

DesignCon 2006

Back-End Methodology and Techniques for a Multi-Protocol Mixed Signal IP Design

Ken Umino, Synopsys Inc. Professional Services
kumino@synopsys.com

Jason Upton, Synopsys Inc. Solutions Group
jupton@synopsys.com

John Stonick, Synopsys Inc. Solutions Group
jts@synopsys.com

Ross Segelken, Synopsys Inc. Solutions Group
roos@synopsys.com

Bill Beale, Synopsys Inc. Solutions Group
beale@synopsys.com

Abstract

ASIC I/Os have been evolving from relatively slow parallel interfaces to high-speed serial links. Examples include PCI to PCI Express (2.5Gbps), ATA to SATA (1.5-3Gbps), and SCSI to SAS (1.5-3Gbps). Additionally, the XAUI interface, which operates at 3.125Gb/s, has gained popularity as an ASIC interface. This evolution has created a difficult design problem for ASIC manufacturers that is increasingly being solved by purchasing rather than developing the required I/O IP. While the analog design aspects of such a development are daunting there is an equally difficult problem for the digital back-end. In this paper we will focus on the back-end challenges and solutions in development of a physical layer, mixed-signal IP (PHY) that supports three standards: PCI-Express, SATA, and XAUI. The PHY is currently designed in TSMC 90nm and 130nm, specifically TSMC90G, TSMC90GT, TSMC130G, TSMC130LV, and TSMC130LVOD recently developed by Synopsys' Hillsboro PHY Group and design services business unit.

Author(s) Biography

Ken Umino is a Consultant with Synopsys Professional Services. He earned a BSEE and a MSEE from the University of Washington (Seattle, Wa). Since then he has been involved with numerous chip design projects as a ASIC, RTL and circuit designer for over 15 years. Most recently, in the past 6 years he has been employed with Synopsys working on various aspects of timing closure, physical design, and synthesis.

Ross Segelken graduated with a Bachelor's degree in Electrical Engineering from the California Institute of Technology in 1997. He worked for five years at Intel Corporation on micro-architecture and circuit design for the Pentium 4 and next generation IA32 processors. He joined Accelerant Networks in 2002 and has worked on architecture and digital design for multiple generations of Serdes and Physical layer implementations. He joined Synopsys, Inc through an acquisition in 2004, and currently leads all digital design activities in the Hillsboro PHY group.

John T. Stonick received his Ph.D. in ECE from North Carolina State U. in 1992. From 1993-1997 he held a postdoctoral research position in the ECE department at Carnegie Mellon University. From 1997-2000, he was an assistant professor with the ECE department at Oregon State University and a co-director for the NSF Center for the Design of Analog-Digital Integrated Circuits (CDADIC). Starting in 2000 he was a principal design engineer with Accelerant Networks until they were acquired by Synopsys in 2004. Since 2004 he has remained with Synopsys where he holds the title of Synopsys Scientist and was the 2004 Synopsys Distinguished Inventor. He is a member of the IEEE and a

member of the ISSCC 2006 technical program committee (wireline). He has been a co-author on 2 IEEE best paper awards: 2006 IEEE CICC Best Invited Paper and 1994 Matti S. Sukola Award for best paper at the IEEE Broadcast Symposium. His interests include system architecture, clock and data recovery, and using adaptive digital techniques to compensate for analog circuit imperfections in wireless and wireline transceivers.

Jason Upton recently joined the Synopsys Hillsboro PHY group to perform RTL to GDS implementations of high-speed SERDES PHYs. He obtained a BSEET from Oregon Institute of Technology in 1993 and has completed many post-graduate courses at Oregon State University. He has previously worked for Hewlett-Packard, Agilent Technologies and Avant! He has been an independent consultant for RTL to GDS services the past five years before joining Synopsys in September 2005.

Bill Beale is an Application Engineer for the Hillsboro PHY group of Synopsys. He earned a BSEE from Santa Clara University in 1992. He worked for 5 years on electronic packaging, specifically the electrical modeling aspects. From there it lead into working for Ansoft as an application engineer supporting the signal integrity tools. In 2000, he joined Accelerant Networks and continued signal integrity work on high speed signals of serial interfaces and moved into application support. He joined Synopsys, Inc through an acquisition in 2004, and is currently the point of support for both internal and external customers.

Introduction

To conserve on schedule and engineering resources, the challenge for the Synopsys Hillsboro Phy design team was to develop a single design that could support multiple I/O protocols, and at the same time, work over wide process corners. One challenge was developing a single RTL to support both 90nm and 130nm designs so as to reduce the required design and verification effort. The RTL was written to be able to guarantee operation at 312.5MHz for the slowest node, TSMC130G, while at the same time the back-end implementation tools reference the fast corner of the fastest node to fix hold violations. This approach virtually eliminates the need for manual fixes to close timing. Another important aspect in closing timing was the constraint strategy. Constraints were embedded in the RTL as comments, and then extracted out later into a constraint file. Combining the constraints with RTL allows team members to better understand the code of a designer. Also, the constraints detect which phase of the flow they are at in order to make adjustments needed. For example, during implementation stages such as placement, clock tree synthesis, and route, the clock periods and clock uncertainties were tightened to add margin. During sign-off static timing analysis (STA), the clock periods and clock uncertainties are relaxed to sign-off levels. In STA, crosstalk is accurately measured and on-chip-variation is used with clocks derated by 25%.

To facilitate physical design a modular approach was used. There were four sub-blocks: an analog transceiver block, a digital transceiver block, an analog clock block, and a digital clock block. The modular approach allowed multiple configurations comprising analog and digital blocks to quickly be created. For example, 1, 2, 4, 8, 12, and 16 lane configurations were easily created from the sub-blocks by abutment. At the lane configuration or top level, careful planning was done to ensure that the long feed-thru nets spanning across the lane blocks had minimal signal integrity and transition problems. Furthermore, drivers and receivers were bounded to be directly underneath the feed-thru nets to ensure that RC effects are minimized. At this top level, only via insertion is necessary to program the I/O protocol that is desired. The end result is a single design that the IP design team needs to focus on, while simultaneously reducing risk for a customer who only has to rely on one IP solution (rather than multiple IP's) for multiple protocols.

Basic architecture description

The PHY, created by the Synopsys Hillsboro PHY group, is a low power, small area, and high performance core that can be configured to support PCI Express 1.1, SATA 2, or XAUI protocols. It can also support a number of forms consistent with independent selections:

- Core technology: 1.0 or 1.2V transistors
- IO Transistor technology: 2.5V or 3.3V
- M6 thickness: thick or thin
- Bond options: 55um linear wirebond or 210um flipchip
- Number of lanes: x1, x2, x4, x8, x12, x16

Below, Figure 1 shows a block diagram of a x1 configuration. On the left is the PAD interface, while on the right is the ASIC interface.

The top block, the Transceiver Lane, has both analog and digital circuits. For transmitting data, the Transceiver lane serializes the parallel data from tx_data[*] into a single bit stream, and sends this data off chip on a differential pair, Txp and Txm. For receiving data, the Transceiver lane deserializes the single serial bit stream on the differential pair Rxm and Rxp into parallel data on rx_data[*]. Tx_data[*] and rx_data[*] can be configured to be 20 bits wide at a slower clock frequency or 10 bits wide at a faster clock frequency for the ASIC interface. 10 bits represent 1 coded byte for all the supported protocols. The receive path uses a slicer block which converts the high-bandwidth analog data to a serial sequence of binary data. The CDR or clock and data recovery block, uses a digital PLL to control the analog PLL which generates the clocks to feed to the slicers and is the source of the recovered clock which becomes the digital word rate clock.

The bottom (control) block handles the common ASIC interface and is the master control for the PHY. It includes a multiplying PLL (MPLL) that generates the required reference clock to be buffered throughout the transceivers. The JTAG block in the clock and control block allow for testing of the device.

There are a number of test features that the PHY has implemented. For example, the transmit channel can be looped back to the receive channel. Bit error rate testing (BERT) creates a pseudo-random pattern for error testing. A high resolution internal eye measurement is available on each receive signal.

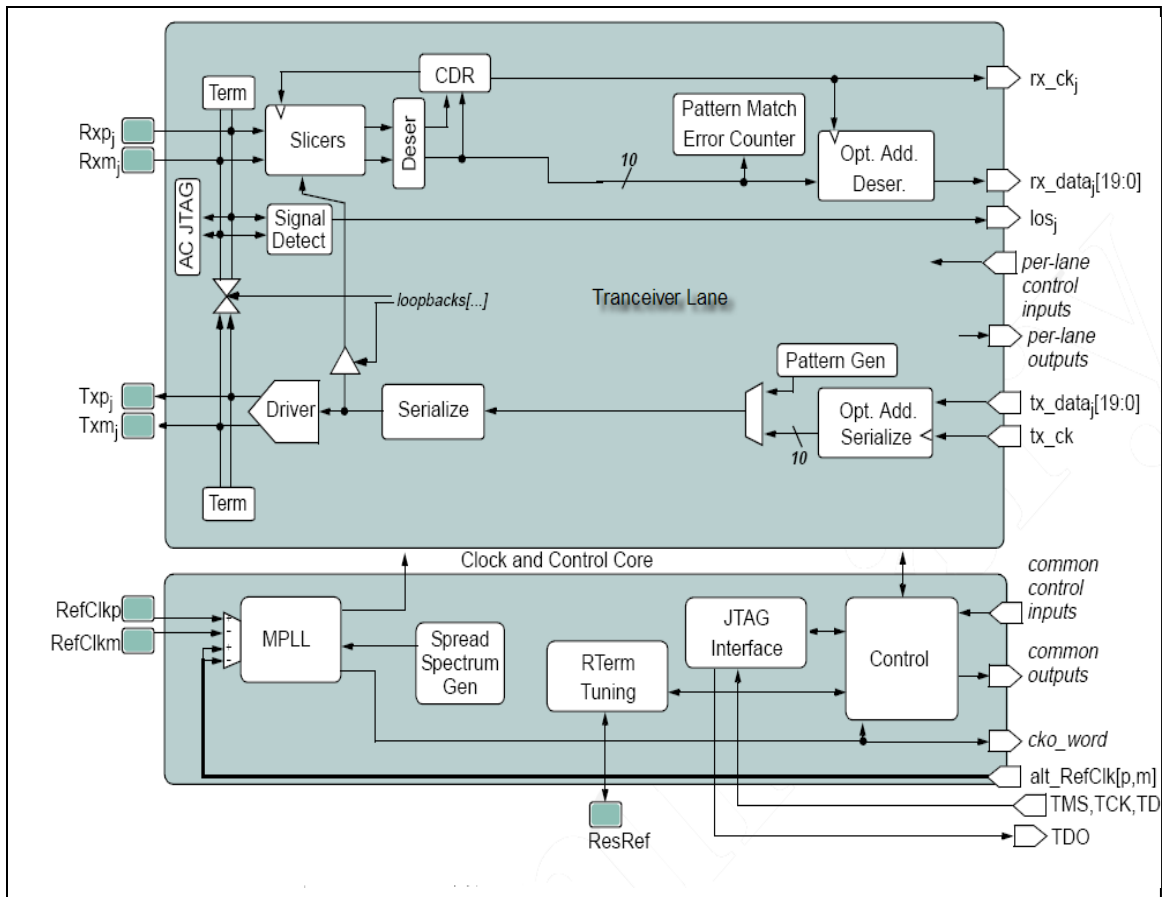


Figure 1 - Diagram of a PHY Lane and PHY Support Block

Sub-block and lane configuration description

Logically, the PHY consists of the transceiver (or lane) block and the clock and control block. In implementation, these two logical blocks are each broken down into digital and analog block shown below:

- Transceiver block
 - Digital transceiver lane block: up3_topd
 - Analog transceiver lane block: up3_topa
- Clock and Control block
 - Digital clock and control block: sup_topd
 - Analog clock and control block: sup_topa

The physical implementations of the digital block for 130nm and 90nm are different, however, they are based on the same RTL and have the same pin locations. The flows are also nearly identical except for the following: They have different standard cell libraries,

LVS/DRC runsets, and timing constraints. The timing constraints are similar, but differ in electrical design rules like transition times, maximum capacitance, and clock skew target. Also, the timing constraints on some of the pins are adjusted to optimize synthesis. The slowest 130nm node is about 15% slower than the slowest 90nm node. The 130nm and 90nm sub-blocks share the same pins and physical width dimensions, but have different heights. The 130nm and 90nm analog blocks have exactly the same pin locations, and the same width and height dimensions. One reason the analog 130nm blocks and 90nm blocks are identical in size, unlike the digital blocks, is because the analog blocks share much of the same physical designs and are custom designed at the transistor level. Also, much of the analog block is designed with the same I/O transistor which does not change significantly between 130nm and 90nm technologies. On the other hand, the digital 130nm blocks and 90nm blocks are designed at the standard-cell level, and today's ASIC flows can't simultaneously work on different standard-cell libraries in different technology nodes. Having just 4 sub-blocks that can be assembled to create all lane configurations conserves on the amount of blocks that need to be designed and maintained.

Additionally, from a timing perspective, the PHY supports wide ranging technology nodes, as well as the worst, nominal, and best process-voltage-temperature (PVT) corners. Table 1 shows the 130nm nodes and PVT corners supported by the 130nm design. Table 2 shows the 90nm nodes and PVT corners supported by the 90nm design.

Process	Best case voltage/temp (V/C)	Nominal case voltage/temp (V/C)	Worst case voltage/temp (V/C)
TSMC130G	1.32V/ 0 C	1.2V / 25 C	1.08V/ 125 C
TSMC130LV	1.1V/ 0 C	1.0V/ 25 C	0.9V/ 125 C
TSMC130LVOD	1.26V/ 0 C	1.2V/ 25 C	1.08V/ 125 C

Table 1 - 130nm nodes and PVT corners supported

Process	Best case voltage/temp (V/C)	Nominal case voltage/temp (V/C)	Worst case voltage/temp (V/C)
TSMC90G	1.1V / -40 C	1.0V/ 25 C	0.9V / 125 C
TSMC90GT	1.26V / -40 C	1.2V/ 25 C	1.08V 125 C

Table 2 - 90nm nodes and PVT corners supported

Below are figures that show the dimensions of a one lane (X1) configuration. Figure 2 is a 130nm version and the Figure 3 is the 90nm version. The analog blocks, as stated before, are the same size for both 130nm and 90nm. The 130nm and 90nm digital blocks are both 400um wide to be aligned with the analog blocks. The height of the 130nm is 556um while the 90nm is 430um in height. The height difference is because the 90nm digital block is significantly shorter than the 130nm block. The 4:1 aspect ratio of the 90nm digital blocks made the 90nm block more difficult to route DRC clean. The final width is padded on each side by a 12.5um wide end-cap block.

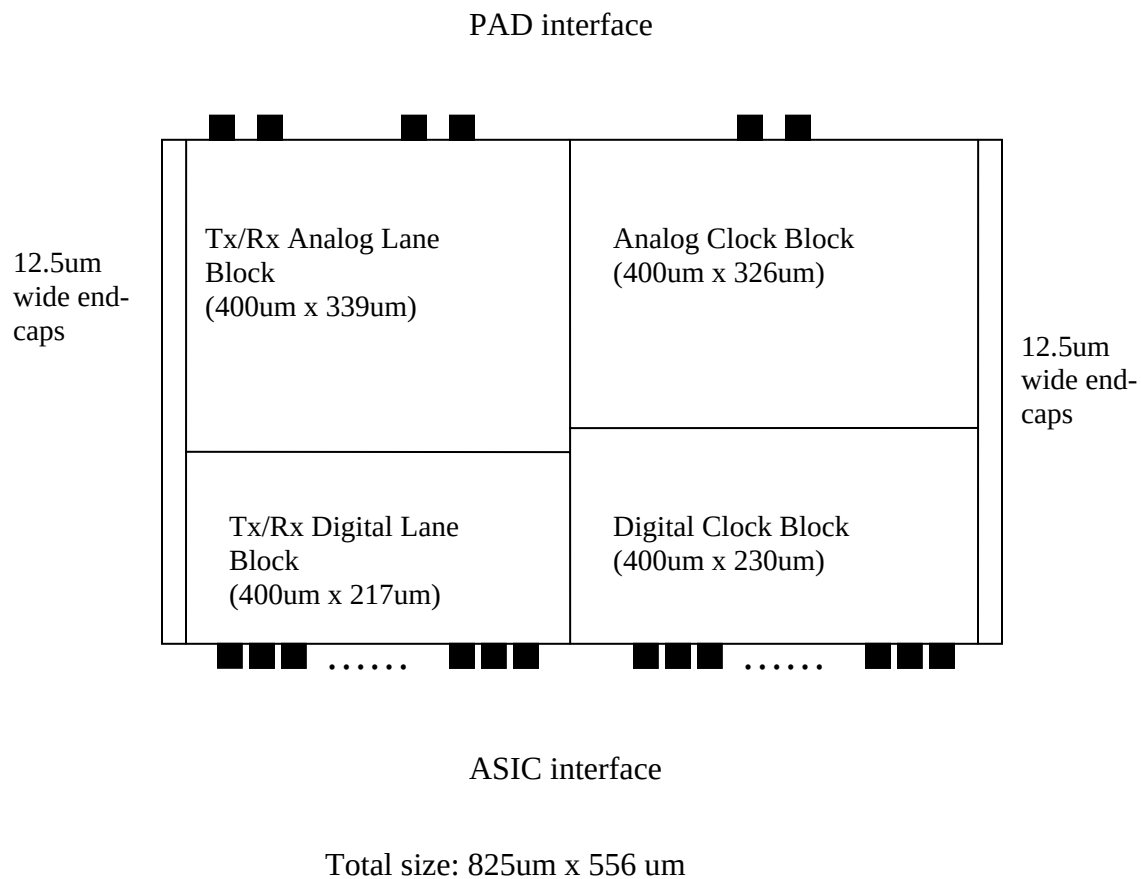


Figure 2 - Physical dimensions of a 130nm X1 lane configuration

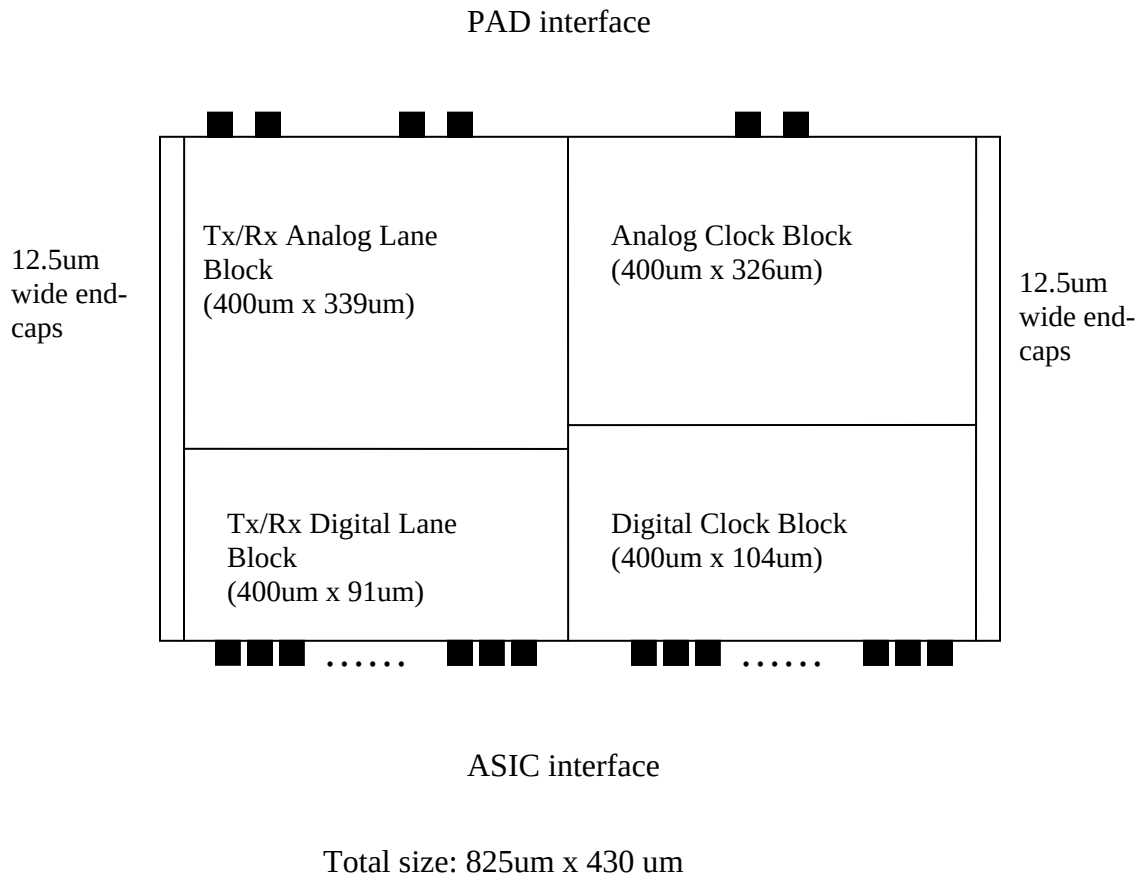


Figure 3 - Physical dimensions of a 90nm X1 lane configuration

From these sub-blocks, the other lane configurations can be created easily. Below, Figure 4 shows all available lane configurations. One challenge was trying to get all configurations to pass all timing runs. All lane configurations have horizontal feed through pre-routes that span across the transceiver (Tx/RX) lane blocks. These feed through routes run on parallel tracks for long distances, and are susceptible to crosstalk. Because of this, care was taken to place and size these preroutes. Also, setup and hold timing was difficult to meet because of the wide difference in lane sizes. Generally, the X1 lane configuration had most of the hold violations. The largest lane configuration, X16, had most of the setup violations as well as some hold violations.

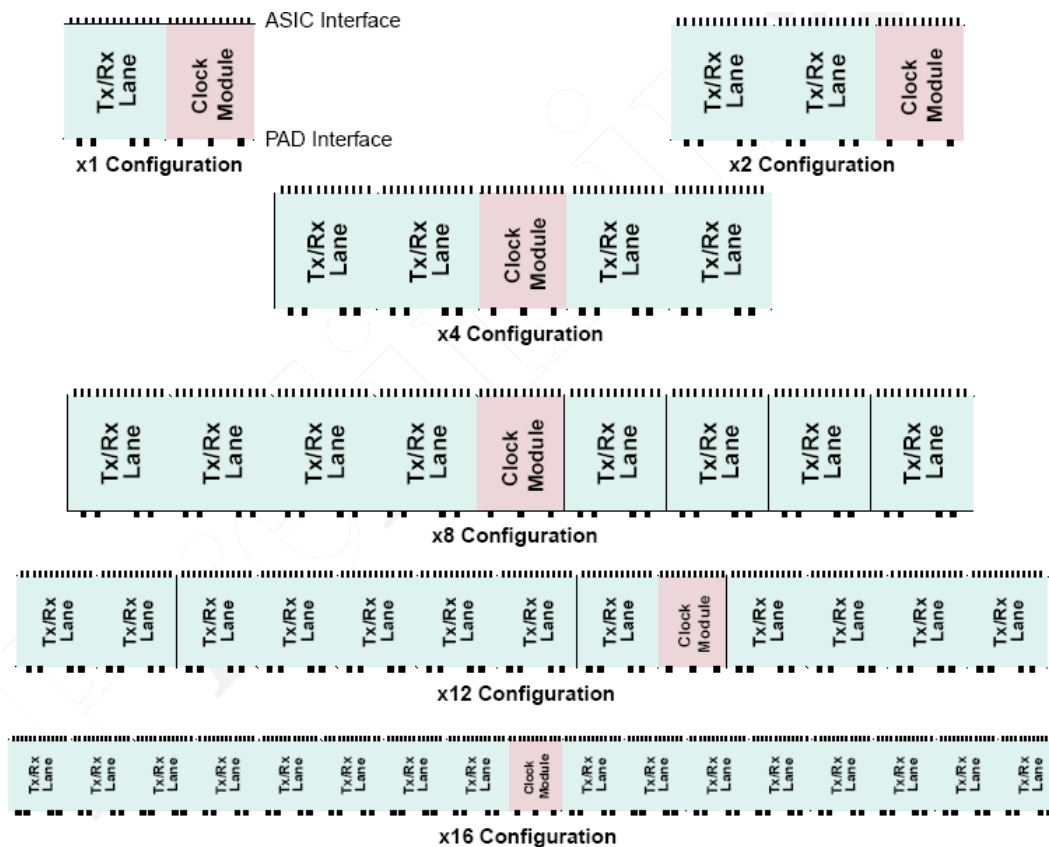


Figure 4 - All available lane configurations

ASIC flow called Tiger

Tiger is an ASIC flow primarily used internally within Synopsys Professional Services. It is a comprehensive flow that can take a design from RTL to GDSII. It incorporates all Synopsys ASIC tools. Tiger includes library preparation, synthesis, test insertion, floor planning, placement, route, DRC/LVS, sign-off static timing analysis, timing model generation, physical view generation, formal verification, power analysis, IR-drop analysis, electro migration analysis, and formal verification flows. The Tiger flow was adopted by the Synopsys Hillsboro Phy group to avoid having to create a flow from scratch.

Figure 5, below, shows a flow chart of the Tiger flow used to create the PHY. For the sub-block flow, all steps from RTL to Final Delivery are executed. For the lane configuration flow, the implementation steps are skipped since at the lane configuration level the physical design is created by abutment only. This is a high level flow chart and hides the many sub-tasks performed in each process. There is no scan insertion, normally a step between synthesis and floorplanning. This is because there are automated test

equipment vectors which test the analog and digital sections of the PHY through the JTAG interface. Following is a description of the major steps.

Library preparation – Vendor library and technology is processed and placed in a directory structure compatible with the Tiger flow.

Constraints – Initial constraints are read in during the synthesis flow. These constraints can be carried to the next flow automatically with the data base. Alternatively, these constraints can be read in to tighten constraints or to filter out specific constraints that may not be applicable to a certain step. For example, setting maximum area and setting high fanout nets as ideal are suitable for synthesis, but would be illegal during a STA step.

Floorplanning – The block boundary, core site array, pin locations, macro placements, placement blockages, and pre-routes are defined at this level. Also, a quick placement, route, and timing are done to assess early impact.

Placement – A good placement engine is crucial for meeting timing and successful routing later. On top of having good placement optimization algorithms, a placer must also be able to do rapid prototype routing (like Steiner routes) accurately. Accurately predicting and avoiding routing congestion at this step is important in being able to do successful final detail route later on, and to close timing.

CTS and Route – High fanout net synthesis, clock tree synthesis, clock route, global route, track assign, detail route, and post-route optimization are the major tasks done at this step. Routing is signal integrity (SI) aware, so SI problems are resolved automatically.

Post route optimization, physical finishing – The last implementation steps are done here. This includes metal fill, filler and decoupling cap insertion, and antenna diode insertion. Final optimizations are done here to make sure that routing is complete and that timing is met.

Extraction, DRC/LVS, STA, power analysis – Final sign off analysis is performed to make sure that the design is ready to be manufactured.

Model creation – LEFs, timing models, and Milkyway physical views (FRAM) are created. GDSII is also streamed out. The data base is ready to be shipped to a customer or foundry.

TIGER ASIC FLOW

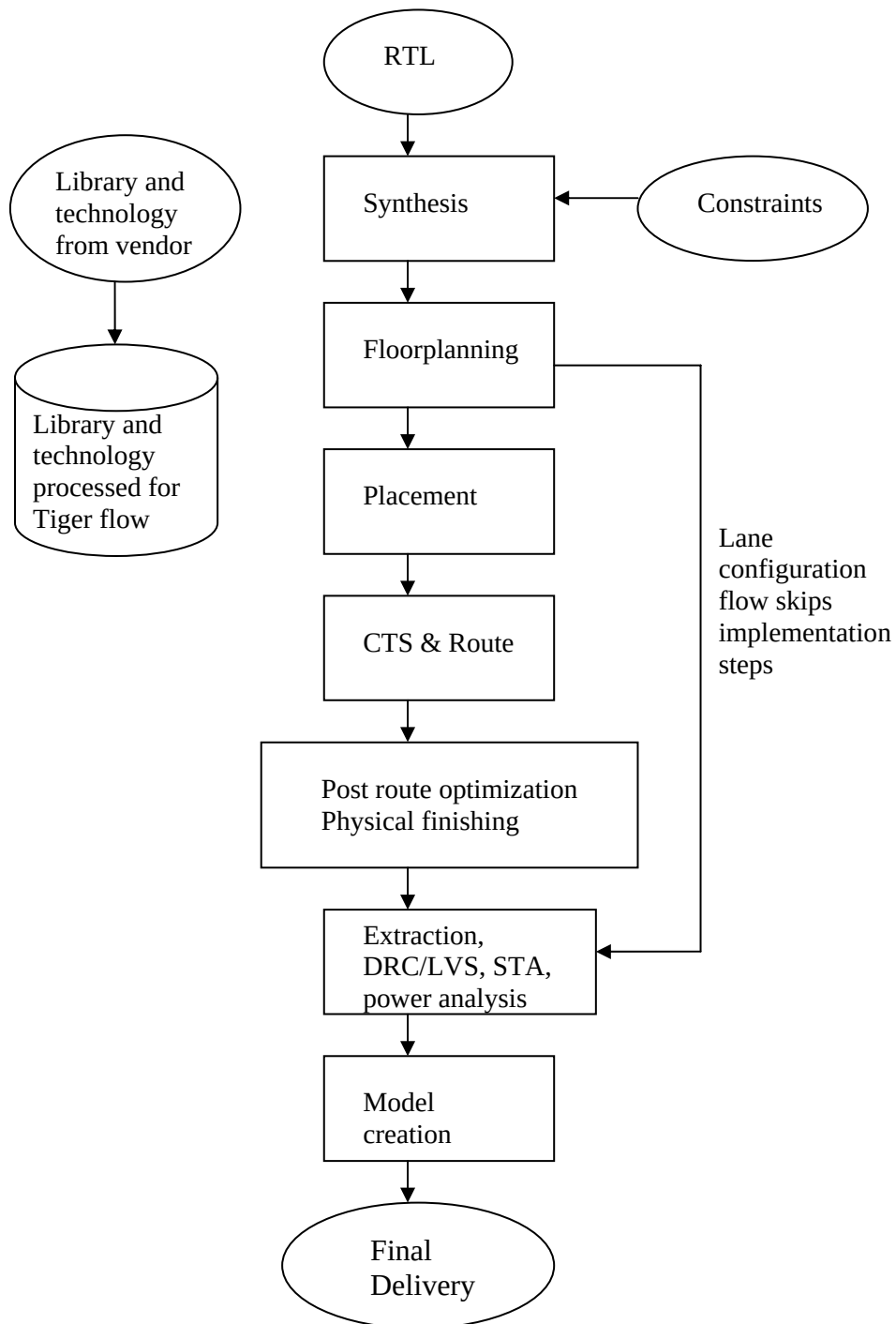


Figure 5 - Tiger flow chart for sub-blocks and lane configurations

Library preparation flow

The library preparation flow mostly copies over library and technology information from a vendor library, to an area that is usable by the Tiger flow. The Tiger flow needs files to be in certain directories or to have specific file names. Besides preparing the vendor library for Tiger, the preparation flow was modified to build “combined” libraries. These libraries were compatible with the G/LV/LVOD processes for 130nm and the G/GT processes for 90nm. Sometimes, files may also need to be processed to be compatible with the Tiger flow or tools. For example, when the Astro library is copied over, antenna information was missing for each cell. A library preparation script was modified to add antenna information to the Astro Milkyway library. Another example is that if timing files are only available in .lib format (ASCII format), then the library preparation flow can compile these into a binary format and attach them as LM views in the Milkyway database. Similarly, if files such as DRC or LVS runsets or Astro technology files or Physical Compiler technology files need to be modified, then this can be done at the library preparation step. The advantage of scripting these changes is that when the library vendor makes available a new library revision, library preparation is an automated step that can be done in a matter of minutes.

From a physical standpoint, it is unknown whether a customer will chose a TSMC node with thick or thin on metal 6. To account for this, the physical design is used with the thick metal 6 rules which are more tightly constrained. For the 130nm node, the pitch/min width/min space is increased from {0.41/0.2/0.2} to {0.86/0.4/0.42}. In addition, fat metal rules for thick metal is used for metal 6, and via enclosure rules assume that the vias are for thick metal. However, for thin metal 6 the via arrays are thin metal sized. If a customer requires thick metal, then during GDSII stream out the thin metal via arrays are swapped with via arrays that are thick metal compliant. LVS and DRC are then re-run on this new GDSII.

To make sure that the timing is closed across all nodes and all PVT corners, the slowest library and slowest parasitic is used for setup timings, and the fastest library and fastest parasitic is used for hold timings in the synthesis, place, and route tools.

For 130nm, the two extreme PVT corners are:

- maximum (slowest) PVT: TSMC130G/ VDD=1.08V/ Temp=125 C
- minimum (fastest) PVT: TSMC130LVOD/ VDD=1.26V/ Temp=0 C

For 130nm, the two extreme parasitic types are:

- maximum (slowest) parasitic: t013s7mg_fsg_cworst.itf
- minimum (fastest) parasitic: t013s7mg_fsg_cbest.itf

During library preparation all maximum/minimum libraries and parasitics are identified for the placement and route tools, Physical Compiler and Astro. For the 130nm node, the absolute maximum combination is library TSMC130G max and parasitic

t013s7mg_fsg_cworst.itf. For the 130nm node, the absolute minimum combination is library TSMC130LVOD min and parasitic t013s7mg_fsg_cbest.itf. By using these absolute worst conditions for the implementation tools, timing is almost guaranteed to be met for all intermediate conditions. This makes the tools work harder and longer, but minimizes the chances that manual ECOs will have to be done later to meet timing.

For final static timing runs, all corners are considered. In this flow, there are a total of 3 transistor process corners: maximum, typical, and minimum. Also, there are a total of 5 parasitic corners: maximum, minimum, and the following three:

- RC maximum: t013s7mg_fsg_rcworst.itf
- RC minimum: t013s7mg_fsg_rcbest.itf
- typical (nominal) parasitic: t013s7mg_fsg_typical.itf

Taken all together, there are 3 transistor corners (TSMC130G, TSMC130LV, TSMC130LVOD), 3 voltage temperature corners (worst, typical, and best case), and 5 parasitic corners (C worst, C best, RC worst, RC best, and typical). The typical voltage temperature corners and typical parasitic corners are not used for STA. The total amount of STA runs works out to:

3 transistors corners X 2 voltage temperature corners X 4 parasitic corners X 6 lane configurations

or 144 total STA runs.

Additionally, STA was run with and without SI, OCV, and IR effects. This added numerous additional runs and reports to review. Running this many different STA corners ensures that there are no outlier timing paths that escape detection. Notice that there are “C” (e.g., t013s7mg_fsg_cworst.itf) based parasitic and “RC” (e.g., t013s7mg_fsg_rcworst.itf) based parasitic files. The “C” is for the worst and best case capacitance while “RC” means the worst and best case R with adjustments to C towards worst or best, but keeping the process planar. Through experience, it has been found that the “C” based extraction provides the worst and best cases over “RC” for internal timing paths. This is because capacitance dominates for short wires. However, for large configurations, inter-block timing paths were often worse with the “RC” worst parasitic files. This is because the R dominates for long wires, which affects the propagation delay of signals from the support block to the furthest transceiver and back.

Constraints strategy

At the lane configuration level, there is one master constraint file for all lane configurations. In addition, the digital transceiver block and digital clock block each have a master constraint file. Having one constraint file with common constraints to handle all cases (e.g. for synthesis and STA) versus managing multiple constraint file reduces the amount of error and maintenance effort. The master constraints define all clocks, clock latencies, clock transitions, clock uncertainties, and clock periods. Also, I/O delays, false

paths, and multi-cycle paths at the port level are defined. The master constraint file for a lane configuration supports all possible lane configurations (X1, X2, X4, X8, X12, and X16) by extracting the lane size embedded in the lane configuration name. Each master constraint file also sources a lower level constraint file that is described next.

A unique feature with this PHY flow is the usage of a script that extracts constraints from RTL comments. Because of PHY design and timing requirements, there are considerable timing exceptions involving internal registers. Embedding the constraints in the RTL makes it easier for a designer to see the connection between the constraint and the RTL that it references. Also, the constraints are more compact since it is based on reference rather than instantiations. Below is a snippet of embedded constraints in RTL (verilog) code. The constraint extraction script keys on the comment “//%%SYNTH:” and processes the constraints after this. The designer doesn’t have to know the instance path, defined by the variable \$inst.

```
// %%SYNTH:
// set_false_path -to [get_pins $inst/hold/lat_bit_*__slice/cn]
// set_false_path -to [get_pins $inst/hold/lat_bit_*__slice/sn]
// set_false_path -through $inst/hold/lat_bit_*__slice/q -to [get_cells $inst/q_reg*]
// set_max_time_borrow 0 [get_cells $inst/hold/lat_bit_*__slice/*]
// set_multicycle_path -setup 2 -to [get_pins $inst/hold/lat_bit_*__slice/d]
// set_multicycle_path -hold 2 -to [get_pins $inst/hold/lat_bit_*__slice/d] -end
//
```

The constraint extraction script generates the Synopsys constraints shown below. These constraints are extracted from the RTL. Notice that variable \$inst is based on the reference name so the designer doesn’t have to worry about hierarchy of the constraint.

```
foreach_in_collection instc [filter_collection [get_cells -hier] -regex
ref_name=~cr_inline_reg.*] {
  set inst [get_object_name $instc]
  set_false_path -to [get_pins $inst/hold/lat_bit_*__slice/cn]
  set_false_path -to [get_pins $inst/hold/lat_bit_*__slice/sn]
  set_false_path -through $inst/hold/lat_bit_*__slice/q -to [get_cells $inst/q_reg*]
  set_max_time_borrow 0 [get_cells $inst/hold/lat_bit_*__slice/*]
  set_multicycle_path -setup 2 -to [get_pins $inst/hold/lat_bit_*__slice/d]
  set_multicycle_path -hold 2 -to [get_pins $inst/hold/lat_bit_*__slice/d] -end
}
```

Next, the extracted constraints are merged and modified to create a multiple lane version of the constraints. Below, Figure 6 shows the timing constraint flow. First, the lower level digital blocks, up3_topd and sup_topd, have their constraints extracted from RTL. When the master constraints of up3_topd and sup_topd are applied to their respective designs, the extracted constraints are also applied. For the multiple lane configurations, a script merges the extracted constraints, and adds the appropriate prefix to the constraint paths to create a new “extracted” constraint version for the multiple lane configuration level.

When the multiple lane configuration (up3_x_top) master constraint is applied, this new “extracted” constraint is also applied.

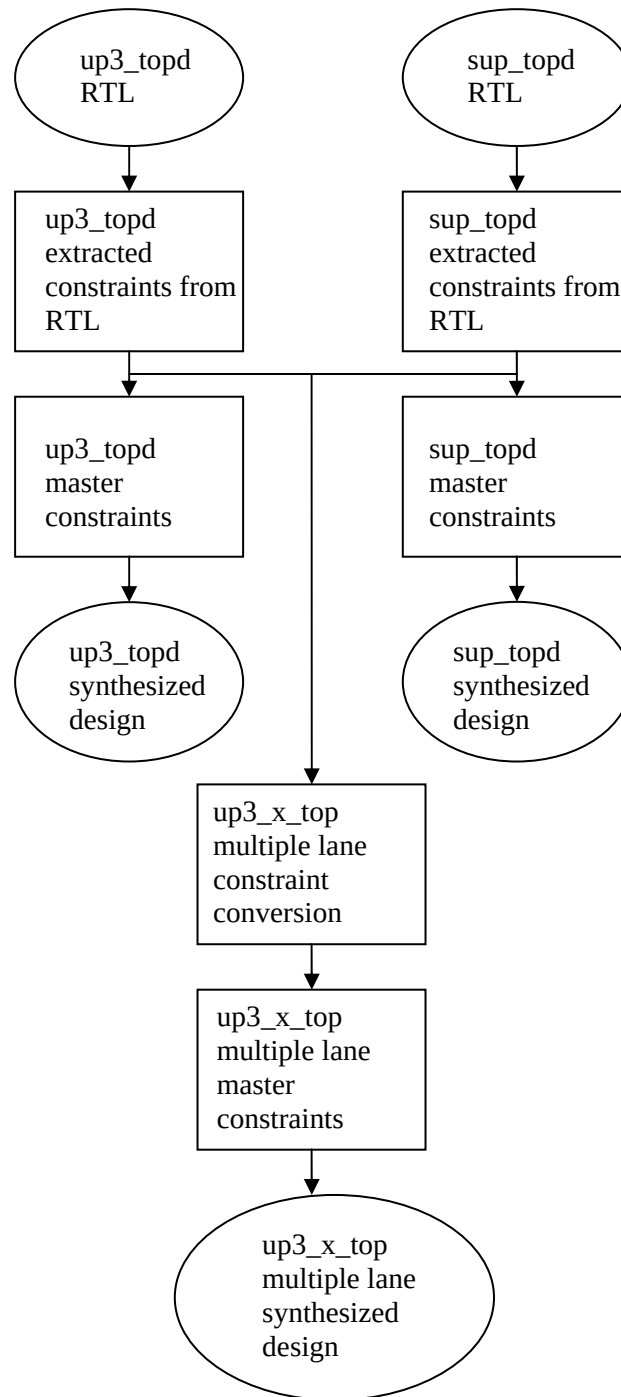


Figure 6 - timing constraint flow

An important aspect of the master constraints is its ability to adjust the content and parameter values of the constraints based on which tool and step the flow is at. One aspect is the handling of implementation constraints such as maximum transition, maximum capacitance, maximum fanout, don't touches, and critical range. Critical range is a constraint that allows Synopsys tools to improve timing paths beyond the worst negative slack of a path group. Some implementation and analysis tools will not understand these constraints. As a solution, when these non-compatible tools are detected a Tiger procedure is used to filter out these unusable constraints, and then a second Tiger procedure reads in these filtered constraints.

Decreasing clock periods during sub-block creation is an important strategy in meeting timing by creating extra margin. For example, during synthesis and placement, a clock period of 2.72ns (367.6MHz) is used for the transmit and receive clocks. During CTS and route, the clock period is increased to 2.8ns (357MHz). Finally, for final sign-off STA, the sign-off periods of 2.88ns (347MHz) is used. In silicon, the real period will be longer depending on the standard used and will range from 3.2ns to 4.0ns (between 312.5MHz and 250MHz).

Another important feature is the adjustment of clock uncertainties. For example, in the synthesis and place and route tools, the transmit and receive clocks have setup uncertainty of 200ps and a hold uncertainty of 100ps. However, in sign-off STA runs, on-chip-variation (OCV) modeling is used. At this point, setup uncertainty is reduced to 50ps to account for jitter and hold uncertainty is reduced to 10ps for extra margin. For the PHY, OCV derates the clock tree by 25% of its delay. However, in most cases the clock skew is almost never 25% of the total clock latency because much of the clock tree is common between timing start and end points.

Physical Design Strategy

Figure 2, above, shows the four sub-blocks: analog transceiver (up3_topa), digital transceiver (up3_topd), analog clock (sup_topa), and digital clock (sup_topd). Each of these four blocks were created separately. Another approach would have been to combine the analog and digital transceivers as one block and the analog and digital clock blocks as another block. Here are the tradeoffs considered when deciding to keep the analog and digital sub-blocks separate:

Pros:

- Analog block development lags digital block development. Adding/removing pins, creating a timing model and a physical view for the analog blocks could take many days more than on the digital side. By keeping the analog and digital separate, there is no need to create and use dummy models until the analog block changes were complete.
- The digital blocks use only standard-cells, so STA and LVS/DRC flows are much less complicated by analog block issues.

- Analog blocks use via-programming feature which would have to be carried through an extra level of physical hierarchy. This would force the creation of a custom flow. By excluding the analog block, a custom flow isn't needed.
- Can tighten I/O constraints along the analog/digital interface for more margin.

Cons:

- Must make estimates of the I/O constraints along the analog/digital interface until an extracted timing model of the analog block is created. In a few cases, estimates of capacitance and transition expectations were not accurate, forcing a new build with updated constraints.
- Even with accurate I/O constraints, such a constraint is never as accurate as a characterized analog block that models timing, drive strength, and parasitics accurately. In one case, where timing was at the edge of passing or failing, timing wasn't fully determined until STA of a lane configuration. Adjustments were then made to the constraints to add the required margin.

In the case of the PHY, since the digital sub-blocks are only about 20K gates in size, the total time for creation, extraction, and timing of a lane configuration was less than four hours. Also, there were only a few critical timing paths between the analog and digital blocks. In the end, the design team felt that keeping the digital and analog blocks separate was the best solution.

For pin placement along the analog and digital interface, a script parses the LEF of an analog block to create a mirror placement of pins for the digital block. If a pin placement on the analog side is planning to change in the future, then the corresponding pin placement change is done manually for the digital side. For pin placement along the ASIC interface and for power pre-routes, a file with pin locations was created manually and is identical for both 130nm and 90nm.

An important part of the physical design are the feed through pre-routes (or nets) that span the transceiver blocks. Figure 4, above, shows the X16 lane configuration. There are 8 transceiver blocks to the left of the clock block and 8 transceiver blocks to the right of the clock block. In this X16 case, the horizontal feed through pre-routes are as long as 8 x 400um or 3200um. Crosstalk can become a serious problem on routes this long. The strategy the design team took was to put 5 of these long pre-routes between VDD and VSS straps as shown in an example below, Figure 7. Here, the tx_term_val[0:4] pre-route signals are placed in a 90nm design on metal 5. In the 90nm node, the minimum spacing is 0.14um and the minimum width is 0.14um. To prevent crosstalk problems, the tx_term_val[0:4] pre-routes are spaced 0.34um apart and the widths are 0.4um to reduce resistance. This is more than double minimum spacing and almost triple minimum width. Also, the spacing of tx_term_val[0:4] is not wide enough that a metal 5 signal could be routed between the pre-routes. An unexpected route between pre-routes may cause a crosstalk problem later on. Clock pre-routes are kept next to a VSS or VDD strap and spaced even further from other pre-routes to further reduce crosstalk problems. Another critical concern is to place driving and receiving gates very close to the pre-routes. Ideally, the driving and receiving gates should be directly underneath the pre-routes. This

is done with a bound command in the placement tool, Physical Compiler. All the pre-routes, like tx_term_val[*], have a receiving gate for each transceiver block. Special consideration was given to these receiving gates. If this gate is, for example, 60um away, that can cause significant transition problems. This adds 60um x 8, or 480um of extra wire which is un-needed cap on this already long wire. If the receiving gate is a large drive strength cell, this adds additional pin cap load at the end of each 60um metal stub. Although the pin cap load of the receiving gate is relatively small compared to the cap of the pre-route, having a large pin cap at the end of a long RC delay amplifies the degradation on transition times. Overall, this can cause the transition times to approximately double from 300ps to 600ps. For these reasons, the design team forced all cells with input pins connected to feedthroughs to be small drive strength and single fan-in per transceiver block. All driving cells of feedthrough nets were forced to be a large drive strength with double width and double via constraints in the route tool.

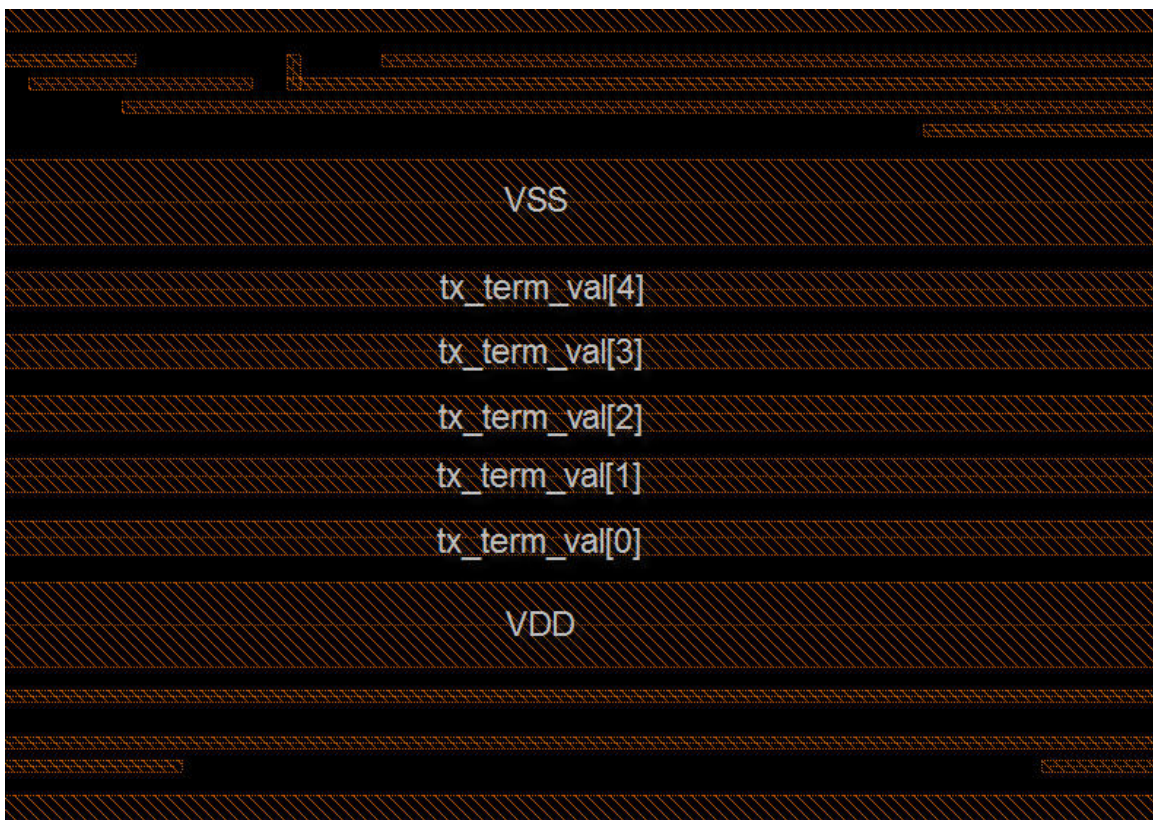


Figure 7 - Feed through pre-routes in 90nm transceiver design

To re-use transceivers independent of mirroring and to reduce the number of horizontal feedthroughs, a via-programming feature is utilized in the floorplanning tool. Figure 8 and 9 show two metal 5 pre-routes bs_io1 and bs_io2. In some cases, bs_io1 is an input pin, goes through some logic, and drives results onto bs_io2. However, in other cases, bs_io2 is an input signal, goes through some logic, and drives results onto bs_io1. The directions of bs_io1 and bs_io2 are dependent on physical characteristics such as whether the transceiver is to the left or right of the clock block, and whether the transceiver is mirrored or not. To facilitate the change in direction of the pins, two other “horse-shoe”

shaped pins on metal 4, bs_in and bs_out are created. Bs_in is always connected to the input logic and bs_out is always connected to the output logic. It's a matter of connecting bs_in to bs_io1 and bs_out to bs_io2, or connecting bs_in to bs_io2 and bs_out to bs_io_1. Notice, because of the “horse-shoe” shape of the pins, bs_in can be connected to either bs_io1 or bs_io2 with a via and bs_out can be connected to either bs_io1 or bs_io2 through a via. In the past, the design team had created a script that would calculate the coordinates of where wires and vias are dropped based on physical knowledge of all lane configurations. However, this scripting was cumbersome and required changes if the pre-routes and “horse-shoe” pins moved. There is a superior method. First, the “horse-shoe” pins are made to be actual pins at the transceiver level. Next, a script automatically generates via programming for the lane configurations. This script has knowledge of the net names to program, and connects the pins of bs_in and bs_out to the correct pins of bs_io1 and bs_io2 during the floorplanning step with the JupiterXT command `axgCreatePrerouteContacts`, to automatically instantiate all vias. No knowledge of the pin locations to add vias to is needed by the script. JupiterXT calculates the via locations itself. In another view, the via programming problem is reduced to relatively easy RTL changes. This same via programming technique is used to program the sub-blocks to operate with a particular protocol.

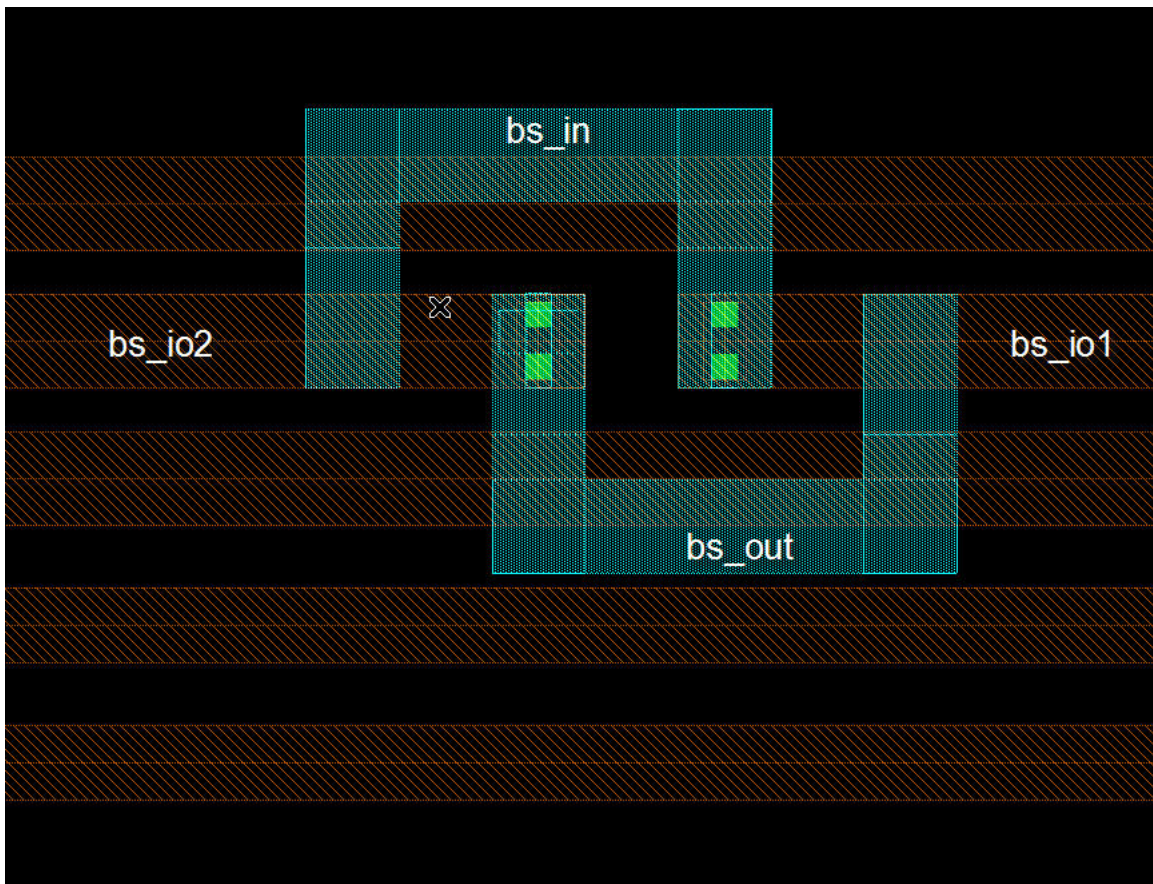


Figure 8 - Via programming example of bs_io1 connected to bs_in and bs_io2 connected to bs_out

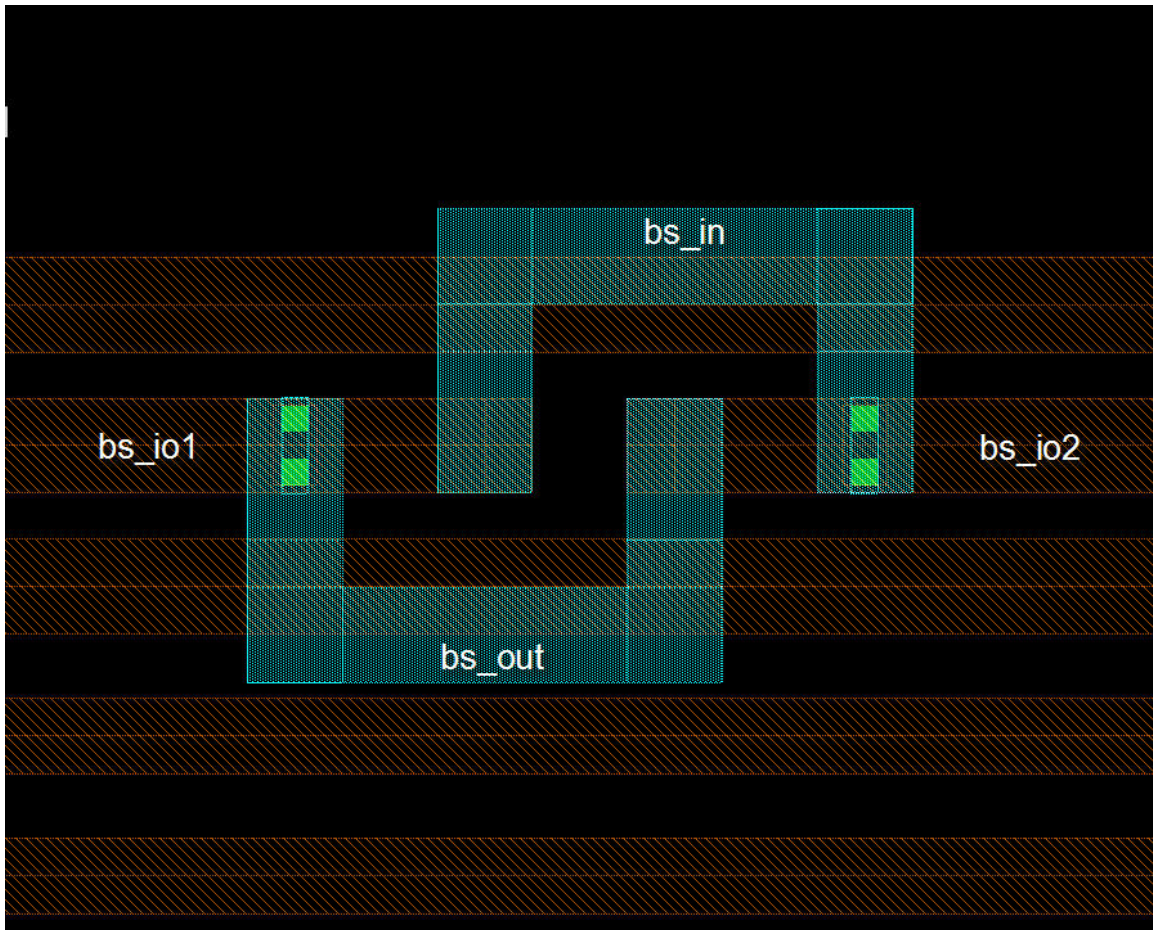


Figure 9 - Via programming example of bs_io2 connected to bs_in and bs_io1 connected to bs_out

Conclusions

Throughout this paper, there were examples of re-using designs and flows. One was having a design that can support multiple protocols. Another was having analog sub-blocks that were common across 130nm and 90nm technology nodes. This paper described the challenges faced on the back-end side, creating the digital sub-blocks, and integrating all the blocks together. Having sub-blocks allowed the easy construction of an array of top level lane configurations through abutment and via programming. Having a single RTL source allowed the 130nm and 90nm designs to have very similar flows and timing constraints. Using Tiger, a proven flow, saved time from having to create a new flow from scratch. The flow was setup to handle the slowest and fastest corners of different nodes making timing closure free of ECOs to meet timing. In the end, a single physical design met timing requirements across a wide range of technology nodes and PVT corners. Having a good constraint strategy was crucial. Embedding constraints in the RTL, having one set of constraints for all tools, and having a script to deal with multiple lane configuration reduced the amount of work maintaining constraints and

reducing error. Although it was extra effort invested at the start of the design, overall time, effort, and mistakes were greatly reduced.