Synapse Design Position Paper

Verification: Build or Simulate?

Executive Summary

Companies are using FPGAs for the variety of benefits they offer, including:

- Rapid-prototyping
- Running large sets of test data
- Software development

The advantages of using FPGAs for verification include:

- Smaller, less complex designs can be verified solely by building them in an FPGA
- FPGAs are fast and therefore can run software and also push large amounts of data as required in some designs

However there are issues to be aware of when using FPGAs, especially as designs become considerably more complex. These include:

- FGPAs place limits on observability and access to internal nodes, making the debug cycle longer (often reverting the debug back to a UVM-based verification phase)
- It is more difficult to create certain corner case tests in an FPGA verification vs. in a UVM-based verification
- There is a large and costly effort to create an FPGA implementation for a large design, such as partitioning the design into multiple FPGAs, and keeping the RTL and FGPA implementations in sync

In the end, the FGPA debug can become a bottleneck on large designs, and then the problem becomes a speed vs. debug tradeoff as to whether an FPGA or UVM-based simulation approach is the better choice.

Some view the issue of build or simulate as a big company vs. small company issue, based on the cost of building and maintaining the two implementations. However, we see it more as an ROI decision, based on the unique attributes of the design being implemented. We will show examples of this below.

For these reasons, we propose blending the two approaches in a way that best suits

the design. The right balance provides the best payback for the cost, while also considering the risk management issues of the project (the ROI decision point).

We also see virtual platforms playing an increasingly vital role for early software development and optimization, an area where FPGAs dominate today. This alleviates some hardware-software integration issues and also offers other benefits such as early architectural analysis.

Synapse Design works on many different types of designs, for many different companies. This includes both large and small companies, and large and small designs. Much of this work centers on verification, and in the past 18 months, we have completed 34 large SOCs and none required even one metal respin, a testament to the thoroughness of the verification methodologies we employ.

A few quick examples of applying our recommended verification approaches follow.

Example 1:

The chip under design was a very large ASIC. We analyzed the partitioning into FPGA and quickly found out that it was not possible to create an efficient partitioning, primarily due to the number of connections between FPGAs, which was not sufficient. We would have needed a team to design the FPGA as big as the team designing the ASIC.

We decided instead to focus on the creation of more powerful verification suites.

Example 2:

A customer needed to create a reference platform with which to debug their firmware and to develop RTL signal processing algorithms. This platform allows us to connect multiple analog circuits to the control logic.

The ability to quickly integrate new RTL and to verify the performance of the new algorithms is what makes the creation of an FPGA worthwhile.

Discussion

Indeed, FPGA has become increasingly capable of doing and implementing more complex designs. Today, one can even implement a complex SOC in FPGA for all its known benefits (fast deployment, re-configurability (flexibility), rapid-prototyping, software development and so on)

One of the benefits has also been verification implication. The concept of verification by bulding an FPGA is actually attractive when chips are not that complex. Not only is the cost is lower, but the debug is still manageable. However, there are limits to it. As designs get more complex, "cost of debug" and "time to debug" begins to become a bottleneck. Debug is helped by "accessibility" and "observability" which is limited in an FPGA (because of the need to route signals, actually multiple signals to the pin so it can be seen on logic analyzer - too much overhead with limitation). This has forced the industry to start doing verification for FPGA, *just the same as ASIC*. In fact the difficulty in controlling and debugging often sends the diagnosis tasks back to simulators in RTL/UVM world.

There are other benefits that are available from a UVM advanced verification methodology that are very critical to success. One of them is coverage. Without coverage we do not know what has escaped testing. This is a big shortcoming of doing verification exclusively in an FPGA. Also, there are always corner cases that are very difficult to create and debug in FPGAs; but in the simulation world we have full control and therefore scenario creation is much easier; and so is debug. Assertions are also a big time saver in debug. For complex designs a key tradeoff is "speed of simulation" vs. "debuggability". For complex designs today, all advanced methodologies (like UVM) are deployed prior to FPGA implementation. **The new mantra** for complex design is "*First Verify by simulating, then verify it by building*". **It isn't either-or anymore**.

There are other considerations too. A complex SOC on FPGA takes a large dedicated team, which can be expensive. Partitioning and developing a methodology to debug takes quite a bit of effort. We are helping some customers to do a lot of FPGA debug because that allows them to stabilize the firmware, which in turn allows them to use ROM much more extensively and OTP/Flash for smaller portions of the code, thus obtaining large die size savings.

At Synapse Design we are helping some small companies with FPGAs and we are working with large companies who are choosing not to do an FPGA. We are working with companies who have complex SOCs with analog/mixed signal as part of the equation. In these cases we create a hybrid environment that mixes modeling with FPGAs for verification and software development.

It's the size and complexity of the design that dictates the methodology, not so much

the size of the company. The ROI needs to be justified.

Often only parts of a design are put on FPGA. Whenever speed is a must (such as 10 M pixel video frame - too much data to push through RTL simulator - just not possible) FPGA is the way to go. There are timing and system integration issues which we routinely find in FPGA that cannot be found in RTL. Our advanced verification in the RTL phase focuses on minimizing the escape of any bug that can be found in the RTL phase. This is where it is cheapest to find them. However, if one needs to deliver a system, one cannot run or develop software on RTL. FPGA is clearly the choice to develop, debug and run software (again, speed is a must). In this phase, one is validating both software and hardware. Today's SOCs are basically full systems. So a blend of "FPGA verification" and "advanced RTL verification" is now common. How we mix these approaches varies from design to design, and customer to customer.

We believe that trend towards higher and higher level of abstraction has a significant effect on verification strategy. The real reason for this is that this model development (virtual platform) can start with the architecture specification, much before RTL. It is a register accurate and fully functional model that can be used by the software team to develop and debug software (much before FPGA). It provides the "speed" and "debuggability" both as well as some other advantages (ref. white paper on the Synapse Design website). It does require some initial investment in developing the virtual platform, but the market advantage gained is big. Since software is debugged much earlier; FPGA "debug and validation" phase is shrunk significantly.

New synthesis tools (from high system-C level to RTL) will eventually be able to handle large classes of designs; and in time formal verification will be developed and mature for this abstraction class as well. Leaving that aside, a virtual platform today can provide a good common model for the software team and the hardware team. This is a big win.

One problem we see in software teams today is that they have their own private version of the virtual hardware model (disconnected with real hardware) to develop their firmware and software. Their software does not see the real world until it hits the FPGA. Then starts the agonizing phase of system validation and hardware-software debug. If one has developed and debugged the software on a Virtual Platform in an earlier phase; only validation task needs to be done in the FPGA phase. It helps that you already have largely debugged software ready to run on the FPGA (based off largely debugged RTL) for final validation. FPGA vendors are beginning to see this as well. Xilinx and Altera are providing links to virtual platforms and the TLM world for early software development.