SYNOPSYS®

WHITE PAPER

Accelerating Software Development with Fast Virtual Prototypes

Author

Sam Tennent
R&D Engineer, Sr. Staff,
Synopsys

Introduction

Most of today's largest semiconductor devices are highly complex system on chip (SoC) designs, which means that they include one or more embedded processors. This indicates that software provides some of the key functionality of the chip. The system cannot be fully verified or validated without both hardware and software. However, software development generally takes more time and resources to develop than hardware. There is no way to meet time-to-market (TTM) requirements if software development cannot begin until the hardware design is complete. SoC projects must embrace earlier and faster development of software so that it can be co-verified with the final hardware design quickly as the final step before tape-out. This paper discusses the challenges and presents solutions using fast virtual prototypes.

Software Development Challenges

With the growth in SoC development, the software content in chips and end products is exploding, as shown in Figure 1. The demanding performance requirements for this software is driving hardware complexity, with heterogeneous multicore processor architectures and virtualization now in wide use. In return, parallelism and complex hardware make the software harder to develop, so there is a loop that puts constant pressure on programmers. Simply adding more people to the team is not the answer; everyone's individual productivity must improve so that the overall software effort starts and finishes much earlier in the project. This requires innovation and new approaches in tools and methodologies that enable diverse distributed teams to develop and test code efficiently as early in the project as possible.

There are several approaches used in the past that no longer suffice. At one time, software teams waited until fabricated chips were available in the bring-up lab to do the bulk of their development. This is much too late in the schedule to meet TTM goals, even if no chip respins are needed. Software can be run with the RTL (register transfer level) hardware on simulation processor models, but these are too slow for efficient software development and test. Faster RTL-based methods such as hardware emulation and FPGA prototyping are available late in the development flow and cannot be scaled up for continuous testing of large regression test runs. One alternative is developing software in parallel with the hardware and running it on simulation processor models.

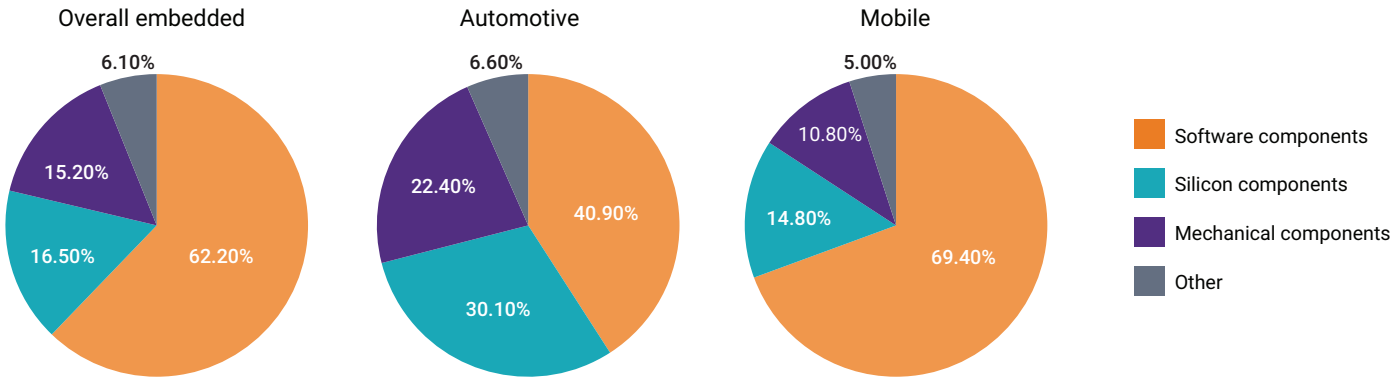

Figure 1: Growing software content in SoC applications

Fast Virtual Prototyping

The key to virtual prototypes is that they are built using abstract hardware models, so they can be developed before, or in parallel with, RTL coding. Because they are available early in the project, they enable a true "shift left" for software development. With parallel hardware and software efforts, system validation is also shifted left, enabling a much earlier tape-out and dramatically shorter TTM. Virtual prototypes can be replicated very easily, so they can be deployed across distributed programming teams and foster collaboration. They run on standard hosts such as personal computers, so cost of deployment is low and there is no specialized hardware to build or maintain.

Virtual prototypes are fast, with execution performance rivalling that of hardware platforms. Users can develop and run applications (apps), operating systems and hardware dependent software on fast instruction set simulators (ISSs) for the embedded processors. The abstract hardware models include all architecturally visible registers and other features so that virtual prototypes can run complex software stacks with no special modifications. This is important because programmers can write and test the same production code that they will later run on FPGA prototypes and actual chips in the bring-up lab. Development of the abstract models is much faster than the full RTL implementation because all details not affecting software can be ignored.

Maximum execution speed is clearly a key requirement for this approach, but the best virtual prototyping solutions have additional valuable features. They should be able to interface with third party tools such as processor-specific software debuggers. They should also be able to interface to the real-world environment, for example to capture realistic bus traffic on an SoC interface. When hardware dependent software such as device drivers is not under test, users should be able to use device virtualization, such as VirtIO, to replace register accurate models. This provides a performance boost when developing apps and other higher-level software. Finally, the virtual prototypes must support agile code flows, including continuous integration (CI) and a path to continuous deployment (CD).

Synopsys Virtualizer Studio

All the requirements and features listed above—and more—are provided by Synopsys Virtualizer™ Studio, the industry's best virtual prototyping solution. It provides fast execution performance coupled with high debug efficiency. Users can build, test and debug software, all the way up to the apps level, with Virtualizer Studio. This enables unrivaled productivity and the much-desired shift left for software development. Virtualizer Studio provides extensive scripting support, particularly useful for doing such tasks as error and fault injection during software test. Users can also configure their virtual prototypes in hybrid emulation and prototyping scenarios with Synopsys HAPS® and ZeBu® products.

Synopsys provides a Virtualizer Development Kit (VDK) family covering many widely used processors. VDKs deliver the highest productivity through fast, register accurate simulation, advanced debug and analysis tools, and synchronized integration with third party software debuggers and embedded software development tools. For virtual prototype developers, Virtualizer Studio delivers the fastest time to custom VDK availability by enabling efficient creation of basic modeling blocks, their rapid assembly into a virtual prototype of the system and the automated packaging of VDKs for easy distribution. Figure 2 shows some of the key features of Synopsys VDKs, including links to both the debugger within Virtualizer Studio and third party tools.

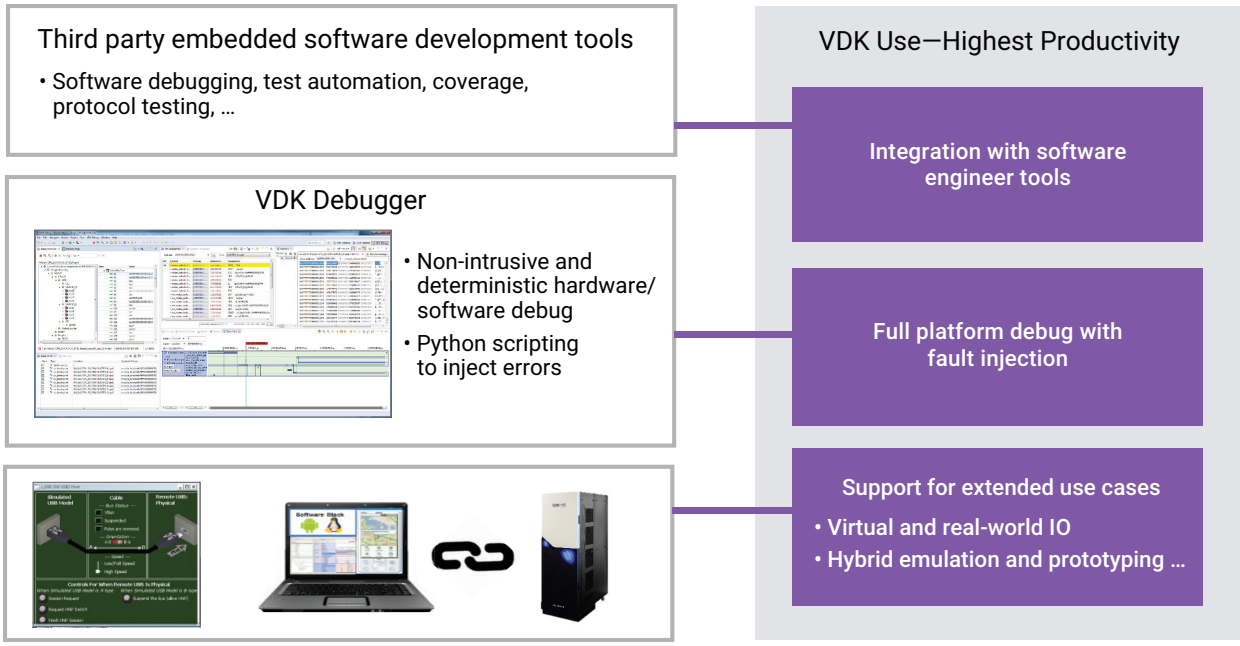

Figure 2: Key VDK features and links to other tools

Virtualizer Studio supports device virtualization using VirtIO, the standardized open interface for simplified devices such as disks, networks and graphics processing units (GPUs). By replacing the detailed device drivers with the VirtIO models, higher level software such as apps runs faster in the virtual prototype. The industry provides standard models for several types of devices, and these are provided as part of Virtualizer Studio. Example models include VIRTIO_BLK, which implements a block device such as a disk drive and uses a file on the host system for the data, and VIRTIO_INPUT, which implements human input devices such as keyboards and mice. It is possible to implement a virtual LCD touchscreen using the keyboard and mouse of the host system.

Since graphics processing consumes many processor cycles, perhaps the most interesting virtual model is VIRTIO_GPU. It offloads OpenGL commands to the host machine's GPU providing a significant performance boost when running graphics intensive operating systems such as Android and applications with high GPU requirements, for example, by running 2D and 3D graphics applications like Angry Birds on a virtual prototype in near real time. Figure 3 shows an example of how this works. The OpenGL calls from the app are handled by the Direct Rendering Manager (DRM), the part of the Linux kernel responsible for interfacing with GPUs. The Linux virglrenderer then interacts with the host GPU to provide hardware-accelerated OpenGL to the virtual prototype.

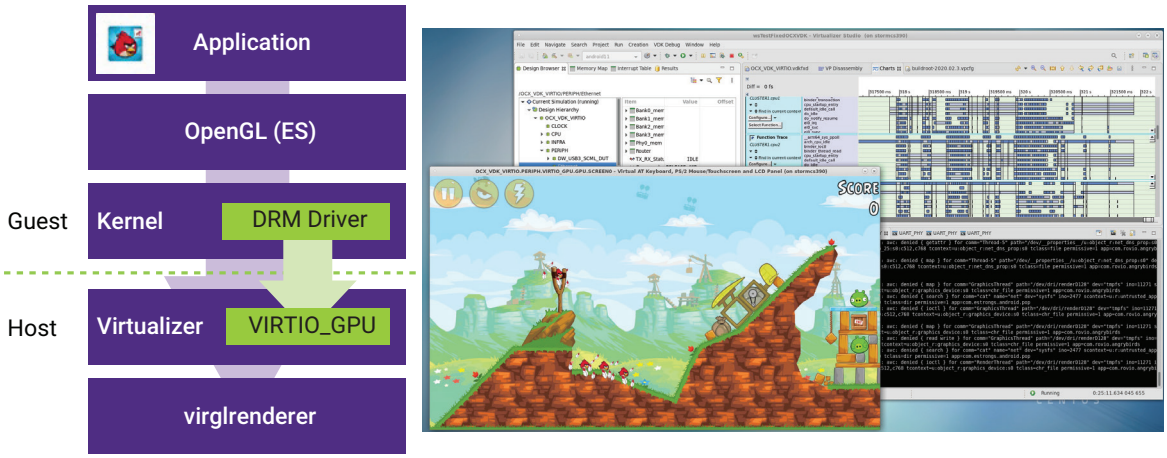

Figure 3: Example of VIRTIO_GPU usage

Industry-Leading Features

Checkpoint Restore

Synopsys Virtualizer Studio provides several unique features that make development and use of virtual prototypes both easier and more flexible. One example is the ability to checkpoint and restore the state of a platform test run. Users can select interesting points to save the current state and then later recover from that same state and continue the run. This supports several valuable use cases:

- Skipping past long initialization phases
- Enabling more tests to be run in a test cycle
- Jumping ahead to points of interest
- Saving error conditions and quickly reproducing them
- Tracking down failures in long runs

Users may set up their runs to take regular snapshots of the system state during long tests. If a failure occurs, the saved states can be used during debug to accurately locate the point of the failure and to quickly get back to the state of the system just before the failure occurred. Figure 4 shows the setup to take advantage of the checkpoint/restore capability.

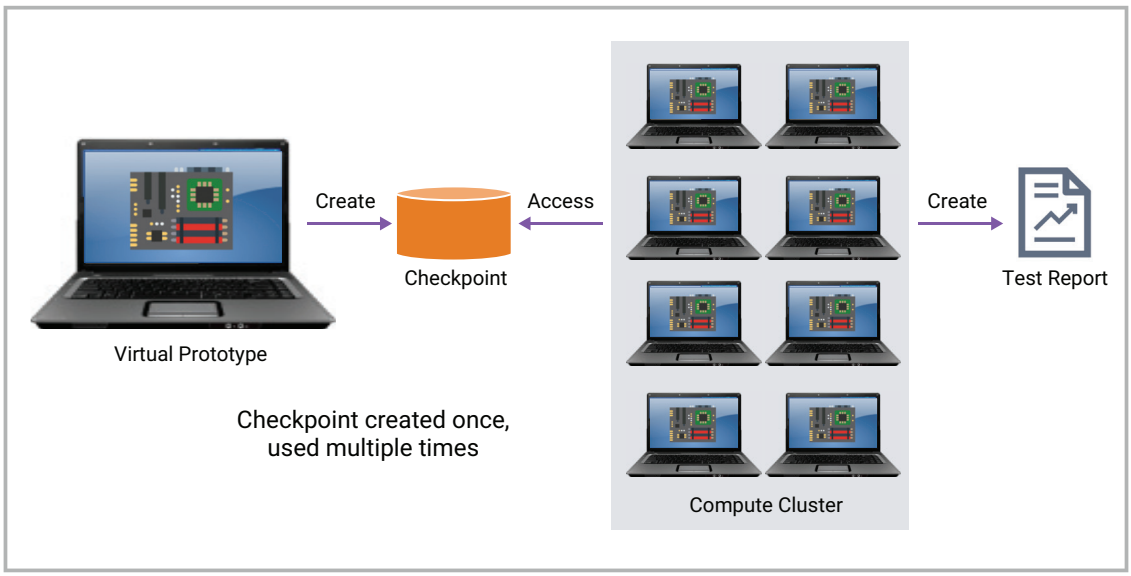

Figure 4: Checkpoint and restore in Virtualizer Studio

Efficient Software Debugging

As mentioned earlier, Virtualizer Studio includes a specialized built-in debugger that is extremely helpful when creating or using VDKs. VDK Debugger has unique features targeted at efficient co-debug of software and VDKs. It monitors activity within both the hardware models and the software, providing the ability to view and analyze both together, correlated in time. It handles both virtual and real-world I/O devices, offers integration with the popular GNU Debugger (GDB) and integrates with other commercial debugging tools. Figure 5 shows just a few of the features that enhance software development productivity on virtual prototypes.

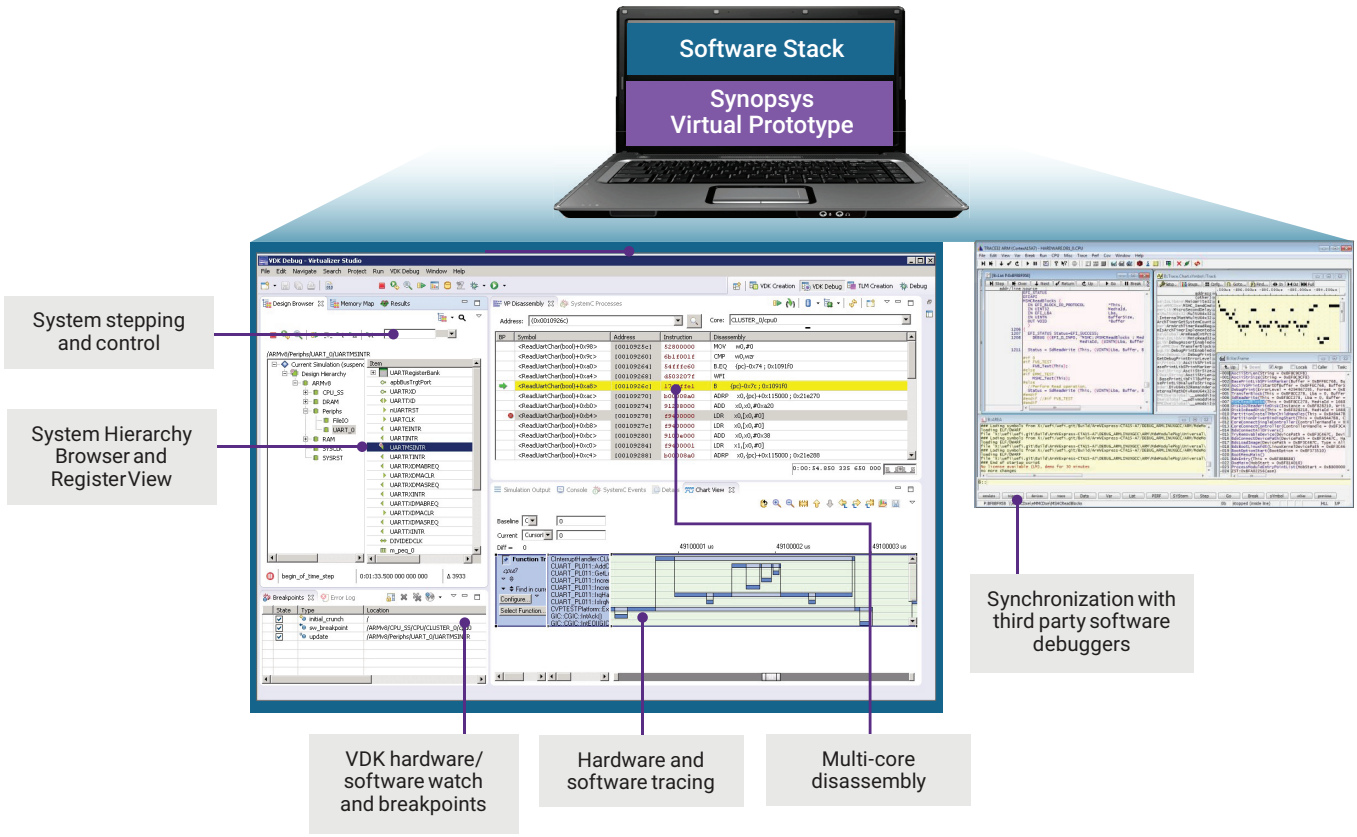

Figure 5: Key features of the VDK Debugger

Real World and Virtual I/O

Providing access to both virtual and real-world I/O devices is another noteworthy capability of Virtualizer Studio. With virtual I/O, the models in the virtual prototype look like real-world devices to the operating system on the host machine. In addition, the virtual models can connect to real physical interfaces, including PCI Express (PCIe), Universal Serial Bus (USB), Ethernet, Serial and more. This capability allows virtual prototypes to interact with the real-world environment and provide the real-world stimulus required for some types of tests in the virtual prototype. Examples include the virtual prototype connecting to a physical Ethernet network and the virtual prototype controlling physical USB devices connected to the host system, as shown in Figure 6. This flexibility enables efficient testing of hardware dependent software by testing software drivers against real devices, testing devices against the host operating system and applications, and debugging software in a realistic environment.

Figure 6: Interfacing with real-world USB I/O

Fast Track Bug Detection

A final unique feature of Virtualizer Studio is Fast Track, which is built into VDKs to quickly detect bugs in configuration or usage. This is invaluable in detecting and diagnosing situations when the host software is not following the correct procedure for model operation. For example, if registers are programmed in an order that violates the specification or if registers are programmed at an inappropriate time, Fast Track traces point this out. Fast Track reports configuration violations, warns of potential fault conditions and indicates the software routine that is behaving incorrectly. This allows the software team to debug the problem quickly, pinpoint the issue and take steps to fix it. Fast Track also provides unique visibility into VDKs by tracing the details of operation. This supports efficient debug of issues found during end-to-end software testing, from apps down to hardware dependent software.

Support for CI and CD

Because continuous integration and continuous deployment are becoming widely adopted by programmers, it is important that any virtual prototype used for software development supports these techniques. CI is the practice of merging all working copies of the software into a shared mainline codebase several times a day. This accelerates development by reducing the effort needed to merge and synchronize the updates from multiple programmers. The integration process includes quality metrics such as code analysis, coverage and performance. A tool such as Jenkins automates the process of building and testing the code while providing a path to CD. If the software is in a constant state of being available for release to customers, then deployment can be automated as well, as shown in Figure 7.

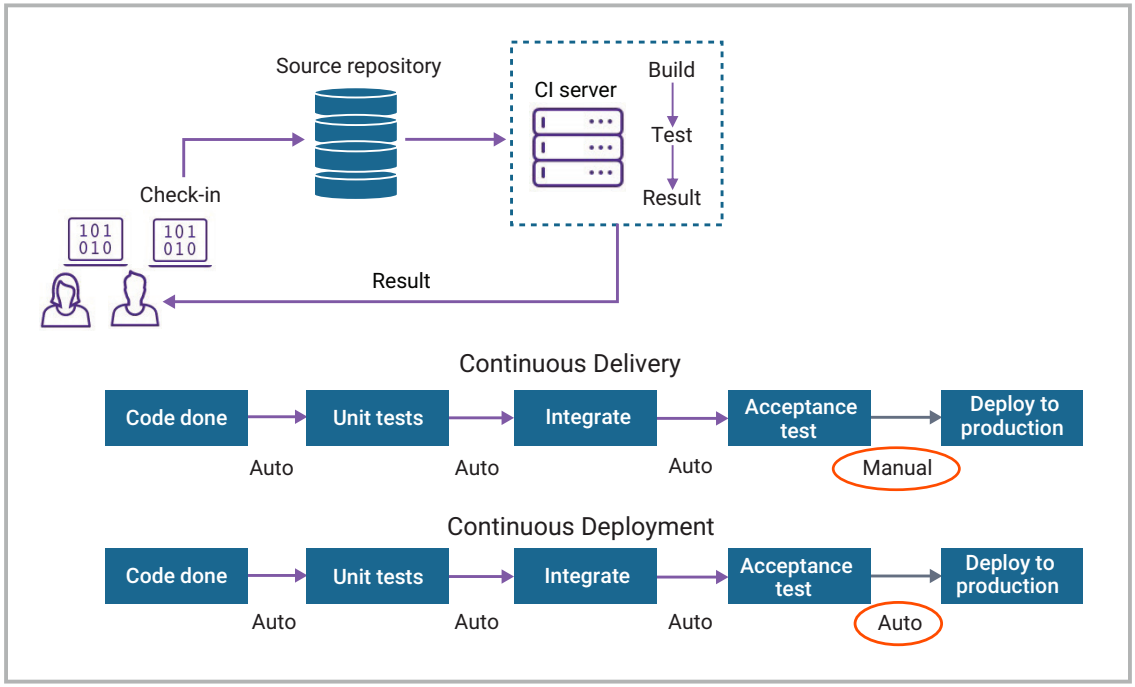

Figure 7: Continuous integration and continuous deployment

Virtualizer Studio supports CI and provides much of the automation needed. Python application programming interfaces (APIs) are available for all features, so tool operations can be scripted. Software build and test systems can use Python scripts to integrate with Virtualizer Studio, for example to configure, start and cleanly shut down a test run. This makes it possible to set up regression runs using virtual prototypes, including cloud deployment. Synopsys provides specific APIs to support regression runs and CI flows. Since Virtualizer Studio is based on Eclipse, it is easy to install plug-ins and to integrate with build and test tools, including Jenkins, and with source code control systems such as Git and Subversion.

Summary

Starting software development earlier while making it more efficient is essential for meeting SoC TTM requirements. Fast virtual prototypes are ideal for this process, and Synopsys Virtualizer Studio provides the industry's best solution. It improves software productivity by providing:

- Fast execution of complex operating systems and apps for end-to-end software testing
- Device virtualization techniques to boost performance
- Checkpoint/restore to quickly get to points of interest
- Unique debug capabilities that allow full visibility of hardware and software operations
- Integration with third party tools to create software development environments and support CI

The result is a much more efficient and productive process, writing and testing software early, with no need to wait for RTL design, using VDKs that can be inexpensively deployed to the entire software development team.

09/01/21.CS729033842-sw-develop-wp.

Pub: August 2021