Designing for Performance on Flash-Based FPGAs

Table of Contents

Abstract

Recently, the market has seen the birth of new FPGAs that embrace for the first time applications and systems with very stringent timing and power requirements. Novice and experienced FPGA and ASIC designers face similar problems when it comes to achieving timing closure in a very limited number of iterations. This paper introduces the main ingredients for tackling this problem without penalties or frustration. The basic ingredient is a good understanding of the FPGA architecture and its embedded features. The exploitation of these features is the key to coping with several hurdles. The second ingredient is a great analysis methodology of the bottlenecks that can be inherent to the design itself, caused by the tools or by the user settings. The third ingredient is the knowledge and understanding of the tools' underlying techniques and potential optimizations and limitations. To make a sauce of these main ingredients—i.e., quickly converge and meet timing—this paper proposes a methodology using a combination of synthesis tools and Designer, the Microsemi physical design toolset.

Introduction

FPGA designers often face daunting timing challenges. To reduce their frustration and to shrink their design time and effort, this application note is intended to help users analyze their design, identify the root causes associated with the timing issues, and finally cope with them.

One other important goal is to help designers predict the outcome of their design decisions or tools settings. This paper highlights when a particular user action may be efficient and what is the effect of combining several user actions. The document will also cover some of the cautions that need to be considered when dealing with some of the timing-critical situations. User actions described in this document cover the following:

- RTL coding
- Synthesis options setting
- Place and route constraints and options setting

This document is organized in four sections.

- "Brief Introduction to Flash FPGA Architecture" on page 2 presents the salient architectural features of the Flash-based Microsemi FPGAs: IGLOO,® Fusion, ProASIC®3, and ProASIC3L.
- "Root Causes of Timing Issues" on page 5 focuses on the main sources of timing challenges and ways to identify them.

- "Main Sources for Design Analysis" on page 7 introduces various design techniques to cope with each of these root causes of timing challenges. These techniques include RTL coding, synthesis flow setting, place-and-route, and physical and routing constraints.
- "Ingredients for Timing Optimizations" on page 9 focuses on inherent design congestion management and provides various results.

Brief Introduction to Flash FPGA Architecture

This section will only briefly cover some features in the IGLOO, Fusion, ProASIC3, and ProASIC3L Flash FPGA families. For full details, the reader can refer to the corresponding product handbooks.

Basic Cell

The basic cell, called VersaTile as depicted in Figure 1, is a LUT with three inputs that can indifferently implement a three-input combinatorial gate or flip-flop with enable. This allows the implementation of any design with any combinatorial vs. register ratio. The nature of this tile makes it close to basic ASIC cells in the sense of its fine granularity, thus allowing ASIC-like cell-based mappers to apply all their optimization potential.
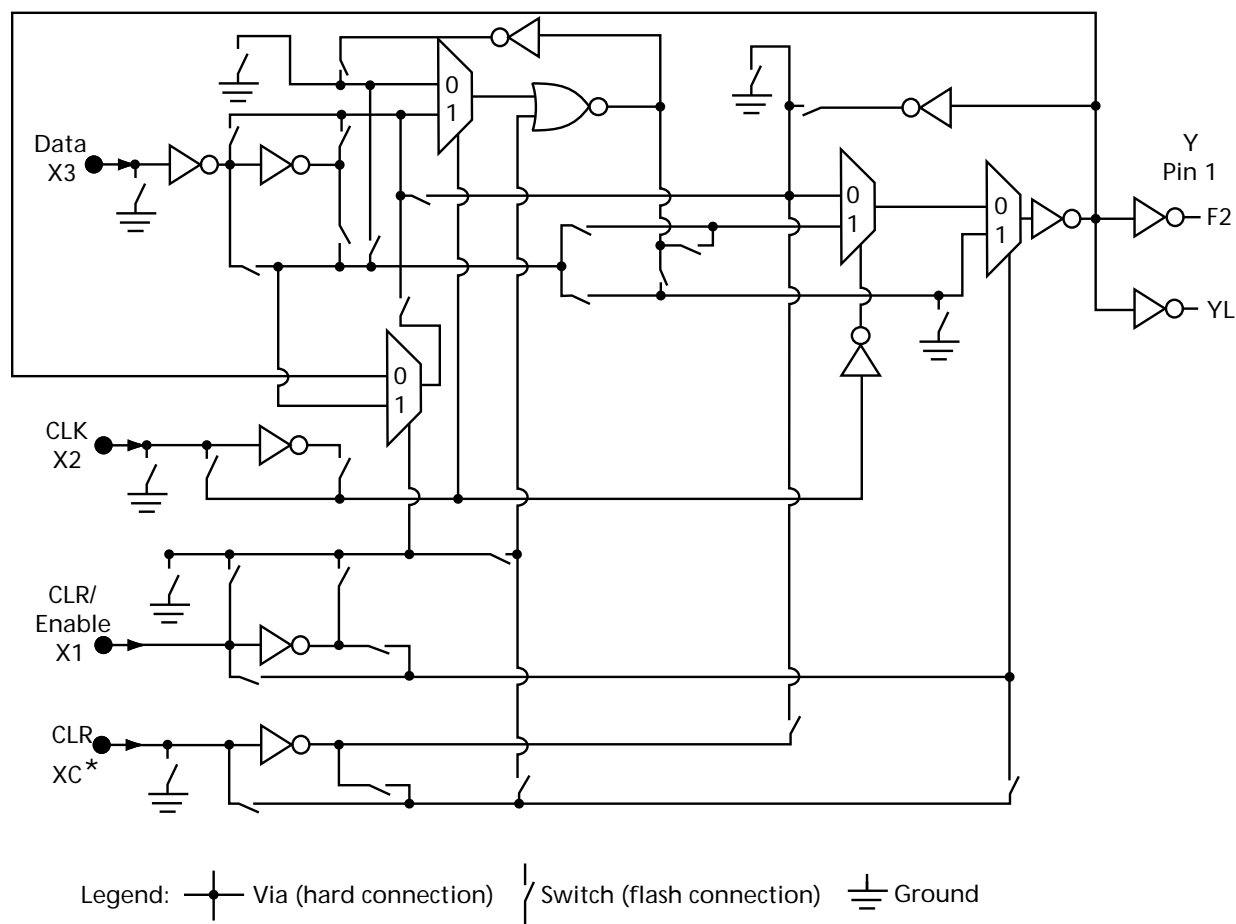

Figure 1 • **Logic Cell or Versatile Architecture**

Routing

An abundance of routing resources has been designed to offer the highest routability possible even when the design's resources utilization is higher than 95%. The focus of this section is on the global routing resources and their flexible aggregation. Figure 2 illustrates the span of the various global and quadrant networks. In essence, each VersaTile in the die can be reached by six global and three quadrant networks. More importantly, these networks can be aggregated to map local clocks or resets or any high-fanout net.

Moreover, the aggregation of one, two, four, or any number of spines introduces neither an overhead insertion delay nor a skew. This is achieved by the embedded MUX tree that offers a local choice between sourcing from the global network or any local driver. Refer to the IGLOO, Fusion, ProASIC3, and ProASIC3L product handbooks for more details.
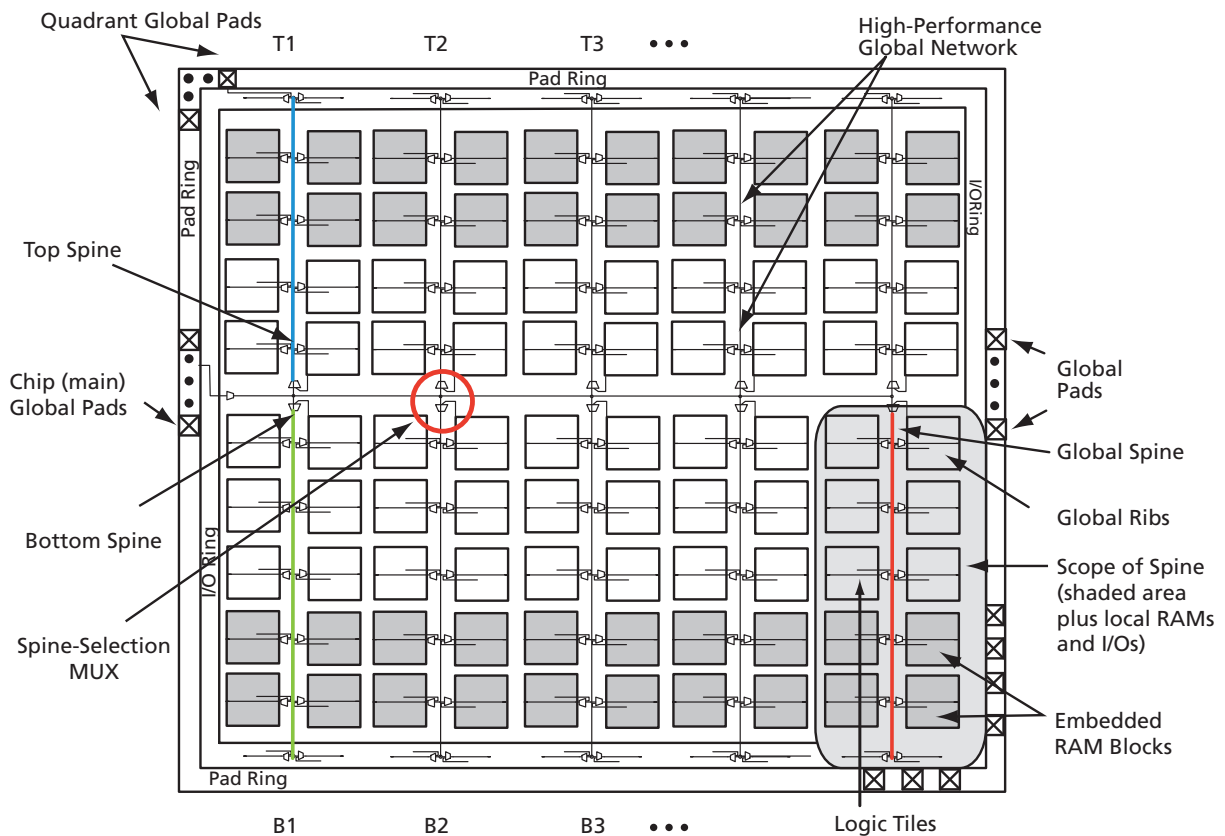

Figure 2 • **Global Networks Distribution**

Clock Conditioning Circuitry (CCC)

For clock synthesis, the CCC offers the highest flexibility with a PLL core that supports very low input frequency and delivers three outputs that can reach 350 MHz (ProASIC3 or ProASIC3L). Figure 3

illustrates the CCC and is self-explanatory. This is mentioned here as it has a link later in the optimization flow.
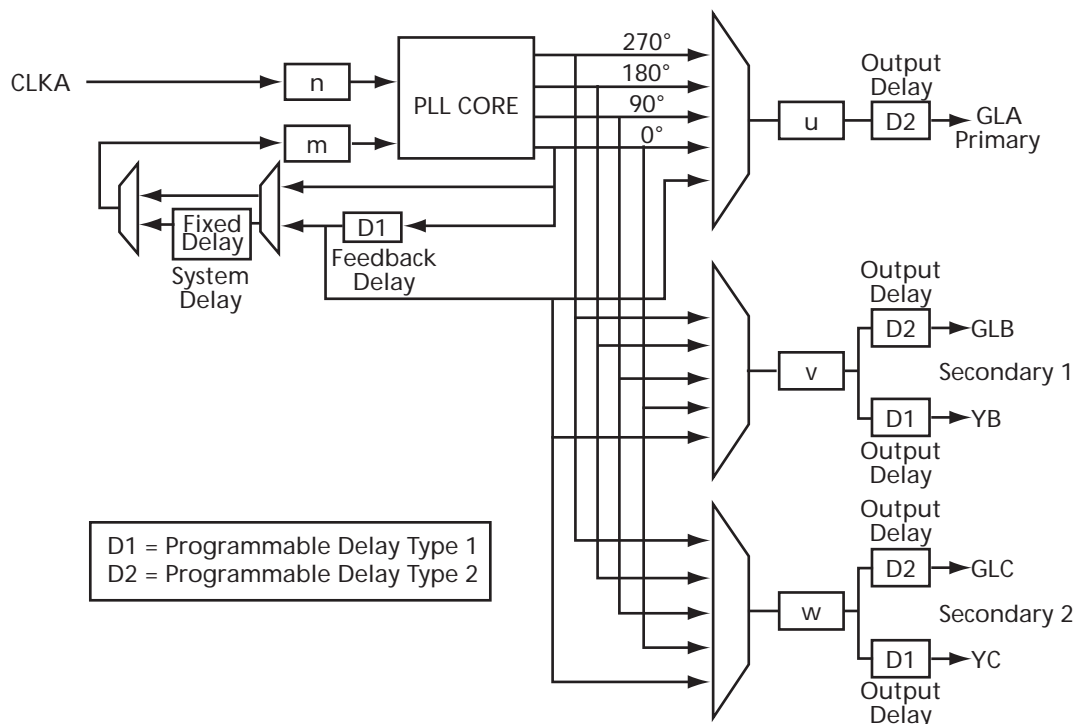

Figure 3 • **Clock Conditioning Circuitry**

Embedded RAM and FIFOs

Embedded (in fixed locations on the die) dual-port RAMs offer variable aspect ratios, clocking, and flexible read and write data widths. Moreover, the user does not need to implement the read/write pointers, and the FIFO flag logic in logic cells as the FIFO controllers are embedded as well. Cascading for wide and deep RAMs and FIFOs is offered through the Microsemi Libero® Integrated Design Environment (IDE) core generator tool. Several synthesis tools do not infer RAMs and FIFOs from the customer code, so users need to instantiate them in the RTL code.

FlashROM

Microsemi IGLOO, Fusion, ProASIC3, and ProASIC3L Flash devices have 1 kbit of on-chip, user-accessible, nonvolatile FlashROM. The FlashROM can be used in diverse system applications such as system calibration settings, device serialization and/or inventory control, or subscription-based business models (for example, set-top boxes).

I/Os

The I/O tiles are flexible to support a large variety of standards ranging from LVTTL to LVDS. They also support DDR (double-data-rate) access with the I/O embedded registers, allowing high data rate and aggressive external timing.

Root Causes of Timing Issues

The main sources that lead to timing challenges when designing with any FPGA are related to one or a combination of the following:

- A large number of paths with a high number of logic levels

    This may be due to a lack of efficiency in the RTL code, a bad setting of the synthesis option, or an issue with the synthesis mapper. They can be inherent to the design itself.

- A large number of high fanout nets

    This situation is mainly due to the reference to a signal or a variable several times in the RTL code. It can also be caused by a high sharing of a combinatorial sub-function.

- High congestion

    There are two types of congestion: netlist congestion and placement congestion. An example of netlist congestion is a crosspoint switch. This type of application has a high degree of interdependence for which the place-and-route engines can do very little. Placement congestion occurs when you have unassociated logic with high degrees of congestion placed in the same area, creating an artificially high demand on routing resources.

    It is important to understand netlist congestion for two reasons. The first is to avoid blaming place-and-route for doing a poor job. The second reason is to give guidance to the placer apportioning internal resources, especially high-drive circuits to handle high-fanout areas.

- A large number of busses

    System control functions involve several blocks communicating wide data and control busses. The routing of these heavily bussed designs becomes challenging when several busses broadcast to several blocks.

- A large number of clock domains

    More and more designs are involving several clocking schemes with asynchronous or related sources to accommodate system-level requirements or to implement parallelism (such as several networking fingers with separate clocks and resets). Moreover, with the integration of clock synthesis circuitries inside the FPGAs, users exploit this feature to manage the clocking of several other devices on the board. The challenge with this type of design is the interdependency and the fuzzy focus of the synthesis and place-and-route engines on the clock domains.

- A large number of embedded memory blocks

    The increasing need for RAM and FIFO size makes the cascading of a large number of embedded RAM blocks a must. In some cases, this leads to a placement challenge, which yields to a complex routing problem. Other situations are challenging because of the non-balance between logic, I/Os, and RAM blocks involved. An easy example is when all the RAM blocks on the top and bottom of the die are used while the core or I/O utilization is very low.

- Poor synthesis, poor placement and/or poor routing

    While some syndromes of this inefficiency are obvious, poor quality of results of these tools is the hardest to figure out and needs a lot of user effort and time to detect.

- User options and constraints settings

    Timing constraints—namely clock domain frequencies, clock dependencies, and input and output delays—are an integral part of the design. Moreover, clock exceptions such as multi-cycle and false paths are of a great help to guide logic and physical synthesis. Several unnecessary iterations were performed because of missing clock exceptions.

    Other constraints, such as floorplanning/placement constraints, can help place-and-route if set appropriately, considering not only congestion but also data flows and timing requirements.

    Notice that some FPGA-specific constraints, such as I/O assignment or clock assignment to global networks or an aggregation of these networks, are routing-based floorplanning and

imply placement constraints of the destination cells. They need care and attentive decision as they can cause serious issues, leading to not only timing headaches, but also expensive board re-spins.

Finally, it is needless to demonstrate the fact that the earlier these sources of timing troubles are identified and dealt with, the sooner the timing closure can be achieved. More importantly, the effort and investment are less for a larger opportunity when the root cause of timing trouble is avoided or identified soon. Figure 4 illustrates this theorem.

As a corollary of this theorem, the best return on investment is in the order of importance associated with the following:

- Quality of the RTL code
- Getting the best of logic synthesis by means of judicious constraints and options setting
- A formal link between synthesis and place-and-route
- Getting the best of place-and-route engines via appropriate constraining and use of the FPGA architecture features
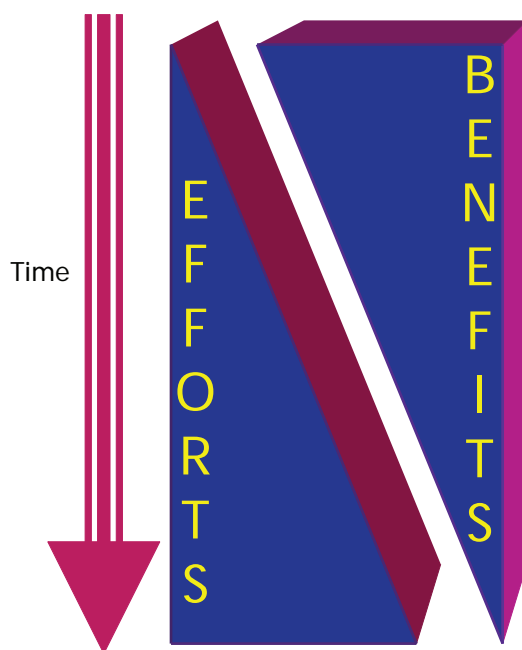

Figure 4 • **Efforts and Opportunities vs. Time**

However, most designs do not meet all the timing specifications and several iterations are needed. The iterative approach that leads to timing convergence in minimal time and effort is based on the following:

- Analysis as a key to the identification of the aforementioned root causes of challenge
- A sort of these causes based on scope of implications and order of importance towards resolution of the timing issues
- Dealing with these sources with an appropriate order of importance, preferably at the highest level of the flow—i.e., the RTL code if possible

The "Main Sources for Design Analysis" section on page 7 summarizes the major sources for pertinent data to identify the bottlenecks.

Main Sources for Design Analysis

It goes without demonstration that not knowing the problem does not help to solve the problem. The best way to tackle performance issues is to find clues that may lead to the root causes. The identification process is not easy and is time- and effort-consuming in most cases, but a systematic approach eases the task. Much data is already available and can be efficiently used to shorten the time and reduce the analysis efforts. The following section will cover most of the available reports and identify the type and nature of data available in each of the reports. Moreover, the missing data is also mentioned so that users check other ways to build the whole picture.

Synthesis Report or Log File

The synthesis quality of results heavily affects the final post-layout timing of the design. The current standard synthesis report files include most of the following data:

- Nets with the associated fanout and buffering
- Register and combinatorial cells replication
- Number of logic levels
- Architecture of the arithmetic and storage or memory elements
- Resource utilization
- Number of clock domains, including explicit and derived domains

Unfortunately, the synthesis reports do not identify the congested blocks, the interconnectivity between blocks, nor the reasons why the synthesis process could not do some of the optimizations.

Microsemi Designer Compile Report

The backend parser performs minor netlist optimizations such as the elimination of some buffers, inverters, cells involved in dangling nets, etc. In some cases, the tool performs combining of I/Os with registers. It also parses the user constraints and translates them into place-and-route constraints. The main data provided in the importer report is summarized as follows:

- Netlist optimization results (number and type of deleted cells)
- Nets promoted to global networks
- Distribution of fanout
- Device resource utilization
- Internal and external nets fanout
- A limited number, generally 20, of high-fanout net candidates for promotion to a global or to a segment of the global network
- Internal clocks

Similar to the synthesis report, the backend compile does not report data related to blocks that are timing-critical or those that are congested. It does not reveal any data on the interconnectivity of blocks or inter-block busses.

ChipPlanner

The ChipPlanner tool helps the user to check several aspects related to the quality of the place-and-route tools. The tool allows users to check the span of the global networks in the die or whether the routed design is congested.

Using the cell selection function of the Viewer, users can check the relative placement of hierarchical blocks.

The ChipPlanner can also help identify if the placement is inefficient. Low-fanout nets that are routed in a very sneaky way are an identification of such a poor placement. However, if the similar sneaky routing is associated with high-fanout nets, this may be due to poor routing.

Unfortunately, the user needs to know a lot more about the design before being able to use the tool to identify block interconnects using ChipPlanner.

Timing Reports

The timing reports are by far the most important source for information to know more about the critical regions of the design, the inherent congestion of blocks, as well as about the quality of results of synthesis, placement, or routing process.

The expanded paths can reveal if the paths are timing-critical because of a large number of logic levels, or because of high delay penalties associated with high-fanout nets. If these high delay penalties are associated with low-fanout nets, this may reveal a poor placement or poor routing. If the expanded paths are made of a long set of two-input gates, the user needs to check if this is due to a poor synthesis or a poor RTL coding style.

Unfortunately, the timing reports do not reveal shared critical nets between paths with worst negative slack, or negative slack distribution. Moreover, the timing reports do not report the hierarchical blocks that meet the timing with a narrow or large margin. To see the importance of these parameters, consider the timing profiles of two designs, given in Figure 5. A first look at the profiles will lead to the conclusion that the design on the left is a lot more complex than the one on the right. However, a deeper look at the common critical nets leading to the negative slack may reveal that the number is very limited for the design profiled on the left. Moreover, a closer look at the profile of the design on the right shows a very high sensitivity to changes in place-and-route; thus, making a decision to tackle the negative slack may induce a new larger set of paths with a negative slack. The moral of this illustration is that analysis must be deep and complete before making decisions.
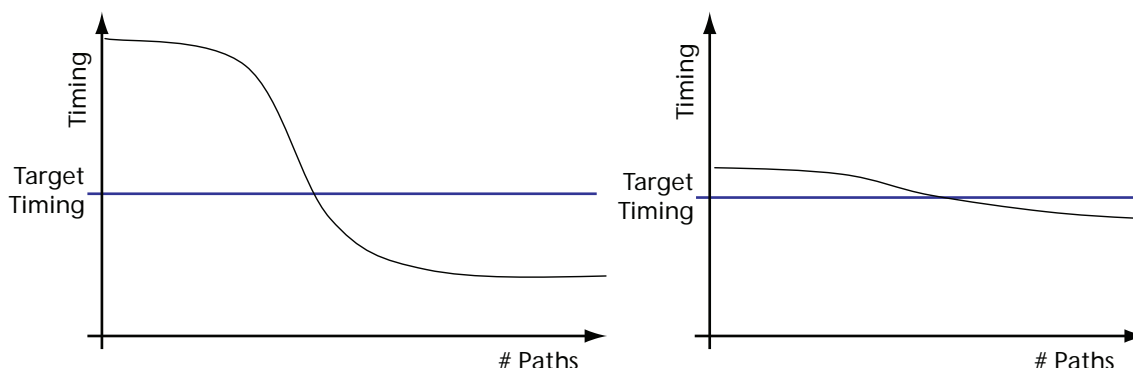

Figure 5 • **Slack/Timing Profiles for Two Different Designs or Implementations**

Ingredients for Timing Optimizations

Before tackling each of the root causes of timing issues, designers are advised to revisit the RTL code and check the various coding optimizations that they can afford. The motivation behind this preliminary step is that a minimal effort at the RTL code level allows a large impact on the final results. The same effect can not be reached at the place-and-route level even if the effort deployed is two or three times larger. Similarly, an appropriate set of synthesis options may lead to a faster convergence if compared to tedious and manual manipulation of a netlist generated with a poor set of synthesis switches.

RTL Coding Tips

Several coding styles have been published. Each of these is related to a technology, architecture, and set of tools. Table 1 gives a list of common-sense rules and should lead to a more efficient implementation. Some of these are automatically optimized by synthesis tools. Other RTL code techniques apply the DDR idea to speed-up the overall performance. This principle doubles the original clock frequency using one of the available CCCs, and performs a part of the processing on one cycle of the doubled clock (i.e., the positive edge of the original clock) and the remaining part of the processing during the second cycle of the doubled clock (or the negative edge of the original clock).

Table 1 • **RTL Coding Rules**

Original Code	New Code
A when A >= 0 else –A	Not A + 1 when A(31) else A
A + 1 when EN = '1' else A	A + EN
X < 0	X(X'HIGH)
X – Y = 0	X=Y
X * 9 X * 15	X SHL 3 + X X SHL 4 – X

Dealing with High-Fanout Nets

Explicit Logic Replication at the RTL Level

In most cases, synthesis performs logic and sequential replication of the drivers blindly and without consideration of the destination cells and where they belong. There are various possibilities for reducing the fanout by explicit replication of the driver. Figure 6 illustrates an example where the explicit replication allows flexibility of the replicated drivers to be placed closer to the internal destination cells and the external ones.
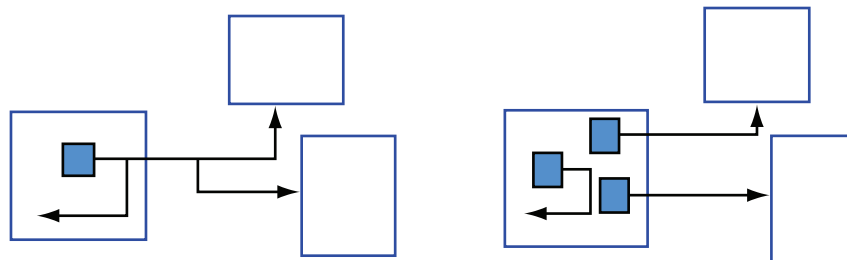

Figure 6 • **Explicit Replication**

In case a block is the source of multiple high fanout nets and its size in terms of logic cells is limited (less than 200 VersaTiles), it is worth investigating the replication of the block itself, as illustrated in Figure 7.
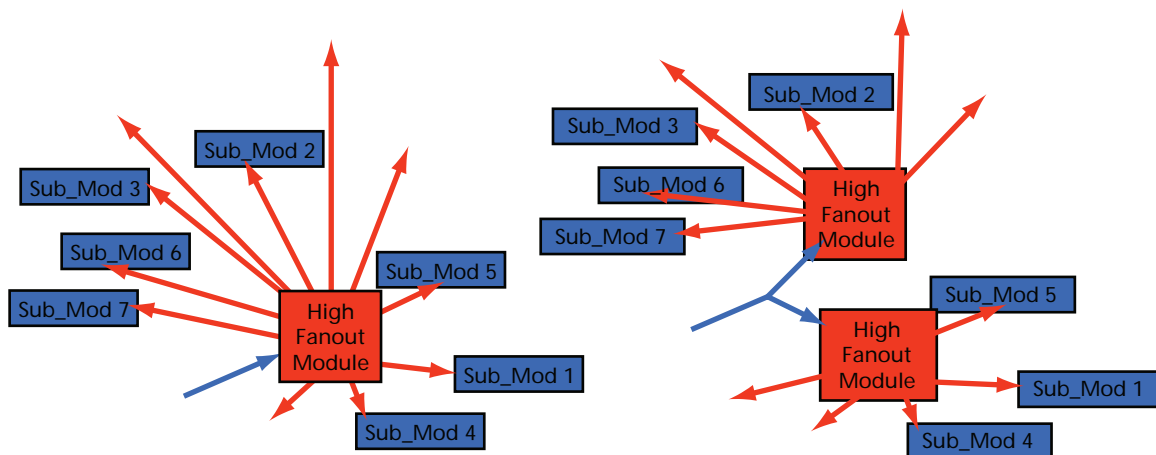

Figure 7 • **Replication of a Block Source of High-Fanout Nets**

In all these cases, designers are cautioned against making these explicit replications blindly. They have to consider this change with care and avoid replicating a driver of a generated clock domain or a synchronized reset. Failing to do so will lead to new clock domains and the headache of making sure they are mapped to low-skew routing resources, or having to analyze removal time for a large set of synchronized resets.

Synthesis Control

Synthesis tools typically offer ways to set fanout limits globally. When available, use local, block-level fanout control only for blocks exhibiting high net fanout.

Backend Control of High-Fanout Nets

The easiest and most efficient way to deal with the delay penalty associated with high-fanout nets is to map these nets to segments of the global networks called spines. Users must also consider the implication of such mapping, as it involves a placement constraint on the driver and the destination cells. They need to fit in the region covered by the segment, called the scope of the spine.

If the global networks are not available or if the placement constraint will introduce high congestion, users can create a so-called net region to limit the skew and the penalty associated with the net.

Finally, if these two techniques do not apply, users can reset the fanout of a net and the tool will work on the shielding of the critical ports and reducing the delay penalties.

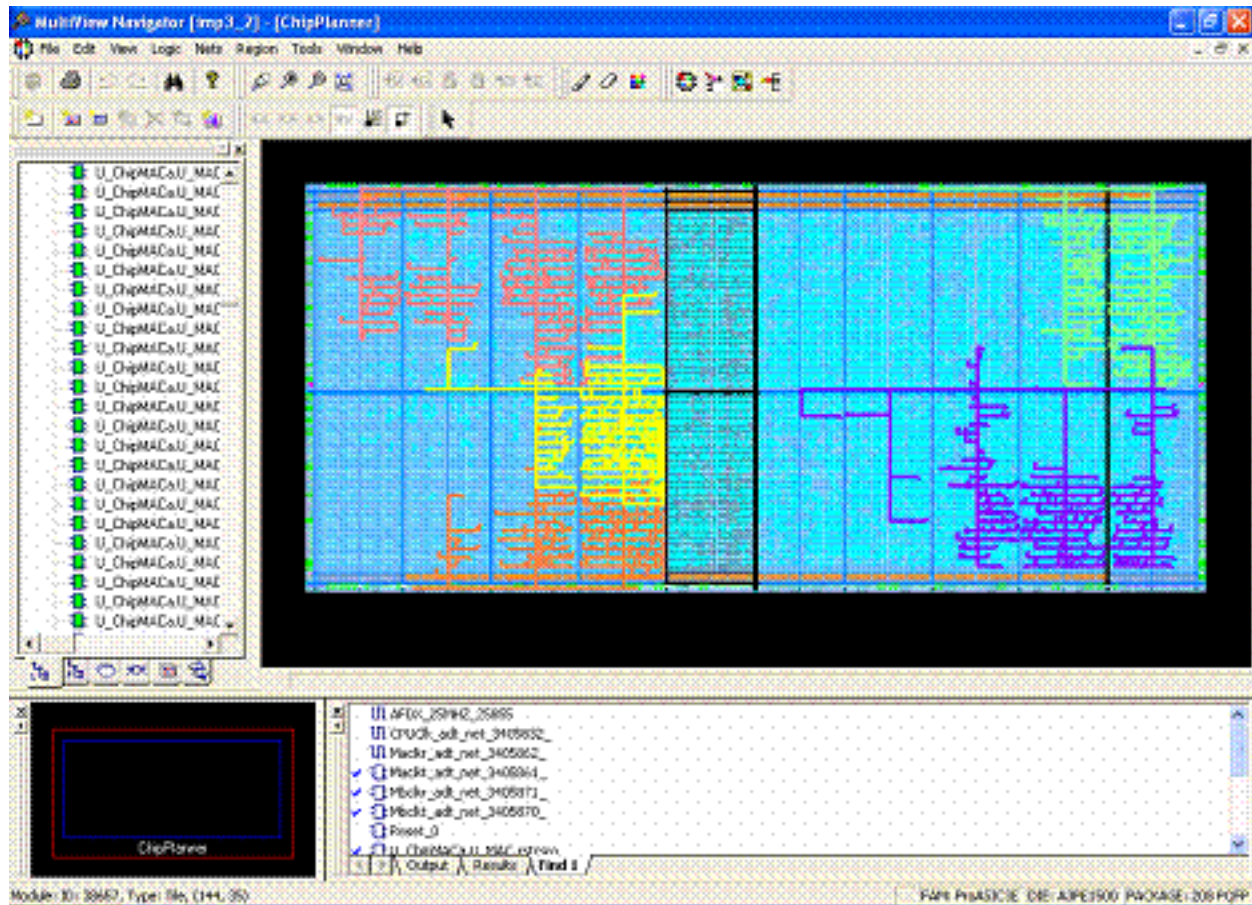Figure 8 illustrates the use of clock network segments to map several high fanout nets.

Figure 8 • **High-Fanout Nets Mapped to Global Network Segments**

Dealing with a High Number of Logic Levels

Revisiting RTL Code

In some cases, the large number of logic levels is inherent to the design. Users can cope with this by adding explicit pipeline stages, adding registers when appropriate. Users may also need to re-architect the timing of their design and anticipate data readiness one cycle ahead of the cycle where they will be processed, allowing two cycles for these paths if possible. Also, when RAM blocks are involved in paths with a high number of logic levels, users can investigate the use of the pipelined configuration of the read port or anticipate the read a cycle ahead, thus allowing two cycles for these paths.

Synthesis Control

The synthesis flow allows for retiming. This register moving around the logic comes with an increase of area, as the number of registers may increase. Users need to watch this increase and monitor the utilization of the device resources. When the number of logic levels involves arithmetic blocks, users need to know that synthesis tools offer a variety of architectures for these blocks. Users need to check the default choice made by the synthesis tool and see whether it is an efficient architecture for the device.

Backend Control

As part of the analysis, users need to verify the placement of the cells as well as the fanout of the nets involved in these paths. In case the fanout of these nets is limited, the tool offers a flexible set of placement constraints allowing the user to confine the placement of these paths/blocks. In case one or more nets are associated with a delay penalty, users can use the shielding technique described above.

Dealing with High Congestion

This section illustrates using an inherently congested block. On a first look at the RTL code depicted in Figure 9, the reader can easily understand the function, but most of us do not realize the underlying complexity of the routing. The same complexity occurs whenever the number of logic cells needed is very limited but the number of routes is extremely high. This is the syndrome of what is called "routing congestion."
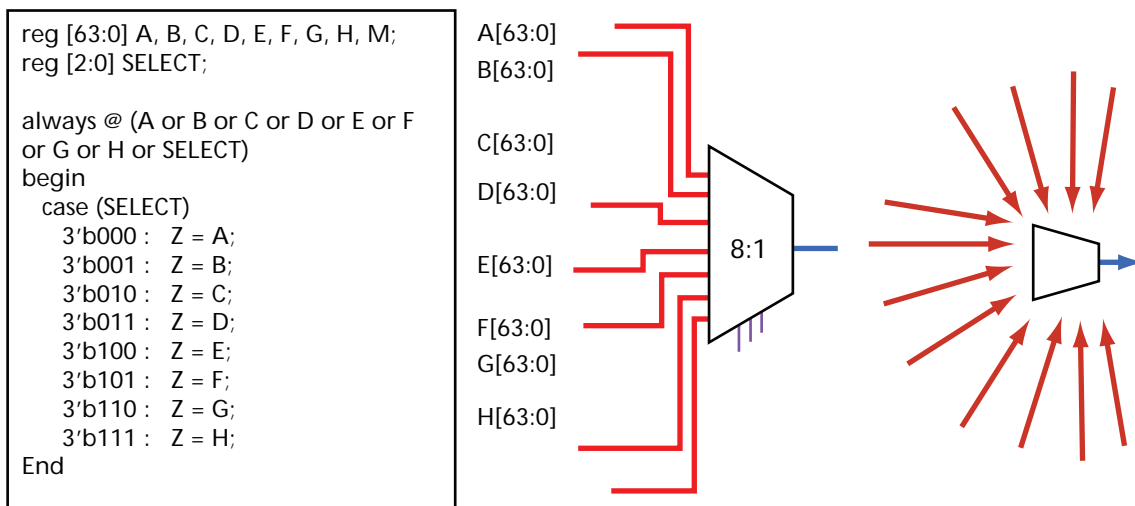
```
reg [63:0] A, B, C, D, E, F, G, H, M;
reg [2:0] SELECT;

always @ (A or B or C or D or E or F
or G or H or SELECT)
begin
  case (SELECT)
    3'b000 :  Z = A;
    3'b001 :  Z = B;
    3'b010 :  Z = C;
    3'b011 :  Z = D;
    3'b100 :  Z = E;
    3'b101 :  Z = F;
    3'b110 :  Z = G;
    3'b111 :  Z = H;
End
```

Figure 9 • **Example of Routing Congestion**

Revisiting RTL Code

For this particular congested block, several ideas can be investigated at the RTL code level. One of the recommended techniques is to decentralize the "routing congestion," as suggested in Figure 10.
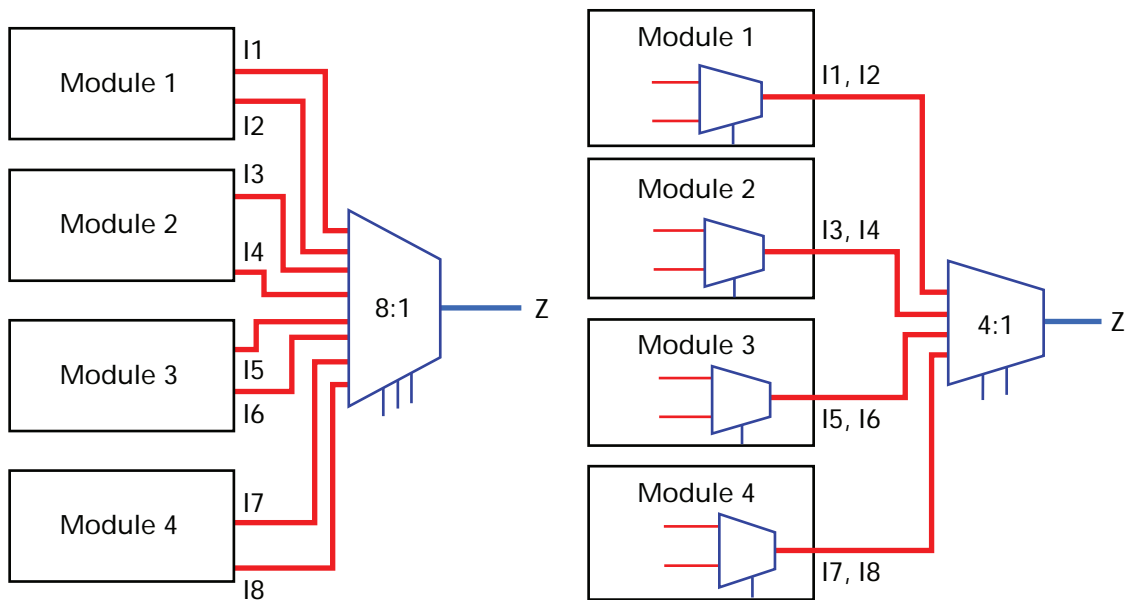

Figure 10 • **MUX Decentralization Technique**

Table 2 shows the area/frequency results obtained when synthesizing the code as is, instantiating a large MUX and implementing the recommended decentralization technique.

Table 2 • **Centralized vs. Decentralized Implementations**

	Pure RTL Synthesis	**Using 4:1 MUX Cores**
32:1, 16-bit wide MUX	148 MHz (130 tiles)	171 MHz (120 tiles)
64:1, 16-bit wide MUX	134 MHz (246 tiles)	160 MHz (217 tiles)

Synthesis Control

Using the same illustration example, users must pay attention to the select lines, as the least significant bits are definitely very high-fanout nets. Moreover, the coding of the select lines, either compact or one-hot, leads to different area and speed results. Table 3 shows these results.

Table 3 • **Compact vs. One-Hot Select Line Encoding**

	Compact Select	**One-Hot Select**
64:1 – 8-bit-wide MUX	134 MHz (132 tiles)	163 MHz (148 tiles)
64:1 – 16-bit-wide MUX	130 MHz (246 tiles)	158 MHz (231 tiles)

As a corollary, users need to think contextually. If the select lines of the large MUX are a state register of a FSM, the encoding of the states of this machine must be one-hot even if this encoding may not look optimal locally.

Another synthesis control is related to resource sharing. The goal of the resource sharing is to reduce design area by sharing large blocks by means of adding MUXes on the inputs of these blocks. Users

need to check the implication and manage the balance between slightly higher utilization, added congestion, and number of logic levels.

Backend Control

One major recommendation is to avoid aggressive placement or timing constraints on these Routing Congestion spots. In other words, users are advised to relax the placement constraints to allow higher porosity. This has to be combined with a relaxation of the timing constraints on the congested block, as well as other non-timing-critical blocks, so that their internal nets do not come into conflict for the use of a critical routing resource.

Another higher-level measure users can adopt is a more data-oriented floorplan for the blocks involved in the congestion.

Dealing with a Large Number of Busses

Revisiting RTL Code

While the margins for maneuvering are tight, designers may revisit the design architecture for bus sharing or attempt to bury some of the busses in larger blocks. Another important aspect that may make the routing of these busses even trickier is the fanout of each of the slices of these busses. If the fanout is not homogeneous, users are dealing with a more complex issue.

Synthesis Control

Unfortunately, at the synthesis level, very little can be done to cope with these situations

Backend Control

The most efficient physical implementation of these busses is the shortest routing possible. This involves placement of the drivers and the destination of each slice of the busses. If the fanout of these slices is not homogeneous, the highest-fanout bus lines can be mapped using low-skew segments of the global networks.

In any case, users need to adopt a data-driven placement of the communicating blocks and relax the placement constraints for higher porosity and ease of routability of blocks and busses.

Dealing with a Large Number of Clock Domains

Revisit the RTL Code

While most of the clocking schemes are defined at an early stage of the system architecture cycle, designers use various techniques to handle clocking of various blocks of the design, to cope with either timing bottlenecks or area/resource constraints. Other designers are power-conscious and may implement various clocking schemes using clock MUXing or gated clocks. For all these techniques, designers must accurately assess the gain before heading up to creating/generating new clock domains. More importantly, designers need to think ahead of time and make sure that the resulting design and its associated clocking scheme keeps it "analysis-friendly," as the static timing analysis phase can become tedious and time consuming.

Synthesis Control

The focus of the designer should be on the setting of all the timing constraints. These include the tightest clock frequencies, the inter-clocks off-sets, the input and output delays. False paths and multi-cycle paths are very critical and need not to be neglected.

Backend Control

The general guidelines are to separate the clock domains and adopt clock-domain-based floorplanning. This will also allow an economy of the global networks and a more effective use of the low-skew segments of the global networks. When doing so, users need to integrate data dependency between domains and take into account paths optimizations.

A case worthy of note is when some of the clock domains drive a large number of RAM/FIFO blocks (deep or shallow RAMs that are mapped cascading several embedded RAM blocks). In such a case, users can consider placing the RAM blocks on the top and bottom of the die, provided their performance degradation does not affect critical paths.

Dealing with a Large Number of I/Os

Revisit the RTL Code

Even if the margins for maneuvering are tight at this level, designers may adopt time multiplexing and lower the number of I/Os if this is possible.

Synthesis Control

Users need to turn off the inference of registered I/Os, as this leads to an inherent placement constraint of the registers associated with these I/Os.

Backend Control

Users need to investigate carefully the ratio of logic and /IO utilization. If the logic utilization is low, the recommendation is to run place-and-route with I/O register combining turned off. If the internal register-to-register performance is satisfactory, then users need to investigate the slack margins and allow register combining for the most critical external setup of clock-to-out timing. If doing so does not resolve the problem, the I/O placement can be modified to accommodate both the internal and external timings.

Dealing with Poor Synthesis, Poor Placement, and Poor Routing

Poor synthesis results can be caused by either inherent limitation in the synthesis engine or by poor setting of options and wrong specification of timing constraints. It can also be related to lack of knowledge of how the mapping algorithms work and what to expect once a particular constraint is added. This mismatch between true capacity of the tool and the user expectation leads to frustration and several unnecessary iterations. In the category of "know your tool," Table 4 provides a sample of results for a very limited number of benchmarks that were processed with an industry synthesis tool and Designer backend toolset.

Table 4 • **Sample of Results on a Small Set of Benchmarks**

Design Name	Synthesis Options Place-and-Route Option	Area-Driven		Timing-Driven Replication On		Timing-Driven Replication Off	
		Default	All Buffers Removed	Default	All Buffers Removed	Default	All Buffers Removed
Design RDES	MUX- and XOR-based Area in VersaTiles SystemClk (MHz	12822 87	12822 87	18181 83.9	16362 87.5	16969 84.2	14359 82.7
Syncop	Bus interfaces / large reg files Area in VersaTiles HighClock (MHz)	6150 71	6124 70	7284 63.5	6886 66	7192 63.5	6855 68.2
HRK	Area in VersaTiles TopClk (MHz)	2540 70.3	2538 71.67	3041 72.7	2909 77.8	3035 75	2905 72.5
CORDIC	Datapath Area in VersaTiles MainClk (MHz)	3249 57.3	3238 53.4	11398 80.5	9792 78.3	11180 76.7	9691 78.6
Imen	Scrambling and Descrambling Area in VersaTiles Speed (MHz) VersaC HardClk RxClk TxClk	4274 88 87 60 60	4250 90 95 57 64	6815 102 103 66 67	6023 108 106 67 66	6586 108 106 67 65	5884 111 104 70 68

Table 4 • **Sample of Results on a Small Set of Benchmarks (continued)**

Design Name	Synthesis Options	Area-Driven		Timing-Driven Replication On		Timing-Driven Replication Off	
	Place-and-Route Option	Default	All Buffers Removed	Default	All Buffers Removed	Default	All Buffers Removed
Newton	System Interfaces and Control Tiles Speed (MHz) SysClk PicClk	22013 44.7 51.2	22008 46.5 53.64	72274 42 37	67231 42.4 35.2	70780 39.8 40	66708 46 38
Boldy	Dual MAC/Memory Intensive Area (VersaTiles) CPUclk (MHz) AFDX_clk (MHz)	16215 121 39	16213 111 35	20729 116 41	19379 117 36	20326 109 44	19326 132 37

A first look at this small sample of results highlights the efficiency of the area-oriented flow, both in terms of compact area and respectable speed. Pushing this tool hard with non-realistic timing constraints leads, in most cases, to a huge overhead of area, particularly when logic replication is ON. For less area penalty and slightly higher frequencies, users can push for timing-driven synthesis with the replication switched OFF.

Conclusions—Forward Looking

As its title suggests, this application note is a tour of various aspects related to timing convergence when targeting Microsemi Flash-based FPGAs using synthesis tools and Designer, the Microsemi backend toolset. The contribution of this document is to help designers focus on the analysis of the timing bottlenecks, identify the root causes, and cope with them. Several suggestions have been provided, which enable designers to tackle each of these timing challenges at the RTL code level, the synthesis setting, and the backend constraints.

List of Changes

The following table shows important changes made in this document for each revision.

Date	Changes	Page
Revision 1 (June 2015)	Non-technical Updates.	NA
Revision 0 (March 2008)	Initial Release.	NA

Note: **The revision number is located in the part number after the hyphen. The part number is displayed at the bottom of the last page of the document. The digits following the slash indicate the month and year of publication.*

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

51900173-1/06.15