2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

OS-aware IP Development Methodology

Hyunjae Woo, Woojoo Kim, Youngsik Kim, Seonil Brian Choi

Samsung Electronics Co., LTD. (hyunjae.woo@samsung.com)

Abstract: In this paper, we propose OS-aware IP development methodology. The main purpose of this methodology is to enable both SW/HW co-design and co-verification prior to building matured SoC, thus contributing to the reduction of total development period for mobile APs by securing early-verified SW/HW IP. The proposed methodology adopts co-emulation of QEMU[1] simulation and HW emulation. Most of the IPs such as CPU, eMMC, CLCD and peripherals, which are used frequently for Android platform boot-up, are allocated to the QEMU simulation side, which maximizes run-time performance of SW. The only target IPs and memory subsystem are allocated to HW emulator side to help start SW development with RTL (Register Transfer Level) design. Except the target IP, all of IPs are the QEMU library models which are already verified. It is fast to bring-up the Platform including compilation and running time. Proposed transactors for external communications such as memory storage, CPU and GIC BFM(Bus functional model), operate in the same way as real HW devices work, and proposed memory coherent interface reduces emulation time by minimize counts of communication. Application results show that booting an Android platform can be completed within 10 minutes on the proposed methodology, and show that SW driver and IP's firmware are successfully developed and validated with RTL before the silicon based development board is ready. Finally, the proposed methodology also helps enhance quality of SW by saving more time for design and validation for SW developers

1. Introduction

Due to the increment of SoC size and complexity, there are a lot of potential technical bugs and issues which can be detected by HW/SW co-verification only. Since HW/SW co-verification is usually performed by running real application on a full SOC emulation at the last stage of design cycle or on a silicon, bugs or issues that are detected by co-verification can delay entire project schedule to resolve the issues by ECO(Engineering Change Order), SW workaround, or even re-spins.

In mobile AP industries, several solutions have been proposed for the reduction of HW/SW co-verification TAT. One of the representative solutions is the hybrid emulation platform which we will describe in Section 2-3, where the CPU is replaced by the ARM fast-model as SW emulation, to accelerate the boot-up speed of OS in full SoC. This emulation methodology, in general, resulted in achieving more than 15 times speed-up on OS booting compared to conventional full chip HW emulation which we will describe in Section 2-2, because the ARM fast model is not CA(Cycle Accurate) model but FA(Functional Accurate) model, which is beneficial for full SOC SW validation point of view.

From the view point of IP design and verification, however, there are some constraint to use hybrid emulation platform directly. When we are interested in verifying our target IP, all other IPs are supposed to be running properly such that other IPs do not cause any side effect such as kennel panic, lock-up and so on. However it is not the case because the maturity of other IPs are not that high either. Also, porting entire SOC to emulation is still heavy and slow, so we propose a new methodology to remove all dependency and constraint with achieving 5times faster android boot up than hybrid emulation platform.

2. Related Research (Previous work)

This chapter introduces conventional SOC develop flow focusing on SW development and verification environments include accelerating by emulator. At the beginning of project, simulation is the best choice to verify IPs and components because it is easy to setup environment and the fastest compile time. Once RTL is getting

matured, the emulation is used to use. After basic pure SoC emulations is passed, Hybrid SoC emulation setup start to run with OS (Android).

2-1. Simulation

At the beginning of the IP design and Verification, RTL simulation is the best candidate for the following reasons. It is easy to setup simulation environment including generation test bench and test vectors, also compilation time and loading time is much faster than emulation. Before RTL is stable, there are a lot of iteration both RTL itself, TB and vectors because of bug fix. However, simulation speed is inversely proportional to design-scale. As the design size grows, simulation speed reduces exponentially. In case of several hundred million gate scale, simulation speed can be dropped to a few dozen KHz [2]. This simulation speed is 10^6 times slower than silicon speed. Therefore, the SW which runs during one second on silicon can consume 10^6 seconds (=11.6 days) for simulation. It is almost impossible for simulation to be used for SW development and verification with long-time duration.

2-2. Pure Emulation

Once RTL is matured, meaning that the basic functionality checks are finished, it is time to port the RTL to emulator to accelerate RTL simulation. It is known that speed of emulation is 1000 times faster than that of simulation. The emulation speed can be from several hundred KHz to several MHz [3][4]. The emulation speed is 10^3 times slower than silicon, it can be applied to SW development and verification. Also, SW which runs during one second on silicon can consume 10^3 seconds (=16.7 minutes) for running. The emulation can be adopted to develop SW such as device drivers which are relatively simple. However, it is not fast enough to be used for the development of Linux kernel and Android Platform which are relatively large scale and complex. Because it can take more than ten hours to boot Linux and Android using emulation platform. Because of above TAT (Turn-Around-Time) constraint, full chip SOC emulation is used to use only for bare metal tests or Linux based simple SW initialize tests

2-3. Hybrid SoC Emulation methodology

Hybrid SoC emulation methodology is the best solution to overcome TAT constraint of pure HW emulation. It uses co-emulation methodology. CPU and memory components are allocated to ESL virtual platform side with TLM LT (Loosely Timed) models, because they issue a lot of transactions during boot up time of Android Platform. Other IPs are allocated to a HW emulator side. Because CPU emulation (ARM Fast Model which is FA model) run over Linux host server, so it is much faster than real CPU over emulator which run ~5 MHz actual operating frequency. As shown Figure1, only CPU and Memory are virtualized to accelerate performance and there are transactors to communicate between virtual platform and emulator. As the result of Hybrid SoC Emulation methodology, it only takes 49min and 15 times faster simulation speed for Android boot-up compared to that of pure HW emulation systems [5].

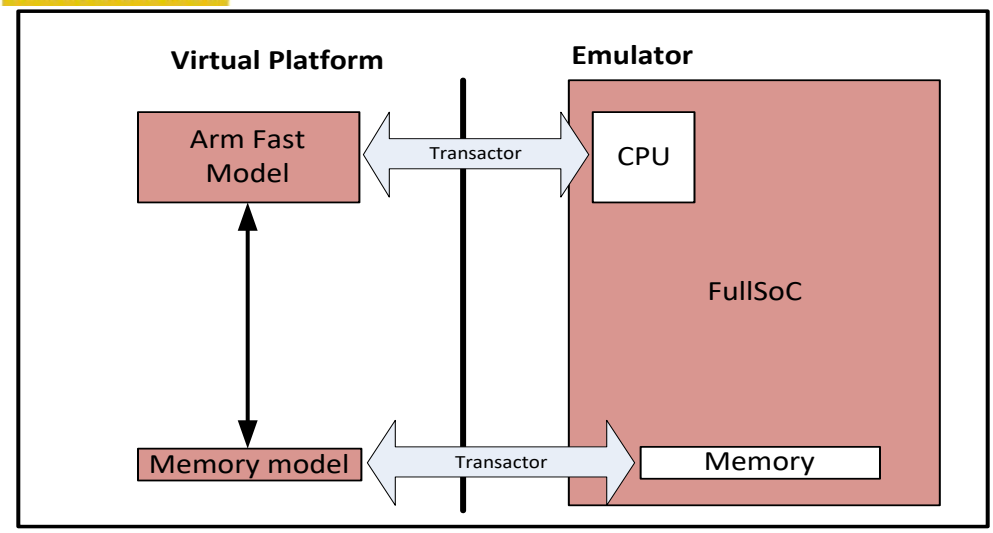

Figure 1 Hybrid SoC Emulation methodology

2-4 Conventional SoC development flow

Hybrid SoC emulation methodology is fast enough for early SW bring up for android, but as shown in the Figure 2, it is setup just few months before MTO (Master Tape Out). It is because it is Full SoC based CPU Hybrid, there are steps which are necessary to build platform, all IPs inside of SoC should be verified and integrated SoC and drivers also should be ready. So as IP's SW developer's perspective it is not enough time to use Hybrid SoC emulation methodology to develop and verify their SW over the OS (android).

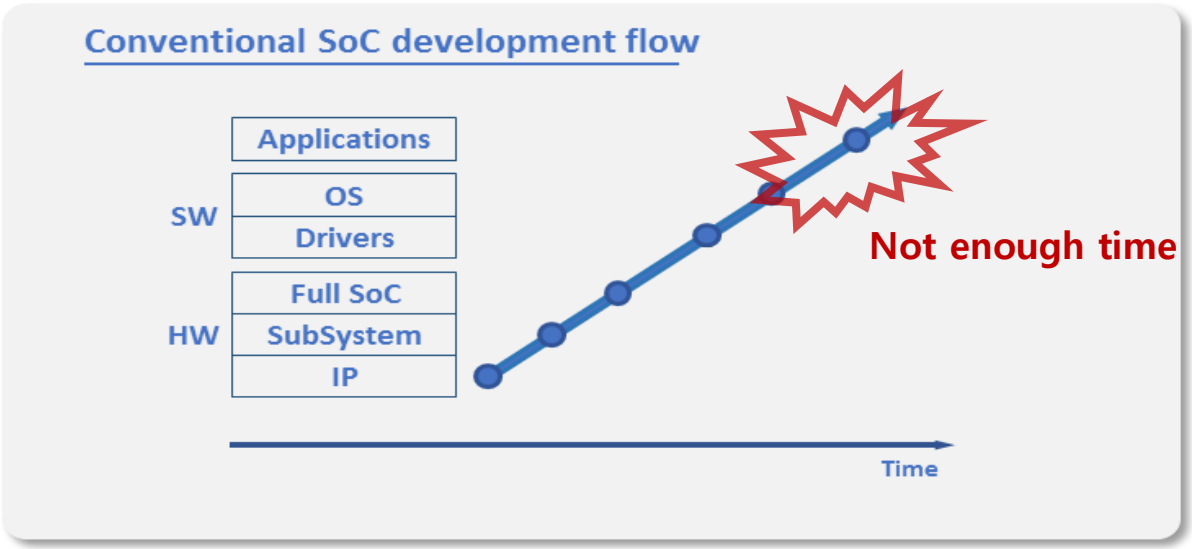

Figure 2 Conventional SoC development flow

3. Proposed Platform

We focused on IP design and verification view, and the goal is to do the SW/HW co-design and co-verification as early as possible. To do this, we have built a platform such that all components related to OS booting are located at virtual side, and the target IP and related components are located at real emulation side, as shown in figure 3.

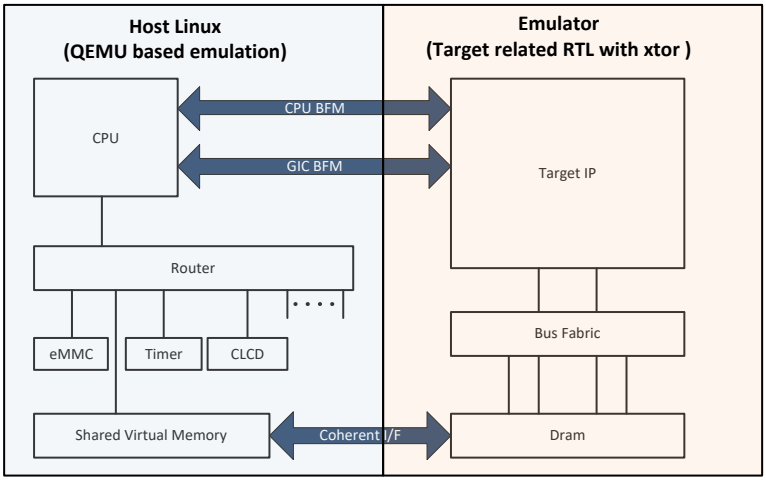

Figure 3 Basic concept

We used QEMU to emulate above virtual side environments as it is already verified open-source. Since OS boot up, benchmark (app) installation and loading them are mainly performed on virtual side, time for the initialization of IP can dramatically reduce such that it is 5 times faster than Hybrid platform.

Moreover, all other IPs except target IP, are located at the virtual side hence we do not need unnecessary time to validate HW/SW for those IPs. For example, we used to spend almost one month to resolve other IPs on a hybrid platform such as audio IP's kennel panic and 2D compositor IP's segmentation fault and so on. These problems can be caused by two factors. The first factor would be from the IP bug from the RTL, or the relevant driver. Another factor could be the totally different time domain between Virtual and Emulator, leading to kernel panic due to timeout. So this methodology could enable IP developer to concentrate on target IP itself. SW developers could also start their work if there are functional model for IP by putting function model on virtual side, and when RTL becomes ready we can easily replace function model with real RTL seamlessly. The following sub-chapters explain more detail about how to implement, further usage model and how to validate.

3.1 The Transactors for communication Host Linux and Emulator

Host Linux machine (Run time host) and Emulator connected by PCIe (Peripheral Component Interconnect Express), so all of transaction through PCIe and the concept of Transactor is now common industry solution which consists of front end Interface and back end interface by physical interconnect (PCIe). Figure 4 shows the example of CPU BFM(Bus Functional Model) transactor, Front End I/F control transactions for reading and writing data from host side, Back End I/F for converting those transaction to Bit-Signals and Target IP that are physically connected by bit-signals. In this paper there are only 3 transactors for CPU, GIC (General Interrupt Controller) and Coherent Memory Interface, but if target IP needs more transactor such as camera or sensor and so on, it is easy to generate or use vendor supported transactor library named AVIP (Accelerate Verification IP).

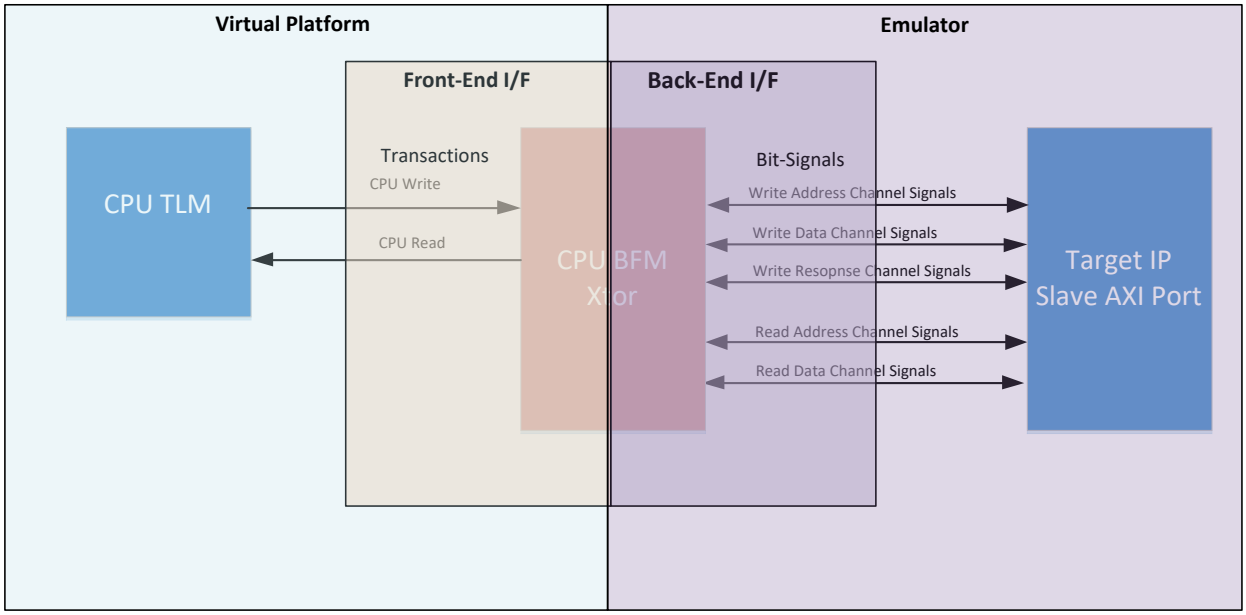

Figure 4 Transactor diagram

3.2 Memory Coherence between shared virtual memory and Emulator

This chapter explains the emulator communication issues, followed by memory coherence. The bandwidth of PCIe is huge enough even though it is highly depending on PCIe version and configuration. It is up to 32 GB/s as current emulator, so the bandwidth is not a bottleneck nowadays, but the problem is latency overhead[6]. For example, once target IP that is in emulator sends a bit-level transaction for reading some data from CPU TLM(i.e., Host Linux), the Back-End I/F converts the bit-level transaction into PCIe transaction with address and command, and sends to Front-end I/F through PCIe. Then the Front-End I/F reads data from Host Linux and transfers those data through PCIe to Back-End I/F, and finally Back-end I/F convert it to bit-level transaction and sends to Target IP. During this transaction sequence (i.e. a DPI function call), the emulator's user clock is stopped to keep the latest value which means clock is not proceeding until the transaction is finished, hence it causes significant latency issue that affects entire emulation performance. Moreover, it is hard to reduce latency itself because the latency is affected by various condition such as fixed time by specification, operating simulator and Host Linux condition. So it is necessary to reduce communication counts between Target IP and CPU as less as possible to enhance emulation speed. Below experiments show how it is important to reduce communication counts and how it can be optimized.
According to the PCIe specification, 4 GB data read transaction only takes less than one second. However, it may take more than five sec in the real emulation, depending on various conditions. One way 4 Byte transaction takes 5us~7us on PCIe specification, but it may take 14 us at the best on real. Let us have more extreme situation. If the transactor is supposed to read 4GB from Host Linux and it sends 4 Byte read transaction for 1G times, it will take over 7 hours. But if transactor sends 64 Byte read transaction, it will take only around 10 minutes.
To overcome the latency issue as mentioned before, memory is required to be existed not only in Host side but also in Emulator side. If the memory exists only in Host side, emulator would frequently stop the clock whenever Target IP accesses the memory in Host side. The memory inside emulator side can be a cache, buffer and real memory model like lpddr4, depending on Target IP's characteristic.
But the most important feature is the coherence. If the coherence between Host side and emulator side memory is broken, it will cause critical function bug. If there are a lot of communications between memories of both sides for coherency, the emulation performance significantly will be dropped by latency issues. So some of techniques and assumptions are used. When Target IP needs to read data from DRAM, CPU

fprepares data to DRAM, controls Target IP by CPU BFM, and then Target IP reads data. When CPU needs to read data from DRAM, assuming that the data is modified by Target IP, Target IP raises interrupt by GIC BIM. So memory coherency is updated right after CPU BFM and GIC BFM operation is done. If Target IP's behavior is different as these assumptions, it is necessary to use Timer or instruction counter which up-counting CPU's instruction and when the timer and counter value reaches the setting value then they communicate to make coherence.

3.3 Further usage model for IP Evaluation Platform.

Since the selection of right IPs for each mobile AP (Application Processor) is important, it is unavoidable to perform the evaluation of PPAB (Performance, Power, Area and Bandwidth) among candidate IPs under the same environments. And it should be evaluated before SoC integration starts. For evaluation, there are a lot of consideration points for selection of GPU such as vendor, IP version, and configurations such as how many shader cores, size of L2 Cache and so on. To evaluate GPUs with each configuration, the environment should be small for parallel emulation, but represent real memory sub-system for accurate evaluation. By using proposed methodology, it is really simple to setup the environment for evaluation, by replacing Target RTL and putting driver from vendor deliverable.

3.4 Validation Platform.

To validate our new methodology, we applied our new platform for GPU IP verification, as shown in Figure 5. In this figure, the Common Platform stands for Android ready virtual platform, Target IP is the GPU SW and HW and related bus fabric and memory and Transactor is to communicate between Common Platform and Target IP.

The memory sub-system including Bus components should be the same as real SoC to evaluate performance and verify functionality. Also the memory transactor supports the same hashing equations. Figure 5 show proposed methodology and validation platform.

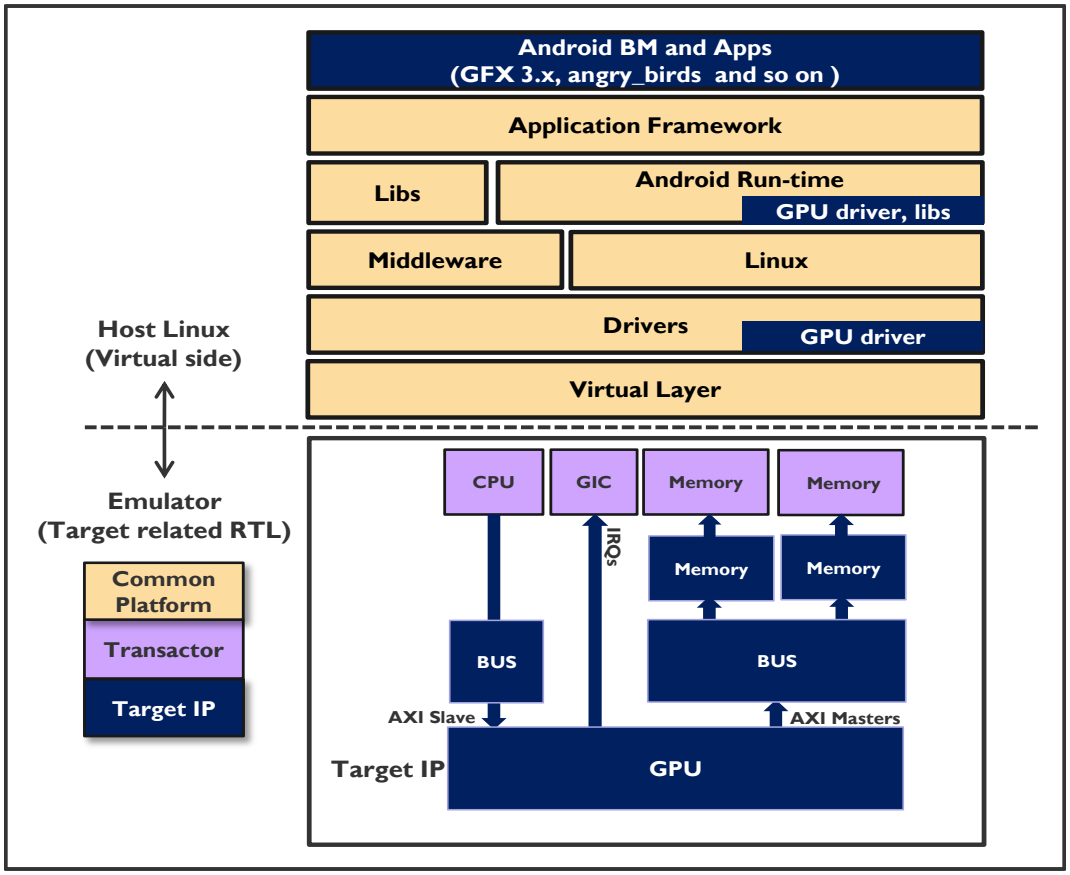

Figure 5 proposed methodology

4. Application Results

We used GFX Benchmark 3.1.0 Manhattan 1080p with 3000ms fixed step time. It take 10 minutes to boot up android Marshmallow, 8 minutes for the APK loading using adb(android debug bridge) and 45 minutes for running the Benchmark. Because it only takes 7 minutes for APK loading, time for rendering one frame takes 1.8 minutes for average.

Figure 6 shows the rendering results of Manhattan and Angry Bird application that are performed on our proposed methodology, and Table 1 shows the summary of our experimental results. In this table, Pure Emulator is the case where full SoC is loaded to real emulator, and Hybrid Platform is the case where the CPU inside of SoC is replaced by Virtual CPU located on virtual side. It is noticed that our Proposed Methodology is 5 times faster than Hybrid Platform, as described before. Because only target released RTL is allocated emulator, it is possible to consume small capacity of emulator and to fast compile comparing Hybrid SoC emulation.

Table 1. Performance comparison.

	Simulation	Pure Emulation	Hybrid SoC Emulation Methodology	Proposed Methodology
Kernel Boot-up (prompt)	125,867 min*	96 min	2 min	1 min
Android Home Screen	741,517 min*	661 min	47 min	9 min

Total Consumed Time	867,384 min*	757 min	49 min	10 min

* Estimated Value

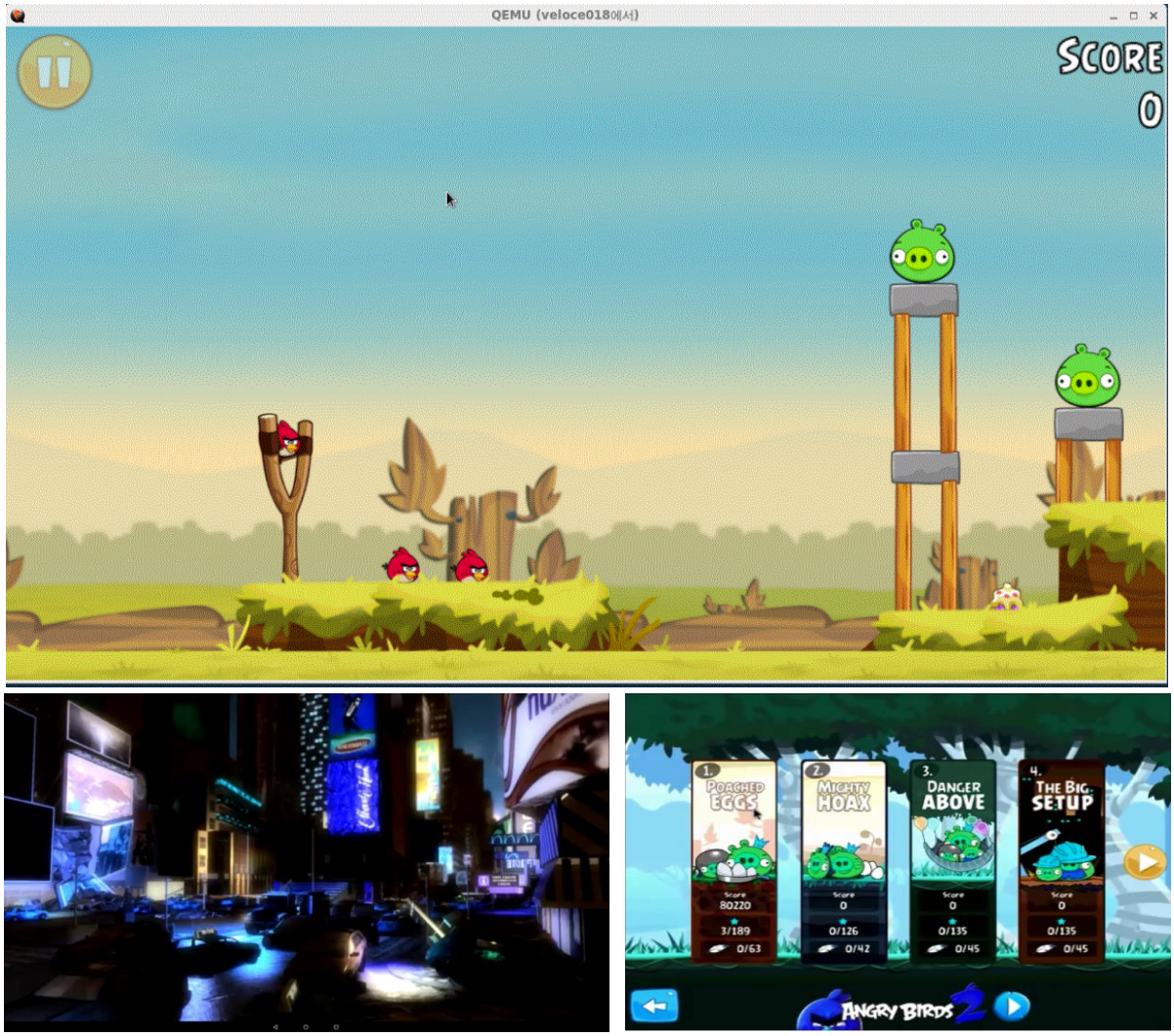

Figure 6 The result image

5. Conclusion

In this paper, a new emulation methodology was presented such that this makes it possible to start development and verification of IP from the earlier stage of design cycle, and speed up the verification compared to the traditional Hybrid Platform. This new methodology has been used in our IP development flow such that we were able to find many SW bugs between firmware and driver, or RTL bugs with real bench mark over the OS even before full SoC RTL integration was finished. This helped us avoid huge design time that could have been wasted for debugging. Future work will be to cover power analysis flow over android, to find out peak power and analysis and optimize.

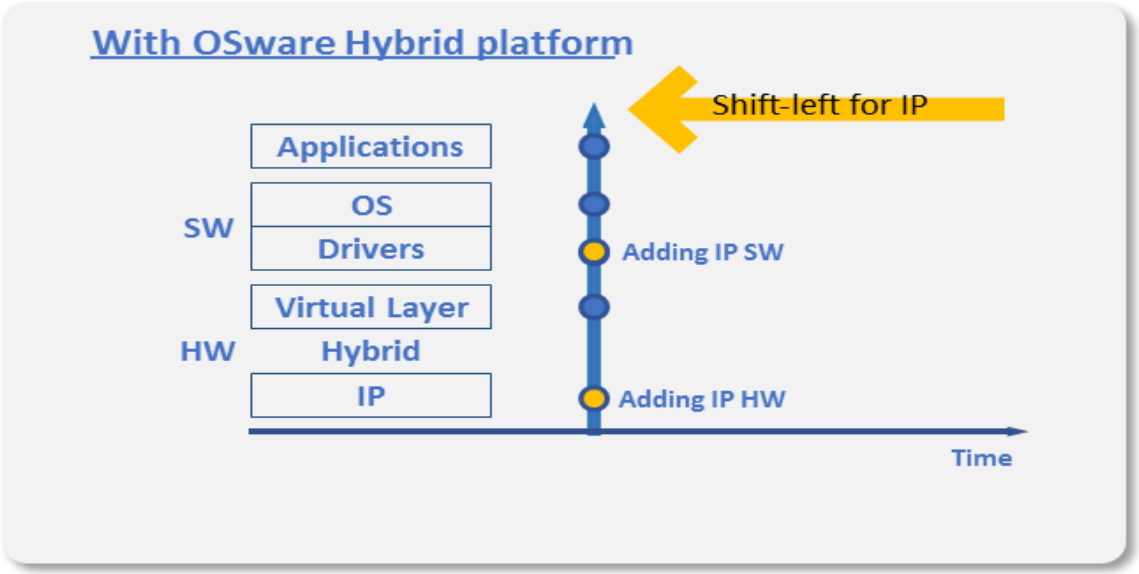

Figure 6 Proposed IP development flow.

References

[1] QEMU is a generic and open source machine emulator and virtualizer. More detail information is at https://www.qemu.org/

[2] Y. Nakamura, K. Hosokawa, I. Kuroda, K, Yoshikawa, and T. Yoshimura, “A Fast Hardware/Software Co-Verification Method for System-On-a-Chip by Using a C/C++ Simulator and FPGA Emulator with Shared Register Communication ,” Proc. DAC, pp. 299-304, San Diego, CA, USA, June 2004

[3] C.Y. Huang, Y.F. Yin, C.J. Hsu, T. Huang, and T.M. Chang, “SoC HW/SW Verification and Validation,” Proc. ASP-DAC, pp. 297-300, Yokohama, Japan, Jan. 2011.

[4] M. Vavouras, K. Papadimitriou, and I. Papaefstathiou, “High-Speed FPGA-Based Implementations of a Generic Algorithm,” Proc. ICSAMOS, pp. 9-16, Samos, Greece, July, 2009.

[5] Woojoo Kim, Haemin Park, Hyundon Kim, and Seonil Brian Choi, SukWon Kim “EARLY SOFTWARE DEVELOPMENT AND VERIFICATION METHODOLOGY USING HYBRID EMULATION PLATFORM,” Proc. of DVCon, San Jose, CA, 2017.

[6] PCIe is a Peripheral Component Interconnect Express. More detail information is at https://en.wikipedia.org/wiki/PCI_Express