prototyping systems too, since most of the work done here is based on the emulator based systems only. The below diagram depicts a typical setup used for hardware software co-validation. The hardware IP is ported to an emulator and the software code (device driver or firmware) is executed in a virtual platform that models everything in the platform except the hardware IP under test. The hardware IP and the associated software communicate via a C-Testbench. The C-Testbench decodes all software messages or APIs (Application Programming Interface) and translate into commands which the emulator understands using the SCE-MI APIs[1] or DPI (Direct Programming Interface) [2] calls.

Figure. 1: Emulation Based Hardware-Software Co-Validation Setup

II. LIMITATIONS OF THE TESTBENCH OF EMULATOR

A. *Single threaded C-Testbench*

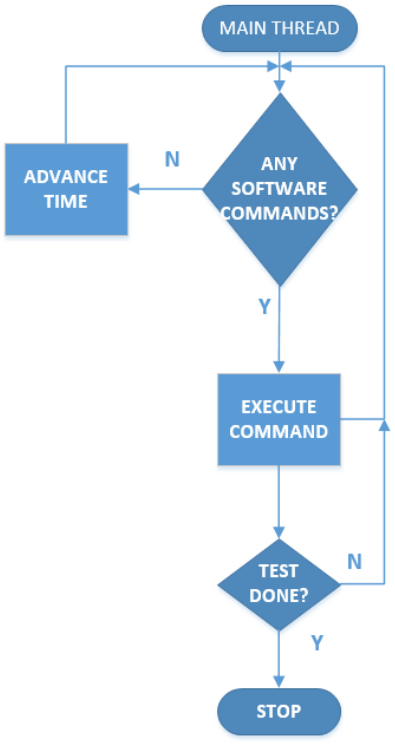

Figure. 2: Single Threaded C-Testbench

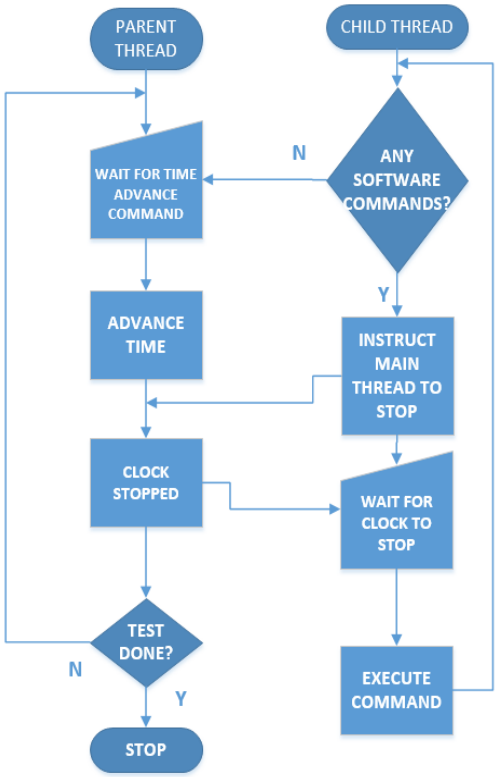

Figure. 3: Multi-Threaded C-Testbench

The Fig. 2 above shows the single threaded C-Testbench flow chart. The C-Testbench is either executing a command from the software or advancing the clocks of the emulators. The emulators start and stop the clocks based on how the C-Testbench instructs them to behave. Since the C-Testbench is single threaded in nature it can either instruct the emulator to advance the clocks or check for any commands from the software side. Due to the single tasking mode of the C-Testbench, both the emulator and software execution is slowed down. This can be improved by making use of a multi-threaded C-Testbench as shown in Fig. 3 in which a parent thread takes care of advancing

clocks in the emulator and a child thread that takes care of servicing commands from the software. The execution of both hardware and software are improved in this case.

B. Clocking related slowness

The emulators use a single root clock and derive all the clocks of the hardware under test by dividing down from the root clock. If all the clocks in the hardware are integral multiples of each other, the division is straightforward, and the fastest clock is as fast as the emulator root clock. However, if the hardware clocks are not integral multiples of each other, the division may not be straightforward leading to poor emulator run-time performance. The simplest approach would be to take the least common multiple of the time periods of all the hardware clocks to determine a virtual clock which will be as fast as the root clock and all the hardware clock periods will be an integral multiple of this virtual clock. These hardware clocks cannot be as fast as root clock since the virtual clock is not used in hardware. This is typically how the clocks are generated in emulators; though it can vary between emulator vendors, the underlying issue will be the same. For hardware software co-validation it is important to validate all the different clocking configurations to avoid any programming issues related to clocking in silicon especially if the software controls the clocking. It is important to test this out in pre-silicon phase, but the emulator efficiency will be drastically reduced in a fully configurable clocking mode. To circumvent this problem, a runtime configurable mode was created for emulation wherein the user could choose to clock the emulator in a fully configurable mode (frequencies selected according to specification) or in a performance optimized mode (frequencies selected were integral multiples of each other). This way, one could choose to use the performance mode to validate non-clocking related programming from the software thereby achieving better efficiencies.

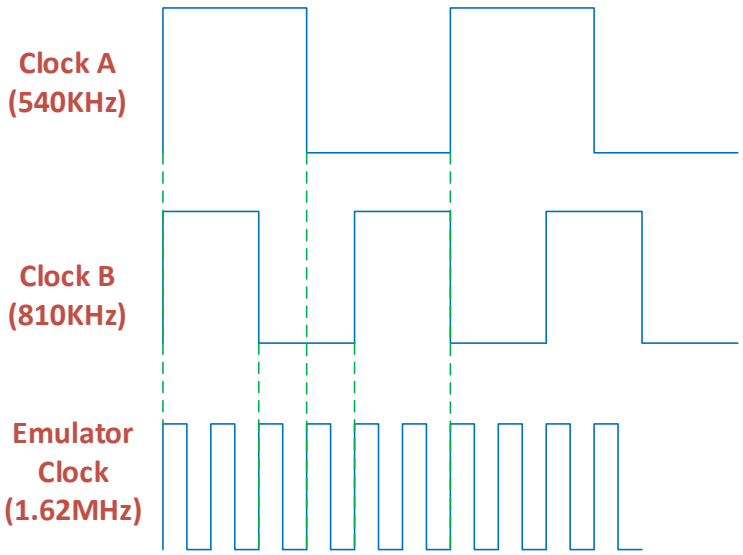

Figure. 4: Regular mode

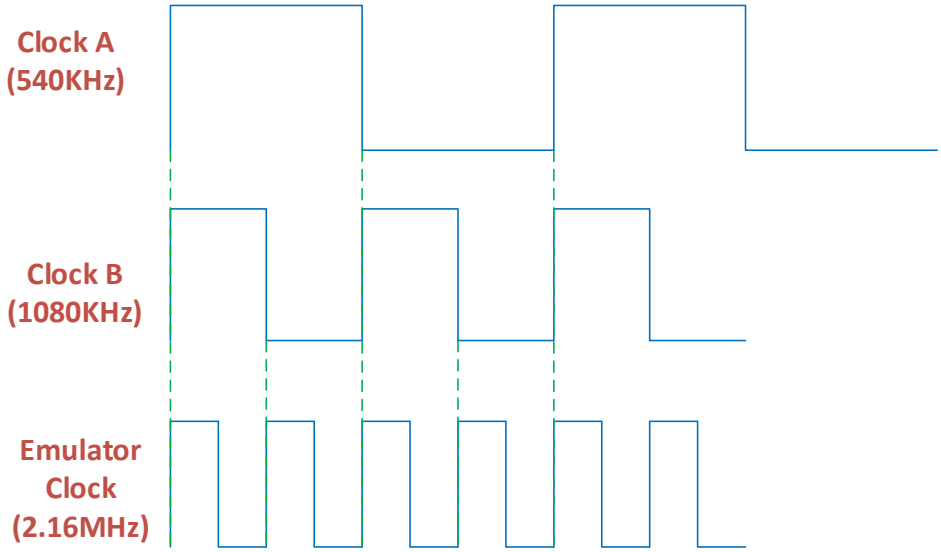

Figure. 5: Performance mode

III. RESULTS

The use of multi-threaded testbench improved the clock efficiency (Total HW clocks) by 10% and the performance optimized mode of clocking resulted in a 3x improvement in the wall clock times.

	Large Test (Runtime)	***Medium Test (Runtime)***	***Small Test (Runtime)***
Single Threaded C-Testbench	106 mins	80 mins	52 mins
Multi-Threaded C-Testbench	46 mins	31 mins	23 mins
Performance Mode	35 mins	29 mins	22 mins
Performance Mode + Multi-Threaded C-Testbench	19 mins	15 mins	9 mins

IV. IMPACT

A. Software Execution Speed

When there is tighter time to market schedules, it is very important to increase the execution speed so that different testcases are covered. This execution speed is often limited by the number of clock cycles required to run a single testcase. Often software runs real world test content which takes millions of clock cycles and this limits software in validating all possible configurations within required time in pre-silicon. The addition of testbench optimizations has increased the software execution speed by 5x. These optimizations have enabled software to run different configurations covering most of the features. The other advantage is the debug time has reduced since the validators can run same test scenarios with the fixes or dump waveform databases very fast which would have taken many hours before. As more content is run, there is a high chance of finding bugs as early as possible in the project lifecycle which reduces overall cost. Finding bugs later in post-silicon is not only expensive but also pushes the product release dates.

B. Increase test content coverage

With new features and new content getting added to each of Intel products it is of utmost important to get 100% coverage in pre-silicon. Design must run millions of clock cycles for all different configurations to attain 100% coverage and with the slower execution speeds, pre-silicon was not a viable solution. With the improvement in wall clock by adding the testbench optimizations software can run many different test cases covering vast number of features there by able to attain full coverage right in pre-silicon. This has opened doors to bring in new customers who used only post-silicon for validating most of the features. Also, using these techniques coverage has increased by 5x in pre-silicon.

V. CONCLUSION

To catch all design bugs, one must rely on pre-silicon verification as it has advantage of full observable and deterministic when compared with post-silicon verification. Post-silicon verification has higher advantage when it comes to execution speed but has limited observability and fixing the bugs is expensive, so it is key to get pre-silicon execution speeds to as close to real silicon. The techniques listed in this paper i.e. use of multi-threaded testbench and performance optimized mode comes with their own caveats and does not suit for all validation scenarios.

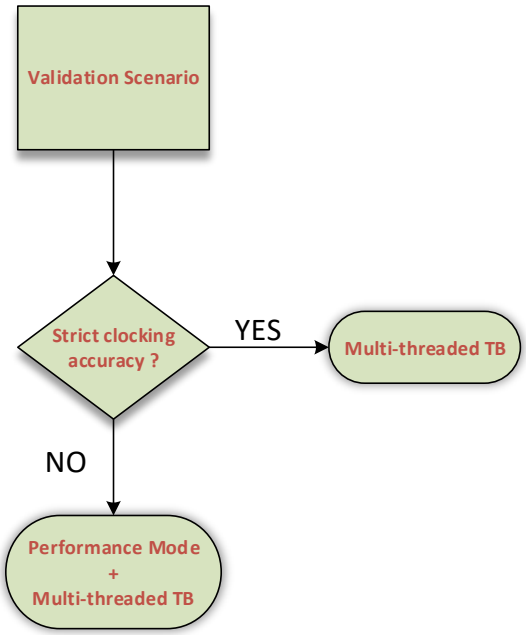

Figure. 6: Testbench Optimizations

If design does not have any strict clocking accuracies, use of multi-threaded testbench and performance mode improves the efficiency by 5x. Apart from improvement in wall clock times, the other advantage with these optimizations is they are runtime changeable and users can pick these optimizations based on their validation scenarios.

VI. REFERENCES

[1] *Standard Co-Emulation Modeling Interface (SCE-MI) Reference Manual - Version 2.4 - November 2016 - Accellera ITC*

[2] *Integrating SystemC Models with Verilog and SystemVerilog Models using the System Verilog Direct Programming Interace [DPI] – Stuart Sutherland, Sutherland HDL, Inc.*

[3] *Post-silicon bug diagnosis with inconsistent executions – 2011 IEEE International Conference on ICCAD – December 2011*