AN EFFICIENT IMPLEMENTATION OF SOC FEEDTHROUGH IN AN IP WITH THE MULTIPLE INSTANTIATED BLOCKS (MIB)

[1]Praveen Pilla and [2]VinayaGudeangadi

[1]Physical Design Engineer, Intel Technology India Pvt. Ltd, Bangalore
[2]Soc Design Manager, Intel Technology India Pvt. Ltd, Bangalore

ABSTRACT

At the SoC level, channels are used to interconnect different IPs. When the number of wires passing through the channels is too high, the area occupied by these channels becomes a substantial portion of the total SoC area. The problem with channels is that they only carry signals, making them an inefficient way to utilize design area. To optimize area, IPs are often abutted at the SoC level, and channels are replaced by SoC feedthrough wires that run across the IPs. In SoC designs where IPs are abutted, it is common practice to route feedthrough wires across the IPs to reduce the area dedicated to channels. In an abutted design methodology, the two IPs connected to each other may not be adjacent due to floorplan constraints. This is why SoC feedthrough wires are routed across intermediate IPs. Modern design planning tools can handle the creation of SoC feedthroughs at the IP level. However, in the case of Multiple Instantiated Blocks (MIBs i.e., multiple instances of the same reference), these tools struggle to handle them efficiently and effectively. This paper discusses various scenarios where design tools fail to manage SoC feedthrough creation and proposes solutions to address these challenges.

KEYWORDS

MIB: Multiple Instantiated blocks, SoC: System on Chip, IP: Intellectual Property, WNS: Worst negative slack, FEP: Failing Endpoints, TNS: Total negative slack.

1. INTRODUCTION

Using dedicated channels to carry signals between IPs can have a significant impact on the total SoC area [1]. Instead, utilizing the IPs themselves to carry feedthrough signals is a novel approach to saving SoC area.The following diagram (Figure 1) compares channel-based connectivity with feedthrough-based connectivity.

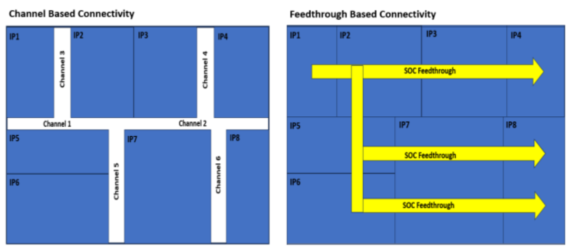


Figure1.Comparison between channel-based connectivity vs feedthrough-based connectivity

The challenge arises when introducing these signals to IPs that contain partitions, which has MIBs. Design tools have not performed efficiently in creating pins for these SoC feedthrough signals, leading to routing & crosstalk issues in the lower nodes [6]. This has required considerable effort to resolve the routing problems and the timing challenges.

The following methodology demonstrates an efficient approach for creating SoC feedthroughs in an IP with MIBs arranged in different scenarios. Figure 2 illustrates how the MIBs are arranged within the IP and the scheme for SoC feedthrough routing.

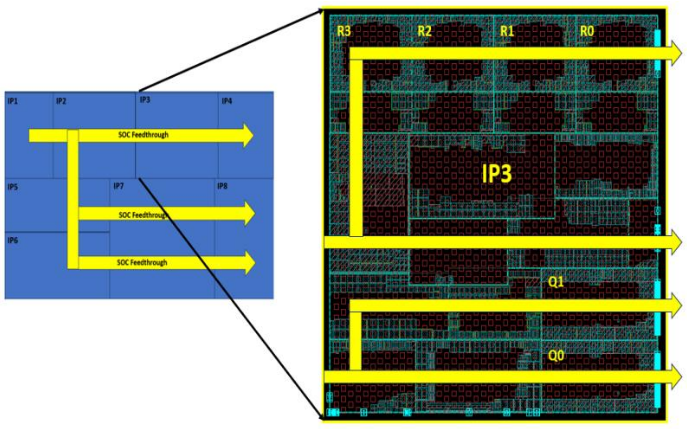


Figure2. Shows the arrangement of MIBs& Soc feedthrough routing.

R0, R1, R2, and R3 are MIBs arranged serially (these are multiple instantiations of the block named R). Q0 and Q1 are MIBs arranged in parallel (these are multiple instantiations of the block named Q). The main challenge is routing SoC feedthroughs across MIBs. The MIB instances in the floorplan can be arranged either serially or in parallel.

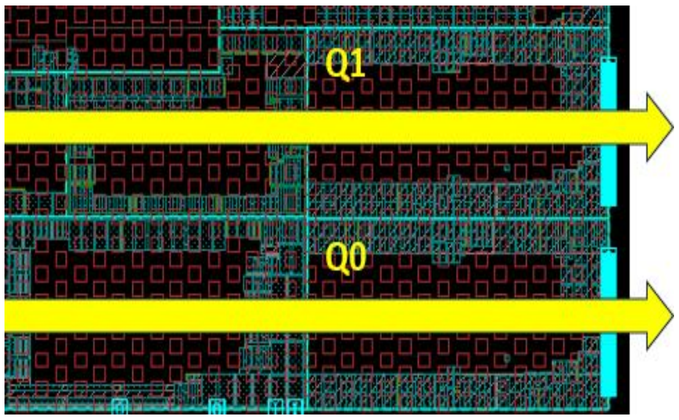
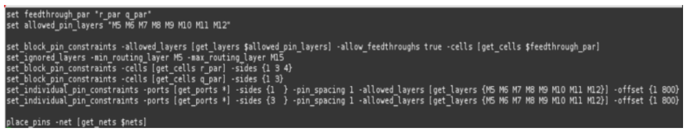


Figure3. Parallel Arrangement of MIBs

Consider a design requirement where we need to route 6,000 SoC feedthrough wires, with the goal of sharing them between the Q0 and Q1 MIB instances of a partition called "Q." During the design planning stage, while placing pins, the following command (Figure 4) was provided to the tool to route 3,000 wires through each instance (Q0 and Q1). However, the tool did not meet the design objective.



```
set feedthrough_par "r_par q_par"
set allowed_pin_layers "M5 M6 M7 M8 M9 M10 M11 M12"

set_block_pin_constraints -allowed_layers [get_layers $allowed_pin_layers] -allow_feedthroughs true -cells [get_cells $feedthrough_par]
set_ignored_layers -min_routing_layer M5 -max_routing_layer M15
set_block_pin_constraints -cells [get_cells r_par] -sides {1 3 4}
set_block_pin_constraints -cells [get_cells q_par] -sides {1 3}
set_individual_pin_constraints -ports [get_ports *] -sides {1  } -pin_spacing 1 -allowed_layers [get_layers {M5 M6 M7 M8 M9 M10 M11 M12}] -offset {1 800}
set_individual_pin_constraints -ports [get_ports *] -sides {3  } -pin_spacing 1 -allowed_layers [get_layers {M5 M6 M7 M8 M9 M10 M11 M12}] -offset {1 800}

place_pins -net [get_nets $nets]
```

Figure 4. Sequence of commands for the pin creation using tool.

The tool placed 3,000 ports on the left and right edges of Q0 to connect the wires passing through itand correspondingly placed 3,000 ports on the left and right edges of Q1, since Q0 and Q1 are MIBs. Similarly, the tool added an additional 3,000 ports on the left and right edges of Q1 to connect the wires passing through itand correspondingly placed 3,000 ports on the left and right edges of Q0.Effectively, instead of routing 6,000 SoC feedthrough wires by sharing 3,000 wires between the two MIB instances, the tool inserted 6,000 wires into each individual instance. This is primarily because the design planning tool does not recognize that the MIBs can share the same 3,000 wires created in each instance to route the 6,000 SoC feedthrough wires.

2. PROPOSED SOLUTION

In this paper, we propose a solution for creating SoC feedthroughs in parallel arrangement scenarios of MIBs during the design planning stage.

2.1. Parallel Arrangement of MIBs

Consider the example of routing "m" number of SoC feedthrough wires across the Q0 and Q1 instances. To begin, assess how many SoC feedthrough wires need to be routed and determine their composition (i.e., how many input and output wires need to be routed).

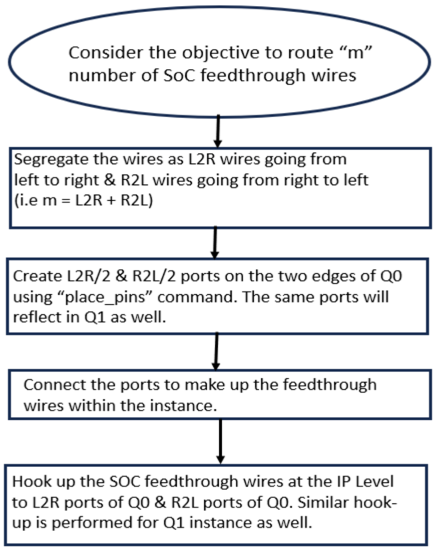

Consider the objective to route “m” number of SoC feedthrough wires
Segregate the wires as L2R wires going from left to right & R2L wires going from right to left (i.e m = L2R + R2L)
Create L2R/2 & R2L/2 ports on the two edges of Q0 using “place_pins” command. The same ports will reflect in Q1 as well.
Connect the ports to make up the feedthrough wires within the instance.
Hook up the SOC feedthrough wires at the IP Level to L2R ports of Q0 & R2L ports of Q0. Similar hook-up is performed for Q1 instance as well.

Figure 5. Flowchart

Assume the "L2R" SoC feedthrough wires go from the left edge of the IP to the right edge, and the "R2L" SoC feedthrough wires go from the right edge of the IP to the left edge. Understanding the segregation of input and output wires and dividing them optimally between the MIB instances is key to utilizing routing resources efficiently in this methodology.

The guidelines to follow when using MIBs arranged in parallel for SoCfeedthroughare:

i. The number of input and output signals passing through each instance is the same.
ii. The order of input & output signals in each instance is the same.

Given the Q0 and Q1 instances of the MIB, each instance can carry “L2R/2” and “R2L/2” wires in each direction. Considering Q0 as the reference instance, create “L2R/2” input ports on the left edge of Q0 and corresponding “L2R/2” output ports on the right edge of Q0. Similarly, create “R2L/2” output ports on the left edge of the Q0 instance and corresponding “R2L/2” input ports on the right edge of Q0. Once the feedthrough ports are created on both edges of the MIB instance, they need to be connected to form the feedthrough wire within the instance. Since Q0 is an MIB, the same number of ports in the same order are created in Q1 as well, and the feedthrough wire connections are also inferred as shown in Figure 6.

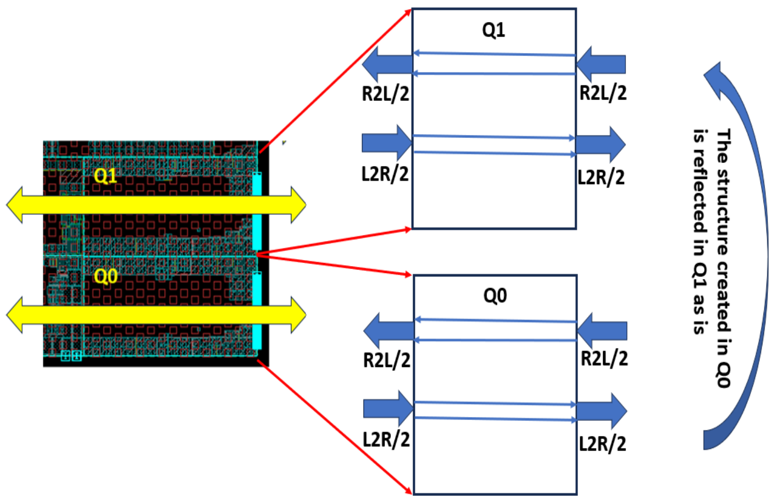


Figure 6.Demonstrating the proposed pin placement for parallel arrangement of MIBs.

At this point, shift the focus to connecting the pins of the MIBs to the SoC feedthrough wires at the IP level. First, connect the “L2R/2” SoC feedthrough wires to the input ports of Q0 on the left edge, and on the right edge of Q0, hook up the same wires to the output ports. Similarly, connect the “R2L/2” SoC feedthrough wires to the input ports of Q0 on the right edge, and on the left edge of Q0, hook up the same wires to the output ports. Now, knowing that Q1 has the same number of ports in the same order, connect the remaining SoC feedthrough wires in the same fashion.

As mentioned earlier, the key is to segregate the SoC feedthrough wires between the MIB instances. For example, consider routing signals with a wide and a narrow SoC feedthrough bus. If the timing criticality is such that these signals cannot be split between the two MIB instances, then accept the routing compromise by allocating the wider bus signals to one instance and the narrower bus signalto the other. In such cases, the wider bus signal dictates the maximum number of wires that can pass through both instances.

3. Experimental Results

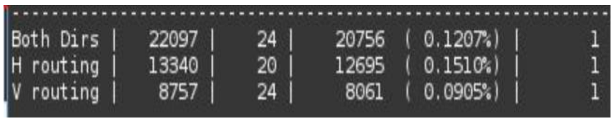
3.1. Parallel Arrangement of MIBs

The objective of this SoC design was to insert 6,000 SoC feedthrough wires using two parallelly stacked MIB instances. When the tool was used to create these feedthrough wires, it resulted in

6,000 wires in each MIB instance.Using the proposed approach, the design is able to route 6,000 SoC feedthrough wires by creating only 3,000 feedthrough wires in each MIB instance. This significantly reduces block congestion and improves timing, as there is less impact due to crosstalk.

3.1.1. Block Congestion & Timing Summary with Traditional Method

Figure 7. shows the total congestion number at place stage which is 0.12%. This congestion score is translating to ~900 shorts in the design after the actual routes.It is time consuming to fix these shorts and timing is also bad (shown in Table 1) due to crosstalk impact.



```
-----------------------------------------------------------
Both Dirs |   22097 |    24 |    20756  ( 0.1207%) |      1
H routing |   13340 |    20 |    12695  ( 0.1510%) |      1
V routing |    8757 |    24 |     8061  ( 0.0905%) |      1
```

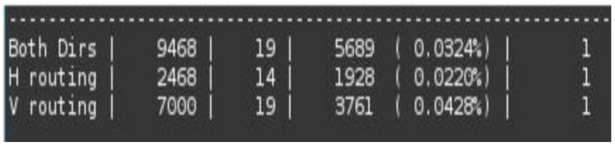
Figure 7. Congestion report from traditional method

Table1.Timing summary from traditional method.

WNS	TNS	FEP
-0.211	-118.77	4588

3.1.2. Block Congestion &Timing Summary with Proposed Method

Figure 8. shows the total congestion number at place stage which is 0.03%. Significant improvement compared to the traditional method. With this approach shorts count has come down to ~150 after the actual design routes. Timing summary has improved as shown in Table 2.



```
-----------------------------------------------------------
Both Dirs |    9468 |    19 |     5689  ( 0.0324%) |      1
H routing |    2468 |    14 |     1928  ( 0.0220%) |      1
V routing |    7000 |    19 |     3761  ( 0.0428%) |      1
```

Figure 8. Congestion report with proposed method.

Table 2.Timing summary with proposed method.

WNS	TNS	FEP
-0.0873	-13.33	2070

3.1.3. Congestion Score Comparison

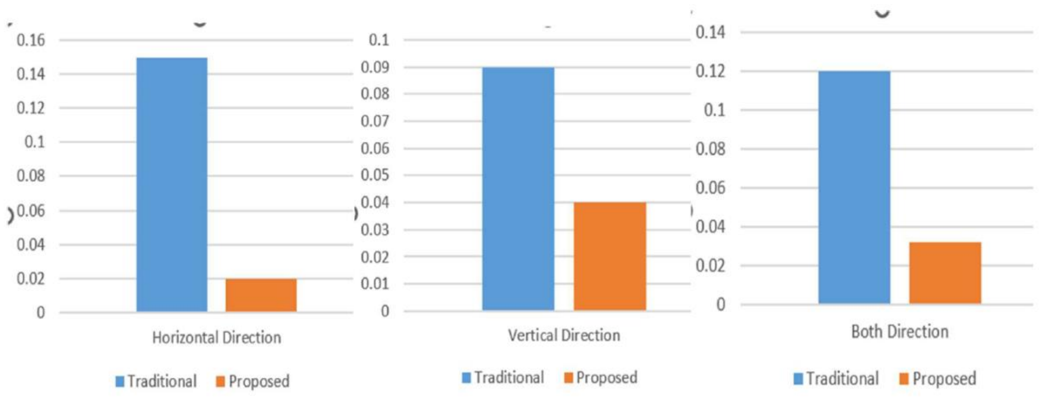


Figure 9.Congestion score comparison b/w Traditional vs Proposed method.

Figure 9. shows the Congestion score comparison between traditional and Proposed method. Overall Congestion has improved from 0.12% to 0.032%, which is significant in lower-node technology.

3.1.4. Timing Summary Comparison

Figure 10. shows the timing summary comparison between traditional method vs proposed method. It is evident that the proposed method shows good improvement on timing.The total negative slack (TNS) has improved from -118 ns to -13 ns, which facilitates better routing in the design.

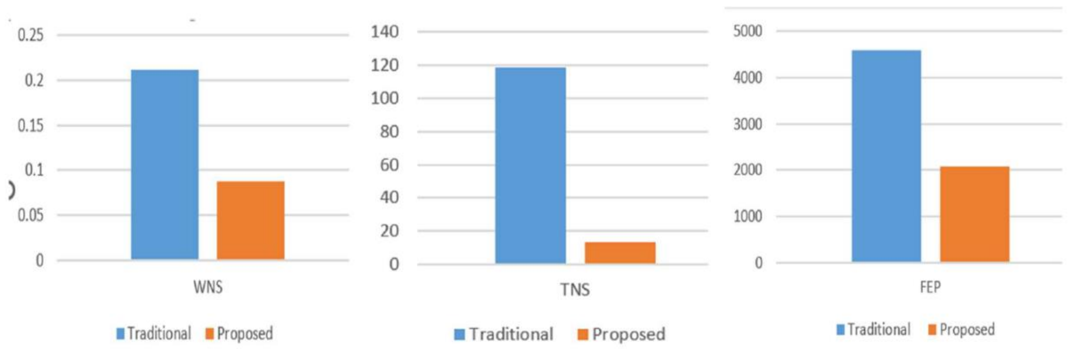


Figure 10.Timing summary comparison b/w Traditional vs Proposed method.

4. CONCLUSION

The proposed approach can be used to enable SoC feedthrough wires even through MIB instances. This approach covers the best possible arrangements of the MIBs and the corresponding pin creation/placements, which in turn helps improve the timing and routing of the design. The successful adaptation of this approach will enable SoC feedthroughs to avoid the presence of inefficiently used channels, optimizing the area utilization of the SoC.

REFERENCES

[1] Yi Hong, Chunyang Huang, Yue Gao, Chuang Li, “Channel Based SOC Feedthrough Insertion Methodology”, 2022 11th International Conference on Communications, Circuits and Systems.

[2] Ashutosh Kumar, Michael Carver, “Automatic Conversion of a channel based design to an abutted design”, SNUG 2017.

[3] S. Hiremath and V. Y, "RTL to GDSII implementation of Advanced High-Performance Bus Lite", 2021 6th International Conference on Communication and Electronics Systems (ICCES), pp. 303-306, 2021.

[4] Yanling Zhou, Yunyao Yan , Wei Yan, “A method to speed up VLSI hierarchical physical design in floorplanning”, 2017 IEEE 12th International Conference on ASIC, pp. 347-350, 2017.

[5] FC User Guide: Implementation, M-2016-sp4ed., Synopsys Inc.

AUTHORS

Praveen Pilla is currentlya Senior Physical Design Engineer at Intel, bringing over 12years of expertise in Physical Design and Timing Closure. His work focuses on high-performance subsystems and SoC, where he contributes to the development and optimization of advanced semiconductor solutions. He has a strong passion to solve critical problems.

Vinaya Gudeangadi is a SOC Design Manager at Intel with 24 years of experience in the field of VLSI having worked in both front-end (RTL) & back-end (Physical design). He has worked on networking, storage, media, modem, IOTG & automotive SoC designs while leading teams. He has the experience of working on Intel, TSMC, Samsung & GF foundries & implementing various processor cores - ARM (CPU/GPU/N2 Cores), Xtensa, Imagination GPU.