FPGA Prototyping in Verification Flows

Application Note

Introduction

Hardware-assisted verification environments often make use of FPGAs to prototype the ASIC in order to provide a faster alternative to simulation and allow software development to proceed in parallel with hardware design. This *application note* addresses the factors that should be taken into account in such a flow in order to maximize the effectiveness of the overall verification environment.

FPGA Prototyping

Hardware-assisted SoC verification flows include acceleration, emulation or prototyping as part of the overall verification strategy in order to enable much higher effective simulation speeds allowing for software and system verification prior to device tape-out. Applications where a representative subset of the device functionality can be ported to an FPGA which is connected to real-world interfaces or laboratory test equipment providing stimulus and response benefit from the following main advantages:

- **Faster "simulation" speed:** much closer to real-time operation enabling much more data to be processed and if controlled correctly perhaps more corner cases to be stimulated.

- **Realistic system environment:** where possible the device can be proven to interface to other known system components and external interfaces.

- **Software development platform:** the improved performance allows concurrent development of the software with the evolving hardware.

It is important to stress that *FPGA prototyping does not replace simulation-based verification*, but rather it is a technique to improve the overall effectiveness of the verification. The main reasons for this are that *the FPGA is not the final product* and that the FPGA prototyping environment *lacks many essential features* provided by simulation-based verification, in particular:

- **Different physical implementation:** The FPGA has a different netlist, layout, clock-tree, reset distribution, memory structures, I/O characteristics, power management, performance, clock frequency, clock domains, etc.

- **Unimplemented or modified modules:** where possible the RTL differences should be minimised to improve maintenance but typically custom cells are missing, analogue modules are not available or implemented in a separate prototype chip with performance considerations, often the full design will not fit in a single FPGA and needs partitioned across several FPGAs, etc.

Copyright © 2006 Verilab

CONFIDENTIAL

- **FPGA debug environment is poor:** in particular the controllability and observability of events inside the FPGA is inadequate even with the most advanced probing techniques. If the prototype environment happens to generate interesting corner cases which cause observable failures these are very time consuming to debug. If the prototype environment does not cause failures, it is difficult to know if it really generated appropriate interesting corner cases or realistic stress conditions. In other words *functional coverage* is difficult to implement and measure in an FPGA prototyping environment.

- **Defect tracing to FPGA, hardware or software:** since the FPGA prototyping environment is being used to debug a printed circuit board (with sockets, connectors, crosstalk, etc.), evolving software and a representative part of the device at the same time, it can be difficult and time consuming to trace the cause of incorrect behaviour.

- **FPGA turnaround time:** for a complex SoC being targeted to an FPGA implementation, the back-end processes of synthesis, placement, static timing analysis, and bit-stream generation is likely to consume a considerable amount of time (of the order 4 to 8 hours is common for some highly utilized high-speed implementations). This means repetitive rebuilds to solve minor issues or connect up a different set of debug signals is prohibitive.

- **Process, voltage and temperature:** any operations on the FPGA prototype represent a limited subset of the overall device environmental operation conditions (and a different physical implementation) – a single point on the PVT curve. It may be possible to perform limited stressing of the voltage and temperature but it will never be possible to emulate the process tolerances for the final chip. Most design teams care about reliable operation over the entire specified environmental and functional operational conditions.

Simulation Environment

Most of the essential features for an effective verification environment that are not met by the FPGA prototyping solution are the corresponding strengths of simulation-based verification flows. Creating an appropriate symbiosis between the two and understanding the relationship is key to maximizing the effectiveness of the overall solution. The particular strengths of simulation in this context are:

- **Debug environment:** digital simulators and supporting tools are the perfect debug environment. All the signals in the design are readily accessible. Once a defect has been reproduced in this environment the debugging can be performed extremely efficiently. For this reason it is recommended the that simulation environment be designed with the *capability* of executing the same tests performed on the FPGA in order to reproduce failing scenarios and enable effective – this has implications for both the testbench design and the test code structuring.

- **Controllability and observability:** simulation provides excellent control and observability of signals. High-level verification languages use constrained-random stimulus generation to target corner case operation and stress conditions. Assertion-based verification can be used to check intended operation at many levels of the device including internal interfaces, clock-domain crossings, state machines and

external protocols. Temporal expressions, protocol monitors and assertions also provide self-checking functionality throughout all simulation runs.

- **Functional coverage:** functional coverage using high-level verification languages and, to a lesser extent, assertion-based verification, provides detailed feedback on which features of the design have been appropriately stimulated and checked. *Code coverage* can also be measured effectively in the simulation environment to provide another metric to assess verification completeness.

- **Turnaround time:** validating a fix to the RTL in the simulation environment typically takes a very short time: with *make* based compile flow and deterministic constrained-random testbench generation techniques the new RTL can be subjected to the failing scenario in a matter of minutes. Gate-level defects that are undetectable in other stages of the flow take considerably longer to turn around.

- **Gate-level simulations:** where necessary, back annotated gate-level simulations can be used to validate static timing analysis, detect reset issues, check for incorrectly synthesized code (together with equivalence checking) and, most importantly, check for dynamic timing defects like clock glitches. These simulations can be performed using best and worst case process/voltage/temperature conditions.

Verification Plan

The key to managing the workload distribution between the simulation and FPGA based components of the verification environment is the verification plan. In particular a properly maintained verification plan enables the team to:

- **Target specific features for system test on FPGA:** the effectiveness of the overall solution is improved by identifying which features are slow to simulate, or hard to emulate, in the simulation environment and targeting these for validation in the FPGA. Likewise the majority of the features which do not fall into this category can be targeted for simulation – ensuring fewer defects propagate to the FPGA which will help contain the difficult debugging situation. There will of course be a large overlap of features stimulated in both cases, but the main point here is to focus on where things are meant to be verified and proven.

- **Identify functional coverage:** the verification plan should identify the functional coverage goals for the simulation environment and where the corresponding gaps should be. Some inferred coverage should be possible for low risk items, however key features identified for validation by FPGA alone should have appropriate coverage mechanisms built-in to the FPGA implementation.

- **Analyze risk and make appropriate decisions:** verification is seldom completed in time for tape-out. It is important to assess the priority and risk associated with every feature to enable informed decisions and avoid “killer” bugs.

Copyright © 2006 Verilab

CONFIDENTIAL

Regressions

All aspects of the verification environment should have the capability to perform regressions, in other words all tests must be repeatable to ensure that when defects are introduced during development they will be detected. A requirement for effective regression is that the verification environment must be self checking; since the FPGA prototype forms part of the verification environment solution it follows that tests ran on the FPGA must also be self checking. A one-off lab test executed on the FPGA to see if something "works" is about as useful as a simulation that requires the user to look at the waveform output to determine success or failure.

Improving Effectiveness

The following items are suggestions of things to consider for improving the overall effectiveness of a verification environment that includes both FPGA prototype and simulation components:

- **Design for verification:** in many applications it is possible to implement functional operational modes which enable many other features of the design to be simulated in a much faster way. For example in telecoms applications where most of the functionality is related to header (overhead) processing rather than data (payload) content, it is possible to define much shorter frame modes which keep all the headers and associated operations like justification, parity and CRC calculations intact. In such cases the simulations are performed primarily on the verification modes and the FPGA used for normal operation.

- **Synthesizable assertions:** functional checks can be synthesized into the FPGA to detect critical protocol violations deep inside the hierarchy or measure functional performance characteristics such as FIFO fill levels and arbitration fairness. Although this logic is redundant, in that it is not a requirement of the specification, including it can save a huge amount of effort when debugging problems or trying to replicate failures in the simulation environment.

- **Functional coverage measurement:** functional coverage monitors can be synthesized into the FPGA to ensure the system tests manage to generate appropriate scenarios. In general the majority of the functional coverage should be collected in the simulation environment; however coverage holes, particularly related to features targeted for validation in the FPGA, can be implemented using embedded counters and state sequencers. Care is required to ensure the accuracy and relevance of this coverage since it consumes FPGA resources. Such coverage can be extracted at the end of a test using techniques such as uploading and post processing the FPGA configuration chain, using dedicated scan chains for the coverage accessed via a JTAG TAP controller, or using the FPGA debug probing infrastructure.

***Mark Litterick**, Verification Consultant, Verilab GmbH.* mark.litterick@verilab.com

Copyright © 2006 Verilab