

Re-timing for Performance Improvement in FPGA Designs

By:
Douang Phanthavong
Technical Marketing Engineer
Mentor Graphics

April 2003

1. Abstract

Retiming is an optimization technique for synchronous circuits introduced by Leiserson and Saxe in 1983 [1]. Since then, the retiming concept has not been widely used and followed-up. It is fair to conclude why this powerful retiming algorithm has not been widely used in the logic synthesis tools until recently due to the following issues:

- 1) Retiming tends to increase area utilization, which was very costly in early ASIC and programmable logic.
- 2) Bandwidth has not always been an urgent need for implementing a typical design.
- 3) Limitation of applying retiming algorithm due to different type of registers with complex control signals.

However, in the recent years, designers have been demanding faster and higher bandwidth. Bandwidth has become the most common design bottleneck in network and telecom systems and it has continued to be until now. At the same time, the designs have become more complex and sophisticated. One of the most compelling technologies to apply this proved effectiveness of retiming algorithm is on Field Programmable Gate Arrays (FPGA). In this paper, we present the following facts:

- 1) How do FPGA synthesis tools implement and utilize retiming functions.
- 2) Why is FPGA a more suitable target technology for retiming algorithm than others?

How is retiming being integrated with today's FPGA synthesis flows?

2. Introduction

Retiming is an intelligent process of moving and balancing registers backward and/or forward across combinatorial delay paths to obtain an optimum timing while maintaining the functional behavior of the circuit. As describing by a well-known author, Leiserson and Saxe [1], retiming algorithm can find an optimum solution for clock period when the circuit is timed by one clock on the same edge without considering interconnect delay. A flip-flop can be moved from each incoming edge of a combinatorial component to each of the outgoing edges of the component. Such a move can reduce the critical path delay associated with the flip-flop. However, it can move across only one combinatorial component each time, so the timing improvement offered by this algorithm is very limited. Also there are numerous limitations concerning interconnect delay and packing rules for each specific FPGA architecture technology. The current retiming algorithms developed by today's FPGA synthesis tools are working towards solving and enhancing these limitations to obtain a much better timing performance.

Pipelining is another optimization technique that involves partitioning logic into stages so that the first stage can begin processing new inputs while the last stage is finishing the previous inputs. This ensures better throughput and faster circuit performance. For pipelining, some synthesis tools may actually introduce more registers in a cycle or delay path from a primary input to primary output pins therefore adding clock latency cycles to the designs.

3. How do FPGA synthesis tools implement and utilize retiming function?

Retiming and Pipelining are powerful techniques, which are widely used in today's FPGA synthesis tools to improve the timing performance of sequential circuits. Pipelining can be intelligently used to balance and move registers from primary I/O into partitioning logic without introducing any extra clock latency to the original design. With a well thought-out integration process between retiming and pipelining, it can prove to be a very effective and efficient technique for FPGA synthesis tools to obtain the best timing performance possible. Some of these terminologies are commonly used in retiming.

Forward retiming is a process of moving registers forward across combinatorial paths. In Figure 1.0, forward retiming would move REG_IN forward across either one of the combinatorial operators depending on the timing slacks and constraints.

Backward retiming would be just in opposite of forward retiming, a process of moving registers backward across combinatorial paths. In Figure 1.0, REG_PL1 would be moved backward across either one of the combinatorial operators depending on the timing slacks and constraints.

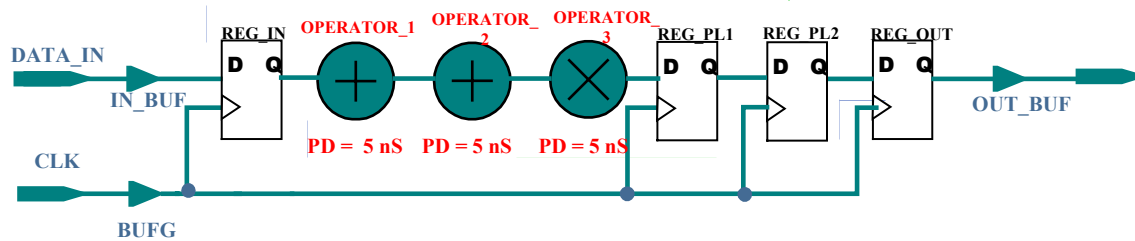


Figure 1.0 Original circuit before retiming

In Figure 1.0, the circuit can be pipelined by hand-coding REG_PL1 and REG_PL2 into the RTL behavior. It may be feasible to hand-code pipeline registers into either adder or subtractor operators, but it is not for a multiplier. Even, if they all can be hand-coded, it still would be much more efficient to have the tools perform this job automatically. In the first scenario, if the retiming engine can only perform forward retiming; the original circuit can be improved by 5 ns (from 15 to 10 ns). Second scenario, if the retiming engine has a capability to move registers forward and backward, the timing can be improved from by 10 ns (from 15 to 5 ns). The point is that depending on the retiming engine's capabilities, the final timing results can be quite substantial.

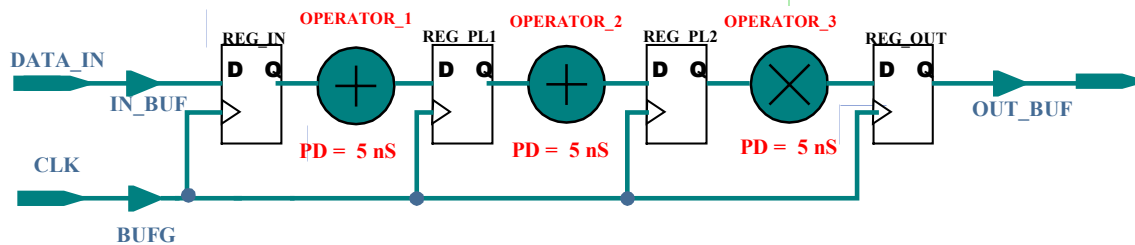


Figure 1.1 Circuit after retiming and pipelining

For a good retiming engine, the original circuit from Figure 1.0 would transform into the circuit in Figure 1.1. The timing is improved by 10 ns (15 to 5 ns) while ensuring the original I/O behavior. One might even be able to take Figure 1.1 and improve it even more. This can possibly be enhanced by taking a cut inside one of the operators instead of in between them.

Because a FPGA provides such a register rich architecture, it is more suitable and applicable to utilize retiming technique to improve the timing performance on sequential circuits. Some of the recent retiming techniques being implemented by FPGA synthesis tools improve abilities to qualify more sequential circuit elements for retiming functions. Some of the current disqualified sequential circuits for retiming may be varied depending on certain FPGA synthesis tools. In general, registers with different clock enable (CE) lines, registers with uncommon set/reset signals or registers with complex control signals are disqualified for retiming.

Some techniques of a new approach called multiple-class retiming presented by Klaus Echl, Jean Christophe Madre, Peter Zepter, and Christian Legl [2], can be realized and implemented very efficiently in today's FPGA synthesis tools. As shown in Figure 1.2, transforming the registers that are not qualified for retiming to those that can be retimed, is one of the techniques being considered.

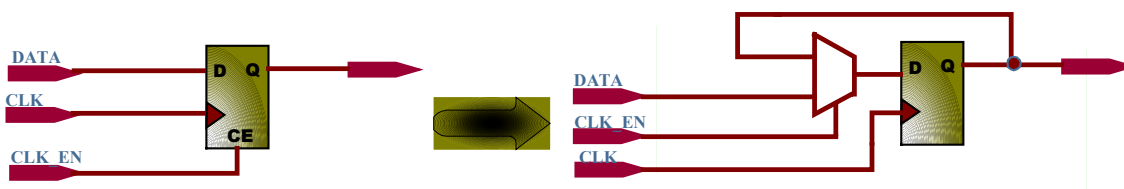


Figure 1.2 Flip-flop transformation

In most of today's FPGA architectures, delay paths can come from resources such as combinatorial circuits, which are basically mapped to LUT (Look-Up Table) or from the dedicated resources such as memories, multipliers and any non-sequential cells. A look-up table with K-inputs and single output is capable of implementing any arbitrarily defined Boolean function of K-inputs. Most combinatorial circuits are implemented using these LUT functions. Dedicated fast look-ahead Carry Logic is often provided for fast arithmetic operations.

The effectiveness and efficiency of the FPGA synthesis tool to retime paths consisting of these different logic blocks is entirely up to its retiming algorithm. This is where the retiming challenges really present themselves to the synthesis tools. As previously mentioned, the final results can be quite significant, depending on how well the retiming algorithm was implemented.

In FPGA synthesis tools, arithmetic operators can be implemented using LUT only or LUT based with Carry Logic or using dedicated resources such as block multiplier or dedicated DSP block.

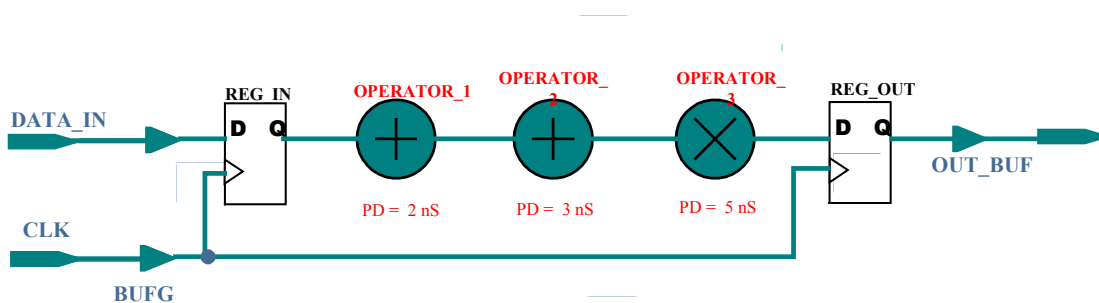


Figure 1.3 Circuit before retiming

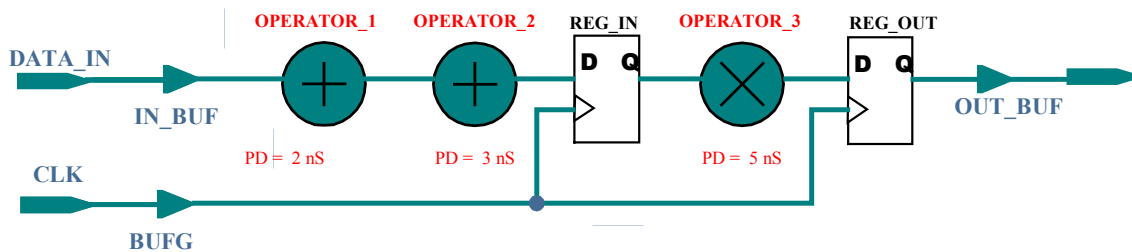


Figure 1.4 Circuit after retiming

In Figure 1.3, assume that OPERATOR_1 is mapped to LUT-only logic, OPERATOR_2 is mapped to LUT and carry logic and OPERATOR_3 is mapped to a designated block multiplier. From this simple illustration, the most effective location to retime is between OPERATOR_2 and OPERATOR_3, which will reduce the register-to-register timing from 10 ns to 5 ns. In the first scenario, if the synthesis tool does not have a capability to retime across operators, this circuit would not be retimed at all. In a second scenario, if a synthesis tool cannot retiming any operator that is implemented by LUT and Carry Logic, then OPERATOR_2 would be ruled out from retiming. It is very imperative for the retiming engine to maximize and extend its qualified circuits for each specific technology. Another important feature that a retiming engine must have is an accurate and reliable timing characterization routine. Without this, retiming could possibly worsen the original circuit's performance. As illustrated by Figure 1.3, the ideal retiming would improve the register-to-register timing by 100 percent granted that both input and output delay constraints are greater than 5 ns.

4. Integrating retiming in FPGA synthesis flows

The retiming engine can be integrated into the FPGA synthesis tools in different phases depending on how the retiming algorithm was implemented. From Figure 1.5, option 1, retiming was performed by using the elaborated netlist. Some synthesis tools have developed and integrated their retiming engines using this option flows. It is not very efficient for FPGA architecture technology due to such proprietary mapping and packing rules for each FPGA vendor. Another big advantage by using this elaborated netlist is the timing characterization can be changed quite drastically by the timing the design reached to mapping phases.

Option 2, on another hand, is quite common among the FPGA synthesis tools because retiming is performed after the design was optimized and mapped. The retiming engine can utilize the mapped netlist and perform retiming as necessary. It can take full advantages of both the logical and physical retiming algorithms where possible. Timing characterization at this phase is far more accurate due to already mapped netlist by a specific target technology.

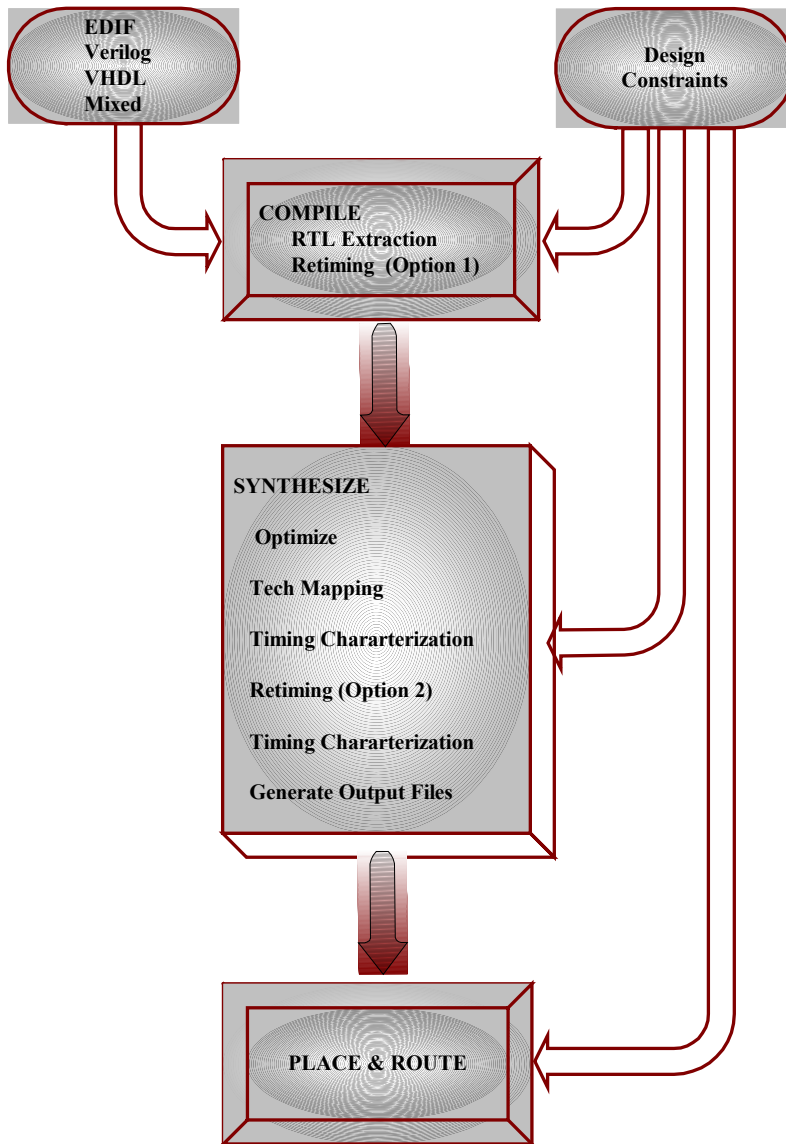


Figure 1.5 General FPGA Synthesis Design Flow with Retiming Engine

5. Summary

Retiming has proven to be one of the most effective and reliable techniques to improve timing performance while ensuring the design behavior. Due to its register-rich architecture and other versatile features, it is reasonable to conclude that FPGA designs can benefit significantly from retiming routines.

Because FPGA design can be technology specific, the retiming algorithm engines are faced with many challenging tasks. Regardless of all these challenging obstacles, retiming development has continued to progress forward very effectively.

Every retiming algorithm can provide a different quality of result depending on how it is developed and implemented. Precision RTL Synthesis and Precision Physical Synthesis tools from Mentor Graphics have a significantly advanced patented retiming engine that combines optimization techniques of register replication, pipelining, and retiming (backwards, forwards, across and in between operators), to help the designer achieve a much better design performance with minimal impact on the design cycle time.

With a recent advance retiming development by Mentor Graphic Synthesis tool, a new release of Precision 2003a, the timing performance can be improved up to 50%. This is by far one of the most impressive gain for the retiming algorithm has ever developed for the FPGA Architecture Technology.

Reference:

- [1] CE Leiserson and JB Saxe. *Optimizing synchronous systems*. Journal of VLSI and Computer Systems. 1(1): 41-67, 1983.
- [2] CONG, J., AND WU, C. 1996. ... 7 Jason Cong, Chang Wu, *FPGA synthesis with retiming and pipelining for clock period minimization of sequential circuits*. UCLA-CSD 970011, Technique Report, March 1997.