A SURVEY OF VIRTUAL PROTOTYPING TECHNIQUES FOR SYSTEM DEVELOPMENT AND VALIDATION

Shunan Mu, Guoqing Pan, Zhihao Tian and Jiancheng Feng

Beijing Aerospace Measurement and Control Technology Co., LTD., Beijing, China

ABSTRACT

Recently, different kinds of computer systems like smart phones, embedded systems and cloud servers, are more and more widely used and the system development and validation is under great pressure. Hardware device, firmware and device driver development account for a significant portion of system development and validation effort. In traditional device, firmware and driver development largely has to wait until a stable version of the device becomes available. This dependency often leaves not enough time for software validation.

Recently, virtual prototyping techniques have been widely explored and utilized by both industry engineers and academic researchers. White box nature of virtual prototyping brings better observability, traceability, debugging support and adaptability. First, virtual prototyping has found their way into enabling early firmware and driver development and validation. Second, there has been some research utilizing virtual prototyping for post-silicon functional validation. Third, the industry has built hybrid emulation and hybrid FPGA systems for system validation using virtual prototyping. In this paper, we demonstrate how recent work and products utilize virtual prototyping techniques for system development and validation in the above three domains.

KEYWORDS

Virtual Prototyping, Early Software Development, System Validation.

1. INTRODUCTION

A recent study by International Business Strategies indicates that a 3-month delay to market reduces revenue by about 30% for chip manufacturers in general, and the penalty is even worse for fast-evolving markets such as mobile devices [1]. The growing system complexity combined with shorten time-to-market has created the following challenges for system development and validation.

- **Lack of early high quality software development.** In the traditional system development process, software like firmware and device drivers largely has to wait until the first silicon prototype becomes available. Before that, it is very difficult for software developers to design and develop high quality firmware and device drivers [2, 3]. Before a silicon device is ready, software developers can only develop firmware and device drivers according to specifications. Such kind of development can lead to a lot of untested code being developed. It usually means that a large amount of time is needed for validating, debugging and rewriting software code once hardware is available.

- **Lack of early post-silicon functional validation.** Since post-silicon functional validation is one major aspect of system validation. To accelerate post-silicon functional validation, we need to face the following challenges [4]: limited silicon observability and traceability, lack of good test coverage estimation, lack of early test readiness.
- **Lack of early system integration validation.** A system includes many different components. Different components interact with each other to achieve system functionalities and desired workflows. For example, the power management unit needs to interact with other system components to realize low power feature. Before all necessary system components are available, it's very challenging for developers to test if one component can interact with other components correctly to realize the desired flow and functionalities [5].

In the past several years, virtual platforms and virtual prototypes have been increasingly applied in hardware and software development, integration and validation before silicon devices are ready [6-8]. Virtual prototyping techniques have been widely explored and used by both industry engineers and academic researchers.

- **Enable early firmware and driver development.** Virtual prototypes are software models developed according to the hardware specification. Such models simulate functional hardware behaviours and enable unmodified software execution on them. With virtual prototyping, software developers can develop and validate firmware and drivers without silicon hardware [9].
- **Accelerate post-silicon functional validation.** Because of the white box nature of virtual prototypes, they can provide better observability, traceability and controllability. The developers can take advantage of these features to enable early coverage evaluation and test generation for post-silicon functional validation [10-12]. Before silicon devices or FPGA prototypes are delivered, the developers can evaluate the coverage of developed post-silicon functional tests and develop better and high-quality post-silicon functional tests.
- **Build hybrid emulation and FPGA systems for integration testing.** Hybrid emulation/FPGA combines emulation/FPGA and virtual prototyping to enable early architecture validation, software development and RTL verification. In this way, unmodified software can be validated on the RTL design. Both software and RTL design can be verified.

The remainder of this paper is structured as follows. Section 2 provides a sample virtual prototype. Section 3 presents how virtual prototyping enables early firmware and driver development. Section 4 illustrates how to accelerate post-silicon functional validation with virtual prototyping. Section 5 elaborates how to build hybrid emulation and FPGA systems for integration testing with virtual prototyping. Section 6 discusses the conclusion.

2. A SAMPLE VIRTUAL PROTOTYPE

Before we demonstrate recent detailed advances in virtual prototyping, we would like to introduce a sample virtual prototype. A virtual prototype is a software functional model which implements the behaviour of the real device. Virtual prototypes can be implemented using different languages such as C, C++, DML and System C, but they have the same nature. Virtual prototypes provide a lot of advantages. First, they provide better observability for developers to observe and capture all interface and internal hardware states. Second, they enable better traceability which supports the debugging and tracing on the models. Third, they support better controllability which allows developers to modify hardware behaviours for software and system validation.

There are many open source available virtual devices. In this paper, we take one virtual device from QEMU [13, 14] as our example. This virtual device models the Intel 8255x 10/100 Mbps

(E100) network adapter. E100 device is controlled by the corresponding driver through interface registers and interrupts. As shown in Figure 1, the E100 virtual device includes the following components:

```
// 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 1. Excerpt of QEMU EEPro100 Virtual Device

- The device state, *EEPRO100State*, which keeps track of the E100 device state and the device PCI configuration;
- The I/O register functions such as *eepro100_write* which are registered as QEMU callback functions to access interface registers and trigger functional behaviours;
- The device behavioural functions such as *tx_command* which are invoked by the I/O register functions to execute the corresponding commands;
- The device specific functions such as *eepro100_receive* which are used for receiving data or packets from the outside environment. For example, when QEMU receives a network packet

from the outside environment, it invokes *eepro100_receive* function to process the packet and fire the interrupt using *eepro100_fr_interrupt* function.

3. EARLY FIRMWARE AND DRIVER DEVELOPMENT

In the past several years, virtual platform and virtual devices have been widely used for enabling early software and firmware development. Since virtual prototypes can behave as the corresponding physical devices, drivers and firmware can be validated with virtual prototypes instead of physical prototypes when silicon prototypes are not ready. Virtual prototype environments include the dedicated ones from Electronic Design Automation (EDA) vendors such as Cadence [15] and Synopsys [16] and those adapted from various virtual machine (VM) environments such as QEMU [13, 14], Simics [17], VMWare [18], Xen [19].

Before the first silicon prototype is ready, it is very challenging to develop the corresponding software. Moreover, silicon prototypes can only provide limited debugging and tracing abilities due to their black box nature. These limitations bring a lot of difficulties to driver and firmware development and validation. Recently virtual prototyping techniques bring advantages in enabling software development without silicon prototypes required [6, 9, 20]. All kinds of virtual platforms have been widely used by industry companies [21]. Those platforms can enable early operating system booting and driver development. Virtual prototypes can greatly shift-left the integration process. Before a silicon platform is ready, the operating systems, drivers and firmware can be validated on a virtual platform. Once a silicon prototype becomes available, the software can be running successfully on the first day. It can greatly reduce the integration cycles. For example, Intel developed a virtual prototype to enable early driver development for their 40G Ethernet network adapter [9]. With the virtual prototyping techniques, the corresponding drivers were developed and driver bugs were found and fixed before a silicon card became available.

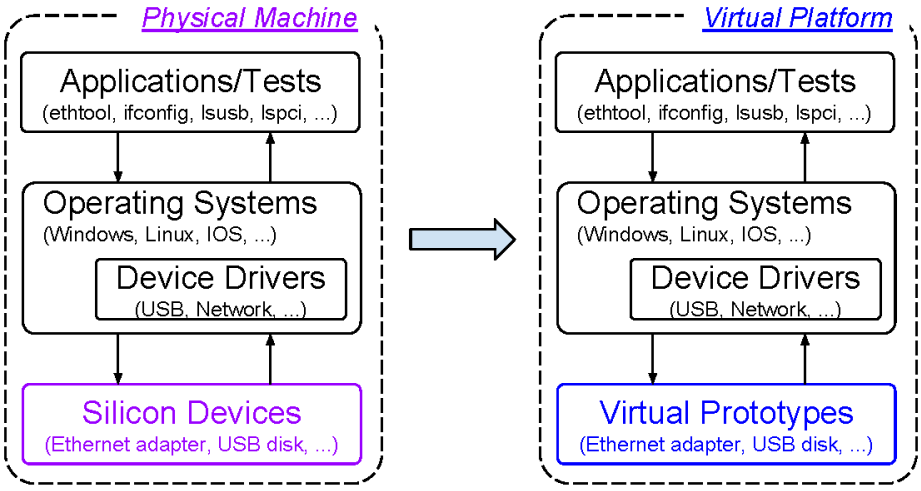

Figure 2. Enable Early Driver Development using Virtual Prototypes

As shown in Figure 2, virtual prototypes are running in virtual platforms while silicon devices are running in physical machines. Virtual prototypes and silicon devices can behave the same to enable software development and validation because they are both developed according to hardware specifications. By using virtual prototypes instead of silicon devices, driver developers can start driver development without a silicon device prototype. The similar setups can be applied for enabling early firmware development. Firmware can be running on virtual prototypes instead of silicon devices so that we can test the firmware functionalities.

4. ACCELERATING POST-SILICON VALIDATION

There are several stages in the product development cycle. Recently post-silicon validation has become more and more important and critical due to high system complexities and short time-to-market. According to some recent reports, more and more overall system development and validation time has been devoted to post-silicon validation [22]. Due to this fact, developers for post-silicon validation face an increasing pressure. It is very critical to develop efficient and innovative approaches and methodologies to reduce the development time and cost of post-silicon validation. There are several key challenges in achieving accelerated and low-cost post-silicon functional validation.

- **Limited Silicon Observability**. The silicon device is typically a black box. The amount of run-time information that can be retrieved from the device internal with build-in test circuitries and advanced logic analysers is still quite limited. Such limited observability makes post-silicon validation difficult.
- **Test Coverage Estimation**. There lacks good test coverage metrics over a silicon device. Therefore, it is difficult to assess the effectiveness of test cases and prioritize their application. In addition, coverage metrics rooted in hardware design are not well suited for testing the integration with software.
- **Test Readiness**. High-quality tests are required for post-silicon validation. Good tests can not only check the correctness and accuracy, but also detect bugs and security problems for post-silicon validation. It is better that developers can develop efficient tests before silicon prototypes become ready so that it can save time and speed up post-silicon validation.

Virtual prototyping techniques provide potentials for solving the above challenges without available silicon devices. In some recent research, Kai *et al.* [10, 12, 23] present a systematic approach to accelerating post-silicon functional validation with virtual prototypes. In the pre-silicon stage, post-silicon test coverage is estimated by evaluating the test cases on the virtual prototypes. With the estimated test coverage results, better test cases can be generated to improve coverage and further validate silicon designs in the post-silicon stage.

4.1. Coverage Evaluation of Validation Tests

In order to save time in the post-silicon stage, it is better to develop high-quality tests before a silicon device is ready [4]. However, how to evaluate if post-silicon tests are good or not is very difficult. One popular evaluation method is test coverage [24]. Test coverage has been widely used in software domain to estimate the quality of a test suite. However, there lacks of good coverage metrics methodologies for evaluating post-silicon tests on hardware devices. In paper [23], Kai *et al.* proposed some hardware-related coverage metrics for evaluating post-silicon tests with virtual prototypes. Their approach applied the validation tests to virtual devices to estimate the coverage on corresponding silicon devices.

They have proposed an online capture and offline replay approach. In their approach, they first run virtual devices and the corresponding drivers within a virtual platform. Then a test suite is issued to trigger hardware functionalities. In this process, hardware states and hardware/software interactions are captured and then consumed by an offline-replay engine to produce coverage reports. The coverage reports give developers good estimation of the test suite. They have applied the approach to estimating coverage of some test suites on several virtual network devices. Furthermore, they have extended their approach to support coverage estimation and conformance checking on silicon devices in the post-silicon validation [11, 25, 26].

4.2. Concolic Test Generation

With coverage evaluation results on virtual prototypes, test generation can be conducted to provide high-quality post-silicon tests before the first silicon prototype becomes available. Kai *et al.* have developed a concolic approach to generation of post-silicon tests with virtual prototypes [10]. They borrow "concolic" from software testing domain literally and conduct concolic test generation by integrating concrete runtime execution and symbolic execution [27].

They first capture concrete traces within a virtual platform. The capture traces are analysed and device states under test are identified. Then they symbolically execute the virtual prototype with a symbolic request from these device states to generate tests. The generated tests are issued concretely to the FPGA prototype and physical device. This approach has been evaluated on several virtual network devices.

As shown in the paper [10], the generated test cases improve test coverage significantly. For some virtual devices, the generated test cases trigger 100% function coverage and improve the branch coverage more than 30%. Both the test suite and the generated tests have been issued to silicon devices. They detected 20 inconsistencies between virtual prototypes and silicon devices with conformance checking using generated tests.

5. HYBRID EMULATION AND FPGA PROTOTYPES

For some certain tasks, the combined virtual prototypes and other methodologies have begun to show the strengths [28, 29]. There are two common frameworks. One is hybrid prototypes which combine virtual prototypes with FPGA-based prototypes [30-32]. The other is hybrid emulation which combines virtual prototypes with RTL emulation.

Hybrid prototypes and hybrid emulation are approaches to mitigate both virtual prototypes and RTL availability. To run a system, the developers can mix virtual prototypes and RTL designs. In this way, they can use what becomes most readily available and reliable 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 design**. Sometimes it is better to use RTL design or third-party IP instead of virtual prototypes. When a new system is designed, it is highly possible that there are some pre-existing RTL designs from a legacy project or there are some IPs provided by third-party companies. If we can reuse them in a hybrid system, it can save time to develop a new virtual prototype.
- **Use necessary RTL design**. For some models such as GPUs, it might not be so easy to model in a virtual prototype. Moreover, some systems require cycle-accurate hardware models for timing and performance verification. Furthermore, developers might want to only verify one specific RTL design. Under the above cases, it is necessary to combine RTL design with FPGA/emulation.
- **Early system integration and architecture validation**. In order to validate system architecture and functionalities, it is better to use hybrid system. When a new system is designed, it is difficult to determine either RTL design or a virtual prototype is available first. It is better to use whatever available as early as possible for early system integration and architecture validation.

Figure 3. Hybrid Framework

The basic frameworks of hybrid prototype and emulation are shown in Figure 3. On the left side, a basic virtual platform is built based on different virtual prototypes. It usually 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 can be complemented by FPGA or emulator. To connect two sides, a transaction-level modelling (TLM) adaptor is required since virtual prototypes are usually implemented at the transaction level. The TLM adaptor acts as a bridge between virtual prototypes and RTL simulation. In this way, a complete system can be simulated for development and verification.

6. CONCLUSIONS

In this paper, we summarize the current research and industry utilization of virtual prototyping techniques. 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 shift-left software development, hardware verification and system integration. In the future, there are still many unexplored areas which can take advantage of virtual prototyping techniques.

REFERENCES

[1] International Business Strategies, Inc., " Global systemIC industry service monthly reports," http://www.ibs-inc.net, 2014.
[2] S. Nelson and P. Waskiewicz, "Virtualization: Writing (and testing) device drivers without hardware," 2011. [Online]. Available: http://www.linuxplumbersconf.org/2011/ocw/sessions/243
[3] T. Eckart and M. Schnieringer, "Development and verification of embedded firmware using virtual system prototypes," in International Symposium on System-on-Chip, 2006.
[4] S. Mitra, S. Seshia, and N. Nicolici, "Post-silicon validation opportunities, challenges and recent advances," in DAC, 2010.
[5] 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.
[6] P. Sampath and B. Rachana Rao, "Efficient embedded software development using QEMU," in 13th Real Time Linux Workshop, 2011.
[7] J. Gladigau, C. Haubelt, and J. Teich, "Model-based virtual prototype acceleration," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2012.
[8] Y.-C. Lee, C.-T. Kuo, and L.-P. Chang, "Design and implementation of a virtual platform of solid-state disks," IEEE Embedded Systems Letters, 2012.
[9] S. Nelson and P. Waskiewicz, "Virtualization: Writing (and testing) device drivers without hardware," in Linux Plumbers Conference, 2011.
[10] K. Cong, F. Xie, and L. Lei, "Automatic concolic test generation with virtual prototypes for post-silicon validation," in ICCAD, 2013.
[11] L. Lei, F. Xie, and K. Cong, "Post-silicon conformance checking with virtual prototypes," in DAC, 2013.
[12] K. Cong, "Post-silicon functional validation with virtual prototypes," Ph.D. dissertation, Portland State University, 2015.
[13] F. Bellard, "QEMU, a fast and portable dynamic translator," in USENIX ATEC, 2005.
[14] B. Fabrice, "QEMU," http://wiki.qemu.org/Main_Page, 2013.
[15] Cadence, "Cadence virtual system platform," http://www.cadence.com/products/sd/virtual_system/pages/default.aspx.

[16] Synopsys, "Synopsys virtual prototyping solutions," http://www.synopsys.com/prototyping/virtualprototyping/Pages/default.aspx.
[17] Windriver, "Simics full system simulator," http://www.windriver.com/products/simics/.
[18] VMware, "Vmware virtualization technology," http://www.vmware.com/virtualization/.
[19] Xen, "The xen project," http://www.xenproject.org/.
[20] C. Shin and Y. Kim, "Development of a virtual platform for IP and firmware verification," in SoC Design Conference (ISOCC), 2014.
[21] A. Khan, W. Ma, C. Wolf, and B. Werner, "Multi-threaded Simics Systemc virtual platform," in ICCAD, 2015.
[22] E. Singerman, Y. Abarbanel, and S. Baartmans, "Transaction based pre-to-post silicon validation," in DAC, 2011.
[23] K. Cong, L. Lei, Z. Yang, and F. Xie, "Coverage evaluation of post-silicon validation tests with virtual prototypes," in DATE, 2014.
[24] K. Balston, M. Karimibiuki, A. Hu, A. Ivanov, and S. J. E. Wilton, "Post-silicon code coverage for multiprocessor system-on-chip designs," IEEE Transactions on Computers, 2011.
[25] L. Lei, K. Cong, and F. Xie, "Optimizing post-silicon conformance checking," in ICCD, 2013.
[26] L. Lei, K. Cong, Z. Yang, and F. Xie, "Validating direct memory access interfaces with conformance checking," in ICCAD, 2014.
[27] K. Cong, F. Xie, and L. Lei, "Symbolic execution of virtual devices," in QSIC, 2013.
[28] H. Li, D. Tong, K. Huang, and X. Cheng, "Femu: A firmware-based emulation framework for soc verification," in CODES+ISSS, 2010.
[29] V. Srinivasan, F. Schirrmeister, V. Singh, and R. Klein, "Why hybrid platforms are needed for pre-silicon hardware and software development," in Electronic Design Process Symposium (EDPS), 2015.
[30] Synopsys, "Synopsys hybrid prototyping," http://www.synopsys.com/Prototyping/FPGABasedPrototyping/Pages/hybrid-prototyping.aspx.
[31] Cadence, "Cadence palladium hybrid," http://www.cadence.com/products/sd/palladium_hybrid/pages/default.aspx.
[32] E. Chung, E. Nurvitadhi, J. Hoe, B. Falsafi, and K. Mai, "PROToFLEX: FPGA-accelerated hybrid functional simulator," in IPDPS, 2007.