UVM Acceleration Using Hardware Emulator at Pre-silicon Stage

Sunil Roe YunGi Um Hyunwoo Koh
Hyunsun Ahn Youngsik Kim Seonil Brian Choi

Samsung Electronics Co., Ltd.
{sunil.roe, yungi76.um, hwkoh, hyunsun.ahn, ys31.kim, seonilb.choi}@samsung.com
1-1, Samsungjenja-ro, Hwaseong-si, Gyeonggi-do, Korea

***Abstract* - Since the complexity of mobile SoCs increases significantly, simulation-only-based verification can't achieve a high quality verification for a given time. In this study, enabling UVM acceleration environment where the simulation is accelerated at the pre-silicon stage is discussed. To maximize hardware resource use and increase the speedup, acceleration candidates are identified through simulation profiling. In addition, by introducing a dual-domain framework, the existing UVM environment can be reused in the hardware acceleration. An automation flow is also proposed that eliminates unnecessary debugging and iteration caused by OOMR (Out-Of Module Reference) signals. As a result, the speedup of 17 ~ 21x is achieved.**

I. INTRODUCTION

The complexity of SOC platforms including both hardware and software significantly increases since more features and performance are required to process recent mobile applications. It requires more development TAT (Turnaround Time) and production cost. However, in order to achieve time-to-market, the development period stays the same or even less. The verification TAT also needs to be shortened.

The current SOC design verifications use UVM (Universal Verification Methodology) simulation environment. While its simulation environment can be developed rapidly, its simulation run time is slow at the SoC level. In comparison, the emulation is 10x to 100x faster than the simulation. Also, the performance degradation in emulation for a large design size is not so high [1]. However, it is not efficient to use the existing UVM based simulation environment to take advantage of the high emulation speed. Thus, the verification methodology utilizing and improving the existing UVM simulation environment in a hardware emulator is proposed [2]. Both performance enhancement by the hardware emulator and the reuse of simulation environment are achieved.

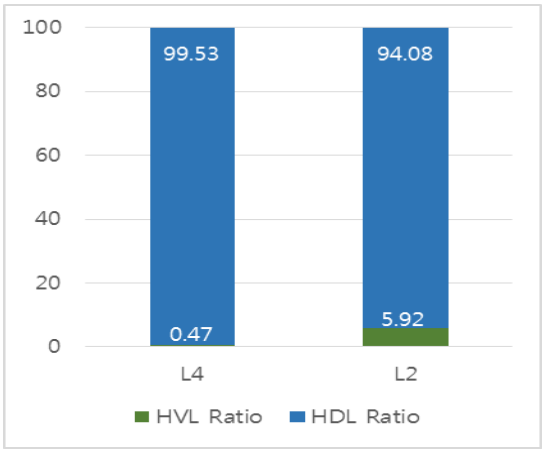
In this study, the challenges to create the UVMA (UVM acceleration) environment at the pre-silicon stage are discussed first. Section II describes simulation profiling results splitting simulation tasks and hardware tasks. In Section III, the flow to build UVMA environments is discussed. The results on CPU core emulation and the power aware emulation with UPF (Unified Power Format) in SOC are described in Section IV.

II. SIMULATION PROFILING

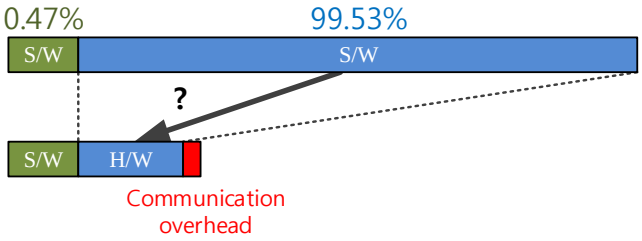
Basically, if the DUT (Design Under Test) resides in the hardware, operations on the DUT would be much faster than the software simulation. However, since all the test vectors used in the existing UVM environment cannot be executed in the hardware, some of them are divided into hardware part and some of them are divided into software part. In this case, if the proportion of the hardware side in the total test time is not large, the emulator cannot be used effectively because there is little room for acceleration using the actual emulator.

In addition, the number of simulation vectors for design verification is very large in thousands of units, but the hardware emulator resources are very limited due to its cost and it is difficult to perform a large number of simulation vectors at the same time. Therefore, in order to maximize the return on investment (ROI) of the emulator, it is necessary to select test vectors that can have a great effect on acceleration and use the emulator preferentially.

Intuitively, it is clear that the acceleration effect of the test vector that operates only in the hardware part will be greatest because there is almost no interruption from the software part. The more communication there is, the more overhead will occur and the performance will be degraded. However, it is very inefficient to analyze the source code of all vectors in order to understand the properties of vectors. So the profiling tool provided by the simulator can be used to figure out which vectors will have a great effect on the emulator.



(a) Hit ratio distribution comparison between CPU vectors(L4) and NonCPU vectors(L2)



(b) Acceleration time estimation

Figure 1 Simulation Profiling and Acceleration Estimation

Figure 1 (a) shows the hit ratio profiling results for the CPU operation-oriented vector (L4) and the CPU-free vector (L2) at the simulation stage. The Hit ratio is a concept that shows the percentage of the signal toggle that occurs when the test vector is executed. The upper blue part of the graph is expected to be executed in the hardware part, and the lower green part is expected to be executed in the software part. As predicted earlier, 99.53% of hits are expected to occur in the hardware part because there is not much communication with TB in the L4 test based on CPU operation. On the other hand, the L2 vector for general IP verification vector with no CPU has a relatively large amount of communication with the TB, so the hardware part accounts for about 94%. Therefore, it can be seen that it is more efficient to accelerate the L4 test vectors that have more than 99% operation in hardware part.

Figure 1 (b) shows how much acceleration will be achieved for L4 cases where more than 99% hit occurs in the hardware part. Because 99.53% of vector execution under simulation environment can be executed in the hardware part under emulation environment, execution time will surely be saved. However, communication overhead between the hardware part and the software part should be considered. Therefore, the acceleration time should be calculated considering software part execution time, hardware part execution time, and communication execution time.

III. UVMA (UVM ACCELERATION) ENVIRONMENT

A. Dual Domain Framework

The conventional verification environment generally consists of a single top that has both synthesizable and non-synthesizable components. DUT and UVM components including UVM driver, monitor and sequencer are instantiated in it all together. But, traditional single TB (testbench) should be divided into two part - HVL (Hardware Verification Language) domain and HDL (Hardware Description Language) domain to be able to use hardware emulator **[3].**

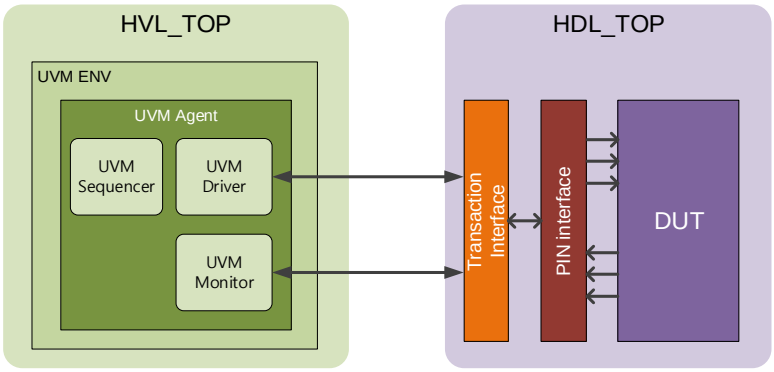


Figure 2 Dual Domain Framework

However, there is a limitation in using this structure. As mentioned in the previous section, minimizing the communicative overhead between the HVL domain and the HDL domain can increase the effect of hardware acceleration. If test vectors are created without this consideration, the results could be slower than simulation. This is illustrated by an example of the PPMU (Platform Performance Monitoring Unit) responsible for performance analysis within the SOC.

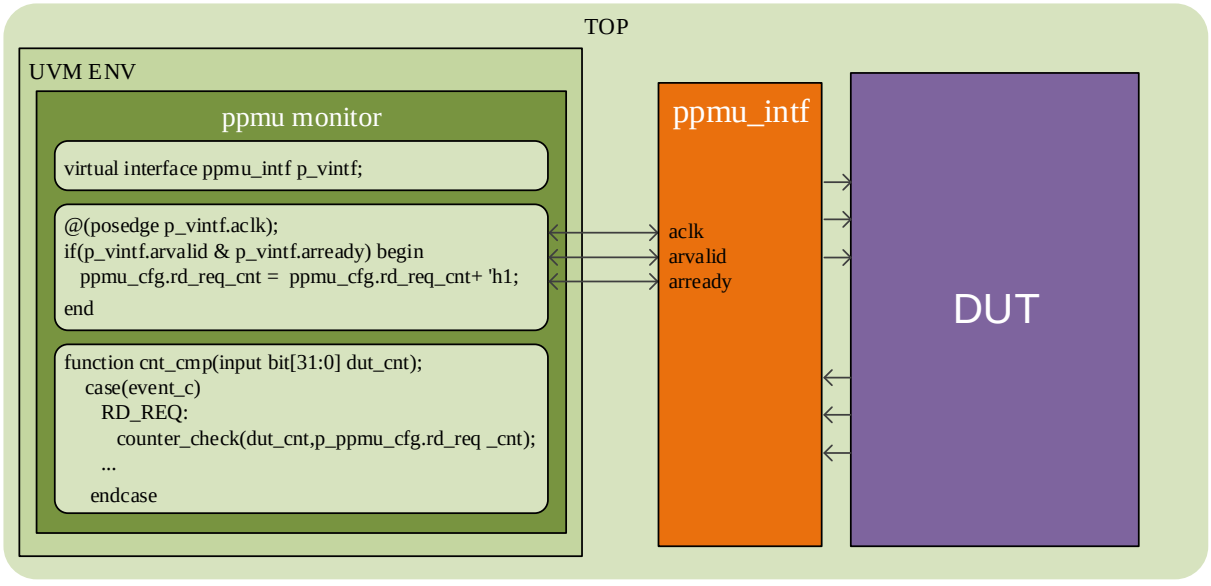


Figure 3 PPMU monitor and interface in UVM simulation environment

Figure 3 shows the UVM simulation environment composed of one TOP, which is expressed in the monitor part of PPMU. The PPMU monitor stores the number of arvalid and arready signals in the rd_req_cnt in the PPMU monitor through the interface to the PPMU block. To do this, the arvalid and arready signals are checked every clock cycle of ppmu_intf. This structure has no problem in the simulation environment, but it can cause serious performance degradation in the UVMA environment separated by HVL_TOP and HDL_TOP.

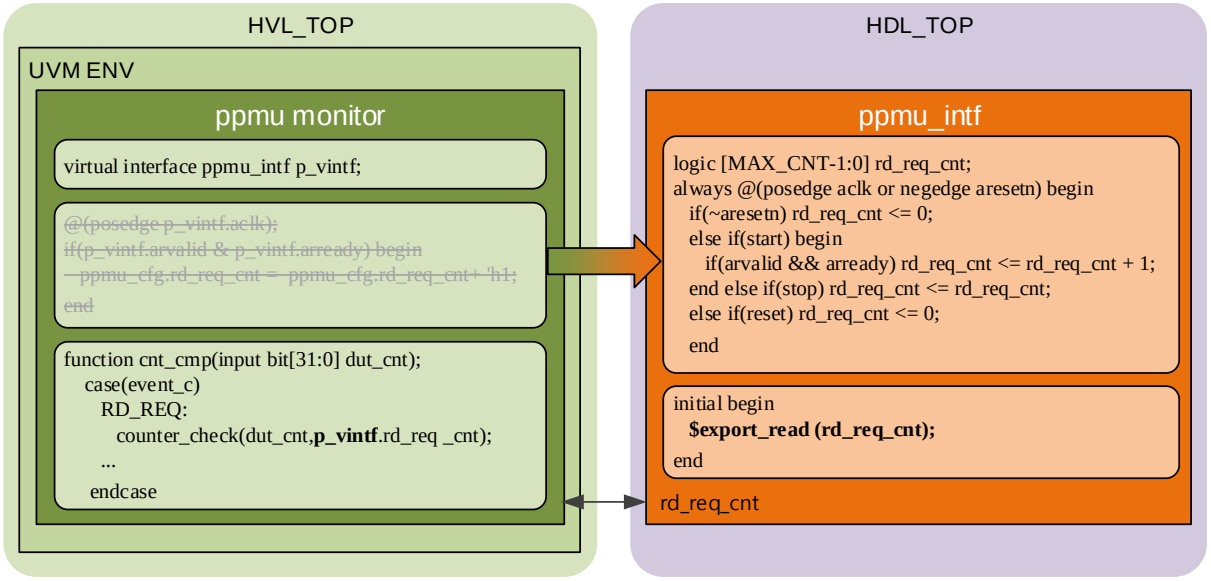


Figure 4 PPMU monitor and interface in UVMA environment

Figure 4 shows the PPMU monitor and interface when configured as a dual domain for UVMA in the single top environment of Figure 3. If aclk of ppmu_intf is used to add rd_req_cnt in the monitor, event between HVL_TOP and HDL_TOP should occur whenever aclk is toggled, so serious performance degradation may occur. Therefore, the part checking rd_req_cnt is moved to the PPMU interface to be calculated in HDL_TOP. In the PPMU monitor of HVL_TOP, the rd_req_cnt in the PPMU interface is changed to be used only when needed later. This allows the same function to be performed without degrading the performance of the hardware emulator. At this time, because rd_req_cnt used in HVL_TOP is located on the HDL_TOP side, it must be registered as an export signal on the HDL_TOP side to be accessed from the PPMU monitor of HVL_TOP.

B. Simulation Acceleration Flow

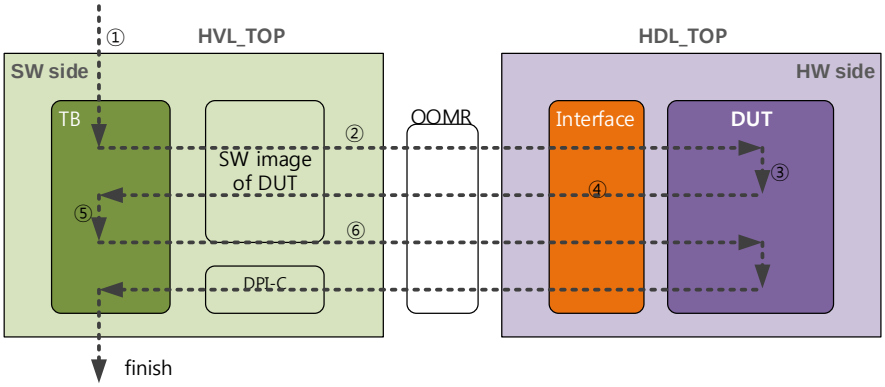


Figure 5. Running Flow of UVM Acceleration

Figure 5 shows the concept of simulation acceleration with a hardware emulator. The right part of the picture means the hardware side including DUT and interfaces. The left part describes the software side consists of TB, software image of DUT and DPI-C interface.

Each number in
Figure **5** represents the running flow of the simulation acceleration. The explanation of each number is as follows.

① As the pure simulation, start from 0 time on the software side first.

② Run on the software side and execute the swap command to transfer control to the hardware side.

③ On the hardware side, a vector for the design of the DUT is performed.

④ If delay of TB is expired during vector execution, or call is made on TB side, hardware-software sync process is performed and control is transferred to software side.

⑤ Execute the vector statement described in TB.

⑥ When the TB statement is finished, control is transferred to the hardware side again, and this process is repeated until the simulation is finished in the software side.

At this time, the signal area between the HVL and the HDL is called OOMR (Out Of Module Reference), and access permission must be grasped from the HDL side that the TB accesses the corresponding signal. This is discussed in the OOMR Signal Handling section below.

C. OOMR (Out Of Module Reference) Signal Handling

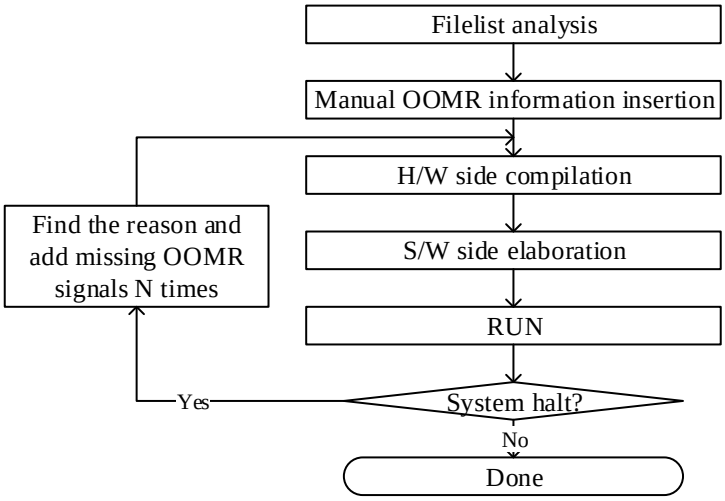


Figure 6 Current - Manual OOMR Insertion Flow

Figure 6 shows the current UVMA environment construction process. After analyzing the files to be included in the HDL side, the compile process for the H / W side is performed. At this time, the information about the signals accessed from the TB should be specified as well. The OOMR information is analyzed by reviewing the contents of the TB and then entered manually to proceed with the build. After completing the hardware compilation, we proceed to elaboration on the software side. And when the image is generated, test vector is executed to verify the desired function.

However, the test vector is sometimes performed well and sometimes stops at a specific point. In order to find the point where the vector stops, it is necessary to generate a waveform and debug for a long time. In many cases, the problem occurs because OOMR signal is not added. The found OOMR signal should be added to the OOMR information list and H/W side compilation step is repeated from this step. If we are lucky, we can get the result of test vector through only 1 iteration. But we may need to repeat the building process several times because it is sometimes difficult to extract all OOMR signals at one time.

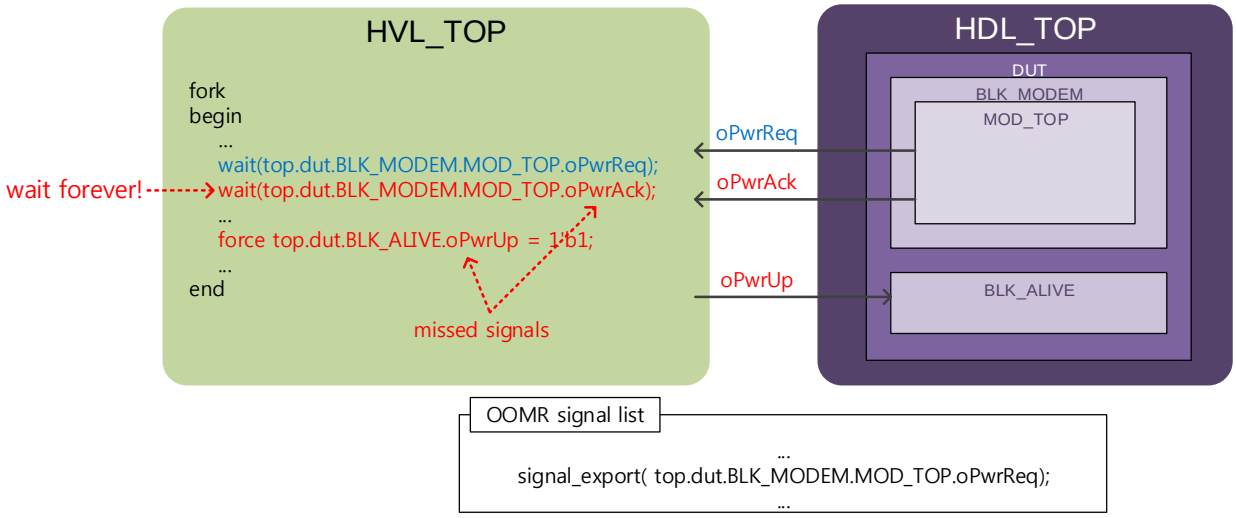


Figure 7 frozen test vector caused by missing export signals

Figure 7 illustrates the problem that occurs when OOMR is not included in the export signal list. oPwrReq signal is included in the OOMR signal list, while oPwrAck signal and oPwrUp signal are not included in the list because they cannot be detected in UVMA environment migration. If we build the emulator image using the incomplete signal list and run a test vector on the image, it will work well to wait statement of oPwrReq signal, but it will not proceed any longer waiting for oPwrAck signal. So, we have to create a waveform and debug the problem. Furthermore, dumping a waveform in the emulator takes much longer than that under simulation. This means that we have to devote much time to the debugging job that we do not have to, which ultimately reduces the efficiency of UVMA.

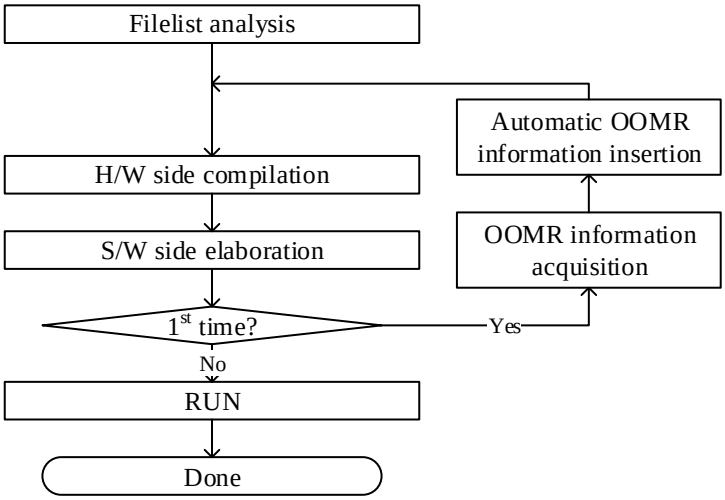


Figure 8 Proposed - Automatic OOMR Insertion Flow

Figure 8 shows how to solve the inconvenience of manually entering OOMR information. The basic build flow is similar as that shown as Figure 6, but it focuses on automatically entering OOMR information to eliminate iterations due to missing information. To extract full OOMR information, hardware compilation and software elaboration should be performed once. Once executed, there would be no missed signal because the build software outputs the OOMR information used in the test vector to be run in the tool. The generated OOMR information is automatically used as input to the second image build, and we can get the desired result with a single iteration. An image build process is required once unconditionally, but this methodology is useful because we can focus on the original test vector, reducing unnecessary work looking for missing signals.

D. Design Library Handling

The synthesizable model of design library can be generally created from the liberty file (IEEE standards containing PVT characterization, input and output characterization, Timing, Power, and so on). But, the liberty file of the specific design library can be invalid at the early stage, so the synthesizable model cannot be created. In this case, the synthesizable model should be generated manually, and this job was one of the main reason which disturbs flow automation. So, we decided not to make the custom synthesizable model when the liberty file is absent. Instead, we use the simulation model which can be executed on the host machine. Then we can build up the automation flow without any compilation error and can run the compile image on the hardware emulator. Of course, the emulation performance can be degraded when the simulation model is used on the host machine due to communication overhead between the emulator and the host machine. But overall performance doesn't slow down much because these kinds of models are not used so much at the early stage.

E. Power Aware Emulation Setup

In general, when configuring an image to use a hardware emulator, not all RTL blocks of the SOC are included. This is because the price of the hardware emulator is expensive and the resource must be divided and used by many people, so the image must be created using only the necessary SOC. Moreover, because the emulation compile time compared to the simulation compile time is 2~3 times longer, eliminating unnecessary blocks can sometimes be a useful technique.

Power aware emulation can be performed with UPF as simulator does. But, compilation errors can occur due to information mismatch between RTL and UPF when some RTL blocks are stubbed out because of machine resources or compilation time. The compilation tool should handle the UPF information at the black boxes without errors.

F. DFS (Dynamic Frequency Scaling) emulation setup

In general, necessary clocks to SOC are directly forced to output port of PLL on the emulator. To execute DFS test vectors, many numbers of clock sources are required. If all clock instances are created at the top in advance, emulator performance is degraded. So, only one clock instance should be created in advance and clock frequency according to parameters of PLL should be defined as local parameter. Then, we can assign local parameters according to parameters to the clock instance without performance degradation.

IV. EXPERIMENTAL RESULTS

We performed our experiments using our mobile AP (Application Processor) according to the proposed flow. We chose CPU operation-oriented test scenarios because those kinds of tests usually take long time and have less interruption from TB. Total number of scenarios is 137 including 16 power aware simulations, and all tests are executed on the hardware emulator without any modification. TABLE 1 shows simulation time, emulation time, and speed-up factor of normal scenarios and power aware scenarios respectively.

TABLE 1
Result of UVM Acceleration – CPU block

Normal tests				Power aware tests			
Test Vectors	Simulation Time(h)	Emulation Time(h)	Speed-up	Test Vectors	Simulation Time(h)	Emulation Time(h)	Speed-up
121	217.7	12.8	x17	16	90.5	4.4	x21

According to the result, the speed-up of normal tests is 17 times and that of power aware tests is 21 times. This score is quite impressive, but speed-up rates are not so high than expected. The reason is thought to be the design size of mobile AP used in this test and the simulation time of test vectors. The size of mobile AP used in this test is just about 100MG and the simulation performance is relatively fast. So, average simulation time and average power aware simulation time are just about 2 hours and 5 hours respectively. Therefore, this result cannot show the advantage of the hardware emulator fully. The effect of speed-up will be remarkable when we use large-size mobile AP over 1000MG which has very slow simulation time such as over several days.

V. CONCLUSION AND FUTURE WORK

In this research, we described how to deal with difficulties occurred when we are trying to make UVM acceleration environment in the pre-silicon stage.

First, we measured the potential acceleration of the test vector through simulation profiling to efficiently use limited hardware resources. Based on the results, the priority of the test vector to perform the hardware acceleration was selected. Second, unlike the simulation where everything is done on the host machine, the UVMA environment is made up of the HVL domain and the HDL domain. Since the communication and synchronization between HVL and HDL are the main factors of performance degradation of the emulator, we moved the clock related codes of the HVL domain to the HDL domain. Third, we examined that the HDL domain should be given access permission to the OOMR signals between the HVL domain and the HDL domain. Since the iteration may have a serious adverse effect on the schedule, we proposed a flow of adding OOMR with only one iteration by automating this task that eliminates exhausting iterations. Finally, we used non-synthesizable simulation model for invalid liberty file and set up power aware emulation and DFS emulation.

According to the pilot test result, we can get about 17~21 times speed-up comparing pure simulation time without any test vector modification. Even though the speed-up result is lower than expected, we can estimate that the result of large-size AP will be better.

REFERENCES

[1] L. Rizzatti, "When to use simulation, when to use emulation," *Electronic Products*, 01 09 2014.

[2] H. Schoot and A. Yehia, "UVM and Emulation: How to Get Your Ultimate Testbench Acceleration Speed-up," *DVCON-EUROPE*, 2015.

[3] S. Hemant and H. Schoot, "Bringing Verification and Validation under One Umbrella," *Verification Academy*, 02 2013.