Comparative Study of Test Generation Methods for Simulation Accelerators

Wisam Kadry, Dimtry Krestyashyn,
Arkadiy Morgenshtein, Amir Nahir, Vitali Sokhin
IBM Research - Haifa, Israel

Jin Sung Park, Sung-Boem Park, Wookyeong Jeong,
Jae Cheol Son
Samsung Electronics, Korea

***Abstract*— Hardware-accelerated simulation platforms are quickly becoming a major vehicle for the functional verification of modern systems and processors. Accelerator platforms provide functional verification with valuable simulation cycles. Yet, the high cost and limited bandwidth of accelerator platforms dictate a requirement for continuous utilization improvement.**

In this work, we perform a comparative analysis of two approaches of test generation for accelerator platforms. An exerciser tool is used as experimental vehicle for the study. An off-platform test generation methodology is implemented and is compared to on-platform test generation typically used in exercisers.

We present experimental results from simulation of latest IBM POWER8 processor on Awan accelerator platform, as well as from simulation of an eight-core ARMv8-based design on Veloce emulation platform. Our results indicate that the utilization of accelerator platforms can be improved by up to ×7 ratio when using off-platform test generation. In addition, increase of up to 24% is observed in test coverage. Off-platform mode features significantly bigger image size, but maintains tolerable build and load times.

I. Introduction

Functional verification is a constantly increasing challenge in modern microprocessor's design, due to the high complexity of the computing systems. With the exponential design growth, sole usage of software-based simulators fails to provide sufficient level of coverage. Hardware-assisted simulation accelerators come to an aid, while providing significant simulation performance boost.

Simulation accelerators consist of large arrays of customized ASIC processors designed for concurrent simulation of basic logic components. The design under verification (DUV) is synthesized and mapped to the acceleration platform. Hardware-accelerated simulation platforms are quickly becoming major vehicles for the functional verification of modern systems and processors [1][2]. This trend is confirmed by the recent market studies indicating a 125% increase in the adoption of acceleration platforms for functional verification between 2007 and 2010 [4]. Accelerators play a key role in establishing a strong link between pre-silicon and post-silicon verification [3].

Functional verification methodology using simulation accelerators is dependent on the type of DUV. We distinguish between two types of accelerator-based verification, as shown in Fig. 1. When the DUV is an application-specific logic design without embedded general-purpose core, the test-cases are generated at a host machine and are delivered to the DUV through a dedicated input buffer. This type incorporates techniques such as transaction-based acceleration (TBA) [11] to cope with frequent host-accelerator communication. Alternatively, in designs such as system-on-chip (SOC) with an integrated processor, verification can be performed by using the embedded testbench approach [5]. In this method, the test-cases are generated by running dedicated software on the embedded processor. This allows for increasing the data transfer speed and eliminating the need for continuous communication with the host machine. In this work we focus on accelerator-based verification using the embedded testbench approach.

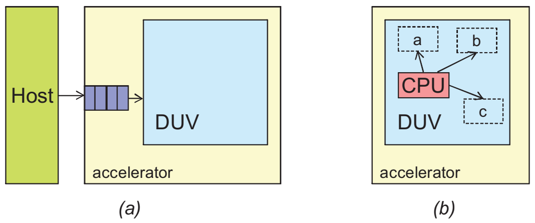

Fig. 1. Accelerator-based verification types: (a) tests generated by a host machine, (b) embedded testbench using a processor integrated in a system on a chip (SoC).

The high cost and limited capacity of accelerator platforms dictate a requirement for continuous utilization improvement. The accelerator platforms are typically shared by many teams and are used for running multiple test suites. Thus, each simulation run has a limited cycle budget in which maximal test coverage should be achieved.

Utilization of an accelerator can be defined as a number of test-case runs being accomplished during a budgeted time. The number of test-cases can be determined by timing components of accelerator flow, as in the following equation:

$$\tau = \frac{T_B}{n} \quad , \quad n = \frac{T_B - m \cdot \left(T_{up} + T_{init} + T_{down}\right)}{T_{gen} + T_{exec} + T_{check}} \qquad (1)$$

where τ is the number of cycles required to accomplish a single test-case (smaller value of τ represents higher platform utilization), n is the number of test-cases run during the budget time T_B, m is the number of interactions between the host and the accelerator platform, T_{up} is the test upload time,

T_{init} is the initialization time, and T_{down} is the data download time. Time required for accomplishing each test-case is represented here by test-case generation time T_{gen}, execution time T_{exec} and checking time T_{check}. Note that the processing at host machines (such as test preparation and results analysis) can be performed in parallel with the accelerator run and thus do not impact the accelerator utilization.

The performance bottleneck caused by host-accelerator communication can be addressed by the embedded testbench approach, which uses the integrated processor for generating the test-cases within the DUV. An implementation of such an approach was recently proposed, based on using post-silicon validation tools [2] on accelerator platforms.

Running post-silicon validation tools, called exercisers [6], on the hardware accelerators is known as a exerciser-on-accelerator (EoA) activity [2]. Such exercisers can effectively run on real silicon, on an accelerator, and on an instruction set simulator (ISS). The common flow of exercisers is to generate, run, and check test-cases on platform in an endless loop, without the need to reset and re-initialize the platform between test-cases. This significantly reduces data transfer between the host and the accelerator.

Post-silicon exercisers spend the majority of time on generation rather than on the actual execution of the test-cases. Despite the short time spent executing test-cases, on-platform generation was chosen as the preferred technique for the silicon platform, as when considering all overheads (initializing, load, and off-loading), this technique maximizes silicon utilization. When considering the differences in speed and scale between silicon and the accelerator, the tradeoffs change, giving rise to the off-platform generation technique proposed in this paper.

Accelerator capacity imposes a limit on the number of cycles we can simulate with it. Consequently, utilization is critical and has to be maximized. Thus, optimizing the flow by performing the test generation off-platform and dedicating the accelerator resources to test-case execution would be advantageous. In this work we analyze the benefits and tradeoffs in off-platform test generation for accelerators.

Contributions: We introduce a method for improving the utilization of acceleration platforms and apply it to a functional validation exerciser. Instead of the traditional on-platform test-case generation by the exercisers, we suggest generating test-cases using the same exerciser while running over an ISS platform. The generated test-cases are collected into a new executable image that can be loaded onto the acceleration platform. In the new image we maintain and run the same exerciser code except the generator part, thus saving much more cycles to execute the prepared test-cases. We demonstrate the effectiveness of the proposed method by implementing it in the Threadmill post-silicon exerciser. We performed experiments on the Awan accelerator platform used for verification of the IBM POWER8 processor, as well as on Veloce emulation platform used for simulation of an eight-core ARMv8-based design. In the comparative analysis of off-platform and on-platform test generation, we present the advantages as well as the trade-offs of the proposed technique.

II. Methodology

In this work, we used the IBM Threadmill post-silicon exerciser [6] to evaluate the effectiveness of on-platform vs. off-platform test generation for accelerator-based verification. The main input to Threadmill is a test-template that specifies the desired scenarios. Other inputs to Threadmill include the architectural model and testing knowledge and the system topology.

The Threadmill execution process starts with a builder application that runs on the host to create an executable exerciser image. The builder converts the data incorporated in the test-template and the architectural model into data structures that are then embedded into the exerciser image. The proposed methods for on-platform and off-platform test generation feature different structures and content of the executable image. In the next sections, we describe the detailed flows and the differences between the two techniques.

i. On-platform Generation Mode

The exerciser image in on-platform generation mode (shown in Fig. 2, on the left) has three major components. The first component is an OS-like layer of basic services required for Threadmill's bare-metal execution. These services included, for example, dedicated interrupt handlers for building virtual-to-physical translation paths and a simple implementation of the *critical section* function. The second component is comprised of data structures representing the test-template, architectural model, and system configuration description. The third component is a fixed (test-template independent) code that is responsible for stimuli generation, test-case execution, and checking.

In on-platform generation mode, the image contained code for the generation of a random test-case based on the test-template, the configuration, and the architectural model. After the image was loaded onto the accelerator/silicon platform, the exerciser indefinitely repeated the process of generating a test-case, executing it, and checking its results. Note that in on-platform generation mode, no test-case data was included in the image.

ii. Off-platform Generation Mode

Threadmill can run both on silicon and on pre-silicon simulation platforms, such as a hardware-assisted accelerator and a software *reference model* [8], a.k.a. instruction set simulator (ISS). The ISS software can run on a variety of machines and operating systems, and at a higher speed than the accelerator, making it a highly available and low-cost platform. In this work, we propose to use the ISS as a basis for the off-platform generation of test-cases that will be later incorporated into an exerciser image and executed on an accelerator platform.

The proposed flow of off-platform test generation is depicted in Fig. 2. The main idea of the proposed methodology is to eliminate the need for test-case generation on the accelerator platform. This was achieved by modifying the exerciser image. The modified image included test-cases that were generated a-priori by executing a regular exerciser image on the ISS.

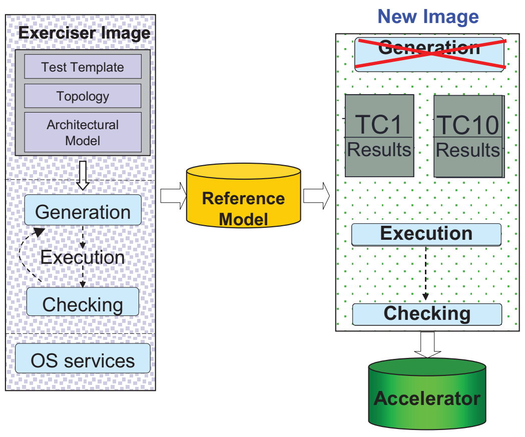

Fig. 2. Test image creation in off-platform generation mode.

At the first stage, we created a regular exerciser image, as in the case of the on-platform generation, with the generator component enabled. The image included empty data structures that were later used for storing the test-cases, memory initializations, translation tables, and expected results. Next, we ran the exerciser image on the ISS. The run included generation, execution and checking of the test-cases. While running on the ISS, we collected the required data, including the generated test-cases' code, resource initializations and translation tables. We also saved the expected results from the run on the ISS.

The harvested data was then incorporated into a new image and was stored in the previously allocated data structures. Furthermore, the generator component was disabled in the new image. The structure of the modified exerciser image is shown in Fig. 2, on the right. Finally, the modified image was loaded onto the acceleration platform. The exerciser executed the stored test-cases, without the overhead associated with test-case generation and initializations, and compared the results with the ones harvested from the ISS run.

III. RESULTS

We compared the effectiveness of on- and off-platform generation in several aspects. We performed our experiments by using the Threadmill exerciser on two different simulation platforms and processor designs. Most of the experiments were performed using the Awan simulation accelerator platform. The DUV model was a processor core model of IBM's POWER8 processor. The off-platform test generation was done using the MAMBO instruction set simulator [8]. In addition, in order to confirm the effectiveness of the proposed methodology on different platforms, we performed experiments on an eight-core ARMv8-based design. For this design, the off-platform test generation was done using Fast Models instruction set simulator [12]. These tests were executed using the Mentor Veloce emulation platform.

A. Platform Utilization

To analyze the platform utilization improvement, we used two typical test templates, a random instruction stream and an irritation scenario [6], and measured the speedup obtained by off-platform generation.

We performed profiling of both test types to assess the generation component, as shown in Table 1. As can be seen, generation is responsible for 71% of the runtime in the random test and 81% in the case of the irritation test. The difference is attributed to higher generation complexity of the irritation test.

Two images were created for each test template. The first image was of the original exerciser image that performs generation on-platform. The second image contained 100 test-cases that were generated off-platform. We measured the number of cycles required for run of a single test-case. The speedup was calculated as the ratio of run times of a single test-case in each mode. The results are presented in Table 2, showing the number of cycles required for the run of a single test-case in both modes. As can be seen, off-platform generation provides ×3.7 speedup for random tests and ×5 speedup for irritation tests requiring more complex generation. The improvement rates of platform utilization are in full correlation with the proportion of the generation component measured during profiling.

Similar experiments were performed for a ARMv8-based design on Veloce accelerator platform. The platform utilization improvement was assessed by profiling the stages of test flow, as shown in Table 3. As can be seen, for both tests, the platform utilization ratio by execution component was increased by more than ×7. These results were also validated by measurement of number of cycles required for a single test-case run on Veloce platform, showing up to ×6.9 improvement for random and irritation templates, respectively.

The image in off-platform mode features bigger size, which is explained by including the test-cases, initializations and expected results in the image. For each thread per test-case, an additional 50KB of memory are required to hold the off-platform-generated data. The upload time increases from 1 to 11 seconds, but the change is justifiable as compared to the overall running time, which is typically several hours.

TABLE 1. PROFILING DATA OF ON-PLATFORM MODE (POWER8 DESIGN ON AWAN PLATFORM).

	Run time %	
Function	**Random Test**	**Irritation Test**
Generator	71	81
Execution	2	3
Test-time barriers	15	8
Page map	9	0
Compare results	0	2
Other	3	6

TABLE 2. PLATFORM UTILIZATION IMPROVEMENT (POWER8 DESIGN ON AWAN PLATFORM) – CYCLES REQUIRED PER TEST-CASE.

Template	**On-platform Generation**	**Off-platform Generation**	**Speedup**
Random	4.8M	1.3M	×3.7
Irritation	7M	1.4M	×5

TABLE 3. PLATFORM UTILIZATION IMPROVEMENT FOR RANDOM TEST (ARMV8-BASED DESIGN ON VELOCE PLATFORM).

	Run time %			
Function	**Random test**		**Irritation test**	
	On-platform	Off-platform	On-platform	Off-platform
Generator	89.7	20.0	90.4	16.1
Execution	3.5	26.1	3.6	32.0
Test-time barriers	0.3	0.6	1.1	8.1
Page map	0.2	2.2	0.1	0.5
Compare results	6.4	51.1	4.9	43.3

B. Coverage Analysis

To evaluate the effectiveness of off-platform generation mode in terms of test coverage, we leveraged coverage monitors synthesized into the DUV model [2]. The provided coverage data includes the count of activations (*hits*) of each event and the *tags* allowing mapping the events by the functionality. The coverage events during an accelerator run may be triggered by execution of the exerciser code and not only by the test-case instructions. To eliminate the influence of exerciser code on coverage analysis, we selected test templates targeting new processor functions that are not used by the exerciser code. In this manner, we were able to selectively analyze the events related to the tested functionality by using the specific tags.

Three test templates were used as benchmarks for the comparative coverage study. The first test template was used for verification of the branch history rolling buffer (BHRB). Two additional templates were used for testing of the transactional memory facility of the IBM POWER8 processor [9]. For each template, an image including 100 random test-cases was created using off-platform generation. An additional image was created for on-platform generation. Both images were run on the accelerator for equal time periods. For each run, coverage data was gathered, including the number of relevant events that were covered, and the total number of times these events were hit during the runtime.

We observed significant increases in the total number of hits in all covered events. The overall number of hits increased by the order of ×2.73 to ×6.06 in off-platform generation mode. The number of covered events also improved in the off-platform mode by up to 24%. Fig. 3 presents the detailed coverage profile of transactional memory test, showing the number of hits of each coverage event is both modes. A higher number of hits can be observed for most of the covered events in the off-platform generation mode. As shown, 52 events out of 267 were covered only in off-platform generation mode. Moreover, most of these events were hit multiple times.

IV. SUMMARY AND FUTURE WORK

We introduced a method for improving the utilization of acceleration platforms and applying it to a functional validation exerciser. Instead of the traditional on-platform test-case generation by the exercisers, we suggest to generate the test-cases using the same exerciser while running over an ISS platform.

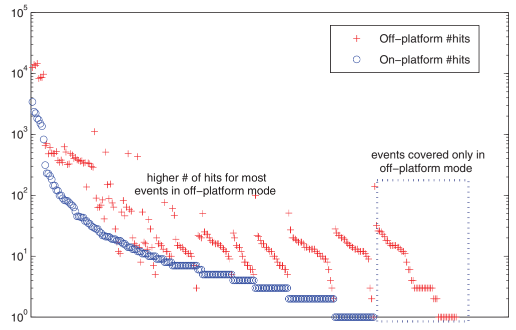

Fig. 3. Hits profile of coverage events of TM test.

We performed a comparative analysis of the two techniques, using the processor core models of IBM's POWER8 processor on the Awan accelerator platform, as well as an ARMv8-based eight-core processor on the Veloce emulation platform. Our results indicate that the utilization of an accelerator platform can be improved by up to ×7 when using off-platform test generation. In addition, test coverage increased by up to 24%. The off-platform mode features significantly bigger image size, but maintains tolerable build and load times.

Bare-metal exercisers typically use simple test generation mechanisms that do not use ISS data. Future research may target the development of off-platform techniques for accelerator-dedicated test generation. Such techniques will allow for the leveraging of the ISS data to create higher-quality tests while maintaining maximal accelerator utilization

REFERENCES

[1] J. A. Darringer et al., "*EDA in IBM: past, present, and future*," IEEE Trans. on CAD of Integrated Circuits and Systems, vol. 19, no. 12, pp. 1476-1497, 2000.

[2] A. Nahir et al., "Post-Silicon Validation of the IBM POWER8 Processor," in DAC, pp. 1-6, 2014.

[3] A. Nahir et al., "Bridging pre-silicon verification and post-silicon validation," in DAC, pp. 94-95, 2010.

[4] H. Foster, "From volume to velocity: The transforming landscape in function verification," in DVCon, 2011.

[5] R. Goering, "*Designer View: Embedded Palladium Testbench Speeds System Bring-Up*", 2013.

[6] A. Adir et al., "Threadmill: a post-silicon exerciser for multi-threaded processors," in DAC, pp. 860-865, 2011.

[7] J. M. Ludden et al., "*Advances in simultaneous multithreading test-case generation methods,*" in HVC, pp. 146-160, 2010.

[8] P. Bohrer et al., "*Mambo: a full system simulator for the Powerpc architecture*," SIGMETRICS Perform. Eval. Rev., 31(4):8-12, 2004.

[9] H.W. Cain et al., "Robust architectural support for transactional memory in the power architecture", in ISCA, pp. 225-236, 2013.

[10] M. D. Moffitt et al., "Scalable scheduling for hardware-accelerated functional verification," in ICAPS, 2011.

[11] S. Matalon et al., "*Building transaction-based acceleration regression environment using plan-driven verification approach*", 2007.

[12] ARM Fast Models, http://www.arm.com/products/tools/models/fast-models/