Microsemi
Power Matters.™

Timing Closure on Microsemi SmartFusion2 SoC, IGLOO2, and RTG4 FPGAs

WP0202 White Paper

January 2016

Timing Closure on Microsemi SmartFusion2 SoC, IGLOO2, and RTG4 FPGAs

Introduction

Timing Closure has become one of the major issues for the Field Programmable Gate Array (FPGA) designers to face when they integrate multiple blocks at the top-level design. Timing Closure issue can easily be fixed, if the root cause or causes are identified and the Timing Closure iterations follow a clear and disciplined methodology. This white paper lists the potential causes of Timing Closure challenges and provides high-level guidelines for the methodology.

This paper is organized in the following three sections:

- Sources of Timing Challenges—lists the potential sources of timing challenges.
- Solving Timing Challenges—goes over some tips to cope with the root causes of timing challenges.
- Methodology Summary—provides additional guidelines when more than one source of timing challenges is identified.

Sources of Timing Challenges

Timing challenges are caused by one or more of the followings:

- High Fanout Logic Nets
- Large Number of Logic Levels
- High Utilization
- Inherent Routing Congestion
- Large Number of Clock Domains
- Inadequate Synthesis, Place-and-Route Settings, and Constraints

High Fanout Logic Nets

Clocks and resets are not considered as logic nets, and the logic nets must be mapped with high-priority using the FPGA clocking resources. Logic nets with high fanout are the result of a common boolean function driving many other logic blocks. When the number of high fanout nets is large, the high fanout net routing can be sub-optimal and can lead to a high delay penalty when the load or the cells driven by this net are scattered on the die. The delay penalty might also be caused by a non-optimal routing when there is a competition among many logic nets for routing resources.

Large Number of Logic Levels

Paths with large number of logic levels are caused by an inherent structure of the design, a poor synthesis, or a mapping. The inherent design with large number of logic levels occurs when large boolean expressions (many terms and variables) or arithmetic operations are concatenated.

Poor synthesis can be caused due to the lack of proper timing constraints, inappropriate state machine encoding, or inadequate Synthesis options such as resource sharing that usually leads to use of mutliplexors at the ports of arithmetic or logic operators.

High Utilization

When the use of logic resources, Look-Up-Tables (LUTs), or embedded resources such as DSP, RAMs, and so on is above 90%, there is a chance that the placement is sub-optimal for a part of the design, and a higher chance to have poor routing because of fight for routing resources between the timing critical logic nets.

Inherent Routing Congestion

Routing congestion can be the result of a large switch matrix, large multiplexors (8 or 16 to 1 with bus width of 8, 16 or 32 bits), or data dependencies between several blocks and modules of the design. For example, when a block interface with many other blocks through various large buses, there is a risk of the logic paths crossing many of these blocks, the placement of the interface block leads to a stress on routing causing unnecessary starvation for routing resource, or congestion.

Large Number of Clock Domains

In itself, the number of clock domains can be easily accommodated by the fragmentable clock networks. All three Microsemi FPGAs such as SmartFusion®2 System-on-Chip (SoC), IGLOO®2, and RTG4™ have fragmentable clock networks. Timing challenges occur when several of these clock domains require high frequency for each or when the I/O assignment has been locked without further investigation of the chip clocking scheme. An example of the first case occurs when some data or control input I/Os placed on one side of the die drive a logic that ends on a RAM block that interfaces the output I/Os placed on the opposite side of the die.

Inadequate Synthesis, Place-and-Route Settings, and Constraints

One of the major sources of timing challenges is the quality of the logic net list generated by the synthesis tool. In many cases, poor mapping is due to the lack of appropriate constraints settings and synthesis options. Timing constraints must be set appropriately, when multiple clock domains are present in the design. Setting the highest frequency as a global constraint for all the clock domains while some of these clocks run at a lower frequency is a major mistake. It simply leads to a larger resource utilization for the blocks driven by lower frequency clocks, routing congestion, and potentially to a conflict during optimization. Poor setting of synthesis options such as state encoding, resource sharing, and maximum fanout before buffering or replicating can be another source of timing challenges.

Place-and-route engines are heavily impacted by timing constraints. If the timing constraints are not set appropriately, the timing challenges can be discovered during the timing analysis phase. Other physical constraints such as I/O assignment, regions allocation for blocks must be timing-aware to avoid timing violations.

Solving Timing Challenges

Before getting into this exercise, you must ensure that all the timing constraints such as clock frequencies and clock input and output delays are complete and correctly set. Additionally, SmartTime, the Static Timing Analysis (STA) Tool, has basic defaults for capacitance on I/Os, and so on. You must update these default values based on the actual board knowledge.

This section describes on how to assess the difficulty of the timing closure exercise, the identification of the root cause, and how to solve it methodically.

How Difficult is a Timing Challenge?

There is a simple way to find out if the timing challenge is easy, moderate, or difficult to solve. The FPGA designers must extract the slack distribution chart for each clock domain. The slack distribution chart shows the slack (negative or positive) of the 20 to 50 most critical paths within the clock domain. Identifying the paths with positive slacks is as important as identifying the ones with negative slack, that is, with timing violations.

Figure 1 shows an example of slack distribution charts for three clock domains—*sys_clock*, *mac_clk* and *house_keeping_ck*. The *sys_clock is* the most critical clock domain as it has the highest frequency and has seven paths grossly violating the 7.5 ns delay. It is also important to notice that the *house_keeping_ck* is critical as it has five paths that meet the relatively relaxed timing of 20 ns. Any disturbance to one of these five paths might lead to timing violations. Finally, the *mac_clk* has enough positive slack even for the first 15 paths. Minor changes to the design might not disturb the timing closure on the *mac_clk* domain.

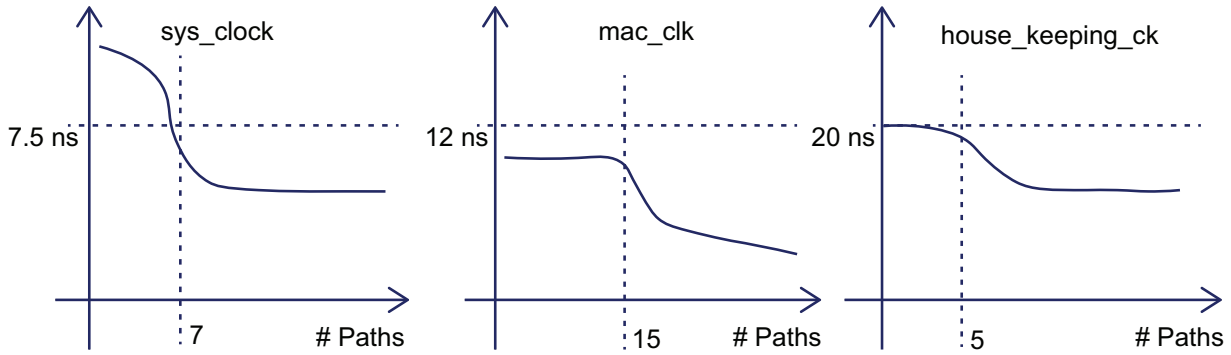


***Figure 1* • Clock Domains and Slack Distribution Charts**

The timing challenges of the design is shown in Figure 1 are not difficult to fix in a couple of iterations. Even the violations for *sys_clock* are high, the designer can improve these by focusing on this clock domain, making sure that *house_keeping_ck* is untouched or slightly improved, and using the margin offered by *mac_clk* domain. To illustrate Figure 1 more, one option is to toughen the Synthesis, and place-and-route frequency constraints, put 150 MHz constraint on *sys_clock* and 60 MHz on *mac_clk*, and relaxing these for *mac_clock* to have 50 MHz.

How to Analyze the Timing Violations and What to Do about Them?

Once the complexity of the timing issues is assessed using the slack distribution charts, and the trouble clock domains are identified, and then a zoom into the violations in each clock domain must take place. The first focus must be answering the following questions:

- Are these paths with negative slack (or timing violations) false paths or multi-cycle paths?
  - **Yes**, the false paths or timing violations can be filtered from the timing report by adding the appropriate constraints.
  - If **No** is the answer, the designer must proceed to the next question.
- Is the clock skew larger than 0.5 ns?
  - **Yes**, the clock is routed using regular routing resources, not a clock network. It is easy to fix by mapping the clock to a global routing or a segment of the global network depending on the fanout of the clock.
- Are the paths with timing violations confined within one logic block? do they cross multiple logic blocks or hierarchical levels?
  - If the paths violating the timing specification are within a module or a logic block, fixing the Synthesis setting for that particular block and setting harder timing constraint on the particular block must be sufficient to get the timing closure.
  - If the paths have not meet the timing specifications cross several blocks, the stringent timing constraints on the blocks might not be enough to solve the issues. Flattening one or more of the blocks, using the attribute /* *syn_hier* */ combined with a stringent *max_delay* timing constraint for the Synthesis tool must be sufficient to close the timing.

- Are there any high delay penalty nets? Are these nets common to a group of violating paths? If yes, what are the fan outs associated with each of these common nets.
  - If nets with a serious delay penalty and high fanout are identified to be common to some of the paths with violations, the issue can be resolved by simply setting lower *max_fanout* on these nets in the Synthesis constraints or using HDL attributes such as /* syn_max_fanout */ to less than the fanout reported in the compiled report. For example, if a particular net contributes more than 2 ns+ to a set of paths and its fanout is 150, setting the *max_fanout* to 50 solves the issue.
  - If the nets with delay penalty have less than 16 fanout, there must be either a placement of the driver and the load cells, or a routing issue. The designer can fix the issue by using the tighter timing constraints on these paths and re-running the place-and-route.
- Are there paths with number of logic levels larger than 5 or 6?
  - **Yes,** it is better to either slightly modify the RTL code or set a more stringent timing constraint on these paths during Synthesis. If a stringent constraint does not lead to lower number of logic levels, you need to enable the re-timing option during Synthesis with even more stringent timing constraint. This must lead to a more balanced logic and registers, which lowers the number of logic levels on each side of the registers.
  - **No,** it requires an investigation of the RTL code associated with these paths as described in the following bullet.
- Are these paths part of state machine logic?
  - **Yes,** the designer must investigate the coding style of the state register. In general, if the paths are related to the state machine output, the solution is to set /* *syn_state_machine* */ to one hot. If the path involves the next state logic, then either a compact code such as Gray code or even disabling of the state machine inference for that part of the RTL code.
- Are these paths involving RAMs?
  - **Yes,** the RAM output port is the source of the path, two things can be considered. The first is to instantiate RAM from the Libero SoC catalog in the RTL code instead of relying on the Synthesis RAM inference. The second is to find out if using a pipeline RAM is functionally viable.

    The RAM is the end point of the path, using Libero SoC catalog instances helps if the setup timing violation is less than 0.5 ns. If larger negative slack is observed, you are advised to check the placement of the logic vs. the placement of the RAM or make the paths to the input port of the RAM critical by setting maximum delay constraints with the to targeting the RAM inputs.

Exploring Block Flow

The block flow is another option to explore and can be used to close on timing. It is a bottom-up design methodology, which enables the designer to use design blocks as building block for the top-level design.

The main advantages of using the block flow:

- It focuses on the timing of critical blocks and ensures the timing across the blocks to meet the requirements before proceeding to integrate the blocks at the top-level.
- Changes in other blocks does not impact the designer's block; the designer can re-use the block without re-optimizing for timing closure.
- The same block can be reused in multiple designs.
- Shorter verification time; the designer must re-verify only the portion of the design that has changed.

This allows the flexibility of focusing on each block so each block has predefined place-and-route that is optimized for timing performance. Using blocks for parts of the top-level design can cut down design time, improve timing, and power performance for the design. The designer can focus on optimizing timing on a block per block basis. After a block is optimized, the designer can publish the block along with its constraints.

Methodology Summary

Step 1: Let us not solve what is not an issue

If the timing constraints do not reflect in the actual timing requirement for the application, the timing violations might not be real. In other words, the designer must revisit the timing constraints they set for the Synthesis flow. The ones for the place-and-route tools, and ensure the ones used during STA are the correct ones.

Step 2: Derive the Slack Distribution Charts

These charts are eye openers. The designer might spend countless iterations unnecessarily without them.

Step 3: Spend quality time to analyze the timing violations for the first 10 paths

This thorough analysis helps identifying the common root causes that are impacting these paths.

Step 4: Follow the list of questions and tips provided in "How to Analyze the Timing Violations and What to Do about Them?" on page 4

Step 5: Do not solve one problem to create 2 or more

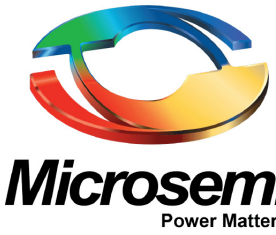
In other words, estimate the impact on other clock domains of the corrective actions you are taking to tackle the timing issues for another clock domain.

Step 6: Explore the Block flow

Explore the option of using the Block flow in "Exploring Block Flow" on page 5, especially, for bottom-up design methodology.

Conclusion

This white paper has discussed the various root causes for potential timing closure challenges when targeting FPGAs. It also proposes a methodology to deal with each or a combination of these root causes based on a proper analysis of the slack distribution chart of the various clock domains. Several systemic and methodical techniques are proposed to cope with the timing challenges and meet the design performance requirements. The timing convergence can be achieved by investigating the quality of RTL coding, the clocking schemes, and the proper timing constraints. Finally, if timing challenges are local to a particular block of the design, the block flow supported by the Libero SoC integrated design tool ensures the timing closure and repeatable results when designing on SmartFusion2, IGLOO2, and RTG4 Microsemi FPGAs.



Microsemi Corporate Headquarters
One Enterprise, Aliso Viejo,
CA 92656 USA

Within the USA: +1 (800) 713-4113
Outside the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996

E-mail: sales.support@microsemi.com

Microsemi Corporation (Nasdaq: MSCC) offers a comprehensive portfolio of semiconductor and system solutions for communications, defense & security, aerospace and industrial markets. Products include high-performance and radiation-hardened analog mixed-signal integrated circuits, FPGAs, SoCs and ASICs; power management products; timing and synchronization devices and precise time solutions, setting the world's standard for time; voice processing devices; RF solutions; discrete components; security technologies and scalable anti-tamper products; Ethernet solutions; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Microsemi is headquartered in Aliso Viejo, Calif., and has approximately 3,600 employees globally. Learn more at **www.microsemi.com**.

55900202-1/1.16