HDL Methodology Offers Fast Design Cycle and Vendor Independence

Joseph Cerra, Senior Design Engineer
Wellfleet Communications Inc.

In the highly competitive data communications field, the ability to bring a product to market quickly is essential for success. Using FPGAs with an HDL methodology offers this fast time to market by providing the flexibility to design, debug and verify, all within the same environment. In addition, the use of HDL allows the relative ease of re-targeting a design between vendors/technologies.

The design consists of a complex multiple port DMA controller for a data communication card that resides in a high speed router environment. The objective of the design was to control the flow of data from two TI TMS380 token ring controllers to a proprietary high speed bus interface utilizing FPGA technology and high speed FIFOs. The design contained two DMA engines, one that controlled the data flow to and from the TMS380's and the other that controlled the data flow to and from the bus interface. The data within the FIFO was monitored by the use of multiple address descriptors and transfer counters within the FPGA.

During the course of the design process, several FPGA vendors were considered and implemented based on the speed, size and architecture of their product offerings. Additionally, there was a future consideration of converting the design to a conventional gate array. By carefully writing scripts files and a "Verilog Wrapper" the core Verilog code itself was "untouched"(i.e., vendor independent) while taking optimal advantage of the architecture offered by the various vendors.

The design process itself had three steps to achieve the desired target. First, the core Verilog code was written and simulated at the behavioral level to ensure circuit functionality. Once this process was completed this would become the baseline that would remain unchanged throughout the design process to ensure vendor independence. Second, Synopsys script files were written based on the vendors architecture to keep reasonable constraints on parameters such as clock period, area, clock trees and internal delays. These constraints would keep logic levels and excessive fanout in check. Third, a "Verilog Wrapper" was written around the Verilog core to take advantage of architectural differences in the pad logic offered by the various vendors.

At the beginning of the design cycle it was estimated that the target design would need approximately 8000 gate array gates. A gate was agreed upon to be a four transistor two input NAND gate (gate array equivalent) and not a FPGA equivalent gate. The desired clock speed of the design was 16 MHz. The toolset for the design was Verilog along with Synopsys FPGA compiler version 3.2 all running on a Sun Sparc 20 workstation. The first FPGA chosen was a Xilinx XC4025 based on its vast amount of gates with additional benefits of re-programmability and on-board RAM. After working on the design for several months with help from the vendor's application staff, it was realized that the implementation of this particular design would only yield a clock speed of about 12.5 MHz.

The benefits of this design flow were realized when a second FPGA vendor was selected. As stated before, the short design cycle in this highly competitive market required that the vast amount of work done on the first FPGA be transferred to a second FPGA as seamlessly as possible. The second FPGA chosen was the Actel ACT 3 A14100 because of its fine grained and "Synthesis Friendly" architecture. The Actel part had an additional benefit of flip-flops in the pad ring. The I/Os were hand instantiated ("Verilog Wrapper") to take advantage of these features. The Synopsys script files also needed to be modified to take advantage of technology specific features like various clock drivers and inter-connect delays.

When the core Verilog was about to be written, the choice was made to use Verilog because at a high level of abstraction the user can conceptually design a system without regard to a specific technology. There was also the future consideration of turning this design into a gate array when the volumes ramped up and it was desired to keep the core Verilog as stable as possible. During this design it was discovered that it was not necessary to select the target technology before the system design was fully functioning and simulated through high level Verilog simulation. The desire was to avoid technology specific code. This would allow the design to be migrated from one technology to another without core Verilog changes. The core Verilog was written as generic as possible in hope that the design tools would ultimately make smart technology specific choices.

In general, technology mappers want one thing—small homogenous building blocks. It is for that reason that most ASICs are mapped reasonably well when it comes to speed

Paper published in the 1995 PLD Conference Proceedings

and density. The basic building block is typically an equivalent of a 2 input NAND gate. Also, the ASIC interconnect is a metal to metal "via" that doesn't represent a significant amount of circuit delay. Therefore the mapper can produce less than optimal solutions and the ASIC technology will be much more forgiving. FPGAs, on the other hand are much less forgiving. This FPGA design pushed both the speed and density envelope and required two additional steps: The first was the Synopsys script file, the second was the hand instantiation of the complex I/Os.

The power of the Synopsys compiler is it's flexibility through scripts. The core Verilog code need not be modified to change a design from a small, medium speed, compact design to a much faster but larger design. This was accomplished through design constraints. In this design two types of constraints were set for each of the chosen technologies: "design rule" and "optimization." In general, "design rule" constraints reflect technology-specific restrictions that must be met for a functional design (such as maximum loading on a net). "Optimization" constraints represent design goals that are desirable, but not crucial to the operation of a design (such as maximum circuit area or delay).

The Synopsys Compiler tries to meet both types of constraints with an emphasis on "design rule" constraints as they are requirements for a functional design. The Synopsys Compiler uses these constraints to guide optimization and implementation of a design. Constraints define the goals of the synthesis process. The constraints for the technology-specific target can be put into a script file to help keep the core Verilog code untouched.

The amount of constraints that can be put into a Synopsys script are virtually unlimited. However, with the described design task in hand, the following assortment of constraints were found to be quite useful. The "max_area" constraint specifies the maximum allowable area for the current design. When the Synopsys Compiler sees this "max_area" constraint it computes the area of a design by adding together the areas of each of its components on the lowest level of the design hierarchy. The area of a cell is obtained from the specific technology file. The constant put after the "max_area" constraint represents the total amount of cells available in the target part.

The most important optimization constraint is maximum delay (max_delay). There are four types of delay categories: Clock to Q (Tcq), Set Up (Tsu), Clock to Out (Tco), and In to Out (Tio). Since the I/Os were hand instantiated, the only delays of concern were internal synchronous paths (register to register). The "max_delay" constraint was carefully set so the longest path from clock to out of one register, through logic to the input setup time of the next register was less than the target clock period of 62.5 ns (16 MHz). The Synopsys Compiler has a built-in static timing analyzer for evaluating timing constraints. A static timing analyzer calculates path delays from local gate and interconnect delays but does not simulate the design. That is to say, it does not check the design for functionality. The Synopsys Compiler timing analyzer performs critical path tracing to check minimum and maximum delay for every timing path in the design.

The last constraint that was found useful was the "dont_touch" constraint. FPGAs have special features like high drive resources (e.g., CLKBUF macro) used for driving high fanout clock nets and special complex I/Os that were hand instantiated in this design. To prevent the Synopsys Compiler from breaking up and adding buffers to the clock net the "dont_touch" constraint was added to this net. After instantiating the I/O cells, the "dont_touch" constraint was added to keep these elements intact.

The final step of the design process was to hand instantiate the complex I/Os. The idea of this final step was to hand instantiate the I/Os and wrap them around the simulated core Verilog. This is known as a "Verilog Wrapper." When it came time to change technologies, it was a simple matter to change "wrappers". The I/O ring was manually instantiated and connected to the top-level of the core Verilog code. When the "Verilog Wrapper" was written, "Direct Instantiation" of the complex I/Os was used and the port order was automatic. For example, when it came time to instantiate a high speed I/O flip-flop like the BRECTH cell (shown in Figure 1) from the Actel ACT 3 family, it was instantiated as follows:

```
BRECTH
I2(.D(<n1>),.PAD(<p1>),.E(<n2>).Y(<n3>),.
CLK(n5>),.IOPCL(<n6>));
```

When all the I/Os were instantiated, the "dont_touch" constraint was added in the Synopsys script file.

When all the above mentioned steps were performed, the design was smoothly transitioned from one FPGA technology to another and the desired speed of 16 MHz was achieved. The next step of this process is to convert the design to a conventional gate array. Although this process has not yet begun, it is certain that by following the process described here, the transition will go smoothly.

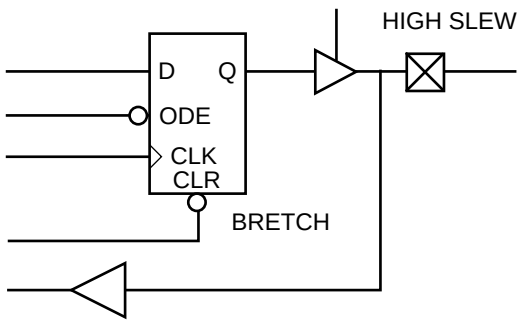

Figure 1 • *BRETCH Actel ACT 3 I/O Macro*