WP163 (v1.1) July 22, 2002

Synthesis Tool Enhancements for Virtex Architectures

(Performance, Mapping, and Back-End Tool Integration)

By: Philippe Garrault

With the advent of platform FPGAs, programmable logic circuits provide an ever-growing set of architectural elements. Days when FPGAs were simply made of LUTs and flip-flops are long gone. Now FPGA fabric is "feature rich" and flexible. Sophisticated synthesis tools are required to further improve performance and logic utilization using these new architecture capabilities. This document presents some critical areas where Xilinx is actively working with synthesis partners to improve quality of results (QoR) while, at the same time, keeping design and verification processes as simple as possible.

Recent Advances in Synthesis Mapping for Virtex Architectures

Synthesis tools originally derived from the ASIC world and therefore first concentrated their efforts on minimizing and optimizing equations extracted from HDL source code. That code was then converted to gates and finally to LUTs. Today, FPGA synthesis tools are operating at higher levels of abstraction. First, they need to parse through the HDL behavioral source code and extract known functions (e.g., arithmetic functions, multiplexers, memories, and so on). Synthesis tools then map these functions by applying a recipe tightly dependent on the target architecture features and context. The context influences inter function optimizations such as replication, merging, re-timing, and pipelining. The context is defined by timing requirements and by the topology of the design.

For a synthesis tool, the key in delivering high-performance circuits is to thoroughly understand the target architecture. Xilinx works closely with its synthesis partners, constantly looking into new optimization techniques that go beyond mere LUT mapping by examining ways to integrate the more and more sophisticated FPGA resources to regular synthesis mapping. Some specific resources include RAM, multipliers, dedicated blocks and circuitry for storage elements, clock management, arithmetic functions, muxes, multiplier, shifters, and so on.

Figure 1 illustrates a loadable counter (an enable signal controls the counting or the output registers loading operation). At the Register Transfer Level (RTL), when parsing through HDL source code, synthesis tools model that functionality with a counter followed by a multiplexer controlling the load function. Previously, it was mapped the same way into the target device. However, a closer look at the Virtex™-E and Virtex-II architectures reveal MULT_AND cells. Taking advantage of the MULT_AND cells allow merging of the counter and MUX functions, resulting in a 50% area reduction (LUT) and 30% performance improvement (fewer levels of logic). The Mentor Graphics Leonardo Spectrum supports this optimization process.

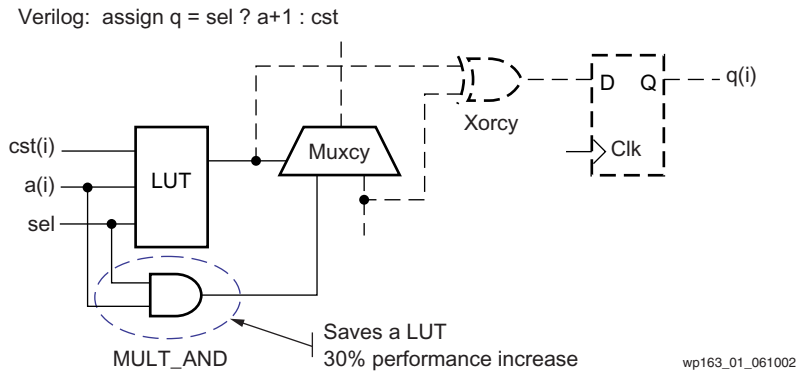

Figure 1: **New and Improved Mapping for Loadable Counter**

Virtex-II block multipliers is another example that shows how providing extensive details of FPGA architectures have enhanced synthesis tools QoR. Synthesis tools algorithms now have built-in intelligence allowing them to make decisions on multiplier mapping (distributed or embedded cells), multiplier decomposition (for larger than embedded multiplier cells) multiplier pipelining (to maximize performance), and so on.

Table 1: **Using Embedded Resources Improves Performance and Device Utilization**

16 bits by 16 bits	Distributed Multiplier	Block Multiplier	Pipelined Block Multiplier
Performance (MHz)	68	96	204.8
Resources	277 LUTs	0 LUTs (1 MULT18X18)	0 LUTs (1 MULT18X18S)

These capabilities are embedded into the tools (Leonardo Spectrum, Synplify), and are transparent to designers. This ensures optimal performance without requiring the designers to have a comprehensive knowledge of the targeted architecture or forcing code tweaks for a particular device. Designers can focus their efforts more on design development than on the tools or target device features.

Sharing Back-End Tool Knowledge with Synthesis Tools

As FPGAs go deeper into submicron technologies, routing delays become more predominant (over cell delays.) Therefore, the influence of cell placement is increasing rapidly. The traditional fanout-based wireload models, for estimating interconnect delay during synthesis, are now considered inaccurate and are a key reason for the lack of timing predictability between post synthesis and post layout results. Synthesis and placement technologies must share more information to help close the timing loop. Xilinx is exploring physical synthesis with its different synthesis partners including Synplicity's Amplify Physical Optimizer (Figure 2). From the early development stages of Amplify physical synthesis algorithms, Xilinx has evaluated the tool and flows with actual customer designs. This provides Synplicity with valuable feedback over performance, resource utilization and tool ease of use. As a result, designers see performance improvements over traditional synthesis flows. Typically, Amplify's standard interactive flow provides +25% improvements; the Interactive Total Optimization (ITOPS) mode provides an additional 10% improvement. Amplify in Automated Total Optimization (ATOPS) mode provides an average +8% improvement. ATOPs is totally transparent to designers because it does not require manual interactions to physically constraint the design.

As a result of this co-operation, Synplicity is able to support Xilinx technologies in its leading edge products prior to any other PLD technology.

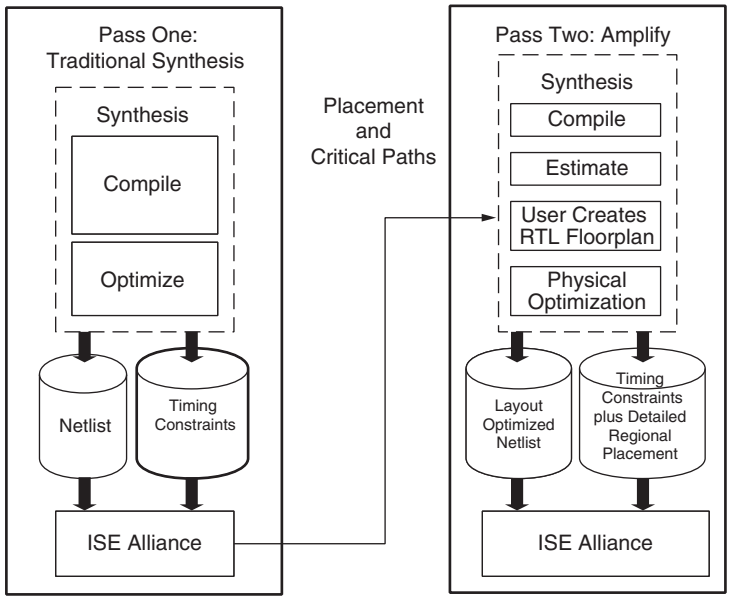

Figure 2: **Physical Synthesis Flow**

Synthesis Tools Integration in the Overall Design Flow

Design size expansion and complexity have increased the number of tools used in FPGA design. Besides synthesis and simulation tools, a new generation of tools is making its way into the FPGA design flow: equivalency checking. How does a designer ensure portability of the HDL code through synthesis, simulation, and the new verification tools? All use different libraries, attributes, and directives. Xilinx addresses this issue by supporting the homogenization of synthesis, simulation, and verification coding styles and libraries.

For instance, synthesis tools supported and documented coding styles for RAM inference in Verilog and VHDL has been changed to match the behavior of the Virtex block RAMs (including modes for simultaneous read and write conflict resolution.) With this change, the exact same code runs through synthesis, simulation, and formal verification tools.

Correct Coding Style

The following coding style is portable into synthesis, simulation and equivalency checking tools because it matches the hardware behavior.

```
// Correct coding style:
//

always @(posedge clk)
   if(we) begin
     q <= d;
     mem[a] <= d;
   end else
     q <= mem[a];
```

The example below shows the previously documented coding style. RAM addresses are registered, inducing a possible mismatch between simulation or equivalency checking tools because it does not match the Virtex block RAM hardware behavior.

```
// Previous documented coding style
//
always @(posedge clk) begin
   if(we) mem[a] <= d;
     read_addr <= a;
end

assign q = mem[read_addr];
```

Conclusion

While Xilinx aggressively pushes both device and software performance limits with the Virtex-II architecture and ISE, it also collaborates closely with third-party vendors to seamlessly make improvements available to designers.

As a result of this joint effort, Xilinx customers are the first to benefit from improved synthesis flows and advanced algorithms that fully utilize silicon features. Therefore, designs achieve the most efficient and best performance with lower logic utilization.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
07/10/02	1.0	Initial Xilinx release.
07/22/02	1.1	Added "Counter" to Figure 1 caption.