Best Practices over Enhancing SoC Verification Efficiency

Bin Liu
Baseband SoC Development Group
Intel Inc.
Xi'an, China
bin.rocker.liu@intel.com

Haibo Shao
Baseband SoC Development Group
Intel Inc.
Xi'an, China
haibo.shao@intel.com

Hongbin Yu
Baseband SoC Development Group
Intel Inc.
Xi'an, China
hong-bin.yu@intel.com

***Abstract*- Modern SoC (System on Chip) design and integration concept focus on the chip's performance, power and size. Based on such a requirement, SoC design turns to select appropriate IPs (Intellectual Property), and integrate them together like a LEGO bricks project. The mainstream SoC process fastens the development procedure, but made SoC verification more complex. Convenient IP reuse does not equal verification reuse, but requires more hierarchical verification scenario coverage. Regarding the SoC matrix communication and growing IP number, the verification management faces more challenges and needs to balance schedule and outcome. Verification efficiency is an amplification factor to the verification outcome. In this paper, the series of applied practices will be introduced to illustrate the contribution to the verification efficiency.**

Keywords- Verification Efficiency; Size Reduction; Simulation Acceleration; System Debug Assistant

I. Introduction

From the revenue perspective, integrating reusable IPs into a single SoC will bring great value. Taking the baseband chip development as example, the sequential communication generation from 2G, 3G to 4G with different network mode requires a single SoC would include the corresponding subsystems. It would be estimated around by 2020, 5G baseband chip will be published. Some data could illustrate the transistor number from 2G, 3G to 4G is growing exponentially. Other persuasive demonstration could be achieved from the baseband chip verifiers experience based on previous projects. What people strong agree is about the chip size became larger, simulation speed became more critical and system test became more complex.

This paper will first explore the reason why the SoC size grows faster than verification ability. Then the paper will discuss the critical technology to complement the gap between verification and SoC size. For each technical topic, it will give several methods as implementation, and also project practice with feedback will be shared.

II. Background

Modern SoC project is composed of reusable IPs and new designs. In general, the more mature a SoC project is, the more reusable IPs it would have. The high reusability not only speed up the integration procedure, but also reduces the system risk. 'Small walk' strategy quite fits the baseband chip development which is in such a competitive era. The market is too fast, and the TTM (Time to Market) window is shrunk to narrower. Therefore, planned chip generations with competitive performance ability is arranged with pipeline projects. The adjacent generations' performance will have steady improvement but not dramatic change. In such context, the SoC should have high reuse rate to set-off the tight schedule and risk. Here the reusability becomes a key word to the chip development.

Comparing with verification reuse, design reuse is easier to handle. Some general way of the design reuse may cover:

- Parameterized design
- 3rd party business design IP
- In-house IP
- Design element and logic reuse
- System architecture reuse

With so extensive range of reusable design, the chip is increased unavoidably. Simultaneously, the size growth leads to another two side effects: simulation speed drop and system debug difficulty.

A. Chip Size Growth

Chip size growth brings new challenges in both design and verification. Design integration got more and more attention, and this is also true for the integration verification. Along with the component number growth, verification complexity embodies in how to validate the overall system works coherently. For instance, the communication network which bridges the masters and slaves is the first to bear the attack. It would be well understood the effort to verify a network (M nodes * N nodes) is in proportion to the product of M * N. As long as the peripheral number increases, the network effort would be asked more.

B. Simulation Speed Drop

Along with the chip size growth, verification got more affected by the simulation speed. Simply speaking, the simulation speed is close related with the chip size. The larger the size is, the slower the simulation will run. When a simulator loads the compiled object, it would first build up the system hierarchy and finish connection before run. This is the required static resource to establish a runtime environment. Once starting the simulation, the event generation and sequentially triggered events will require the dynamic resource. For a simple case, if a chip is not fed a valid clock, the simulation would be ran quickly because no event input and no new event is triggered. If a chip is fed the available clock, then the simulation speed would be related with the active region and the event trigger frequency. Therefore, simulation speed is affected by both the static resource and dynamic resource.

C. System Debug Difficulty

In module level verification, both design size and data path are still within the scope of understanding. With design specification and verification requirement, the verifiers would create exhaustive random test or formal method for coverage. Whether it is the overall design or states to be covered, the verifiers have more confidence in module level verification comparing with system level test. Actually, the verification hierarchy depends on the design integration depth. Some design would experience both module and chip level test stage, some design would be directly verified in chip level, and sometimes, a subsystem level verification would be inserted as an intermediate stage. When a chip level test failed, the verifiers would still apply the methods of module level and check the signal transition through the data path or check the internal signal flag bit. The method would not only consume time, but also easily makes verifiers to be lost in the system test. Existing simulator does not supply useful system monitor and debug utilities, and this paper would introduce several practical ways to open extensive system scope for verifiers.

III. Runtime Load Reduction

Based on the chip level verification environment release flow, the verification maintainer should prepare verification release after each time design release. During the preparation, the compilation time is also a point of interest. For the overall chip compilation, it would take several hours from scratch. Here are some compilation suggestion to shorten the overall time:

- Prepare each subsystem and compile them into independent libraries.
- VCS supports incremental compilation which allows partial compilation [1].
- QuestaSim supports library refresh and enable partial compilation [2].
- VCS support multiple sever compiling the same object in parallel.

With available compilation shorten means, simulation runtime load is the key factor to every verifier. As it is explained, the runtime load could be divided into two types: static load and dynamic load. Here are some practices to release the two load types:

- Static load release
    - Blackbox
    - Software model replacement.
    - Timing model
- Dynamic load release
    - Clock gating

- Reset hold
- Frequency multiplication

A. Blackbox

The blackbox way is a usual verification method, which is to configure the chip to instantiate some specific module as empty body. Fig.1 gives an easy example and shows some specific module could be instantiated as an empty blackbox. The more empty instances are involved in the chip, the smaller size the total simulation object could be.

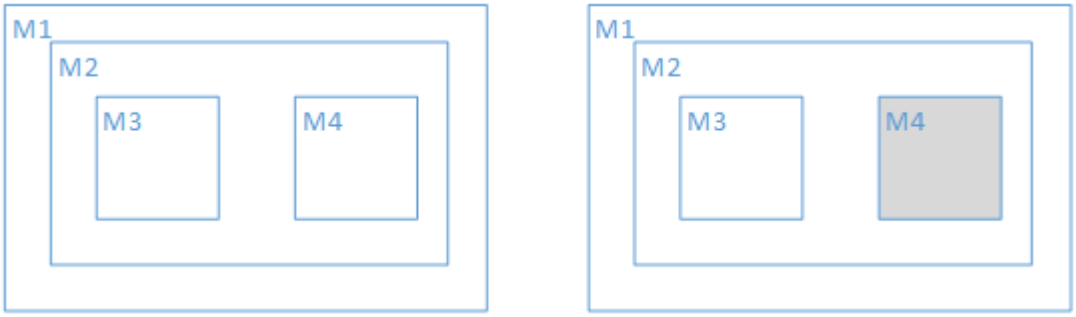

Figure1. Full chip instantiation and partial chip instantiation

In real project, different test scenario would involve different parts of chip to be organized. From the static memory load release view, if the chip configuration could be configured more flexibly, it would bring more benefit to the verifiers. The Fig.2 gives two blackbox configurations. Assuming M3 and M4 work independently and not interact with each other, then M3 and M4 verifiers could get better runtime experience with the two configurations below.

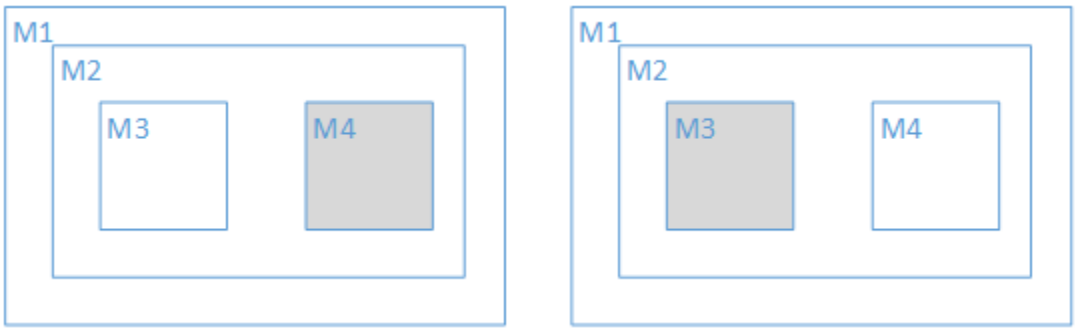

Figure2. Different blackbox configurations

It would be another topic to supply flexible configuration. Normally, the empty body and corresponding configuration need to be created manually. At the same time, the output boundary of those empty bodies needs to be tied with correct value. Here is the practice to generate the blackbox.

1. Apply internal integration tooling and different configuration input. Then the corresponding blackbox bodies could be created.
2. Run a real full chip simulation, and do a snapshot of target module output after hardware reset. The snapshot would be recorded with simple format, which is used to write back the snapshot value to the empty bodies.
3. Compile different chip configuration together as selective testbench configurations.

B. Software model replacement

In parallel with chip development, the software development is also asked to be pulled in. The purpose is to prepare the driver and firmware code as early as possible, and shift left the software development cycle. So far, there are several available ways serving this requirement:

- FPGA and emulator
- Software model

FPGA and emulator are also based on hardware maturity, and in general, the software development could be kicked off when the hardware RTL code is almost verified. This topic is independent of simulation runtime load, and we would focus more on the second way: software model.

The software model is generally prototyped with C or SystemC, which could represent abstract level behavior (C and SystemC) and also hardware level behavior (SystemC). Fig.3 explains the hardware design and software model development flow. Both of them are based on the design specification, but RTL would cover low level logic and software model would mainly cover high level behavior.

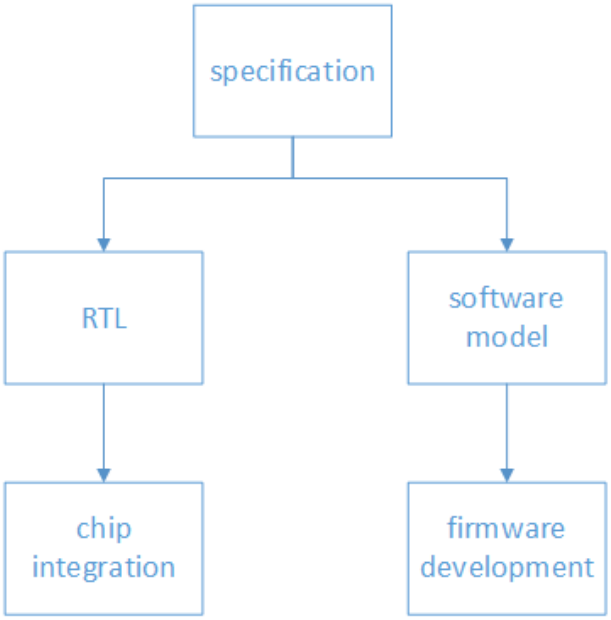

Figure3. RTL and software model development procedure

Here are the examples to illustrate how to replace the RTL model with C or SystemC model.

When integrating C model, SV DPI-C language interface would be helpful. From the Fig.4, the module M4 body could finish the interaction between HW (hardware) side and SW (software) side via DPI-C exported method:

- In HW side, the input signal of M4 would be combined to generate specific events, which are used to call the DPI-C function to call the SW side method.
- In SW side, it would also call the DPI-C task to invoke HW side.

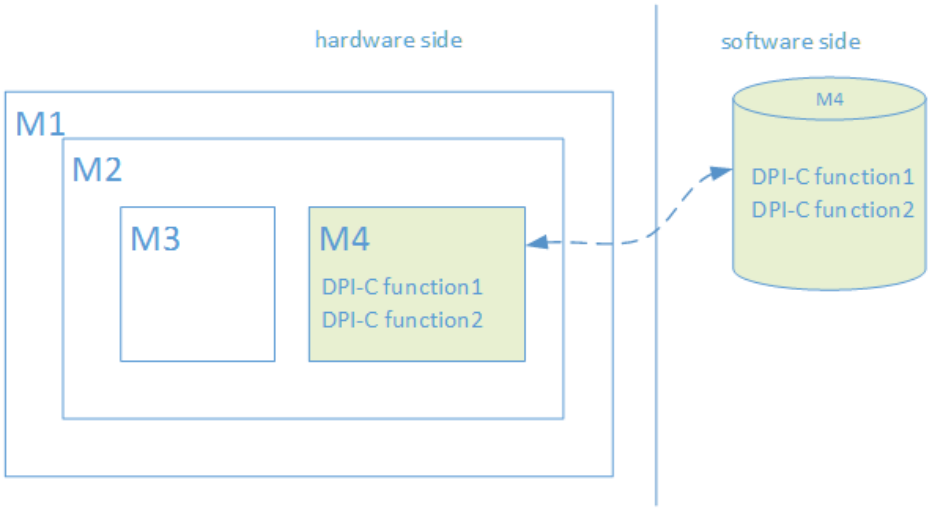

Figure4. Hardware replacement with C model

It is different integrating SystemC model from integrating C model. Since SystemC model could be directly established in the simulator, it does not need DPI-C interface. Here is the practice to show the SystemC model replacement from Fig.5:

- Instantiate SystemC model in M4 body.
- In most cases, the SystemC model would have TLM (Transaction Level Modeling) interface instead of pins. Therefore, the adapters which could translate the M4 input pin signal to TLM interface of SystemC model is necessary. The inversed path from SystemC model to M4 output pin also needs an adapter.

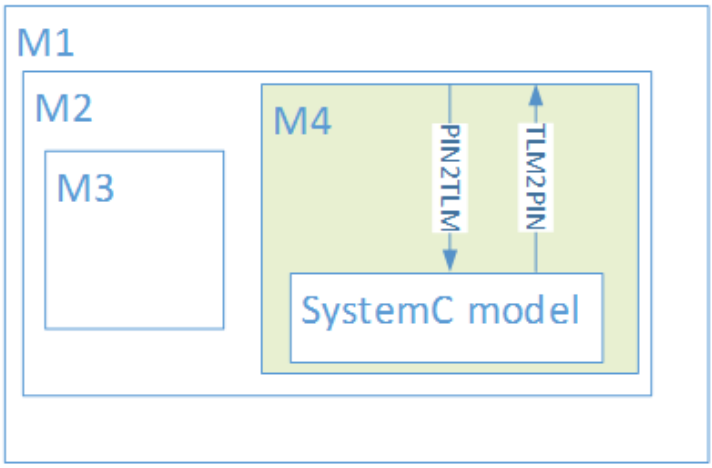

Figure5. Hardware replacement with SystemC model

The users for the SW model replacement solution should be noticed that the test scenario could not care much hardware details like timing and state machines due to the SW model behavior limitation.

C. Timing model

Timing model is extensively applied in gate level simulation stage. Sometimes, the IP is only delivered with a timing shell model instead of netlist with SDF (Standard Delay Format). The delivery's prerequisite is to ensure the IP's internal timing gets clean by backend process. For more check, the IP level testbench should be set up the gate level environment and to validate the internal timing. When a timing model is used instead of a netlist, it supplies those abilities:

- Input data port timing check based on the reference clock.
- Output data timing delay annotation based on the reference clock.
- The timing model is composed of a timing wrapper and the RTL model.

From the simulation speed point of view, the timing model would absolutely speed up the gate level simulation. Because netlist model is replaced with RTL model, and also fully hierarchical timing annotation is replaced with only boundary timing check and annotation.

However, the timing model creation faces several challenges, and the timing model precision is positively correlated with the IP's complexity. If the IP owns more boundary signals and more clock inputs, it would be more difficult to prepare a timing model. This conclusion also matches the practice when we tried to replace a larger subsystem netlist with a timing model.

D. Clock Gating

In runtime stage, the core idea to release simulator load is to decrease the total event frequency. For the simulation, the main contributors of event source are clock and combination logic. The clock gating technology is a common method to save power. For chip use case, the clock switch is frequently configured on and off. In chip level test, when the processor is waked up, it would execute clock initialization program to switch off unnecessary clocks. Then for each specific test, it is required to turn on the clocks along the test data path. This way not only fit the use case but also make the unnecessary scope to be inactive. The 'dead' scope would have infrequent activities, and contribute the runtime load release.

E. Reset hold

The reset hold idea is similar with clock gating. Although it is not really used to hold long time reset in use case, the reset hold could also make the scopes totally inactive. Both of registers and combination logic would be affected by the reset hold. The difference between clock gating and reset hold is that clock gating is more close to practical software usage.

F. Frequency multiplication

In several cases, it is needed to speed up some block execution, and get to a specific state as soon as possible. For instance, the processor subsystem would require initialization program which is a common context before all test scenario start. Then it could be set an over configured frequency to finish the initialization sooner.

The Fig6. gives a module M3 which is supposed to work in normal mode with 200Mhz. In practical usage, the module cannot run with higher frequency because the timing is not satisfied. However, RTL function simulation makes it possible to configure a higher frequency for M3. If it is acceptable to configure M3 clock frequency as 1GHz and no side effects (clock cross domain concern) is predicted, M3 could be configured to work with 1GHz clock, which would only fasten M3 initialization. Once the M3 finished the initialization stage, its clock could be configured back to 200Mhz and M3 could be ready for specific test. This solution shorten the simulator's average load along the M3 initialization, and make M3 as post silicon work frequency after the initialization. Therefore, the verification risk is low. This solution is limited in RTL verification phase, and cannot be taken in gate level simulation phase.

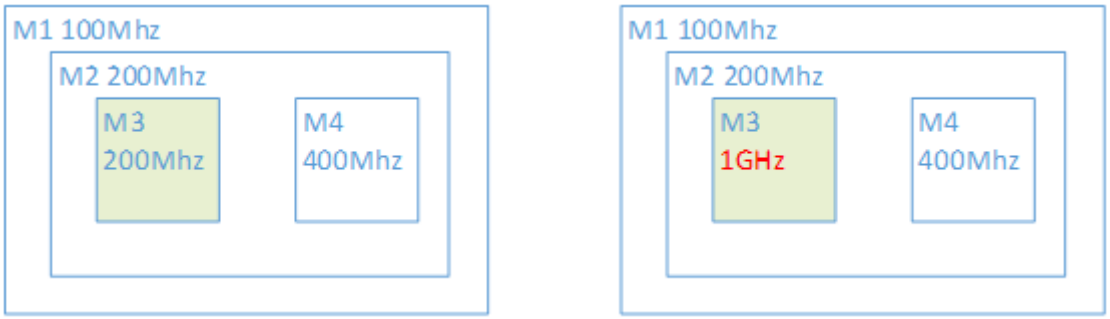

Figure6. Frequency multiplication of specific block

IV. Simulation Acceleration

Reducing simulation runtime load would intuitively solve the size problem. When the simulation load is decreased significantly, there are still some ways to accelerate the simulation. In chip level verification, a test scenario is mainly composed of several processors' program. Those distributed processors would execute the code independently or handshake with each other. Behind the processors' execution, it would configure specific block's registers and access memories. In general, a test program is composed of two phases:

- Common phase
- Specific phase

The common phase involves power up, hardware reset, and processor initialization. The most resource consuming part is the processor initialization. In this part, the processor would go through a long journey to a stable state of work. Here the simulation acceleration idea comes out from how to shorten or skip the processor initialization phase. This paper gives two methods:

- Virtual processor
- Golden state restoration

A. Virtual processor

From the name, virtual processor is a replacement of the real processor which plays main features of a processor:

- Register and memory access
- Idle state for estimated timing

- Interrupt handling
- Multiple processor parallel execution and handshake
- Reset for program re-execution
- Power sequence entry and exit

When the real processor executes the program, it would fetch the instruction and data from the memory, which has been dumped the binary code generated by compiler. This execution flow is illustrated by Fig.7.

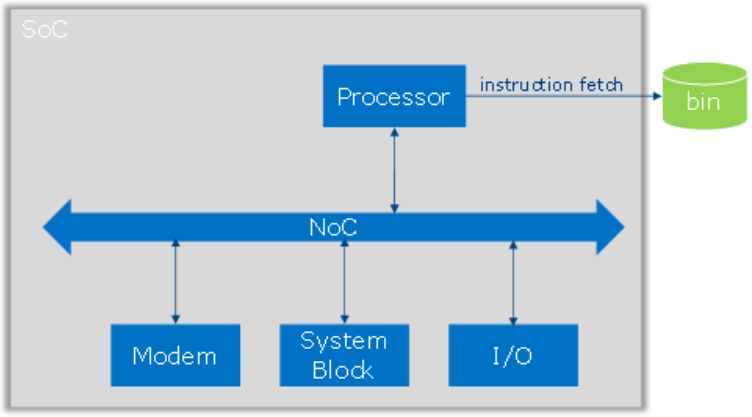

Figure7. Real processor instruction fetch and execution

When the processor is replaced with a virtual processor, the program execution is divided into two types:

- Workstation local execution
- Hardware interaction

From the Fig.8, the virtual processor is developed internally and given a name 'Wukong'. Wukong has been enabled all of main features listed above, and plays the intermediate role between the workstation and the simulator hardware world. The workstation local execution means the test C code part would be executed by the local server. For example, the algorithm computation will be finished by the server. When the C code recognized specific task for accessing register and memory, then it would call the DPI task via Wukong, and finally drive bus visit via hardware. Wukong not only responses the workstation local invoke, but also feedbacks when interrupts and reset are raised. For example, when an interrupt is asserted, Wukong would jump to the interrupt service routine, and jump back to the original program place once finishing the interrupt routine and the interrupt is deasserted.

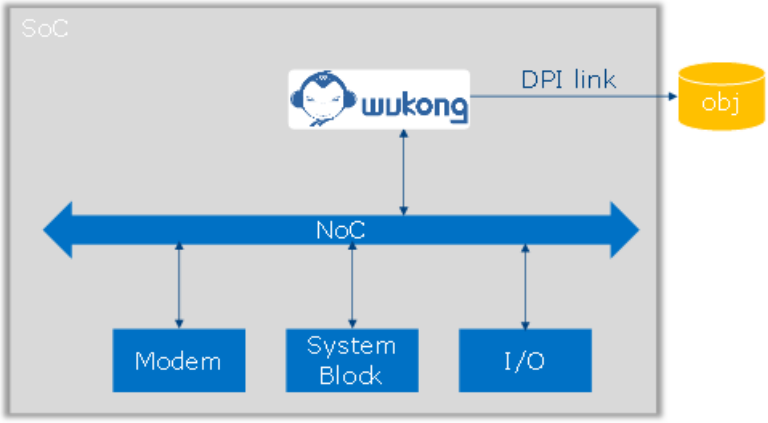

Figure8. Virtual processor execution by DPI interface

The assumption of virtual processor replacement is that the real processor has been verified on the IP level and chip integration. The obvious benefits brought by virtual processor include:

- Processor body is blackbox and much smaller.
- The processor initialization phase is skipped and this saves a lot of time.

- Supply more debug information and better user experience from the virtual processor.
- Verifiers would not feel any difference for test execution and result between real processor and virtual processor.

B. Golden state restoration

With virtual processor, it could cover over 80% test scenarios. The 20% test scenarios still requires the real processor. For instance, the real power sequence is so complex that it should take real processor to test the design implementation. Besides, the processor would be taken in power aware verification to check the isolation value and power domain switch. Those cases still get pain for simulation time with real processor due to the long initialization phase.

So far each big EDA vendor supplies the similar solution for technical support. The technology could be divided into two types:

- Checkpoint
- Save and restore

i. Checkpoint

In the same simulation session, the verifiers could record different simulation time points as checkpoints. The checkpoint records the precise full chip states, and convenient for state rewind. Once the state rewinds, the state would jump to the previously stored state, and run from the rewind time point. This feature is quite useful for test debug because it could shorten the issue time frame, and reduce the time cost of debug.

ii. Save and restore

Save and restore are one pair of operations. Save operation is to record all of current states as a solid database, and restore operation is to load the saved database and to jump to the saved time point. In general, the real processor would first go through the initialization program, and ready to jump to the fixed code section for test program. It is time to save the time point after the initialization and before the test code execution. The whole time frame before test program is a common program, and could be skipped for each test case. Then the saved time point is called golden state. The golden state is saved in a common place and could be accessed for all verifiers to restore directly to the golden time point. Once the test case restores to the golden state, it will skip the common phase and starts test program. For a complex chip system, the restore operation needs to be deeply customized to ensure the success:

- Once restoration finished, the compiled test program binary code which is different for each case should be loaded to the specific memory section to replace the saved test program.
- Update the test flow and make it could recognize the restoration operation. For example, environment setup and signal set should be avoided duplication.

V. System Debug Assistant

For a verifier, it is easy to get lost in a large chip system if the test requires lots of design for coordination. The module level debug means is not a best way to be adapted to chip level. If the verifier could be supplied with a higher vision, then it would give better chance to debug first at system level. To assist the system debug, several tools are developed:

- Online register debug tool
- Processor operation parser
- Network performance analyzer

A. Online register debug tool

It is different experience to debug the chip between pre-silicon and post-silicon. The pre-silicon verifiers expect the system debug tool of post-silicon testers, and post-silicon tests expect the internal signal vision of pre-silicon verifiers. From the pre-silicon chip verification efficiency, it is more effective to introduce the system vision referring to the tools of chip testers. An enviable system debug assistant is the tool Lauterbach [3] which connects the chip via JTAG (Joint Test Action Group) pins and would master all of internal registers' real time status value. Meanwhile, the tool could be also used to modify any accessible register value. This tool makes chip debug easier with system view instead of internal signals.

The in-house online debug tool RegCaP is based on the uniform format register file which is described by xml (Extensible Markup Language). It is well known the suggested register file format could refer to the IP-XACT [4], and different companies have their own register description format. With the uniform format, all of SoC register could be extracted to all possible kinds of header files or register model. For example, the C register access header file and UVM register model could be generated from the register description file. With this prerequisite, the full register map for RegCaP would be created with all of registers information e.g. address, and reset value. Normally to visit a register value, the verifier could write C code and ask the processor to execute and get register value or monitor the specific register signal value. However, neither of the ways cannot give a system view to verifier.

With the RegCaP mechanism shown below, it is available to get all of register value by the signal probe and event trigger. What's more, with the dynamic register value from RegCaP, the verifier could easily modify the hardware register value by the signal force. This way is a direct method and does not need to modify testcase or to restart the long test scenario. The register signal probe and force rely on the exact signal path, and the template generated register design by the register file gives the similar registers path except the register parent path name's difference. Therefore, this tool could give a system view and convenient register access.

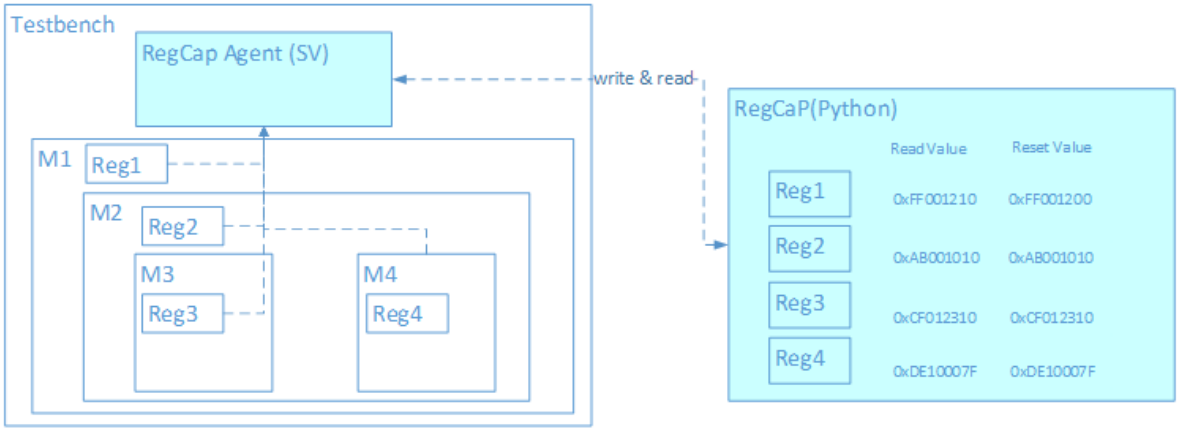

Figure9. RegCaP: an online register simulation debug tool

B. Processor operation parser

With the RegCaP register debug tool, it is convenient to check any register value and modify them. At the same time, it is better to monitor all of processors bus operation. In general, the bus operation is a register access or a memory access. If it could be given the dynamic processor operation abstract information, the verifier would get enough information from the simulation log instead of checking the bus by cycles. The processor operation parser is divided as two types:

- Processor vendor supplied official parser
- In-house developed processor operation parser

The official given parser by the processor IP vendor could be used to track the processor internal details like the processor's internal registers and instruction execution history. Instead of too much details, the verifiers will take first care of the accessed register name and value, and relate the each access with the test code. Therefore, it needs to map the bus operation to the register or memory. An in-house developed processor operation parser is shown in Fig.10, and it is implemented by a monitor and a chip register map. The monitor is a standard bus protocol analyzer to catch the bus operation type, address and data. The register map is created by the uniform register files which contains all of register information. Here it is suggested to only generate a plain text which involve the register's address instead of over thousands of UVM registers. This concern is a practical guide since the huge UVM register model objects require much dynamic memory allocation for instantiation and runtime resource.

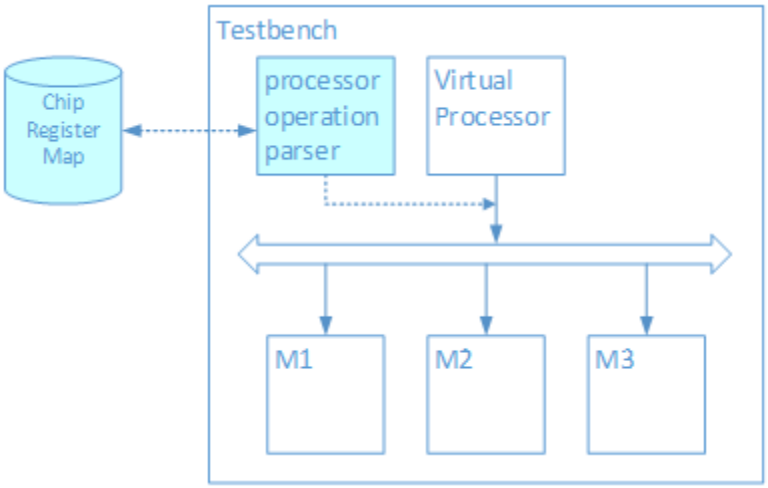

Figure10. In-house developed processor operation parser

With the processor operation parser and bind them to the distributed processors, the whole chip cores' pulse should be mastered easily from the log and visualized transaction record.

C. Network performance analyzer

The internetwork of the chip is another critical place to monitor. Besides the core's access, there are also other function blocks' access as maters or slaves e.g. DMA (Direct Memory Access). The idea for network performance analyzer is to monitor all of nodes of the network (masters and slaves) and record the bus access information. With the bus access frequency, it is feasible to calculate those data:

- The traffic data along a specific path from a master to a slave.
- The overall network performance by all distributed bus monitors.

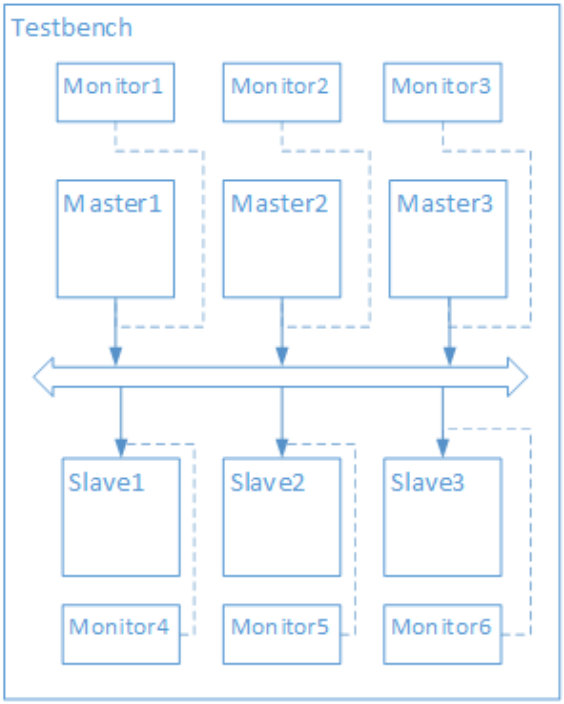

Figure11. Distributed network performance analyzers

By the network performance analyzer, it becomes doable to create pressure test scenarios and get the network performance on the fly. From the report, it could be achieved the pressure time frame, and also the bottleneck along the data path. The network performance analyzer makes it available to the pre-silicon performance evaluation.

VI. Summary

Since the chip verification architecture could be designed centrally and unified as an overall solution, the chip verification efficiency has the chance to get continual improvement. Along with the generations of serial chips, not only the chip hardware performance but also verification performance have been increased. It is compulsory to supply better verification efficiency regarding the rapid growth in chip size and complexity. It is great to have the performance improvement of simulator, computing server and verification methodology, and more favorable to develop customized efficiency solutions. The runtime load reduction, simulation acceleration and system debug assistant are three valuable directions deserved to long time contribution. From the feedback of executed chip projects, those given solutions indeed enhanced the verifier experience and efficiency.

ACKNOWLEDGMENT

We thank to all the people who contributed and responded to this paper. This paper would not be possible to be finished without their open experience sharing and opinions. As it is noted in the paper, the verification efficiency is based on historical contributions and summarized as an integral picture. We appreciate the verification efficiency contributors, Hua Zhang for the blackbox automation, Wenjia Yun and Shanmin Guo for the SystemC model replacement practice, consultant Fengguo Zuo for the clock multiplication, Chao Zhai for the customized golden state restoration implementation, Kun Wei for the RegCaP debug assistant prototype, Wenqiang Ren for the full chip register map build of processor operation parser, and Mao Li's team for the network performance analyzer initiative. Those valuable experience crossed over 3 years, and there are still new ideas submitted and under development to contribute the verification efficiency. We would also appreciate the open and relaxed environment which take the same importance between project execution and sustainable improvement, and here especially thank to the innovation group lead Fuzhen Yu and the head of department Lin Wang.

REFERENCE

[1] Synopsys VCS MX User Guide, June 2015.
[2] MentorGraphics Questa SIM User's Manual, 2014.
[3] Lauterbach website http://www.lauterbach.com/
[4] IP-XACT of Accellera webiste http://www.accellera.org/downloads/standards/ip-xact/
[5] ARM official website http://www.arm.com/