Imperial Journal of Interdisciplinary Research (IJIR)
Vol-3, Issue-4, 2017
ISSN: 2454-1362, http://www.onlinejournal.in

Overview of Virtual Prototyping Techniques for System Development and Validation

Jayashree G S[1] & Mrs. Rekha B S[2]
[1]Dept. of ISE, R V College of Engineering, Bengaluru, India
[2]Asst. Professor, Dept. of ISE, R V College of Engineering, Bengaluru, India

Abstract— *As today's electronic industry is shifting its focus from ASICs(Application Specific-Integrated Circuit) to SoCs(System-on-Chip), it is very important to reduce design cycle time to meet different demands of the market, such as time to market. SoCs incorporate multiple functional blocks on a single chip whereas ASICs have single dedicated application specific block on the chip. SoCs are used in different kinds of computer systems like smart phones, tablets and embedded systems. A significant portion of system development and validation effort is spent on hardware device, device driver and firmware development. In traditional device, there is not enough time for software validation as driver and firmware development largely have to wait till a stable version of the device is available.*

Recently, virtual prototyping strategies have been generally investigated and used by both industry specialists and academic researchers. virtual prototyping brings better observability, traceability, debugging support and adaptability due to its white box nature. In the first place, virtual prototying has discovered their way into enabling early driver and firmware development and validation. Second, there has been some research for post-silicon functional validation using virtual prototype. Third, virtual prototyping is used for system validation by industry that had built hybrid emulation and hybrid FPGA(Field Programmable Gate Array) systems. This paper demonstrates how virtual prototyping is used for software/hardware co-design, system development and validation for the above three domains.

Keywords— *Virtual prototyping, software/hardware co-design, early software development, system validation.*

1. Introduction

According to a study conducted by the Internal Business Strategies, revenue reduces by 30% for chip manufacturers if there is a 3-month delay to market [1, 2]. The penalty is worse for evolving markets such as mobile devices. The following challenges are created for system development and validation due to the growing system complexity combined with early time-to-market.

1.1 Absence of early high quality software development

The traditional system development process begins with both functional and business requirements written in natural language. A system specification document is developed by combining marketing and system architecture resources. This specification defines the high-level architectural partition between software and hardware. The detailed hardware architecture is determined manually once the specification is complete. The designers can begin coding RTL (or drawing schematics) to complete the design, which is then fabricated to produce hardware prototypes. It is difficult for software developers to design and develop high quality firmware and device drivers as they have to wait for the first hardware/silicon prototype. The firmware and device drivers can be developed by the software engineers only according to the specification before a silicon device is ready. This leads to a lot of untested code development. This usually means that once the hardware is available a lot of time and effort is spent on validating, debugging and re-writing software code. The traditional system development process is shown in Figure 1.

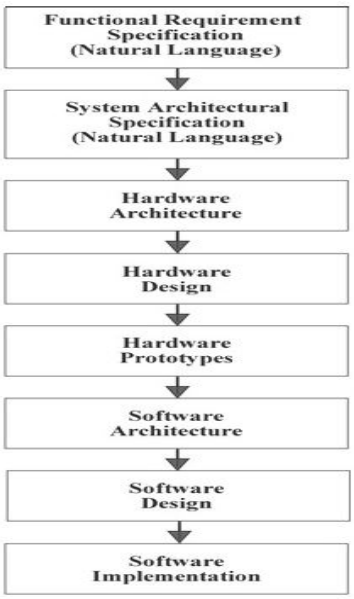

Figure 1: The traditional sequential development process serializes hardware and software development.

1.2 Absence of early post-silicon functional validation

Post-silicon validation is used for detecting and fixing bugs in integrated circuits and systems after the hardware is manufactured. It is a major aspect of system validation. There are many challenges faced when the post-silicon functional validation is accelerated [3]: limited silicon traceability and observability, lack of good test coverage metrics, lack of early test readiness.

1.3. Absence of early system integration validation

The different components of a system interact with each other to achieve system functionalities and the desired workflows. For example, Low power feature of a system can only be detected if the power management unit interacts with other system components. It is very challenging for developers to test if one of the component can interact with the other components correctly before all the system components are available. Hence it is difficult to realize if the desired flow and functionalities are obtained [4].

2. Virtual Prototype Overview

Over the past several years, virtual platforms and virtual prototypes have been applied in software and hardware development, integration and validation before the silicon devices are ready [5-7]. Virtual prototyping techniques have been investigated and used by both industry specialists and academic researchers.

2.1. Enable early firmware and driver development.

Virtual prototypes are software-simulation based models that are designed and developed according to the hardware specification. These model simulate functional hardware behaviours and enable unmodified software execution on them. The software developers develop and validate drivers and firmware using virtual prototypes without silicon hardware [8].

2.2. Accelerate post-silicon functional validation.

Virtual prototypes provide better observability, traceability and controllability due to its white box nature. These features help developers enable early evaluation and test generation for post-silicon functional validation. The coverage of developed post-silicon functional tests can be evaluated by developers and better high-quality post-silicon functional tests can be developed even before the silicon devices or the FPGA prototypes are available.

2.3. Build hybrid emulation and FPGA systems for integration testing.

Emulation/FPGA and virtual prototyping are combined to produce hybrid emulation/FPGA. It enables early architecture validation, software development and RTL verification. Unmodified software can be validated on RTL design. Both the software and RTL design can be verified.

3. A Sample Virtual Prototype

Before the recent detailed advances in virtual prototyping are discussed, a sample virtual prototype is introduced. A virtual prototype is a software functional model that implements the bahaviour of the real hardware device. It can be implemented using different languages such as C, C++, System C and DML. Virtual prototypes provide a lot of advantages. Better observability is provided for developers to observe and capture all interface and internal hardware states. The models can be debugged and traced as they enable better traceability. Better controllability is supported which enables developers to modify hardware behaviours for software and system validation.

There are many open source virtual devices available. In this paper, the virtual device from QEMU [9,10] which models the Intel 8255x 10/100 Mbps(E100) network adapter is taken as an example. E100 device is controlled by the corresponding driver through interface registers and interrupts. The components included in the E100 virtual device is shown in Figure 2.

```
// Device state Structure
typedef struct
{
  //PCI configuration
  PCIDevice dev;
  //Device I/O registers
  uint8_t mem[PCI_MEM_SIZE];
  ......
  //SCB stat/ack byte
  uint8_t scb_stat;
  ......
} EEPRO100State;

// 2. Memory-mapped I/O register function
static void eepro100_write (void *opaque, hwaddr addr, uint64_t data, unsigned size)
{
  EEPRO100State *s = (EEPRO100State *) opaque;
  ......
  tx_command(s);
  ......
}

// 3. Device behavioral function
static void tx_command (EEPRO100State *s)
{
  ......
  //Send a network packet
  qemu_send_packet();
  ......
}

// 4. Network receive function
static ssize_t eepro100_receive (NetClientState *nc, const uint8_t *buf, size_t size)
{
  ......
  //Fire an interrupt
  eepro100_fr_interrupt(s);
}
```

Figure 2: Excerpt of QEMU EEPro100 Virtual Device

1. The device state, EEPRO100State keeps track of the E100 device state and the device PCI configuration.
2. The I/O register functions such as eepro100_write are registered as QEMU callback functions to access interface registers and trigger functional behaviours.
3. The device behavioural functions such as tx_command are invoked by the I/O register functions to execute the corresponding commands.
4. The device specific functions such as eepro100_recieve are used for receiving data or packets from the outside environment. For example, When a network packet is received by the QEMU from the outside environment, it invokes the function eepro100_receive to process the packet and fire the interrupt using the function eepro100_fr_interrupt.

4. Early Firmware and Driver Development

Over the past several years, virtual platforms and virtual prototypes and devices have been widely used for enabling early firmware and software development. Virtual prototypes are software models that behave as the corresponding physical devices. The firmware and drivers can be validated with virtual prototypes instead of physical devices when silicon prototypes are unavailable. Virtual prototype environments include the dedicated ones from Electronic Design Automation (EDA) vendors such as Synopsys [11] and Cadence [12] and those adapted from various virtual machine (VM) environments such as Simics [13], VMWare [14],Xen [15], QEMU [9,10].

It is challenging to develop the software before the first silicon prototype is ready. Due to the black box nature of silicon prototypes, they have limited debugging and tracing abilities. These limitations bring a lot of difficulties to firmware and driver development and validation. Virtual prototyping techniques bring a lot of advantages in enabling software development without silicon prototypes [5, 8]. The industry companies use all kinds of virtual platforms. Those platforms can enable early operating system booting and driver development. Virtual prototypes can greatly left-shift the integration process. The firmware, drivers and the operating systems can be validated on the virtual platform even before the silicon platform is ready. The integration cycle can be greatly reduced as the software can run successfully on the first day once the silicon prototype becomes available. For example, A virtual prototype was developed by Intel to enable early driver development for their 40G Ethernet network adapter [9]. With the virtual prototyping techniques, the corresponding drivers were developed and the bugs were found and fixed before a silicon card became available.

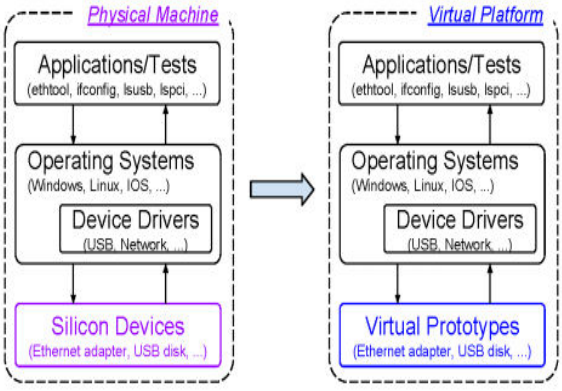

Figure 3: Enable Early Driver Development using Virtual Prototypes

As shown in Figure 3, virtual prototypes are running in virtual platforms while silicon devices are running in physical machines. The silicon devices and virtual prototypes are developed according the hardware specifications. Hence, they behave the same to enable software development and validation.

Virtual prototypes can be used by the driver developers to start driver development even without a silicon device prototype. The similar setups can be applied for enabling early firmware development. Firmware functionalities can be tested using the virtual prototypes instead of silicon devices.

5. Acceleration Post-Silicon Validation

The product development cycle has many stages. Recently post-silicon validation has turned out to be increasingly imperative because of high system complexities and short time-to-market. As per some recent reports, a lot of overall system development and validation time has been dedicated to post-silicon validation [16]. Because of this fact, the post-silicon validation developers face an increasing pressure. It is very critical to reduce the development time and cost of post-silicon validation by developing efficient and innovative approaches and methodologies. There are numerous key challenges faced in achieving accelerated and low cost post-silicon functional validation.

- **Limited Silicon Observability.** The nature of the silicon device is typically a black box. Only a limited amount of run-time information can be obtained from the device internal with built-in test circuitries and advanced logic analysers. Post-silicon validation is difficult due to such limited observability.

- **Test Coverage Estimation.** The lack of good test coverage metrics over a silicon device makes it difficult to assess the effectiveness of test cases and to prioritize their application. Furthermore, coverage metrics established in hardware design are not appropriate for testing the integration with software.

- **Test Readiness.** There is a need for high-quality tests for post-silicon validation. Great tests can check the correctness and accuracy, as well as detect bugs and security issues for post-silicon validation. A lot of time can be saved and post-silicon validation can be accelerated if developers can develop efficient tests before the silicon prototypes are ready.

There is potential to solve these challenges without the available silicon devices by using virtual prototyping techniques. In some current research, a systematic approach to accelerate post-silicon functional validation using virtual prototypes is presented by Kai et al [17, 18]. In the pre-silicon stage, test cases are evaluated on virtual prototypes to estimate the post-silicon test coverage. With the estimated test coverage results, better test cases can be generated to improve the coverage and further validate silicon designs in the post-silicon stage.

5.1. Coverage Evaluation of Validation Test

A lot of time can be saved in the post-silicon stage if high-quality tests are developed before a silicon device is ready [4]. However, it is difficult to evaluate if the post-silicon tests are good or not. Test coverage is a popular evaluation method. In software domain, test coverage has been widely used to estimate the quality of a test suite. However, there is a lack of good coverage metrics methodologies for evaluating post-silicon tests on hardware devices. In paper [19], some hardware-related coverage metrics for evaluating post-silicon tests with virtual prototypes was proposed by Kai et al. Their approach applied the validation tests to virtual devices to estimate the coverage on corresponding silicon devices.

An online capture and offline replay approach is proposed. In their approach, virtual devices and the corresponding drivers are run within a virtual platform. A test suite is then issued to trigger the hardware functionalities. Coverage reports are produced during this process by an offline-replay engine that captures and consumes the hardware states and hardware/software interactions. The coverage reports gives good estimation of the test suite to developers. They have applied this approach to estimating coverage of some test suites on many virtual network devices. Furthermore, they have extended their approach to support coverage estimation and conformance checking on silicon devices in the post-silicon validation.

5.2. Concolic Test Generation

Once the coverage evaluation results are obtained on virtual prototypes, test generation can be conducted to provide high-quality post-silicon tests much before the first silicon prototype is available. Kai et al. developed a concolic approach to generate post-silicon tests with virtual prototypes [17]. The "concolic" is borrowed from the software testing domain literally and the concolic test generation is conducted by integrating concrete runtime execution and symbolic execution.

The concreate traces within a virtual platform is first captured. The traces captured are analyzed and the device state under test are identified. Tests are then generated by symbolically executing the virtual prototype with a symbolic request from these device states. The tests generated are then issued concretely to the FPGA prototype and the physical device. This approach has been evaluated on several virtual network devices.

The test coverage is improved significantly because of the the generated test cases [17]. For many

virtual devices, the generated test cases trigger 100% function coverage and the branch coverage is improved by more than 30%. Both the generated tests and the test suite have been issued to the silicon devices. 20 inconsistencies between the virtual prototype and silicon devices with conformance checking using generated tests were detected.

6. Hybrid Emulation and FPGA Prototypes

The combined virtual prototypes and other methodologies have begun to show the strengths [20] for certain tasks. There are 2 common frameworks. First is the hybrid prototypes which combine virtual prototypes with FPGA-based prototypes. Second is the hybrid emulation which combines virtual prototypes with RTL emulation. Both the hybrid prototype and the hybrid emulation approaches are used to mitigate virtual prototypes and RTL availability. The developers can mix RTL designs and virtual prototypes to run a system. Using this method they can use what becomes most readily available to build a system as early as possible.

Hybrid prototypes and hybrid emulation have been employed to different kinds of use cases.

- **Reuse available RTL designive.** In many cases it is better to use RTL design or third-party IP instead of virtual prototypes. When a new system is being designed, there is a high possibility of finding some pre-existing RTL designs from legacy projects or there are some third-party companies that provide some IP's. It is wiser to reuse them in the hybrid system as it saves the time spent on developing a new virtual prototype.
- **Use only necessary RTL design.** For several models such as GPUs, it might be difficult to model them in a virtual prototype. Moreover, there are several systems that require cycle-accurate hardware models for timing and performance verification. Furthermore, only one specific RTL design may be verified by the developers. Under such cases, it is necessary to combine RTL design with FPGA/emulation.
- **Early system integration and architecture validation.** It is better to use a hybrid system to validate the system architecture and the functionalities. It is not easy to determine if either RTL design or a virtual prototype is available first when a new system is being designed. It is always better to use whatever is available first for early system integration and architecture validation.

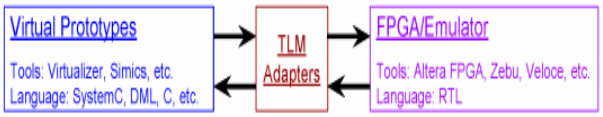

Figure 4. Hybrid Framework

The basic frameworks of hybrid prototype and emulation are shown in Figure 4. A basic virtual platform is built based on different virtual prototypes on the left side. It typically implements a basic system framework with only a few components missing. On the right side, some RTL designs or IPs are simulated using FPGA or hardware emulators. Therefore, the functionalities missing on the left side is complemented by the FGPA or the emulator. To connect the two sides, a transaction level modelling (TLM) adaptor is required as virtual prototypes are generally implemented at the transaction level. The TLM adaptor acts as a bridge between the virtual prototypes and the RTL simulation. Thus, a complete system can be simulated for development and verification.

7. Conclusion

In this paper, a summary of the current research and industry utilization of virtual prototyping techniques is presented. Virtual prototyping techniques have shown their powerfulness and strengths in enabling early software development and accelerating post-silicon functional validation. The hybrid prototypes and emulation can better left-shift software development, hardware verification and system integration. In the future, there are still many unexplored areas which can take advantage of virtual prototyping techniques.

8. References

[1] Shunan Mu, Guoqing Pan, Zhihao Tian and Jiancheng Feng, “A survey of virtual prototyping techniques for system development and validation,” International Journal of Computer Science & Engineering Survey, December 2015.

[2] International Business Strategies, Inc., “ Global systemIC industry service monthly reports,” http://www.ibs-inc.net, 2014.

[3] S. Mitra, S. Seshia, and N. Nicolici, “Post-silicon validation opportunities, challenges and recent advances,” in DAC, 2010.

[4] Q. Wang, R. Kassa, W. Shen, N. Ijih, B. Chitlur, M. Konow, D. Liu, A. Sheiman, and P. Gupta, “An fpga based hybrid processor emulation platform,” in FPL, 2010.

[5] P. Sampath and B. Rachana Rao, “Efficient embedded software development using QEMU,” in 13th Real Time Linux Workshop, 2011.

[6] J. Gladigau, C. Haubelt, and J. Teich, “Model-based virtual prototype acceleration,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2012.

[7] Y.-C. Lee, C.-T. Kuo, and L.-P. Chang, “Design and implementation of a virtual platform of

solidstate disks," IEEE Embedded Systems Letters, 2012.

[8] J. Gladigau, C. Haubelt, and J. Teich, "Model-based virtual prototype acceleration," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2012.

[9] F. Bellard, "QEMU, a fast and portable dynamic translator," in USENIX ATEC, 2005.

[10] B. Fabrice, "QEMU," http://wiki.qemu.org/Main_Page, 2013.

[11] Synopsys, "Synopsys virtual prototyping solutions," http://www.synopsys.com/prototyping/virtualprototyping/Pages/default.aspx.

[12] Cadence, "Cadence virtual system platform," http://www.cadence.com/products/sd/virtual_system/pages/default.aspx.

[13] Windriver, "Simics full system simulator," http://www.windriver.com/products/simics/.

[14] VMware, "Vmware virtualization technology," http://www.vmware.com/virtualization/.

[15] Xen, "The xen project," http://www.xenproject.org/.

[16] E. Singerman, Y. Abarbanel, and S. Baartmans, "Transaction based pre-to-post silicon validation," in DAC, 2011.

[17] K. Cong, F. Xie, and L. Lei, "Automatic concolic test generation with virtual prototypes for postsilicon validation," in ICCAD, 2013.

[18] K. Cong, "Post-silicon functional validation with virtual prototypes," Ph.D. dissertation, Portland State University, 2015.

[19] K. Cong, L. Lei, Z. Yang, and F. Xie, "Coverage evaluation of post-silicon validation tests with virtual prototypes," in DATE, 2014.

[20] V. Srinivasan, F. Schirrmeister, V. Singh, and R. Klein, "Why hybrid platforms are needed for presilicon hardware and software development," in Electronic Design Process Symposium (EDPS), 2015.