Optimal Usage of Global Network Spines in ProASIC^PLUS Devices

Table of Contents

Introduction . . . 1
What is a Spine? . . . 1
Design Considerations . . . 3
Assigning Signals to Spines . . . 3
Spine Assignment Strategies . . . 4
High-Fanout Nets Driving Multiple Spines . . . 7
Spines and Middle Die Rows . . . 7
Placement Conflicts . . . 7
Routing or Placement Congestion . . . 8
Using Spines of Occupied Global Networks . . . 9
Spine Assignment Verification in Designs . . . 11
Conclusion . . . 11
Appendix: Scope of the Logic Tiles in ProASICPLUS Devices . . . 12
Appendix: Buffer Insertion . . . 15
List of Changes . . . 16

Introduction

The ProASIC^PLUS® architecture contains four segmented global networks that can access all the logic, memory, and I/O tiles on the die. These global networks offer low-skew and fast routing resources for high-fanout nets including clock signals. In addition, these global networks, which are segmented, offer users the flexibility to create low-skew local networks using spines for up to 88 (in an APA1000 die) internal/external clocks or other high-fanout nets in ProASIC^PLUS devices. Optimal usage of these low-skew networks results in significant improvement of design performance on ProASIC^PLUS devices. The usage of these local networks for high-fanout signals eliminates the need for buffer trees and decreases the utilization of the logic tiles and routing resources.

The ProASIC^PLUS devices contain two PLL blocks. The outputs of the PLLs drive the global networks through the global MUX architecture. The interaction of the PLL and the global networks is discussed in the *AC306: Using ProASICPLUS Clock Conditioning Circuits Application Note*. Any portion of the global networks that is not used by a PLL output or other global signal can be accessed by other high-fanout signals using the global network spines.

This application note discusses the segmentation of the global networks in ProASIC^PLUS devices and their usage in designs containing multiple clock domains or a large number of high-fanout nets.

What is a Spine?

Spines are the vertical branches of the global network tree, as shown in Figure 1 on page 2. Each vertical column of a global network is divided into two spines, one in the top and one in the bottom half of the die. Unlike ProASIC 500 K devices, ProASIC^PLUS field programmable gate arrays (FPGAs) use top and bottom half spines of the same height. The physical location of each spine is identified by the letter top (T) or bottom (B) and an accompanying number (example, Tn or Bn). The number n indicates the horizontal location of the spine, with 1 referring to the first spine on the left side of the die. Because there

are four global networks on each die, there are up to four spines available for each combination of T or B and n (example, four T1 spines). Refer to Table 1 for the total number of spines on each die.

Each spine covers a certain area of the die (the scope of the spine), as shown in Figure 1. Each spine is accessed by the MUX architecture, which defines how a particular spine is driven–by either the signal on the global network or another net, defined by the user (Figure 1). The detail of a spine-selection MUX is shown in Figure 2 on page 3. The spine drivers for each spine are located in the middle of the die.

Note the following from the architecture in Figure 2 on page 3:

- If a top or bottom spine of a global resource (example, T1) is driven by a signal coming from the global network, the complementary spine of the same global resource (example, B1) cannot be driven by another internal or external signal.
- If neither of the complementary top and bottom spines of a global resource are driven by a global network, they can be driven independently by internal or external signals.

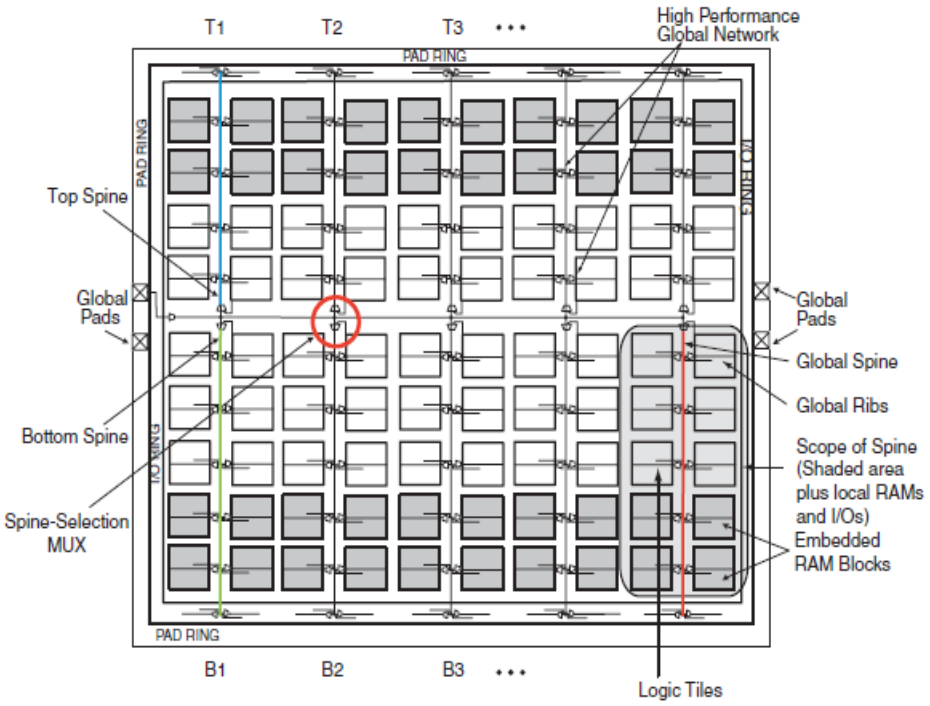

***Figure 1* • Spines in a Global Clock Tree Network**

***Table 1* • Clock Spines of ProASIC^PLUS Devices**

	APA075	APA150	APA300	APA450	APA600	APA750	APA1000
Global clock networks	4	4	4	4	4	4	4
Spines per global	6	8	8	12	14	16	22
Total spines	24	32	32	48	56	64	88
Spine height	16	24	32	32	48	64	80
Logic tiles In each spine	512	768	1024	1024	1536	2048	2560
Total logic tiles	3072	6144	8912	12288	21504	32768	56320

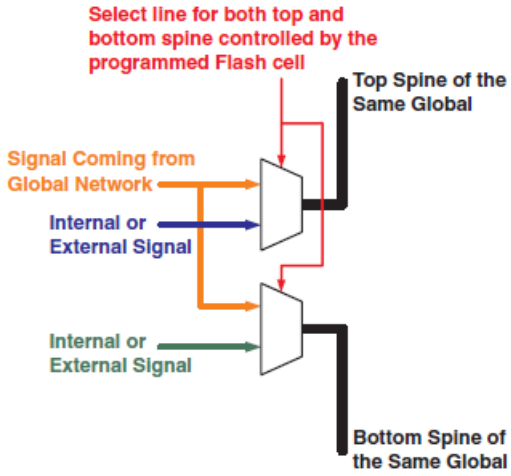

***Figure 2* • Spine Selection MUX of a Global Tree**

Design Considerations

The optimal strategies for successful usage and assignment of spines may vary from one design to another. Different design parameters such as utilization, routing congestion, design sub-block interconnections, number of global signals, fanout of the assigned signal, and other user constraints dictate the best approach. This section presents strategies for assigning signals to spines as well as guidelines for floorplanning. In addition, using spines of global networks is discussed, which are already partially used.

Assigning Signals to Spines

The spine assignment is carried out by the `use_global` command in the GCF file. The `use_global` command can be exploited for both single and multiple spine assignments. The following are some examples of legal usage of this command:

- `use_global T1 net1`

This command assigns the signal net1 to the T1 spine of any of the available global resources.

- `use_global T1,B3 net1`

This command defines the boundaries of the spine assignment. The example assigns the net1 signal to T1, T2, T3, B1, B2, and B3 of any of the available global resources. The spines do not necessarily from the same global resource (designer software automatically selects the appropriate and available global networks). For example, T1 and T2 can be the spines of one global network while for T3, B1 and B2 are spines of another global network. Note that use_global T1, B3 net1 accomplishes the same result.

- `use_global B2,B4 net1`

Assigns signal net1 to the B2, B3, and B4 spines. These spines can be from multiple global resources or from the same one. It is up to the place-and-route tool to select the spines from different available global networks.

Spine Assignment Strategies

Placement Tips

A single net cannot be assigned to two spines that are not adjacent (horizontal or vertical). In other words, only one `use_global` statement is allowed per net. If more than one `use_global` command operates on one net, only the latest statement is taken into account. For example, consider the case in which the GCF file contains the following commands (which are read from top to bottom):

```
use_global T1 net1;
use_global T3 net1;
```

After place-and-route, signal net1 is only assigned to the T3 spine (most recent GCF file assignment takes precedence), and all the logic driven by this net is placed in the scope of the T3 spine. If a net is needed to drive two nonadjacent spines, a buffer must be inserted to create a different, buffered net from the original one. In the above example the assignment can be changed to the following:

```
use_global T1 net1;
use_global T3 net1_buffered;
```

Refer to "Appendix: Buffer Insertion" on page 15 for guidelines on buffer insertion in ProASIC^{PLUS} devices.

To assign two different signals to the same spine area (example, B3), two global networks with free spines in that particular area (example, two free B3 spines) are required. Then the following commands in the GCF file accomplish the assignment:

```
use_global B3 net_1;
use_global B3 net_2;
```

In some cases, designers need to use three or more global networks for other signals while still wanting to assign more than one signal to the same spine area. The `use_global` command only operates successfully, if there are enough free spines available in the specified spine location. Guidelines for using the `set_net_region` command to provide those free spines are provided in the "Using Spines of Occupied Global Networks" on page 9.

Spines Driven by Input Buffers

If the signal assigned to a spine is directly driven by an input buffer, Microsemi Designer software automatically add a buffer, called <OriginalNetName>_BUF_AUTO, and route the output of this buffer, called <OriginalNetName>_BUF_AUTOnet, to the designated spine. The `use_global` command operates on nets not ports. Therefore, in order to assign an input signal to the spine, the user needs to apply the `use_global` command to the output net of the input buffer. For example, consider a design input port called myclock. In the netlist, myclock is the input to IB33:

```
MyInputBuffer: IB33
port map( PAD => myclock,
Y=> myclock_c);
```

The GCF file must contain the following command to assign the myclock port signal to a spine:

```
use_global B2,B4 myclock_c;
```

Spines Driving SRAM Blocks

Each spine covers only a limited number of the embedded SRAM blocks on the ProASIC^{PLUS} die, as shown in Figure 1 on page 2. In other words, if a spine drives an embedded SRAM block, the memory block must be placed (using the `set_location` command in the GCF file) right at the top (T-type spine) or bottom (B-type spine) of the scope of the spine. The following example demonstrates a case in which the spine signal drives two embedded SRAM blocks.

Figure 3 is a simplified block diagram of the example design. The design targeted for an APA300 device, includes a memory block (called U_mem), which consists of two basic embedded SRAM blocks (RAM256x9SST), called M0 and M1.

The read clock of the memory, along with the rest of the design (identified simply as Logic in Figure 3 on page 5) is driven from the clka input, which is a global signal (GL33 input buffer). The write clock of the U_mem block is driven by another signal, named clkb. The clkb signal must drive the T2 spine. The following GCF commands result in a successful spine assignment for this design:

```
use_global T2 clkb_c;
set_location (33,71) U_mem/M0;
set_location (49,71) U_mem/M1;
```

The first command assigns clkb_c, the output net of the IB33 macro, to the designated spine. The next two instructions place the RAM building blocks of U_mem, M0 and M1, onto the scope of the T2 spine. The coordinates of the memory rows in ProASIC^{PLUS} devices can be found in Table 2.

Because the clkb_c signal is coming directly from the input pad, Microsemi Designer software automatically add a buffer to this net before entering the spine. Figure 4 on page 6 shows the layout results with the clkb_BUF_AUTOnet highlighted. the clkb signal drives the T2 spine, and the memory block is located at the top of the T2 spine coverage area (highlighted with large blue blocks), as shown in Figure 4 on page 6.

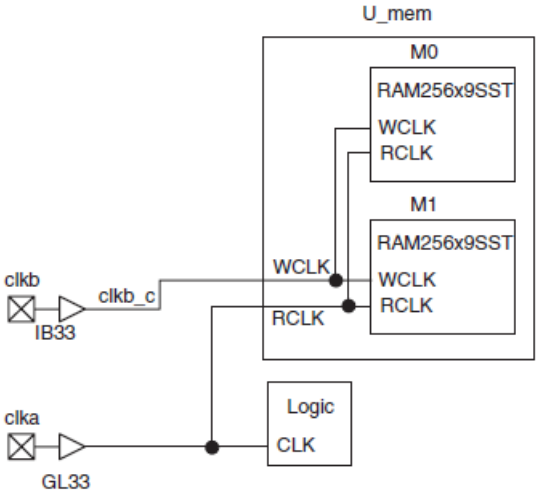

***Figure 3* • Simplified Block Diagram of the Example Design**

Table 2 • **Memory Location for ProASIC^{PLUS} Family**

Part	Possible RAM Locations	Formula
APA075	(1,33),(17,33), …, (81,33)(1,35),(17,35), …, (81,35)	x = 16*n+1;n = {0,1,2,3,4,5}, y = {33,35}
APA150	(1,49), (17,49), ..., (113,49) (1, 51), (17, 51), ..., (113, 51)	x = 16*n+1; n = {0,1,2,3,4,5,6,7}, y = {49, 51}
APA300	1,1), (17,1), ..., (113,1)(1, 3), (17, 3), ..., (113, 3)(1,69), (17,69), ..., (113,69)(1, 71), (17, 71), ..., (113, 71)	x = 16*n+1; n = {0,1,2,3,4,5,6,7}, y = {1,3,69,71}
APA450	(1,1), (17,1), ..., (177,1)(1, 3), (17, 3), ..., (177, 3)(1,69), (17,69), ..., (177,69)(1, 71), (17, 71), ..., (177, 71)	x = 16*n+1; n= {0,1,2,3,4,5,6,7,8,9,10,11}, y = {1,3,69,71}
APA600	(1,1), (17,1), ..., (177,1)(1, 3), (17, 3), ..., (177, 3)(1,101), (17,101), ..., (209,101)(1, 103), (17, 103), ..., (209, 103)	x = 16*n+1; n = {0,1,2,3,4,5,6,7,8,9,10,11,12,13}, y = {1,3,101,103}
APA750	(1,1), (17,1), ..., (241,1)(1,3), (17, 3), ..., (241, 3)(1,133), (17,133), ..., (241,133)(1, 136), (17, 136), ..., (241, 136)	x = 16*n+1; n = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15}, y = {1,3,133,136}
APA1000	(1,1), (17,1), ..., (337,1)(1,3), (17, 3), ..., (337, 3)(1,165), (17,165), ..., (337,165)(1, 167), (17, 167), ..., (337, 167)	x = 16*n+1; n = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19, 20,21}, y = {1,3,165,167}

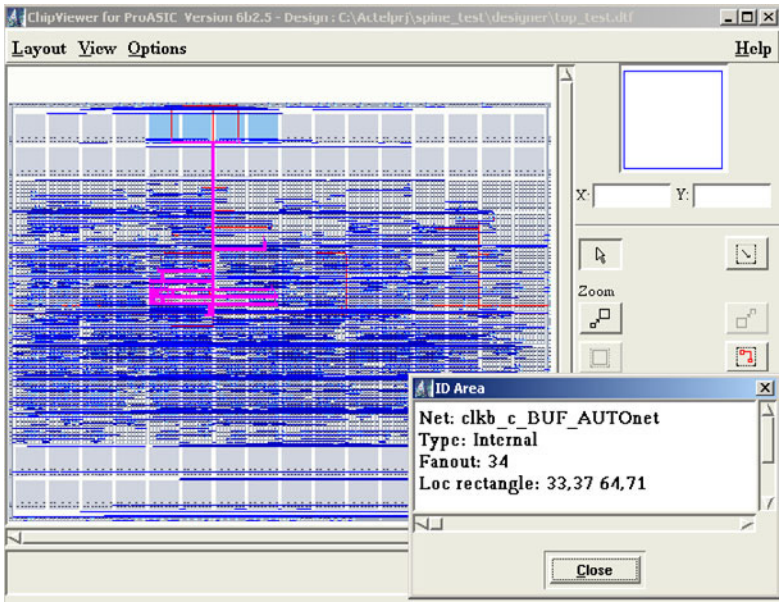

Figure 4 • **Layout of the Example Design**

By the same token, if the net that is assigned to a spine drives an I/O tile, then the I/O tile must be manually placed within the access range of the spine.

High-Fanout Nets Driving Multiple Spines

If a high-fanout clock signal is required to drive more than one spine, Microsemi recommends using vertically-adjacent spines to reduce the skew between the logic placed on the scope of multiple spines. Consider the following spine assignment commands:

```
use_global T1,T2 CLK1;
use_global T3,B3 CLK2;
```

The CLK1 signal needs to be routed to the MUX drivers of both T1 and T2. The routing delay between the MUX drivers of T1 and T2 introduces additional skew to the CLK1 network. Conversely, if the CLK2 signal is routed to the MUX driver of T3, almost no routing resources are required to connect CLK2 to the MUX driver of B3, because the drivers are adjacent. In this case, the skew to the clock network is minimal compared to the first command in this GCF example.

Spines and Middle Die Rows

If two vertically-adjacent spines (example, T1 and B1) are not assigned to the same net (example, net1 and net2), due to the architecture of the routing resources, there is a chance that those spines cannot access the logic tiles of the two middle rows of the die (example, rows 16 and 17 of APA075 die). In other words, for every position in the two middle rows only two of the four spines can drive this position. For example, only two of four B1 spines can reach the lower of two middle rows, and similarly only two of four T1 spines can reach the upper row of the two middle rows. The software does not allow specifying the particular spine layer when assigning the spines.

Figure 5 on page 8 shows this by highlighting upper, lower, and middle rows with different colors.

However, if Tn and Bn are assigned to the same signal, there is no consideration driving the logic tiles of the two middle rows. Referring to Figure 5 on page 8, when Tn and Bn are assigned to different signals, Designer software tries to place the driven logic tiles on the orange or green area. For any reason such as timing requirements or user constraints, the logic tiles driven by Tn or Bn spines are placed on the two middle rows, then there is only a 50 % chance of successful spine assignment. Note the place-and-route tool arbitrarily decides, if the top or bottom spine reaches the middle rows during the routing stage and not the placement stage. Spine assignments that fail due to this limitation isreported during routing the design.

As a result, Microsemi recommends users to avoid placing the logic driven by the spines on the two middle rows unless the two vertically adjacent spines are assigned to the same logic. Again there is no consideration if Tn and Bn are from different global resources or assigned to the same signal. Refer to the tables in "Appendix: Scope of the Logic Tiles in ProASICPLUS Devices" on page 12 for the coordinates of the top, bottom, and middle rows in ProASIC^{PLUS} devices.

Placement Conflicts

A spine assignment constraint with the `use_global` command is inherently a placement constraint. It forces all the logic driven by a spine signal to place in the scope of that spine. Therefore, any other design limitation or user constraint that conflicts with this placement causes the router to be unsuccessful in driving the spine with the desired signal. For example, consider a design consisting of numerous blocks including U1 and U2. Figure 6 on page 8 shows a simplified block diagram of U1 and U2 and their interactions.

If this design is targeted for an APA300 device, the following commands in the GCF file results in a compilation error or unsuccessful spine assignment:

```
use_global T1 CLK_c;
set_location (66,10 95,30) U2/*;
```

The reason for the failure is the placement conflict between the two commands. The first command forces the place-and-route tool to assign m1, m2, and m3 to the region (1,31 32,68), which is T1s scope, while the second command compels the placer to assign m3 to the region (66,10 95,30). A problem occurs because these two regions are mutually exclusive, If the two defined regions overlap each other, the layout tool attempts to place U2 in the area shared by the two commands.

Another case of placement conflict occurs when two signals, assigned to different spines, drive the same logic cell. If the scopes of the spines are mutually exclusive, then this sort of command results in errors or unsuccessful spine assignment.

If necessary, many placement conflicts can be resolved by manual buffer insertion. For example, in the design in Figure 6, the conflict can be resolved by inserting a buffer before the clock port of the m3 register. However, this makes the design more vulnerable to setup or hold-time violations because the clock signal of m3 is no longer on a global network and is therefore subject to greater clock skew.

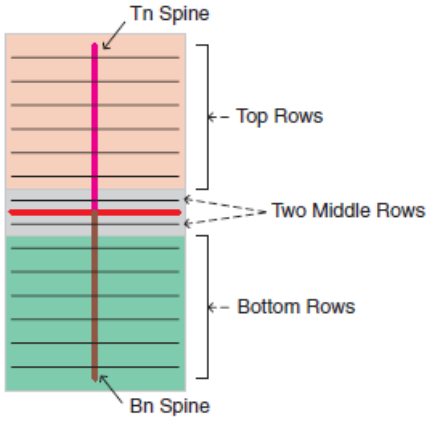

***Figure 5* • Diagram of Spines and Middle Die Rows**

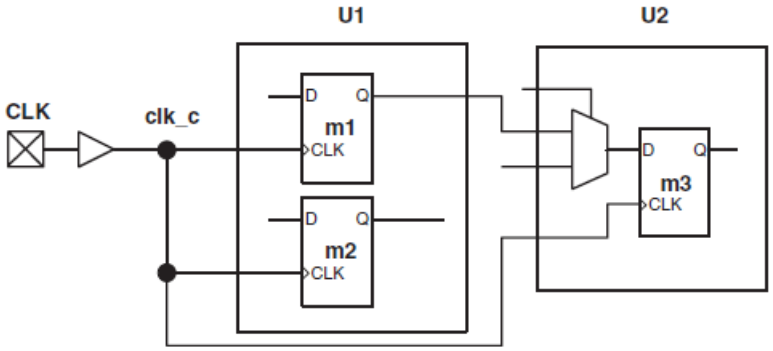

***Figure 6* • Block Diagram of Design Example on Placement Conflicts**

Routing or Placement Congestion

In some large designs, there are many location and timing constraints that can cause routing or placement congestion. In these cases, the place-and-route tool may find it difficult to route the spines driving signal to the desired spine selection MUX (MUX architecture in Figure 2 on page 3). This result in an unsuccessful attempt to assign the signal to the designated spine.

Access to a spine can be improved by placing spine driving signal logic near the spines entry point. The entry points for the spines are located vertically in the middle of the die and horizontally in the middle of the scope of the spine. For example, in an APA750 device, the best locations for access to the T1 spine entry points are (16,69) and (17,69), while the best placements for access to the B2 spine entry are (48,68) and (49,68). Designers use the "Appendix: Scope of the Logic Tiles in ProASICPLUS Devices" on page 12 to locate the entry point of each spine in a particular die. For instance, the closest locations to the T1 spine entry point in an APA075 die are (16,17) and (17,17). If users include these individual placements in their GCF file, Microsemi recommends putting them at the end of the file to ensure overriding of prior command lines in the GCF file.

For cases in which designers confront spine assignment failures due to routing or placement congestion, manual placement of the spine driver may increase the chances of a successful spine assignment; however, success of this approach is not guaranteed.

Using Spines of Occupied Global Networks

In order to assign a signal to a spine, the spine itself and the entry MUX (Figure 2 on page 3) must be free. However, this does NOT necessarily require the corresponding global network to be unused. For example, it is possible to have all four global networks used and still assign non-global signals to spines. This requires using some of the techniques described in this section.

Closer look at Figure 2 on page 3 indicates that if a spine is used as part of the global network, the complimentary spine of the same global network (that is T1/B1, T2/B2, and so on) cannot be driven by another signal because the MUX drivers are sharing the same select line. However, if a spine is driven by a nonglobal net then the vertically complimentary spine of the same global network is also available for the same or another nonglobal net and cannot be used for the net on that global network.

If a signal is driving the global network, the Flash switches are programmed to set the MUX select lines (Figure 2 on page 3) to drive the spines of that network with the global net. However, if the global net is restricted from reaching into the scope of a spine, the MUX drivers of that spine and the vertically-adjacent one are available to the user for other high-fanout or critical signals. The `set_net_region` command limits the access region of a net to a specified area. As a result, `set_net_region` also contains a set_location constraint on the logic, driven by or driving the specified net. Refer to the *Libero IDE v9.1 User Guide* and *Libero IDE v9.1 Online Help* for more information on the `set_net_region` command syntax. The following example represents simple usage of the `set_net_region` constraint.

Consider a design consisting of four building blocks and targeted for an APA300 die (Figure 7 on page 10). The example design consists of a global reset (reset_n) and four clock domains (clk_a, clk_b, clk_c,and clk_d) driving the different blocks of the design. Reset_n, clk_a, clk_b, and clk_c have the highest fanout in this design and must be placed on the global networks available on the ProASIC^{PLUS} device. However, in order to meet the timing requirement for clk_d, which has a lower fanout, this signal must be placed on a low-skew network such as the T4 spine.

The following GCF constraints can be used along with the netlist to assign three clocks and the reset signal to the global networks, while the clk_d clock is assigned to the T4 spine:

```
set_global clk_a;
set_global clk_b;
set_global clk_c;
set_global reset_n;
set_net_region (1,5 96,68) clk_a;
use_global T4 clk_d; (or use_global T4,B4 clk_d;)
```

Figure 8 shows the chip layout after place-and-route of the netlist with the above GCF file. In Figure 8, the clk_a and clk_d signals are highlighted in pink and brown, respectively. The clk_a global network is limited to the scope of the T1 through B3 spines and T4 and B4 are left free. The clk_d signal drives the T4 spine of the same global network and the Block D logic module is placed in the scope of T4.

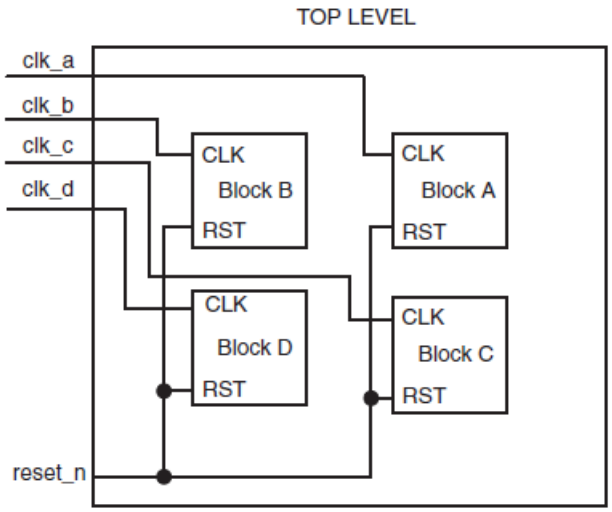

***Figure 7* • Block Diagram of the set_net_region Usage Example Design**

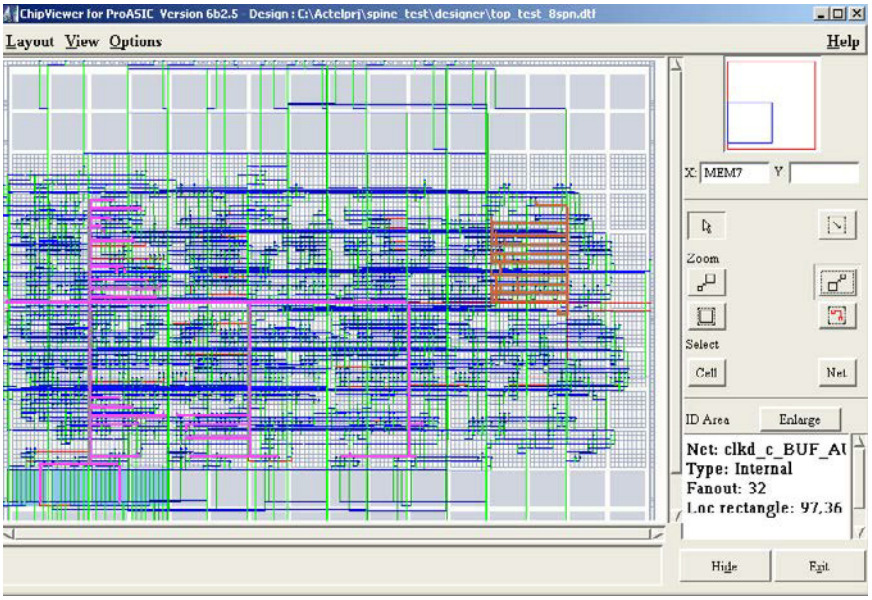

***Figure 8* • Chip Layout**

Consider the following hints in applying the `set_net_region` constraint to different signals, especially global nets:

- Applying `set_net_region` to a signal limits the access area of the signal to the specified region. This may cause layout problems, if other logic driven by the same signal is forced to be placed outside the access area. For instance, in the previous example (Figure 7), applying `set_net_region` to the reset_n signal instead of clk_a and assigning clk_d to T4 causes a conflict in layout because reset_n drives Block D and this block is placed in the scope of T4. Now T4 is no

longer accessible to reset_n. In these cases, buffer insertion (to take the net outside the designated area) could solve the problem; however, it will introduce skew to the buffered signal.

- As seen in Figure 2 on page 3, it is not possible to limit the global signal to just the upper half and use the Bn spines for other signals or vice versa.
- I/O placement is critical, if the limited net is driving an output or driven by a regular user I/O. In the previous example (Figure 7 on page 10), if the clk_a signal is driving an output of the design, the output pin cannot be assigned to the pins on the right side of the die (example, pin 135 of the PQ208 package) because it cannot extend to the right side due to the `set_net_region` constraint. If no pin location constraint is applied, Microsemi Designer software assigns the pins to a proper arbitrary location.
- If users instantiate RAM blocks in their designs, care must be taken in using `set_net_region` and spine assignment if any of these limited or assigned nets are driving the RAM blocks. Manual placement of the embedded RAM blocks, as explained in the "Assigning Signals to Spines" section on page 3, may be necessary for successful spine assignment or execution of the `set_net_region` command.

Spine Assignment Verification in Designs

Major conflicts or errors in the GCF file are caught by Designer software during compile with an error message regarding the failing commands in the GCF file. The Designer software also issues notes when there is a possibility of spine assignment failure. For instance, if a signal assigned to a spine drives a memory block in the design, Designer issues the following note:

```
NOTE [spine_no_core]:
The use_global statement on net clka_c_BUF_AUTOnet in region (97,38 128,73) may not work.
The instance U_mem/M1/TILE6 of type RAM256x9SST/TILE6, driven by this net is not a core cell. And thus this instance may not be accessible from this spine.
```

However, the above message does not necessarily mean that the assignment fails if the RAM block is placed in a location accessible to the spine.

The spine assignment failures can be monitored in layout during routing. If Designer Layout fails to reach the spine drivers for any reason, the following note appears in the Designer log window (a status report shows the same thing):

```
Cannot reach Global Spine for net
clka_c_BUF_AUTOnet
Net is converted to regular resources
```

If the above message does not appear in the Designer report (log), successful assignment is achieved.

Conclusion

ProASIC^{PLUS} devices contain four global networks. However, users can have more than four regional global signals in their design employing spine assignments (up to 88 in an APA1000 die). The global networks of ProASIC^{PLUS} devices can be segmented into local low-skew networks, called spines. The spines provide low-skew networks for high-fanout signals of a design. This document guidelines and methodologies to assign signals to spines. Optimal usage of the available low-skew resources in a ProASIC^{PLUS} device enables users to improve the performance of their designs with the minimum penalty and without modifying the design rerunning synthesis.

Successful assignment can be verified by observing the final layout in the ChipViewer tool. Samples of this approach in which an assigned net is selected and highlighted are shown in Figure 4 on page 6 and Figure 7 on page 10.

Appendix: Scope of the Logic Tiles in ProASIC^{PLUS} Devices

Table 3 • **Top Row Logic Tiles in ProASIC^{PLUS} Devices**

		APA075	APA150	APA300	APA450	APA600	APA750	APA1000
T1	(LL UR)	(1,18 32,32)	(1,26 32,48)	(1,38 32,68)	(1,38 32,68)	(1,54 32,100)	(1,70 32,132)	(1,86 32,164)
T2	(LL UR)	(33,18 64,32)	(33,26 64,48)	(33,38 64,68)	(33,38 64,68)	(33,54 64,100)	(33,70 64,132)	(33,86 64,164)
T3	(LL UR)	(65,18 96,32)	(65,26 96,48)	(65,38 96,68)	(65,38 96,68)	(65,54 96,100)	(65,70 96,132)	(65,86 96,164)
T4	(LL UR)	N/A	(97,26 128,48)	(97,38 128,68)	(97,38 128,68)	(97,54 128,100)	(97,70 128,132)	(97,86 128,164)
T5	(LL UR)	N/A	N/A	N/A	(129,38 160,68)	(129,54 160,100)	(129,70 160,132)	129,86 160,164)
T6	(LL UR)	N/A	N/A	N/A	(161,38 192,68)	(161,54 192,100)	(161,70 192,132)	(161,86 192,164)
T7	(LL UR)	N/A	N/A	N/A	N/A	(193,54 224,100)	(193,70 224,132)	(193,86 224,164)
T8	(LL UR)	N/A	N/A	N/A	N/A	N/A	(225,70 256,132)	(225,86 256,164)
T9	(LL UR)	N/A	N/A	N/A	N/A	N/A	N/A	(257,86 288,164)
T10	(LL UR)	N/A	N/A	N/A	N/A	N/A	N/A	(289,86 320,164)
T11	(LL UR)	N/A	N/A	N/A	N/A	N/A	N/A	(321,86 352,164)

Notes:

1. *LL: Lower Left Corner*
2. *UR: Upper Right Corner*

Table 4 • **Bottom Row Logic Tiles in ProASIC^{PLUS} Devices**

		APA075	APA150	APA300	APA450	APA600	APA750	APA1000
B1	(LL UR)	(1,1 32,15)	(1,1 32,23)	(1,5 32,35)	(1,5 32,35)	(1,5 32,51)	(1,5 32,67)	(1,5 32,83)
B2	(LL UR)	(33,1 64,15)	(33,1 64,23)	(33,5 64,35)	(33,5 64,35)	(33,5 64,51)	(33,5 64,67)	(33,5 64,83)
B3	(LL UR)	(65,1 96,15)	(65,1 96,23)	(65,5 96,35)	(65,5 96,35)	(65,5 96,51)	(65,5 96,67)	(65,5 96,83)
B4	(LL UR)	N/A	(97,1 128,23)	(97,5 128,35)	(97,5 128,35)	(97,5 128,51)	(97,5 128,67)	(97,5 128,83)
B5	(LL UR)	N/A	N/A	N/A	(129,5 160,35)	(129,5 160,51)	(129,5 160,67)	(129,5 160,83)
B6	(LL UR)	N/A	N/A	N/A	(161,5 192,35)	(161,5 192,51)	(161,5 192,67)	(161,5 192,83)
B7	(LL UR)	N/A	N/A	N/A	N/A	(193,5 225,51)	(193,5 224,67)	(193,5 224,83)
B8	(LL UR)	N/A	N/A	N/A	N/A	N/A	(225,5 256,67)	(225,5 256,83)
B9	(LL UR)	N/A	N/A	N/A	N/A	N/A	N/A	(257,5 288,83)
B10	(LL UR)	N/A	N/A	N/A	N/A	N/A	N/A	(289,5 320,83)
B11	(LL UR)	N/A	N/A	N/A	N/A	N/A	N/A	(321,5 352,83)

Notes:

1. *LL: Lower Left Corner*
2. *UR: Upper Right Corner*

Table 5 • **Middle Row Logic Tiles in ProASIC^{PLUS} Devices**

		APA075	APA150	APA300	APA450	APA600	APA750	APA1000
T1	(LL UR)	(1,17 32,17)	(1,25 32,25)	(1,31 32,31)	(1,37 32,37)	(1,53 32,53)	(1,69 32,69)	(1,85 32,85)
B1	(LL UR)	(1,16 32,16)	(1,24 32,24)	(1,36 32,36)	(1,36 32,36)	(1,52 32,52)	(1,68 32,68)	(1,84 32,84)
T2	(LL UR)	(33,17 64,17)	(33,25 64,25)	(33,37 64,37)	(33,37 64,37)	(33,53 64,53)	(33,69 64,69)	(33,85 64,85)
B2	(LL UR)	(33,16 64,16)	(33,24 64,24)	(33,36 64,36)	(33,36 64,36)	(33,52 64,52)	(33,68 64,68)	(33,84 64,84)
T3	(LL UR)	(65,17 96,17)	(65,25 96,25)	(65,37 96,37)	(65,37 96,37)	(65,53 96,53)	(65,69 96,69)	(65,85 96,85)
B3	(LL UR)	(65,16 96,16)	(65,24 96,24)	(65,36 96,36)	(65,36 96,36)	(65,52 96,52)	(65,68 96,68)	(65,84 96,84)
T4	(LL UR)	N/A	(97,25 128,25)	(97,37 128,37)	(97,37 128,37)	(97,53 128,53)	(97,69 128,69)	(97,85 128,85)
B4	(LL UR)	N/A	(97,24 128,24)	(97,36 128,36)	(97,36 128,36)	(97,52 128,52)	(97,68 128,68)	(97,84 128,84)
T5	(LL UR)	N/A	N/A	N/A	(129,37 160,37)	(129,53 160,53)	(129,69 160,69)	(129,85 160,85)
B5	(LL UR)	N/A	N/A	N/A	(129,36 160,36)	(129,52 160,52)	(129,68 160,68)	129,84 160,84)
T6	(LL UR)	N/A	N/A	N/A	(161,37 192,37)	(161,53 192,53)	(161,69 192,69)	(161,85 192,85)
B6	(LL UR)	N/A	N/A	N/A	(161,36 192,36)	(161,52 192,52)	(161,68 192,68)	(161,84 192,84)
T7	(LL UR)	N/A	N/A	N/A	N/A	(193,53 224,53)	(193,69 224,69)	(193,85 224,85)
B7	(LL UR)	N/A	N/A	N/A	N/A	(193,52 225,52)	(193,68 224,68)	(193,84 224,84)
T8	(LL UR)	N/A	N/A	N/A	N/A	N/A	(225,69 256,69)	(225,85 256,85)
B8	(LL UR)	N/A	N/A	N/A	N/A	N/A	(225,68 256,68)	(225,84 256,84)
T9	(LL UR)	N/A	N/A	N/A	N/A	N/A	N/A	(257,85 288,85)
B9	(LL UR)	N/A	N/A	N/A	N/A	N/A	N/A	(257,84 288,84)
T10	(LL UR)	N/A	N/A	N/A	N/A	N/A	N/A	(289,85 320,85)
B10	(LL UR)	N/A	N/A	N/A	N/A	N/A	N/A	(289,84 320,84)
T11	(LL UR)	N/A	N/A	N/A	N/A	N/A	N/A	(321,85 352,85)
B11	(LL UR)	N/A	N/A	N/A	N/A	N/A	N/A	(321,84 352,84)

Appendix: Buffer Insertion

Users can instantiate buffers in either the behavioral source code or the gate-level netlist. Because the buffers are being optimized in Designer software during Compile, the user needs to protect them by using the `dont_touch` command in the GCF file. The syntax is as follows:

```
dont_touch hier_instance_name
[,hier_instance_name … ];
```

For example, if the block U1 of the design hierarchy contains one of these buffers called my_buff, the following is used to protect this buffer from optimization:

```
dont_touch top_level_module/U1/mybuff ;
```

Users may choose to manually add the buffer in their HDL or EDN netlists. This works fine; however, the user still needs to use the dont_touch command in the GCF file in order to prevent this buffer from being optimized away.

List of Changes

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

Revision	Changes	Page
Revision 2 (July 2016)	Non-technical updates.	N/A
Revision 1 (July 2003)	Initial release.	N/A

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
www.microsemi.com

About Microsemi

Microsemi Corporation (Nasdaq: MSCC) offers a comprehensive portfolio of semiconductor and system solutions for aerospace & defense, communications, data center 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; enterprise storage and communication solutions, 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 4,800 employees globally. Learn more at www.microsemi.com.

51900039-2/7.16