Hindawi Publishing Corporation
VLSI Design
Volume 2016, Article ID 7283471, 14 pages
http://dx.doi.org/10.1155/2016/7283471

Research Article

Unified and Modular Modeling and Functional Verification Framework of Real-Time Image Signal Processors

Abhishek Jain[1,2] and Richa Gupta[1]

[1]*ECE Department, Jaypee Institute of Information Technology, Noida 201309, India*
[2]*Systems & Platforms Solutions, STMicroelectronics, Greater Noida 201308, India*

Correspondence should be addressed to Abhishek Jain; abhishek-mmc.jain@st.com

Received 31 December 2015; Revised 31 May 2016; Accepted 31 July 2016

Academic Editor: Mohamed Masmoudi

In VLSI industry, image signal processing algorithms are developed and evaluated using software models before implementation of RTL and firmware. After the finalization of the algorithm, software models are used as a golden reference model for the image signal processor (ISP) RTL and firmware development. In this paper, we are describing the unified and modular modeling framework of image signal processing algorithms used for different applications such as ISP algorithms development, reference for hardware (HW) implementation, reference for firmware (FW) implementation, and bit-true certification. The universal verification methodology- (UVM-) based functional verification framework of image signal processors using software reference models is described. Further, IP-XACT based tools for automatic generation of functional verification environment files and model map files are described. The proposed framework is developed both with host interface and with core using virtual register interface (VRI) approach. This modeling and functional verification framework is used in real-time image signal processing applications including cellphone, smart cameras, and image compression. The main motivation behind this work is to propose the best efficient, reusable, and automated framework for modeling and verification of image signal processor (ISP) designs. The proposed framework shows better results and significant improvement is observed in product verification time, verification cost, and quality of the designs.

1. Introduction

Image signal processor (ISP) mainly corrects the output images of the sensor so that the best possible defects and noise-free images can be generated. It processes the stream of image data into a form which can be easily managed by upstream mobile baseband or multimedia processor chipsets [1, 2]. Different markets, including gaming, smartphones, surveillance, and medical and automotive applications, are mainly covered by the ISP. The integration of image processors has become a simple process by the use of industry standard interfaces and a rich set of application programming interfaces (APIs). It also helps to lower time to market of end-product.

Restoration engine (RE) of the ISP is responsible for removing the noise and the artifacts and generating the RGB data from the Bayer image. Color engine (CE) of ISP is responsible for scaling the image as per the output requirements, sharpening the image, and converting the RGB data to YUV.

Universal verification methodology (UVM) is a generic methodology for the functional verification of hardware designs, mainly using simulation. The hardware design which is to be verified can be described using VHDL, Verilog, SystemVerilog, or SystemC at any appropriate abstraction level. This can be register transfer level or behavioral or gate level. Assertion-based verification and hardware emulation or acceleration can also be used along with the universal verification methodology [2, 3].

A SystemVerilog UVM test bench consists of reusable verification components. A verification component is a configurable, encapsulated, ready-to-use verification environment for a design submodule, an interface protocol, or a full system [4, 5]. Each verification component follows a standard architecture for stimulus generation, data/protocol checking, and obtaining coverage information for a specific

design or protocol. The verification environment is applied to the designs to verify implementation of the protocol or design architecture [6].

In our case, development and evaluation of image signal processing algorithms are done using Python software models before implementation of RTL and firmware. After the finalization of the algorithm, Python models are used as a golden reference model for the development of ISP RTL and firmware. Then, Python reference models are used in universal verification methodology based verification framework of ISP RTL for its bit-true verification. This proposed framework shows better results and significant improvement is observed in product verification time (~ 50% improvement), verification cost (~20% reduction), and quality of the designs (~75–80% improvement).

This paper is organized in different sections. It presents the ISP development flow that we have developed going through algorithm investigation, algorithm development, ISP RTL implementation, and its functional verification using UVM-based verification environment. Section 2 discusses the earlier functional verification frameworks. Section 3 describes the ISP development flow and modeling framework. Section 4 discusses the proposed functional verification framework. Section 5 discusses methodology of early development of verification framework. Section 6 discusses IP-XACT flow for automatic generation of verification environment files and map files. Section 7 discusses results of experiments done by us and Section 8 concludes by listing the performance and the advantages of the proposed framework.

2. Related Work

Marconi et al. proposed a verification framework based on SystemVerilog and universal verification methodology (UVM) for high energy physics (HEP) applications [7]. This verification framework described the reusability and high flexibility of verification components. The framework helped verification engineers in verification of various DUTs and architectures at different abstraction levels.

Liang et al. proposed a SystemVerilog UVM-based methodology for mixed-signal level verification of wireless power receiver family MCU [8]. Usage of the universal verification methodology- (UVM-) based verification environment with analog design enhanced the verification quality and efficiency.

Kim et al. proposed a FPGA-based verification methodology for the image signal processor (ISP) of CMOS image sensor [9]. A four-step verification methodology composed of ARM core based verification, system verification, algorithm verification, and performance verification was used.

Kannavara proposed the idea of a validation framework [10]. Main issues were mentioned which need to be resolved for success of such validation framework.

The studied papers were describing the different frameworks for verification of image signal processors and other applications. Earlier frameworks were not taking into account the flexibility in the verification environment to verify designs using both host interface and actual core in a unified manner. Modeling part of ISP designs was also not described. In our approach, we are describing unified and modular modeling and functional verification framework of real-time image signal processors. Common infrastructure is proposed for all the IP/subsystem/SoC level verification environment. The proposed framework provides flexibility to use verification components in different languages (SV, SC, "e," etc.). The proposed framework is developed both with host interface and with core using virtual register interface (VRI) approach. The methodology of early development of verification environment before arrival of RTL is developed. IP-XACT based tools are developed for automatic configuration and development of the verification environment for various IPs/SoCs.

3. ISP Development Flow

ISP development flow is clearly understood from Figures 1 and 2. At the start, the Python model is written for evaluation of the algorithms. At that stage, there is no hardware or firmware partitioning done in the model. After evaluation, this model is used as a reference model for the algorithmic ISP RTL. After the evaluation stage, hardware and firmware partitioning is done in reference model. An implementation model is developed where both hardware and firmware parts of the algorithm exist. Now the firmware part of this model is used by the firmware developer to verify the firmware drivers and the hardware part of Python reference model is used at unit level for bit-true verification of the ISP RTL.

In the initial stage of verification of the ISP IP RTL, directed test scenarios are driven from UVM-based verification environment. Once the ISP IP RTL becomes stable for few directed scenarios, random test data is generated from verification environment. Generated random test data is driven to both ISP IP RTL and software reference model. Comparison between the output of ISP IP RTL and software reference model is done for its bit-true verification. After initial verification of the ISP IP RTL by running few directed and random test scenario simulations, light regression is run using regression tool so as to make sure that 60–70% of the ISP IP RTL has been verified and can be used for ISP HW system integration. Then, full regression at ISP IP level is run to complete verification of the ISP IP RTL in parallel with initial verification of the ISP HW at subsystem/SoC level. Functional coverage is also measured to know whether all possible functional test scenarios are covered or not.

At ISP level, ISP block diagram represents ISP hardware specifications. Verified ISP IPs are integrated to make ISP HW system and ISP reference model be used for the verification. The ISP is then continued to be tuned for more improvements and better performance.

Mapping between configuration and status registers of ISP RTL and attributes of reference model is provided through a Python map file with the following:

(i) A function which translates HW inputs (parameters, register values, memory contents, and images) to model attributes

(ii) A function which translates model output attributes to HW outputs (register values, memory contents, and images)

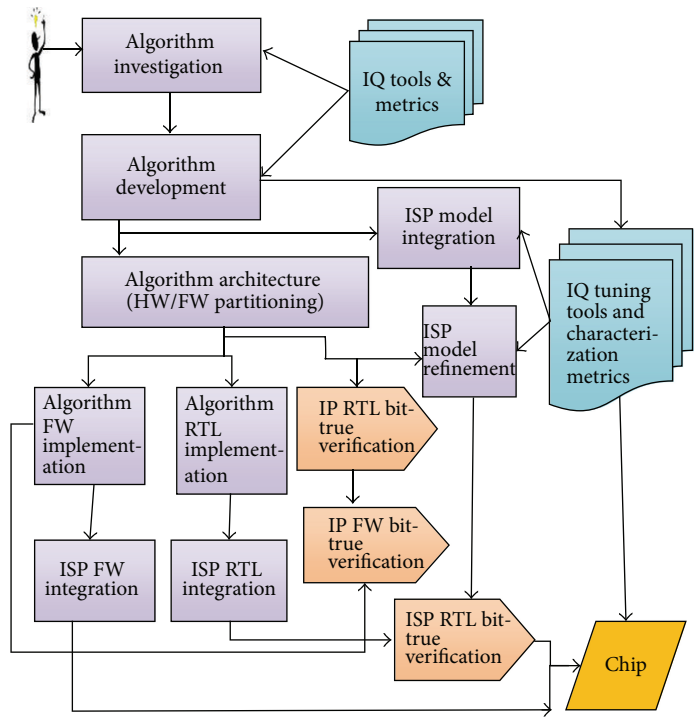

Figure 1: Algorithm development methodology from idea to product.

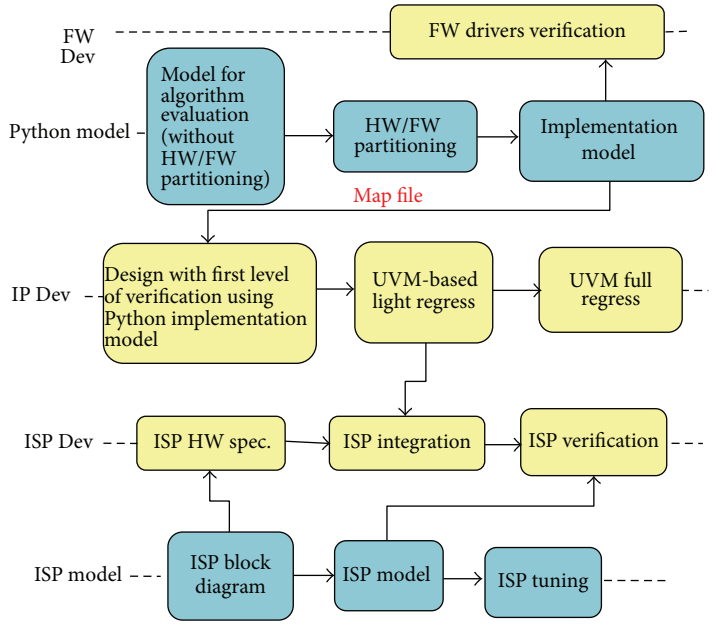

Figure 2: IP and ISP development flow.

(iii) Python mapping file that describes how to transfer hardware inputs to model inputs and model outputs to hardware outputs. The mapping file does not describe how hardware inputs have to be provided or how hardware outputs have to be processed. Python script file provides those data and processes the results

It is clear from Figure 3 that the map file contains both the RTL-to-model and model-to-RTL mapping. The most basic requirement to write a map file is to have the register description file and instantiation parameters description file. These files are in standard XML format, so that they are easily converted to Python map file via IP-XACT flow.

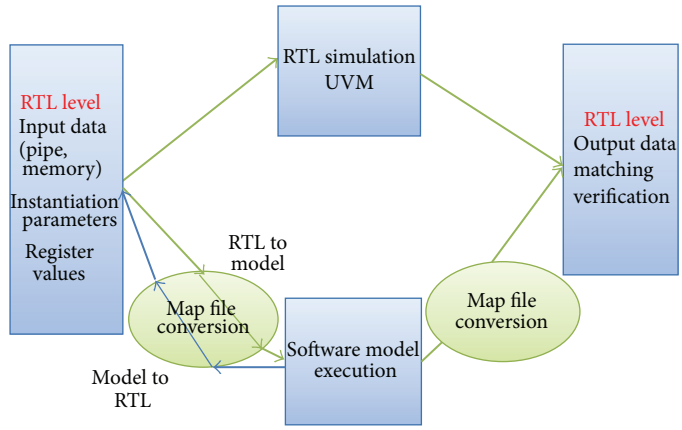

Figure 3: Bit-true verification overview.

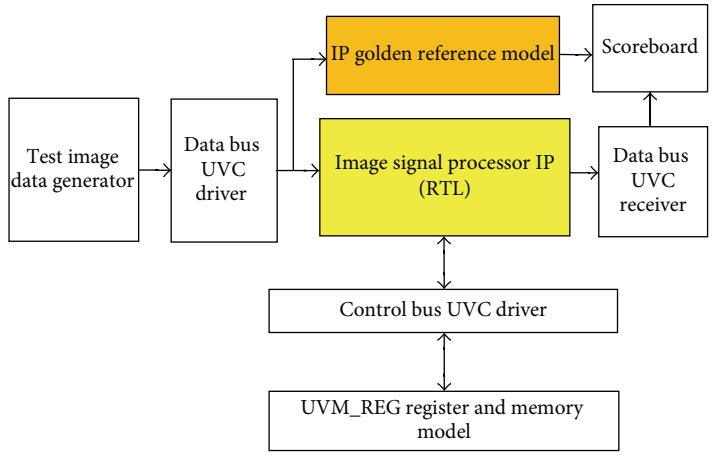

Figure 4: Image signal processor RTL verification environment.

The development of unified and modular modeling framework of image signal processing algorithms provides the following benefits:

(i) Reduction of the development time

(ii) Coherent modeling approach making development and support not reserved to algorithms experts

(iii) Maximum reuse of IP

(iv) Bit-true certification between models and products

(v) Algorithms/ISP developers, HW designers, FW engineers, HW verification engineers, and application and marketing people using the same modeling framework

4. Functional Verification

ISP RTL is verified using UVM-based verification environment as shown in Figure 4. Coverage-driven verification (CDV) feature of universal verification methodology (UVM) is used to generate both user-defined and random input image test data for the ISP IP RTL [4, 11]. Self-checking mechanism in test bench is performed by comparison of output of software reference model and ISP IP RTL. Coverage metrics are measured to know whether all desired test scenarios are covered or not.

Registers and memories of ISP DUT are modeled by using UVM_REG register and memory model [5, 12]. UVM_REG register and memory model integrated with control bus UVC accesses ISP DUT registers and memories using host interface. Control bus UVC acts as initiator and ISP RTL control bus interface acts as target. Control bus UVC drives the target control bus of the ISP RTL. After programming of ISP RTL registers and/or memories, data bus UVC drives the user-defined/random test image data to the data bus interface. The same test image data is also sent to the reference model.

Data bus UVC monitor receives the output data of the ISP RTL. The output data of ISP RTL and reference model is compared by the scoreboard and it gives the result saying whether both outputs (image/status data) match or not.

Register sequencer executes ISP RTL register operation sequences according to the register model specification and

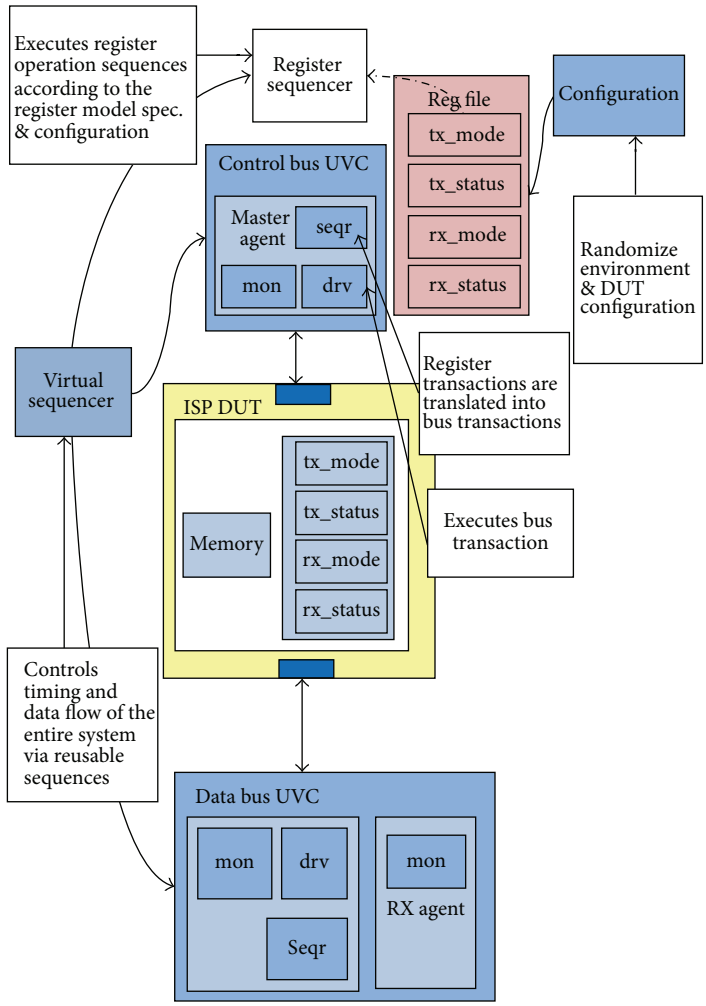

Figure 5: Synchronization between register sequences and data sequences.

configuration. Similarly, data bus sequencer also executes data operation sequences according to the data bus specification and configuration. Virtual sequencer controls timing and data flow of the entire system via reusable sequences [13, 14].

UVM_REG API is developed to write simpler directed tests which require less or no SystemVerilog/UVM understanding. Our own register/memory sequences are also developed to address the SoC level register and memory testing [12].

Since register or data sequences, sequencers, and drivers are focused on point interfaces, ISP RTL verification environment has a virtual sequence to coordinate the stimulus across register and data interfaces and the interactions between them [13, 15]. Virtual sequence controls the register and data sequences as shown in Figure 5.

When host BFM is replaced with actual core, then it becomes challenging to reuse the existing verification environment as "C" test code is used to do functional verification with core in place. At SoC level, it is important to verify that the hardware and software work seamlessly together to deliver the functionality and performance of the image signal processor.

As shown in Figure 6, virtual register interface (VRI) layer is a virtual layer over verification components to make it configurable from embedded software. It also gives high level C-APIs and masks low level implementation details from the users. A VRI layer is developed over verification components to configure the SystemVerilog UVM-based data bus UVC and other verification components from the "C" test cases [16]. VRI layer provides the access of the sequences of these verification components to the embedded software [17]. It enables the configuration and control of these verification environments with actual core and performs the same exhaustive functional verification at SoC level [2].

VRI approach covers the following aspects of ISP functional verification at SoC level:

(i) Embedded software configures ISP verification components.

(ii) ISP verification environments are reused from IP level to SoC level.

(iii) ISP test cases are reused from IP level to SoC level.

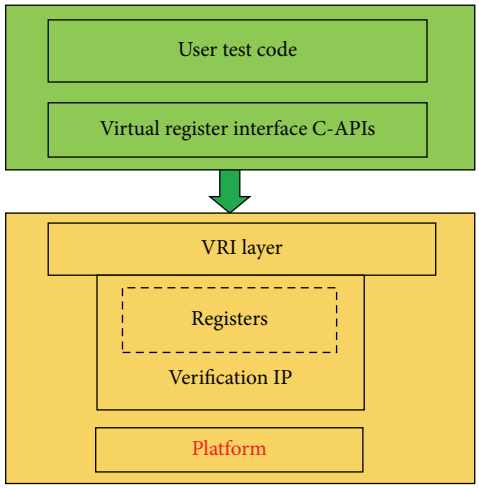

Figure 6: Virtual register interface (VRI).

(iv) ISP integration test cases created by IP verification teams are used by SoC team.

The main benefit of the VRI approach is the reuse of verification IP (VIP) and "C" test code. The same VIP and "C" test cases are reused from IP level to SoC level verification.

Figure 7 shows SoC level functional verification environment where using virtual sequences and ISP SoC DUT is simultaneously bombarded at all interfaces in an automated and coordinated way [18, 19]. Virtual sequence is controlling the stimulus generation using several sequencers of host bus UVC, USB UVC, Ethernet UVC, and so forth [20]. Here, programming of ISP SoC registers is done using control bus UVC and not the actual core. All test cases are written in SystemVerilog language.

Figure 8 shows SoC level functional verification environment where test software mixed with host BFM. VRI APIs are converted to host bus UVC's transactions.

The basic interface between embedded processors and internal IPs (e.g., peripheral devices) in an ISP SoC is a set of control and status registers [16]. These registers that are usually located in the memory space of the system (memory mapped) are part of the ISP IP implementation. These registers represent features of the ISP IP itself. Similarly, any external verification IP like Ethernet VIP used for the verification of an ISP SoC is interfaced to an embedded processor through a set of control and status registers in the same way we do with hardware ISP IPs.

VRI exposes the functionality of the VIPs (e.g., VIP configuration or VIP sequences) in the form of a register map suitable to be controlled by an embedded processor. Using VRI layer, verification environment users can write both "C" and SystemVerilog test cases to control the same VIP. Software data randomization can also be done using VRI layer.

Figure 9 shows ISP SoC level functional verification environment where test software mixed with bare metal software. ISP DUT processor acts like virtual sequencer. The core is executing the "C" test cases along with virtual register interface API.

An example of "C" test case for Ethernet VIP using virtual register interface is described below:

```
vri_ethernet_packet tx_pkt;
vri_ethernet_packet rx_pkt;
rx_pkt.data = new vri_uint8_t[4000];
tx_pkt.packet_kind = ETHERNET_802_3;
tx_pkt.data_length = 0; //Data length 0 means sending
random number of data
tx_pkt.src_address_high = 0x2000;
tx_pkt.dest_address_high = 0x1000;
tx_pkt.error_code = 0;
for (int j = 0; j < 200; j++) {

    tx_pkt.src_address_low = j + 1;
    tx_pkt.dest_address_low = j;
    ethernet_send_packet(1, &tx_pkt); //send packet
    (MAC)
    ethernet_receive_packet(0,&rx_pkt);     //receive
    packet (PHY)
    compare_packet(tx_pkt,rx_pkt);

};
```

The main advantages of using VRI layer in the proposed framework are as follows:

(1) Using VRI layer, verification framework users can write C test cases to control the SystemVerilog/e VIPs. This gives flexibility to verification framework users to use the VIPs without knowing SystemVerilog/e.

(2) Through VRI layer, SW data randomization can also be done.

(3) Verification framework users can write both C and SystemVerilog test cases to control the same VIP.

(4) VRI layer gives verification framework users the flexibility to use the same test cases at IP/SoC level functional verification as well as for validation.

Static Formal Verification. Static formal verification is used to compliment the dynamic Metric Driven Verification (MDV) methodology in ISP verification. Static formal verification methodology provides a substitute for some of the verification tasks usually done under dynamic simulation. Static formal verification reduces the regression reruns to achieve coverage goals and to reduce the effort to write additional test scenarios.

Assertions are embedded within the ISP design code and they are written both within and outside the design code. Both designers and verifiers are using it independently. The ISP verification flow that includes the usage of assertions is shown in Figure 10.

Assertions are used in verification environments of image signal processor to

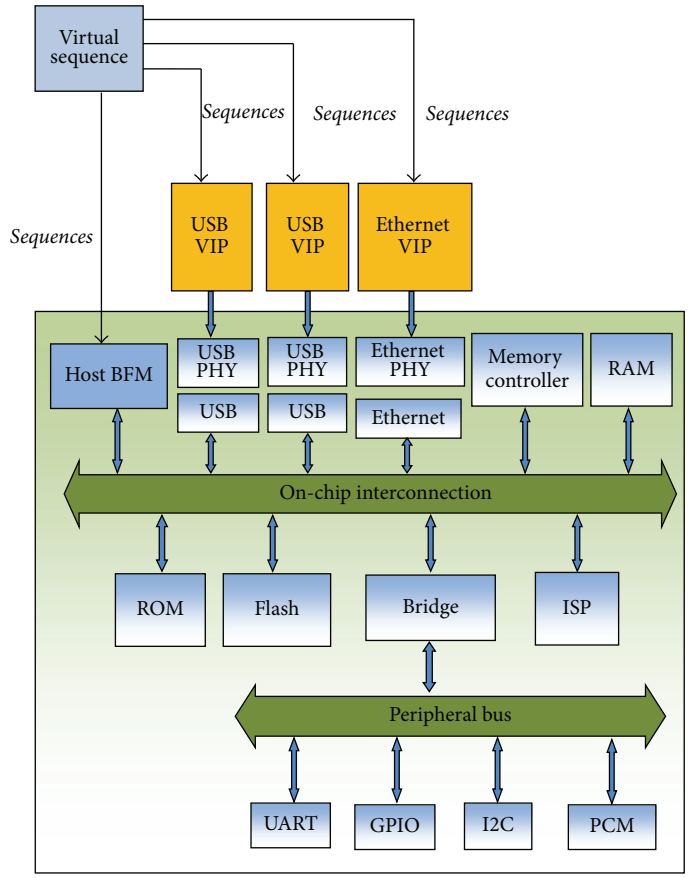

Figure 7: ISP SoC level functional verification environment using UVM.

(i) monitor signals on an interface that connects different blocks;

(ii) track the expected behavior of a logic gate, flip-flop, or module;

(iii) watch for forbidden behavior within a design block.

Some of the examples of usage of assertion-based VIP (ABVIP) in ISP verification are described as follows.

(1) Protocol Compliance Checking. As described in Figure 11, a top-level ISP module contains a Master IP and an ABVIP monitor. Master behavior of IP is checked with ABVIP master properties. Slave behavior is assumed with ABVIP slave properties.

(2) As a Driver (Constraints) to Verify Other Functionalities. As described in Figure 12, ABVIP drives defined, constrained random bus traffic to ISP DUT following interface bus protocol. Thus, ABVIP offload this time consuming task from dynamic simulation and verify functionality of the ISP RTL. Here, ISP RTL is acting as a receiver. Interface bus protocol behavior is assumed with ABVIP interface bus properties.

(3) Protocol Conversion (Bridge) Verification. As described in Figure 13, ABVIP is used for verification of protocol conversion (bridge) in ISP SoC RTL. Formal verification is a very efficient way for verification of bridges. Using only dynamic simulations for verification of bridges cannot efficiently complete the full verification of bridges as some test scenarios may miss in dynamic simulations.

(4) As Protocol Monitor in Simulation. As described in Figure 14, ABVIP is used in dynamic simulations to monitor the bus protocol. ABVIP also provides functional coverage of the properties covered.

5. Early Development of Verification Framework

For early development of UVM-based verification framework of image signal processing designs, TLM/SystemC reference model of ISP RTL is created from software reference model [21, 22]. Regressive verification of the TLM/SystemC model with user-defined/random test image data is done. After regression, the model is used as behavioral replica of ISP DUT. This speeds up development and better validation of the verification framework with wider test image data without waiting for ISP RTL to be available.

Standard "interfaces" are used for reusing the verification framework components. In addition to the standard method of signals level connectivity, UVM Multilanguage (UVM-ML) Open Architecture is used to connect SystemVerilog

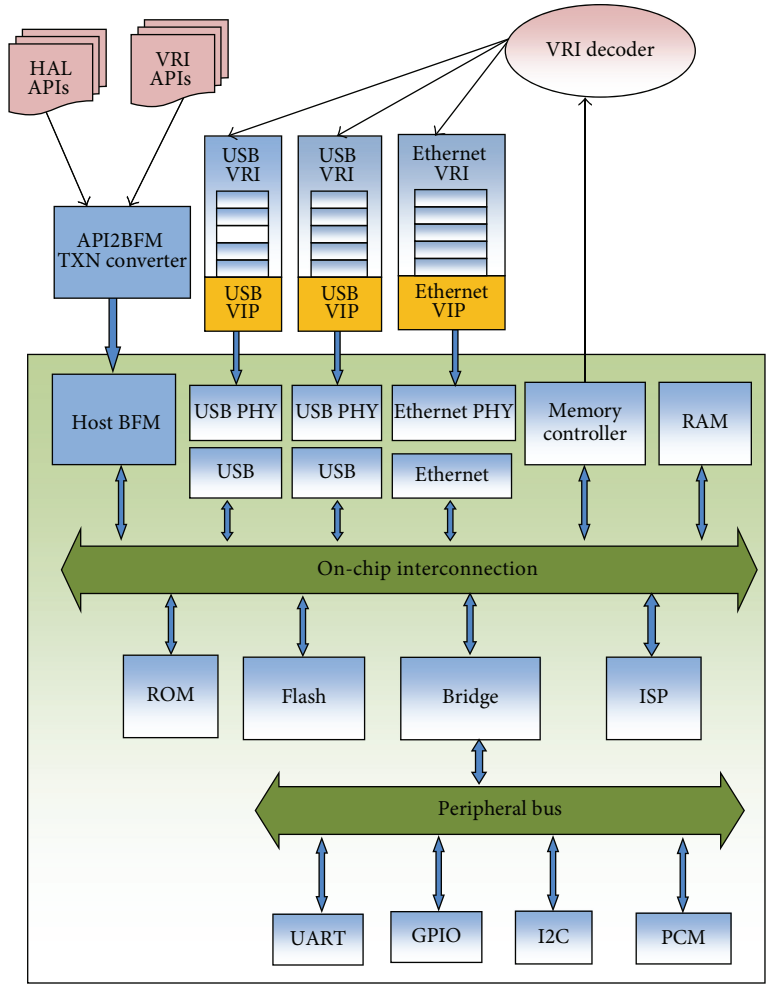

Figure 8: Test software mixed with host BFM.

TLM port directly to SystemC TLM port. This gives benefit of better simulation speed and better development cycle in addition to clean and easy integration of blocks [1, 22]. Usage of TLM components gives flexibility to make backdoor direct access to the ISP RTL registers and memories.

As shown in Figure 15, a TLM/SystemC model of processor is used for early development of "C" test cases for configuration of ISP RTL registers/memories via CPU interface [23]. The same "C" test cases are used for configuring the SystemVerilog UVM-based data bus VIPs using virtual register interface layer. In ISP verification environment, alternative host interface path is used to perform configuration of registers/memories using SystemVerilog UVM-based test cases. In both cases, image data and control flow across both bus interface and TLM boundaries [24]. This method improves the possibility of reusing already existing verification components in the verification flow.

Complete verification framework and image test data are ready with sufficient quality before the arrival of ISP RTL, thus eliminating the number of verification framework problems which may arise when actual ISP RTL verification is started. When ISP RTL arrives, the TLM/SystemC model is replaced with ISP RTL block with reuse of maximum of other verification components. This enables the regress testing of ISP design immediately. Also, the same C test cases are run on the actual core.

6. IP-XACT Flow

In UVM-based verification framework of image signal processor, register definition file for UVM_REG register model, top-level address map file, register and data sequences file, data checker (scoreboard) file to compare the output of ISP RTL with output of Python reference model, and functional coverage file are ISP IP/SoC specific. These files are modified for every ISP IP/SoC. IP-XACT based tools are developed for generation of these verification environment files [25]. First, the register map description has to be provided in XML format.

Description of registers and memories of ISP RTL in XML format is automatically generated from the register

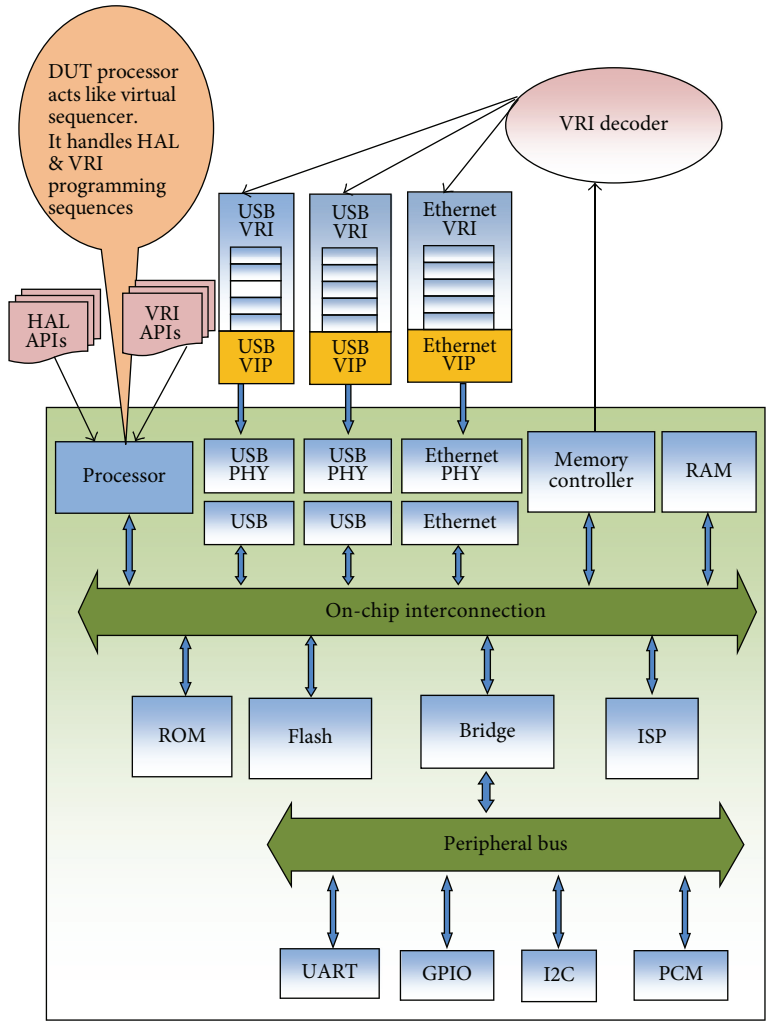

Figure 9: ISP functional verification environment using virtual register interface (VRI).

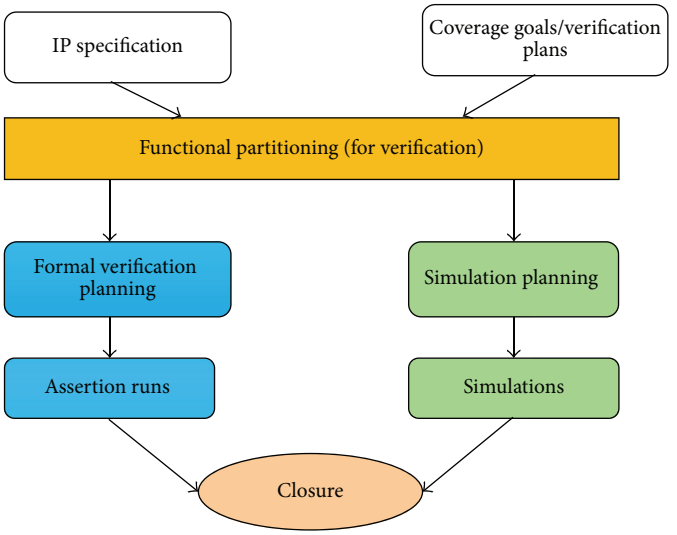

Figure 10: ISP verification flow including usage of assertions.

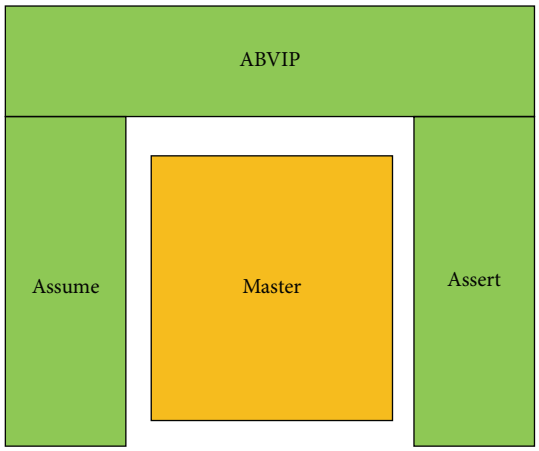

Figure 11: Protocol compliance checking.

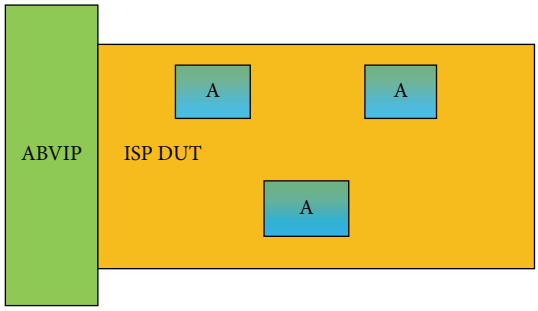

Figure 12: ABVIP as a driver (constraints).

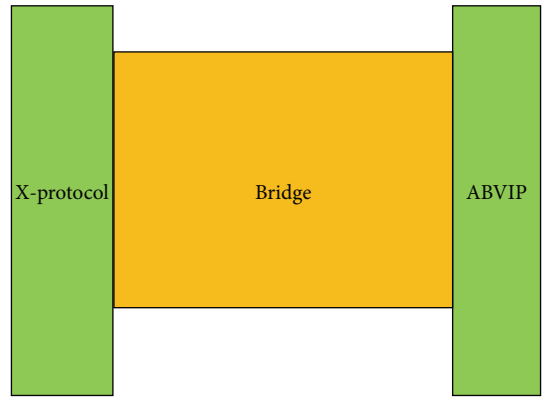

Figure 13: ABVIP for protocol conversion (bridge) verification in ISP SoC RTL.

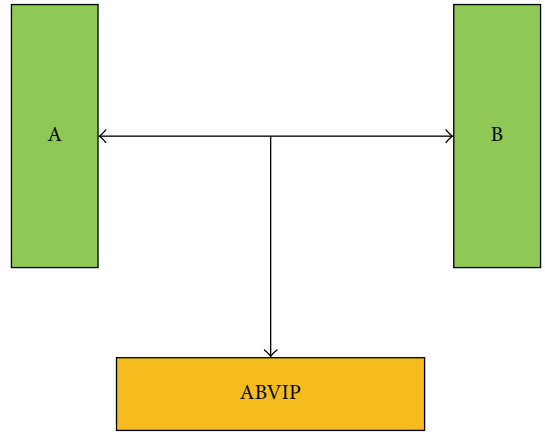

Figure 14: ABVIP as protocol monitor.

specification document using spec2spirit (specification-to-SPIRIT) script as shown in Figure 16.

In data checker (scoreboard) file, Python reference model which contains attributes is executed using system command. Thus, automatic generation of data checker file requires the mapping between the registers/register-fields/parameters of RTL and the attributes of Python reference model. In spirit2uvm (SPIRIT-to-UVM environment files) script, there are two input files:

(i) XML file for register map information which contains the registers and memories description of ISP RTL

(ii) Data and control interface information file

IP-XACT based spirit2uvm tool generates ISP IP/SoC specific files which are used in the UVM-based verification framework as shown in Figure 17. Map file for mapping between the registers/register-fields/parameters of RTL and the attributes of Python model is also generated from IP-XACT based tool. Development of IP-XACT based tools helps to generate thousands of lines of code of verification environment in a very short time.

Figure 18 represents one example of input XML file and generated output register definition file. The XML file contains all the required information of the ISP IP/SoC such as base address (<spirit:baseAddress>0x100</spirit:baseAddress>), register name (<spirit:name>MUX<spirit:name>), register-field bit width (<spirit:bitWidth>4</spirit:bitWidth>), register-field bit offset (<spirit:bitOffset>0</spirit:bitOffset>), and register-field accessibility (<spirit:access>read-only</spirit:access>). Using the information in input XML file, spirit2uvm tool generates register definition file.

7. Results and Discussion

The development of unified and modular modeling framework of image signal processing algorithms provides the following benefits:

(i) Reduction of the development time

(ii) Coherent modeling approach making development and support not reserved to algorithms experts

(iii) Maximum reuse of IP

(iv) Bit-true certification between models and products

(v) Algorithms/ISP developers, HW designers, FW engineers, HW verification engineers, and application and

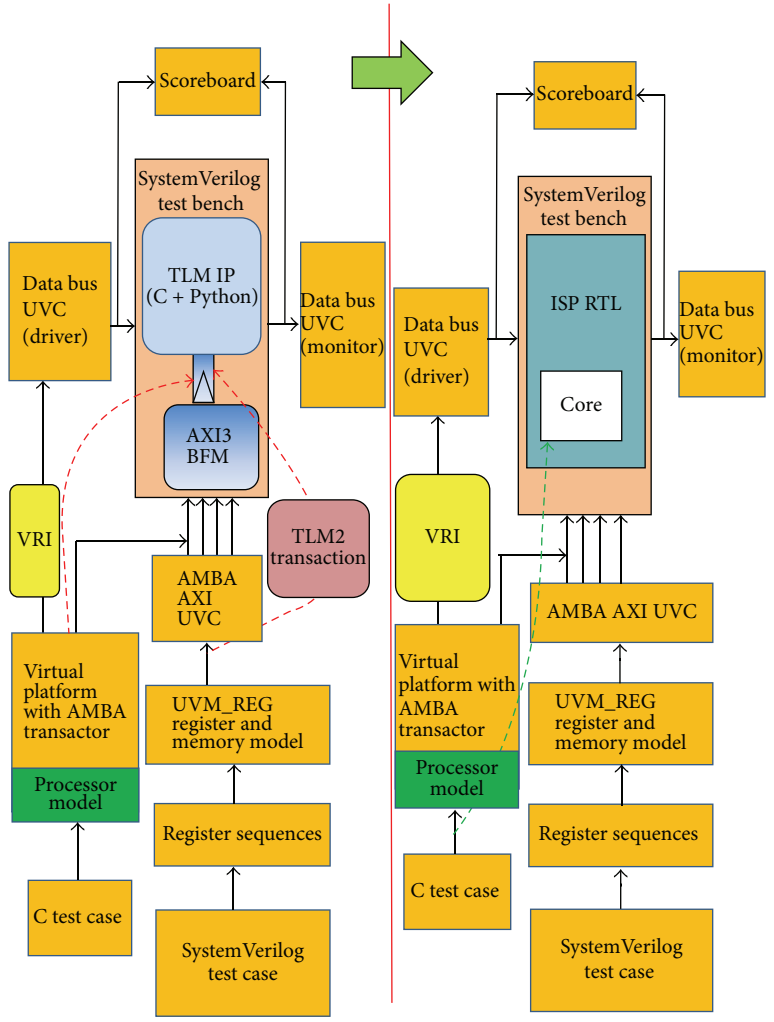

Figure 15: Reuse of early developed verification environment.

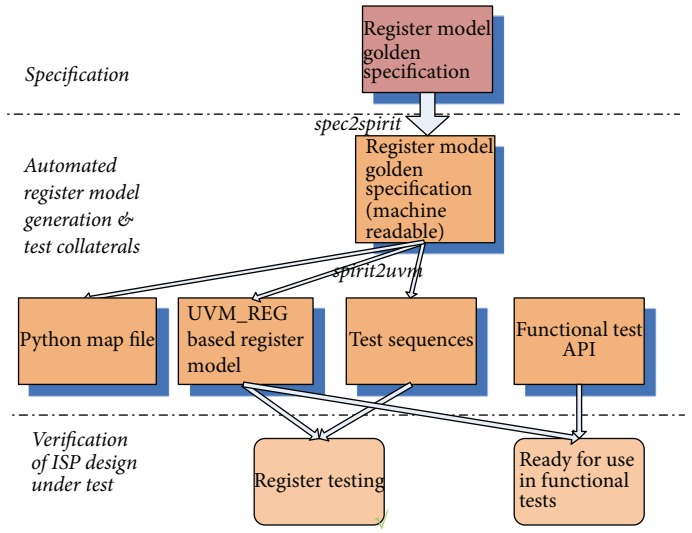

Figure 16: IP-XACT flow.

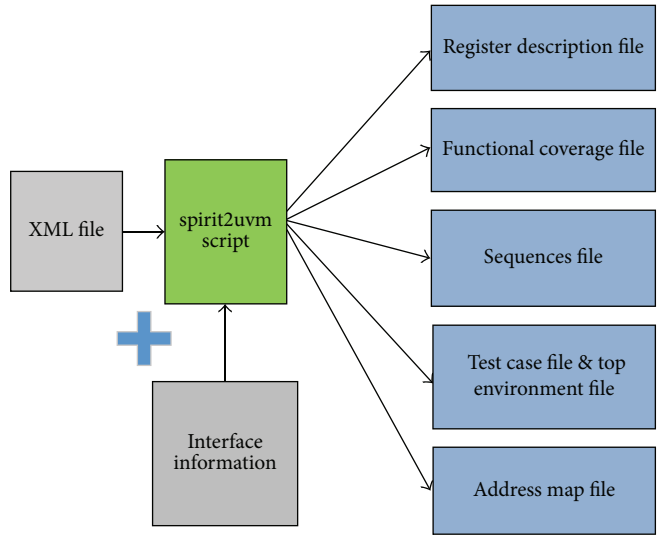

Figure 17: spirit2uvm script.

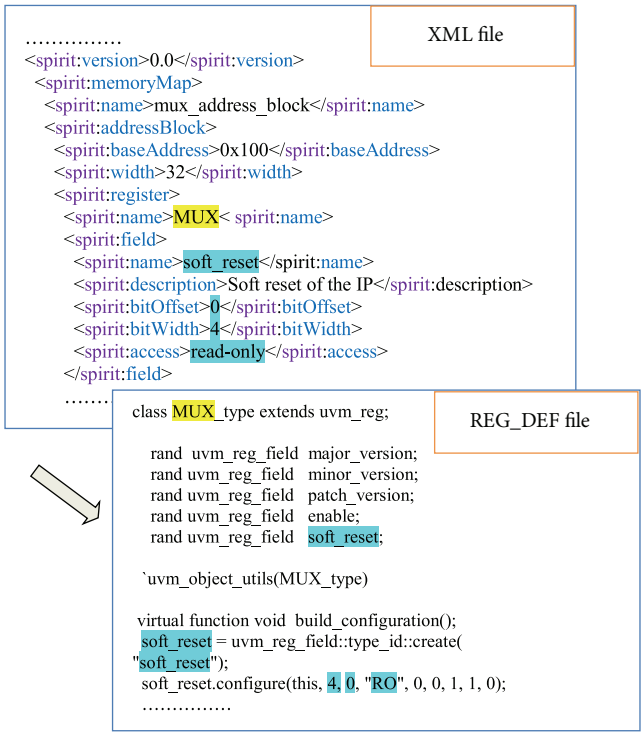

Figure 18: XML to REG_DEF file conversion.

marketing people using the same modeling framework

As compared to traditional methodology, development of UVM-based verification framework for ISP designs helped in saving verification cost and efforts by

(i) decreasing the maintenance of multiple verification environments;

(ii) improving the quality of the verification framework;

Table 1: Challenges of previous verification framework.

Factors	Description
Reusability	Different verification languages at unit level (Specman(e)/Verilog) and system level (C/C++) verification framework. Different verification methodologies both horizontally (across projects) and vertically (unit to system level verification).
Reproducibility	Significant time was spent in reproducing the issue reported at SoC level at IP/subsystem level.

(iii) early development of reusable verification framework;

(iv) flexibility in the verification environment to verify designs using both host interface and actual core in a unified manner;

(v) automatic development of verification environment files;

(vi) license cost saving; UVM is open standard supported by multiple vendor tools; thus, there is no need to pay extra license cost for one vendor specific solution.

Simultaneously, development of IP-XACT based tools helped to generate thousands of lines of code of verification environment in a very short time. This resulted in a significant reduction in product verification time and improvement in verification quality.

In our case of image signal processor designs, Specman (e)/Verilog based verification framework for IP/subsystem level verification and C/C++/Verilog based directed verification framework for SoC level verification were traditionally used for functional verification. The main challenges of the previous verification framework are described in Table 1.

Experimentation results which we listed in Table 2 are the comparison between the previous framework and the

Table 2: Experimentation results.

Comparison features	Previous framework	The proposed framework	Description
Product verification time (in weeks) of one ISP IP/SoC	IP: ~4 weeks SoC: ~12 weeks	IP: ~2 weeks System: ~6-7 weeks (verification productivity is increased by ~50% percent)	*Reusability*. The proposed verification framework can be reused both vertically (unit level to system level) and horizontally (across different projects). *Automatic Checkers*. Development and usage of automatic checkers (assertions and scoreboard) helped to automatically find bugs while running simulations. Formal tools were really helpful to find real complex bugs/problems. *Automation*. Automatic generation of verification framework files using IP-XACT flow helped us to generate thousands of lines of code of verification environment in a very short time.
Verification cost (in our case)	~500k dollars	~400k dollars (~20% cost reduction)	*License Cost Saving*. UVM is open standard supported by multiple vendor tools. Thus, there is no need to pay extra license cost for one vendor specific solution. Specman tool cost for the old eRM methodology was ~4000$/per license/year. *Source: ST license cost data sheet*. Minimum 40,000$ license cost saving per year (10 licenses × 4000$). *Man-Hour Saving*. Automatic development of verification environment and reuse of verification components helped to save the man-hour cost. Minimum 60,000$ manpower cost saving per year.
Quality of the designs (number of bugs)	15–20 functional bugs/problems per project	3-4 functional bugs/problems per project (~75–80% improvement)	*Automatic Checkers*. Development and usage of automatic checkers (assertions and scoreboard) helped to automatically find bugs while running simulations. Formal tools were really helpful to find real complex bugs/problems. Randomization, coverage-driven approach, and other features of verification framework helped to generate/analyze complex corner scenarios which resulted in resolving corner bugs. Interfacing issues and many corner case bugs are identified and resolved.

proposed framework in terms of product verification time, verification cost, and quality of the product for image signal processor designs. After analyzing results of the proposed framework used in imaging group of STMicroelectronics, it is found that the proposed framework shows better results and significant improvement is observed in product verification time (~50% improvement), verification cost (~20% reduction), and quality of the designs (~75–80% improvement).

Figure 19 represents a comparison chart between previous framework(s) and the proposed modeling and functional verification framework in terms of product verification time (~50% improvement), verification cost (~20% reduction), and quality of the designs (~75–80% improvement).

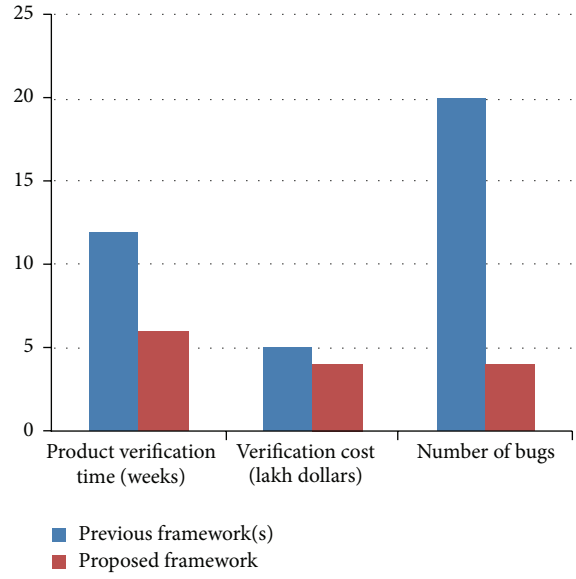

Figure 19: Comparison chart between previous framework(s) and the proposed framework.

8. Conclusion

This paper presented the work done in developing unified and modular modeling and functional verification framework of real-time image signal processors. SystemVerilog UVM-based verification environment with UVM_REG register and memory model integrated is used to verify a variety of image signal processor devices covering various protocols,

applications, and domains. VRI approach addresses the configuration of verification components from embedded software. IP-XACT based tools are used for automatic generation of verification environment files and model map files. This paper is a very good reference for modeling the image signal processing algorithms and for applying the advanced and novel techniques of verification and automation for development of verification environment and for functional verification of image signal processing designs.

Competing Interests

The authors declare that there are no competing interests regarding the publication of this paper.

Acknowledgments

The authors would like to thank Giuseppe Bonanno (Manager, STMicroelectronics), Piyush Kumar Gupta (Manager, STMicroelectronics), Dr. Vineet Khandelwal (JIIT, Noida), and Professor Alka Tripathi (JIIT, Noida) for their guidance and support. They would also like to thank management and team members of Imaging Division, STMicroelectronics, and faculty members and peer scholars of ECE Department, Jaypee Institute of Information Technology, for their support and guidance.

References

[1] A. Jain, S. Jana, H. Gupta, and K. Kumar, "Early development of UVM based verification environment of image signal processing designs using TLM reference model of RTL," *International Journal of Advanced Computer Science and Applications*, vol. 5, no. 2, pp. 77–82, 2014.

[2] A. Jain, G. Bonanno, H. Gupta, and A. Goyal, "Generic system verilog universal verification methodology based reusable verification environment for efficient verification of image signal processing IPS/SOCS," *International Journal of VLSI Design & Communication Systems*, vol. 3, no. 6, pp. 13–25, 2012.

[3] A. Jain, P. K. Gupta, H. Gupta, and S. Dhar, "Accelerating system verilog UVM based VIP to improve methodology for verification of image signal processing designs using HW emulator," *International Journal of VLSI Design & Communication Systems*, vol. 4, no. 6, pp. 13–25, 2013.

[4] S. Rosenberg and K. Meade, *A Practical Guide to Adopting the Universal Verification Methodology (UVM)*, Cadence Design Systems, San Jose, Calif, USA, 2nd edition, 2010.

[5] J. Bergeron, *Writing Testbenches: Functional Verification of HDL Model*, Kluwer Academic, Boston, Mass, USA, 2nd edition, 2013.

[6] N. Kitchen and A. Kuehlmann, "Stimulus generation for constrained random simulation," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD '07)*, pp. 258–265, San Jose, Calif, USA, November 2007.

[7] S. Marconi, E. Conti, J. Christiansen, and P. Placidi, "Reusable SystemVerilog-UVM design framework with constrained stimuli modeling for High Energy Physics applications," in *Proceedings of the 1st IEEE International Symposium on Systems Engineering (ISSE '15)*, pp. 391–397, September 2015.

[8] C. Liang, G. Zhong, S. Huang, and B. Xia, "UVM-AMS based sub-system verification of wireless power receiver SoC," in *Proceedings of the 12th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT '14)*, pp. 1–3, October 2014.

[9] Y. Kim, H.-S. Kim, R. Lee, and S. Kang, "FPGA-based verification methodology of SoC-type CMOS image signal processor," in *Proceedings of the IEEE International SOC Conference (SOCC '09)*, pp. 231–234, IEEE, Belfast, Ireland, September 2009.

[10] R. Kannavara, "Towards a unified framework for pre-silicon validation," in *Proceedings of the 4th International Conference on Information, Intelligence, Systems and Applications (IISA '13)*, pp. 321–326, July 2013.

[11] M. Glasser, "Open Verification Methodology Cookbook," 2009, http://www.springer.com.

[12] A. Jain and R. Gupta, "Scaling the UVM_REG model towards automation and simplicity of use," in *Proceedings of the 28th International Conference on VLSI Design (VLSID '15)*, pp. 164–169, Bangalore, India, January 2015.

[13] R. Edelman, A. Rose, A. Meyer, R. Ardeishar, and J. Polychronopoulos, "You are in a maze of twisty little sequences, all alike—or layering sequences for stimulus abstraction," in *Proceedings of the DVCon*, pp. 1–9, San Jose, Calif, USA, 2010.

[14] S. Iman, *Step-by-Step Functional Verification with System Verilog and OVM*, Hansen Brown, San Francisco, Calif, USA, 1st edition, 2008.

[15] Universal Verification Methodology, UVM 1.0, 2011.

[16] J. Andrews, "Unified Verification of SoC Hardware and Embedded Software," Chip Design Magazine, 2007, http://chipdesignmag.com/display.php?articleId=1267.

[17] Virtual Register Interface Layer over VIPs, Cadence Design System.

[18] Mentor Graphics, "UVM/OVM Cookbook," 2015, https://verificationacademy.com/cookbook.

[19] Accellera, "Standard Universal Verification Methodology Class Reference, Release 1.2.," 2015, http://www.accellera.org/.

[20] IEEE Standard for SystemVerilog-Unified Hardware Design, Specification, and Verification Language, IEEE 1800–2009, 2009.

[21] S. Swan, *An Introduction to System Level Modeling in System C 2.0*, Cadence Design Systems, 2001.

[22] A. Rose, S. Swan, J. Pierce, and J. M. Fernandez, *Transaction Level Modeling in SystemC*, Cadence Design Systems, 2005.

[23] F. Ghenassia, *Transaction Level Modeling with System C—TLM Concepts and Applications for Embedded Systems*, Springer, Amsterdam, The Netherlands, 2010.

[24] T. Grotker, S. Liao, G. Martin, and S. Swan, *System Design with System C*, Kluwer, Boston, Mass, USA, 2002.

[25] Accellera, "Spirit information," November 2015, http://accellera.org/xmlschema/spirit.