Python-based trace and debug of FPGA designs

By Jonathan Stenroos and Hugo Sjödin

When building integrated circuits, it is important to have systems in place for making sure that they work as intended. For this purpose, there exists computer programs called simulators that can mimic the functionality of the circuit and show how the internal signals react when exposed to different stimulus. Simulators are very convenient to work with, controlled through intuitive user interfaces and gives complete system observability. All problems cannot be solved through simulators however, a tester needs to be able to see how the circuit behaves in the real world and how it interacts with the hardware that it is connected to. One way to do this is to install a so called embedded logic analyzer (ELA) inside the circuit itself. An ELA can be seen as an advanced multimeter, it represents dedicated verification hardware capable of monitoring signals inside the circuit. To some extent, the goal of an ELA is to provide the functionality of a simulator but working on the real implemented system rather than a software model.

This project is about implementing a custom ELA module written in the hardware description language System Verilog. It is meant to be used for verifying a type of programmable circuit board called FPGAs. There already exists commercially available ELAs today, but the system that we have developed has a few key features that distinguish it from these. Firstly, it is built to be completely controlled and configured through the Python programming language. By wrapping complex verification systems behind a user friendly Python library we aim to create a streamlined verification process. Secondly, it is extensively reconfigurable to fit a tester's needs. Our design has systems in place for dynamically specifying things such as which signals to monitor even after the circuit is fully constructed and in operation. For popular commercial ELA solutions such as Xilinx Vivado's Integrated Logic Analyzer (ILA), these types of configurations usually have to be done before they are built and placed in the FPGA circuit.

We have evaluated our test system against Vivado's ILA in terms of metrics such as how complex and resource intensive the systems are and how easy they are to interact with for a tester. We found that our system presents some key benefits, but also noteworthy drawbacks. On the one hand, our system proved to be more resource efficient than Vivado's ILA in many circumstances, in part due to the dynamic reconfiguration feature allowing a tester to cover a large amount of signals cheaply. On the other hand, Vivado's ILA offers a greater breadth of features as well as a more intuitive user interface.