REGULAR PAPERS

Property-driven functional verification technique for high-speed vision system-on-chip processor

Japanese Journal of Applied Physics **56**, 04CF15 (2017)
https://doi.org/10.7567/JJAP.56.04CF15

REGULAR PAPER

Property-driven functional verification technique for high-speed vision system-on-chip processor

Victor Nshunguyimfura, Jie Yang, Liyuan Liu, and Nanjian Wu*

State Key Laboratory for Superlattices and Microstructures, Institute of Semiconductors, Chinese Academy of Sciences, University of Chinese Academy of Sciences, Beijing 100083, P. R. China

*E-mail: nanjian@red.semi.ac.cn

Received October 13, 2016; revised January 8, 2017; accepted January 10, 2017; published online March 28, 2017

The implementation of functional verification in a fast, reliable, and effective manner is a challenging task in a vision chip verification process. The main reason for this challenge is the stepwise nature of existing functional verification techniques. This vision chip verification complexity is also related to the fact that in most vision chip design cycles, extensive efforts are focused on how to optimize chip metrics such as performance, power, and area. Design functional verification is not explicitly considered at an earlier stage at which the most sound decisions are made. In this paper, we propose a semi-automatic property-driven verification technique. The implementation of all verification components is based on design properties. We introduce a low-dimension property space between the specification space and the implementation space. The aim of this technique is to speed up the verification process for high-performance parallel processing vision chips. Our experimentation results show that the proposed technique can effectively improve the verification effort up to 20% for the complex vision chip design while reducing the simulation and debugging overheads. © 2017 The Japan Society of Applied Physics

1. Introduction

Today's high-integration and -performance demands make the vision chip design more complex. A vision chip integrates image sensors with multilevel heterogeneous parallel processors on a single chip and performs real-time image processing.[1–6] This type of chips are found in a wide range of critical application domains, such as video and image processing,[7] defect detection, robot vision, and control systems. Our device under verification (DUV) is a heterogeneous parallel processor for real-time vision applications. As shown in Fig. 1, this high-speed vision chip is based on multilevel parallel processors. The chip mainly consists of a control processor, functional modules, and four computational clusters. Each cluster consists of four processing cores. The architecture provides two possible ways of collecting data. Image data are captured from an image sensor to the buffer or collected from a computer to the external input/output (I/O) memory through the peripheral component interconnect express (PCIe) interface. The collected image data are then distributed to different local memories for further operations depending on the program stored in the program memory. The chip is designed to support three kinds of parallel processing, namely, single instruction multiple data (SIMD) for parallel pixel processing, multiple instruction multiple data (MIMD) for parallel image slice processing, and single program multiple data (SPMD) for parallel image patch processing. This parallel processing technique speeds up vision operations and achieves high throughput for various kinds of vision processing operations from low-level processing, feature vector building, to classification. This high integration has significantly increased the system verification complexity.

A variety of verification techniques such as pre- and post-silicon verification solution techniques have been suggested[8,9] on the basis of where they intervene in a chip life cycle. In this work, we focus only on the pre-silicon techniques as they are extensively used in the early stages of vision chip design and industry. Static methods and dynamic methods[10–12] are commonly applied in the verification field. A trade-off exists between these two methods.

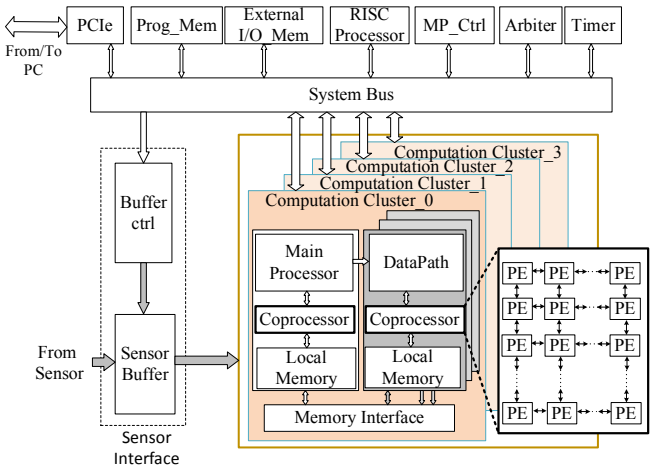

Fig. 1. (Color online) Simplified architecture of the high-speed vision chip.

Static verification methods yield a comprehensive and complete verification but have capacity constraints, i.e., they are limited to small functional blocks. On the other hand, dynamic methods are characterized by the simulation of the device under verification. These simulation-based methods involve stimuli generation,[13–19] coverage recording,[20–22] and bug detection techniques.[23,24] Compared with static methods, dynamic methods have no capacity limitations but suffer from completeness problems. They cannot provide a complete verification solution. Although different techniques were reported to improve the verification effort,[25,26] the chip functional verification task is still facing different challenges. A number of problems such as verification delay remain unresolved. The debugging task consumes more time than any other activity during the verification cycle. The number of verification engineers keeps increasing at 3.5 times the rate of increase in number of designer engineers.[27] Traditional verification approaches become inefficient to verify such complex system. These approaches still consume tremendous efforts during the design development. Measuring verification progress and determining the end of the verification task remains poorly covered.

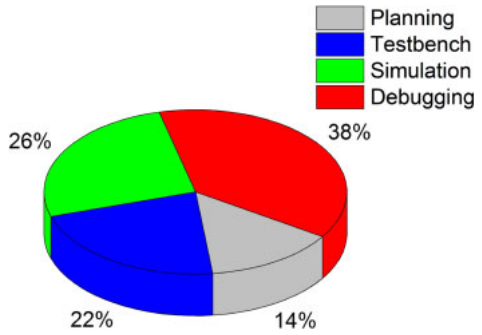

Fig. 2. (Color online) Effort allocated at different stages in common verification cycle.

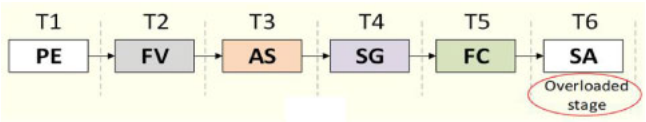

Fig. 3. (Color online) Stages involved in vision chip verification cycle: Stepwise approach commonly found in most existing verification flow.

In this paper, we propose a property-driven approach that results in a practical and efficient verification environment. Compared with the existing verification methods, the proposed approach decreases both the verification complexity and debugging time, while shortening the verification cycle.

2. Vision processor verification architecture

The functional verification process is the most critical element during the vision chip development. It requires a significant amount of effort and resources. This process goes through many iterations starting from the specification stage to the debugging stage. Figure 2 shows the average time spent at each stage in a common verification process.[28] The last two stages are the most overloaded. From this point, we noted that the verification effort is not well distributed all along the verification flow. There are two major causes of this behavior. The first is the intrinsic stepwise nature of the process. The task in one stage starts after the task in the previous stage is accomplished. This high inter-stage dependence induces stress propagation, which causes high stress accumulation at the simulation and debugging levels.

The second cause is the big gap existing between the specification and the verification implementation. This gap is likely due to the fact that the verification task is considered later in the vision chip design life cycle, which pushes the verification team to focus mainly on the last two stages of the process. This gap increases the verification complexity and presents a challenge for the verification scheduling during the planning process. In order to reduce the verification burden considerably, we analyzed where we spend most of the time during the functional verification cycle. We group specific verification tasks into the corresponding logical stages, as shown in Fig. 3. The following are the six main stages of our vision chip verification flow: property extraction (PE), formal verification (FV), assertions (AS), stimulus generation (SG), functional Coverage (FC), and simulation and analysis tasks (SA).

In our proposed methodology, each stage corresponds to one verification component, and specific techniques are used for its implementation. These techniques are used to minimize the overhead existing in the last stage (SA stage) of the verification flow. Each component is independently implemented considering the extracted property. In this concept, the quality of the component in each stage is increased and the deliverable time from one stage to the next stage is minimized. In the following section we explain in detail the task accomplished in each stage and the respective techniques used for its practical implementation.

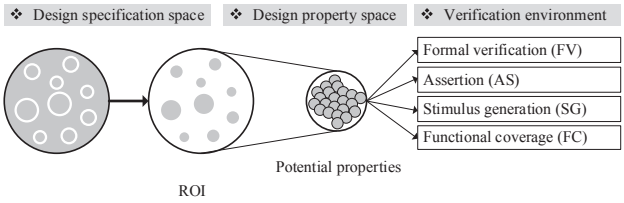

Fig. 4. Mapping the design specification space into the intermediate property space.

2.1 Property extraction

In pre-silicon verification we distinguish different types of faults that cause system failures. These bugs are obviously related to erroneous design transformations at different abstraction levels. Design specifications are normally provided in informal language, incomplete in itself most of the time. A straight translation from specifications to the design implementation increases verification ambiguities and this is frequently encountered during the functional verification task.

To reduce these kinds of ambiguities, these informal descriptions need to be converted into a formal description prior to the implementation. Another major concern in the real-world verification flow is the turnaround behavior of functional verification. The verification task follows several abstraction steps, at which the result of a certain level is used as the specification of the following lower level. In some cases, the constraints at a given abstraction level are not met as expected; hence, the previous step has to be reiterated with different parameters.

As a solution to these problems, we introduce the property space into the verification flow, between the specification space and the verification implementation, as shown in Fig. 4. The intermediate space consists of design intents extracted independently from a limited region of interest (ROI) within a design specification. These properties are used later in the verification process to monitor the verification progress.

By mapping the design behaviors into property space, we create a common root for different subsequent abstraction steps. All remaining tasks such as FV, AS, SG, and FC are developed on the basis of these potential properties. This procedure can effectively decrease the wasted time in reiterating different steps.

A block diagram illustrated in Fig. 5 is a simplified forwarding circuit. This circuit is a small part of the design under verification and it is designed to solve data hazards. Circuit A uses its outputs to control circuit B. Let us take two design intents extracted from a given design specification. We know that the control signals Ctrl1 and Ctrl2 should only take the value 2'b00, 2'b01, or 2'b10. We also know that the

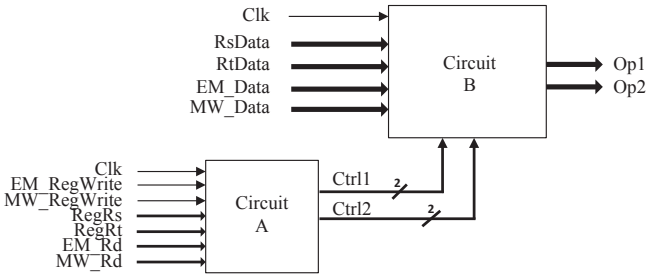

Fig. 5. Simplified forwarding circuit form the design under verification.

control signal Ctrl1 is equal to 2'b01 only if EM_RegWrite is one, EM_Rd is not equal to zero, and EM_Rd is equal to RegRs.

(1) From the above requirements, we extracted the following four properties.

Property_01: The control signal ctrl1 should never be equal to 2'b11.

Property_02: If the control signal ctrl1 == 2'b01, then EM_Rd must be any value other than 0.

Property_03: The control signal ctrl1 == 2'b01 when two conditions are satisfied:

EM_RegWrite == 1'b1 and EM_Rd == RegRs.

Property_04: Avoid unknown output control signals.

(2) Once a list of properties is already elaborated, we then translate these four properties in a formal language. We formally describe certain desired and undesired behaviors of the simplified forwarding circuit. Figure 6 shows the formal description of these four properties.

These properties express the temporal relationship among Boolean expressions, which correspond to the temporal logic. We focus on the temporal properties,[29] safety properties, liveness properties, fairness properties, and non determinism properties throughout our property extraction. In this case, we use systemverilog assertions (SVA)[30,31] to describe both the legal and illegal behaviors of the design. In order to reduce the debugging exposure, we break the complex properties into multiple simple properties.

2.2 Formal verification

Our target from the beginning is to prove that the implementation behaves as stated in specification for all inputs and all time instances. The simulation is usually used to discover design errors. To realize full confidence that our DUV is bug-free, we would have to perform an exhaustive simulation that covers all circuit states and all possible input sequences. The fact that the number of necessary input combinations increases exponentially for the vision chip design, this approach becomes impractical and potential bugs remain undiscovered.

As a viable alternative to reduce the simulation overheads, we adopt mathematical reasoning[32–37] that exhaustively prove or disprove the consistency between the design specification and the design implementation. We concisely describe the design property in the formal specification. This formal specification plays a crucial role at the first stage of the whole verification process. We use this formal approach to increase the abstract level of the system view while focusing on what the design should do instead of how it can be implemented. The mathematical description enables discussion between different teams involved in the design project to remove the ambiguities related to the incompleteness nature of the design specification.

2.3 Assertions

There are two basic techniques that ensure that a design under verification is bug free. In one technique, we use correctness checking principles to avoid the introduction of errors during the design cycle. This approach requires manual guidance and considerably more computation effort. Hence, they are not suitable for large designs. In the other technique, faults are detected and eliminated at a certain abstraction level during the design phase. Today, simulation is the usual validation method to discover design errors. From the simulation outputs, it is possible to judge whether the DUV behaves erroneously or not.

Relying on simulation is inefficient for verifying a complex vision system chip in two aspects. On one hand, using the simulation technique, the debugging task takes much more time to detect and localize the bug. Identifying the source of bugs in terms of time and space requires much more time and efforts as well. On the other hand, to improve the design quality in terms of area, speed, or power consumption, optimization techniques are often used. For this point of concern, additional simulation and debugging time are inevitable.

The assertions are good candidates for solving these challenges related to the bug tracking process because they are most effective for the design observability and controllability during the verification process.[31,38] We adopt the assertion-based verification (ABV) methodology[25,39,40] to improve verification quality and allow our verification environment to be self-checking. In addition to these advantages, assertions are used to improve the debugging effort in terms of bug detection and bug avoidance. The assertion statements used throughout our verification process are grouped into three categories, namely, block level, interface level, and system level assertions. For the verification task to be highly systematic, we introduce two essential concepts. Primarily, we build assertions on the basis of design properties productively to enhance the analysis stage. We identify the commonly encountered properties to be asserted; they are named, parameterized, and grouped in library units for further reuse. We propose a non exhaustive list of these properties: valid parameters, conditional expression, mutual exclusive signals, valid signals,

```
/*----------------------Design properties in the formal language---*/
  ctrl1 != 2'b11;
 (ctrl1 == 2'b01) |->  EM_Rd !=0 ;
 (ctrl1 == 2'b01) |-> (EM_RegWrite == 1'b1) && ( EM_Rd == RegRs);
  $isunknown({ctrl1,ctrl2}) != 1'b1 ;
/*-------------------------------------------------------------------*/
```

Fig. 6. (Color online) Formal expression of the design properties extracted from the design specification.

invalid signals, event-bounded window, priority order, valid transaction, valid transition, incorrect dependencies, sequence implication, and timing relationships.

Secondly, the assertion control mechanism is used to control the verification environment. The declarative nature of SVA provides a way to embed the specific information for each assertion, such as a message to be displayed in case of assertion failure. We assign the severity level to each assertion so that we can control the simulation in real time. This mechanism enables system verification based on different levels of granularity.

2.4 Stimulus generation

Generating high-quality verification patterns becomes a critical task for the vision chip simulation owing to the increasing design complexity. The general practice to achieve this task requires a very large amount of test data, which is a labor-intensive and very time consuming task.[16,18] Different techniques such as coverage-driven verification methodology have been developed and become most widely used in the field of large-scale chip verification processes. These techniques mainly focus on the functional coverage of the design under verification.

The drawback of these approaches is that the DUV is required for the testbench simulation to determine how far functional coverage is achieved. In this context, the RTL model should be available. This behavior proves to be problematic for complex vision chip design and impedes the development of the verification components parallel to the design implementation. This results in a need of new verification methodologies that enable the verification team to efficiently explore the entire stimulus space within the constrained time. On the basis of all the scenarios defined inside the property space, we propose a novel mechanism that efficiently generates high-quality stimuli. The proposed stimulus generation algorithm combines the advantages of coverage-driven techniques with the constraint random techniques to enhance the flexibility and effort of generating good verification patterns.

In the proposed algorithm, the first step is to map the design properties to the corresponding stimulus bins so that we can effectively monitor the real-time stimulus coverage. Once the mapping is completed, we initialize the random stimulus referring to the extracted properties.

As shown in Fig. 7, we simulate the model to generate the test cases while optimizing progressively the constraints until we realize that no more optimization space is available. At this point we need to insert the directed stimulus into the process to reach hard-to-generate stimuli for the corner cases.

During each simulation cycle, we analyze the patterns coverage results. Once we find that holes exist in our results, we need to fix all of them so that the maximum stimulus coverage can be reached. Two types of holes exist in the process, one includes holes related to the incompleteness of constraints or some possible violations that exist among constraints. We call this type of holes "reachable holes". We fix this type of holes by optimizing our constraints. The other type includes the design corner cases. They require much effort to be fixed. Directed stimuli[10] are developed to cover this kind of holes. After exploring the entire stimulus space, we save all stimuli as our verification patterns for the simulation stage. The main goal of this approach is to efficiently cover a wide range of stimuli in a faster and accurate manner so that we can reduce the wasted time in generating and simulating useless patterns.

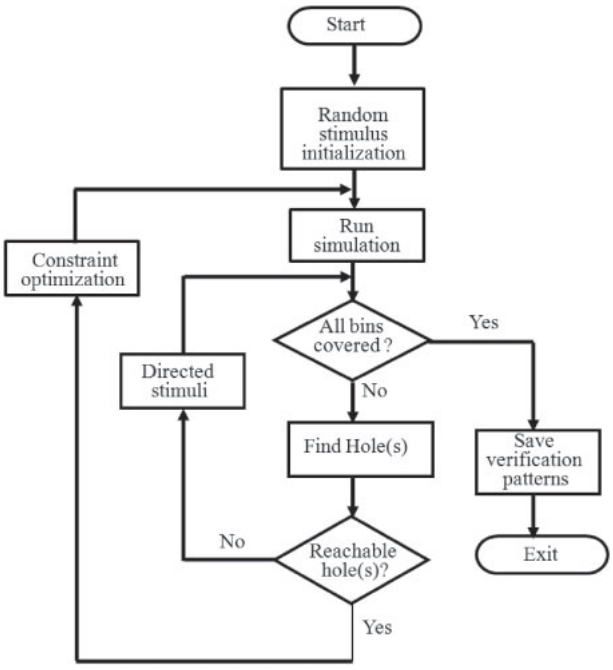

Fig. 7. High-quality stimulus generation algorithm for high-speed vision chip.

2.5 Functional coverage

Another major issue we face during the vision chip verification cycle is completeness. We need to measure how far we have covered the property space and verified all the defined functions. We introduce three types of coverage in our verification process.

Stimulus coverage: This type of coverage metric is used as an evaluation criterion to judge whether or not the set of defined scenarios has been sufficiently generated.

Assertion coverage: This kind of coverage is developed to serve as a quality and capacity metric both at a block level and at the interface level. Assertion coverage is implemented in our approach to determine how far the chip has been verified. At the analysis stage where we monitor all chip activities, we check if any property violation occurs from any particular assertion code.

Design functional coverage: This coverage is used to ensure the completeness of the verification task, i.e., we need to cover all the scenarios that are supposed to be verified. We use the design functional coverage to enhance the advantage of the assertions in checking the run-time behavior of the design.

2.6 Simulation and analysis

This last step is the most overloaded stage in most of existing verification methodologies. We analyze the main reasons of the overheads at the simulation and analysis stages that cause the high time consumption. Different causes were enumerated including inefficient generation of stimuli, which leads to the simulation of useless verification patterns, the back tracking behavior of the techniques used to detect bugs and find root cause of the error, and the inefficiency of the verification control mechanism. Once the simulation is launched, the verification engineer needs to wait until the simulation is

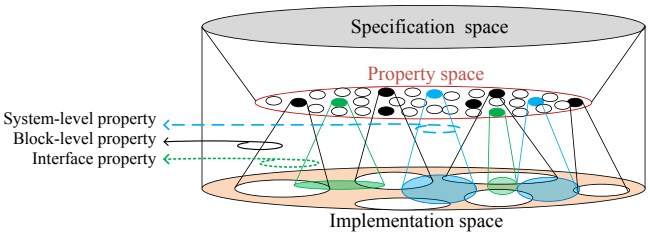

Fig. 8. (Color online) Mapping the design specification space into the intermediate property space.

finished to check the report, and some design properties are not practical to be analyzed on the basis of the waveform generated during the simulation time. All the above-mentioned causes are inherent in the previous verification tasks and propagated to this last stage. The significant part of our work deals with the anterior verification tasks to tackle these challenges efficiently. At this stage, we run simulations to detect the bug and track their sources and location. The analysis is conducted once the bug is detected to make sure that the fault alert is caused by the circuit bug. Once the bug is confirmed, the report is generated for the design debugging task.

3. Verification implementation

In general, the functional verification effort is not well distributed to prove the vision chip correctness. To the contrary, it is mostly oriented to the implementation of the verification environment. By introducing the low-dimension property space into the verification cycle, we can reduce the verification gap problem encountered in verification flow. We extract the potential properties from the design specification space to generate an intermediate property space, as shown in Fig. 8. The property space includes the ROI, the block-level, interface, and system-level properties. All the tasks in each verification stage depend on the properties defined in this intermediate space. This technique breaks down the stepwise behavior of the process to decrease the dependence among stages, yet improves the verification environment flexibility and scalability. We can add more properties as long as the design functionality increases without changing the existing environment.

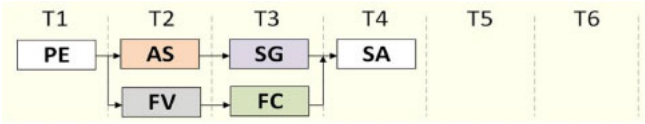

Fig. 9. (Color online) Structure of the proposed verification flow.

4. Experimental results and analysis

Figure 9 shows how to increase the design verifiability and verification productivity. We combined AS with FV in one stage (T2) and SG with FC in the following stage (T3). Using this parallel technique we were able to decrease the number of stages to four. At the same time, we were able to shorten the verification cycles.

We integrate all verification components in one single verification environment as shown in Fig. 10. We elaborated the verification specification document, which covers all the requirements of a chip under consideration. After all requirements are analyzed, we start the planning process. The total design cost becomes considerably higher and inacceptable when the bug is found at the post-silicon level. Therefore, an effective plan for the pre-silicon verification is necessary at the earlier stage to serve as a roadmap for all verification activities to be performed. We describe the scope of the verification problem and define what must be verified, how the verification task will be conducted, and when each component will be verified.

The main part of this environment is the design property block. The role of this block in the environment is very significant. The first advantage is that it explicitly resolves the design ambiguities before moving to the next verification stage. The second advantage is the improvement in coverage metrics. From Fig. 11 we can see how effective the stimulus generation becomes. Using the ROI defined inside the property block, we can improve the quality of the verification patterns while shortening the simulation cycle. The third advantage is the fact that minimizing the dependences among

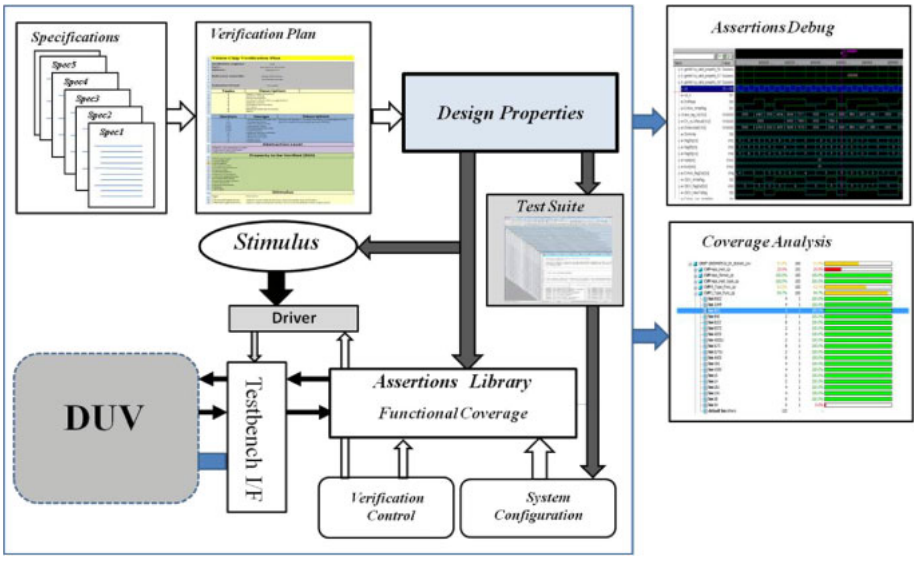

Fig. 10. (Color online) Proposed verification environment for the vision chip.

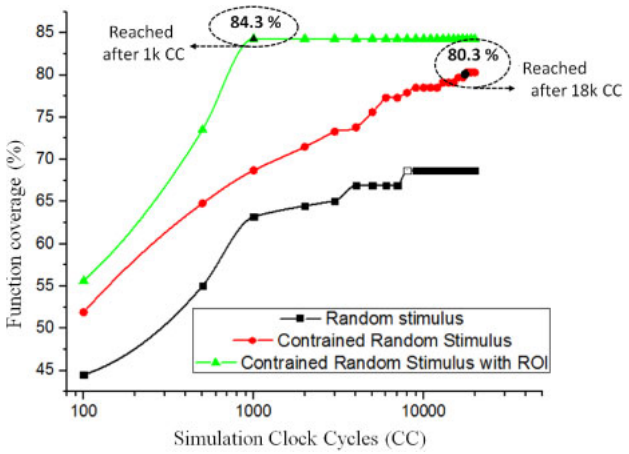

Fig. 11. (Color online) Result showing the improvement in stimulus generation for the vision chip based on ROI. (Directed stimulus not included in the graph).

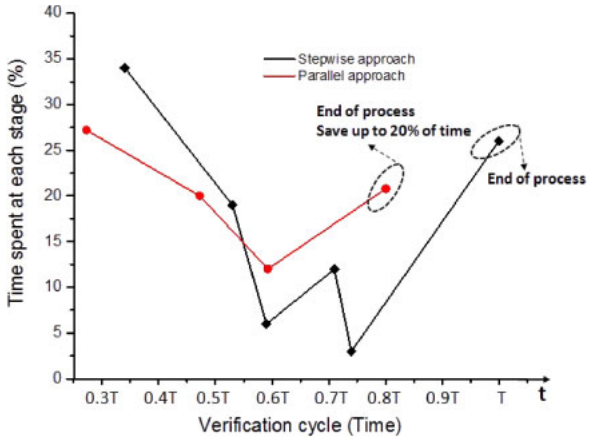

Fig. 12. (Color online) Result before (black line) and after (red line) reducing the interstage dependence in vision chip verification cycle.

verification tasks leads to the parallel implementation of the verification components. Introducing the low-dimension space and the parallel technique improves considerably the verification effort. The deliverable time from one task to starting another task is minimized. As shown in Fig. 12, by using our proposed approach, we can effectively overcome the overheads existing at the simulation and analysis stages during the common verification cycle. In this parallel approach, our verification effort is not only concentrated at the two last stages but distributed to other tasks especially at the PE stage. Hence, a 20% reduction in overall verification time is achieved. Converting the extracted properties into Systemverilog assertions and coverage parameters has improved considerably the debugging effort. The proposed procedure can be applied to other vision chips as they shared the common features. The proposed stimulus generation algorithm is suitable for verifying heterogeneous systems with high parallel computation capability. It can be applied to configurable systems that use the reduced instruction set as well.

5. Conclusions

In this paper, we propose a property-driven functional verification approach. The proposed technique provides an effective pre-silicon verification solution to the challenges encountered in the vision SoC verification. We introduce a new property stage into the verification flow to reduce the gap existing between the specification space and the implementation space. All verification components are designed on the basis of the property space. With this technique we break down the stepwise nature of the existing functional verification approaches. We also show that reducing the existing interdependence between two consecutive verification stages increases considerably the design simulation and debugging effort. The proposed technique is independent of any tools and can be compatible with other existing verification methodologies. On the basis of SystemVerilog constructs, we have implemented a high-performance verification environment. The results show that the verification effort is improved by up to 20% at different abstraction levels during the verification process.

Acknowledgment

This work was supported by Brain Project of Beijing (Grant No. Z161100000216129).

1) C. Shi, J. Yang, Y. Han, Z. Cao, Q. Qin, L. Liu, N. Wu, and Z. Wang, IEEE J. Solid-State Circuits **49**, 2067 (2014).
2) W. Zhang, Q. Fu, and N. Wu, IEEE J. Solid-State Circuits **46**, 2132 (2011).
3) Q. Lin, W. Miao, W. Zhang, Q. Fu, and N. Wu, J. Sens. **9**, 5933 (2009).
4) T. Komuro, S. Kagami, and M. Ishikawa, IEEE J. Solid-State Circuits **39**, 265 (2004).
5) M. Yasuda and M. Watanabe, Int. Conf. Field-Programmable Logic and its Application, 2010, p. 508.
6) Y. Watanabe, T. Komuro, and M. Ishikawa, IEEE Int. Conf. Image Processing (ICIP), 2007, p. 177.
7) T. Isshiki, H. Xiao, H.-C. Liao, D. Li, and H. Kunieda, Proc. Int. SoC Design Conf. (ISOCC), 2012, p. 611.
8) S. Mitra, S. A. Seshia, and N. Nicolici, Proc. Design Automation Conf. (ACM), 2010, p. 12.
9) P. Rashinkar, P. Paterson, and L. Singh, *System-on-a-Chip Verification: Methodology and Techniques* (Springer, New York, 2002) p. 153.
10) J. Bergeron, *Writing Testbenches: Functional Verification of HDL Models* (Springer, New York, 2000) pp. 7, 69.
11) T. Kropf, *Introduction to Formal Hardware Verification* (Springer, Berlin, 2013) p. 12.
12) A. Piziali, *Functional Verification Coverage Measurement and Analysis* (Springer, New York, 2008) p. 17.
13) R. Dechter, K. Kask, E. Bin, and R. Emek, Proc. Natl. Conf. Artificial Intelligence (AAAI02), 2002, p. 15.
14) A. Chandra, V. Iyengar, D. Jameson, R. Jawalekar, I. Nair, B. Rosen, M. Mullen, J. Yoon, R. Armoni, D. Geist, and Y. Wolfsthal, IEEE Trans. VLSI **3**, 188 (1995).
15) K. Shimizu and D. L. Dill, Proc. Design Automation Conf., 2002, p. 801.
16) J. Yuan, A. Aziz, C. Pixley, and K. Albin, IEEE Trans. Comput. Aided Design IC Syst. **23**, 412 (2004).
17) N. Kitchen and A. Kuehlmann, Proc. IEEE/ACM Int. Conf. Computer-Aided Design (ICCAD), 2007, p. 258.
18) L. Piccolboni and G. Pravadelli, IEEE Latin-American Test Workshop (LATW), 2014, p. 1.
19) R. A. DeMilli and A. J. Offutt, IEEE Trans. Software Eng. **17**, 900 (1991).
20) V. Subedha and S. Sridhar, Eur. J. Sci. Res. **81**, 533 (2012).
21) S. Yang, R. Wille, D. Große, and R. Drechler, Euromicro Conf. Digital System Design, 2012, p. 525.
22) S. Tasiran and K. Keutzer, IEEE Des. Test Comput. **18** [4], 36 (2001).
23) W. K. Lam, *Hardware Design Verification: Simulation and Formal Method-Based Approaches* (Prentice-Hall, Upper Saddle River, NJ, 2005) p. 287.
24) P. Wilcox, *Professional Verification: A Guide to Advanced Functional Verification* (Springer, New York, 2004) p. 31.
25) V. Nshunguyimfura, J. Yang, L. Liu, and N. Wu, Int. Conf. ASIC (ASICON), 2015, p. 1.
26) D. Bustan, D. Korchemny, E. Seligman, and J. Yang, IEEE Des. Test

Comput. **29** [2], 23 (2012).
27) W. Rhines, Design and Verification Conf. (DVCon), 2016, p. 1.
28) H. Foster, IEEE Design Automation Conf., 2015, p. 1.
29) R. Armoni, L. Fix, A. Flaisher, R. Gerth, B. Ginsburg, T. Kanza, A. Landver, S. Mador-Haim, E. Singerman, A. Tiemeyer, M. Vardi, and Y. Zbar, *The ForSpec Temporal Logic: A New Temporal Property-Specification Language* (Springer, Berlin, 2002) p. 296.
30) A. B. Mehta, *SystemVerilog Assertions and Functional Coverage* (Springer, New York, 2014) p. 251.
31) S. Vijayaraghavan and R. Meyyappan, *A Practical Guide for SystemVerilog Assertions* (Springer, New York, 2005) Part II, Chap. 3.
32) R. Drechsler, *Advanced Formal Verification* (Springer, New York, 2004) p. 81.
33) D. Kroening and N. Sharygina, Proc. IEEE Int. Conf. Formal Methods and Models for Co-design, 2005, p. 101.
34) T. Kropf, *Introduction to Formal Hardware Verification* (Springer, Berlin, 1999) Chap. 2.
35) L. Fix, *Fifteen Years of Formal Property Verification in Intel* (Springer, Berlin, 2008) p. 139.
36) D. Tabakov, K. Y. Rozier, and M. Y. Vardi, Formal Methods Syst. Des. **41**, 236 (2012).
37) G. Rozenberg and A. Salomaa, *Handbook of Formal Languages* (Springer, New York, 1997) Vol. 3, p. 254.
38) S. Takeuchi, K. Hamaguchi, and T. Kashiwabara, IPSJ Trans. Syst. LSI Des. Methodol. **2**, 80 (2009).
39) J. Bergeron, E. Cerny, A. Hunter, and A. Nightingale, *Verification Methodology Manual for SystemVerilog* (Springer, New York, 2006) Chap. 3.
40) E. Cerny, S. Dudani, J. Havlicek, and D. Korchemny, *The Power of Assertions in SystemVerilog* (Springer, New York, 2010) p. 229.