

# **Breaking the Gigahertz Speed Barrier with an Automated Flow Using Commercial Standard Cell Libraries and Memories**

Soumya Banerjee  
Avishek Panigrahi

MIPS Technologies, Inc.

sban@mips.com  
avishek@mips.com

Dan Lefrancois  
Sharrone Smith

Synopsys, Inc.

daniel.lefrancois@synopsys.com  
sharrone.smith@synopsys.com

## **ABSTRACT**

Traditionally, developing a high performance embedded processor required a custom design methodology, hand-crafted libraries and memories, and a team of specialized layout and circuit designers dedicated to the design and implementation of the processor. MIPS Technologies and Synopsys have worked together to develop an automated design methodology based on IC Compiler for the next-generation MIPS32<sup>®</sup> 74K<sup>™</sup> core family – enabling SoC designers to achieve near-custom results exceeding 1GHz, using off-the-shelf 65nm process, libraries and memories. This paper highlights how the collaborative efforts of both companies resulted in an automated RTL-to-GDSII flow.

## Table of Contents

1.0	Introduction.....	4
2.0	Key ingredients for achieving 1+ GHz.....	4
2.1	Overview of the MIPS32 <sup>®</sup> 74K <sup>™</sup> core architecture.....	4
2.2	Design considerations for high-speed processor IP.....	6
2.2.1	Design constraints.....	6
2.2.2	Clock gating.....	6
2.2.3	Setup and access paths to and from the memories.....	7
2.2.4	Optimized implementation scripts.....	7
2.3	High-performance off-the-shelf physical IP and process.....	7
3.0	High-performance tools and methodology.....	8
3.1	Timing challenges in the MIPS32 <sup>®</sup> 74K <sup>™</sup> core.....	8
3.1.1	Challenge #1: Clock enable timing.....	8
3.1.2	Challenge #2: Memory interface timing paths.....	9
3.2	Generating an optimal netlist using DC Topographical (DC-T) technology.....	10
3.2.1	Overview of the DC-T flow.....	10
3.2.2	DC-T to ICC flow description.....	11
3.2.3	XG mode and testability.....	11
3.2.4	Review of DC-T results.....	11
3.3	Generating an optimal floorplan using ICC design planning.....	12
3.3.1	Overview of the ICC design planning - floorplan exploration flow.....	12
3.3.2	ICC design planning floorplan results.....	16
3.3.3	Description of custom power grid.....	18
3.3.4	Power network analysis (PNA).....	21
3.4	Overview of ICC flow.....	22
3.4.1	Overview of the ICC Recommended Methodology (ICC RM) flow.....	22
3.4.2	<i>place_opt</i> optimization strategies.....	24
3.4.3	<i>clock_opt</i> and useful skew optimization strategies.....	24
3.4.4	<i>route_opt</i> / signal integrity optimization techniques.....	28
3.4.5	Correlation between ICC, Star-RCXT and PrimeTime SI.....	28
3.5	Review of ICC results.....	32
4.0	Conclusions and recommendations.....	32
4.1	Benefits of the methodology.....	32
4.2	Areas for future investigation and improvement.....	32
4.3	Reference flow for the 74K <sup>™</sup> core and other MIPS <sup>®</sup> cores.....	33
5.0	Acknowledgements.....	33
6.0	References.....	33
7.0	Appendix.....	33

## Table of Figures

Figure 1 – Pipeline architecture of the MIPS32® 74K™ processor core .....	5
Figure 2 - Clock gating enable timing paths.....	9
Figure 3 - Memory interface timing paths .....	10
Figure 4 - Design Compiler results summary .....	11
Figure 5 - Explore mode flow in ICC .....	13
Figure 6 - Explore mode results in ICC .....	14
Figure 7 - <i>explore_mode.tcl</i> script .....	15
Figure 8 - Floorplan strategy - initial (left) and revised (right) .....	16
Figure 9 - <i>gen_tdf.tcl</i> .....	17
Figure 10 - Generated TDF file .....	17
Figure 11 – Custom power grid for the 74K™ core. ....	18
Figure 12 - PG pin connections .....	19
Figure 13 - Only M6 to M1 stacked vias allowed via <i>set_preroute_drc_strategy</i> .....	19
Figure 14 – Layer-to-layer power connections set via <i>set_preroute_drc_strategy</i> .....	20
Figure 15 - ICC PNA IR drop map.....	21
Figure 16 - The 74K™ core ICC flow .....	23
Figure 17 - <i>write_port_latencies.tcl</i> script.....	26
Figure 18 - <i>adjust_latency.tcl</i> script .....	27
Figure 19 - Total capacitance correlation (values in fF).....	29
Figure 20 – Coupling capacitance correlation (cap values in fF).....	30
Figure 21- Example of PT SI script for accuracy (pessimism is removed).....	31
Figure 22 - ICC final results .....	32

## 1.0 Introduction

This paper highlights the key “ingredients”, investigations and results from a joint MIPS Technologies/Synopsys project to deliver a high-performance design methodology to end users. The goal of the project was simple – develop a methodology for the MIPS32<sup>®</sup> 74K<sup>™</sup> synthesizable core that would enable it to achieve processing speeds of 1 GHz or greater, using off-the-shelf 65nm process, standard cells and memories. In this process, the companies identified a number of tradeoffs in the design methodology and refined it to produce reliable, high-quality results.

## 2.0 Key ingredients for achieving 1+ GHz

- An automation-friendly processor core, the MIPS32 74K core, with clean constraints
- Commercially available 65nm standard cells and memories
- An optimized implementation methodology using Galaxy<sup>™</sup> Design Platform, which includes Design Compiler<sup>®</sup> Ultra (DC Ultra<sup>™</sup>) and IC Compiler (ICC) from Synopsys

The first two elements were fixed per our project goals and general industry requirements. The objective was to deliver the third key element, a straightforward methodology combining high performance with efficient power consumption. The remainder of this section provides additional background on the first two elements, and the following sections outline the investigations and decisions used to create and tune the methodology.

### 2.1 Overview of the MIPS32<sup>®</sup> 74K<sup>™</sup> core architecture

The MIPS32 74K core family features a superscalar, out-of-order pipeline architecture, designed to maximize the performance achievable by synthesizable methodology. The 74K core family implements the MIPS32 Release 2 instruction set as well as the MIPS16e<sup>™</sup> Application Specific Extension (ASE) for code compression and the MIPS<sup>®</sup> DSP ASE for signal processing performance. The CorExtend<sup>®</sup> capability of the 74K core also provides support for the addition of user-defined instructions. The core supports standard interfaces including the OCP2.1 system interface and EJTAG3.2 interface.

The 74K core family was designed to achieve a significant performance boost over MIPS Technologies’ existing line of single-threaded processors. The performance boost is achieved through a combination of frequency and micro-architectural enhancements, including the ability to dispatch two instructions per cycle as well as the ability to issue instructions out of order.

The dual issue capability is achieved through two pipelines referred to as the AGEN and ALU pipelines. The AGEN pipeline is 17 stages long and implements all Memory Transfer (Load/Store) and Control Transfer (Branch/Jump) instructions. The ALU pipeline is 16 stages long and implements all the other instructions from the MIPS32 Release 2 instruction set. Instructions can be dispatched and completed in these pipelines out-of-order.

The deeply pipelined 17-stage pipe was instrumental in achieving the frequency boost while maintaining a fully synthesizable approach. Extra pipeline stages were added to some of the key

frequency bottleneck areas. For example, the data cache and instruction cache accesses span 3 stages each in order to enable the use of generator-based SRAMs and synthesizable technology. Similarly the Arithmetic Logic Unit (ALU) is pipelined and spans two pipeline stages as compared to one stage in previous generations of synthesizable cores from MIPS Technologies.

Figure 1 shows the 74K core pipeline in further detail. The pipeline itself operates as several mini-pipelines that communicate via buffers. The ALU and AGEN execution pipelines are fed by a common 8-stage front end which is responsible for Instruction Fetch, Decode and Dispatch. The ALU and AGEN execution pipelines are also supported by a common 2-stage backend pipeline which is responsible for instruction graduation. As instructions complete out-of-order in the execution pipelines, their results are maintained in temporary storage referred to as completion buffers. The 2-stage graduation pipe is responsible for committing these results in order to the architectural state. In addition there is a separate Multiply Divide Pipeline which is an offshoot of the ALU pipeline as well as a separate Floating Point Pipeline which is totally decoupled from the integer execution pipelines. The Floating Point Pipeline is also capable of executing 2 instructions per cycle.

Control signals such as stalls are critical signals on any processor. Splitting the pipelines into multiple mini-pipelines helps alleviate the problem associated with the propagation of global stall signals. Additionally, the AGEN and ALU execution pipelines are stall free in order to reduce the need for stall signal propagation.

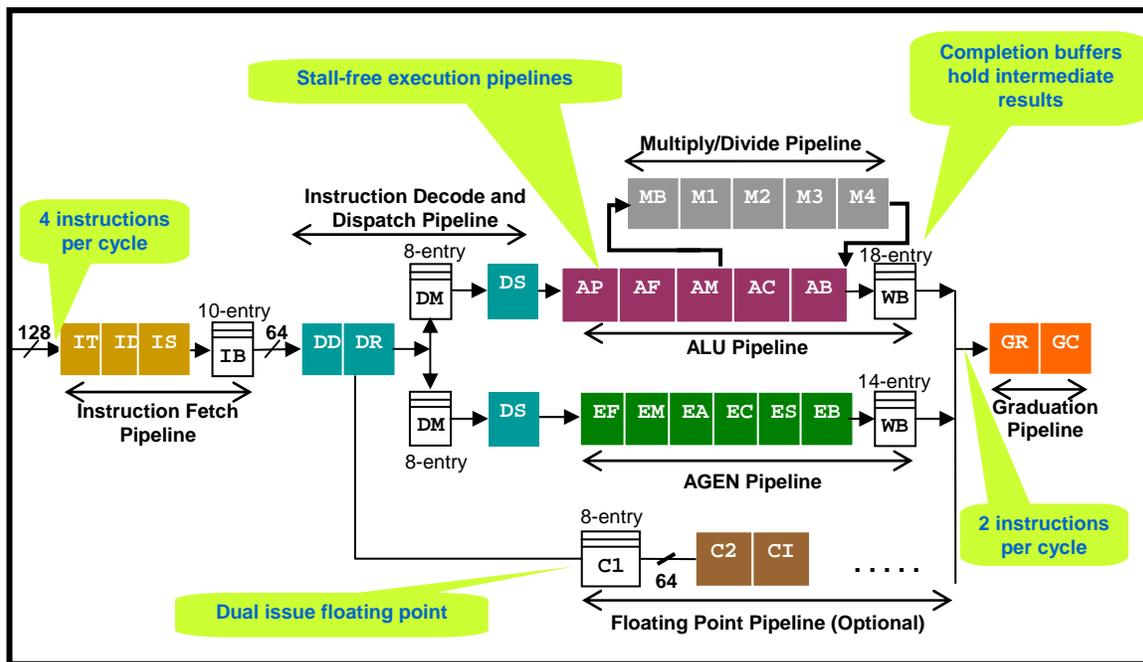


Figure 1 – Pipeline architecture of the MIPS32<sup>®</sup> 74K<sup>™</sup> processor core

## 2.2 Design considerations for high-speed processor IP

A synthesizable IP core dramatically shortens the design cycle, reduces system costs and accelerates time-to-market for SoC designers. Additionally, synthesizable IP can be targeted to the process/fab combination of the customer's choice without having to deal with the long physical implementation process associated with custom or semi-custom processor cores. Synthesizable IP cores also provide the additional advantage of supporting various configuration and build options easily. For example, these options could include different sizes of caches, inclusion or exclusion of memory BIST, and implementation choices like use of clock gating and use of memories for register files. These options are essential to optimize the processor for the distinct requirements of different embedded markets and SoC methodologies.

The IP core must, however, be designed to ensure high-quality, while at the same time providing a consistent and easy implementation by the customer. The following sections describe some of the steps taken in designing the 74K processor core for this purpose.

### 2.2.1 Design constraints

Since the licensee of an IP core is not intimately familiar with the design, the core must be easily constrainable, using only a small number of constraints.

- In the 74K core, all IOs are fully registered, except for a minimal number of IOs that cannot necessarily be registered due to requirements of the OCP bus protocol. Fully registered IOs make it much easier to write constraints that are portable from one technology to another. Additionally, registered IOs make it easy for the core to be hardened independent of the context in which it is being deployed on the SoC and thus easier to reuse.
- The 74K core design uses only 2 mutually asynchronous clock domains: the core clock and the EJTAG clock.
- There are no false paths or multi-cycle paths in the design other than paths that cross over from one clock domain to the other. These domain-crossing paths are false by design, as the two clock domains are meant to be asynchronous to one another.

### 2.2.2 Clock gating

The MIPS32 74K core uses extensive clock gating for minimizing power consumption. Multiple granularities or levels of clock gating exist in the design as described below:

- **Core-level clock gating** – The purpose of clock gating at this level is to shut down the clocks to most of the core. This mode is invoked by an architectural sleep state for the processor. This clock gater resides close to the root of the clock tree, and the flop generating the enable condition for this gater is usually at the leaf-level. The topological location of this clock gater within the overall clock tree creates an inherently challenging family of timing paths in the cone of logic building up to the enable pin of this gater. To ease implementation of this path, the 74K core design ensures that there is no combinational logic on this path – the entire cone of logic consists of one flop.

- **Block-level gating** – The purpose of clock gating at this level is to shut down various blocks within the core depending on activity in the core. For example, if the instruction flowing down the pipe does not use the floating point unit, that unit is shut down. The design issues surrounding block-level clock gating are similar to core-level clock gating and handled similarly within the design.
- **Leaf-level gating** – More than 90% of the flops in the design are locally gated. Because paths ending at these gaters' enables can have critical timing, and because the ideal clock assumption systematically under-represents the constraint on such paths (by the amount of delay through the gater), the flow scripts include compensatory over-constraint for gater-enable paths. The compensatory over-constraint is only in place for those parts of the flow which assume ideal clocks.

### 2.2.3 Setup and access paths to and from the memories

Most microprocessor designs have the setup and access paths to and from the memory at critical or near-critical timing. The 74K core design alleviated these paths as much as possible through these means:

- The micro-architecture eliminates short timing path recurrences in the immediate neighborhood of large memories. In other words, there is no logic path that goes directly from the output back to the input of such memory, nor a pair of paths that do the same thing indirectly through one flop, nor a trio that do so through two flops. This style of the design buys the freedom to provide the memory, and the flops in the following cycle, with early or late clocks without impacting the design or implementation constraints of the rest of the core.
- The logic paths in the flop-flop cycle immediately following RAM accesses are designed to be shorter than other flop-flop paths. This is the source of slack that can be transferred to the memory access paths via clock skewing by the flow scripts.

### 2.2.4 Optimized implementation scripts

Ease of implementation is one of the prime challenges in using third-party IP. The 74K core is packaged together with all synthesis, place-and-route, static timing analysis and verification scripts necessary to take the core from RTL all the way to GDSII. These scripts are set up so that an end-user has to make minimal changes such that the design can be easily ported to any process node with the user's choice of libraries and memories to obtain the best results.

## 2.3 High-performance off-the-shelf physical IP and process

When designing the high-performance 74K processor core, MIPS chose a process and library combination that would be attractive to customers and reflective of current design trends. The process node choice was 65nm, and the TSMC 65GP process was selected for this evaluation. In addition, the decision was made upfront to use standard off-the-shelf libraries from TSMC and memories from Dolphin Technologies. Other premium high-speed libraries and memories could be used to improve the performance of the 74K core but would increase the cost of development for licensees and limit the use of the core.

### 3.0 High-performance tools and methodology

The third key element in achieving frequencies beyond one gigahertz is an optimized implementation methodology. Earlier MIPS reference flows were based on Synopsys Design Compiler (DC), Physical Compiler (PC) and Astro tools. The objective of this evaluation was to transition to a Design Compiler topographical technology (DC-T) and IC Compiler (ICC) based methodology for maximum performance.

#### 3.1 Timing challenges in the MIPS32<sup>®</sup> 74K<sup>™</sup> core

The MIPS/Synopsys methodology for the 74K core was developed with knowledge of and strategies for managing the most pressing timing challenges of the core.

##### 3.1.1 Challenge #1: Clock enable timing

The 74K core has several levels of clock gating. The first level of clock gating is at the root of the clock tree which can shutdown the whole core and is referred to as core-level clock gating. The next level of clock gating is at the block-level which can selectively disable various blocks within the 74K core and is referred to as block-level clock gating. The final level of clock gating is at the level of individual registers and is referred to as the leaf-level clock gating. Figure 2, on the next page, describes the various levels of clock gating in the 74K core.

During early stages of synthesis (pre-clock tree), the tools assume that the clocks arrive at the gaters at the same time as the flops. However, the clock tree synthesis tool traces through these gaters and tries to match the insertion delay across all flops to achieve minimum skew. This by definition makes the clock to the gaters earlier than the flops. As a result, the paths that end at the enable pins of the Integrated Clock Gating cells (ICGs) have an inherent adverse skew after Clock Tree Synthesis (CTS).

However, during synthesis with ideal clocks, these paths do not see the effects of this adverse skew and maybe left suboptimal by the synthesis tool. To avoid this problem, the *set\_clock\_latency* command in DC was used to model an earlier clock to the gaters. This problem is particularly worse with respect to the core-and block-level gaters where the adverse skew is higher in magnitude. To alleviate this problem, the condition signal driving the enable pin of the core and block level gaters is registered to minimize logic on the paths ending at these enable pins.

Special techniques were used in ICC to optimize paths that end at the enable pins of these various levels of clock gating cells used in the 74K core and are discussed later in section 3.4.3.

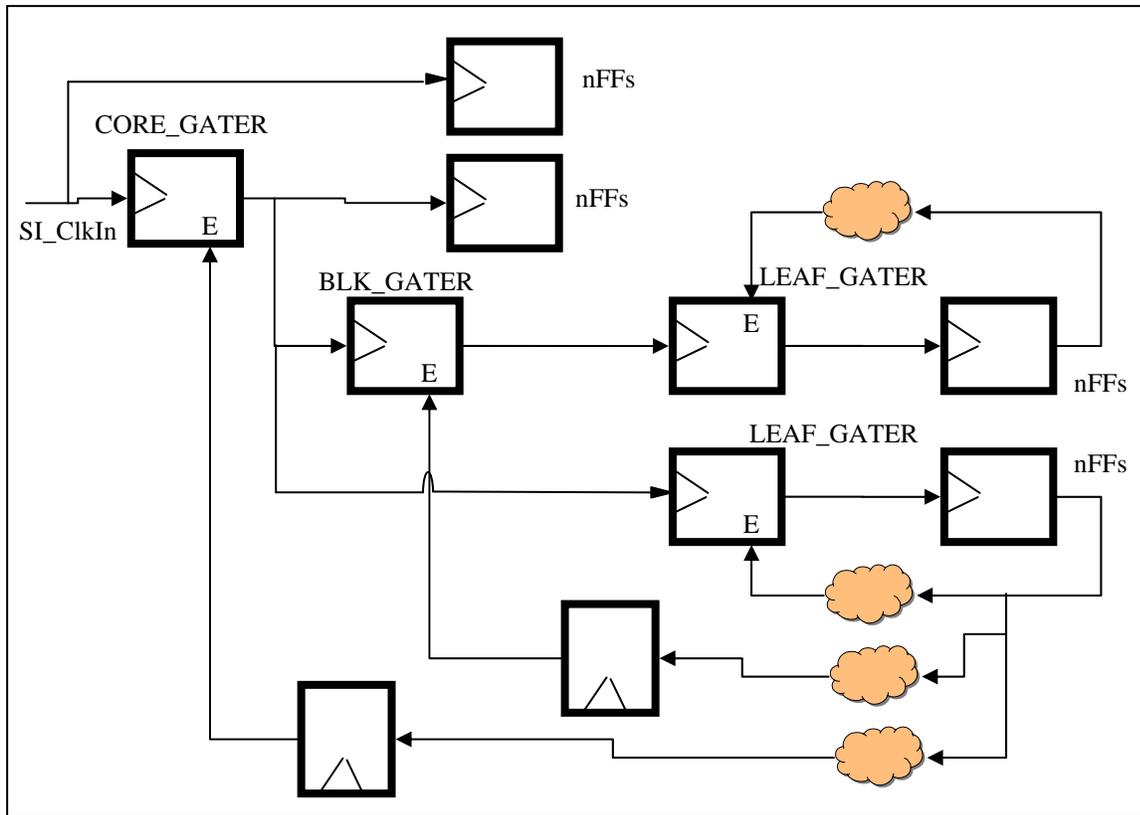


Figure 2 - Clock gating enable timing paths

### 3.1.2 Challenge #2: Memory interface timing paths

Another timing challenge for the 74K core are paths that setup to the SRAMs used for caches (FF\_A to RAM in Figure 3, on the next page) and paths that start with accessing the SRAMs (RAM to FF\_B). However, the micro-architectural implementation of the 74K processor core has kept the paths that start from FF\_B and end in FF\_C (the next stage of the pipeline) intentionally shorter than the rest of the pipeline. While the setup paths can be eased by delaying the clocks to the SRAMs, it is important to be able to skew the capture flops (FF\_B) so that the SRAM access paths do not limit the frequency of the design.

In Figure 3, for example, the clocks to the SRAMs as well as the capture flops (FF\_B) have been delayed by an additional 300ps as compared to the average insertion delay to other flops in the design.

Traditionally in Astro, this was done using *ataDefineSyncPin* which was a manual approach to useful skew. This was usually accomplished in Astro by first running Astro CTS and determining how much delay was needed and then rerunning CTS with *ataDefineSyncPin* to define the required latency to the clock pin of each of the SRAMs and the capture flops (FF\_B).

In ICC, this flow is automated by skew optimization (*skew\_opt*) which is discussed in section 3.4.3 of this paper. Automated useful skew allows the tool to have better estimation of latencies for each of the SRAMs and flops depending on their placement and clock insertion delay and thus usually ends up with a more optimal design as compared to the manual latency specification.

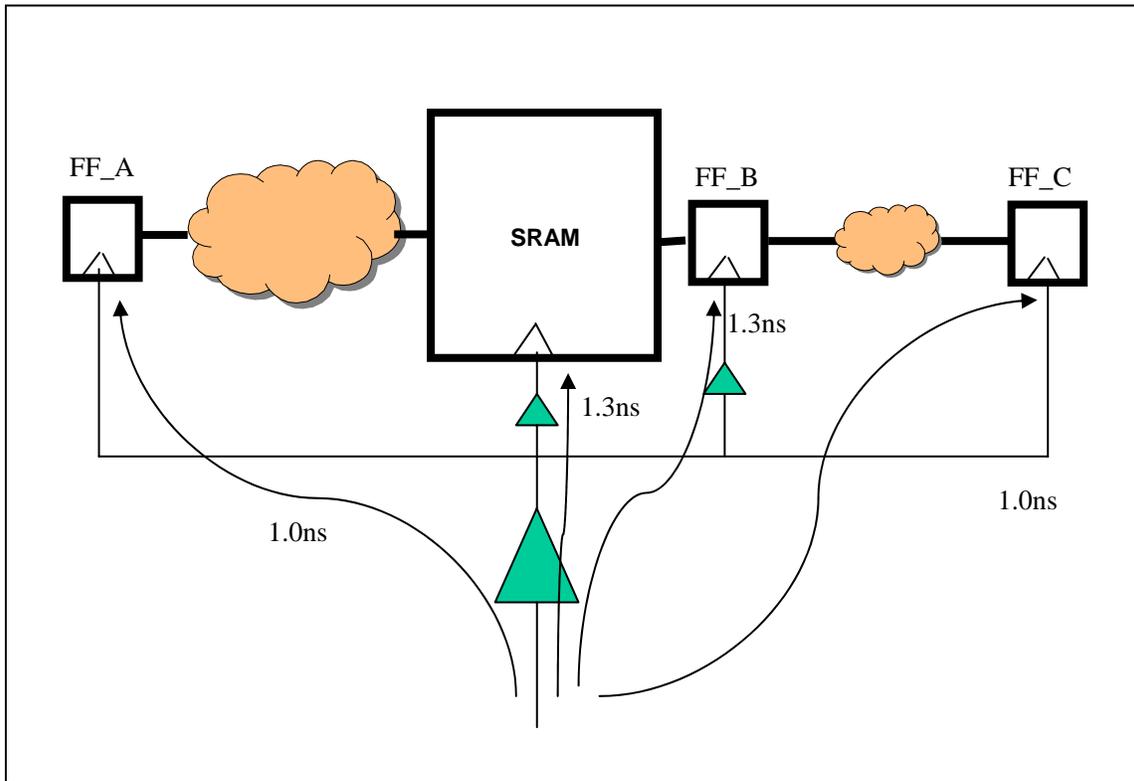


Figure 3 - Memory interface timing paths

## 3.2 Generating an optimal netlist using DC Topographical (DC-T) technology

### 3.2.1 Overview of the DC-T flow

Earlier MIPS reference flows used optimistic wireload models from library vendors for synthesis in a top-down *compile\_ultra* flow. Additionally, the designs were over-constrained and were using predicted clock latencies to RAM and clock gater logic. DC-T technology provides the ability to eliminate usage of wireload model from synthesis and target better timing and area correlation enabling better prediction of place-and-route results for designs at 65nm nodes or below. This is achieved through multiple stages of refined net modeling throughout the synthesis flow during *compile\_ultra* using ICC based placement technology along with physical libraries and constraints. DC-T technology within DC Ultra requires use of the latest XG mode Tcl shell and has its own mode entered using the `-topographic` option to *dc\_shell-xg-t*.

### 3.2.2 DC-T to ICC flow description

The DC-T setup requires the addition of Synopsys-compatible Milkyway™ physical libraries including TLUplus extraction RC data used for pre-route designs. Current versions of DC-T technology have the ability to use actual floorplan inputs when doing the refined placement, and this was necessary in the 74K core design to achieve reasonable results given the number of fixed macro cells and the significant area of floorplan they require. In this flow a DEF floorplan was generated using ICC and DC-T extracted the floorplan information from the DEF using the *extract\_physical\_constraints* command and saved this in Tcl format for later use with the *-output* option. Rather than allow the pins to be randomly placed, a generic pin constraint was applied using “*set\_port\_side [get\_ports \*] -side top*” to force the pins to the top side of the floorplan as desired. *report\_physical\_constraint* can be used to see that the floorplan constraints are accepted and available.

### 3.2.3 XG mode and testability

Switching to XG mode for DC was relatively easy. Generally, wherever db design binary commands were used, the ddc commands were used to replace them, while continuing to use the same db timing libraries. However, some hierarchy manipulation procedures that used the *current\_design* command had to be upgraded to operate in a top-down manner similar to PrimeTime® (PT) so that constraints related to latencies associated with the clocks to memories and clock gaters were not lost.

Additionally, since DC-XG mode DFT commanding has been upgraded, the command syntax itself had changed and required some updating. The latest ATPG-based test design rule checking flow using *dft\_drc* was also used inside DC. Finally, ICC uses a SCANDEF file for scan optimization and expects a stitched netlist from DC, so in addition to compiling a scan-ready design, the scan chain was stitched inside DC using *insert\_dft*. A follow-up DC-T incremental *compile\_ultra* is explicitly done since the *insert\_dft* quick incremental compile is not physically aware and turned off in DC-T. The *write\_scandef* command (xg only) was used to output the scan information for ICC. Final outputs included physically optimized Verilog netlist, SCANDEF, ddc binary design, Milkyway design library and reports.

### 3.2.4 Review of DC-T results

DC-T results correlate better than DC to the post-placement results. This can be useful in providing a shorter feedback loop on timing paths during RTL or floorplan iterations. The table below compares area and timing for the 74K core including results from DC/DC-T and the corresponding results post-placement from ICC.

Clk = 0.84ns	DC clk WNS ns	Place_opt clk WNS ns	WNS Corr	DC Area K	Place_opt Area K	Area Corr
DCxg->ICC	0	-0.16ns	19%	1,699	2,016	16%
DC-T-> ICC	-0.1	-0.15ns	6%	1,796	2,043	12%

Figure 4 - Design Compiler results summary

As expected, from the initial observation of DC reports, timing appears significantly slower with much larger area, but inspection of results after placement indicates that DC-T shows more predictable results with improved correlation in both area and timing.

### **3.3 Generating an optimal floorplan using ICC design planning**

#### **3.3.1 Overview of the ICC design planning - floorplan exploration flow**

Determining the optimal floorplan for a design is a time-consuming manual effort involving several iterations through the backend tools and then analyzing the design for timing and congestion issues. Macro and pin placement along with choosing an appropriate aspect ratio for the design can be quite challenging. Synopsys ICC has built-in design capabilities that help automate this process and make it easier to come up with a good baseline floorplan.

The 74K core as implemented in this evaluation has 18 hard macros and the biggest challenge in floorplanning the 74K core is in finding an optimal location for these hard macros. The design exploration capabilities inside ICC design planning were used in this evaluation to generate several floorplans for the 74K core. Parameters that affect the quality of macro placement were the main variables involved in generating these floorplans. ICC automatically places the macros as well as the standard cells and allows the direct measurement of QoR dependency on the hard macro placement.

The “ICC design planning recommended methodology” was used for the work described above. This make-based flow consists of two major steps. The first step creates the initial starting design and floorplan based on user settings. The next step could be doing timing and routing feasibility analysis for a given netlist and floorplan or running a design exploration flow to generate several different floorplans to find an optimal solution. In this evaluation, the design exploration flow was used to come up with an optimal floorplan for the 74K core.

Figure 5, below, shows the details of the exploration flow utilizing the IC design planning capabilities within ICC.

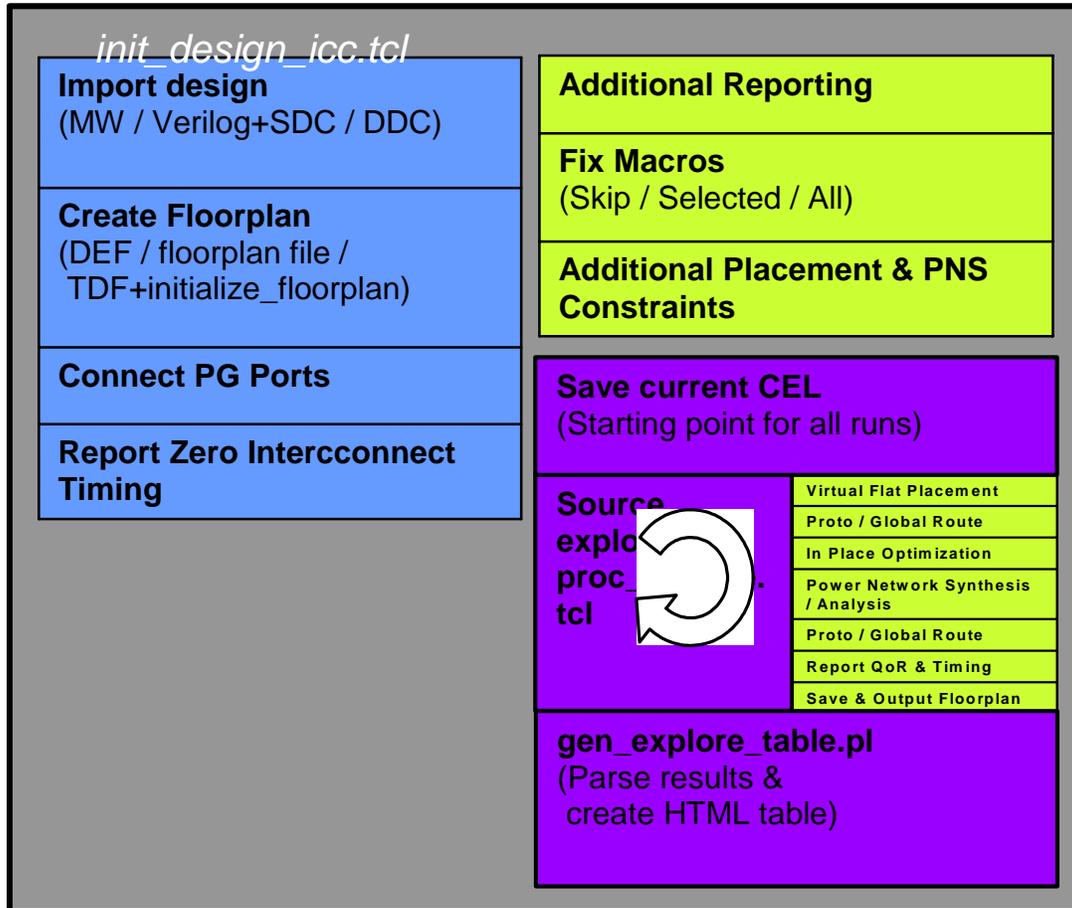
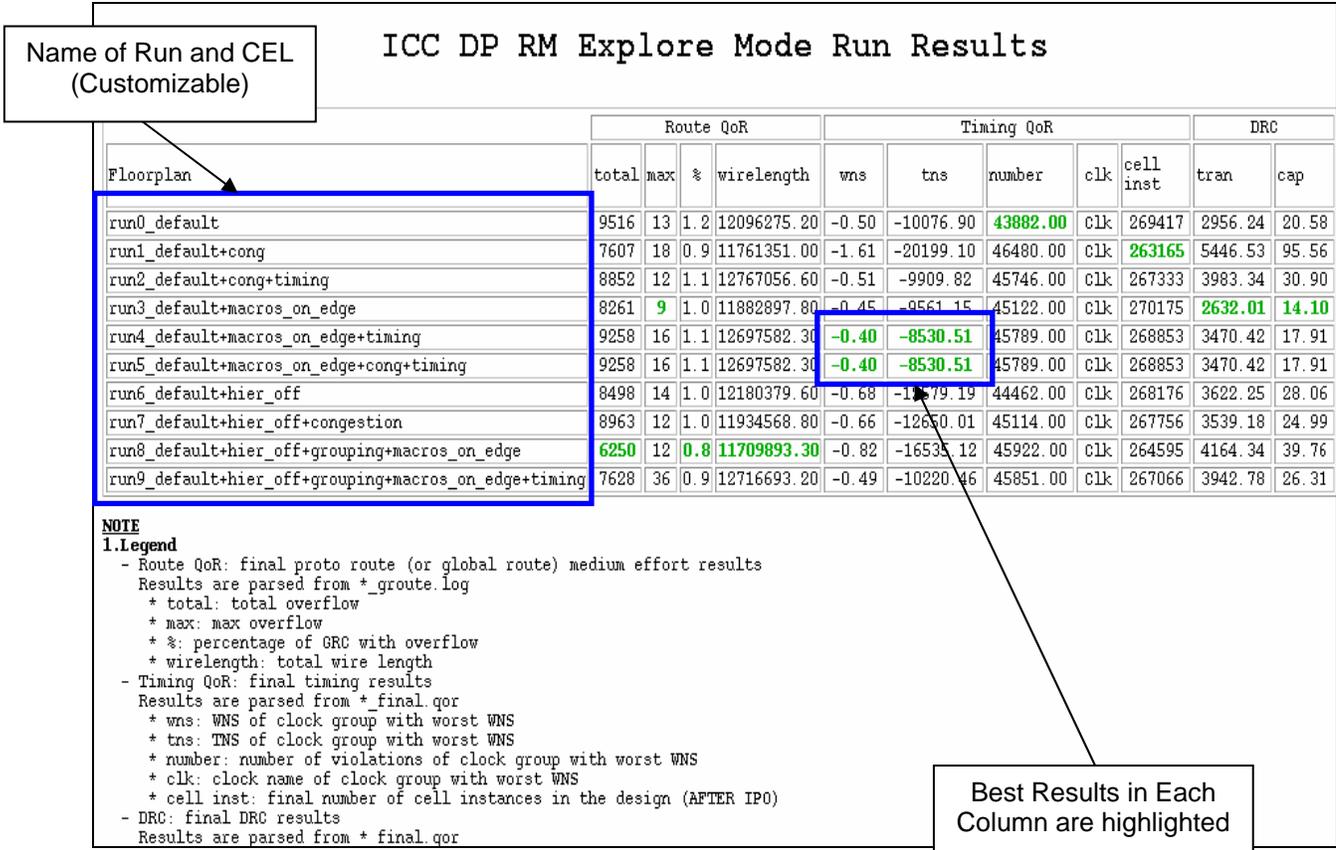


Figure 5 - Explore mode flow in ICC



**Figure 6 - Explore mode results in ICC**

Figure 6, above, shows the QoR results for various floorplans by changing various parameters in explore mode. A fixed aspect ratio was chosen for all the floorplans and only the following parameters were varied in the course of the experiments in this evaluation:

- congestion-driven placement
- timing-driven placement
- *macro\_on\_edge*
- auto grouping

The first two in the list above affect placement of the standard cells and the last two affect placement of the hard macros. The *macro\_on\_edge* parameter forces all the macro cells to be placed on the boundary of the block and the auto grouping parameter clusters macros of the same type together. The tool offers other choices of parameters that can be tweaked in the design exploration flow but were not evaluated in this trial. The best results on the 74K core were obtained by turning on timing-driven and congestion-driven placement knobs in the tool, as well as using the *macro\_on\_edge* parameter along with auto grouping for the hard macros as can be seen in Figure 8.

Figure 7, below, shows the design exploration Tcl script used for generating the various floorplans along with settings used to generate these floorplans and directly reflect the experiments carried out in the course of this evaluation.

```
#####
###
## ICC Design Planning RM
## Version 2007.03
#####
###
## Variable defaults and legal values based on ICC 2007.03
## N/A means not a create_fp_placement option or parameter
## "Brief_Description" column below intends to give you quick reference of what the
option does.
## Please refer to man page of create_fp_placement or set_fp_placement_strategy for
more details.

# Legend          Default      <Range>          (Correspondent command)
Brief_Description
# A run number          N/A      <integer>    for tracking purpose
# B objective           N/A      <string>     for tracking purpose
# C macros on edge      off      <on,off>    (set_fp_placement_strategy -
macros_on_edge) place macros on the edges of chip or plan group
# D auto grouping       low      <none,user_only,low,high> (set_fp_placement_strategy -
auto_grouping) controls amount of macro array packing
#                       none: no grpng | user_only: only user defined
ones created | low: array for small macros | high: array for all macros
# E hierarchy gravity   on       <on,off>    (create_fp_placement -
no_hierarchy_gravity) on : with hier gravity on
# F congestion driven   off      <on,off>    (create_fp_placement -congestion_driven)
# G timing driven       off      <on,off>    (create_fp_placement -timing_driven) ...
#   A   B             C   D   E   F   G   ...
dp_explore run0 default                off low on off    ...
dp_explore run1 default+cong            off low on on     ...
dp_explore run2 default+cong+timing      off low on on     ...
dp_explore run3 default+macros_on_edge  on low on off     ...
dp_explore run4 default+macros_on_edge+timing on low on off     ...
dp_explore run5 default+macros_on_edge+cong+timing on low on on     ...
dp_explore run6 default+hier_off         off low off off   ...
dp_explore run7 default+hier_off+congestion off low off on    ...
dp_explore run8 default+hier_off+grouping+macros_on_edge on high off off   ...
dp_explore run9 default+hier_off+grouping+macros_on_edge+timing on high off off   ...
```

Figure 7 - *explore\_mode.tcl* script

### 3.3.2 ICC design planning floorplan results

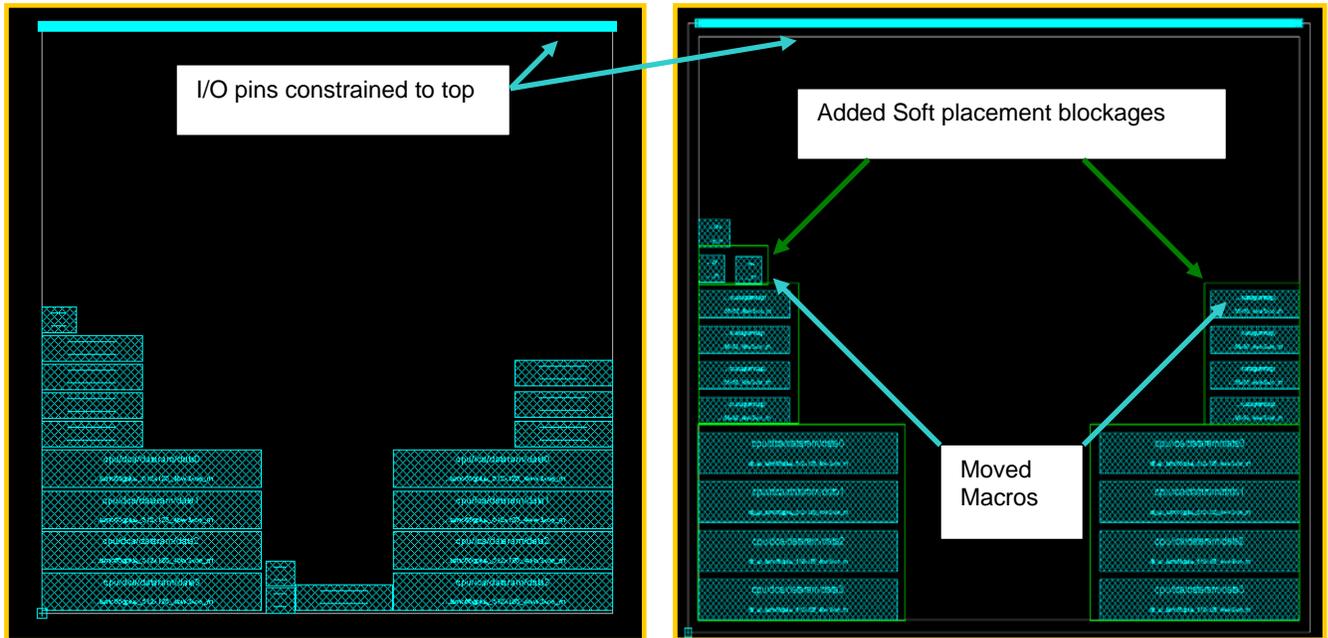


Figure 8 - Floorplan strategy - initial (left) and revised (right)

During this evaluation, it has been observed that placing the pins on one side of the core generates best timing results. Additionally this allows the core to be placed in a corner of the SoC, out of the way of all other logic.

A Tcl script was used to automatically create the TDF (Top Design Format) file on-the-fly in ICC that specifies the pin locations for all the pins and is shown in Figure 9, on the next page. The *PIN\_LAYER* and *PIN\_SIDE* variables allow the user to customize the flow with respect to their choice of layer and side for the pins. The width of the pin was chosen to be the minimum width of the specified *PIN\_LAYER*. The *gen\_tdf.tcl* script will get all of the ports in the design, and write a TDF file in the appropriate format. Figure 10 on the next page, shows a sample of the final TDF output file.

```

#=====
# Filename: gen_tdf.tcl
# Description: File to generate TDF constraints for ports
#
#=====

sh touch io_new.tdf
set layer_num [get_layer_attribute -layer $PIN_LAYER layer_number]
set layer_width [get_layer_attribute -layer $PIN_LAYER minWidth]

if [ file exists $ICC_IN_TDF_FILE ] {
    file remove $ICC_IN_TDF_FILE
}

foreach_in_collection pname [get_ports *] {
    echo "pin [get_attribute $pname full_name] $layer_num
\
        $layer_width $layer_width \"$PIN_SIDE\"" >>
$ICC_IN_TDF_FILE
}

```

**Figure 9 - *gen\_tdf.tcl***

```

pin OC_MData[61]      33      0.100000 0.100000 "top"
pin OC_DMA_SData[50] 33      0.100000 0.100000 "top"
pin L2_cerr_indx[0]   33      0.100000 0.100000 "top"
pin OC_DMA_SData[40] 33      0.100000 0.100000 "top"
pin OC_DMA_MData[1]  33      0.100000 0.100000 "top"
pin OC_SData[26]     33      0.100000 0.100000 "top"
pin OC_DMA_SData[56] 33      0.100000 0.100000 "top"
pin OC_SData[54]     33      0.100000 0.100000 "top"
pin SI_ClkOut        33      0.100000 0.100000 "top"
pin SI_ExceptionBase[17] 33      0.100000 0.100000 "top"
.
.
.

```

**Figure 10 - Generated TDF file**

### 3.3.3 Description of custom power grid

ICC provides the capability to automatically synthesize a power grid. However, for this evaluation a custom power grid was used based on previous Astro flow experiences with IR drop analysis and routability. Metal layers 6, 7, 8 and 9 were used for the power grid with very wide straps in METAL9. A side effect of the heavy METAL9 power grid was that automatic Power Network Synthesis (PNS) would add stacked vias all the way from METAL9 to standard cell power in METAL1 resulting in serious congestion problems as a lot of routing tracks were cutoff in lower layers.

To avoid this problem with stacked vias, stacked vias were only allowed from the lowest layer of the power grid, METAL6 in this case. This was accomplished by setting the MIN and MAX layer setting using the *set\_preroute\_drc\_strategy* available in ICC. METAL9 connects down only to METAL8, METAL8 connects down only to METAL7, METAL7 connects down to METAL6, and finally METAL6 connects with a stacked via down to METAL1. This avoids having a stacked via from METAL9 all the way down to METAL1.

Figures 11 to 14 that follow, describe the power grid used for this implementation of the 74K™ core. Figure 12 on the next page shows the Tcl script used to generate the custom power grid.

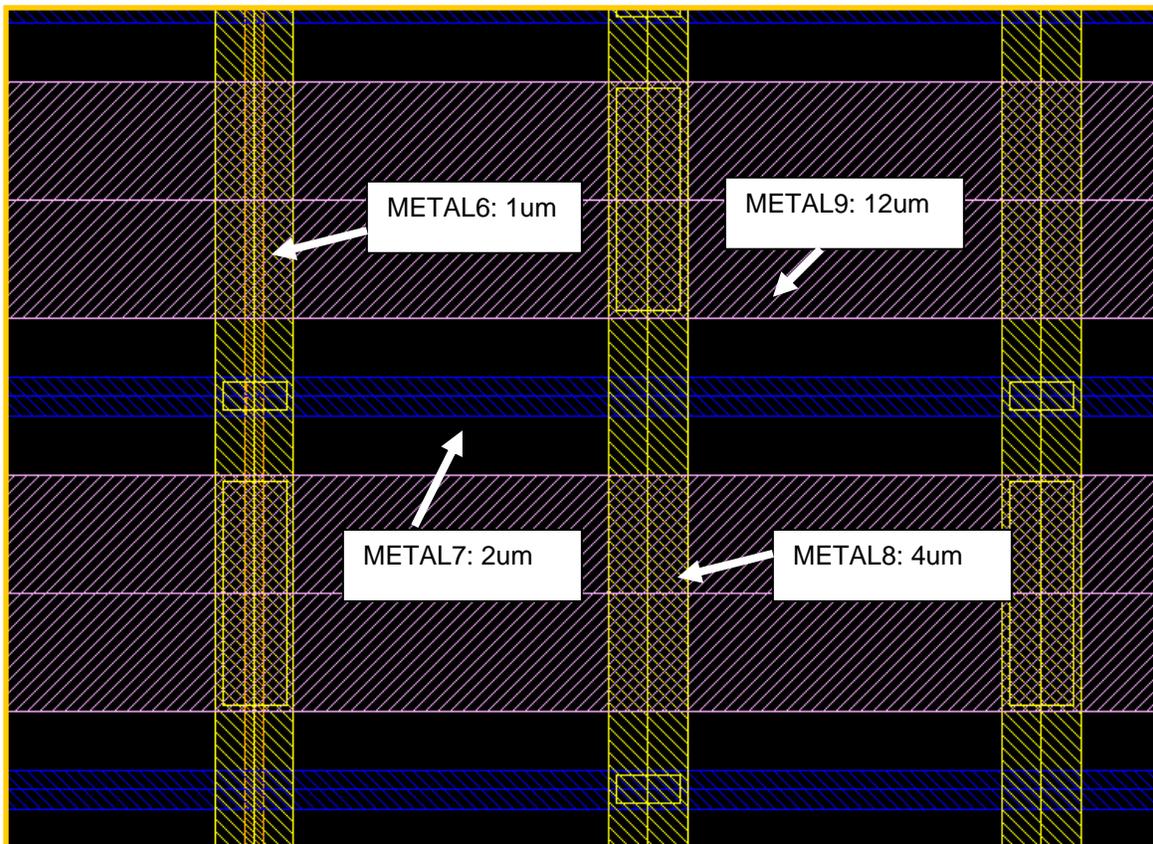


Figure 11 – Custom power grid for the 74K™ core.

```

#-----
# Vertical straps
#-----

set die_area_urx [ get_attribute [current_design] die_area_urx ]
set die_area_ury [ expr $die_area_ury/1000 ]
set die_area_urx [ expr $die_area_urx/1000 ]
set die_area_ury [ get_attribute [current_design] die_area_ury ]
set die_area_ury [ expr $die_area_ury/1000 ]

# Top(Horizontal)

set_preroute_drc_strategy -min_layer M8 -max_layer M9

create_power_straps -direction $PG_TOP_DIRECTION -nets $MW_GROUND_NET -layer
$PG_TOP_METAL -width $PG_WIDTH_TOP \
-configure_step_and_stop -step $PG_PITCH_TOP -stop $die_area_ury -start_at 0 \
-keep_floating_wire_pieces \
-do_not_merge_targets -ignore_parallel_targets

```

Figure 12 - PG pin connections

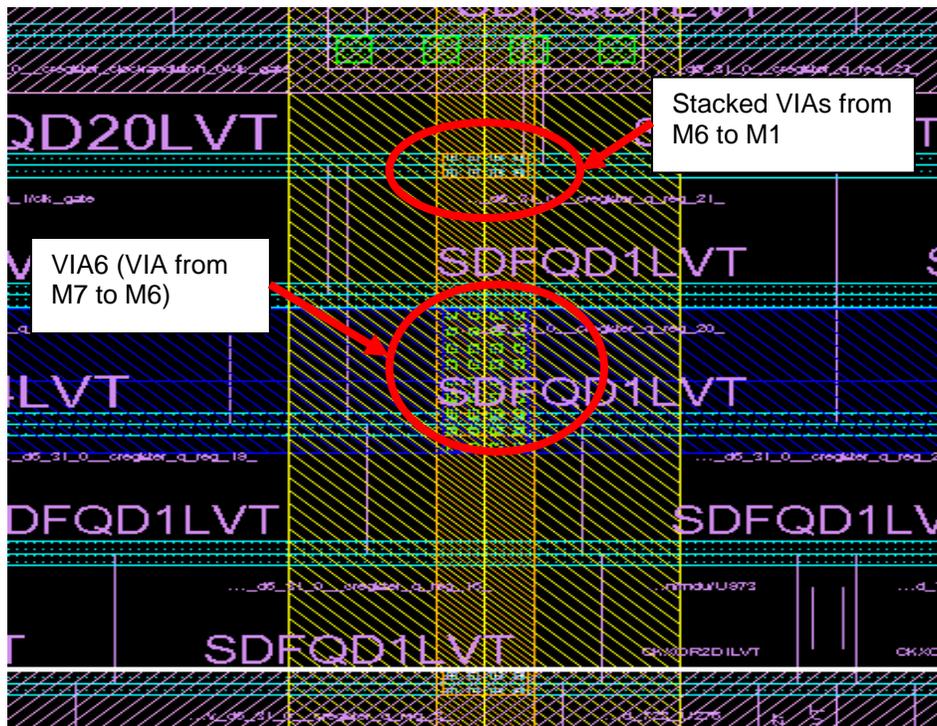


Figure 13 - Only M6 to M1 stacked vias allowed via *set\_preroute\_drc\_strategy*

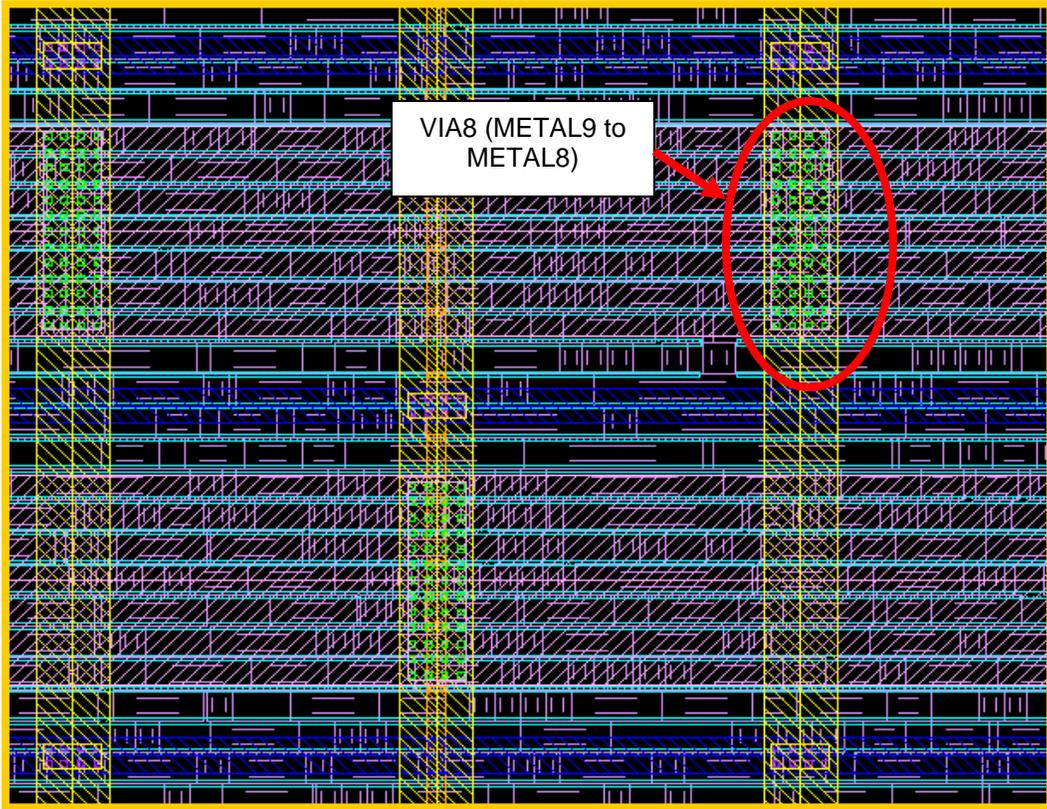


Figure 14 – Layer-to-layer power connections set via *set\_preroute\_drc\_strategy*

### 3.3.4 Power network analysis (PNA)

Some preliminary PNA was done in ICC as a sanity check to verify that the power grid was adequate. The maximum drop was observed to be 12 mV and Figure 15, below, shows the IR drop map. To do preliminary PNA, we used virtual PG rails and virtual PG sources arranged uniformly around the core.

The IR drop analysis map, below, shows a typical hotspot pattern and gives us a good sense that the core will not have IR drop issues when used with this methodology for a SoC.

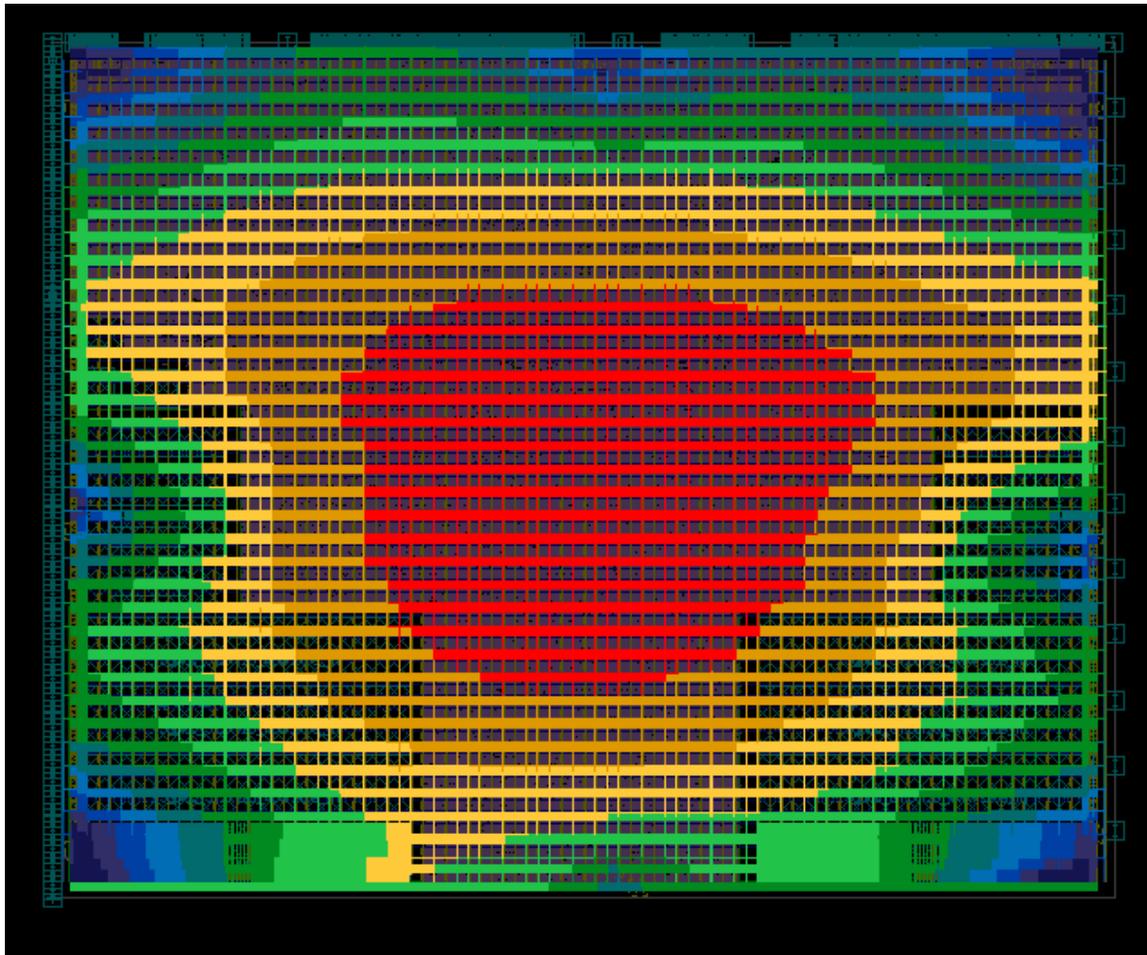


Figure 15 - ICC PNA IR drop map

## 3.4 Overview of ICC flow

### 3.4.1 Overview of the ICC Recommended Methodology (ICC RM) flow

The ICC RM flow was used as a starting point for implementing the RTL-to-GDSII flow for the 74K core. The ICC RM flow is offered by Synopsys as a set of scripts that provide good “out-of-the-box” (OOTB) results for ICC. Users can customize these set of scripts for their specific design environment. Best practices from Synopsys application consultants as well as Synopsys R&D have been incorporated into the ICC RM methodology.

The ICC RM flow includes three steps for placement and optimization, clock tree synthesis and optimization, and routing and post-route optimization. These simple steps have been implemented in ICC using the *place\_opt*, *clock\_opt* and *route\_opt* core commands

Figure 16 on the following page, shows the flow from DC/DC-T through all these steps:

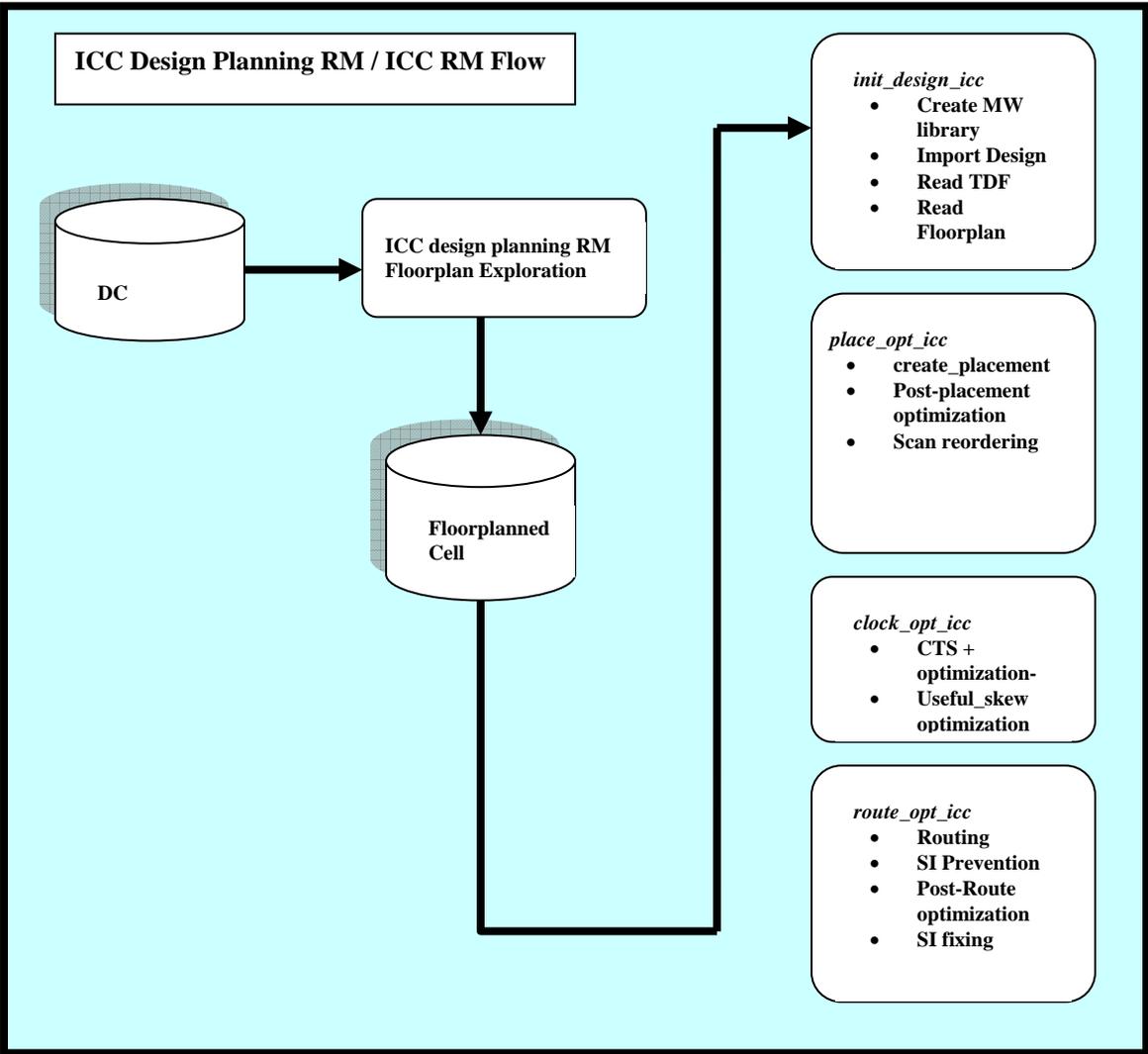


Figure 16 - The 74K™ core ICC flow

### 3.4.2 *place\_opt* optimization strategies

The following strategies were used during placement and optimization for the 74K core:

1. *set\_case\_analysis\_with\_logic\_constants* “true”
2. Path groups were defined with group weights
3. Setting a critical range of about 20% of the cycle time.
4. Special treatment of ICG enable paths
5. *place\_opt* commands used:
  - o *ungroup -all -flatten*
  - o *place\_opt -area\_recovery -effort high -congestion -optimize\_dft*
  - o *place\_fp\_pins -block\_level -verbose* (optimize block-level pins based on placement)
  - o *set physopt\_enable\_adjust\_placement true*
  - o *psynopt -effort high*

The Dolphin memories used in this evaluation have conditional timing based on the settings of the *read\_write\_margin* bus (RWM[2:0] in our specific example). The access time of the memory is dependent on the static value chosen for these pins. By default, the timing engine in DC and ICC will pick the worst timing arc. However, the intent of the design is to pick the timing corresponding to the static setting of the RWM pins. Setting the *set\_case\_analysis\_with\_logic\_constants* variable to “true” ensures that the timing engine in DC and ICC picks up the appropriate timing.

To improve the timing QoR, path groups were created for all paths in the high-speed clock domain (Clk) as well paths that end in the enable pin of the clock gating elements. The following code snippet shows the creation of the path groups and the usage of a higher weight in these path groups.

```
set icg_enable_pins [get_pins -hier "*clk_gate/E"]
group_path -name Clk -weight 10 -critical_range 100 -to Clk
group_path -name icg_enable -weight 10 -critical_range 100 -to
$icg_enable_pins
```

### 3.4.3 *clock\_opt* and useful skew optimization strategies

Useful skew optimization was used in ICC to obtain the best timing QoR for the 74K core. The *skew\_opt* command was used before CTS to write out a clock tree exceptions file which can be used by *compile\_clock\_tree* or the *clock\_opt* command. The *skew\_opt* command analyzes the slacks in the design and tries to optimize the slack with useful skew and these useful skew values are written out as clock tree exceptions. In contrast, when using Astro, such useful skew has to be manually calculated and applied as clock delay targets using the *ataDefineSyncPin* command.

The following section describes the strategy used for CTS in the implementation of the 74K core in ICC in this evaluation:

1. Define triple space Non-Default Rules (NDR) for clocks (SI prevention)

- (Other NDR rules defined, double space and Double-width NDR for use during `route_opt`)
2. Enable Clock Reconvergence Pessimism Removal (CRPR) on recovery and removal arcs
  3. `set_max_transition` on high-speed clock (10% of clock period - SI prevention)
  4. `check_clock_tree` (checks for potential problems that could affect clock tree synthesis-CTS QoR).
  5. Run first-pass `clock_opt` to get estimate of clock latencies
    - a. Run Tcl script to write out IO port latencies
    - b. Run Tcl script to adjust setup time of clock gaters to account for early clock latencies
  6. Read IO port latencies, adjust setup time of clock gaters
  7. Run `skew_opt` (useful skew optimization) prior to CTS
  8. Specific commands used:
    - `check_clock_tree`
    - `clock_opt -inter_clock_balance -no_clock_route -only_cts`
    - `set_propagated_clock [all_fanout -clock_tree -flat]`
    - `source -e adjust_latency.tcl` (has Tcl procedure called `adjust_latency`)
    - `adjust_latency`
    - `set_latency_adjustment_options -from_clock Clk -to_clock vc_Clk`
    - `set_latency_adjustment_options -from_clock EJClk -to_clock vc_EJClk`
    - `update_clock_latency`
    - `source -e write_port_latencies.tcl` (has Tcl procedure called `write_port_latencies`)
    - `write_port_latencies "port_latencies"`
    - `source clock_adj.tcl` (this file is generated by the `adjust_latency` procedure)
    - `skew_opt`
    - `clock_opt -inter_clock_balance -no_clock_route -only_cts`
    - `psynopt -area_recovery`
    - `set_physopt_enable_adjust_placement true`
    - `psynopt -effort high -area_recovery`

Because the IO requirement times (`set_input_delay/set_output_delay`) in the SDC file do not account for clock latencies, IO requirement times need to be adjusted post-CTS (`update_clock_latency`).

However, the recommendation is to adjust latencies before running `skew_opt` since addition of useful skew will bias the average insertion delay in the design. The first-pass of `clock_opt` is to get an estimate of this latency adjustment for the IOs. Additionally, this data is also used to get a better estimate of the insertion delays to the clock gaters which in general have earlier clocks arriving to them as compared to the registers. This difference in insertion delay of clocks to the registers and clock gaters is added as an additional setup margin for each clock gater independently. This enables `skew_opt` to better predict the real QoR of the design including the QoR of paths from register to the enable pins of the clock gaters. The second pass of `clock_opt`

uses the *clock\_tree\_exceptions* generated by *skew\_opt* and is the useful skew scheme used in this evaluation.

Figures 17 and 18, below, show the Tcl procedures used in the CTS methodology used for implementing the 74K core in ICC.

```
proc write_port_latencies { file_name } {
    if { "${file_name}" == "" } {
        puts "usage: write_port_latencies <file_name>"
        return 0
    }
    write_sdc "${file_name}.sdc"
    sh grep set_clock_latency "${file_name}.sdc" > "${file_name}.scl"
    sh grep get_clock "${file_name}.scl" > "${file_name}"
    sh rm -f "${file_name}.sdc"
    sh rm -f "${file_name}.scl"
    return 1
}
```

**Figure 17 - *write\_port\_latencies.tcl* script**

```

# adjust_latency.tcl Script
remove_clock_gating_check [current_design]
remove_clock_gating_check [get_pins -hier "*clk_gate/E" ]

#startpoint_clock_latency
#endpoint_clock_latency

proc adjust_latency { } {
    set design_tns 0
    set design_wns 100000
    set design_tps 0

    set adj_setup_time 0
    foreach_in_collection clk_enable [get_pins -hier "*clk_gate/E" ] {

        foreach_in_collection path [get_timing_paths -nworst 1 -to $clk_enable] {
            set setup_time [get_attribute $path endpoint_setup_time_value]
            set endpoint [get_attribute $path endpoint]
            set endpoint_name [get_attribute $endpoint full_name]
            if { $endpoint_name == "cpu/clock_gate/gate_clockgate/clk_gate/E" } {
                set adj_setup_time [expr $setup_time + 0.0]
                set endpoint_setup($endpoint_name) $adj_setup_time
                set setup_time [get_attribute $path endpoint_setup_time_value]
                set endpoint_latency($endpoint_name) [get_attribute $path endpoint_clock_latency]
                set startpoint_latency($endpoint_name) [get_attribute $path startpoint_clock_latency]
            } else {
                set adj_setup_time [expr $setup_time + 0.0]
                set endpoint_setup($endpoint_name) $adj_setup_time
                set endpoint_latency($endpoint_name) [get_attribute $path endpoint_clock_latency]
                set startpoint_latency($endpoint_name) [get_attribute $path startpoint_clock_latency]
            }
        }
    }

    set fileId [open "clock_adj.tcl" w]

    puts $fileId "set timing_scgc_override_library_setup_hold true"
    puts $fileId "remove_clock_gating_check \[current_design\]"
    puts $fileId "remove_clock_gating_check \[get_pins -hier \"*clk_gate/E\" \]"

    foreach {key value} [array get endpoint_setup] {

        set adj_setup_time [expr $startpoint_latency($key) - $endpoint_latency($key) ]
        set adj_setup_time [expr $adj_setup_time + ${value}]
        if { $adj_setup_time > 0 } {
            puts $fileId "set_clock_gating_check -setup $adj_setup_time [get_attribute $key full_name ]"
        }
    }

    close $fileId
}

```

**Figure 18 - *adjust\_latency.tcl* script**

### 3.4.4 *route\_opt* / signal integrity optimization techniques

The following list describes some of the steps taken to obtain the best timing QoR with and without accounting for signal integrity (SI) effects. In particular, using a tight transition time on the main clock (SI\_ClkIn) as well as using triple spacing to route the clock nets helped improve the SI timing on the 74K core implementation in ICC.

1. Used CWORST extraction corner TLUPlus models for implementation since that extraction corner has the worst SI impact
2. Cross-talk prevention during track assignment
3. Timing-driven global route
4. Turned off crosstalk delta delay during initial fixing
5. Turned on crosstalk delta delay for timing closure with crosstalk
6. Wire spreading to minimize crosstalk effects
7. Used non-default routing rules defined during *clock\_opt* to do automatic wire sizing on critical nets
8. Additional *route\_opt* runs were added to attack the signal integrity challenges

Specific *route\_opt* commands:

- *set\_route\_options -grouse\_clock\_routing normal*
- *route\_group -all\_clock\_nets -search\_repair\_loop 20*
- *set\_si\_options -delta\_delay false*
- *route\_opt -initial\_route\_only*
- *route\_opt -skip\_initial\_route -effort high*
- *route\_opt -effort high -incremental -only\_design\_rule*
- *set\_si\_options -delta\_delay true*
- *route\_opt -xtalk\_reduction*
- *route\_opt -optimize\_wire\_via*
- *route\_opt -incr -only\_size\_wire*
- *route\_spreadwires -widen*
- *route\_opt -incr -only\_hold\_time*

### 3.4.5 Correlation between ICC, Star-RCXT and PrimeTime SI

As stated before, the C-worst extraction corner was chosen for ICC as well as Star-RCXT™ in order to use the most pessimistic corner for implementation and sign-off analysis. Star-RCXT has different modes referred to as MODE 100, MODE 200 and MODE 400 which imply varying levels of accuracy. For purposes of this evaluation, MODE 400 was chosen to obtain the most accurate extraction result which is recommended for technology nodes of 65nm or below.

As shown in Figures 19 and 20, the correlation between ICC and Star-RCXT is excellent for total as well as coupling capacitances. Good correlation is important to ensure that the optimization tool actually works on paths that are seen by the sign-off tool. For this evaluation, since the correlation was quite good, no multiplier for capacitances was used in ICC.

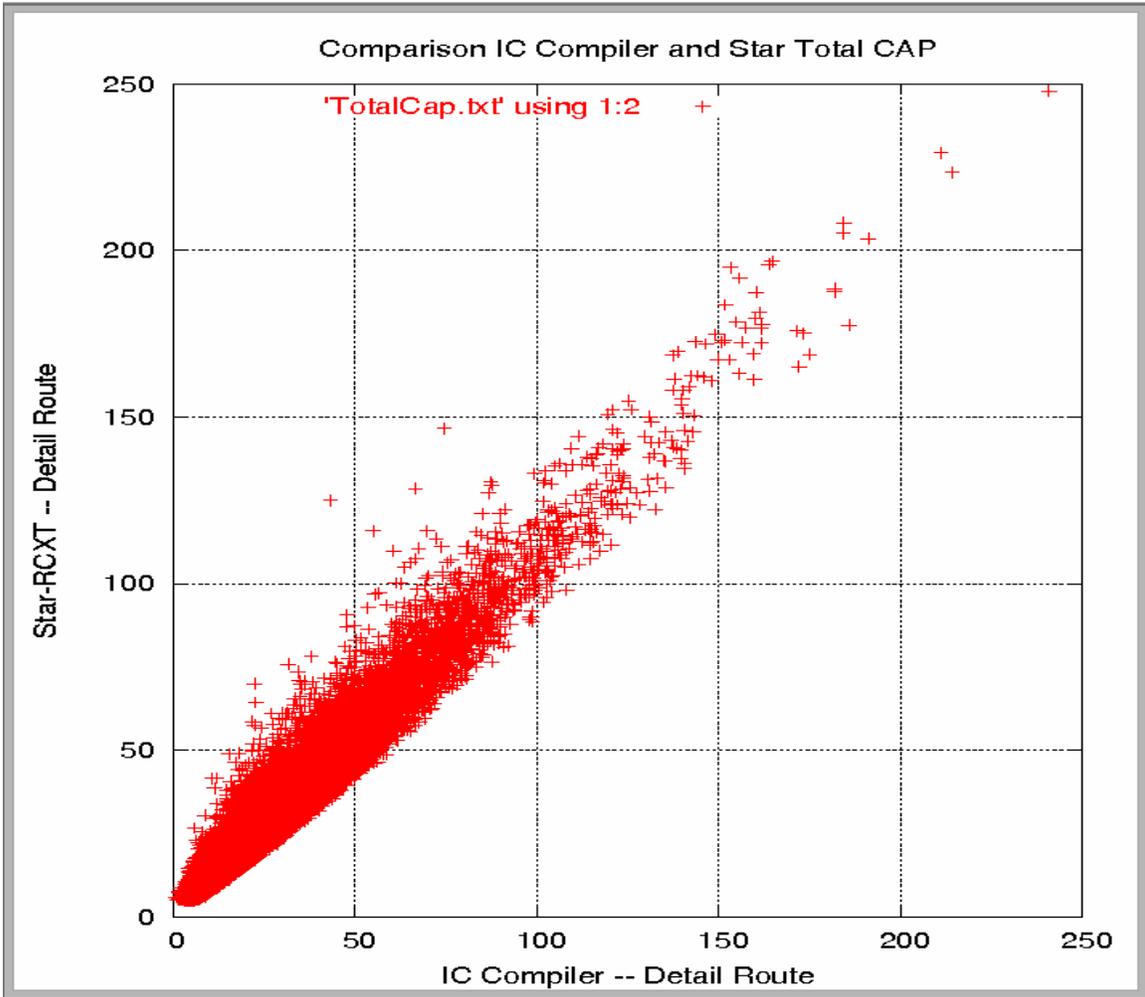
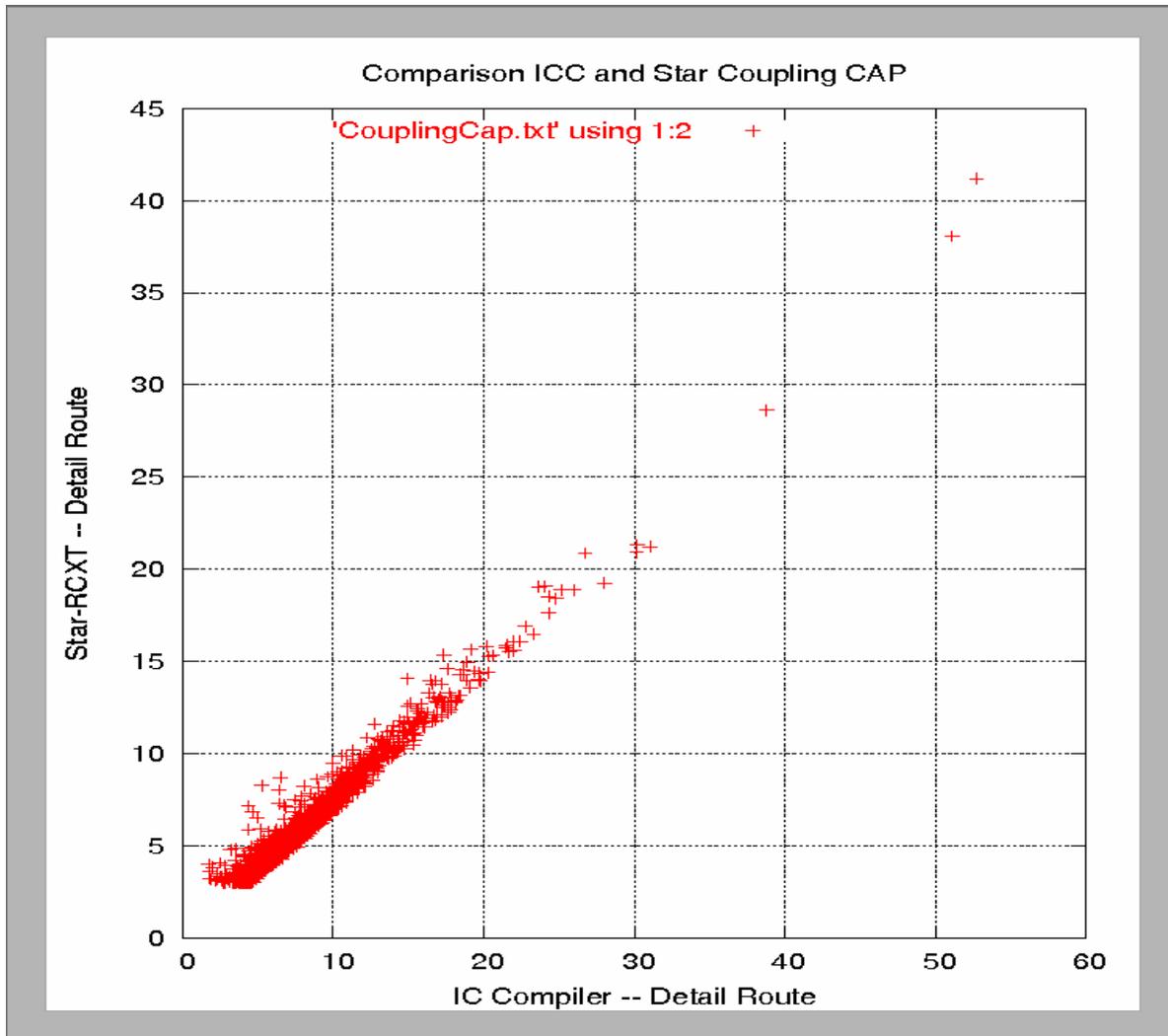


Figure 19 - Total capacitance correlation (values in fF)



**Figure 20 – Coupling capacitance correlation (cap values in fF)**

To achieve good timing correlation between ICC and PT SI, the following settings in ICC and PT SI were used:

ICC:

```
set timing_remove_clock_reconvergence_pessimism true
set_delay_calculatiion -arnoldi
```

PT SI:

```
set si_exit_on_max_iteration_count 3
set si_xtalk_delay_analysis_mode "all_violating_paths"
set timing_remove_clock_reconvergence_pessimism "true"
```

It is important to note that if both the min and max versions of the libraries are loaded in PT SI using the `set_min_library -min_version` command, on-chip-variation (OCV) mode is

automatically activated inside of PT SI regardless of the condition set using the *set\_operating\_conditions* command. In OCV mode with both min and max libraries loaded, the results are unrealistic and pessimistic since two extremes of process corners are used for picking up the fast and slow paths for OCV analysis.

To avoid this problem, setup and hold analysis were carried out in PT SI by loading libraries for a single corner at a time. The code snippet in Figure 21 shows the basic script used for completing the PT SI analysis.

```
#
#set_min_library
#{LIB_PATH}/dti_sp_tsmc65gplus_1024x64_8bw3xoe_m_worst.db \
# -min_version
#{LIB_PATH}dti_sp_tsmc65gplus_1024x64_8bw3xoe_m_best.db
#

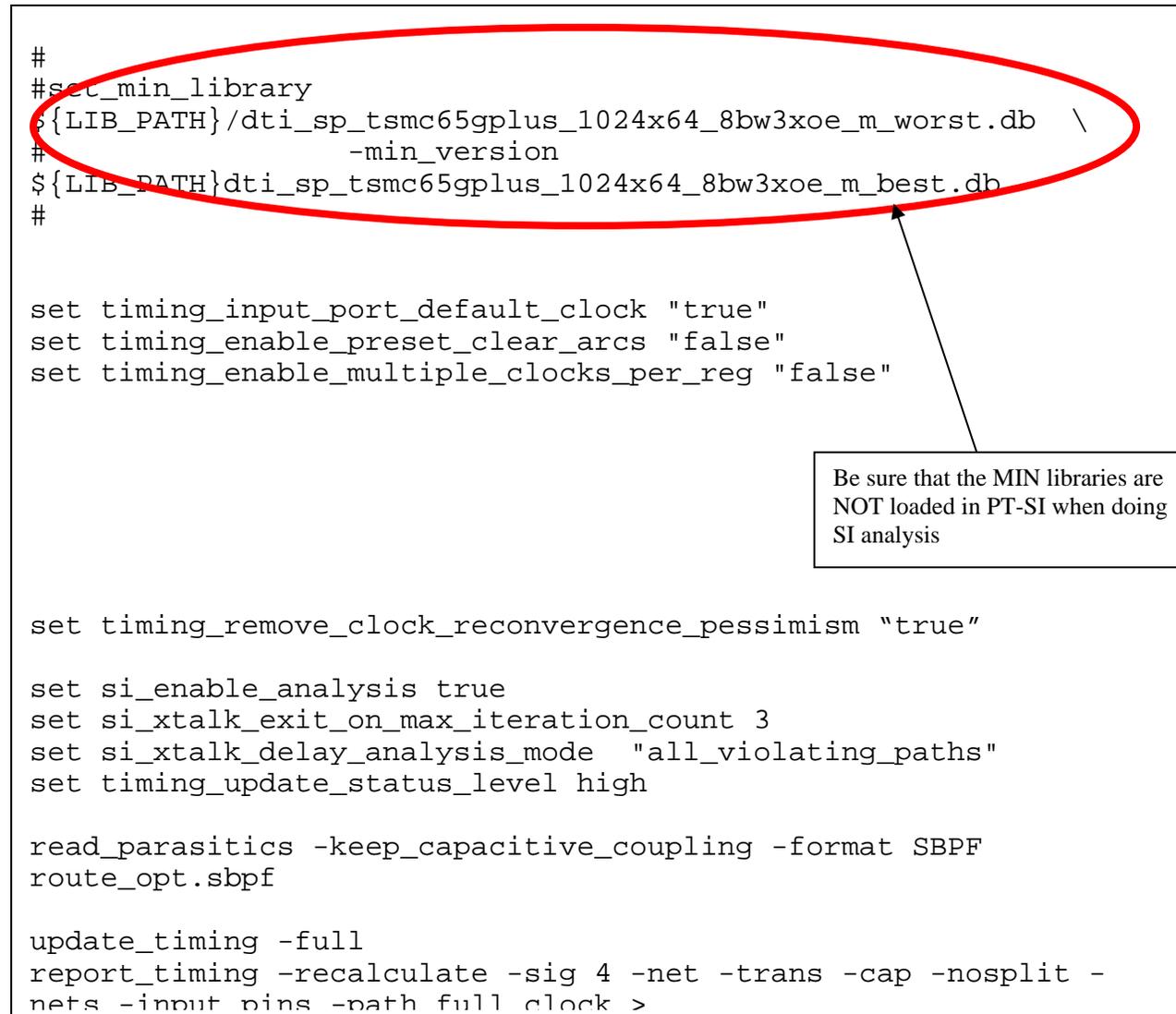
set timing_input_port_default_clock "true"
set timing_enable_preset_clear_arcs "false"
set timing_enable_multiple_clocks_per_reg "false"

set timing_remove_clock_reconvergence_pessimism "true"

set si_enable_analysis true
set si_xtalk_exit_on_max_iteration_count 3
set si_xtalk_delay_analysis_mode "all_violating_paths"
set timing_update_status_level high

read_parasitics -keep_capacitive_coupling -format SBPF
route_opt.sbpf

update_timing -full
report_timing -recalculate -sig 4 -net -trans -cap -nosplit -
nets -input pins -path full clock >
```



Be sure that the MIN libraries are NOT loaded in PT-SI when doing SI analysis

**Figure 21- Example of PT SI script for accuracy (pessimism is removed)**

### 3.5 Review of ICC results

Figure 22, below, summarizes the results achieved for the 74K core as implemented with the above describe DC/DC-T and ICC methodology described above. The PT SI report shows that the 74K core achieved a frequency of 1.12GHz, better than the original target (1GHz) with no significant manual intervention or custom scripts.

Flow Step	WNS (ns)	TNS (ns)	# Violating paths
ICC	<b>-0.071</b>	<b>-2.985</b>	<b>322</b>
Star-RCXT/PT SI	<b>-0.047</b>	<b>-0.966</b>	<b>83</b>

Figure 22 - ICC final results

## 4.0 Conclusions and recommendations

This paper has introduced an automated flow that can achieve frequencies above 1GHz with the 74K core using commercial standard cell libraries, memories and Synopsys tools. The paper has also demonstrated the value of using DC/DC-T in conjunction with ICC design planning and exploration capabilities to provide the best starting point for implementation.

### 4.1 Benefits of the methodology

The methodology described in this paper enables the use of an automated synthesizable approach to achieve very high frequencies with embedded processor cores from MIPS Technologies, Inc. using Synopsys tools.

### 4.2 Areas for future investigation and improvement

The authors will continue working on the methodology as the tools continue to improve. The particular areas of interest are:

- Investigate sign-off driven design closure.
- Investigate better and/or automated ways of handling inherent skews on paths ending in clock gating enable pins pre-CTS.
- Enhance ICC CTS, so that in a single pass, the useful skew optimization to an ICG (for the sake of its enable path) is considered simultaneously with the potential useful skews to that ICG's flops. This would require defining non-stop sync pins on the clock pins of the ICG.
- Enhance the DC-T/ICC flow for low power implementations.
- Add a hierarchical reference flow for integrating the MIPS 74K core in the context of SoC designs.
- Investigate benefits of adding Multi-Corner Multi-Mode (MCMM) optimization to MIPS ICC flow.

### 4.3 Reference flow for the 74K™ core and other MIPS® cores

The reference flow described in this paper based on DC/DC-T and ICC will be available for all embedded processor cores from MIPS Technologies, Inc. including the 74K family of cores with the next maintenance release of the cores.

## 5.0 Acknowledgements

The authors would like to acknowledge Harold Levy and Kevin Kranen from Synopsys, Inc. and Vidya Rajagopalan, Tom Chanak and Sunil Mudunuri from MIPS Technologies, Inc. for their contribution and feedback.

## 6.0 References

- [MIPS32® 74K™ core – White paper](http://www.mips.com/content/74K_home.html)  
([http://www.mips.com/content/74K\\_home.html](http://www.mips.com/content/74K_home.html))
- [MIPS32® 74K™ core – Data sheet](http://www.mips.com/products/cores/32-bit_cores/MIPS32_74K_Family.php#resources)  
([http://www.mips.com/products/cores/32-bit\\_cores/MIPS32\\_74K\\_Family.php#resources](http://www.mips.com/products/cores/32-bit_cores/MIPS32_74K_Family.php#resources))
- Solvnet Article #020097: Useful Skew in IC Compiler
- Solvnet Article #021023: Design Compiler Reference Methodology (DC-RM)
- Solvnet Article #021179: IC Compiler Block-level Floorplanning and Pin Assignment
- Solvnet Article #021197: Design Planning- related functionality inside the IC Compiler GUI

## 7.0 Appendix

The following version of Synopsys tools were used in the course of the evaluation:

- DC Ultra – 2007.03-SP3
- ICC – 2007.03-SP3
- Star-RCXT - 2007.06-1
- PT SI - 2007.06

© 2007 MIPS Technologies, Inc. All rights reserved.

Specifications and information subject to change without notice.

The products described in this document are subject to continuous development and improvement. MIPS, MIPS TECHNOLOGIES Logo, MIPS32, MIPS64, MIPS-BASED, and FS2 First Silicon Solutions Logo are trademarks of MIPS Technologies, Inc. and Registered in the United States Patent and Trademark Office. MIPS, MIPS-3D, MIPS16, MIPS16e, MIPS32, MIPS64, MIPS-Based, MIPSsim, MIPSpro, MIPS Logo, MIPS Technologies Logo, 4K, 4Kc, 4Km, 4Kp, 4KE, 4KEc, 4KEm, 4KEp, 4KS, 4KSc, 4KSd, M4K, 5K, 5Kc, 5Kf, 14KE, 20Kc, 24K, 24Kc, 24Kf, 24KE, 24KEc, 24Kef, 25Kf, 34K, 34Kc, 34Kf, 74K, 74Kc, 74Kf, CNMIPS, “at the core of the user experience.”, Bus Navigator, Clam, cnMIPS, CorExtend, FPGA View, FS2, FS2 First Silicon Solutions Logo, FS2 Navigator, HyperDebug, HyperJTAG, Logic Navigator, MIPS Everywhere Logo, MED, OCI, the Pipeline, Pro Series, Safe-SOC, SmartMIPS, SOC-it, and System Navigator are trademarks or registered trademarks of MIPS technologies, Inc. in the United States and other countries. All other trademarks referred to herein are the property of their respective owners.