

Designing FPGAs with HDLs and Synthesis Tools

As the size and complexity of FPGA-based designs continue to grow, some users are turning to hardware design languages (HDLs) and logic synthesis tools to enter their designs.

To effectively use an HDL, users must understand the language syntax as well as the potential and limitations of this design entry method, especially when creating generic code intended for different devices.

To aid in this process, Xilinx is preparing a comprehensive *HDL Synthesis Design Guide for FPGAs* which will be available to users around mid-year. It will describe design methodologies and illustrate them with design examples. The guide is intended to help HDL users produce successful FPGA designs.

This article gives a brief description of some of the advantages of HDL-based design, and then briefly reviews some of the material included in the forthcoming design guide.

HDL Advantages

HDLs and synthesis tools allow the logic designer to enter designs at a higher level of abstraction, much like the software engineer who programs in 'C' instead of assembly language. The designer specifies the needed system-level functions, whereupon error-free gate level implementations are generated by the synthesis tool, freeing the designer for more creative tasks.

Users look to high-level languages and logic synthesis to provide an efficient means of migrating designs between technologies — the design's high-level description is not necessarily bound to a given device architecture or process technology. Like schematics, HDL-based designs also

are self-documenting since the HDL file is the functional description of the design.

Logic synthesis tools and FPGAs complement each other in providing a flexible design environment. With HDLs, decisions can be tested early in the design cycle through functional simulation of the HDL description. Design changes are made easily, allowing experimentation and the exploration of architectural trade-offs. Synthesis tools then convert the HDL description to a gate-level implementation for the target FPGA architecture.

FPGAs further enhance this design flexibility by allowing the user to implement and test the design at the workbench. Typically, the synthesis compilation time is short enough to allow for exploration of design trade-offs at their gate-level implementation. SRAM-based FPGAs can be reprogrammed an unlimited number of times, so multiple iterations of the design can be implemented with no additional hardware cost.

FPGA-Optimized Synthesis Tools

For a top-down, HDL-based design methodology to be useful, the synthesis tools must be effective in producing a gate-level design for the target technology. Synthesis algorithms for FPGAs can be dramatically different from those used for gate arrays. Fortunately, many synthesis tools have special optimi-

“HDLs and synthesis tools allow the logic designer to enter designs at a higher level of abstraction, much like the software engineer who programs in 'C' instead of assembly language.”

Continued on next page

HDLs and Synthesis Tools

Continued from previous page

24

“Device performance and area utilization can be optimized by creating HDL code that takes advantage of special FPGA architectural features.”

zation algorithms for Xilinx FPGAs.

For example, in the Synopsys FPGA Compiler, logic is synthesized into the building blocks of the FPGA: function generators (look-up tables) and registers. The FPGA Compiler reports the area utilization and critical path delays in terms of CLBs, not gates. This guarantees a high level of correlation between the reported area/speed numbers and the actual results after implementation by the place and route tools.

To address the demand for effective synthesis tools, Xilinx initiated its Synthesis Syndicate Program about two years ago, a program to assist and encourage third-party EDA vendors in developing synthesis tools for Xilinx components.

ASIC vs. FPGA Design

The methods and techniques used in ASIC design are not always best for FPGAs. ASICs typically have more gates and routing resources than FPGAs. Since ASICs have a large number of available logic resources, inefficient code that results in a larger-than-necessary number of gates often can be tolerated. When de-

signing FPGAs, the results of inefficient coding will be magnified.

Language Issues

Not all the constructs available in an HDL may be applicable to FPGA design. For example, VHDL, which was originally developed as a simulation language and later adopted for IC design, includes a “wait” statement instructing a

VHDL simulator to wait for a specified time before a condition is executed. This statement does not synthesize to any components. In a design that includes this statement, the functionality of the simulated design may not match the functionality of the synthesized design.

Various synthesis tools may also use different subsets of the VHDL language. Furthermore, constraints and compiling options can perform differently, depending on the target device.

Using FPGA System Features

Device performance and area utilization can be optimized by creating HDL code that takes advantage of special FPGA architectural features such as global resets, wide decoders, on-chip memory and carry logic — common knowledge for experienced FPGA users. However, users must learn how to access these features in an HDL-based design. Some must be “instantiated” into the code (that is, the user must explicitly specify that these resources are to be used by embedding directives within the HDL code). The user must therefore be familiar with the target FPGA architecture. The code is architecture-specific and not easily transferred to other devices.

Hierarchical Design

Hierarchical design is important in the implementation of an FPGA, particularly during floorplanning and debugging, or when using incremental design techniques. Large designs (greater than 5,000 gates) should be partitioned into smaller modules. The size and the contents of the modules can affect synthesis and implementation results.

HDL Design Flow

The design flow for FPGAs when using HDLs is very similar to that of ASICs. The basic steps are listed below:

1. Entry of the HDL code; the design should be evaluated for inefficient coding styles and for possible usage of FPGA system features.
2. Functional simulation with a VHDL or Verilog simulator. Xilinx provides VITAL compliant simulation modules that can be used with the Synopsys VSS simulator.

CONTINUED

3. Synthesis of the design's modules into XNF or EDIF netlists. The area and speed requirements should be specified before the design is synthesized.
4. Translation of the XNF file or EDIF file into a Xilinx Unified Library XNF file.
5. Functional simulation with a gate-level simulator (optional).
6. Floorplanning of structured design elements such as RPMs and on-chip memory blocks to improve routability and performance (optional).
7. Implementation of the design with the automated "place & route" tools.
8. Timing simulation with a gate-level simulator.

In summary, an increasing number of FPGA users are adopting top-down design methodologies using HDLs and logic synthesis. While still not a panacea, synthesis technology is starting to live up to its promise of enabling efficient, high-level FPGA design. ♦

Measuring Speed and Temperature

All CMOS circuits experience increased signal delay with increasing chip temperature, typically about a one percent speed degradation for every three degrees centigrade temperature increase. This is a basic phenomenon, and cannot be changed by any manufacturer.

A chip's temperature is affected by ambient temperature and device power dissipation. More specifically:

$$T_j = T_A + P_D * \theta_{JA}$$

That is, the silicon junction temperature exceeds the ambient temperature by the product of the dissipated power multiplied by the thermal resistance of the package. This thermal resistance is primarily a function of package size, package material, internal package structure and air velocity.

SRAM-based FPGAs have no significant static power consumption. Practically all internal power dissipation is due to the dynamic charging and discharging of capacitive nodes. This makes it impossible to generalize the device power consumption of Xilinx FPGAs; it can vary by orders of magnitude, depending on the application. Years ago, the power consumption was always relatively low because FPGAs were limited to less than 5,000 gate density, were often not used fully and employed clocks of 20 to 30 MHz. Today, capacity has increased beyond 20,000 gates, better software allows

utilization of up to 100 percent, and clock rates can go well beyond 50 MHz. As a result, power dissipation can be several watts for the largest FPGA devices running at full speed.

This has rendered the traditional 70°C specification unsatisfactory for most demanding applications.

Xilinx has responded to this problem. We are now testing commercial devices at 85°C and industrial devices at 100°C. The new edition of the *1994 Data Book* (3rd edition) provides derating factors for higher junction temperatures (0.35 percent per °C for XC4000 devices, 0.30 percent per °C for XC3000 and XC2000 devices). The thermal resistance for the various device/package combinations, with derating values for airflow, are listed in this new edition as well.

The change to the higher test temperatures was implemented in April. All devices with date codes 9512 or later are tested in this new manner.

SRAM-based or antifuse-based FPGA performance parameters cannot be guaranteed at a specified ambient temperature, independent of power consumption and package type. Depending on the design, the clock rate and the package, the device junction temperature might vary by more than 50°C.

It is our goal to provide clearly-defined device parameters that are meaningful for the user. ♦