Image Processing Verification beyond Simulation, Emulation, FPGA synthesis

Enabling Image Process ASIC HDL verification with real-size test data without suffering from the slowness of Simulators, the high cost of Emulators, or having to learn FPGA synthesis.

- **Yasu Sakakibara, CTO, VAXEL Incorporated**

Image Process hardware design verification requires a set of very large test data. This has become truer with the recent increase in the frame sizes from FHD to 4K, 8K, and larger. RTL designers, however, are forced to conduct design verifications with reduced-size test data for two reasons: first, because today's Simulators take a long time to run large test data, and second, because the hardware emulation tools from the EDA vendors are just too expensive for most of the projects. As a result, RTL designers need to build unnecessary "modes and features" in their design just to accommodate the reduced-size test data. More tragically, they have no options but to take the higher risk of discovery of their RTL design flaws in a much later phase in the project, which would result in a much higher cost and more time to find and apply the "fixes." In this paper, the author, who has ten years of hands-on RTL design verification experience, will introduce a new platform for the Image Process ASIC HDL design verification that is fast and economical. It uses the FPGA technology, but with his implementation, the users do not need knowledge of and skills with FPGA, such as "synthesis," which itself could be a time- and money-consuming task.

Verification remains the bottleneck in recent ASIC development, especially if it requires much data and large numbers of test scenarios. The importance of verification increases with the Time to Market pressure of the hardware projects in the emerging IoT/AI era. Conventional verification tools such as Simulator, Emulator and FPGA Prototyping have their own pros and cons. This article introduces a new verification platform that has consolidated all the "pros," with some additional features that can further improve the verification efficiency.

1. Introduction

In my last project, where I oversaw the design verification of Image Processing blocks in an ASIC for a 4K TV, I was given a Simulator. However, this project was running on a tight schedule, and I could not afford to run a desirable number of test cases on it because it required nearly 12 hours for just single-frame 4K-size image data. I had to be creative, and I took an approach many of my fellow verification engineers would take under such circumstances. I made a set of test data with a frame size of 3840 x 4. By changing the frame size into lines of 3840 $\times$ 4 (instead of the full size for 4K of 3840 x 1920), I managed to improve the test execution speed of the Simulator significantly. This way, I achieved many of the performance verifications within a relatively short amount of time. However, as I had been concerned from the beginning, multiple problems were

found around the border conditions when “real” simulation was finally performed using the full frame of 3840 x 1920 lines toward the end of the development cycle. I faced an unexpected number of reworks and I struggled to fix them all on schedule.

This shows that with a “creative approach” like I took, although the verification time was shortened, the coverage of the simulation would be insufficient and the risks to blow up the project cost and release schedule would increase significantly. I have also performed some of the verification with an Emulator to accelerate the verification time using full frame at different projects. They did not produce satisfactory results either. Not only did I find the time required for preparing and setting up the right Emulator environment for my design was unexpectedly long, but I also found the results of the simulation using an Emulator were not very effective. In almost all of my Image Process HDL verification experiences, by the time I reached the verification coverage that I set as my target, I was exhausted mentally from stress. Because of this, I became highly motivated and determined to develop a new verification tool.

2. Considerations about the pros and cons of the conventional HDL verification methodologies

2.1. Simulator

A Simulator is without a doubt the verification tool that is most widely used in chip development projects. On one hand, it provides a large degree of latitude and helps you construct a verification environment where you can check operations closely by viewing waveforms and performing highly flexible input/output simulations. On the other hand, its operation speed is devastatingly slow. Therefore, you need creative approaches to the verification environment to compensate for the slowness. The following are examples:

- Bypassing some of the time-consuming and repetitive processes, such as initial settings.
- Adding some debug functions to the HDL design where verification can be performed with downsized input data.
- Preparing a set of specific verification data that is likely to extend to “boundary conditions” with a reduced amount of data.
- Executing “backdoor” memory initialization and dump.

These approaches will speed up a Simulator, but the downside is that they all lead to more complicated configuration procedures and more cumbersome code maintenance.

2.2. Emulator

While a conventional Emulator can perform high-speed verification because it is constructed with dedicated hardware to perform an HDL operation, the first and foremost issue with the Emulator is its cost. It is a very powerful tool and the capabilities are vast. But, the license for both initial deployment and the subsequent renewals is extraordinarily expensive. Because of this, Emulators are usually shared among multiple projects, even within the largest OEMs in the industry. Not only from the economical viewpoint, but also from the verification procedure

viewpoint, an Emulator requires you to prepare behavior models to substitute peripheral functions. This is also time-consuming work.

For these reasons, it is always a challenge to find an Emulator available with the right environment and at the right time to perform design verification specific to your needs. If you are lucky enough to find one available when you want it, to guarantee effective use of it, you need to consider the following:

- Defining a control method for the Emulator accommodating a large test data transfer.
- Formulating an effective verification strategy considering that only limited verifications were performed in the prior phase with the Simulator.

Design verification on an Emulator can often result in a poor perspective in terms of error correction due to the discovery of block-level bugs that were not caught in the previous process using the Simulator because of those cutting-corners approaches you took to compensate for the slowness.

2.3. FPGA Prototyping

FPGA Prototyping is an effective low-cost verification method compared to an Emulator. However, to effectively prepare the dedicated hardware requires extensive knowledge of FPGA. In many cases, an FPGA specialist is assigned to the project in addition to the design engineer and the verification engineer. As a result, the benefit of FPGA Prototyping diminishes rather quickly. The following illustrates other problems that need to be addressed with FPGA prototyping:

- The speed difference between on-board DRAM on FPGA and DUT.
- The necessity of RTL modification specifically for FPGA to run DUT at high speed.
- The observability of internal states.
- Defining the transfer method of input and output data.
- Determination of a control method for the FPGA board.
- That preparation of the FPGA board can be a time-consuming and costly project itself.

3. Design goals of “New Tools” (Simulation Accelerators)

As discussed in the previous sections, after analyzing the pros and cons of the existing verification tools, I have started to design my own New Tool that would include the positive aspects and exclude the negative ones of all the conventional tools as much as possible. In short, my goal was to build a tool that executes a simulation fast, with uncompromised debuggability features. At this point, I knew I would have no choice but to use FPGA, meaning that I had to consider that the tool would require minimum modifications to the target design (without any trial and error) before running it on FPGA. Also, the tool should come with a ready-to-use set of standard hardware function IP blocks, such as bridges and interfaces, so that the setup would be easy and fast. I also decided to give it a host-board architecture, such that the operation tools and other libraries would be readily available on the host side, and the communication between the host

and the board would be established in no time. Furthermore, I needed to make sure that the tool would be affordable, and that it would require little expertise in FPGA.
To accomplish the above, I have decided to implement the below.

- The tool will be a software package available on the Internet that users can easily download, install, and use.
- The tool will support standard off-the-shelf FPGA evaluation boards for low-cost deployment.
- The tool will utilize an embedded processor inside the FPGA for controls to provide "visibility" even in some complex verification processes.
- The tool will use HDL native features for waveform generation and assertion settings (as opposed to using the FPGA features or any other proprietary technologies) so that the users can operate such verification techniques with a skillset that they are already familiar with.
- The tool will deploy an operation scheme and structure that are similar to those of the Simulator, such that users do not have to learn new operations/methodologies.
- The tool has an interface to allow the Simulator to execute simulation on the tool.
- The tool will come with a migration assisting tool when users already have a verification environment set up on the Simulator and will have to relocate it to the tool.
- The tool will come with an automated FPGA synthesis script generator so that little expertise in FPGA is needed to set up the tool.
- The tool will come with CPU libraries to manage the verification environment.
- The tool will support USB for host-board communication.
- Verification processes are run by firmware on the embedded processor inside FPGA by command from the host.
- The tool will support commonly used programming language libraries such as C and Python as well as CLI, so that test data generation can be done by non-HDL design engineers.

The last point will broaden the engineering resources available for the HDL verification. I thought about adding this feature because in many projects, I saw that software engineers were just waiting until the hardware was reasonably ready to start developing their software. I thought this was inefficient. With the interface libraries of C, Python, and CLI, the software engineers can not only generate their own test data, but fully participate in the verification process. This will result in more tests, and thus better coverage. Consequentially, this may also bring a reduction to the overall budget of the hardware design project because a much larger number of software engineers are available compared to hardware engineers. With my tool, software engineers can now join the development of hardware verification via CLI and Python library. With regards to debuggability, because waveform generation and hardware assertion IPs are provided using HDL features as opposed to the FPGA-specific functions, the generation of waveforms before and after specific assertions can be easily done to help flexible debugging processes.

The next important point is that the tool was designed to require almost no knowledge of FPGA and FPGA synthesis. This feature was given some of the heaviest emphasis in my development of the New Tool. Specifically, what made it possible was the automated generation of an FPGA synthesis script. With this feature, users can generate files needed for FPGA synthesis by preparing a few simple parameter files. Additionally, the methodological differences, or the "gap" between an Emulator or FPGA prototyping and a Simulator had long been a blockage in environmental transition of verification. The New Tool provides an environment that minimizes this "gap," where the transition from initial smoke testing to the FPGA environment can be smoothly conducted. An entire verification process for the New Tool can even be performed transparently from a Simulator.

Lastly, I decided to implement a feature that lets you control multiple target boards from a single host. This is because the DUT size (gate count) is currently limited by the size of the FPGA gate scale. By enabling the handling of multiple FPGA boards, it partially eliminated the size limitations of the New Tool. I must note, however, that for the initial release of the tool, I was not able to include features for checking the inside status of the DUT and for expanding lines of hardware IPs corresponding to various DUTs. They are under development and will be released in the enhanced versions later in 2018.

4. Example Image Process ASIC design flow with the New Tool

This section will describe how users can easily and effectively proceed with verification with the New Tool. Conceptually, users will take these simple 3 steps shown in the chart below, starting by running the "smoke test" with the Simulator.

Steps	Verification Environment	Execution Platform	Actions
1	Simulator Environment	Simulator	Complete Smoke test
2	New Tool's Environment	Simulator	Confirm connection
3	New Tool's Environment	FPGA	Conduct verification

Step 1

When conducting the "smoke test" with the Simulator, take the following preparatory measures to make migration to the New Tool's environment smooth and easy.

(a) Design an input/output bridge to the DUT by synthesizable description.

(b) Use the New Tool's BFM (bus functional model for simulation) along with the bridge prepared at a) and start the smoke test.

Step 2

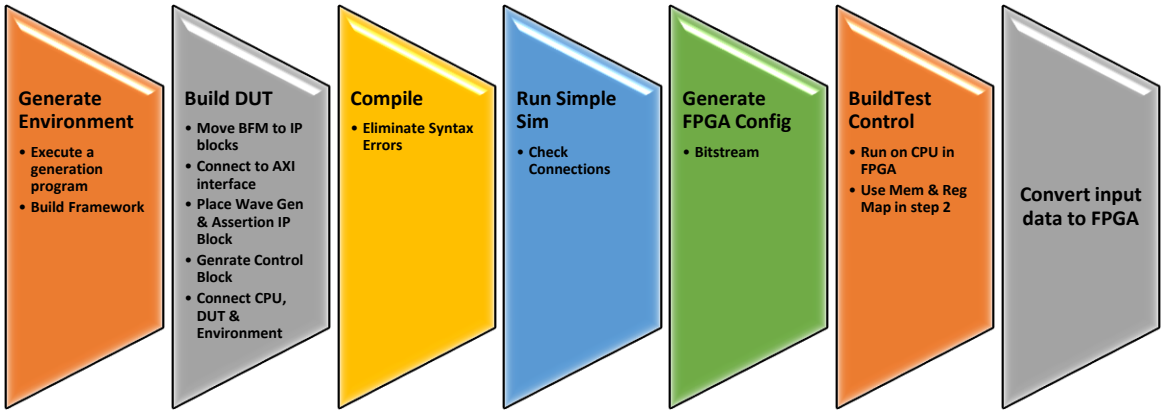
When the smoke test is done, move the verification environment to the New Tool and run the simulation from the Simulator. This will make the two verification environments compatible with each other in terms of syntax and functionality.

(a) Execute the New tool's automated verification environment generation program and build the verification framework in the New Tool.

(b) Build the DUT specific verification environment for the New Tool.

- Move the BFMs to the functional IP blocks and connect them to the AXI interface of the FPGA. (Nothing changes connection between the DUT and the IP blocks.)
- Place the waveform generation IP block and the assertion IP block and connect them.
- Generate a control register block by using the New Tool's feature and establish a connection between the CPU and the DUT and its environment.

(c) Compile the environment prepared above and eliminate syntax errors.
(d) Run a simple simulation to make sure all connections are good.
(e) Using FPGA synthesis script that was created at (a) above, generate FPGA configuration data (Bitstream).
(f) Using the APP-C library of the New Tool, build a control program (test scenario) that would run on the CPU in FPGA in accordance with the memory map and register map that were produced at (b) above.
(g) Convert input data to the FPGA format (text to binary).



Step 3

Now you are ready to start verification on the New Tool and its FPGA. Controlling the FPGA board, loading firmware to the CPU on FPGA, and transmitting and extracting the input/output data are all done by the host PC.

(a) The New Tool has a set of commands to control the FPGA board. Users can also use its Python library for the control.
(b) Load data below to the FPGA board

- Configuration data (Bitstream) for the New Tool's Environment
- Control firmware for the CPU on the FPGA
- Input data

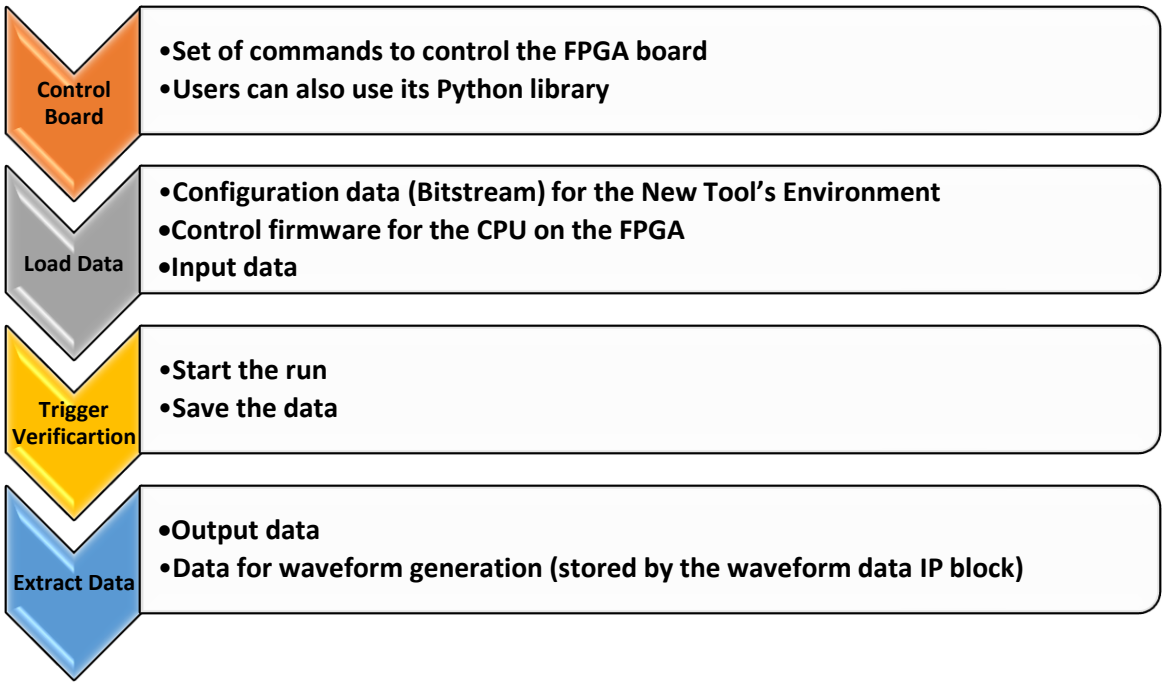
(c) Trigger verification

- Start the run
- Save the Data

(d) Extract data below when verification is done

- Output data
- Data for waveform generation (stored by the waveform data IP block)

(e) Use tools provided at the host PC view waveform and verify your HDL design.



5. Market for the New Tool

I am not a marketing specialist, so I am totally relying on my "gut feeling" to write this section. This does not go much further than my personal observation, but it seems that the evolution of HDL design verification tools has been driven predominantly by the needs of mainstream hardware development, such as processors (CPUs) and network controllers. To verify designs for such technologies, the "payload" side of test data is much less important than it is for Image Process designs. Thus, the amount of test data can be relatively small. For Image Process technology, and for other audio-visual related technologies, such as NLP, conducting verification with the payload of test data is critical. Thus, the amount of test data can be large. But the verification tool evolution was able to ignore it because it was a smaller proportion of the market. I have reached this conclusion because I have personally experienced shortcomings in the conventional verification tools and suffered from it. For this reason, I am certain that the New Tool will have a reasonable level of demand in the market. As a matter of fact, I think I am releasing this New Tool to the market at the exact right time, as I know that Image Process hardware development is one of the fastest-growing areas because of the need for high-resolution digital video and computer vision for applications like robotics and autonomous vehicles.

6. Conclusion

The New Tool can execute a large number of large data tests fast. The New Tool is no less effective in terms of debuggability when compared against Simulators. The usability and ease of operation of the New Tool is well designed and fine-tuned. It uses FPGA but requires little FPGA expertise and it is very affordable. You do not need to worry about the execution speed with my New Tool, meaning that you can use the original DUT without modification. Unproductive considerations such as back-door access will be eliminated. It sounds like a dream tool for Image Process design verification engineers. However, there are several limitations. The tool is based on the concept of Directed Tests. The New Tool does not support the Constraint Random test generation approach. (By the way, on this point, my colleague and I are now developing a new and more effective way to generate test patterns.) Another point I wanted to raise with the introduction of my tool is that the conventional verification tools are still deeply based on this historical and industry-wide notion that hardware engineers only are responsible for hardware verification, which in reality, I believe, is a very bad practice because it would lead to lack of verification "viewpoints" from software and application development. I added C, Python, and CLI interfaces to my tool. My New Tool will open doors to software engineers to participate in the product development from very early phases. I am also hoping that it will help close those technological "gaps" between the conventional verification tools. Presently, each of the verification tools needs some specific skills and knowledge to fully utilize its capabilities. This is not productive. I am hoping to improve this situation from which I have been suffering for years by inviting non-HDL fluent engineers into the world of HDL verification, by minimizing the need for FPGA-specific skills, and by lowering the economic barrier to deploying this new tool.

Actual effect of the New Tool at a customer project

- Application**: Video Surveillance Engine**
- Logical gate scale**: 1M Gates**
- Original time allocated for verification**: 8 weeks**
- Actual time for verification with a New Tool**: 2 weeks**