A Novel Feedthrough Insertion Methodology for Hierarchical SOC Designs: Achieving Reduction in Die Area

Rajanikant Sakariya
Subhadeep Aich
Vivek Joshi
Roger Griesmer

Texas Instruments

62 THE CHIPS TO SYSTEMS CONFERENCE

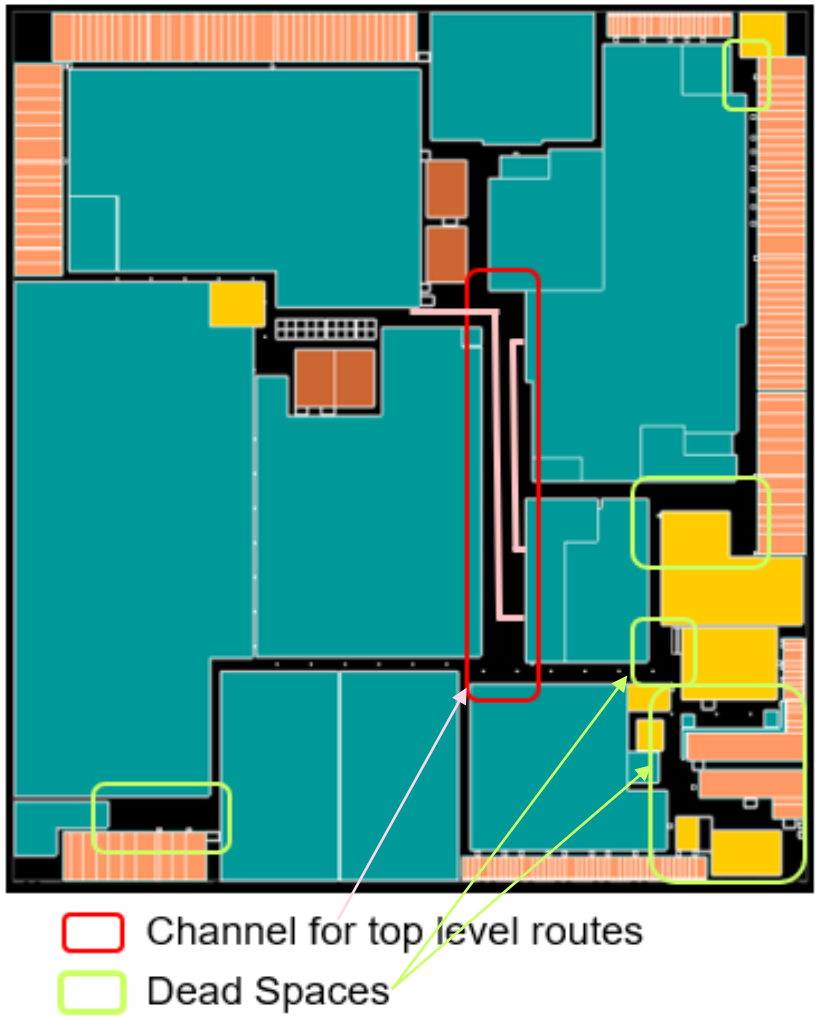
SPONSORED BY CEDA IEEE Council on Electronic Design Automation sigda acm

Rajanikant Sakariya

Physical Design Engineer

Texas Instruments Incorporated

Problem Statement | Motivation



- The reduction of die size has been a primary goal in systems-on-chip (SoC) designs
- When adopting a hierarchical design strategy in physical design implementation, it's essential to **create sufficient routing channels** within the top level floorplan. It's quite -
  - Challenging to **reduce the die area** by **decongesting** and **shrinking** route channels
  - Challenging to reclaim **unused or dead spaces**
  - Challenging to push top **route factor** beyond 80-82%
- While previous approaches focused on reducing congestion of the chip as a whole, there has been little effort to specifically alleviate congestion in the top channel
- **Motivation**
  - In channel-based hierarchical SoC designs, inserting feedthroughs can be a crucial step during the floorplan stage to enhance routing resource utilization, alleviate congestion in the top channels, and ultimately **save die area**

Main Idea(1/4) | Proposed Feedthru Insertion Methodology

- The figure illustrates how the proposed methodology is implemented at the top and subchip level physical design cycle
- This methodology, ensuring superior efficiency and accuracy in the implementation, provides an innovative solution for
  - Feedthrough Identification
  - Feedthrough Insertion
  - Feedthrough collateral generation for subchips and its implementation
  - Logical equivalence closure checkout (LEC)
- The proposed methodology eliminates the need for coordination between the RTL and physical design
- As no feedthrough information is sent back to the RTL, achieving logical equivalence with the RTL presents a unique challenge in this case
- The proposed methodology tackles all the challenges associated with feedthrough insertion such as the high dependency on RTL, the integration of new feedthrough ports at the subchip level, and the verification of logical equivalence closure (LEC)
- It showcases die area savings resulting from the reduction of the top channel area in a hierarchical SoC design

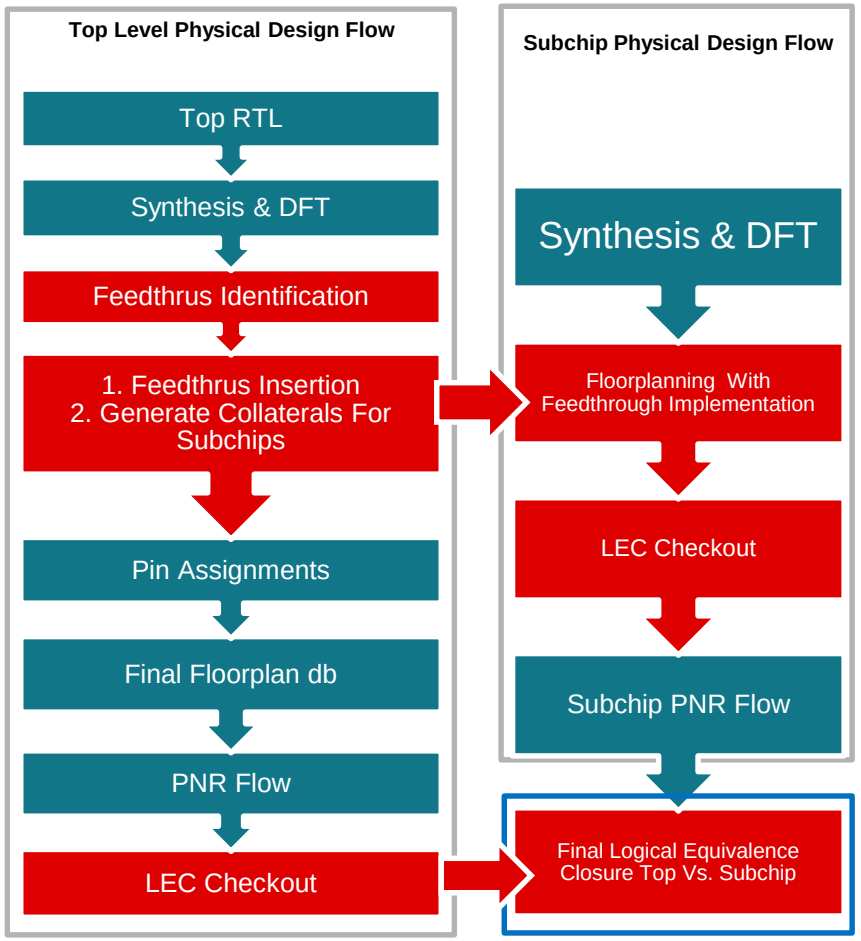


Figure: A typical top and subchip-level physical design flow with stages proposed methodology changes highlighted in red

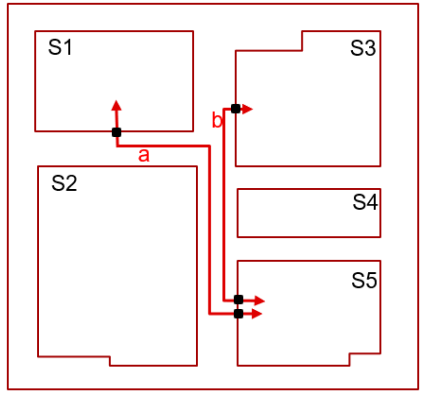
Main Idea(2/4) | Feedthrough Identification Algorithm

1. *Algorithm IDENTIFY_FEEDTHRU () {*
2. *Create FeedthruLookupTable {subchip_i subchip_j } = {visually identified feedthrough subchips between i and j}*
3. *For each subchip i in the design {*
4. *For each subchip j in the design {*
5. *Get feedthrough subchip list from FeedthruLookupTable for {i j}*
6. *GetPoint2PointPinList between (i j)*
7. *Create FeedthruSignalList.Table ()}}*
8. *}*

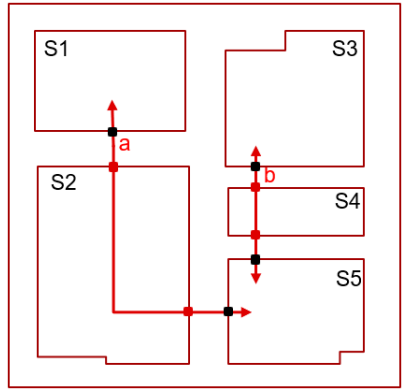
Step1 *FeedthruLookupTable { S3 S5 } = { S4 }*
FeedthruLookupTable { S1 S5 } = { S2 }

Step2

FROM SUBCHIP	PIN1	TO SUBCHIP	PIN2	FEEDTHRU SUBCHIP
S3	Pin1	S5	Pin2	S4
S1	pin3	S5	pin4	S2
S3	pin5	S4	pin6	NONE



No Feedthrough



Post Feedthrough

Step1 : IDENTIFY_FEEDTHRU algorithm begins by **creating a FeedthruLookupTable** as depicted above. An entry in the lookup table indicates that “FEEDTHRU SUBCHIP”(eg. S4) serves as a feedthrough for signals traveling from “FROM SUBCHIP”(eg. S3) to “TO SUBCHIP”(eg. S5) . The identification of “FEEDTHRU SUBCHIP” is done by visually examining the physical view of the initial floorplan database along with the signal's over-the-subchip route topology

Step2: After creating the lookup table, the GetPoint2PointPinList between subchips (i, j) iterates over each subchip pair to find point-to-point signals. It then queries the feedthrough subchip entry in the FeedthruLookupTable corresponding to the subchip pair. Finally, the **FeedthruSignalList.Table() create**s above table with all the collected information required for the feedthrough Insertion algorithm discussed in the next slide

Main Idea(3/4) | Feedthrough Insertion Algorithm

Feedthrough Insertion Algorithm

Step 1) Create topological file

```
1.  Algorithm CREATE_TOPO_FILE() {
2.    For each entry in FeedthruSignalList.Table {
3.    Create TopoFileInToolSupportedFormat ($entry) }
4.  }
```

```
#topological _file format
net net1
  hterm-hterm S3/pin1 S4/ft_in_pin1;
  hterm-hterm S4/ft_in_pin1 S4/ft_out_pin1;
  hterm-hterm S4/ft_out_pin1 S5/pin2 ;
end net
```

Step 2) Insert feedthroughs

```
1.  Algorithm CREATE_FEEDTHRU($topo_file) {
2.  EDA tool(Innovus) command creates feedthrus on
    subchips as per topological file description
3.  }
```

- **Step1 generates a topological file** supported by EDA tools using the *FeedthruSignalList.Table* created in previous section
- The topological file contains the new topology for each of the feedthrough nets. As depicted in center figure, feedthrough net1, originally attached to *S3/pin1* and S5/pin2, now connects through S4 by creating two feedthrough pins, ft_in_pin1 and ft_out_pin1, on the boundary of S4

- **Step2** algorithm CREATE_FEEDTHRU() parses the topological file generated in Step1. The EDA tool command **creates feedthroughs** according to the net descriptions provided in the topological file

Feedthrough Collaterals Generation for Subchip

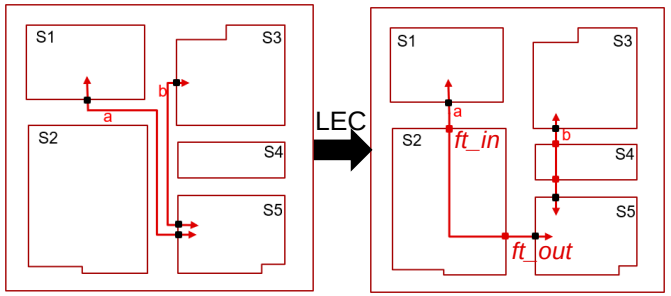
- Subchip Blackbox Verilog Netlist

```
module S4 ( );
Input ft_in_pin1;
Output ft_out_pin1;
<original ports list>
assign ft_out = ft_in;
endmodule
```

- This step involves generating deliverables for the standalone implementation of the subchip
- The “Subchip Blackbox Verilog Netlist” is generated using EDA tool command and provided to the subchip PD execution flow
- As shown above, the Subchip Blackbox Verilog Netlist includes only the original ports, new feedthrough ports, and their connectivity information
- The subchip PD execution flow parses the provided blackbox netlist to create an ECO file with new feedthrough ports information
- This ECO is implemented on the fly during subchip PD implementation ensuring feedthrough ports are punched on subchip hierarchy along with its connectivity

Main Idea(4/4) | Proposed Logical Equivalence Closure Methodology

Top level LEC Checkout



Golden (original dft netlist without feedthrus)

Revised (post feedthrus insertion)

Problem:

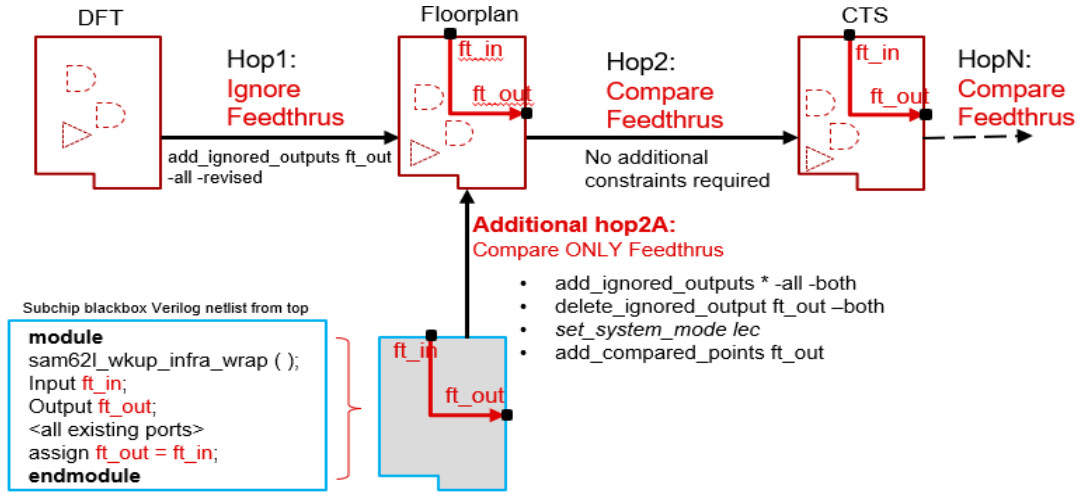
- All subchips are treated as **black-box** in top-level LEC runs. This approach is preferred in hierarchical designs as it **speeds up the runtime**
- The **topology of the S2** subchip has been modified to include two new ports and a connecting net, which were added through feedthrough insertion
- However, this new **net connectivity is not visible** due to the black-box nature of the subchip, resulting in LEC status of NONEQ

Solution:

- Additional constraints are needed to indicate that the new feedthrough pins are equivalent

1. *add_pin_equivalence ft_in -INPUT_OUTPUT ft_out -module S2 -revised*
2. *add_ignored_inputs ft_in -module S2 -revised*

Subchip level LEC Checkout



Problem:

- Since subchip PD implements feedthrus during the floorplan stage, the DFT netlist does not have information on new feedthrough pins that were added. This leads to a mismatch in DFT versus Floorplan LEC, resulting in NONEQ
- To ensure the correctness of feedthrough implementation within a subchip during the floorplan stage, the flow should be able to detect any inversions, criss-cross connections, or missing feedthrough connections

Solution:

- Ignoring feedthrough comparison at DFT vs Floorplan LEC solves this problem: *add_ignored_outputs ft_out -all -revised*
- We need an additional LEC hop comparing the “Subchip black box netlist from the top” with the “Floorplan netlist.”
- Compares **ONLY** feedthroughs, ignoring all other compare points*: add_compared_points ft_oot*
- Any incorrect implementation will be identified here (inversion, criss-cross, missing feedthroughs etc)

Results Summary | Top SoC Level

- The design under test is a **16ffc** device with **500K** gate count at top level and around 20 subchip
- **Figure** illustrates the reduced top channel after feedthrough insertion. Using the feedthrough identification algorithm, we identified 4722 top level signals that require feedthroughs. By pushing these signals into subchips instead of routing them around the subchips, the routing **channel requirement** dropped from **200um** wide to **120um**.
- Proposed methodology led to a **3% reduction in die size**. We also demonstrated that post feedthrough insertion, the design is fully routable with manageable shorts and DRCs. No negative impact was observed on top-level timing closure post feedthrough implementation

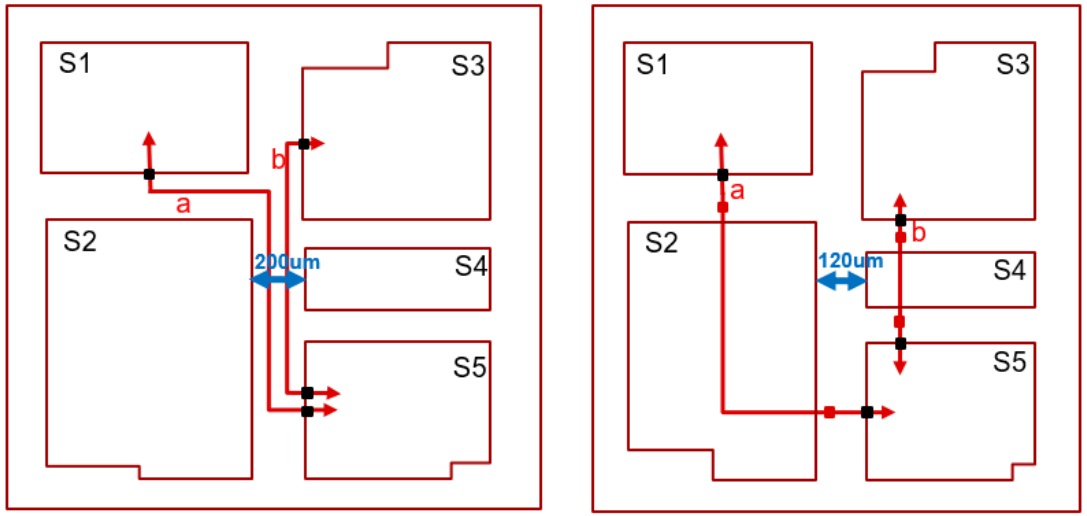


Figure: *Top channel without feedthrough vs. Top channel reduction with feedthrough insertion*

	NO Feedthru Approach	Proposed Methodology
Top channel width	200 um	120 um
Routes present in top vertical channel	5500	3550
Top level buffer count	213624	172992 (38k reduction)
Timing metrics	Baseline	Similar
Metal Shorts	< 5	< 10
Runtime impact	Baseline	Additional 1.5 hours for feedthrough implementation and 1 hour for LEC runs
Top level route factor	84%	93% (Indicates that only 7% die area is accounted for top channels + dead spaces)
Die area	14.32 mm2	13.91 mm2 (0.41mm2 die size reduction)

Table: *Comparison of top-level results, without and with using proposed methodology*

Results Summary | Subchip Level

- We also demonstrated the seamless execution of subchip level physical design flow using the proposed methodology. We chose subchip S4 for the case study because a large portion of the 4722 top signals identified as feedthroughs, 1563 signals are to be feedthrough in S4 alone. This requires a total of 3126 (i.e. 1563 x 2) feedthrough ports to be punched on the S4 hierarchy, as each feedthrough signal requires 2 ports, an input, and an output.

- **Figure** highlights the feedthrough ports punched on the subchip and the buffers added on feedthrough nets during PNR cycle. There is no significant degradation seen at subchip level. As shown in Table 3, design is fully routable with manageable shorts and DRCs. Design is timing clean without impacting the runtime.

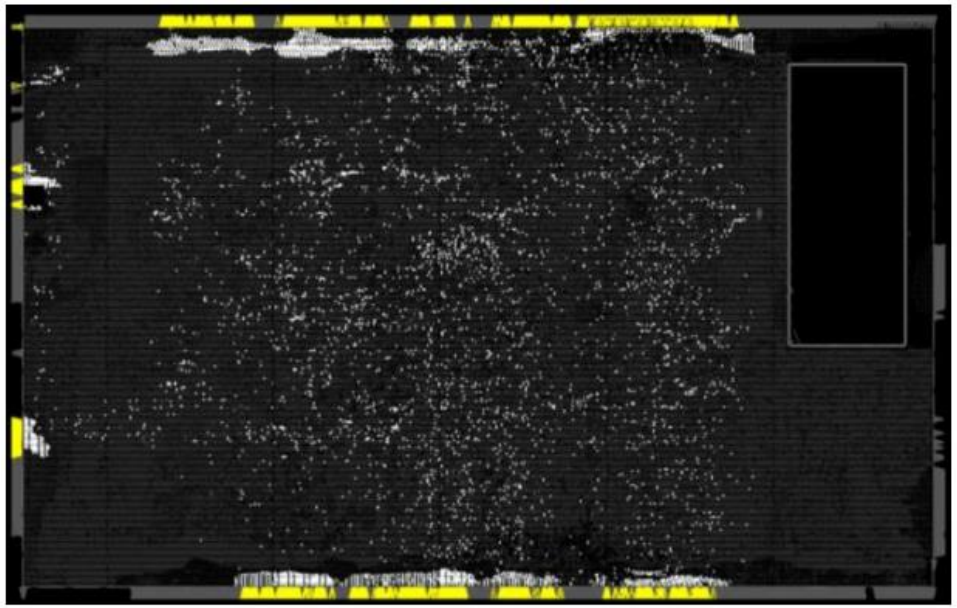


Figure: *Feedthrough ports created (yellow) and buffers added on feedthrough nets (white) at subchip level*

	NO Feedthru Approach	Proposed Methodology
# Feedthrough signals added	NA	3126
Feedthrough buffer count	NA	8154
Subchip stdcell utilization	76%	78%
Timing metrics	Baseline	Similar
Metal Shorts	< 5	< 10
Runtime impact	Baseline	Similar

Table: *Comparison of subchip level results, without and with using proposed methodology*

Conclusion

- A novel feedthrough insertion methodology is presented

- The proposed methodology -
  - Specifically focuses on alleviate congestion in the **top channel**, and ultimately **save die area** without compromising signoff QoR (Quality of Results)
  - **Eliminates the need for coordination** between the RTL and physical design
  - Provides **seamless integration** of new feedthrough ports at the subchip level
  - Proposes **foolproof logical equivalence checkout** methodology
  - Is independent of **technology**, making it reusable across all technology nodes