

High Fanout Without High Stress: Synthesis and Optimization of High-fanout Nets Using Design Compiler 2000.11

Rick Furtner

Tensilica, Inc

rfurtner@tensilica.com

ABSTRACT

High fanout nets, especially resets and gated clock nets, typically result in long synthesis runtimes, and gives poor results. Fortunately, Design Compiler 2000.11 has added some improvements that can help designers overcome these problems. This paper will first show some of the problems caused by high fanout nets. Then, the new commands available in DC 2000.11 for improving the synthesis results of high-fanout nets will be discussed. Last, synthesis results on test circuits containing some large-fanout nets will be presented.

Table of Contents

Table of Contents	2
Table of Figures	2
1 Introduction.....	3
2 Requisite Knowledge	3
2.1 Design Compiler Constraint Priority	3
2.2 Delay Calculation Review	4
2.3 “Single Path” Visibility of Design Compiler.....	6
3 Design Considerations for High-fanout Nets	7
4 Design Compiler and High-fanout Nets	8
5 Clock Nets.....	9
6 Reset Nets	11
7 “General Purpose” Nets	13
7.1 New/Improved High Fanout Net Commands in DC 2000.11.....	14
7.2 High Fanout Nets Which Will Be Buffered by Design Compiler	15
7.3 High Fanout Nets Which Will Be Fixed in the Back End.....	17
7.4 Caveats on using the <code>high_fanout_net_threshold</code>	20
8 Avoiding Top-Level High Fanout Nets.....	20
8.1 Load Budgeting.....	21
8.2 Avoiding High Fanout Nets in a Configurable Processor	22
9 Conclusions and Recommendations	23
10 Enhancements and Areas for Further Study	24
11 Acknowledgements	25
12 References.....	25
12 Appendix.....	26

Table of Figures

Figure 1	4
Example Circuit for Delay Calculation Discussion.....	4
Figure 2	7
Balanced Buffer Tree.....	7
Figure 3	8
Unbalanced Tree	8
Figure 4	9
Circuit with 3 Types of High-Fanout Nets	9
Figure 5	17
Results of <code>compile</code> on circuit of Figure 4.....	17
(<code>sync_set net</code>).....	17
Figure 6	21
High Fanout Net Caused by Integration	21
Figure 7	23
Tensilica Xtensa Block Diagram.....	23

1 Introduction

High fanout nets, especially resets and gated clock nets, typically result in long synthesis runtimes, and gives poor results. This is due to the disconnect between the pre-layout assumptions made by the static timing analyzer, and the actual post-layout circuit. This paper will focus on the disconnect between pre and post-layout high-fanout nets, and strategies to minimize this disconnect.

2 Requisite Knowledge

In order to understand Design Compiler's behavior when high-fanout nets are encountered and synthesized, it is important to understand three basic concepts about design compiler. These are Design Compiler's:

- Constraint Priority;
- Timing Calculation Mechanics; and
- "Single-path" visibility

Each of these concepts will be briefly explained in the next 3 sections.

2.1 Design Compiler Constraint Priority

A Design Compiler user or library vendor can place several constraints on a circuit. These constraints are numerous and sometimes contradictory. A detailed listing of all possible constraints is beyond the scope of this paper, but can be found in [1]. For the purposes of understanding Design Compiler when encountering high-fanout nets, we can view the constraints and constraint priority that Design Compiler places on a circuit as:

- 1) ***Functional Correctness;***
- 2) **Design Rules (DRC)**
 - i) `max_transition;`
 - ii) `max_capacitance;`
 - iii) `max_fanout;`
- 3) **Setup Time** (`max_delay`);
- 4) Others...

Since items 1) and 2) have a higher default priority than the rest, Design Compiler will often spend an undue amount of time fixing DRC violations on a high fanout net, often at the expense of circuit timing. Although the DRC priority can be overridden with `set_ideal_net` or "`compile -no_design_rule`", DC may still spend a great deal of time fixing any timing violations that may occur as a result of the large net capacitance and pin loading.

2.2 Delay Calculation Review

In order to understand why high-fanout nets cause problems for Design Compiler, knowledge of how cell and net delays are calculated is required. A brief, greatly simplified overview is presented here. For a more thorough overview, please refer to [2]. The remaining discussion will refer to Figure 1 below.

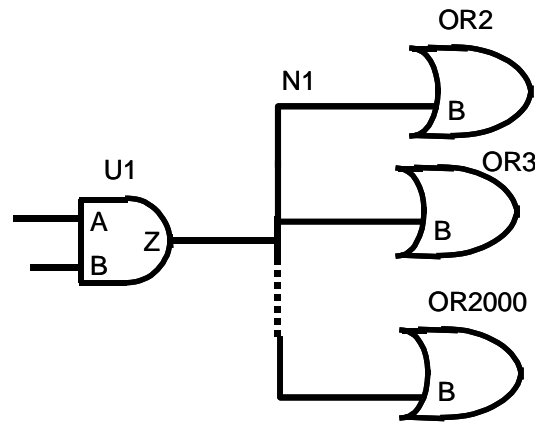


Figure 1
Example Circuit for Delay Calculation Discussion

Cell Delay

Cell delay is affected by several parameters. Two of the parameters that relate to high-fanout nets are:

- a) The *input transition time* on an input pin; and
- b) The *total load* seen by the output pin. The total load is given by the equation:

$$total_load = Perceived_net_load + \sum pin_loads$$

In general, cell delay is directly related to these parameters. This means that the larger the input transition time, the larger the delay of the cell. Also, the larger the total load seen by the cell's output pin, the larger the delay of the cell.

It should be noted that the *input transition time* of a cell is equal to the *output transition time* of the previous, or "upstream" cell in the path. The *output transition time* of a cell is also a function of these two parameters, and will increase as input transition or output load increases.

Net Delay

Net delay is defined as the time it takes a signal to propagate down the RC network formed by the interconnect between logic gates. Although detailed RC timing information is not known until after place and route, Design Compiler will estimate these numbers based on a wire load model. A wire load model is merely a table, which contains an estimated R and C value for every net in the design, indexed by fanout.

After the R and C of the net is determined, the net delay is determined by using the following formula:

$$net_delay = R \times C \times OC_{scale}; \text{ where}$$
$$OC_{scale} = \begin{array}{ll} 1 & \text{for worst_case_tree} \\ \frac{1}{N_{et_fanout}} & \text{for balanced_tree; or} \\ 0 & \text{for best_case_tree} \end{array}$$

worst_case/balanced/best_case tree attributes are part of the operating conditions placed on the current_design, and can be determined via report_lib.

Results

With a basic understanding of Design Compiler's delay calculation engine, it is easy to see why high-fanout nets pose difficulties to Design Compiler. Referring to Figure 1, we can see that the circuit, as shown, will have very poor timing and DRC performance. In fact, several problems will occur in the circuit above:

Gate U1:

- a) **Will have a very large delay**, due to the large capacitive loading the output pin of U1. This is because net "N1" is presenting a large net capacitance, due to its high fanout. And, each of the "OR" gates is contributing an additive pin capacitance to the total capacitance driven by U1.
- b) **Will have a max_transition or max_capacitance violation**, due to the large overall capacitance presented on pin Z of U1
- c) **Will have a very large output transition**, due to the large capacitance seen by pin Z.

Net N1:

Since the net has a high fanout, the wire load model will present a very large R and C component to the delay calculation engine. However, Design Compiler will scale these R and C values based on the *operating condition selected*. In either case, Design Compiler will choose an “incorrect” number for timing and DRC analysis:

- a) **Design Compiler might** choose a very large, and overly pessimistic, delay component. This will happen if an operating condition with ***worst_case_tree*** attribute is selected. When this type of operating condition is used on a design, the very large R and C will be multiplied together. This will result in a huge, and overly conservative, net timing and capacitance.
- b) **Design Compiler might** choose a very small, and overly optimistic, delay component. This will happen if an operating condition with ***balanced_tree*** attribute is selected. When this type of operating condition is used on a design, the very large R and C will be multiplied, and then *divided by the fanout*. When the fanout is high, the net delay can begin to approach 0.

Cells OR2 .. OR2000:

The large transition time presented at the “B” pins of these devices will cause large delays on these cells. It will also cause large transition times on the output of these cells, which will continue to propagate down the timing path.

Clearly, if left unattended, a high-fanout net can wreak havoc on a circuit’s timing and DRC properties. Design Compiler will spend a lot of time attempting to “fix” these violations.

2.3 “Single Path” Visibility of Design Compiler

Design Compiler uses a “cost function” in order to mathematically judge how its design decisions affect the circuit. The details of this cost function can be found in [2]. It can be summed up as:

$$\text{cost} = \sum_0^{\text{all_violators}} \text{viol}_{\text{DRC}} + \sum_0^{\text{critical_range}} \text{viol}_{\text{critical_path}}$$

Basically, Design Compiler keeps track of all the DRC violations, and all of the timing violations, and strives to eliminate the violations by getting the cost function to zero. Algorithmically, Design Compiler accomplishes this task by focusing on a single path at a time, making a fix, and then re-calculating the cost function. A lower overall cost means that the circuit has “improved”.

Although this algorithm is useful for ‘generic’ designs, high-fanout nets often require a more ‘global’ view, in order to properly buffer the many paths contained by the net.

3 Design Considerations for High-fanout Nets

Typically, a high-fanout net will be buffered to reduce the overall load on the driving gate, and decrease the transition time of the net. For signals with identical endpoint timing requirements, such as clock nets and reset lines, a balanced buffer tree can be used, to reduce overall skew between the edges.

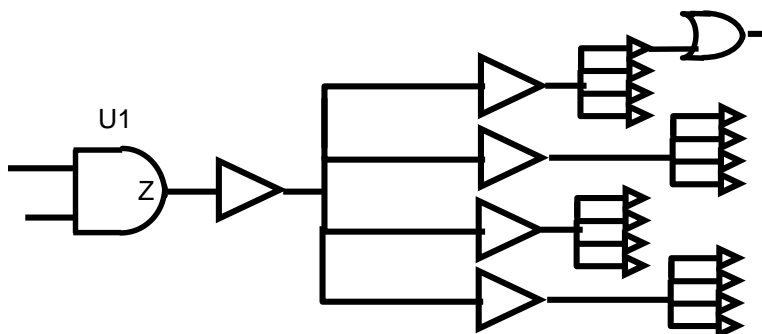


Figure 2
Balanced Buffer Tree

In general, a “physically-aware”, or backend tool, is used to insert clock and balance buffer trees. This is because net lengths and net parasitics must be accurately known in order to properly balance a buffer tree. These parameters cannot be determined until cells have been legally placed, and detailed routing has been performed.

This is *especially true* of “edge-sensitive” nets, such as clock and reset nets. In this case, it is advisable to **have Design Compiler leave the net alone**, and allow the clock-tree insertion tool to do the work.

On high-fanout nets with *unequal* endpoint timing, a ‘balanced tree’ approach is often an efficient implementation. However, it might not be an optimal solution. In the case of Figure 2, above, if we assume that the path through the OR gate has a tighter timing requirement than the other paths, we can see how an ‘unbalanced’ tree could lead to a better result timing result, as shown below:

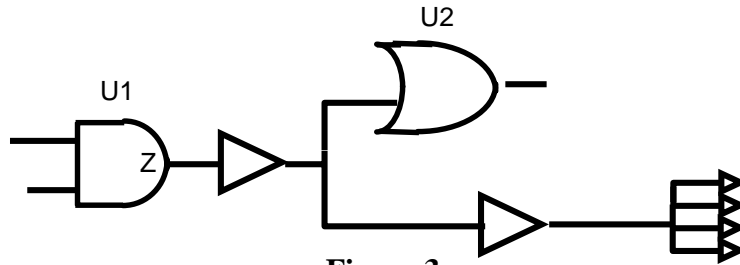


Figure 3
Unbalanced Tree

Clearly, “pulling” the timing critical path closer to the front of the tree will allow the OR gate to receive the signal from U1 sooner than any cells further down the tree.

Given this information, the most important decision a designer can make when a high-fanout net is encountered is:

- a) Should Design Compiler be allowed to “buffer up” the net? Or,
- b) Should it be buffered and balanced in the back end toolchain?

Although there is no “right” answer for all cases, the rest of this paper will present some general rules that a designer can follow to guide this decision.

4 Design Compiler and High-fanout Nets

Design Compiler’s handling of these nets, as well as the commands that apply to these nets, have changed somewhat in the last few releases. Rather than introduce these commands, and then explain them “up front”, they will be introduced as we explore Design Compiler’s behavior on certain high-fanout nets.

Figure 4 below shows a test circuit, which will be used to show how Design Compiler behaves when encountering one of the typical high-fanout nets that are common in most designs:

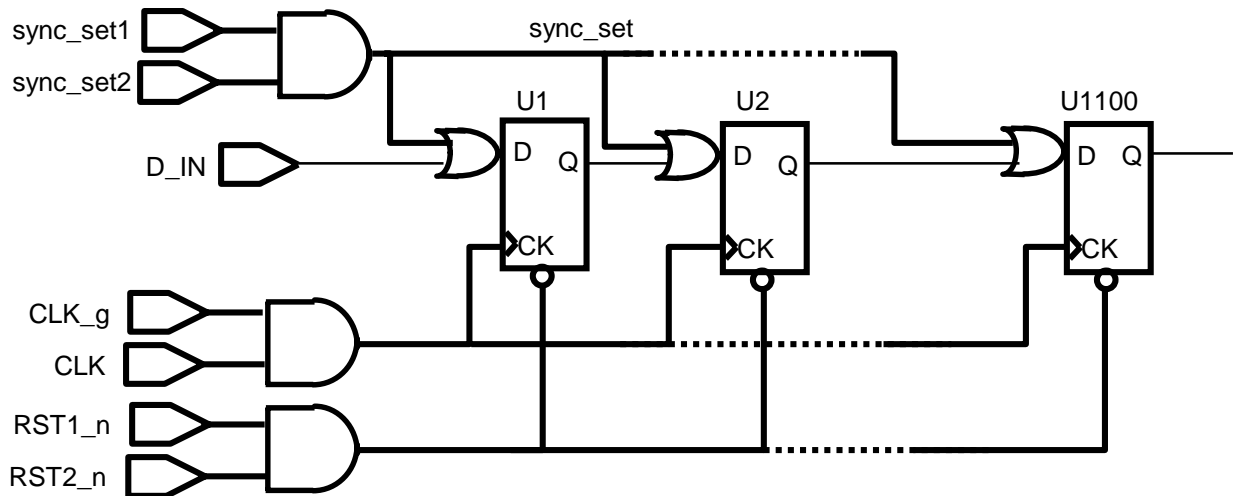


Figure 4
Circuit with 3 Types of High-Fanout Nets

The verilog code that was used to generate this testcase is shown in Appendix A. In particular, there are 3 classes of high-fanout nets that will be looked at in this paper:

- a) A clock net;
- b) A reset net; and
- c) A “general purpose” high-fanout net.

Let’s explore some common myths, and develop some rules, which can help us efficiently design and synthesize circuits containing one of these classes of high-fanout nets.

5 Clock Nets

This is a perfect example of a net that Design Compiler should *leave alone*. These are also the easiest high-fanout nets to handle, because Design Compiler is “smart enough” to know that this net really should be left to a tool with more specialized capabilities. Let’s look at some of the common myths regarding clock nets:

- Myth #1:** If you’re not careful, Design Compiler will “buffer up” the clock net;
- Myth #2:** set_dont_touch_network must be applied to the clock, in order to ensure:
- a) Design Compiler will not ‘buffer up’ a clock,
 - b) Design Compiler will not ‘buffer up’ a gated clock,
- Myth #3:** Design Compiler will calculate a huge transition time for a heavily-loaded gated clock, and all flip-flop clock->Q timing will be incorrect.

Reality As long as you have issued “create_clock” on your clock port, the latest versions of Design Compiler will keep the clock net “buffer free”, even if it is gated. Also, Design Compiler will keep the clock ideal (0 transition time), even if the clock passes through a heavily loaded gate.

Recommendation:

When synthesizing a circuit with a high-fanout clock net, Design Compiler will leave the clock net unbuffered as long as the create_clock command has been issued on the clock port. No other commands need to be issued prior to synthesis. This is true regardless of the presence or absence of clock gating.

A sample “old methodology” gated clock script is:

```
create_clock -p 10 find(port, clk)
set_dont_touch_network find(clock, clk)
if (post_PR_extract)
    set_propagated_clock find(clock, clk)
else
    set_clock_transition 0.1 find(clock, clk).
```

Now, this script can be simplified to::

```
create_clock -p 10 find(port, clk)

if (post_PR_extract)
    set_propagated_clock find(clock, clk)
```

Reasoning:

Design Compiler (as of version 2000.05) places an ideal_net attribute on the transitive fanout of the clock port. This effectively takes the clock net(s) out of the DRC calculation, preventing buffering of the clock to fix max_capacitance and max_transition violations.

Also, although the on-line documentation seems to indicate the contrary, the ideal_net attribute also forces the delay calculator to use an “ideal” clock waveform, even when passing through a heavily loaded clock gate!

Running the “simple” testcase above through 4 different libraries, two 0.18-micron libraries, and 2 different 0.13-micron libraries has shown this contention to be true on a variety of libraries.

To make sure that the simplicity of the testcase is not skewing the results, the test was re-run. This time, I tested a Tensilica XtensaTM processor, which was purposely configured to have a high-fanout gated clock net. This was accomplished by configuring a large (64-place) register file and clock gating. Synthesis was performed on this “real-life” circuit as well. The results of these runs are shown in Table 1 below:

		# of buffers	on clock
Circuit	Library	2000.11	2000.05
Simple Testcase	0.18; Vendor "I"	0	0
	0.18; Vendor "L"	0	0
	0.13; Vendor "N"	0	0
	0.13; Vendor "A"	0	0
Xtensa Processor	0.18; Vendor "I"	0	0
	0.18; Vendor "L"	0	0
	0.13; Vendor "N"	0	0
	0.13; Vendor "A"	0	0

Table 1
Summary of Results for High-fanout Gated Clock
(Only `create_clock` Applied)

6 Reset Nets

Typically, a reset is driven from a state machine, which ensures proper reset pulse duration, deglitching, etc. Due to the nature of the reset signal, it is important that the reset signal be asserted and deasserted in a controlled fashion, and nearly simultaneously for each flop. This will ensure that

- a) All flops come out of reset at the same time; and
- b) Recovery/removal constraints on flop reset pins can be met.

This is another classic case of a high-fanout net which needs to be balanced by a back-end tool. So, we must convey to Design Compiler the fact that the circuit used for synthesis is “incomplete” (buffers will be inserted later), and timing and/or DRC violations on the reset net can be ignored. Let’s look at some of the common pitfalls regarding reset nets:

Myth: As long as I apply `set_dont_touch` to my reset net, Design Compiler will leave it alone. I can then pass the netlist to the backend tools to do the rest.

Reality Even though `set_dont_touch` will prevent Design Compiler from buffering the reset net, the large DRC violation will still be in the cost function. This will result in two main disadvantages:

- a) Design Compiler’s need to preserve *functional correctness* may cause it to add inverters to the reset net
- b) Compile time will increase dramatically, because Design Compiler will attempt to fix a large transition violation by “upsizing” cells prior to the

- high-fanout reset line and “downsizing” any flip-flops loading the reset net; and
- c) Given the priority of DRC violation over timing, and Design Compiler’s critical-path-centric algorithm, the large DRC violation could result in the *rest of your circuit not being fully optimized*.

Recommendation:

Regarding item a) above, to avoid inverters on the reset line, it will be necessary to impose a coding guideline which requires designers to use the correct sense.

Unlike clock nets, other high-fanout nets do not automatically inherit the `ideal_net` or `dont_touch` attribute automatically. So, these attributes must be applied manually. Although the name of the reset net may not be obvious, finding it and identifying it *before* compile can greatly reduce synthesis runtimes, and give a higher QOR (quality of results). A sample script which shows proper reset constraining is shown below:

```
DESIGN_NAME=tst
read -f verilog DESIGN_NAME + ".v"
link

create_clock -period 5.0 find(port, clk*)

set_dont_touch find(net, resetG)
set_ideal_net find(net, resetG)

compile
. . .
```

To show the benefits of using these commands, the simple testcase was compiled in three ways:

- “Straight” compile, no `set_dont_touch` or `set_ideal_net`
- “old” methodology: (`set_dont_touch` (SDT) only); and
- Recommended methodology: `set_dont_touch` and `set_ideal_net` (SDT/SIN)

The results of these runs are shown in Table 2 below:

Library	Straight	SDT Only	SDT/SIN
0.18; Vendor “I”	47	0/187 min	0/39 min
0.18; Vendor “L”	53	0/201	0/47
0.13; Vendor “N”	49	0/162	0/35

Table 2
Number of buffers on reset net/Compile time
for 3 different methodologies

Note that by telling Design Compiler to leave the net alone (via `set_dont_touch`), and to disregard DRC violations (via `set_ideal_net`), the fastest runtimes will occur.

Note: In rare cases, applying `set_dont_touch` and `set_ideal_net` will not result in a buffer-free reset net. This is due to STAR 114839, which has been fixed in 2000.11-SP1 [3]

It should be noted that even though the reset net will now be “left alone” by Design Compiler, *the delay calculator will still show a large delay* on the reset net! This is because even though the net is set as “ideal”, it is important to remember that `set_ideal_net` only takes the net’s DRC calculation out of the cost function; it does not change the timing calculation. This can be shown by executing the command:

```
report_timing -net -from find(port, rst*)
```

on the circuit of Figure 4. The results are shown below:

Point	Fanout	Incr	Path

input external delay		1.00	1.00 r
rst_N2 (in)		0.00	1.00 r
rst_N2 (net)	1	0.00	1.00 r
U2303/Y (NANDxxx)		67.47	68.52 r
n6118 (net)	1100	0.00	68.52 r
dout_reg/RN (DFFxxx)		0.00	68.52 r
data arrival time			68.52

(Path is unconstrained)			

Note the extremely large delay on the NAND gate driving the reset line of the flip-flops. This shows that the `set_ideal_net` command did not have an affect on the delay calculator. In the case of the reset net, this large delay did not cause a violation, since most libraries (including the 6 used as ‘test cases’ for this paper) do not place a timing constraint on the reset pin. If the library was characterized such that there was a setup or hold requirement (with respect to clock) on the reset pin, the large delay would cause a timing violation. The designer would then have to modify the script used to constrain the reset net, as outlined in the next section.

7 “General Purpose” Nets

As mentioned in Section 3, “backend” tools that have knowledge of the location of the net driver and receivers often handle large fanout nets better than Design Compiler. Prior to DC 2000.11, if a designer chose to fix a high fanout net with the layout tools, there was no way to tell Design Compiler to completely ignore these nets.

Now that we have looked at the common special-purpose high fanout nets, let’s explore methodologies for getting “general purpose” high fanout nets through synthesis. Keep in mind

that there are no “hard and fast” rules that can be given regarding how large of a high fanout net can be safely synthesized in Design Compiler. The following ‘broad guideline’ is the only recommendation that can be made with some certainty:

High Fanout Net Guideline: *nets with a fanout of thousands are strong candidates for ‘back end’ layout.* Conversely, it’s also safe to say that nets with a fanout of a few hundred or less can usually be handled by Design Compiler.

Results will vary greatly with vendor library, DRCs, timing constraints, and availability of high-drive buffers in the synthesis libraries.

7.1 New/Improved High Fanout Net Commands in DC 2000.11

DC 2000.11 and DC2001.08 have implemented several improvements for synthesis of nets with a high fanout. The improvements come from two main factors; a new delay model for high fanout nets, and improvements to the `balance_buffer` command

Identifying High Fanout Nets

With DC 2000.11, identifying high fanout nets has become easier. To identify high fanout nets prior to synthesis, you can simply issue the commands:

```
report_high_fanout; (DC 2000.11)
report_net_fanout (DC 2000.11-SP1 and later)
```

Once extremely high-fanout nets have been identified, a decision can be made as to whether or not to let Design Compiler buffer up the net.

High Fanout Net Delay Model Improvement

DC 2000.11 prevents the synthesis design rule and delay calculator from bogging down on high fanout nets by “clamping” the fanout of the net during *delay and DRC calculation*. This means that a net exceeding a threshold fanout value is assumed to have a smaller fanout during delay/DRC calculations (the actual fanout of the net is unchanged). Setting the Design Compiler variable `high_fanout_net_threshold` controls this threshold value. For example, issuing the command

```
high_fanout_net_threshold=200
```

prior to compile causes Design Compiler to “pretend” that the net has a fanout of 200 during delay and DRC calculations, regardless of its actual fanout. This will reduce any timing or DRC violations calculated for the net, and can greatly speed up synthesis times. (Pin capacitance is given by the `high_fanout_net_pin_capacitance` variable). Runtime is also improved because the delay attributed to the net is only calculated once ([4], [5]). This variable defaults to 1000; in order to have DC revert to its “old” behavior, it should be set to zero.

balance_buffer Improvements

DC 2000.11 has added several improvements to the `balance_buffer` command, including:

- a) “New and improved” algorithms
- b) Can be executed on paths with timing violations (previously only executed on paths with DRC violations)
- c) `-force` option allows `balance_buffer` to be executed on any path
- d) `-prefer` option allows user to indicate a preferred buffer to use
- e) `-hierarchy` switch allows `balance_buffer` to cross hierarchy
- f) Design Compiler now uses the `balance_buffer` algorithm during synthesis, for high fanout nets

With these high fanout net improvements to Design Compiler, we can now explore some methodologies for getting circuits that contain such nets through synthesis with less pain than in previous versions.

7.2 High Fanout Nets Which Will Be Buffered by Design Compiler

Given the improvements made to Design Compiler, several truths regarding the synthesis of high fanout nets have now turned into “myths”. Specifically:

Myth: The `compile` command

- a) Does not build efficient buffer trees
- b) Creates inefficient buffer chains
- c) Cannot efficiently optimize high fanout nets for timing

Reality Given the improvements to high-fanout net optimization outlined in section 7.1 above, Design Compiler now does an efficient job of synthesizing high fanout nets.

To show this, the circuit of Figure 4 was again synthesized, using 3 different methods:

- a) With no constraints;
- b) With a 1ns constraint on the `sync_set` inputs; and
- c) Using the `balance_buffer` command

The full script that was used to compile the design is shown below:

```
DESIGN_NAME=tst
CLK_PERIOD = 2.5
read -f verilog DESIGN_NAME + ".v"

/* Use full net fanout for delay calculation */
high_fanout_net_threshold = 0

/* Pass 1 - no constraints */
create_clock -period CLK_PERIOD find(port, clk)
```

```
write -hier -o elab.db
compile
write -hier -o run1.db

/* Pass 2 - compile w/ lns constraint */
remove_design DESIGN_NAME
read elab.db
set_input_delay 1.5 -clock clk find(port, sync_set*)
compile
write -hier -o run2.db

/* Pass 3 - Balance Buffer */
remove_design DESIGN_NAME
read elab.db
set_input_delay 1.5 -clock clk find(port, sync_set*)
set_dont_touch find(net, sync_set)
set_ideal_net find(net, sync_set)
compile
balance_buffer -net find(net, sync_set)
write -hier -o run3.db
```

A comparison of the results are shown in Figure 5 below:

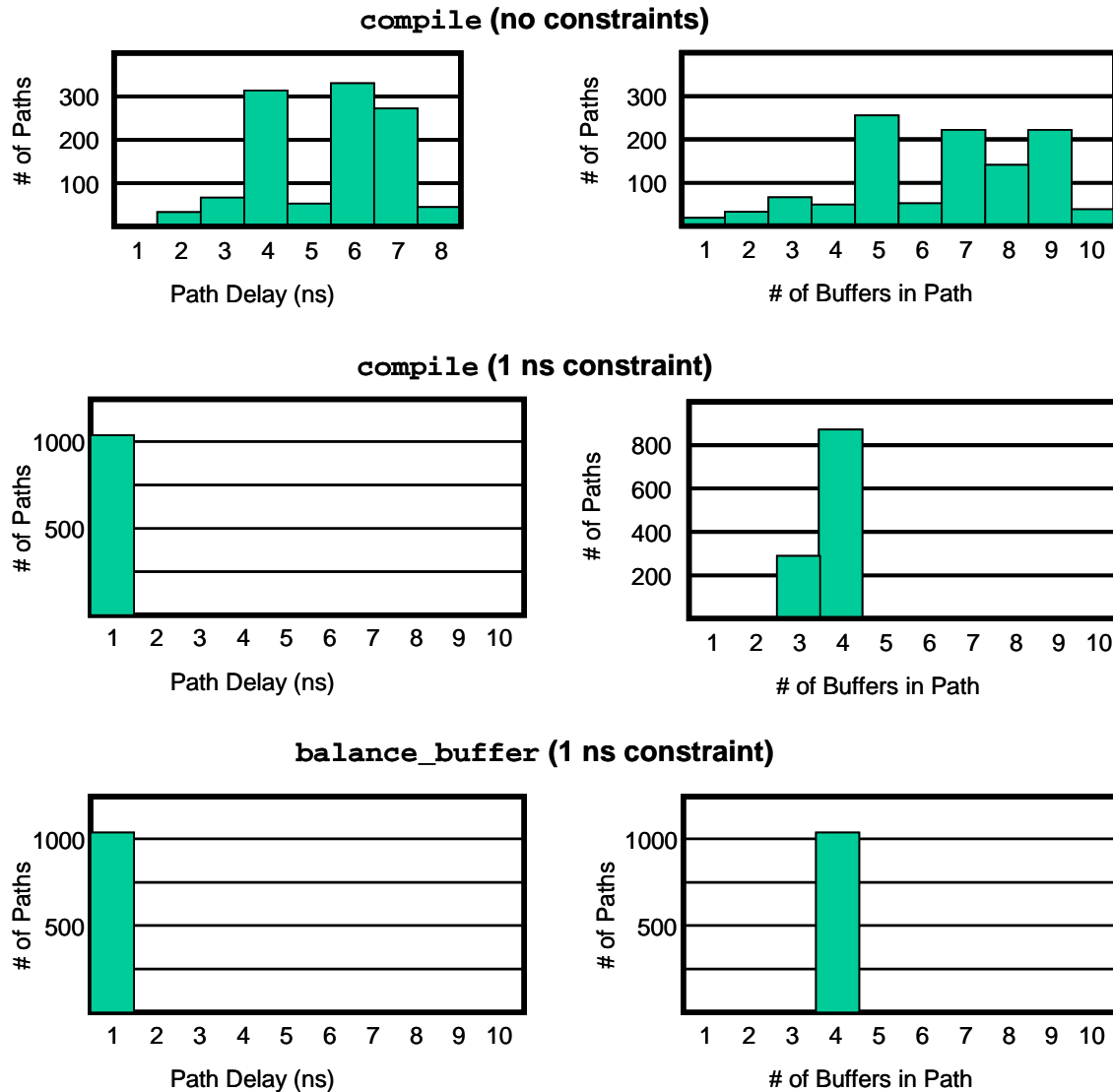


Figure 5
Results of `compile` on circuit of Figure 4
(`sync_set` net)

The above figure shows that `compile` now does almost as good of a job on high-fanout nets as `balance_buffer` does!

7.3 High Fanout Nets Which Will Be Fixed in the Back End

For these nets, we would like Design Compiler to leave the net alone, but also to “ignore” the effects of the large fanout, so that the back-end tools can balance the tree. Unfortunately, none of the previously existing commands would allow Design Compiler to completely ignore the net. Let’s briefly examine why:

set_ideal_net <net> DC only ignores DRC violations on the net during synthesis. Buffers can still be inserted on the net to fix timing violations

set_dont_touch <net> DC does not ignore DRC violations, and could upsize driving cells and downsize receiving cells in order to reduce a DRC violation caused by a high fanout net

set_dont_touch <net>/ **set_ideal_net** <net> Most effective for getting DC to leave a net unbuffered (see Table 2). Unfortunately, runtime is still affected by constant recalculation of path delay as gates at the end of the net change. Also, although the **set_ideal_net** command causes Design Compiler to ignore DRC violations, the delay calculator *does not take into account* the fact that the designer plans on fixing the net later.

To emphasize the effect of a “dont_touch”ed high fanout net on the delay calculator, the circuit of Figure 4 was synthesized after applying the commands:

```
set_dont_touch find(net, sync_set)
set_ideal_net find(net, sync_set)
```

A portion of the resulting timing report is shown below:

Point	Incr	Path
-----	-----	-----
clock clk (rise edge)	0.00	0.00
clock network delay (ideal)	0.00	0.00
input external delay	1.00	1.00 r
sync_set2 (in)	0.00	1.00 r
U6406/Z (ANDxxxx)	59.71	60.71 r
U13112/Z (ND2xxxxx)	4.96	65.66 r
tmp_reg[91]/D (FD2xxxx)	0.00	65.66 r
data arrival time		65.66
clock clk (rise edge)	2.50	2.50
clock network delay (ideal)	0.00	2.50
tmp_reg[91]/CP (FD2xxxx)	0.00	2.50 r
library setup time	-0.42	2.08
data required time		2.08
-----	-----	-----
data required time		2.08
data arrival time		-65.66
-----	-----	-----
slack (VIOLATED)		-63.59

Note the extremely large delay of the AND gate (U6406). This is a direct consequence of the large capacitance presented by the high-fanout net. Besides causing timing constraints that are impossible to meet, the large transition time caused by the high-fanout net will propagate to all downstream paths in a more complex circuit.

In DC 2000.11 and later, the excessive runtime and incorrect timing reports can be further improved by using the `high_fanout_net_threshold` variable. A sample script that shows how to accomplish this result on the simple `sync_set` net of Figure 4 is shown below:

```

/* Tell DC not to buffer the net */
set_dont_touch find(net, sync_set)
set_ideal_net find(net, sync_set)

/* Tell DC to "pretend" sync_set is not a */
/* high fanout net for timing & DRC calculation */
/* (net timing will only be calculated once) */

high_fanout_net_threshold=100

compile
. . .

```

In the script above, by setting the `high_fanout_net_threshold` to a number that is less than the fanout of `sync_set`, the delay of the gate driving the `sync_set` net will be greatly reduced. An added benefit is that the delay of the net itself will only be calculated once. This will further reduce DC runtime. The results of executing the above script file on the circuit of Figure 4 is shown in the timing report excerpt below:

```

# A fanout number of 100 was used for high fanout net
computations.
...

```

Point	Incr	Path
-----		-----
clock clk (rise edge)	0.00	0.00
clock network delay (ideal)	0.00	0.00
input external delay	1.00	1.00 r
sync_set2 (in)	0.00	1.00 r
U6406/Z (ANDxxxxx)	2.17 #	3.17 r
U13112/Z (ND2xxxxx)	0.35 #	3.52 r
tmp_reg[91]/D (FD2xxxx)	0.00	3.52 r
data arrival time		3.52
clock clk (rise edge)	2.50	2.50
clock network delay (ideal)	0.00	2.50
tmp_reg[91]/CP (FDxxxxx)	0.00	2.50 r
library setup time	-0.27	2.23
data required time		2.23
-----		-----
data required time		2.23
data arrival time		-3.52
-----		-----
slack (VIOLATED)		-1.29

Note that the timing of the AND gate is much lower. Although there is still a timing violation, it is much more reasonable. Although the violation could be reduced further by setting the `high_fanout_net_threshold` lower, lowering the threshold too much could cause inaccurate timing results elsewhere in the circuit. This is because an extremely low threshold would result in several hundred net delays being calculated using the threshold.

7.4 Caveats on using the `high_fanout_net_threshold`

Since the previous two sections mentioned the `high_fanout_net_threshold`, a few points need to be made regarding its use:

- 1) **The `high_fanout_net_threshold` is not saved in `.db`!** This means that the variable needs to be set in the script file each time the `.db` file is read (Otherwise, the default of 1000 will be used)
- 2) **The `high_fanout_net_threshold` applies to all nets above the threshold value.** There is currently no way to apply this value to some high-fanout nets, and have DC use the actual fanout of others.
- 3) When calculating the delay of the driving cell a net whose fanout is larger than the `high_fanout_net_threshold`, DC will no longer calculate the total pin loading of the driving cell. Instead, the pin loading will be the `high_fanout_net_threshold` multiplied by the **`high_fanout_net_pin_capacitance`** variable.
- 4) The `high_fanout_net_pin_capacitance` variable *is not saved in the `.db`*
- 5) The `high_fanout_net_pin_capacitance` variable *defaults* to 1.0. This may not be a valid pin capacitance! It's better to set this variable manually, based on a typical library pin capacitance, as shown below:

```
get_attribute <libname>/<buffname>/<pinName> capacitance
/* libname  = logical name of target library
   buffname = name of medium-drive buffer
   pinName  = name of buffer's input pin */
high_fanout_net_pin_capacitance = dc_shell_status
```

8 Avoiding Top-Level High Fanout Nets

It is best to avoid high-fanout nets wherever possible. Of course, avoiding a high fanout on clock and reset nets is not possible. However, “general purpose” high-fanout nets can sometimes be avoided.

In many cases, high fanout nets are a result of integrating several blocks, which contain “medium fanout nets”. In Figure 6 below, the enable line of each individual design does not have a particularly high fanout. After integration, however, the fanout is considerably larger.

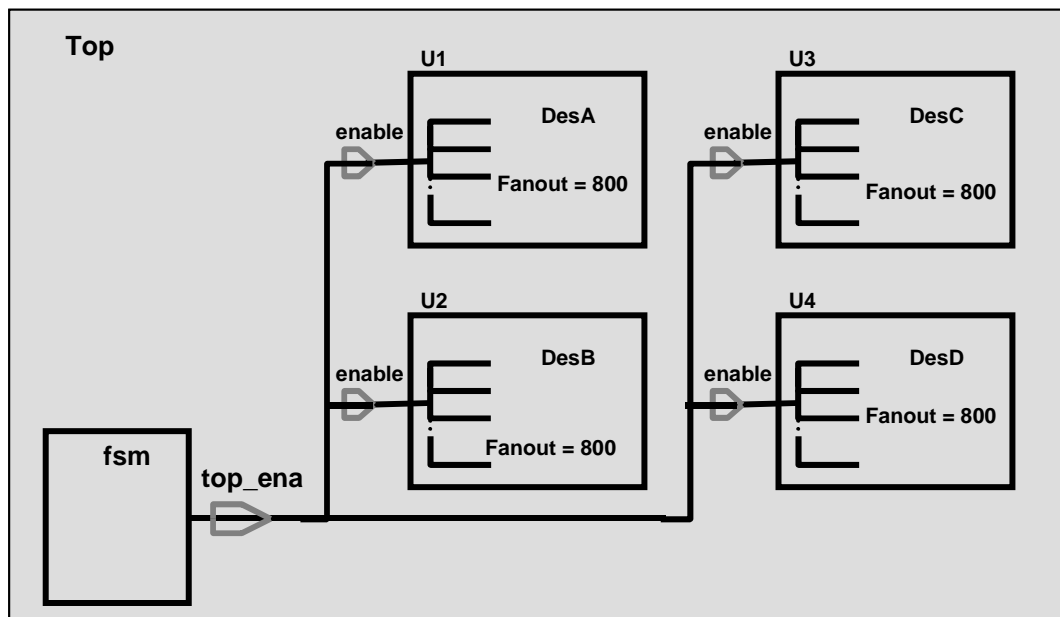


Figure 6
High Fanout Net Caused by Integration

8.1 Load Budgeting

One way to prevent slow transition times on the `top_ena` net is to employ *load budgeting* during bottom-up compiles. In other words, when DesA – DesD are being compiled, we want to limit the load on the inputs by setting a `max_capacitance` (or `max_fanout` or `max_transition`) on the inputs. This will force DC to “buffer up” higher-fanout inputs during compile.

An example script for compiling the lower designs would look like:

```
/* Bottom-level compile script for DC */
/* Will force the enable pin to be buffered during compile */

DRIVE_CELL = BUF1A
DRIVE_PIN = A
DRIVE_CELL_LIB = tech

DRV_CELL = DRIVE_CELL_LIB + "/" + DRIVE_CELL + "/" + DRIVE_PIN

/* Budget the input capacitance of each pin */
get_attribute find(pin, DRV_CELL) max_capacitance
max_input_cap = dc_shell_status
if (max_input_cap == {})
    max_input_cap = get_attribute DRIVE_CELL_LIB default_max_capacitance

max_input_cap = max_input_cap/4

include otherConstraints.scr
set_driving_cell -lib_cell DRIVE_CELL -pin DRIVE_PIN \
    -library DRIVE_CELL_LIB all_inputs() - clk
set_max_capacitance max_input_cap all_inputs() - clk

compile

. . .
```

The above script will result in cells U1-U4 having a buffer tree on their enable input, thereby presenting a lesser load to the finite state machine (FSM) generating `top_ena`

After top-level integration, `clean_buffer_tree` and then `balance_buffer` could be applied to net `enable_top` if necessary, in order to achieve a more desirable balanced tree.

8.2 Avoiding High Fanout Nets in a Configurable Processor

Most processors have several high-fanout signals, including instruction decodes, pipeline status, and, of course, clock and reset [6]. One challenge faced by Tensilica when designing the Xtensa configurable processor is that each user can configure the processor differently, which means that the fanout of these nets is not a constant. This can be further explained by noting the block diagram of the Xtensa processor in Figure 7 below:

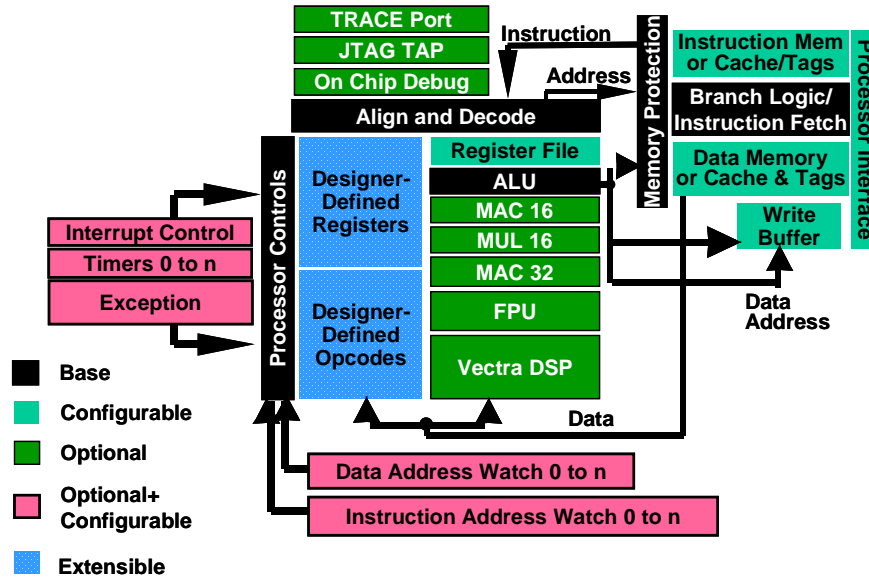


Figure 7
Tensilica Xtensa Block Diagram

As can be seen from Figure 7 above, besides any existing high fanout signals in the base Xtensa core, an Xtensa licensee has the opportunity to increase the fanout of the processor control/status signals, or the clock and reset line, by configuring additional blocks into the processor, including (but not limited to):

- i) A larger register file;
- ii) An IEEE compatible FPU (and its associated registers)
- iii) A Quad-MAC DSP with 40-bit accumulate (and its associated registers)
- iv) On-chip Address and Data Breakpoint register (for JTAG_ICE functionality)
- v) User defined processor registers and register files (any number, up to 128 bits wide)

Since Xtensa users have a great deal of control over the instruction-set architecture (ISA), and the processor is synthesizable to any ASIC library, identifying and buffering high fanout nets prior to synthesis is not possible. Instead, Tensilica provides customers with a “bottom up”, make-based synthesis scripting environment. These scripts use load budgeting and splitting in order to ensure there is no problems in top-level synthesis.

9 Conclusions and Recommendations

I believe that the following conclusions can be drawn based on the experiments conducted for this paper:

Clock nets:

Clock nets will pass ‘cleanly’ through Design Compiler, as long as create_clock has been issued. This is true whether or not

- a) *The clock is gated.*
- b) *set_dont_touch_network has been issued*

Reset nets:

Reset nets will pass ‘cleanly’ through Design Compiler, as long as set_ideal_net and set_dont_touch have been applied to the net prior to compile. To avoid inverters on the reset net, keep the ‘sense’ of the reset line consistent for all modules.

Other high-fanout nets:

If the signals traversing the net only have simple ‘clock-cycle’ timing constraints, DC will do a reasonable job of buffering and balancing the net.

If a low-skew, balanced net is desired, the net should be passed ‘clean’ through synthesis, with as little impact on synthesis runtime as possible. To do this, issue the commands:

:

```
/* Tell DC not to buffer the net */
set_dont_touch find(net, <net_name>)

set high_fanout_net_threshold 200 /* Or some other low number */

/* next line is optional, because high_fanout_net_threshold
   can make the high-fanout net a "non-problem", DRC-wise
set_ideal_net <net_name>
*/

compile
. . .
```

10 Enhancements and Areas for Further Study

Admittedly, the test circuit used for analysis throughout this paper was not very complex. Testing these new high fanout net commands on a more robust circuit would be beneficial. Also, no methodology yet exists for applying the high_fanout_net_threshold to individual nets. Testing the conclusions drawn above on a more complex circuit would be beneficial.

A necessary enhancement for a robust high fanout net methodology would be to make the high_fanout_net_threshold an attribute that could be applied on an individual net. This is because a likely scenario is that some nets that are to be fixed with clock-tree-like methods, and they should “inherit” the high_fanout_net_threshold, so that pre- and post- clock tree insertion timing values are closer. Other nets (which are not candidates for back-end buffer tree insertion) would not want to inherit this value, since DC will be responsible for buffering the *entire* net fanout.

11 Acknowledgements

Many thanks to Paul Stein of the Department of Defense, for taking the time to review this paper and provide constructive feedback. Also, thanks to JoAnne Wegener for her infinite patience!

12 References

- [1] Conistis, Furtner, & Islam, "Static Timing Analysis in Design Compiler and PrimeTime"; BSNUG 2000 Tutorial
- [2] Design Compiler User's Guide
- [3] Synthesis-749.html, "Preventing Buffers on RESET Line", Solv-Net Article
- [4] Synthesis-835.html, "Reducing Runtime Impact of High-fanout Nets in Design Compiler", Solv-Net Article
- [5] Synthesis-780.html; "Commands for high fanout nets", Solv-Net Article
- [6] Computer Architecture : A Quantitative Approach, by John Hennessy and David Patterson

12 Appendix

Verilog listing for circuit of Figure 4

```
module tst(clk, clk_G, d_in, s_r1, s_r2, rst_N1, rst_N2,
           dout);

parameter size = 1100;

input d_in, rst_N1, rst_N2, s_r1, s_r2, clk_G, clk;

output dout;
reg dout;

reg [size-1:1] tmp;
wire G_clk, rst_N, s_r;
integer i;

assign G_clk      = clk & clk_G;
assign rst_N      = rst_N1 & rst_N2;
assign s_r        = s_r1 & s_r2;

always @(posedge G_clk or negedge rst_N) begin

    if (!rst_N) begin
        dout <= 0;
        tmp <= 0;

    end else begin
        dout <= tmp[size-1] | s_r;

        for (i=size-1; i>1; i=i-1)
            tmp[i] <= tmp[i-1] | s_r;

        tmp[1] <= d_in | s_r;

    end
end

endmodule
```