Assertion-Based Verification

In recent years, *Assertion-Based Verification (ABV)* is being widely accepted as a key methodology in the pre-silicon validation of system-on-chip (SOC) designs. *Assertions* (i.e., properties that must hold at all times) are used to formally express and capture the design behaviour, internal synchronization, and expected operations, and either through simulation or formal verification of these assertions, verify that the design correctly implements that intent. Assertions can be expressed at multiple levels of the device including internal and external interfaces (to detect critical protocol violations), cross-domain crossings and state machines. ABV guides the verification task and quickens the verification because it provides feedback at the internal level of the device. Therefore, it is possible to locate the cause of a problem really fast. Examples of assertion languages are *Forspec*, *Sugar/PSL* (*Property Specification Language*), *SVA* (*SystemVerilog Assertions*).

The concept was originally introduced in the scope of formal (static) verification. This approach uses a simple language based on a temporal logic (such as *linear temporal logic (LTL)* or *computation tree logic (CTL)*) to describe specification of parts of a system in the form of a set of temporal formulae. In the case of formal verification tools, the state space of the system is then exhaustively searched to check whether all specified temporal formulae are valid in all accessible states of the verified system. If a state that invalidates some formula is found, a *counter-example* (also called a *witness*) is given in the form of a waveform of a sequence of input and internal signals that leads to the failing state. In the case of dynamic verification tools (based on the simulation), in each cycle of the simulation all formulae are checked for validity and if a failure occurs, the simulation run is stopped with an error message.

As the complexity of modern hardware systems rises rapidly, the verification takes a significant amount of time. It is a challenging task to find appropriate acceleration techniques for this process. As a result of my master thesis I have proposed HAVEN, a freely available open verification framework that exploits the field-programmable gate array (FPGA) technology for cycle-accurate acceleration of simulation-based verification runs. HAVEN takes advantage of the inherent parallelism of hardware systems and moves the verified system together with interface components of the verification environment from software into an FPGA. However, during the acceleration it is not possible to check assertions implicitly as it is common in a simulator. To solve this problem, an independent component in the HAVEN verification environment can be implemented and its main purpose will be checking of assertion(s) during the verification run in an FPGA. It could be feasible thanks to the linear-time nature of assertions, because every assertion formula can be effectively transformed into a Büchi automaton, which is in turn easily synthesisable into a finite-state machine in an FPGA.

One part of my dissertation thesis will discuss the synthesis of assertions into hardware as it is supposed to be an improvement of current HAVEN framework. There are also other reasons that motivate me to investigate this topic, namely:

1. To extend ABV to hardware emulation and early design prototypes (such as FPGA).
2. To debug post-silicon violations of temporal properties.
3. To enable a hardware designer to build recovery modules in safety-critical applications that are triggered on a match of the assertion checker, modelling the temporal fault.