Constraining Virtex Design in 2.1i

XAPP400 (v1.0) October 1, 1999

Application Note

Summary/ Introduction

Constraining a Virtex Design is different in 2.1i compared to older versions of the software. There are improvements in the Trace, Timing Analyzer, FloorPlanner, Constraints Editor, and other implementation tools to help make the designing procedure easier for Virtex. This paper is devoted to describing some of the simple steps necessary to constraining a Virtex design with the new 2.1i implementation tools. The major focus of this paper is to explain how to constrain with a CLKDLL in Virtex and the new look of the Timing Analyzer Reports.

Creating a UCF File for the Design

The Constraints Editor is a very useful tool that helps the user to constraint the design without knowing the User Constraints File (UCF) syntax. The Constraints Editor inputs are the design.ngd file and, if it exists, a UCF file. The ngd file is created by the Translate/NGDBUILD step of implementation tools. There are three ways to invoke the Constraint Editor. The first is selecting the Utilities -> Constraints Editor menu item, when a version or revision is selected and the Flow Engine runs Translation automatically, and brings up the Constraints Editor. The second is to run the Translate Step, then open the Constraints Editor by selecting Utilities -> Constraints Editor. The third option it to use the command line mode and run ngdbuild then constraints_editor.

There are several ways to constraint in a Virtex design. Timing constraints are used to get the performance that the designer requires. Location constraints are used to lockdown the IOBs. If the Floorplanner is used, all location constraints in the UCF file are overwritten.

Other items can be constrained such as slew rate, whether to use IOB/CLB locations and so forth. These have not been changed in 2.1i, and will not be covered in this paper. Please see the online documentation of the *Constraints Editor User Guide* (http://toolbox.xilinx.com/docsan/2_1i/data/alliance/cst/cst.htm) for more information.

The CLKDLL provides many powerful features that aid in the design of high-speed digital circuits. Associated with these features is the ability of the Xilinx tool set to perform timing based place and route. The new processing for TNMs, TNM_NET, PERIOD, OFFSET, FROM TO constraints, the Low Skew Routing Resources information, the MAXSKEW constraint, and the Priority of Constraints will be illustrated in detail.

CLKDLL & TNM Processing

The rules regarding property tracing through the CLKDLL have changed in 2.1i. When a TNM_NET property is traced into the CLKIN pin of a Virtex CLKDLL component, the TNM group and its usage will be examined. The TNM will be pushed through the CLKDLL only if the following conditions are met:

1. The TNM group name is used in exactly one PERIOD specification.
2. The TNM group name is not used in any FROM-TO or OFFSET specifications.
3. The TNM group name is not referenced in any user group definition.

If any of the above conditions are not met, the TNM will not be pushed through the CLKDLL, and the following error message will be issued:

```
ERROR:NgdHelpers:702 - The TNM “PAD_CLK” drives the CLKIN pin of CLKDLL
“$I1”.
```

This TNM cannot be traced through the CLKDLL because it is not used in exactly one PERIOD specification. This TNM is used in the following user groups and/or specifications:

```
TS_PAD_CLK=PERIOD PAD_CLK 20.0 ns HIGH 50.000000%
TS_01=FROM PAD_CLK TO PADS 20.0 ns
```

If the above conditions are met each clock output pin on the CLKDLL will be examined to see if it is connected to a net with at least one other connection (i.e., it is not a dangling net). If the output pin has a net, a new TNM group will be created on that net, and a new PERIOD specification will be created for that group. The new specification will be copied from the original PERIOD specification, and then modified as shown in Table 1.

Table 1: Period Modifications

Output Pin	Modifications to PERIOD Specification
CLK90 CLK180 CLK270	CLK0 If the DUTY_CYCLE_CORRECTION=TRUE property is found on or above the CLKDLL, the duty cycle will be adjusted to 50%. If DUTY_CYCLE_CORRECTION=FALSE is found, the duty cycle will be unchanged from the original PERIOD specification. If no DUTY_CYCLE_CORRECTION property is found, the default value of TRUE will be assumed.
CLK2X	The PERIOD value will be doubled (if originally expressed as a frequency) or divided in half (if originally expressed as a delay). The duty cycle will also be adjusted to 50%.
CLKDV	The PERIOD value will be divided (if a frequency) or multiplied (if a delay) by the value in the CLKDV_DIVIDE property. If no such property is found on or above the CLKDLL, the default value of 2.0 will be used. The duty cycle will also be adjusted to 50%.

If the original TNM_NET property is pushed only into the CLKDLL CLKIN pin (i.e., it does not trace to any appropriate elements without going through the CLKDLL), the original TNM group and the original PERIOD specification will be eliminated from the design. If a newly-created TNM group is pushed through a CLKDLL output and encounters the CLKIN input of a second CLKDLL (such as in the 4X configuration), the above process will be repeated to further adjust the PERIOD specification(s) per the behavior of the second CLKDLL. If the group created for the first CLKDLL traces only into the second CLKDLL, that group and its PERIOD specification become unnecessary and will be eliminated.

For further information regarding property tracing through the CLKDLL refer to the Developmental System Reference Guide Chapter 6.

CLKDLL PERIOD Example

A PERIOD constraint can be applied to the CLKDLL in Figure 1, and the following PERIOD constraint will be pushed through to the outputs of the CLKDLL. The Constraint Editor creates PERIOD constraints based upon the CLKDLL input name. Care still needs to be taken with respect to duty cycle correction, multiplication, and division.

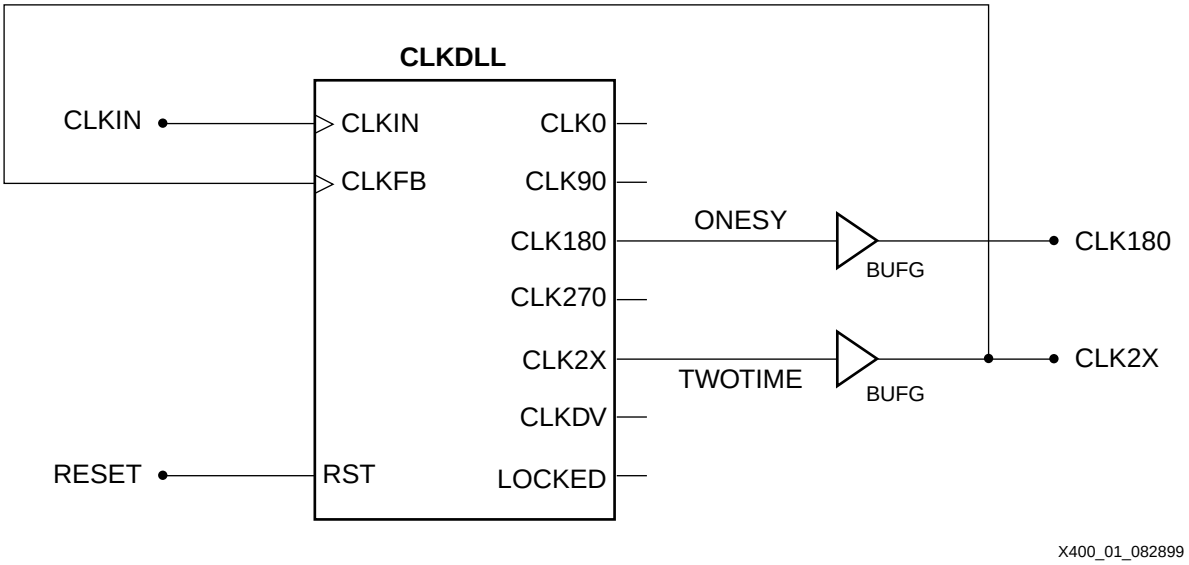

Figure 1: CLKDLL Implementation

An example:

```
NET “CLKIN” TNM_NET = “CLKIN”;
TIMESPEC “TS_CLKIN” = PERIOD “CLKIN” 9 ns HIGH
50%;
```

When NGDBUILD/Translate sees the previous example in the UCF file, it will produce the following message:

```
INFO:NgdHelpers - TNM "CLKIN", used in period specification
"TS_CLKIN", was traced into CLKDLL instance "$I1". The following new
TNM groups and period specifications were generated at the CLKDLL
output(s). TNM "CLKIN" and
specification "TS_CLKIN" are no longer needed and have been removed
from the design.

TS_ONESY=PERIOD ONESY 9.0 nS HIGH 50.000000%

TS_TWOTIME=PERIOD TWOTIME 4.5 nS HIGH 50.000000%
```

After implementing the design, the timing report shows the two new timespecs that were created in NGDBUILD/Translate, which are shown below. The pushing of the PERIOD constraint occurs in the Timing Engine and is reported in the Trace Report (See Figure 2).

```
Timing constraint: TS_ONESY = PERIOD TIMEGRP "ONESY" 9 ns HIGH 50.000%;
32 items analyzed, 0 timing errors detected.
Minimum period is 9.869 ns.
```

```
Timing constraint:TS_TWOTIME = PERIOD TIMEGRP "TWOTIME" 4.5 ns HIGH 50.000%;
1 item analyzed, 0 timing errors detected.
Minimum period is 4.384 ns.
```

Figure 2: Trace/Timing Report

So, the PERIOD constraint on the CLKIN will be pushed to be applied to net ONESY and TWOTIME, as seen in Figure 1. The action of pushing the Period constraint forward through the DLL will not happen if the TNM Path Tracing Control conditions are not met.

Note: Timing Analyzer and Trace will be modified to no longer check pad-to-register paths relative to setup requirements or period and frequency constraints. The general OFFSET IN will be need to cover this path. The OFFSET in/after and OFFSET out/before constraints will need the period constraint to calculate the appropriate offset

PAD to SETUP (OFFSET IN BEFORE)

When creating pad-to-setup requirements, care should be taken to incorporate any phase or PERIOD adjustment factor into the value specified for an OFFSET IN constraint. When creating pad or register specific OFFSET constraints in the Constraints Editor the user must specify the clock pad net name for flip-flops driven by the CLKDLL. In Figure 3, PADCLKIN is the pad net name used in OFFSET constraints.

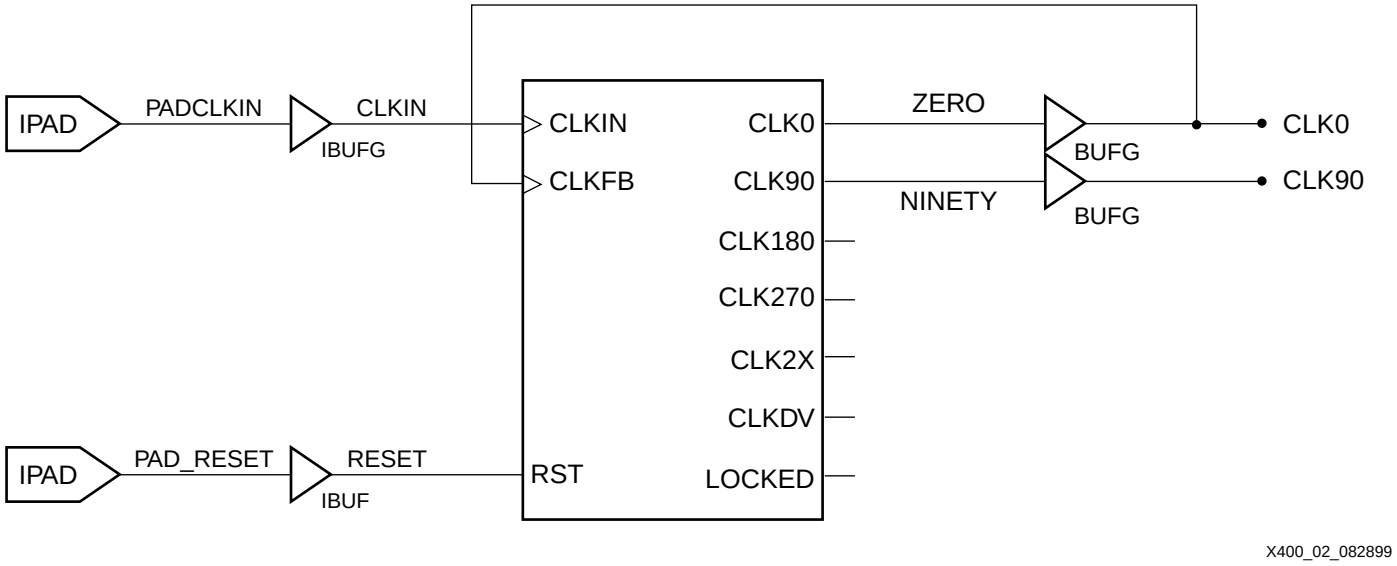

Figure 3: CLKDLL Implementation

For example: If your register is clocked by the net from the CLK90 pin of the CLKDLL which has a PERIOD of 20 ns, then the OFFSET value should be adjusted by an additional 5 ns as seen in Table 2 for Figure 3. The equations is to add 75% of the Period constraint value from the original constraint, to give it 15 ns = 10 ns + (0.75 x 20 ns). If the original constraint was with respect to the CLK0 pin of the CLKDLL, then one would subtract the 75% of the Period constraint value.

Original Constraint: `NET “PAD_IN” OFFSET = IN 10 BEFORE “PADCLKIN”;`
Modified Constraint: `NET “PAD_IN” OFFSET = IN 15 BEFORE “PADCLKIN”;`

Note: The clock net name required for OFFSET constraints is the clock net name attached to the IPAD. In this case it is PADCLKIN not CLK90, refer to Figure 4 for the clk network.

CLOCK to PAD (OFFSET OUT AFTER)

When creating clock-to-pad requirements, care should be taken to incorporate any phase or PERIOD adjustment factor into the value specified for an OFFSET OUT constraint. In the Constraints Editor the user must specify the clock pad net name for flip-flops driven by the CLKDLL when specifying a specific OFFSET constraint. In Figure 4, PAD CLKIN is the pad net name used in OFFSET constraints.

For example: If a register is clocked by the net from the CLK180 pin of the CLKDLL which has a PERIOD of 20 ns, Then the OFFSET value should be adjusted by 10 ns less than the original constraint as shown in Table 2 for Figure 3. The equations is to subtract 50% of the Period constraint value from the Original constraint, to give it 5 ns = 15 ns - (0.5 x 20 ns).

Original Constraint: `NET “PAD_OUT” OFFSET = OUT 15 BEFORE “PADCLKIN”;`
Modified Constraint: `NET “PAD_IN” OFFSET = OUT 5 BEFORE “PADCLKIN”;`

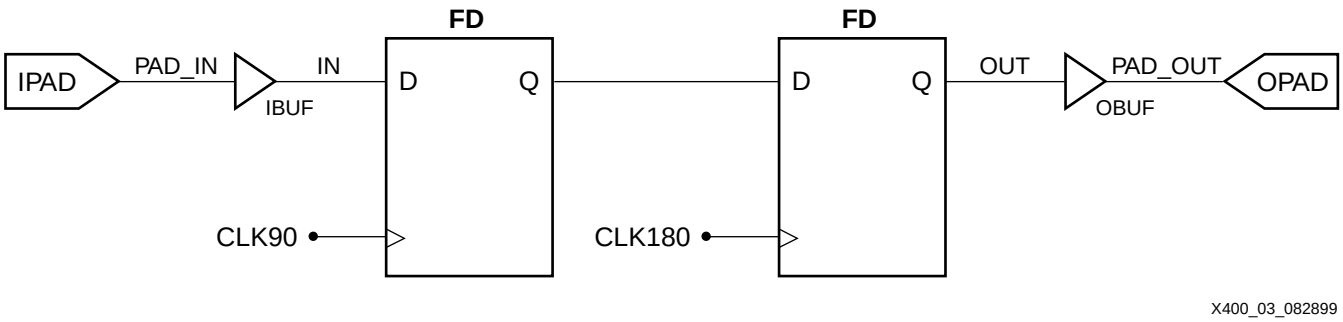

Figure 4: Schematic Layout

FROM TO In Multiple Clock Domains

When using PERIOD constraints, the user must properly constrain the paths between multiple clock domains as seen in Figure 5. If a PERIOD constraint is applied to the CLK90 and CLK0 pins of the CLKDLL, the CLK0 PERIOD constraint will constrain the paths between flip-flop A and B. This type of constraining can lead to setup violations.

Consider the waveforms in Figure 6. Because of the phase shift between CLK90 and CLK0, the path from A to B has 25% less time than the PERIOD constraint allows. To properly constrain these paths use a FROM TO constraint. For example:

```
NET “CLK90” TNM_NET = “CLK90”;
NET “CLK0” TNM_NET = “CLK0”;
TIMESPEC “TS_CLK90_2_CLK0” = FROM “CLK90” TO “CLK0” 15ns;
```

The TIMESPEC is set at 15 ns because the PERIOD on CLK0 is 20 ns.

This type of correction is required for all clock domain interactions. Table 2 contains some of the possible configurations and the associated corrections.

Table 2: Correct Percentage for Different Phase Clocks

CLKDLL	Required Correction
CLK0 to CLK90	Subtract 75% of Period from Original Offset
CLK0 to CLK180	Subtract 50% of Period from Original Offset
CLK0 to CLK270	Subtract 25% of Period from Original Offset
CLK270 to CLK0	Subtract 75% of Period from Original Offset
CLK270 to CLK90	Subtract 50% of Period from Original Offset
CLK270 to CLK180	Subtract 25% of Period from Original Offset

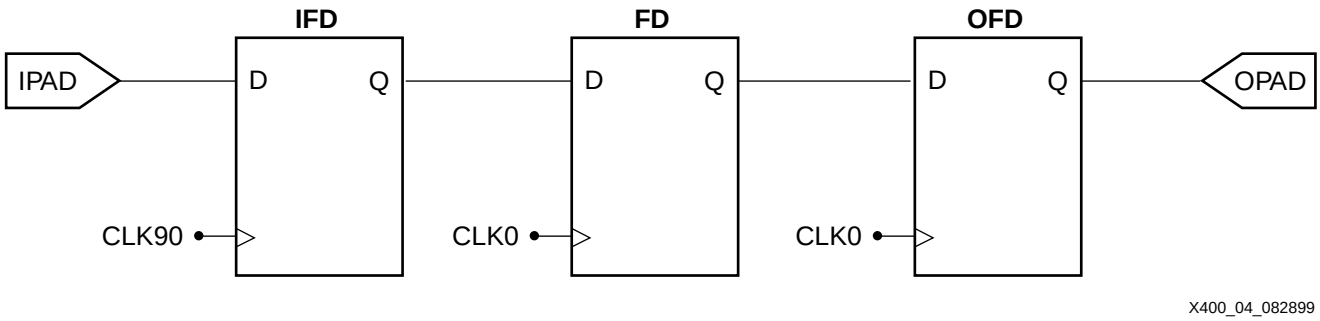

Figure 5: FROM TO Schematic

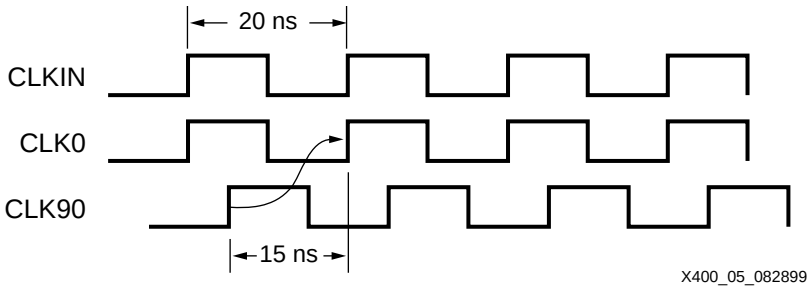

Figure 6: Waveform Illustration of CLKIN, CLK0, and CLK90

Low Skew Routing in Virtex

The Virtex architectures have 24 horizontal long lines for use as low skew routing resources. There are 12 low skew routing resources on the top edge and another 12 on the bottom edge of the device. All the IOBs along the top and bottom edges can get on to the low skew routing resources directly. Figure 7 shows the low skew routing resources on the top edge of the device. Since the top and bottom edges are similar, one will refer to the top edge only in the discussion below unless noted otherwise.

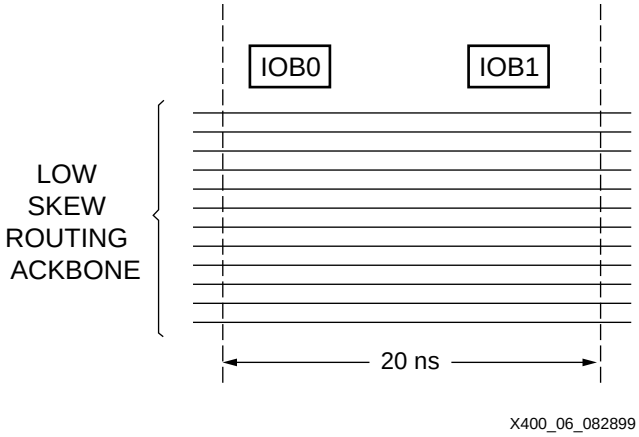

Figure 7: Low Skew Routing Resources

From every column, there are connections from the IOB switch box (Figure 8) to two of 12 low skew routing resources. The specific low skew routings change varies per column but every sixth column always connects to the same low skew routing resources. This places the restriction that one cannot use more than two low skew routings for all columns that are apart from each other by six rows. The same two low skew routings individually drive six hexes each in the top tile for a total of 12 hexes per tile column being driven by the low skew routings. Each of these hexes then connects to vertical long lines and onwards to other pins. The hierarchy of routing resources used to connect to different pins is shown in the Figure 8.

As can be seen in Figure 8, a net that uses the low skew routings and connects to pins of different types will use different resources and hence will have different delays. Connecting to the block RAMs is the fastest while connecting to the general inputs of a slice is the slowest. An important component of the skew comes from the horizontal span of the low skew routing that is used. In other words, by limiting the distance between the left most vertical long line and the right most vertical long line (horizontal span) the skew can be reduced considerably. Further, by keeping the loads as close as possible to the low skew routing the skew can be limited

A similar routing hierarchy is present when an internally generated net is required to use the low skew routing. While all slices can reach the low skew routings through the general routing, for the purpose of reducing delay it is essential to use only the top and bottom row of slices if they are the sources to nets that require the low skew routings.

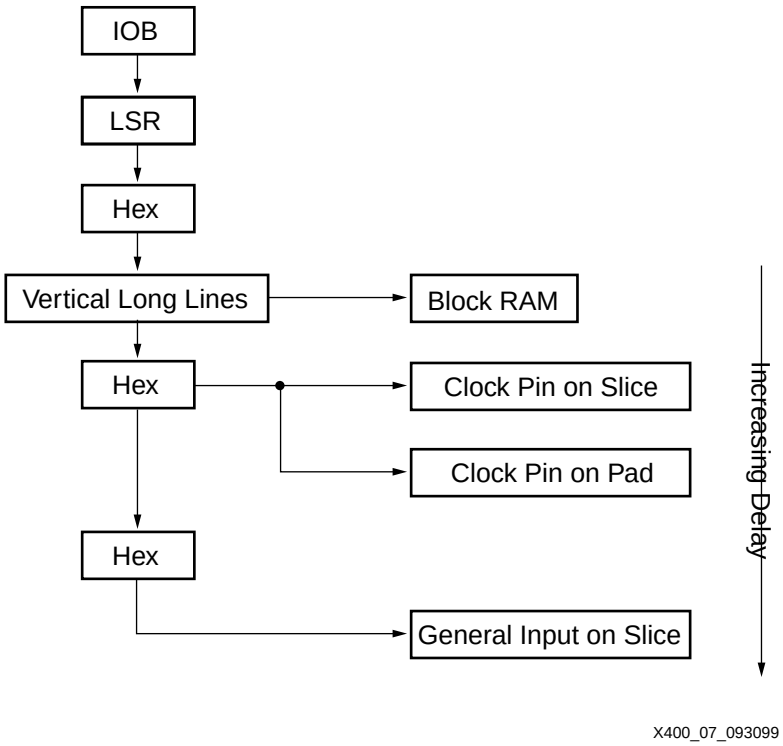

Figure 8: Routing Resource Hierarchy

Currently, there is no way to properly place slices that source nets requiring low skew routings. These slices ought to be automatically placed in the top and bottom rows, adjacent to the low skew routings. However, in 2.1i, the user has to explicitly either lock the slices down to the top or bottom rows or put range constraints on the slices. Xilinx is currently in the process of developing solutions for the two issues mentioned above.

MAXSKEW for Low Skew Lines

The nets that are sourced by IOBs and connect either clock pins or control pins such as CE, RST, etc. are considered for special placement rules. Other candidates for these special placement rules are nets that are tagged with a MAXSKEW constraint or very high fanout nets that are sourced by IOBs. These IOB sources of these nets are placed in the top and bottom edges of the device. Once these are placed in the top and bottom edges, the router routes these nets using low skew routings in the preassignment phase. Care is taken in the initial IOB placement to ensure that no more than two low skew routings are used in all the columns that differ from each other in multiples of six.

The MAXSKEW constraint is used to tell AR to use the low skew lines for a net. The MAXSKEW value does not have to be a tight value. An example is as follows:

```
NET “net_needs_low_skew” MAXSKEW=12ns;
```

Constraint Priority

Timing constraints have a priority system for when a constraint is covering the same path. The following priority is from highest priority to the lowest priority with in a particular source.

Table 3: Timing Constraints Priority

Priority	Timing Constraints
Highest	TIG (Timing Ignores)
	FROM:THRU:TO spec • Source & Destination defined by User • Source or Destination defined by User • Source & Destination are Pre-Defined Groups
	FROM:TO spec • Source & Destination defined by User • Source or Destination defined by User • Source & Destination are Pre-Defined Groups
	OFFSET spec • Specific Data IOB • Time Group of Data IOBs • For all Data IOBs
	PERIOD specs
Lowest	ALLPATHS specs (`.pcf` only)

Timing Report Review

The timing reports are created when either Trace is run after AR or Timing Analyzer is run. Both Timing Analyzer and Trace can produce similar reports, depending on what kind of report is needed. In reviewing the reports, it is important to know if your constraints were met or not and to know if more constraints are needed or not. In this section the new look of the Timing Analyzer GUI will be discussed, along with a portion of the Timing Report, the datasheet IO section, answer the question: what has happened when a constraint has zero items analyzed, and why is the coverage less than 100%.

Timing Reports in Timing Analyzer

Timing Analyzer displays FPGA and CPLD analysis reports in a hierarchical format. The report window (used to display all `.twr` files) now has three panes, as seen below. The lower-right pane is the text view. It displays the text of the.twr file. The left pane is the index or outline view. Click on the labels to scroll the corresponding line in the report to the top of the text view. Click on the "+" or "-" buttons to expand or collapse topics. If there are timing errors, the text will appear in red. The other standard mouse and keyboard manipulations for tree views work, too. The upper-right pane is the context or path view. It shows the path through the topic hierarchy to the item currently selected in the index view. The contents of the context view change only when the selection in the index view changes. Click on items in the context view to scroll the text view to the appropriate place. This can be handy if you are deep in a list of paths for a timing constraint and need to jump back to the timing constraint definition (see Figure 9).

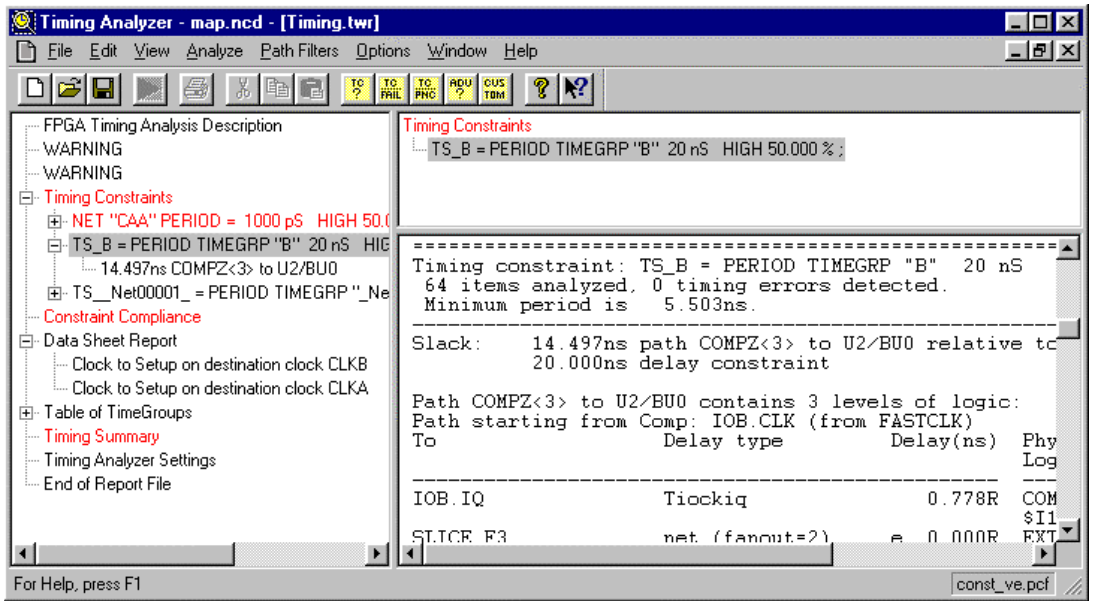

Figure 9: Timing Analyzer

Datasheet IO Section of the Timing Report

The Data Sheet IO Timing Report contains the setup/hold and clock to pad times for all synchronous inputs and outputs with respect to the appropriate clock edge, it also contains the pad-to-pad paths and a clock-to-setup table.

```
Setup/Hold to clock CLK2 example
+---------------+-----------+-----------+
|               | Setup to  | Hold to   |
|Source Pad     | clk (edge)| clk (edge)|
+---------------+-----------+-----------+
|INA<1>         |    1.058(R)|    0.000(R)|
|INA<0>         |    1.847(R)|    0.000(R)|
|INA<3>         |    1.220(R)|    0.000(R)|
|INA<2>         |    1.578(R)|    0.000(R)|
|INA<5>         |    1.152(R)|    0.000(R)|
|INA<4>         |    1.700(R)|    0.000(R)|
|INA<7>         |    1.516(R)|    0.000(R)|
|INA<6>         |    1.850(R)|    0.000(R)|
+---------------+-----------+-----------+
Clock CLK1 to Pad example
+---------------+-----------+
|               | clk (edge)|
|Destination Pad|   to PAD  |
+---------------+-----------+
|OUT<0>         |    8.086(R)|
|OUT<1>         |    8.164(R)|
|OUT<2>         |    8.288(R)|
|OUT<3>         |    8.335(R)|
|OUT<4>         |    8.034(R)|
|OUT<5>         |    8.008(R)|
|OUT<6>         |    7.824(R)|
|OUT<7>         |    8.486(R)|
```

The Data Sheet IO Timing Report is included in the Post Layout Timing Report, Timing Analyzer Reports, and Trace Reports when any of the following criteria are met.

1. Designs containing timing constraints that constrain the input and outputs. For example: OFFSET IN, OFFSET OUT, PADS to FFS, and FFS to PADS. Also a Pad-to-Pad and PERIOD constraints are needed to present all data possible.
2. Unconstrained timing analysis is run in Trace or Timing Analyzer.
3. Advanced timing analysis is run in Trace or Timing Analyzer.

Zero Items Analyzed

There are many reasons for a constraint to be reported as having zero items analyzed. The most common reason is that two constraints cover the same paths. After all paths are accounted for under the constraints in the PCF, the Timing Wizard will determine which paths are covered by more than one constraint. By comparing the relative scopes of each constraint a path is left in a single constraint. After all paths and constraints have been examined in this way it is common for some constraints to end up with no items to analyze. This extraction process allows the tool to analyze slow exceptions as well as reduce the run times by only performing the analysis a single time per constraint where possible.

Be aware also that if there are two or more constraints, which define the same set of paths, the one which appears last in the PCF is the one that will get analyzed, regardless of the value placed on the constraint. One may force a constraint appearing earlier in the PCF to be analyzed by applying a priority to the desired constraint.

Another common reason that a constraint may have no items analyzed is that TIG constraints have been placed on elements in every path or on each path which might alternatively be covered by the constraint.

It is also possible that all the paths which you may expect to be covered by the constraint include a component delay path which is subject to path tracing controls.

Coverage is Less Than 100%

The coverage statistic is not a measure of the percentage of paths covered by the constraints, it is a measure of the percentage of total connections in the design covered by the constraints. Thus a design can have all valid paths covered by constraints, but still have a coverage statistic that isn't 100%. Xilinx is aware of this confusion and has been considering alternatives to the current report, but there are no plans to change it in the near future.

The most common reason for connection coverage not hitting 100% is that elements in the design have TIGs. If the Trace/Timing Analyzer encounters a TIGged element when tracing a path, the trace will stop there, possibly leaving connections on the "other side" of the element uncovered. A TIG on a *path,* on the other hand will have all of its connections accounted for in the coverage statistic.

There are less obvious reasons for less than 100% coverage. One is that the total number of connections in a design includes some which cannot be covered by constraints. An example is the connections on the STARTUP component. Another example is the case where a static pin drives a LUT, like a logical one or zero, which combines with no other signals and then drives other logic. This can happen at the start of a carry chain where a FORCE mode is used from a logical "1" or "0". Also if terms for carry logic are connected to a CLB, but go unused within the CLB, these connections will never be traced. These are just obscure cases that are not handled.

If the coverage is less than 100%, the user can run an Unconstrained Paths Report from either the command line or in the Timing Analyzer GUI. The command line is `'trce -u design.ncd design.pcf'` and the Timing Analyzer GUI button is under the Analyze Menu, then Report Paths Not Covered by Timing Constraints. This report will give the user the maximum percentage of coverage for that design. So, if the initial constraints gives coverage of 82%, and the Unconstrained Paths Report gives coverage of 91%, then the maximum percentage of coverage is 91%. The user then can add more constraints to get the coverage up to 91%, but the coverage will never go above that percentage for that version of the design.

Revision History

Date	Version	Revision
10.01.99	1.0	Initial Xilinx release.