Architectures to Tradeoff Performance vs. Debug for Software Development on Emulation Platform

Loganath Ramachandran, Ph.D. Verikwest Systems Inc, USA

Ragavender Swamisai, Belmont Computing, USA

Prof Makoto Ikeda, Ph.D, University of Tokyo, Japan

***Abstract*—In pursuit of facilitating early software development, it has always been necessary to speed up simulation and emulation platforms. Currently, emulation platforms can reach peak performances of several Mhz. However, when debugging is required, the speed drops to 10-100 kHz. In this paper, we propose switchable architectures to enable performance and debugging ends of the software development spectrum.**

Keywords—emulation; verification; debug; performance

I. INTRODUCTION

Development of software is an essential ingredient of any SOC project. This software effort is required for many reasons. Some of them include (a) developing the boot code for the SOC (b) establishing the critical security parameters of the SOC (c) low level drivers for each of the peripheral components and (d) software applications that interact with the hardware. For example, a video processing software running on a SOC may require interacting with a video processing hardware unit.

In the past, software development for the SOC would begin after the chip that has been fabricated, as the silicon can run at gigahertz speed. With current time-to-market considerations this approach does not work, as it may take several months (or even years) to complete the software development. In addition, any hardware change is extremely costly at this stage. Hence it is essential to start software development early in the SOC development cycle.

Early software development ensures that the hardware and software work seamlessly with each other. Any RTL bugs exposed during software development can be easily fixed by changing the RTL. Using this shift-left strategy results in software being ready and available when the chip samples come back from the fab. But there is one main question that needs to be answered, "*What is the right platform to use for such early software development?*". There are several possibilities, each with its own pros and cons. These include:

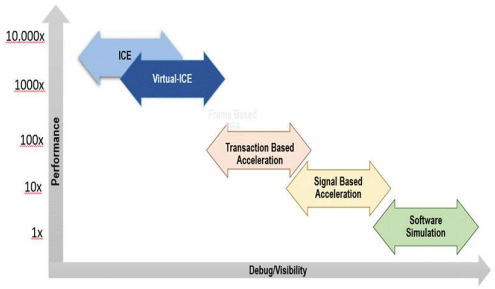
(a) ***Simulation platforms***: As the RTL, testbench and the tests are already running on a simulator, it appears to be an easy choice to use this platform for software development. One can load the entire software stack in RAM using the $readmemh() and start the simulation. This platform is cycle-accurate and offers full visibility to each signal. Complete debug of both hardware and software is feasible. However, the main drawback is performance. Software teams are not going to be excited compiling and running their software on a platform that runs at kilohertz speeds.

(b) ***Instruction Set Simulators***: This is typically a simulation model of processor. For example, Spike [7] is a RISC-V functional ISA simulator. It models the instruction set architecture of RISC-V core and cache system. The ISS model mimics the behavior of the processor by reading object code from a file and maintaining the values of internal registers. The ISS models are instruction-accurate (but not cycle accurate). They output a trace of all the instructions as it is being executed. The performance of this platform is very high as we are executing a fast model of the processor, but it does not have any knowledge of the rest of the SOC.

(c) ***Virtual Prototyping platform***: With these platforms one can model the entire SOC including all the peripheral components. Software development can start on this model, even before the RTL is complete. These models

tradeoff cycle accuracy for higher level performance. Although software developers have a high-performance platform, developing and maintaining this platform is labor intensive. Moreover, the bugs in the RTL may not be exposed by these platforms as the RTL is not used in these models.

(d) ***Emulation platforms***. These platforms allow users to synthesize the DUT and map them to internal hardware components (e.g., FPGA, or special purpose processors) that reside on the emulation boards. The testbench is typically not synthesizable and runs in the simulator on a Linux workstation (sometimes referred to as co-modeling host). The performance of the emulation platforms which can be 1000X better than simulator performance depends on the interaction between the testbench (running on a Linux workstation) and the DUT (running on the emulator). The lesser the interaction the higher the performance gain. This is shown in the following figure:



- In *signal-based acceleration*, the interaction (between DUT and testbench) is purely signal based, and each signal transition is communicated between the two domains. Performance gain is limited by the constant synchronization required between the simulator and emulator.
- *In transaction-based acceleration*, the communication between the testbench and the DUT is based on transactions (which are typically larger chunks of data). This method further improves emulation performance as the frequency of syncs between the testbench and emulator decreases, as compared to signal-based acceleration.
- *In-circuit emulation* mode provides the highest performance on the emulator because the entire testbench and DUT are running on the emulator. To monitor the signal activity from the DUT, hardware peripherals (e.g., Speedbridge) are sometimes attached to the emulator.

(e) Hybrid emulation platforms: Here [8] the RTL of the CPU is replaced by C-based models (Arm fastmodels), but the reset of the SOC is synthesized onto the emulator. This approach improves boot performance. The debuggability of the SOC is still good because the rest of the RTL is synthesized on the emulator. Due to the interaction required between hardware and the software (i.e., fastmodel of the processor), the performance will not be equal to the ICE mode performance.

When using the emulation platform, we use the RTL (mostly as is). There is no need to develop a separate model of the hardware for the purpose of verification or software development. The original RTL is directly synthesized onto the emulator. Any potential bugs in the original RTL will be exposed during emulation.

However, the interaction and sync with the simulator and emulator has a significant impact on overall performance. Our goal was to come up with a solution, that can be obtain the best possible performance on the emulator, while retaining the ability to run debug when necessary.

II. PERFORMANCE AND DEBUGGABILITY REQUIREMENTS

As shown above, the verification performance varies several orders of magnitude, depending on the architecture chosen for emulation. A higher verification performance (such as ICE mode) also implies lower debuggability. While ICE mode is useful for software development, the lack of debuggability impedes the chances of debugging the software running on the processor. Also, a pure ICE mode requires Speedbridge adapters [9] for peripherals, restricting the scope of the usage.

An essential *requirement for debugging* a SOC running on the emulator, is that the DUT (running on the emulator) and the rest of the testbench (running on the simulator) be driven in a lock-step fashion synchronously by a single clock. Effective debugging also requires the use of certain components such as (a) tracer, a component that prints out the details of each retired instruction, and (b) debugger, a component that enables us to set breakpoints on the

software and examine the values. The tracer and debugger must be in sync with the rest of the RTL running on the emulator.

On the other hand, an essential *requirement for high performance* is that the DUT (running on the emulator) executes independently of the testbench (running on the simulator) and synchronizes as infrequently as possible. Such a synchronization method will ensure that the hardware can perform at the maximum allowable speed without stopping and syncing with the simulator. Components such as the tracer and debugger are not required when running the SOC in high performance mode.

It is also essential that the solution is well designed to be easily "switchable" between a performance mode and debug mode. The testbench architecture and the tests need to easily switchable between performance mode and debug mode of the emulator.

III. OUR SOLUTION

We propose a switchable architecture solution to manage the contradictory requirements of debugging and performance. The proposed architectures are.

(A) **Debug-focused** architecture and

(B) **Performance-focused** architecture.

We realize that building and maintaining two separate architectures for emulation would be an expensive proposition. Hence, we designed the architectures to share as much code as possible. We ensure that the RTL used in both the architectures are same. This provides the following benefits.

(a) A bug discovered in the RTL in one mode is easily reproducible with the other mode.

(b) Synthesis effort is shared across both the modes, as the DUT is the same. However, in the performance-focused architecture there are additional components instantiated in the emulator.

Our solution also uses virtual peripherals. We built virtual peripherals for GPIO, UART and other required external interfaces. These virtual peripherals receive data from the SOC using the communication channel between the emulator and the simulator. The received data is processed and either printed on the terminal or stored in a file. This makes it easy to understand the cause of failures. The use virtual peripherals impact the performance, but it enables broader usage of the emulator.

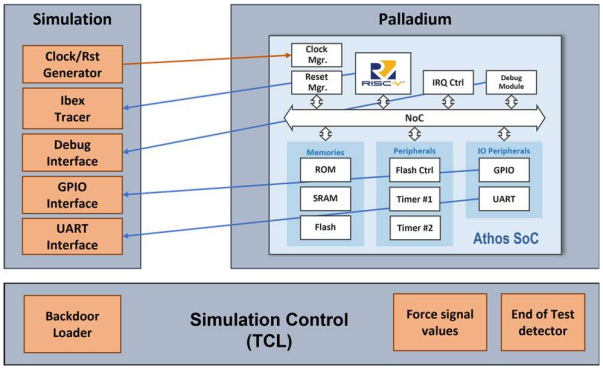
Although our solution is generic and can be implemented in any of the commercial emulators, we have used the Palladium emulator to implement our solution. This implies that some of the internal features offered by Palladium have been used in our implementation. For example, the **ixclkgen** has been used to generate the clock generator component on the emulator.

Debug Focused Architecture (DFA)

The debug-focused architecture is shown in the figure below and contains three parts.

1. The main design (SOC) is synthesized and runs on emulation hardware.
2. A simulator running on the Linux server contains the following.
    a. Virtual peripheral components
    b. Debugger interface component, that interfaces to gdb
    c. Processor (Ibex) tracer component uses the RVFI interface on Ibex. Signals are transferred to the simulator using the Signal Based Acceleration. The tracer receives the retired instructions and prints them to the tracer.log file.
    d. Clock and reset generators are included in the simulation side and values are synchronized with the Palladium hardware

3. The Simulation control engine serves the following purpose:
    a. loads the boot code
    b. forcing of signals if needed
    c. end of test detection mechanism to detect the test completion and exiting the emulation engine.



DBA implementation is characterized by the following:

(a) ***Software debugger interface*** to debug the binary code executing on the SOC. Such a debug interface will enable software developers to set breakpoints and examine the values of the variables and registers.

(b) ***Instruction tracer*** to trace every instruction being retired on the CPU. Tracer capability will give a real-time picture of the instructions as they are fetched and processed.

```
554500          525  000080f0   08458593   addi       x11,x11,132  x11:0x00008000 x11=0x00008084
555500          526  000080f4   00052283   lw         x5,0(x10)    x10:0x00008080  x5=0x0040006f PA:0x00008080 store:0x00000000
556500          527  000080f8   0509       c.addi     x10,2        x10:0x00008080 x10=0x00008082
557500          528  000080fa   feb56de3   bltu       x10,x11,80f4            x10:0x00008082 x11:0x00008084  x0=0x00000000
561500          532  000080f4   00052283   lw         x5,0(x10)    x10:0x00008082  x5=0x40810040 PA:0x00008082 store:0x00000000
562500          533  000080f8   0509       c.addi     x10,2        x10:0x00008082 x10=0x00008084
563500          534  000080fa   feb56de3   bltu       x10,x11,80f4            x10:0x00008084 x11:0x00008084  x0=0x00000000
564500          535  000080fe   80818513   addi       x10,x3,-2040             x3:0x10000800 x10=0x10000008
565500          536  00008102   81018593   addi       x11,x3,-2032             x3:0x10000800 x11=0x10000010
```

(c) ***Model all the peripherals*** that can process all the signals sent out of the SOC. For example, the output from the UART port is decoded by a peripheral (real or virtual) and transferred the data to an output device.

(d) ***Waveform dumps*** may be necessary for post-simulation debug activities using tools like SimVision. Emulators typically have powerful features for managing and viewing waveforms.

Performance-Focused Architecture (PFA)

The performance-focused architecture contains three parts.

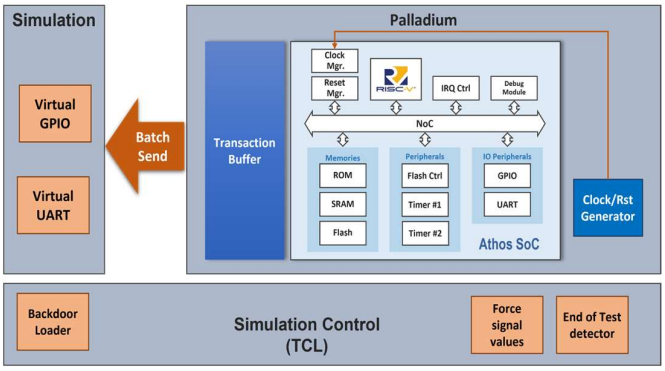
1. The synthesized design running on the emulator includes:
    a. The main design (SOC)
    b. Clock and Reset generation circuitry for the SOC. The native clock generation capabilities of the emulation platform are used to specify the clock features. A "qel" file is used to specify the clock parameters, and an internal tool is used to generate the clock component. The generated clock component uses the native "CAKE" API.

```
$ less clkgen.qel
clockSource -add clk_i
clockOption -add {technology CAKE 1}
clockFrequency -add {clk_i 1Mhz}
```

    c. Transaction buffer. To keep the emulator and the simulator running asynchronously, it is necessary to capture all the output from the I/O components of the SOC. In this case, we have built a transaction buffer that captures all the outputs from the UART and GPIO. This transaction buffer is retrieved periodically by the simulator and the data stored in the logs.

2. The testbench running on the simulator only contains the virtual GPIO and virtual UART components. Here these components are driven by an independent clock that is not synchronized with the emulation clock.

3. The Simulation control engine serves the following purpose:
    a. loads the boot code
    b. forcing of signals if needed
    c. end of test detection mechanism to detect the test completion and exiting the emulation engine.

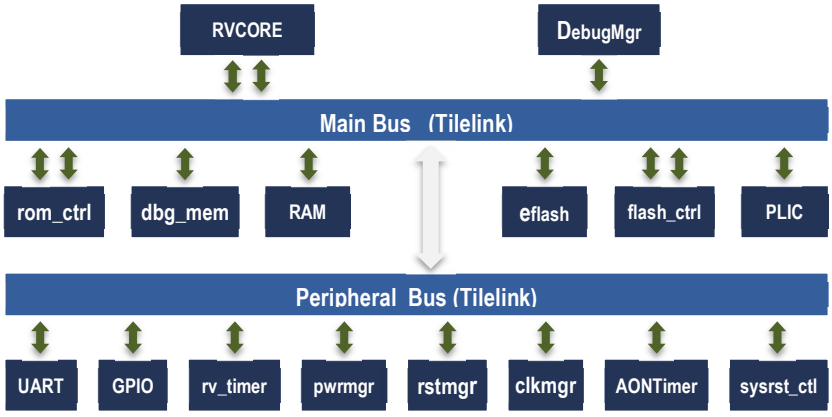


In summary the main differences between these architectures are highlighted below:

Debug Focused Implementation (DFA)	Performance-Focused Implementation (PFA)
Cycle accurate sync between TB and design	Transaction accurate sync between TB and design
Single common clock between TB and design	Two separate clocks driving TB and the design
Common software-based drivers and monitors	Hardware transactors
Performance in Khz range	Performance in Mhz range
Tracer, Debug support	No tracer, debug required as we are not debugging.
UART, GPIO agents support interactive debugging	Virtual UART, GPIO components support post simulation debugging

IV. RESULTS

We developed PFA and DFA for a RISC-V-based SOC platform (internal codename Athos) based on the Ibex [2] core. This internal SOC was designed in SystemVerilog [3] with open source lowRisc components. The SOC contains two busses (a) the main bus and (b) the Peripheral bus. These interconnect structures are based on the Tilelink[4] protocol. The SOC also contains a couple of peripherals (UART and GPIO) that communicate with the external world. The entire SOC is designed for expandability. We are using this SOC as the base building for other projects.



The software stack for the SOC contains many layers, including (a) Hardware abstraction layer, (b) Run time library (c) Boot code for the SOC, (d) Test Harness code and (e) User code (applications). Software running on the CPU controls the initialization, reading, and writing of the values onto these peripherals. All the layers of the software are written in "C". Our goal was to run all the software in both the modes.

We implemented the PFA and DFA architectures for the Athos SOC, on the Palladium [6] emulator. The implementation highlights:

- We connected the tracer through the RVFI interface available on the Ibex for DFA.
- DFA was debug enabled through the OpenOCD interface.
- Peripherals were modeled as virtual I/O blocks.
- Compilation of both hardware and software was done with ProtoForge, a proprietary build engine that manages both hardware and software builds.
- RISC-V toolchain [7] was used to compile the boot code, the libraries, and the tests run on the SOC.

We compiled the coremark benchmark testcase for the RISC-V processor and ran it on both the DFA and PFA using Palladium. We measured the number of tbsyncs for the benchmark. As expected, the number of tbsyncs was 300% higher for DFA.

The results show close to a 100x difference in performance between DFA and PFA for the coremark test case. The emulator usage percentage increases significantly in PFA mode as the emulator is running with its own clock and minimizing the syncs with the testbench. These architectures can be switched easily depending on the use. For nightly regressions the PFA mode would be ideal, while for debugging failures the DFA mode would be ideal.

Architecture	DUT speed	#Tbsyncs	HW Busy %
DFA	26.92 kHz	900700	1.04%
PFA (with dumping enabled)	2236.76 kHz	29	86.82

V. REFERENCES

[1] ETH Zurich, University of Bologna, Ibex Core Documentation, https://ibex-core.readthedocs.io/en/latest/01_overview/index.html, ETH Zurich and University of Bologna

[2] SiFive, Tilelink Specification

[3] RISC-V External Debug Support, specifications

[4] OpenOCD, https://openocd.org"

[5] RISC-V Toolchain, "https://github.com/riscv-collab/riscv-gnu-toolchain

[6] Cadence, Palladium Emulation in http://www.cadence.com

[7] Spike, riscv-software-src/riscv-isa-sim: Spike, a RISC-V ISA Simulator (github.com)

[8] Isaac Zacharia, Jiterndra Aggarwal, "Hybrid Emulation: Accelerating Software driven verification and debug", DVCON India 2021