Hybrid Emulation: Accelerating Software driven verification and debug

Issac P Zacharia, Arm Ltd, Cambridge, UK (issac.zacharia@arm.com)

Jitendra Aggarwal, Arm Ltd, Bengaluru, India (jitendra.aggarwal@arm.com)

***Abstract*- Time to market is one of the key factors in the semiconductor industry, not only for commercial success but also to meet the technology demands of today's world. This can be achieved through a highly optimized verification strategy including early software development and testing. As the size and complexity of the designs increases, it is no longer possible to load the whole design into the emulator especially where OS boot and software applications need to be validated. This is where Hybrid Emulation fills the gap. Debugging hardware and software issues using real OS Apps on FPGA based Juno platform was challenging due to low design visibility. Hence, it takes significant amount of time to root cause the underlying issues. Hybrid system consists of a Virtual Platform connected to the SoC sub block (GPU) running in the emulator. It can run full software stack GPU graphics and compute content with full design visibility. Now it is easier to reproduce, debug and find the root cause of Software or Hardware issues reported in FPGA platform. Hybrid platform also enables faster design bring-up and verification closure.**

I. INTRODUCTION

Traditionally the Arm GPU hardware software verification was performed in an FPGA-based platform, which is a silicon level SoC board with an onboard FPGA where we physically map the GPU. Even though the runtime throughput is high for this FPGA platform, the lack of design visibility and the difficulty in extracting trace dumps affected the debug time. This motivated us to bring up an architecturally equivalent hardware emulation platform using Hybrid methodology. Hybrid emulation is a custom-FPGA or processor based technology based on different EDA partners which is different from FPGA prototyping. With this Hybrid Platform, where GPU running inside emulator, we can reach a full design visibility which was lacking in FPGA platforms. Since RTL CPU is replaced by C-based models (Arm fastmodels or Virtual CPU models), we get faster boot time. Since software stack runs in CPU main memory, it makes the entire execution of any application faster while maintaining the full accuracy of the rest of the SoC. It can greatly improve simulation performance, allowing us to boot an operating system and execute functionality on an accurate model of the SoC in a significantly shorter time. This enables the development and verification of drivers, kernels, applications, validation with the firmware of the SoC way before silicon tape-out/GDS2.

II. MOTIVATION TOWARDS HYBRID PLATFORM

The Hybrid Platform bring-up was motivated by the limitations of the existing FPGA platform.
The major challenges we had to address were:

1) Prolonged debug time due to poor design visibility and the difficulty in getting the traces.
2) Complex real OS App issues could take weeks to generate the stimulus and reproduce at unit level simulation platforms.
3) GPU Hardware and Software live debug is challenging in the existing platform.
4) Issues reported in different verification platforms and tools are typically not cross reproducible.

In addition to that, the Hybrid platform became the best way to shift left the hardware software co-verification, allowing us to perform software development and verification even when the RTL of a critical block isn't available. It gives us much easier platform upgradability, when there is a need of changing the configuration or moving to a latest CPU subsystem it is much more convenient and quicker to execute the software workloads rather than waiting for the hardware board to be ready.
Figure 1 represents the Internal FPGA platform where the CPU subsystem and other SoC peripherals are in silicon.

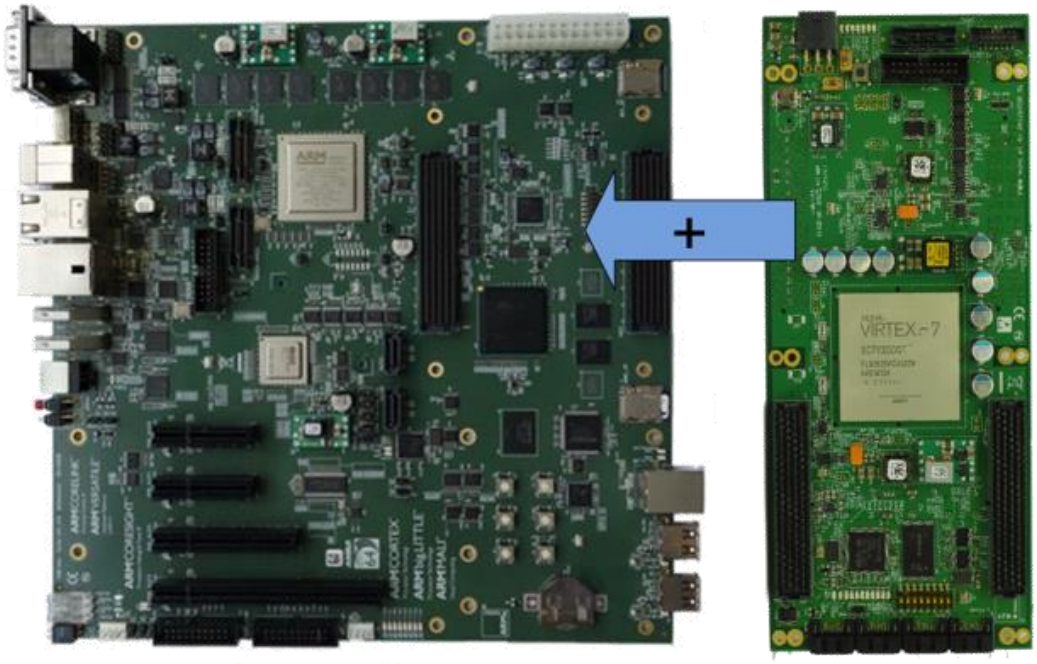

Figure 1. FPGA based Juno Platform

III. HYBRID PLATFORM INTEGRATION

On the Hybrid Platform, we replaced the Arm Cortex A55 CPU RTL core with its corresponding fastmodel along with the GIC600 and the smart memory. Replacing this RTL block with a fastmodel not only helps achieving runtime throughput but it also reduces the hardware footprint. The RTL running on the emulator will be connected to this Virtual platform through TLM (transaction level model) bridges which provides a communication gateway between virtual and RTL. This transactor bridge provides a thin layer of BFM + Proxy models where we can connect the RTL ports to BFM and Software transactions to Proxy side.

We also use a shared memory for CPU and rest of the system implemented specifically for hybrid usage and we usually term it as “Smart memory”. Smart memory will be visible to virtual as well as RTL side, and it allows CPU to access the memory through backdoor (not via TLM channel). With this implementation, the CPU subsystem can access the memory within the virtual system without crossing the RTL boundary. Similarly, the GPU on the Emulator will access the memory through the RTL channel. Therefore, the software execution will be faster as there is very minimal cross boundary communication required. At some point, if there is a sync required between fastmodel and emulator, it is done using clock pause of emulator to pass the architectural state of CPU and cache memory data sync with rest of system. Overall, if there are frequent transactions done between Virtual Platform and Emulator, it will degrade the overall runtime performance.
The Figure 2 shows the Virtual Platform (VP) on left side connected to the RTL Platform on right side. The virtual CPU, GIC600, Smart Memory and other IPs are connected to the virtual router bridge. The other end of the router is connected to the TLM transaction pipes coming from the RTL platform. The router on the virtual side is an EDA vendor specific interconnect, which can be configured to define the number of masters and slaves, width, address space etc. of the virtual components. The GIC600 have been moved to the virtual platform, leaving GIC BFM on RTL side to have port level connection with NIC. The virtual router is connected to the CCI550 on RTL through which the virtual CPU access rest of the peripherals on the RTL. Smart memory is intended to have a localized memory for CPU within the virtual platform.

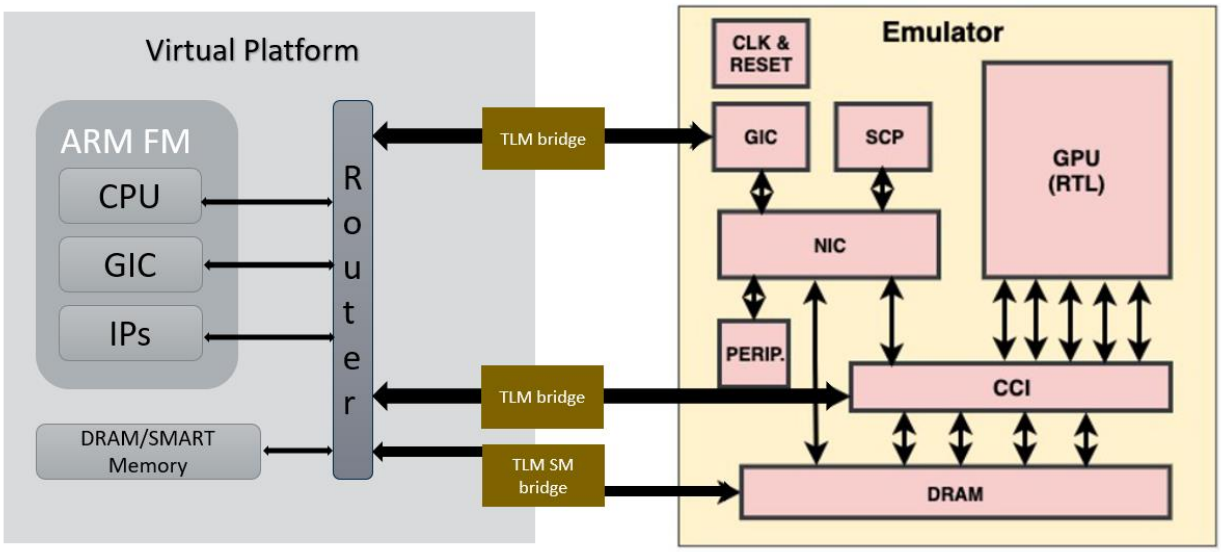

Figure 2. Hybrid platform block diagram

There are built-in models available by EDA Hybrid toolchain to take care of RTL to virtual change in form of transactions to/from CPU. Internally, these models use SCEMI-DPI to transfer data back and forth on TLM channel across boundary.

To validate the integration of above system, we booted the Linux and ran multiple baremetal tests to access the GPU registers and smart memory. Later we booted Android 10, Debian Buster 11, as well as different graphic tests. The Hybrid methodology was especially efficient for high level software use cases such as booting Linux, Android, Linux drivers and different tests that run on top of these operating systems. Also, it is recommended to maximize the transaction size and minimize the use of high traffic transactors to achieve best runtime performance.

IV. Runtime Performance and Results

The Hybrid platform helped reproducing the hardware and software issues captured in FPGA platform by rerunning the same test with this functionally accurate Hybrid model with full design visibility. This significant reduction in debug time helped fixing the software and hardware bugs much earlier in the verification cycle compared to previous projects.

TABLE I
HYBRID RUNTIME THROUGHPUT

Test	Runtime
Linux Boot	62 sec
Android 10	2640 sec
Debian Buster 11	185 sec

The Figure 3 shows the graphics test running on hybrid platform after the Android boot. On a Linux boot benchmark, moving from the legacy full RTL emulation to the Hybrid platform brought us a 45X improvement in runtime.

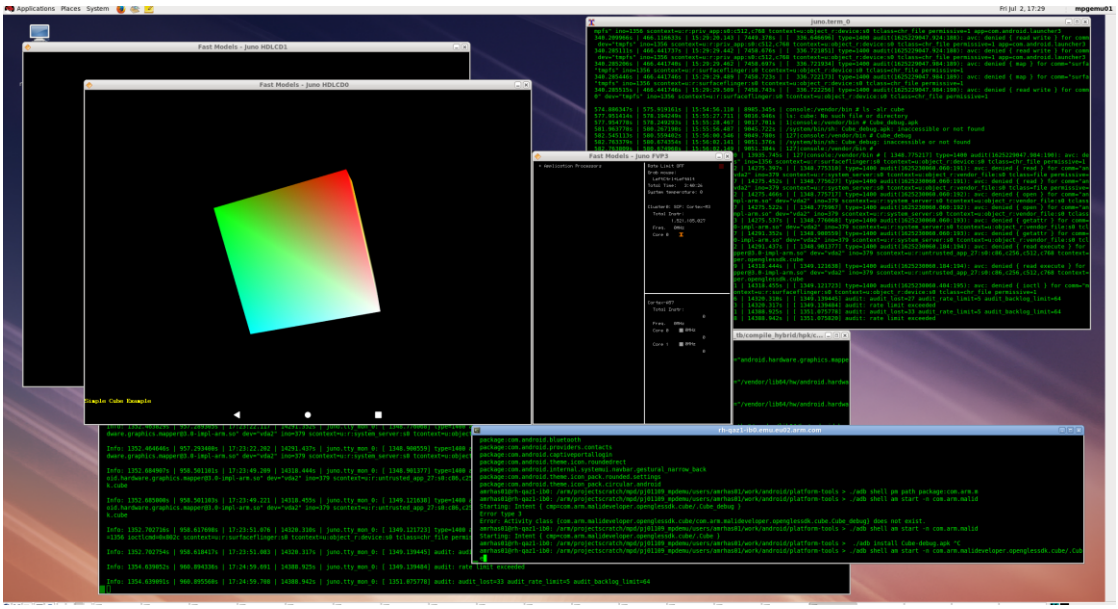

Figure 3. Graphic test running on Android 10

V. DEBUG CAPABILITIES

EDA deliverables around Hybrid/Virtual Components of Arm CPUs provide additional utilities to debug CPUs such as Detailed Logging, TLM (Transaction level model) visualization, Register viewer of CPU, Memory map viewer of CPU, Smart memory debug tools etc.

RTL CPU consumes clock cycles to execute its state machines along with rest of the subsystem whereas Hybrid CPU is running in asynchronous manner in 0 simulation time. In this case, Virtual CPU can be of non-deterministic behavior with rest of sub-system in RTL to debug and replicate any issues in consecutive runs. To overcome this, the Hybrid Capture Replay feature enables more efficient debug by recording stimuli along with its execution and replaying it in consecutive runs/debugs. The Hybrid Capture Replay feature captures or records the inputs from the Virtual platform against the design clock and enables you to replay those inputs to the design in a subsequent session.

Apart from CPU, debug of rest of sub-system is done using native methods such as RTL monitors, waveforms, memory dumps, and so on.

VI. NEXT STEPS

We observed a significant gain using Hybrid usage in Emulator which led us to explore and deploy it on bigger eco-systems of Arm notably on the latest family of released CPUs and some unreleased ones. Arm has a variety of use-cases where it is essential for the CPU to be cycle-accurate, but Hybrid caters an effective co-ownership of FPGA based challenges in Software development, therefore it is certainly not a complete replacement of our validation mind-set. We also have plans to help using this work in the infrastructure space (Server, Enterprise) where complete system visualization on a real emulated hardware is gated by maximum emulator size.

REFERENCES

[1] Developer. Arm tools-and-software/development-boards/Juno-development-board