Challenges in CAD for the One Million Gate FPGA

Kurt Keutzer
Synopsys, Inc.

Abstract

FPGA and CPLD densities have typically lagged their gate-array and standard-cell counterparts in effective density by a factor of 10. This means that the design problems encountered by ASIC designers will typically be encountered by FPGA designers 5-6 years later and tools used in the ASIC market to address them will ultimately be adopted by FPGA designers. Thus anyone familiar with the evolution of ASIC design tools can amaze and impress their FPGA-focussed friends with their ability to predict the future of FPGA design tools. In this short abstract we will outline the probable evolution of FPGA devices as they grow to one million gate-equivalents and then will chart the evolution of FPGA tool needs as they grow to meet the challenges of the million gate FPGA.

1 Defining the 1M Gate FPGA

CAD tool needs for a system implementation medium, such as an application-specific integrated circuit (ASIC) or a field-programmable gate array (FPGA) are largely determined by the capabilties of the device. These include the speed at which the device can be used, the number of gates of logic or bits of memory that can be implemented in the device, and finally the cost of the device itself. Estimates from FPGA and CPLD vendors indicate that 500K gate devices will be achieved by the year 2000 with the 1M FPGA of our discussion soon to follow. These devices will run clock speeds well in excess of 100MHz. It seems almost certain that FPGA vendors will follow the trend of ASIC's and incorporate at least one microprocessor or microcontroller together with a significant amount of on-chip memory into these devices as well. The resulting combination of a processor, memory sub-system and un-characterized logic creates the FPGA incarnation of a *system on a chip*. Because the microprocessor and memory sub-systems do not pay the overhead of the programmable-logic portion of the chip the relative cost of the FPGA system-on-a-chip will draw increasingly closer to its ASIC relative. For these reasons the FPGA system-on-a-chip appears to be a device that is technically viable and commercially attractive.

For these reasons, in the remainder of the paper we will review the CAD needs for servicing the FPGA version of a system on a chip. It may be of interest to note that the FPGA incarnation of the system-on-a-chip has the promise to be more than simply an FPGA version of its gate-array or standard-cell cousin. A system-on-a-chip implemented on an FPGA or CPLD has the ability to be *fully user-programmable* through the mixture of electrical programming of the FPGA or CPLD portions and through software programming of the software sub-system. Thus it may be more accurate to call the FPGA system-on-a-chip, a fully user-programmable system-on-a-chip (FUP-SOC). Such a device has the promise to help to overcome the verification bottleneck that will be briefly discussed later.

In discussing CAD needs for the FUP-SOC it may be valuable to divide the design process into three activities: design entry, design implementation and verification. These will be discussed individually in the following sections.

2 Design Entry for the 1M Gate FPGA

Because a FUP-SOC incorporates both the hardware portion *and* the software portion of a system on a single die, to enter the design of a FUP-SOC requires the abilty to model a system with both hardware and software components. There are at least a couple different approaches to this problem. The most widely used approach is to simply use a common programming language such as *C* or *C++* for modeling the hardware-software system. In this approach the partitioning of the model into modules for hardware and software implementation is made manually and the individual modules are gradually refined into efficient *C*-code for the software portion or into an HDL for the hardware portion. A natural evolution of this approach is to improve synthesis tools to handle the *C*-code directly so that hardware can be generated directly from the system description. Another potential evolution would be the birth of more general languages from which either hardware or software can be generated.

A different approach is to use a heterogeneous simulation environment to integrate a variety of *domain-specific* approaches, such as statecharts for control-oriented applications, and dataflow diagrams for dataflow-oriented applications. Either hardware or software can be automatically generated from these descriptions but the generation of efficient implementations of either requires manual intervention. In addition, the partitioning into hardware and software is manual and is likely to remain so for some time.

3 Implementing the 1M Gate FPGA

The software implementation route for an FUP-SOC is likely to rely on traditional software development environments for embedded processors. These include compilers, assemblers and debuggers. Due to limited on-chip memory there is still a certain amount of hand-coded programming and a reticence to use memory gobbling real-time operating systems; however, the use of higher-level languages and RTOS's in embedded systems is steadily increasing.

The hardware implementation route for a FUP-SOC is certain to rely on an orchestration of HDL synthesis, module generators and re-use of large blocks. HDL synthesis appears ready for the challenge of FPGA and CPLD architectures and efficient module generators are currently offered by the major programmable-logic vendors. The increased used of static timing verification and a closer integration of physical design and synthesis are two other certain developments. Finally, an independent market of re-useable designs for programmable devices is emerging to meet the need of the FUP-SOC.

4 Verification of the 1M Gate FPGA

While the design entry and implementation of the FUP-SOC closely follows the evolution of the system ASIC, the verification of the FUP-SOC may be significantly different. For the ASIC-SOC a significant amount of verification is required at the system level. Designers are reticent to commit to the labor of implementing a design that has not been verified against its system specification. This verification process continues through each of the stages of design refinement. Along the way the actual implementation of the hardware must be co-verified with the implementation of the software. In addition each step of refinement of the implementation must be verified for consistency against the last. For example the HDL model that is used for synthesis must be verified against the system-level model that inspired it. Currently for many integrated circuit designs the verification of the functional correctness of an integrated circuit design ultimately serves as the bottleneck to the entire design process. As a result it is often echoed in the industry that the gap between the complexity of devices that processing makes available and the complexity of devices that can be efficiently designed is growing wider with each new generation of semiconductor processing.

The FUP-SOC promises to be one device that can bridge this gap. Because it is a fully programmable device, a system development paradigm can be used that is much closer to software development. It is likely that some time will still need to be invested in system-level verification to make some initial feasibility analysis before commiting to an implementation. After this phase one can easily imagine the bulk of the functional verification being done through programming the actual device. To understand how this helps bridge the verification gap it may be helpful to better understand the verification gap itself.

One of the major difficulties in functional design verification is developing the test stimuli that are used to verify the design. Billions of vectors may be required to adequately verify devices which perform complex functions such as protocol processing or image processing. A second difficulty is that the speeds at which the stimuli can be applied through software simulation are typically less that 50 cycles-per-second in a simulator. This number may reach 500K cycles-per-second in an emulator but even this number falls still short of system speeds of over 100MHz. The FUP-SOC provides a device that can simply inserted directly into the final operating environment and can operate at the system speed. The test stimuli and verification monitoring are then naturally provided by the environment and in this way all the issues regarding stimuli generation, monitoring and speed are finessed.

5 Conclusion

Advances in semiconductor processing will give FPGA and CPLD vendors the ability to create a *fully-user-programmable system-on-a-chip* (FUP-SOC) by the end of the decade. These devices will incorporate at least one processor, a significant amount of on-chip memory, and hundreds of thousands of gates of un-characterized logic all on a single die. In addition these devices will be fully user-programmable through the mixture of electrical programming of the FPGA portions and through software programming of the software sub-system. It seems certain that the CAD tools to perform the top-down design and implementation of these devices will be readily available as the devices themselves mature. Moreover, these fully programmable devices offer the promise to bridge the increasing gap between what can be designed and what can be efficiently verified.