Chapter 4

Using Timing Constraints

The timing constraints described in this chapter are compatible with the following families.

- XC3000A / L
- XC3100A / L
- XC4000E / L
- XC4000EX / XL / XLA / XV
- XC5200
- Virtex
- Spartan
- SpartanXL

This chapter describes how you specify timing constraints, and contains the following sections.

- “Timing Requirements and Xilinx Software”
- “Entering Timing Specifications”
- “Specifying Groups”
- “Defining a Clock Period (PERIOD Constraint)”
- “OFFSET Timing Specifications”
- “Ignoring Selected Paths (TIG)”
- “Basic FROM –TO Syntax”
- “Specifying Timing Points”
- “Using TPTHRU or TPSYNC in a FROM–TO Constraint”
- “Specifying Time Delay in TS Attributes”

- "Using the PRIORITY Keyword"
- "Sample Schematic Using TIMESPEC/TIMEGRP Symbol"
- "Prorating Constraints"
- "Additional Timing Constraints"
- "Constraints Priority"
- "Syntax Summary"

Timing Requirements and Xilinx Software

Xilinx software enables you to specify precise timing requirements for your Xilinx FPGA designs. You can specify the timing requirements for any nets or paths in your design. One way of specifying path requirements is to first identify a set of paths by identifying a group of start and end points. The start and end points can be flip-flops, I/O pads, latches, or RAMs. You can then control the worst-case timing on the set of paths by specifying a single delay requirement for all paths in the set.

The primary method of specifying timing requirements is by entering them on the schematic. However, you can also specify timing requirements in constraints files (UCF and PCF). For detailed information about the constraints you can use with your schematic-entry software, refer to the "Attributes, Constraints, and Carry Logic" chapter of the *Libraries Guide*.

Once you define timing specifications and then map the design, PAR places and routes your design based on these requirements.

To analyze the results of your timing specifications, use TRACE (Timing Report and Circuit Evaluator). Refer to the "TRACE" chapter for more information.

Entering Timing Specifications

This section describes the basic methods for entering timing specifications in a schematic or User Constraints File (UCF).

The following notes apply to Mentor Graphics users.

- The term *attribute* in this chapter is equivalent to *property* as used in the Mentor Graphics environment.

- The Mentor netlist writer (ENWRITE) converts all property names to lowercase letters, and the Xilinx netlist reader EDIF2NGD then converts the property names to uppercase letters. Because property names are processed in this way, you must enter variable text in certain constraints in upper case letters only. This requirement is discussed in the following sections.
  - "Entering Timing Specifications in a Schematic"
  - "Creating New Groups from Existing Groups"

Entering Timing Specifications in a Schematic

The TIMESPEC schematic primitive, as illustrated in the "TIMESPEC Primitive" figure, serves as a placeholder for timing specifications, which are called TS attribute definitions. Every TS attribute must be defined in a TIMESPEC primitive, and only TIMESPEC primitives can carry TS attribute definitions. Every TS attribute begins with the letters "TS" and ends with a unique identifier that can consist of letters, numbers, or the underscore character (_).

TS attribute definitions can be any length, but only 30 characters are displayed in the TIMESPEC window. Each TIMESPEC primitive can hold up to eight TS attributes. If you want to include more than eight TS attributes, you can use multiple TIMESPEC primitives in your schematic.

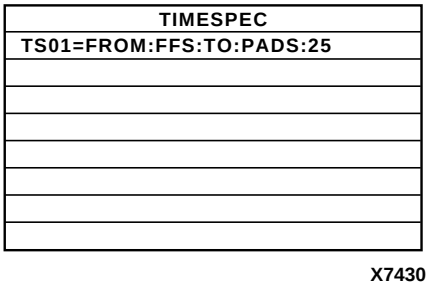


Figure 4-1 TIMESPEC Primitive

How you add a TIMESPEC primitive to your schematic depends on your specific schematic-entry software. Refer to the appropriate Xilinx *Interface User Guide* for step-by-step instructions.

A TS attribute defines the allowable delay for paths in your design. The basic syntax for a TS attribute is as follows.

TS*identifier***=FROM** *source_group* **TO** *dest_group delay*

TS*identifier* is a unique name for the TS attribute, *source_group* and *dest_group* are groups of start points and end points, and *delay* defines the maximum delay for the paths between the start points and end points. The *delay* parameter defines the maximum delay for the attribute. Nanoseconds are the default units for specifying delay time in TS attributes. You can also specify delay using other units, such as picoseconds or megahertz.

Note: Keywords, such as FROM, TO, and TS appear in the documentation in upper case; however, you can enter them in the TIMESPEC primitive in either upper or lower case. The characters in the keywords must be all upper case or all lower case. Examples of acceptable keywords are: FROM, TO, from, to. Examples of unacceptable keywords are: From, To, fRoM, tO.

Note: The Mentor netlist writer (ENWRITE) converts all property names to lower case letters, and the Xilinx netlist reader EDIF2NGD then converts the property names to upper case letters. To ensure references from one constraint to another are processed correctly, a **TS***identifier* name should contain only upper case letters on a Mentor Schematic (TSMAIN, for example, but not TSmain or TSMain). Also, if a **TS***identifier* name is referenced in a property value, it must be entered in upper case letters. For example, the TSID1 in the second constraint below must be entered in upper case letters to match the TSID1 name in the first constraint.

```
TSID1 = FROM gr1 TO gr2 50;
TSMAIN = FROM here TO there TSID1 /2;
```

The basic TS attribute is described in detail in the “Basic FROM –TO Syntax” section. More detailed forms of the attribute are also described in that section.

Note: A colon may be used as a separator instead of a space in all timing specifications.

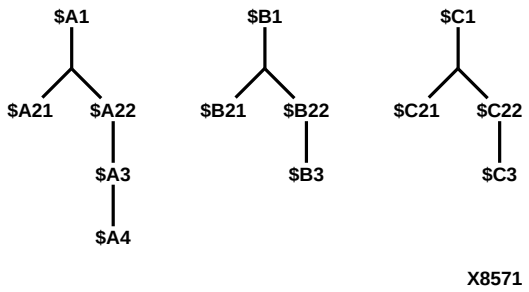
Entering Timing Specifications in a Constraints File

You can enter timing specifications as constraints in a UCF file. When you then run NGDBuild on your design, your timing specifications are added to the design database as part of the NGD file.

The basic syntax for a timing specification entered in a constraints file is the TS attribute syntax described in the “Basic FROM –TO Syntax” section.

Although not required, Xilinx recommends that NET and INST names be enclosed in double quotes to avoid errors. Additionally, inverted signal names that contain a tilde, for example, ~OUTSIG1, must always be enclosed in double quotes. Other special characters that must be enclosed in quotes are the asterisk (*) and question mark (?).

You can use the wildcard character (*) to traverse the hierarchy of a directory within a UCF or NCF file. Consider the following directory hierarchy.



With the example hierarchy, the following specifications illustrate the scope of the wildcard.

```
INST *          => <everything>
INST /*         => <everything>
INST /*/        => <$A1,$B1,$C1>
INST $A1/*      => <$A21,$A22,$A3,$A4>
INST $A1/*/     => <$A21,$A22>
INST $A1/*/*    => <$A3,$A4>
INST $A1/*/*/   => <$A3>
INST $A1/*/*/*  => <$A4>
INST $A1/*/*/*/ => <$A4>
INST /*/*22/    => <$A22,$B22,$C22>
```

```
INST /*/*22     =>
<$A22,$A3,$A4,$B22,$B3,$C22,$C3>
INST /*/*22/*   => <$A3,$A4,$B3,$C22,$C3>
```

Specifying Groups

In a TS attribute, you specify the set of paths to be analyzed by grouping start and end points in one of the following ways.

- Refer to a predefined group by specifying one of the corresponding keywords — FFS, PADS, LATCHES, or RAMS.
- Create your own groups within a predefined group by tagging symbols with TNM (pronounced tee-name) attributes.
- Create groups that are combinations of existing groups using TIMEGRP symbols.
- Create groups by pattern matching on net names.

The following sections discuss each method in detail.

Using Predefined Groups

You can refer to a group of flip-flops, input latches, pads, or RAMs by using the corresponding keywords.

Keyword	Description
FFS	CLB or IOB flip-flops only; not flip-flops built from function generators (Shift Register LUTs in Virtex also)
LATCHES	CLB or IOB latches only; not latches built from function generators
PADS	Input/Output pads
RAMS	For architectures with RAMS (LUT RAMS and Block RAMS for Virtex)

From-To statements enable you to define timing specifications for paths between predefined groups. The following examples are TS attributes that reside in the TIMESPEC primitive or are entered in the UCF. This method enables you to easily define default timing specifications for the design, as illustrated by the following examples.

Schematic syntax in TIMESPEC primitive

```
TS01=FROM FFS TO FFS 30
TS02=FROM LATCHES TO LATCHES 25
TS03=FROM PADS TO RAMS 70
TS04=FROM FFS TO PADS 55
```

UCF syntax

```
TIMESPEC TS01=FROM FFS TO FFS 30;
TIMESPEC TS02=FROM LATCHES TO LATCHES 25;
TIMESPEC TS03=FROM PADS TO RAMS 70;
TIMESPEC TS04=FROM FFS TO PADS 55;
```

A predefined group can also carry a name qualifier; the qualifier can appear any place where the predefined group is used. This name qualifier restricts the number of elements being referred to. The syntax used is as follows.

predefined group **(** *name_qualifier* [*name_qualifier*] **)**

name_qualifier is the full hierarchical name of the net that is sourced by the primitive being identified.

The name qualifier can include wildcard characters (*) to indicate any number of characters (or ? to indicate a single character) which allows the specification of more than one net or allows you to shorten the full hierarchical name to something that is easier to type.

As an example, specifying the group FFS(MACRO_A/Q?) selects only the flip-flops driving the Q0, Q1, Q2 and Q3 nets in the following macro.

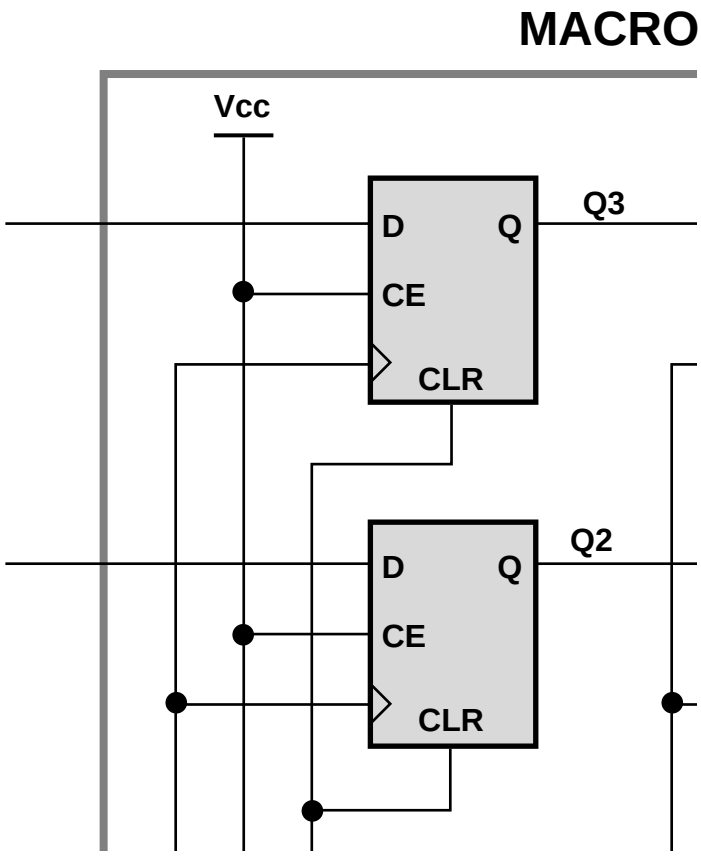


Figure 4-2 Using Qualifiers with Predefined Groups

To create more specific groups see the following section.

Creating User-Defined Groups Using TNMs

A TNM (timing name) is an attribute that can be used to identify the elements that make up a group which can then be used in a timing specification. A TNM is a property that you place directly on your schematic to tag a specific net, element pin, primitive or macro. All symbols tagged with the TNM identifier are considered a group. Place TNM attributes directly on your schematic or in a UCF file using the following syntax.

Schematic syntax

TNM=*identifier*

UCF syntax

{**NET** | **INST** | **PIN**} *object_name* **TNM=***identifier***;**

identifier is a value that consists of any combination of letters, numbers, or underscores. Keep the TNM short for convenience and clarity.

Do not use the reserved words FFS, LATCHES, PADS, RAMS, RISING, FALLING, TRANSHI, TRANSLO, or EXCEPT, as *identifier*s. The constraints in the table below are also reserved words and should not be used as *identifiers*.

Reserved Words (Constraints)		
ADD	FAST	NODELAY
ALU	FBKINV	OPT
ASSIGN	FILE	OSC
BEL	F_SET	RES
BLKNM	HBLKNM	RLOC
CAP	HU_SET	RLOC_ORIGIN
CLKDV_DIVIDE	H_SET	RLOC_RANGE
CLBNM	INIT	SCHNM
CMOS	INIT OX	SLOW
CYMODE	INTERNAL	STARTUP_WAIT
DECODE	IOB	SYSTEM
DEF	IOSTANDARD	TNM
DIVIDE1_BY	LIBVER	TRIM
DIVIDE2_BY	LOC	TS
DOUBLE	LOWPWR	TTL
DRIVE	MAP	TYPE
DUTY_CYCLE_ CORRECTION	MEDFAST	USE_RLOC
EQN	MEDSLOW	U_SET
FAST	MINIM	

Note: If you want to use a keyword as an identifier, you can enclose the keyword in quotation marks. In the TNM statement **TNM=RAMS "CMOS"**, CMOS is treated as an identifier instead of a keyword.

You can specify as many groups of end points as are necessary to describe the performance requirements of your design. However, to simplify the specification process and reduce the place and route time, use as few groups as possible.

A predefined group can be used in a TNM specification, using the following syntax on a schematic or UCF file.

Schematic syntax

TNM=*predefined_group identifier*

UCF syntax

{**NET** | **INST** | **PIN**} *object_name* **TNM=***predefined_group identifier***;**

The *object_name* is the net, pin, or instance name.

The *predefined_group* is one of the groups (for example, FFS or RAMS) defined in the "Using Predefined Groups" section and *identifier* is a value that consists of any combination of letters, numbers, or underscores. Paths defined by the TNM are traced forward if placed on a net or pin, through any number of gates or buffers, until they reach a member of the *predefined_group*. That element is added to the specified TNM group. TNM does not trace through the element to the next element; forward tracing stops at the element. This mechanism is called forward tracing. If TNM is placed on an instance, paths are traced "downward" through a hierarchy instead of forward along a net.

Note: If a TNM is placed on an input pad net, the constraint only applies to the input pad. In that case, refer to the "Creating User-Defined Groups Using TNM_NET" section.

The specification shown below, when attached to a net, would create a group called FIFO_CORE consisting of all of the RAM primitives traced forward on the net. The specification shows the schematic and UCF syntax.

Schematic syntax

TNM=RAMS FIFO_CORE

UCF syntax

NET *net_name* **TNM=RAMS FIFO_CORE;**

The following figure illustrates the preceding TNM identifier. The two RAMs traced forward from the net are included in the group. The flip flop is not.

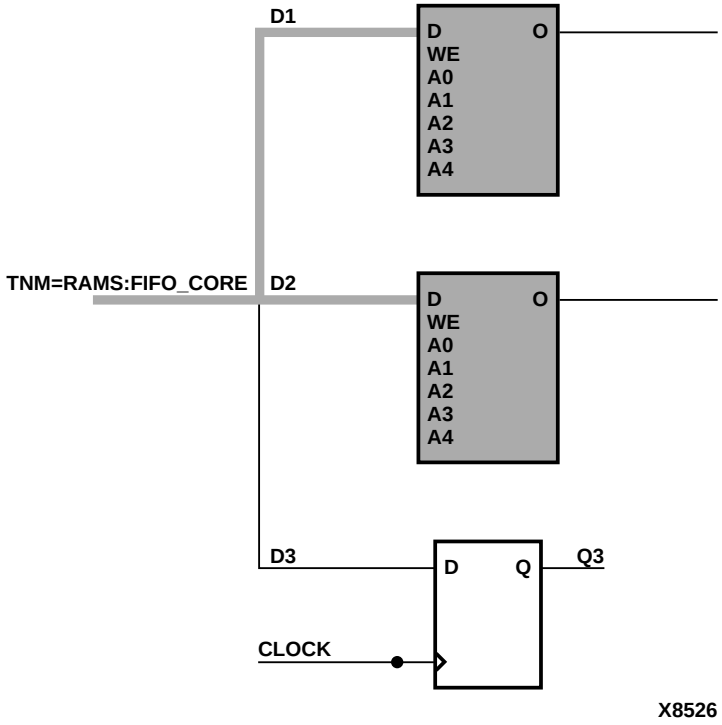


Figure 4-3 TNM Placed on a Net

A defined net in a TNM statement can have a name qualifier (for example, TMM=FFS (FRED*) GRP_A), as described in the “Creating Groups by Pattern Matching” section.

You can use several methods for tagging groups of end points: placing identifiers on nets, macro or primitive pins, primitives, or macro symbols. Which method you choose depends on how the path end points are related in your design. For each of these elements, you can use the predefined group syntax described earlier in this section.

The following subsections discuss the different methods of placing TNMs in your design. The same TNM attribute can be used as many ways and as many times as necessary to get the TNM applied to all of the elements in the desired group.

You can place TNM attributes in either of two places: in the schematic as discussed in this section or in a constraints file (UCF or NCF).

The syntax for specifying TNMs in a UCF or NCF constraints file is described in the "Attributes, Constraints, and Carry Logic" chapter of the *Libraries Guide*.

Placing TNMs on Nets

The TNM attribute can be placed on any net in the design. The attribute indicates that the TNM value should be attached to all valid elements fed by all paths that fan forward from the tagged net. Forward tracing stops at any flip-flop, latch, RAM or pad. See the "TNM Placed on a Net" figure. TNMs do not propagate across IBUFs if they are attached to the input pad net. Also refer to the "Creating User-Defined Groups Using TNM_NET" section.

Placing TNMs on Macro or Primitive Pins

The TNM attribute can be placed on any macro or component pin in the design if the design entry package allows placement of attributes on macro or primitive pins. The attribute indicates that the TNM value should be attached to all valid elements fed by all paths that fan forward from the tagged pin. Forward tracing stops at any flip-flop, latch, RAM or pad. The following illustration shows the valid elements for a TNM attached to the schematic a macro pin.

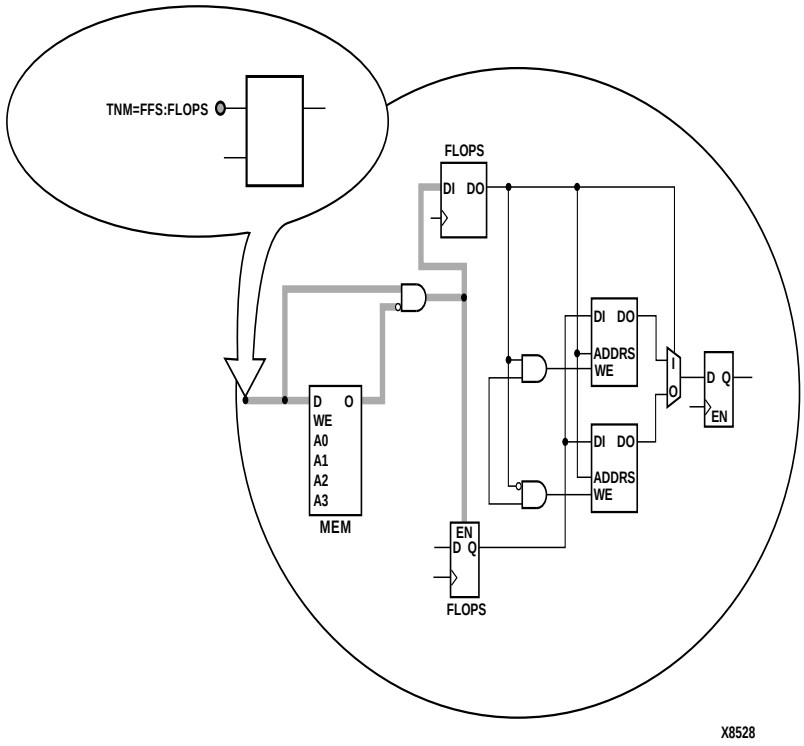


Figure 4-4 TNM Placed on a Macro Pin

The syntax for the UCF file would be

PIN *pin_name* **TNM=FFS FLOPS;**

Placing TNMs on Primitive Symbols

You can group individual logic primitives explicitly by flagging each symbol, as illustrated by the following figure.

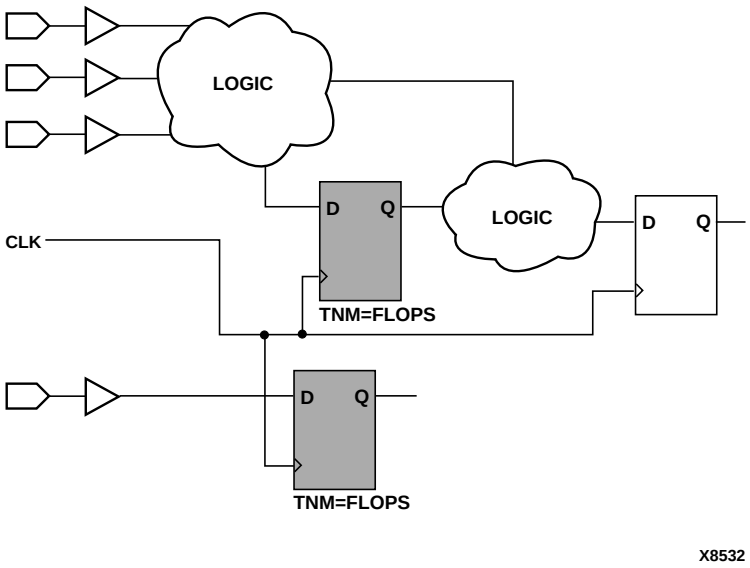


Figure 4-5 TNM on Primitive Symbols

In the figure, the flip-flops tagged with the TNM form a group called "'FLOPS." The untagged flip-flop on the right side of the drawing is not part of the group.

Place only one TNM on each symbol, driver pin, or macro driver pin.

Schematic syntax

```
TNM=FLOPS
```

UCF syntax

```
INST symbol_name TNM=FLOPS;
```

Placing TNMs on Macro Symbols

A macro is an element that performs some general purpose higher level function. It typically has a lower level design that consists of primitives, other macros, or both, connected together to implement the higher level function. An example of a macro function is a 16-bit counter.

A TNM attribute attached to a macro indicates that all elements inside the macro (at all levels of hierarchy below the tagged macro) are part of the named group.

When a macro contains more than one symbol type and you want to group only a single type, use the TNM identifier in conjunction with one of the predefined groups: FFS, RAMS, PADS, or LATCHES as indicated by the following syntax examples.

Schematic syntax

```
TNM=FFS identifier
TNM=RAMS identifier
TNM=LATCHES identifier
TNM=PADS identifier
```

UCF syntax

```
INST macro_name TNM=FFS identifier;
INST macro_name TNM=RAMS identifier;
INST macro_name TNM=LATCHES identifier;
INST macro_name TNM=PADS identifier;
```

If multiple symbols of the same type are contained in the same hierarchical block, you can simply flag that hierarchical symbol, as illustrated by the following figure. In the figure, all flip-flops included in the macro are tagged with the TNM "FLOPS". By tagging the macro symbol, you do not have to tag each underlying symbol individually.

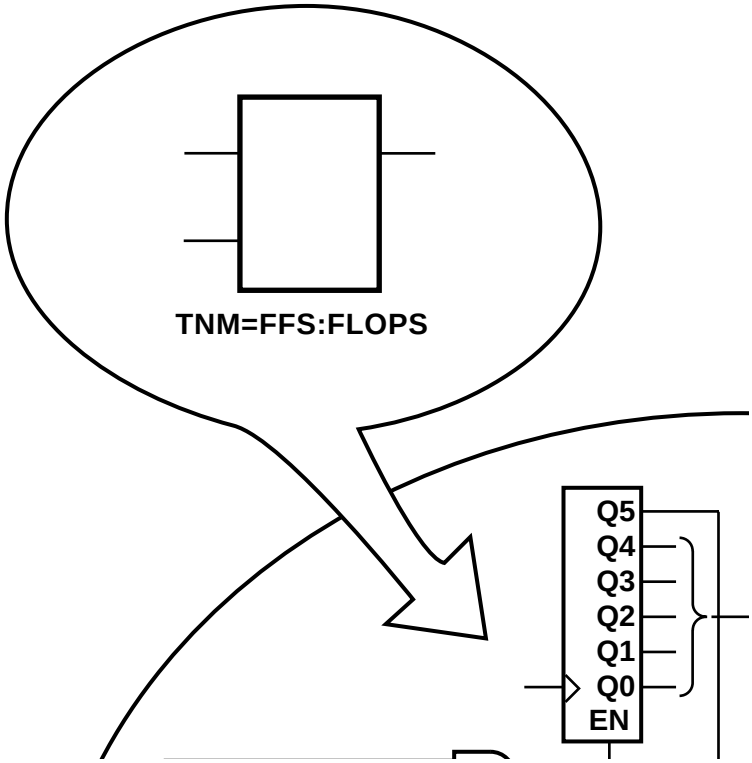


Figure 4-6 TNM on Macro Symbol

Placing TNMs on Nets or Pins to Group Flip-Flops and Latches

You can easily group flip-flops, latches, or both by flagging a common input net, typically either a clock net or an enable net. If you attach a TNM to a net or driver pin, that TNM applies to all flip-flops and input latches that are reached through the net or pin. That is, that path is traced forward, through any number of gates or buffers, until it reaches a flip-flop or input latch. That element is added to the specified TNM group.

The following figure illustrates the use of a TNM on a net that traces forward to create a group of flip-flops.

In the figure, the attribute TNM=FLOPS traces forward to the first two flip-flops, which form a group called FLOPS. The bottom flip-flop is not part of the group FLOPS

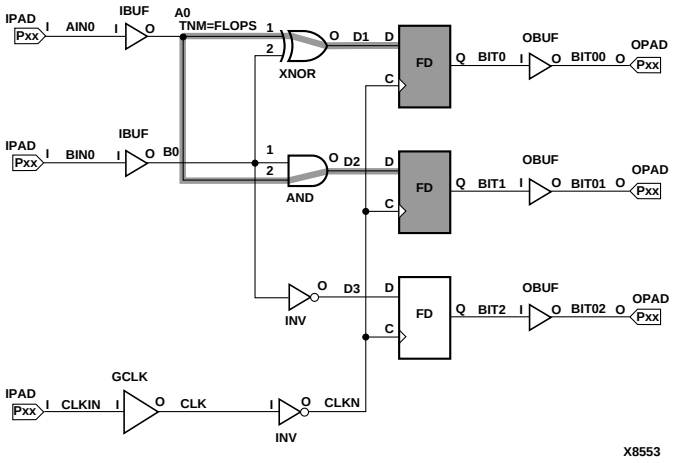


Figure 4-7 TNM on Net Used to Group Flip-Flops

The following figure illustrates placing a TNM on a clock net, which traces forward to all three flip-flops and forms the group Q_FLOPS.

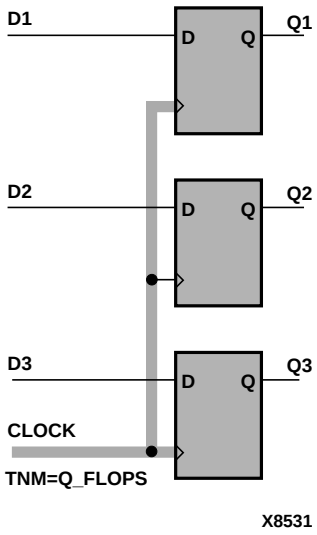


Figure 4-8 TNM on Clock Pin Used to Group Flip-Flops

The TNM parameter on nets or pins is allowed to have a qualifier.

For example, on schematics

TNM=FFS data

TNM=RAMS fifo

TNM=LATCHES capture

In UCF files

{**NET** | **PIN**} *net_or_pin_name* **TNM=FFS data;**

{**NET** | **PIN**} *net_or_pin_name* **TNM=RAMS fifo;**

{**NET** | **PIN**} *net_or_pin_name* **TNM=LATCHES capture;**

A qualified TNM is traced forward until it reaches the first storage element (flip-flop, latch, or RAM). If that type of storage element matches the qualifier, the storage element is given that TNM value. Whether or not there is a match, the TNM is *not* traced through that storage element.

TNM parameters on nets or pins are never traced through a storage element (flip-flop, latch or RAM). In previous XACT*step*® software releases, they were traced through some pins on input latches and RAMs. If you rely on this behavior, move the TNM parameter so that it reaches the target flip-flop directly or place a TNM parameter on the target flip-flop symbol.

Creating User-Defined Groups Using TNM_NET

A TNM_NET (timing name for nets) is an attribute that can be used to identify the elements that make up a group which can then be used in a timing specification. Essentially TNM_NET is equivalent to TNM on a net *except* for pad nets.

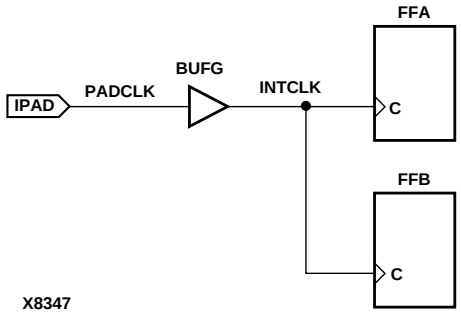
A TNM_NET is a property that you normally use in conjunction with an HDL design to tag a specific net. All nets tagged with the TNM_NET identifier are considered a group. The UCF syntax is as follows.

NET *net_name* **TNM_NET=***identifier***;**

identifier is a value that consists of any combination of letters, numbers, or underscores. Keep the TNM_NET short for convenience and clarity. The basic syntax rules for TNM_NET and TNM are identical. Refer to the “Creating User-Defined Groups Using TNMs” section for details.

The TNM_NET attribute can be used to define certain types of nets that cannot be adequately described by the TNM constraint. This attribute is specifically targeted for use in HDL designs.

For example, consider the following design.



In the preceding design, a TNM associated with the PAD symbol only includes the PAD symbol as a member in a timing analysis group. For example, the following UCF file entry creates a time group that includes the IPAD symbol only.

NET PADCLK TNM=PADS(*) PADGRP; (UCF file example)

However, using TNM to define a time group for the net PADCLK creates an empty time group.

NET PADCLK TNM=FFS(*) FFGRP;(UCF file example)

All properties that apply to a pad are transferred from the net to the PAD symbol. Since the TNM is transferred from the net to the PAD symbol, the qualifier, "FFS(*)" does not match the PAD symbol.

To overcome this obstacle for schematic designs using TNM, you can create a time group for the INTCLK net.

NET INTCLK TNM=FFS(*) FFGRP;(UCF file example)

However, for HDL designs, the only meaningful net names are the ones connected directly to pads. Then, use TNM_NET to create the FFGRP time group.

NET PADCLK TNM_NET=FFS(*) FFGRP;(UCF file example)

NGDBuild does not transfer a TNM_NET attribute from a net to a PAD as it does with TNM.

TNM_NET can be used in NCF or UCF files as a property attached to an object in an input netlist (EDIF or XNF). TNM_NET is not supported in PCF files.

TMN_NET can only be used with nets. If TNM_NET is used with any other object such as a pin or symbol, a warning is generated and the TNM_NET definition is ignored.

Creating New Groups from Existing Groups

In addition to naming groups using the TNM identifier, you can also define groups in terms of other groups. You can create a group that is a combination of existing groups by defining a TIMEGRP attribute as follows.

Schematic syntax in TIMEGRP primitive

*newgroup***=***existing_grp1 existing_grp2* [*existing_grp3* . . .]

UCF syntax

TIMEGRP *newgroup*=*existing_grp1 existing_grp2* [*existing_grp3 . . .*];

newgroup is a newly created group that consists of existing groups created via TNMs, predefined groups, or other TIMEGRP attributes.

The Mentor netlist writer (ENWRITE™) converts all property names to lower case letters, and the Xilinx netlist reader EDIF2NGD then converts the property names to upper case letters. To ensure references from one constraint to another are processed correctly,

- Group names should contain only upper case letters on a Mentor Schematic (MY_FLOPS, for example, but not my_flops or My_flops).
- If a group name appears in a property value, it must also be expressed in upper case letters. For example, the GROUP3 in the first constraint below must be entered in upper case letters to match the GROUP3 in the second constraint.

  Schematic syntax in TIMEGRP primitive

  ```
  GROUP1 = gr2 GROUP3
  GROUP3 = FFS except grp5
  ```

  UCF syntax

  ```
  TIMEGRP GROUP1 = gr2 GROUP3;
  TIMEGRP GROUP3 = FFS except grp5;
  ```

TIMEGRP attributes reside in the TIMEGRP primitive, as illustrated in the figure below. Once you create a TIMEGRP attribute definition within a TIMEGRP primitive, you can use it in the TIMESPEC primitive. Each TIMEGRP primitive can hold up to eight group definitions. Since your design might include more than eight TIMEGRP attributes, you can use multiple TIMEGRP primitives.

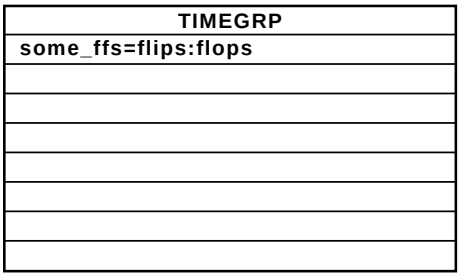


Figure 4-9 TIMEGRP Primitive

You can place TIMEGRP attributes in either of two places: in the TIMEGRP primitive on the schematic as discussed in this section or in a constraints file (UCF or NCF). The syntax for specifying TIMEGRPs in a UCF or NCF constraints file is described in the "Attributes, Constraints, and Carry Logic" chapter of the *Libraries Guide*.

You can use TIMEGRP attributes to create groups using the following methods.

- Combining multiple groups into one
- Creating groups by exclusion
- Defining flip-flop subgroups by clock sense

The following subsections discuss each method in detail.

Combining Multiple Groups into One

You can define a group by combining other groups. The following syntax example illustrates the simple combining of two groups.

Schematic syntax in TIMEGRP primitive

```
big_group=small_group medium_group
```

UCF syntax

```
TIMEGRP big_group=small_group medium_group;
```

In this syntax example, small_group and medium_group are existing groups defined using a TNM or TIMEGRP attribute. Within the TIMEGRP primitive, TIMEGRP attributes can be listed in any order;

that is, you can create a TIMEGRP attribute that references another TIMEGRP attribute that appears after the initial definition.

Warning: A circular definition, as shown below, causes an error when you run your design through NGDBuild.

Schematic syntax in TIMEGRP primitive

```
many_ffs=ffs1 ffs2
ffs1=many_ffs ffs3
```

UCF syntax

```
TIMEGRP many_ffs=ffs1 ffs2;
TIMEGRP ffs1=many_ffs ffs3;
```

Creating Groups by Exclusion

You can define a group that includes all elements of one group except the elements that belong to another group, as illustrated by the following syntax examples.

Schematic syntax in TIMEGRP primitive

group1=*group2* **EXCEPT** *group3*

UCF syntax

TIMEGRP *group1*=*group2* **EXCEPT** *group3*;

- *group1* represents the group being defined. It contains all of the elements in *group2* except those that are also in *group3*.
- *group2* and *group3* can be a valid TNM, predefined group, or TIMEGRP attribute.

As illustrated by the following example, you can specify multiple groups to include or exclude when creating the new group.

Schematic syntax in TIMEGRP primitive

group1=*group2* *group3*:EXCEPT *group4* *group5*

UCF syntax

TIMEGRP *group1*=*group2* *group3*:EXCEPT *group4* *group5*;

The example defines a *group1* that includes the members of *group2* and *group3*, except for those members that are part of *group4* or *group5*. All of the groups before the keyword EXCEPT are included, and all of the groups after the keyword are excluded.

Certain reserved words cannot be used as group names. These reserved words are described in the "Creating User-Defined Groups Using TNMs" section.

Defining Flip-Flop Subgroups by Clock Sense

You can create subgroups using the keywords RISING and FALLING to group flip-flops triggered by rising and falling edges.

Schematic syntax in TIMEGRP primitive

```
group1=RISING ffs
group2=RISING ffs_group
group3=FALLING ffs
group4=FALLING ffs_group
```

UCF syntax

```
TIMEGRP group1=RISING ffs;
TIMEGRP group2=RISING ffs_group;
TIMEGRP group3=FALLING ffs;
TIMEGRP group4=FALLING ffs_group;
```

group1 to *group4* are new groups being defined. The *ffs_group* must be a group that includes only flip-flops.

Note: Keywords, such as EXCEPT, RISING, and FALLING, appear in the documentation in upper case; however, you can enter them in the TIMEGRP primitive in either lower or upper case. You cannot enter them in a combination of lower and upper case.

The following example defines a group of flip-flops that switch on the falling edge of the clock.

Schematic syntax in TIMEGRP primitive

```
falling_ffs=FALLING ffs
```

UCF syntax

```
TIMEGRP falling_ffs=FALLING ffs;
```

Defining Latch Subgroups by Gate Sense

Groups of type LATCHES (no matter how these groups are defined) can be easily separated into transparent high and transparent low subgroups. The TRANSHI and TRANSLO keywords are provided for this purpose, and are used in TIMEGRP statements like the RISING and FALLING keywords for flip-flop groups. For example

Schematic syntax in TIMEGRP primitive

```
lowgroup=TRANSLO latchgroup
highgroup=TRANSHI latchgroup
```

UCF syntax

```
TIMEGRP lowgroup=TRANSLO latchgroup;
TIMEGRP highgroup=TRANSHI latchgroup;
```

Creating Groups by Pattern Matching

When creating groups, you can use wildcard characters to define groups of symbols whose associated net names match a specific pattern.

How to Use Wildcards to Specify Net Names

The wildcard characters, * and ?, enable you to select a group of symbols whose output net names match a specific string or pattern. The asterisk (*) represents any string of zero or more characters. The question mark (?) indicates a single character.

For example, DATA* indicates any net name that begins with “DATA,” such as DATA, DATA1, DATA22, DATABASE, and so on. The string NUMBER? specifies any net names that begin with ‘‘NUMBER” and end with one single character, for example, NUMBER1, NUMBERS but not NUMBER or NUMBER12.

You can also specify more than one wildcard character. For example, *AT? specifies any net names that begin with any series of characters followed by ‘‘AT” and end with any one character such as BAT1, CAT2, and THAT5. If you specify *AT??, you would match BAT11, CAT26, and THAT50.

Pattern Matching Syntax

The syntax for creating a group using pattern matching is shown below.

Schematic syntax in TIMEGRP primitive

*group***=***predefined_group***(***pattern***)**

UCF syntax

TIMEGRP *group***=***predefined_group***(***pattern***);**

predefined_group can only be one of the following predefined groups—FFS, LATCHES, PADS, or RAMS. The *pattern* is any string of characters used in conjunction with one or more wildcard characters.

Warning: When specifying a net name, you must use its full hierarchical path name so PAR can find the net in the flattened design.

For flip-flops, input latches, and RAMs, specify the output net name. For pads, specify the external net name.

The following example illustrates creating a group that includes the flip-flops that source nets whose names begin with $1I3/FRED.

Schematic syntax in TIMEGRP primitive

```
group1=ffs($1I3/FRED*)
```

UCF syntax

```
TIMEGRP group1=ffs($1I3/FRED*);
```

The following example illustrates a group that excludes certain flip-flops whose output net names match the specified pattern.

Schematic syntax in TIMEGRP primitive

```
this_group=ffs EXCEPT ffs(a*)
```

UCF syntax

```
TIMEGRP this_group=ffs EXCEPT ffs(a*);
```

In this example, `this_group` includes all flip-flops except those whose output net names begin with the letter “a.”

The following defines a group named “some_latches”.

Schematic syntax in TIMEGRP primitive

```
some_latches=latches($1I3/xyz*)
```

UCF syntax

```
TIMEGRP some_latches=latches($113/xyz*);
```

The group `some_latches` contains all input latches whose output net names start with "$1I3/xyz."

Additional Pattern Matching Details

In addition to using pattern matching when you create timing groups, you can specify a predefined group qualified by a pattern any place you specify a predefined group. The syntax below illustrates how pattern matching can be used within a timing specification.

Schematic syntax in TIMESPEC primitive

TS*identifier***=FROM** *predefined_group***(***pattern***) TO** *predefined_group* **(***pattern***)** *delay*

UCF syntax

TIMESPEC TS*identifier***=FROM** *predefined_group***(***pattern***) TO** *predefined_group* **(***pattern***)** *delay***;**

Instead of specifying just one pattern, you can also specify a list of patterns separated by a colon (:) as illustrated below.

Schematic syntax in TIMEGRP primitive

```
some_ffs=ffs(a*:b?:c*d)
```

UCF syntax

```
TIMEGRP some_ffs=ffs(a*:b?:c*d);
```

The group `some_ffs` contains flip-flops whose output net names

- Start with the letter "a"

  or

- Contain two characters; the first character is "b"

  or

- Start with "c" and end with "d"

Defining a Clock Period (PERIOD Constraint)

A clock period specification checks timing clocked by the net (all paths that terminate at a register clocked by the specified net).

The period specification is attached to the clock net. The definition of a clock period is unlike a FROM-TO style specification because the timing analysis tools automatically take into account any inversions of the clock net at register clock pins.

A PERIOD constraint on the clock net in the following figure would generate a check for delays on all paths that terminate at a pin that has a setup or hold timing constraint relative to the clock net. This could include the data paths D1 to CLB1.D, CLB1.Q to CLB2.D, as well as the path EN to CLB2.EC (if the reset / enable were synchronous with respect to the clock).

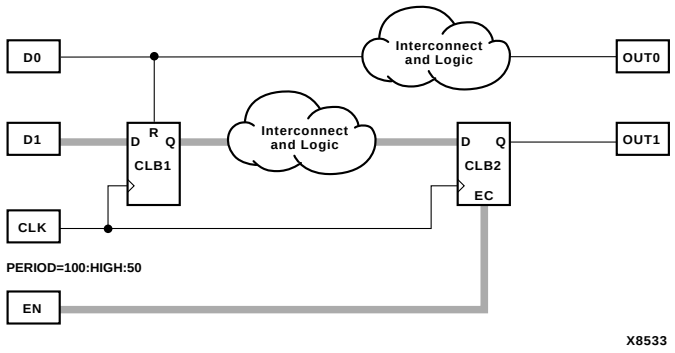


Figure 4-10 Paths for PERIOD Constraint

Simple Method

A simple method of defining a clock period is to attach the following attribute directly to a net in the path that drives the register clock pins.

Schematic syntax

PERIOD = *period* { **HIGH** | **LOW** } [*high_or_low_time*]

UCF syntax

[*period_item*] **PERIOD =** *period* { **HIGH** | **LOW** } [*high_or_low_time*]**;**

period_item is one of the following,

- NET "*net_name*"
- TIMEGRP *"group_name"*
- ALLCLOCKNETS (PCF only)

period is the required clock period. The default units are nanoseconds, but the timing number can be followed by ps, ns, us, or ms. Units may be entered with or without a leading space, and are case-insensitive. The **HIGH**|**LOW** keyword indicates whether the first pulse in the period is high or low, and the optional *high_or_low_time* is the duty cycle of the first pulse. If an actual time is specified, it must be less than the period. If no high or low time is specified the default duty cycle is 50%. The default units for *high_or_low_time* is ns, but the number can be followed by % or by ps, ns, us or ms if you want to specify an actual time measurement.

The PERIOD constraint is forward traced in exactly the same way a TNM would be and attaches itself to all of the flip-flops that the forward tracing reaches. If a more complex form of tracing behavior is required (for example, where gated clocks are used in the design), you must place the PERIOD on a particular net or use the preferred method described next.

Preferred Method

The preferred method for defining a clock period allows more complex derivative relationships to be defined as well as a simple clock period. The following attribute is attached to a TIMESPEC symbol in conjunction with a TNM attribute attached to the relevant clock net.

Schematic syntax in a TIMSPEC symbol

TS*identifier***=PERIOD** *TNM_reference period* {**HIGH** | **LOW**} [*high_or_low_time*]

UCF syntax

TIMESPEC TS*identifier***=PERIOD** *TNM_reference period* {**HIGH** | **LOW**} [*high_or_low_time*]**;**

identifier is a reference identifier that has a unique name.

TNM_reference is the identifier name that is attached to a clock net (or a net in the clock path) using a TNM attribute.

- NET "*net_name*"
- TIMEGRP *"group_name"*
- ALLCLOCKNETS

The variable name *period* is the required clock period. The default units for *period* are nanoseconds, but the number can be followed by ps, ns, us, or ms. Units may be entered with or without a leading space, and are case-insensitive. The **HIGH** | **LOW** keyword indicates whether the first pulse in the period is high or low, and the optional *high_or_low_time* is the polarity of the first pulse. If an actual time is specified, it must be less than the period. If no high or low time is specified the default duty cycle is 50%. The default units for *high_or_low_time* is ns, but the number can be followed by % or by ps, ns, us, or ms if you want to specify an actual time measurement.

Example

Clock net sys_clk has the attribute `tnm=master_clk` attached to it and the following attribute is attached to a TIMESPEC primitive.

Schematic syntax in a TIMESPEC symbol

TS_master=PERIOD master_clk 50 HIGH 30

UCF syntax

```
TIMESPEC TS_master=PERIOD master_clk 50 HIGH 30;
```

This period constraint applies to the net master_clk, and defines a clock period of 50 nanoseconds, with an initial 30 nanosecond high time.

Specifying Derived Clocks

The preferred method of defining a clock period uses an identifier, allowing another clock period specification to reference it. To define the relationship in the case of a derived clock, use the following syntax.

Schematic syntax in a TIMSPEC symbol

TS*identifier***=PERIOD** *TNM_reference another_PERIOD_identifier* [{**/** | *****}*number*] [{**HIGH** | **LOW**} *high_or_low_time*]

UCF syntax

TIMESPEC TS*identifier***=PERIOD** *TNM_reference another_PERIOD_identifier* [{**/** | *****}*number*] [{**HIGH** | **LOW**} *high_or_low_time*]**;**

- *identifier* is a reference identifier that has a unique name.
- *TNM_reference* is the identifier name that is attached to a clock net or a net in the clock path using a TNM attribute.
- a*nother_PERIOD_identifier* is the name of the identifier used on another period specification.
- *number* is a floating point number.
- The **HIGH** | **LOW** keyword indicates whether the first pulse in the period is high or low, and the optional *high_or_low_time* is the polarity of the first pulse. If an actual time is specified it must be less than the period. If no high or low time is specified, the default duty cycle is 50%. The default units for *high_or_low_time* is ns, but the number can be followed by % or by ps, ns, us, or ms if you want to specify an actual time measurement.

Example

A clock net has the attribute tnm=slave_clk attached to it and the following attribute is attached to a TIMESPEC primitive.

Schematic syntax in a TIMESPEC symbol

```
ts_slave1=PERIOD slave_clk TS_master *4
```

UCF syntax

```
TIMESPEC ts_slave1=PERIOD slave_clk TS_master
*4;
```

OFFSET Timing Specifications

Offsets are used to define the timing relationship between an external clock and its associated data-in or data-out pin. Using this option allows you to do the following.

- Calculate whether a setup time is being violated at a flip-flop whose data and clock inputs are derived from external nets.
- Specify the delay of an external output net derived from the Q output of an internal flip-flop being clocked from an external device pin.

Following are some of the advantages of using the OFFSET constraint.

- Includes the clock path delay for each individual synchronous elements
- Subtracts the clock path delay from the data path delay for inputs and adds the clock path delay to the data path delay for outputs
- Includes paths for all synchronous element types (FFS, RAMS, and LATCHES)
- Utilizes a global syntax that allows all inputs or outputs to be constrained by a clock
- Allows specifying IO constraints either directly as the setup and clock-to-out required by a device (IN BEFORE and OUT AFTER) or indirectly as the time used by the path external to the device (IN AFTER and OUT BEFORE)

There are basically three types of offset specifications.

- Global
- Net-specific
- Group

Since the global and group OFFSET constraints are not associated with a single data net or component, these two types can also be entered on a TIMESPEC symbol in the design netlist with Ts*id*.

Schematic syntax in a TIMESPEC symbol

TS*id*=[**TIMEGRP** *name*] **OFFSET =** {**IN**|**OUT**} *offset_time* [*units*] {**BEFORE**|**AFTER**} *clk_name* [**TIMEGRP** *group_name*]

UCF syntax

[**TIMEGRP** *name*] **OFFSET =** {**IN**|**OUT**} *offset_time* [*units*] {**BEFORE**|**AFTER**} *clk_name* [**TIMEGRP** *group_name*]**;**

Note: In the UCF file, you cannot specify the TS*id* format.

See the next section and the "Group OFFSET" section for syntax details. As with the PERIOD and MAXDELAY timing specifications, if the same TS*id* is defined in the design netlist (or NCF) and the UCF file, the UCF file takes precedence.

The following subsections describe the use of each type of OFFSET in PCF and UCF files and explain the scope of each specification.

Global OFFSET

Release 1.5 supports the use of the global OFFSET constraint. Release 1.5 also supports the use of time groups within global OFFSET constraints. On a schematic, enter the global OFFSET in the TIMESPEC symbol.

UCF syntax

OFFSET = {**IN**|**OUT**} *offset_time* [*units*] {**BEFORE**|**AFTER**} *clk_name* [**TIMEGRP** *group_name*]**;**

PCF syntax

OFFSET = {**IN**|**OUT**} *offset_time* [*units*] {**BEFORE**|**AFTER**} **COMP** *clk_iob_name* [**TIMEGRP** *group_name*]**;**

offset_time is the external offset and *units* is an optional field that indicates the units for the offset time. The default units are nanoseconds, but the timing number can be followed by ps, ns, us, ms, GHz, MHz, or KHz to show the intended units.

The UCF syntax variable *clk_name* is the fully hierarchical net name of the clock net between its pad and its input buffer.

The *clk_iob_name* is the block name of the clock IOB.

The optional TIMEGRP *group_name* defines a group of registers that will be analyzed. By default, all registers clocked by *clk_name* will be analyzed.

IN | **OUT** specifies that the offset is computed with respect to an input IOB or an output IOB. For a bidirectional IOB, the **IN** | **OUT** syntax lets you specify the flow of data (input or output) on the IOB.

BEFORE | **AFTER** indicates whether data is to arrive (input) or leave (output) the device before or after the clock input.

All inputs / outputs are offset relative to *clk_name* or *iob_name*. For example, **OFFSET IN 20 ns BEFORE clk1** dictates that all inputs will have data present at the pad at least 20 ns before the triggering edge of clk1 arrives at the pad.

Net-Specific OFFSET Constraints

The OFFSET constraint can also be used to specify a constraint for a specific data net in a UCF file or schematic or a specific input or output component in a PCF file.

Schematic syntax when attached to a net

OFFSET = {**IN** | **OUT**} *offset_time* [*units*] {**BEFORE** | **AFTER**} *clk_name* [**TIMEGRP** *group_name*]

UCF syntax

NET *name* **OFFSET =** {**IN** | **OUT**} *offset_time* [*units*] {**BEFORE** | **AFTER**} *clk_name* [**TIMEGRP** *group_name*]**;**

PCF syntax

COMP "*iob_name***" OFFSET =** {**IN** | **OUT**} *offset_time* [*units*] {**BEFORE** | **AFTER**} **COMP** "*clk_iob_name*" [**TIMEGRP** "*group_name*"]**;**

The PCF file uses blocks (comps) instead of nets.

If **COMP** "iob_name" is omitted in the PCF or NET "name" is omitted in the UCF, the specification is assumed to be global.

offset_time is the external offset.

units is an optional field that indicates the units for offset time. The default units are in nanoseconds, but the timing number can be followed by ps, ns, us, GHz, MHz, or KHz to indicate the intended units.

clk_iob_name is the block name of the clock IOB.

It is possible for one offset constraint to generate multiple data and clock paths (for example, when both data and clock inputs have more than a single sequential element in common).

Examples

The offset constraint examples in this section apply to the following figures.

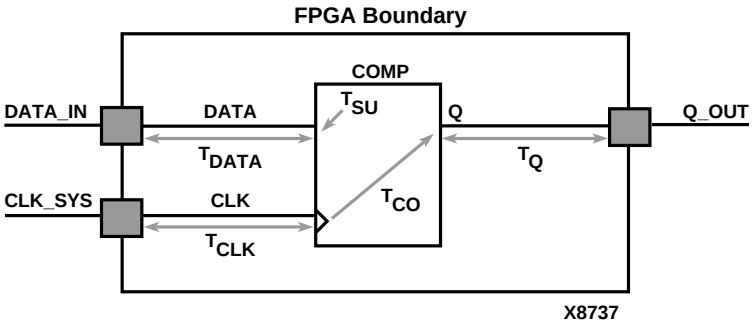


Figure 4-11 OFFSET Example Schematic

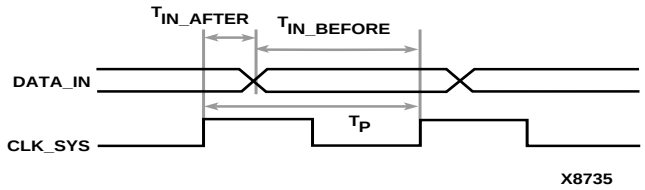


Figure 4-12 OFFSET IN Timing Diagram

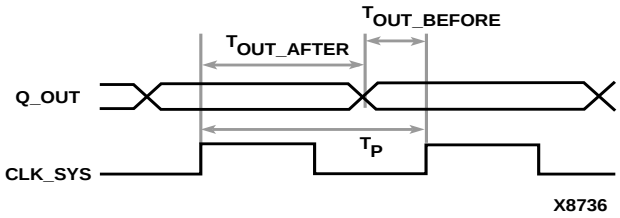


Figure 4-13 OFFSET OUT Timing Diagram

Example 1— OFFSET IN BEFORE

OFFSET IN BEFORE defines the available time for data to propagate from the pad and setup at the synchronous element (COMP). The time can be thought of as the time differential of data arriving at the edge of the device before the next clock edge arrives at the device. See the “OFFSET Example Schematic” figure and the “OFFSET IN Timing Diagram” figure. The equation that defines this relationship is as follows.

$T_{DATA} + T_{SU} - T_{CLK} \leq T_{IN\_BEFORE}$

For example, if T_{IN_BEFORE} equals 20 ns, the following syntax applies.

Schematic syntax attached to DATA_IN

```
OFFSET=IN 20.0 BEFORE CLK_SYS
```

UCF syntax

```
NET DATA_IN OFFSET=IN 20.0 BEFORE CLK_SYS;
```

PCF syntax

```
COMP DATA_IN OFFSET=IN 20.0 ns BEFORE COMP
CLK_SYS;
```

This constraint indicates that the data will be present on the DATA_IN pad at least 20 ns before the triggering edge of the clock net arrives at the clock pad.

To ensure that the timing requirements are met, the timing analysis software verifies that the maximum delay along the path DATAIN to COMP (minus the 20.0 ns offset) would be less than or equal to the minimum delay along the reference path CLOCK to COMP.

Example 2 — OFFSET IN AFTER

This constraint describes the time used by the data external to the FPGA. OFFSET subtracts this time from the PERIOD declared for the clock to determine the available time for the data to propagate from the pad and setup at the synchronous element. The time can be thought of as the differential of data arriving at the edge of the device after the current clock edge arrives at the edge of the device. See the “OFFSET Example Schematic” figure and the “OFFSET OUT Timing Diagram” figure. The equation that defines this relationship is as follows.

$T_{DATA} + T_{SU} - T_{CLK} \leq T_P - T_{IN\_AFTER}$

T_P is the clock period.

For example, if T_{IN_AFTER} equals 30 ns, the following syntax applies.

Schematic syntax attached to DATA_IN

```
OFFSET=IN 30.0 AFTER CLK_SYS;
```

UCF syntax

```
NET DATA_IN OFFSET=IN 30.0 AFTER CLK_SYS;
```

PCF syntax

```
COMP DATA_IN OFFSET=IN 30.0 ns AFTER COMP
CLK_SYS;
```

This constraint indicates that the data will arrive at the pad of the device (COMP) no more than 30 ns after the triggering edge of the clock arrives at the clock pad. The path DATA_IN to COMP would contain the setup time for the COMP data input relative to the CLK_SYS input.

Verification is almost identical to Example 1, except that the offset margin (30.0 ns) is added to the data path delay. This is caused by the data arriving after the reference input. The timing analysis software verifies that the data can be clocked in prior to the next triggering edge of the clock.

A PERIOD or FREQUENCY is required only for offset **OUT** constraints with the **BEFORE** keyword or offset **IN** with the **AFTER** keyword.

Example 3 — OFFSET OUT AFTER

This constraint defines the time available for the data to propagate from the synchronous element to the pad. This time can also be considered as the differential of data leaving the edge of the device after the current clock edge arrives at the edge of the device. See the "OFFSET Example Schematic" figure and the "OFFSET OUT Timing Diagram" figure.

The equation that defines this relationship is as follows.

$T_Q + T_{CO} - T_{CLK} \leq T_{OUT\_AFTER}$

For example, if T_{OUT_AFTER} equals 35 ns, the following syntax applies.

Schematic syntax attached to Q_OUT

```
OFFSET=OUT 35.0 AFTER CLK_SYS
```

UCF syntax

```
NET Q_OUT OFFSET=OUT 35.0 AFTER CLOCK;
```

PCF syntax

```
COMP Q_OUT OFFSET=OUT 35.0 ns AFTER COMP
CLK_SYS;
```

This constraint calls for the data to leave the FPGA 35 ns after the present clock input arrives at the clock pad. The path COMP to Q_OUT would include the CLOCK-to-Q delay (component delay).

Verification involves ensuring that the maximum delay along the reference path (CLK_SYS to COMP) and the maximum delay along the data path (COMP to Q_OUT) does not exceed the specified offset.

Example 4 — OFFSET OUT BEFORE

This constraint defines the time used by the data external to the FPGA. OFFSET subtracts this time from the clock PERIOD to determine the available time for the data to propagate from the synchronous element to the pad. The time can also be considered as the differential of data leaving the edge of the device before the next clock edge arrives at the edge of the device See the “OFFSET Example Schematic” figure and the “OFFSET OUT Timing Diagram” figure. The equation that defines this relationship is as follows.

$T_Q + T_{CO} + T_{CLK} \leq T_P - T_{OUT\_BEFORE}$

For example, if T_{OUT_BEFORE} equals 15 ns, the following syntax applies.

Schematic syntax attached to Q_OUT

```
OFFSET=OUT 15.0 BEFORE CLK_SYS
```

UCF syntax

```
NET Q_OUT OFFSET=OUT 15.0 BEFORE CLK_SYS;
```

PCF syntax

```
COMP Q_OUT OFFSET=OUT 15.0 ns BEFORE COMP
CLK_SYS;
```

This constraint states that the data clocked to Q_OUT must leave the FPGA 15 ns before the next triggering edge of the clock arrives at the clock pad. The path COMP to Q_OUT includes the CLK_SYS-to-Q delay (component delay). The data clocked to Q_OUT will leave the FPGA 15.0 ns before the next clock input.

Verification involves ensuring that the maximum delay along the reference path (CLK_SYS to COMP) and the maximum delay along the data path (COMP to Q_OUT) do not exceed the clock period minus the specified offset.

As in Example 2, a PERIOD or FREQUENCY constraint is required only for offset **OUT** constraints with the **BEFORE** keyword or offset **IN** with the **AFTER** keyword.

Specific OFFSET Constraints with Timegroups

A clock register time group allows you to define a specific set of registers to which an OFFSET constraint applies based on a clock edge. Consider the following example.

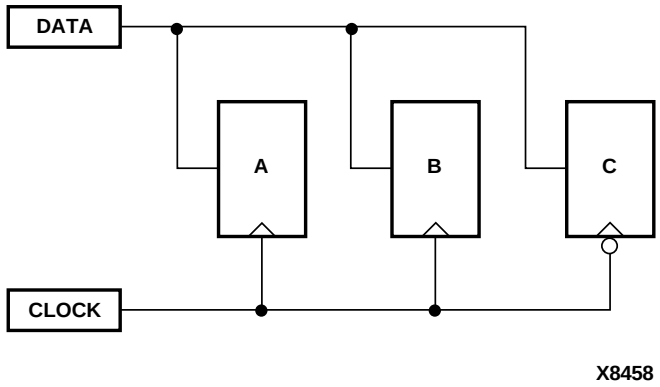


Figure 4-14 Using Timegroups with Registers

You can define time groups for the registers A, B and C, even though these registers have the same data and clock source. The syntax is as follows.

Schematic syntax in TIMEGRP primitive

```
AB=RISING FFS
C =FALLING FFS;
```

UCF / PCF syntax

```
TIMEGRP AB=RISING FFS;
TIMEGRP C =FALLING FFS;
```

Schematic syntax attached to DATA

```
OFFSET=IN 10 BEFORE CLOCK TIMEGRP AB

OFFSET=IN 20 BEFORE CLOCK TIMEGRP C
```

UCF syntax

```
NET DATA OFFSET=IN 10 BEFORE CLOCK TIMEGRP AB;

NET DATA OFFSET=IN 20 BEFORE CLOCK TIMEGRP C;
```

PCF syntax

```
COMP DATA OFFSET=IN 10 BEFORE COMP CLOCK TIMEGRP
AB;

COMP DATA OFFSET=IN 20 BEFORE COMP CLOCK TIMEGRP
C;
```

Even though the registers A, B and C have a common data and clock source, timing analysis applies two different offsets (10 ns and 20 ns). Registers A and B belong to the offset with 10 ns and Register C belongs to the offset with 20 ns.

However, you must use some caution when using timegroups with registers. Consider the following diagram.

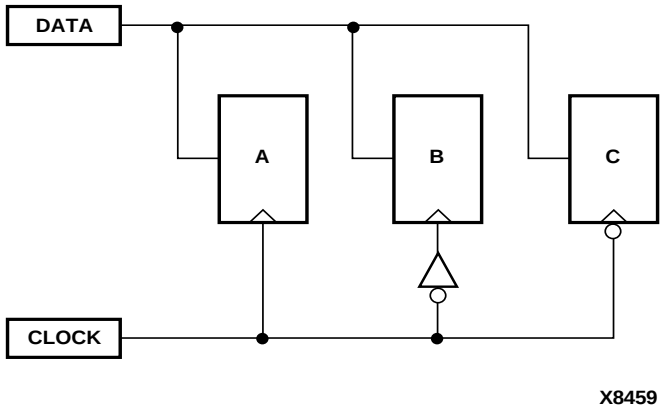


Figure 4-15 Problematic Timegroup Definition

This circuit is identical to the "Using Timegroups with Registers" figure except that an inverter has been inserted in the path to Register B. In this instance, even though this register is a member of the time group whose offset triggers on the rising edge, the addition of the inverter classifies register B as triggering on the falling edge like Register C.

Normally, the tools will move an inverter to the register, in which case, B would be a part of the timegroup "Falling". However if the clock is gated with logic that inverts, then the inverter will not become part of the register. In that case, one way to solve this problem is to create a timegroup with an exception for Register B. See the "Creating Groups by Exclusion" section for details.

Group OFFSET

You can also define OFFSET constraints within the TIMESPEC primitive with a leading TIMEGRP reference.

Schematic syntax in TIMESPEC primitive

TS*identifier***=TIMEGRP** *name* **OFFSET=** {**IN**|**OUT**} *offset_time* [*units*] {**BEFORE**|**AFTER**} *clk_name* [**TIMEGRP** *group_name*]

The UCF and PCF syntax do not require the **TS***identifier*.

UCF syntax

[**TIMEGRP** *name*] **OFFSET=** {**IN**|**OUT**} *offset_time* [*units*] {**BEFORE**|**AFTER**} *clk_name* [**TIMEGRP** *group_name*]**;**

PCF syntax

[**TIMEGRP** *name*] **OFFSET=** {**IN**|**OUT**} *offset_time* [*units*] {**BEFORE**|**AFTER**} **COMP** *clk_iob_name* [**TIMEGRP** *group_name*]**;**

The timing group specified at the beginning has a different purpose than the timegroup specified at the end. The first time group is a list of data pads that the OFFSET applies to, while the last time group (register time group) is a list of synchronous elements that specifies which register elements clocked by *clk_name* or *clk_iob_nam*e should be analyzed.

Note: If the first group has FFs or the second group has PADS, NGDBuild generates an error.

offset_time is the external offset.

units is an optional field that indicates the units for offset time. The default units are in nanoseconds, but the timing number can be followed by ps, ns, us, GHz, MHz, or KHz to indicate the intended units.

clk_iob_name is the block name of the clock IOB.

Ignoring Selected Paths (TIG)

In a design, some paths do not require timing analysis. These are paths that exist in the design, but are never used during time-critical operations. If you indicate a timing requirement on these paths, more important paths might be slower, which can result in failure to meet the timing requirements.

The value of TIG may be any of the following.

- Empty (global TIG that blocks all paths)
- A single TSid to block
- A comma separated list of TSids to block, for example

```
NET $1I567/$Sig_5 TIG=TS_fast, TS_even_faster;
```

To indicate that all timing specifications through a net, primitive pin or macro pin are to be ignored, attach the following attribute to the desired element.

Schematic syntax

TIG

UCF syntax

{**NET** | **PIN** | **INSTANCE**} *name* **TIG;**

If this attribute is attached to a net, primitive pin, or macro pin, all paths that fan forward from the point of application of the attribute are treated as if they don't exist for the purposes of timing analysis during implementation. In the following figure, NET C is ignored. However, note that the lower path of NET B that runs through the two OR gates would not be ignored.

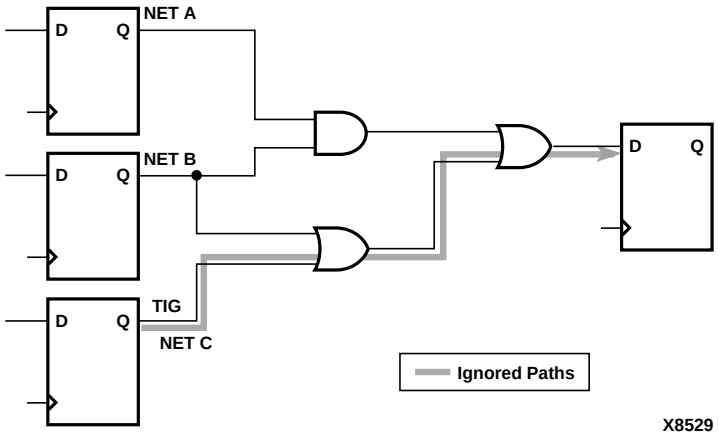


Figure 4-16 TIG Example

The following attribute would be attached to a net to inform the timing analysis tools that it should ignore paths through the net for specification TS43:

Schematic syntax

TIG = TS43

UCF syntax

NET *net_name* **TIG = TS43;**

You cannot perform path analysis in the presence of combinatorial loops. Therefore, the timing tools ignore certain connections to break combinatorial loops. You can use the TIG constraint to direct the timing tools to ignore specified nets or load pins, consequently controlling how loops are broken.

Basic FROM –TO Syntax

Within the TIMESPEC primitive, you use the following syntax to specify timing requirements between specific end points.

TS*identifier***=FROM** *source_group* **TO** *dest_group delay*

TS*identifier***=FROM** *source_group delay*

TS*identifier***=TO** *dest_group delay*

Unspecified FROM or TO, as in the second and third syntax statements, implies all points.

The From-To statements are TS attributes that reside in the TIMESPEC primitive. The parameters *source_group* and *dest_group* must be one of the following.

- Predefined groups
- Previously created TNM identifiers
- Groups defined in TIMEGRP symbols
- TPSYNC groups

Predefined groups consist of FFS, LATCHES, RAMS, or PADS and are discussed in the “Using Predefined Groups” section. TNMs are introduced in the “Creating User-Defined Groups Using TNMs” section. TIMEGRP symbols are introduced in the “Creating New Groups from Existing Groups” section.

Note: Keywords, such as FROM, TO, and TS appear in the documentation in upper case; however, you can enter them in the TIMESPEC primitive in either upper or lower case. You cannot enter them in a combination of lower and upper case.

The *delay* parameter defines the maximum delay for the attribute. Nanoseconds are the default units for specifying delay time in TS attributes. You can also specify delay using other units, such as picoseconds or megahertz.

Refer to the "Specifying Time Delay in TS Attributes" section later in this chapter for more information on time delay. The delay can be a function of another TIMESPEC (TS01*2).

The following examples illustrate the use of From-To TS attributes.

Schematic syntax in TIMESPEC primitive

```
TS01=FROM FFS TO FFS 30
TS_OTHER=FROM PADS TO FFS 25
TS_THIS=FROM FFS TO RAMS 35
TS_THAT=FROM PADS TO LATCHES 35
```

UCF syntax

```
TIMESPEC TS01=FROM FFS TO FFS 30;
TIMESPEC TS_OTHER=FROM PADS TO FFS 25;
TIMESPEC TS_THIS=FROM FFS TO RAMS 35;
TIMESPEC TS_THAT=FROM PADS TO LATCHES 35;
```

You can place TS attributes containing From-To statements in either of two places: in the TIMESPEC primitive on the schematic as discussed in this chapter or in a constraints (UCF) file. See the "Attributes, Constraints, and Carry Logic" chapter of the *Libraries Guide* for more information about specifying timing requirements in a constraints file.

Specifying Timing Points

There are situations where a particular point or set of points in your design needs to be flagged for reference in subsequent timing specifications. *Timing points* are used for these specifications.

There are two types of timing points.

- A TPSYNC timing point is used to allow a point to be used as the start or the end of timing path, even though the point may not apply to a flip-flop, latch, RAM or I/O pad.
- A TPTHRU timing point identifies an intermediate point on a path.

The following sections describe how these timing points are specified in a schematic. The syntax for specifying TPSYNC and TPTHRU constraints in a UCF or NCF constraints file is described in the "Attributes, Constraints, and Carry Logic" chapter of the *Libraries Guide*.

Using TPSYNC to Define Synchronous Points

There are cases where the timing of a design must be defined from or to a point in the design that is not a flip-flop, latch, RAM or I/O pad. For example, you might want to specify a point at the output of a latch defined using a function generator instead of a latch symbol. The TPSYNC timing point identifies one or a group of these points.

A TPSYNC attribute has the following syntax.

Schematic syntax

TPSYNC = *identifier*

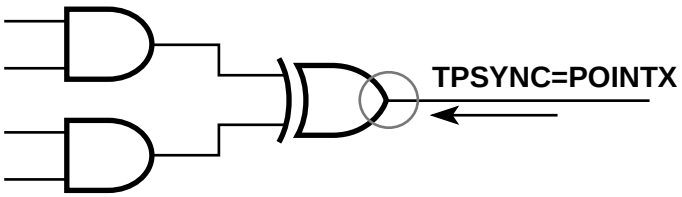
UCF syntax

{**NET** | **PIN** | **INST**} **TPSYNC=** *identifier***;**

identifier is a name that is used in timing specifications in the same way that groups are used. The same identifier can be used on several points which are then treated as a group from the point of view of timing analysis. The *identifier* must be different from any identifier used for a TNM attribute.

The way a TPSYNC timing point is used depends on the object to which it is attached.

- Attached to a net, TPSYNC identifies the source of the net as a potential source or destination for timing specifications.



- Attached to an output macro pin, TPSYNC identifies all of the sources inside the macro that drive the pin to which the attribute is attached as potential sources or destinations for timing specifications. In the following diagram. POINTY applies to the inverter.

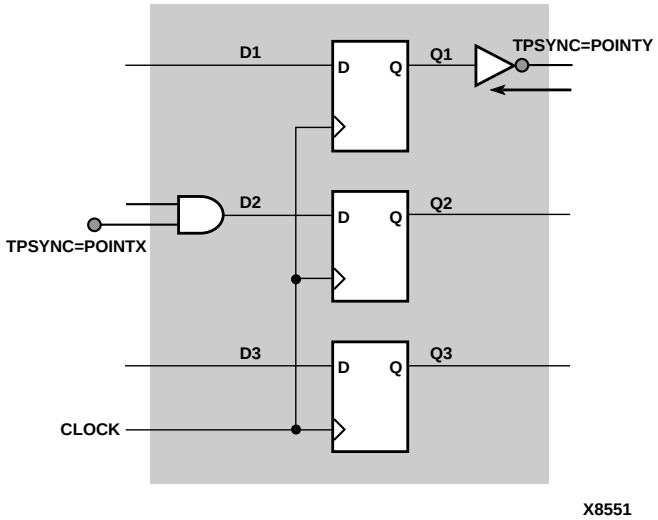
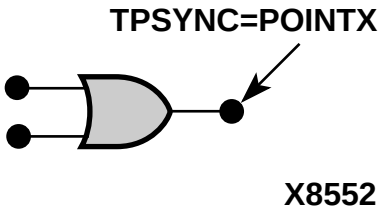


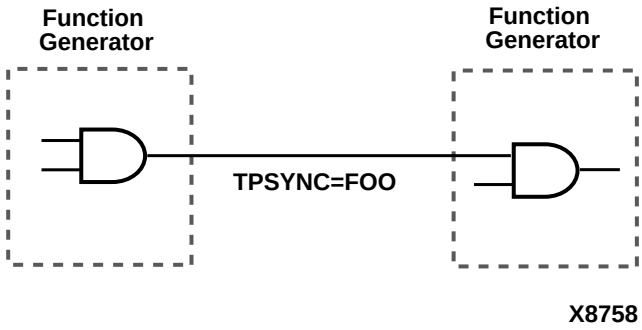
Figure 4-17 TPSYNCs Attached to Macro Pins

If the macro pin is an input pin (that is, there are no sources for the pin in the macro), then all of the load pins in the macro are flagged as synchronous points. In the preceding figure POINTX applies to the input AND gate.

- Attached to a primitive pin, TPSYNC flags the primitive's pin as a potential source or destination for timing specifications; TPSYNC applies to the pin it is attached to.
- Attached to a primitive symbol, TPSYNC identifies the output(s) of that element as a potential source or destination for timing specifications. See the following figure.



The use of a TPSYNC timing point to define a synchronous point in a design implies that the flagged point cannot be merged into a function generator. For example, consider the following diagram.



In this example, because of the TPSYNC definition, the two gates cannot be merged into a single function generator.

Using TPTHRU to Define Through Points

The TPTHRU attribute defines an intermediate point in a path. A point or group defined with TPTHRU attributes is used in detailed timing specifications.

A TPTHRU attribute has the following syntax.

TPTHRU = *identifier*

identifier is a name that is used in timing specifications in the same way that groups are used. The same identifier can be used on several points which are then treated as a group from the point of view of timing analysis.

The *identifier* must be different from any identifier used for a TNM attribute or TPSYNC.

Timing specifications using TPTHRU groups are described in the “Specifying Time Delay in TS Attributes” section.

Using TPTHRU or TPSYNC in a FROM–TO Constraint

It is sometimes convenient to define intermediate points on a path to which a specification applies. This defines the maximum allowable delay and has the following syntax.

Schematic syntax in TIMESPEC primitive

TS*identifier***=FROM** *source_group* **THRU** *thru_point* [**THRU** *thru_point*] **TO** *dest_group* *allowable_delay* [*units*]

TS*identifier***=FROM** *source_group* **THRU** *thru_point* [**THRU** *thru_point*] *allowable_delay* [*units*]

TS*identifier***=THRU** *thru_point* [**THRU** *thru_point*] **TO** *dest_group* *allowable_delay* [*units*]

UCF syntax

TIMESPEC TS*identifier***=FROM** *source_group* **THRU** *thru_point* [**THRU** *thru_point*] **TO** *dest_group* *allowable_delay* [*units*]**;**

TIMESPEC TS*identifier***=FROM** *source_group* **THRU** *thru_point* [**THRU** *thru_point*] *allowable_delay* [*units*]**;**

TIMESPEC TS*identifier***=THRU** *thru_point* [**THRU** *thru_point*] *allowable_delay* [*units*]**;**

Unspecified FROM or TO, as in the second and third syntax statements, implies all points.

- *identifier* is an ASCII string made up of the characters A..Z, a..z, 0..9, underbar (_), and forward slash (/).
- *source_group* and *dest_group* are user-defined, predefined groups or TPSYNCs.
- *thru_point* is an intermediate point used to qualify the path, defined using a TPTHRU attribute.
- *allowable_delay* is the timing requirement.
- *units* is an optional field to indicate the units for the allowable delay. Default units are nanoseconds, but the timing number can be followed by ps, ns, us, ms, GHz, MHz, or KHz to indicate the intended units.

The example shows how to use the TPTHRU attribute with the THRU attribute on a schematic. The UCF syntax is as follows.

```
INST FLOPA TNM=A;
INST FLOPB TNM=B;
NET MYNET TPTHRU=ABC
TIMESPEC TSpath1=FROM A THRU ABC TO B 30;
```

The following schematic shows the placement of the TPTHRU attribute and the resultant path that is defined.

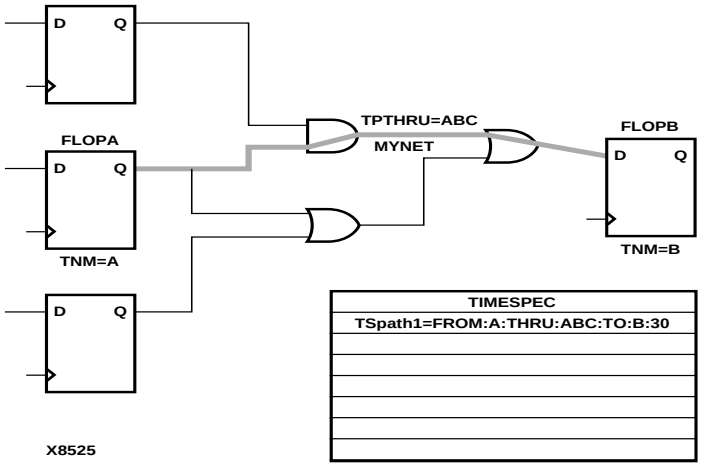


Figure 4-18 TPTHRU Example

Specifying Time Delay in TS Attributes

Nanoseconds are the default units for specifying delay times in TS attributes. However, after specifying the maximum delay or minimum frequency numerically, you can enter the unit of measure by specifying the following.

- PS for picoseconds, NS for nanoseconds, US for microseconds, or MS for milliseconds
- MHZ for megahertz, KHZ for kilohertz, or GHz for gigahertz

As an alternate way of specifying time delay, you can specify one time delay in terms of another. Instead of specifying a time or frequency in a TS attribute definition, you can specify a multiple or division of another TS attribute. This is useful in a system where all clocks are derived from a master clock; in this situation, changing the timing specification for the master clock changes the specification for all clocks in the system.

Use the syntax below to specify a TS attribute delay in terms of another.

Schematic syntax attached to TIMESPEC primitive

TS*identifier***=***specification reference_TS_attribute*[{***** | **/**}*number*]

UCF syntax

TIMESPEC TS*identifier***=***specification***:***reference_TS_attribute*[{***** | **/** }*number*]**;**

number can be either a whole number or a decimal. The *specification* can be any From-To statement as illustrated by the following examples.

```
FROM PADS TO PADS
FROM group1 TO group2
FROM tnm_identifier TO FFS
FROM LATCHES TO group1
```

Use "*" to represent multiplication and "/" to represent division. The specification type of the reference TS attribute does not need to be the same as the TS attribute being defined; however, it must not be specified in terms of TIG.

Examples

Examples of specifying a TS attribute in terms of another are as follows. In these cases, assume that the reference attributes were specified as delays (not frequencies).

In the example below, the paths between flip-flops and pads are placed and routed so that their delay is at most 10 times the delay specified in the TS05 attribute.

Schematic syntax in TIMESPEC primitive

```
TS08=FROM FFS TO PADS TS05*10
```

UCF syntax

```
TIMESPEC TS08=FROM FFS TO PADS TS05*10;
```

In the example below, the paths between input and output pads are placed and routed so that their delay is at most one-eighth the delay specified in the TS07 attribute.

Schematic syntax in TIMESPEC primitive

```
TS1=FROM PADS TO PADS TS07/8
```

UCF syntax

```
TIMESPEC TS1=FROM PADS TO PADS TS07/8;
```

Note: When a reference attribute is specified as a frequency, a multiple represents a faster specification; a division represents a slower specification.

You can also specify a TS attribute in terms of a TS attribute that is already a specification of another. The following example provides an illustration.

Schematic syntax in TIMESPEC primitive

```
TS09=FROM FFS TO FFS 50
TS10=FROM FFS TO PADS TS09*2
TS11=FROM PADS TO PADS TS10*4
```

UCF syntax

```
TIMESPEC TS09=FROM FFS TO FFS 50;
TIMESPEC TS10=FROM FFS TO PADS TS09*2;
TIMESPEC TS11=FROM PADS TO PADS TS10*4;
```

Using the PRIORITY Keyword

There may be situations where there is a conflict between two TIMESPECs that cover the same path. In these cases you can define the priority of a TIMESPEC using the following syntax.

normal_timespec_syntax **PRIORITY** *integer*

normal_timespec_syntax is a legal TIMESPEC and *integer* represents the priority (the smaller the number, the higher the priority). The number can be positive, negative, or zero, and the value only has meaning when compared with other PRIORITY values.

Sample Schematic Using TIMESPEC/TIMEGRP Symbol

TNM identifiers define symbols or groups of symbols that are used in timing specifications. They can also define other groups. The following figure shows an example of a TNM attribute attached to an individual symbol. In this circuit, the flip-flop D_FF has the attribute TNM=D_FF attached to it.

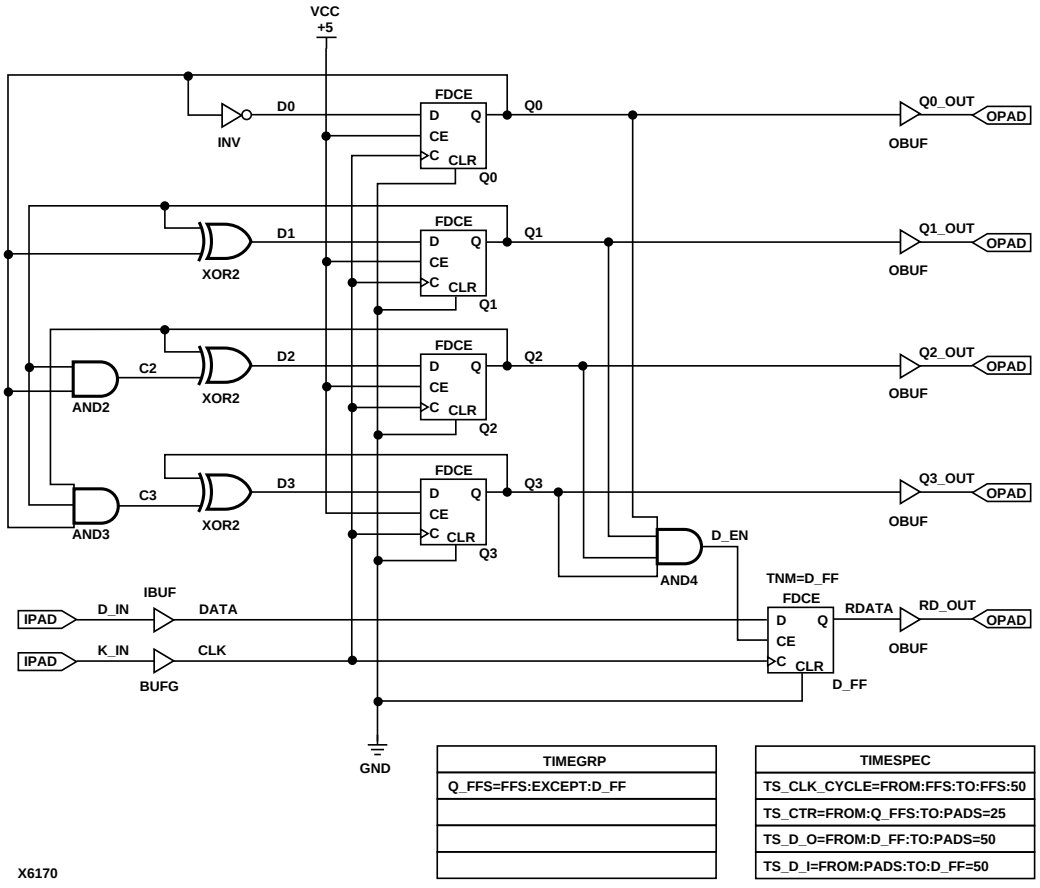


Figure 4-19 Example of Using TNMs and TIMEGRPs in Your Schematic

The TIMEGRP symbol contains an attribute that defines a group of flip-flops called Q_FFS, which includes all flip-flops in the schematic except the one labeled D_FF. You can then use the group Q_FFS to create timing specifications in the TIMESPEC primitive. The flip-flop D_FF has its clock enable driven at 1/2 of the clock frequency; therefore, its flip-flop to pad and pad to flip-flop timing specifications are longer than the flip-flop to pad specifications in the Q_FFS group.

Prorating Constraints

The prorating constraints, VOLTAGE and TEMPERATURE, provide a method for determining timing delay characteristics based on known environmental parameters. On a schematic, you can enter these constraints in any empty space. For Release 1.5 these two constraints are supported only for the XC4000XL. New speed file releases for existing architectures will support these two constraints.

VOLTAGE Constraint

This constraint allows the specification of the operating voltage. This provides a means of prorating delay characteristics based on the specified voltage.

Note: Each architecture has its own specific range of supported voltages. If the entered voltage does not fall within the supported range, the constraint is ignored and an architecture-specific default value is used instead. The UCF syntax is as follows.

VOLTAGE=*value*[*units*]

value is an integer or real number specifying the voltage and units is an optional parameter specifying the unit of measure. V specifies volts, the default voltage unit.

TEMPERATURE Constraint

This constraint allows the specification of the operating temperature which provides a means of prorating device delay characteristics based on the specified junction temperature. Prorating is a linear scaling operation on existing speed file delays and is applied globally to all delays.

Note: Each architecture has its own specific range of valid operating temperatures. If the entered temperature does not fall within the supported range, the constraint is ignored and an architecture-specific default value is used instead. The UCF syntax is as follows.

TEMPERATURE=*value*[**C** | **F** | **K**]

value is an integer or a real number specifying the temperature. C, K, and F are the temperature units: F is degrees Fahrenheit, K is degrees Kelvin, and C is degrees Celsius, the default.

Additional Timing Constraints

There are additional properties and constraints you can specify for the timing analysis tools. They are the following.

- Net skew control (MAXSKEW)
- Net delay control
- Path tracing control
- The DROP_SPEC constraint

Controlling Net Skew (MAXSKEW)

Skew is the difference between the minimum and maximum of the maximum load delays on a net. You can control the maximum allowable skew on a net by attaching the MAXSKEW attribute directly to the net. Syntax is as follows.

skew_item **MAXSKEW=***allowable_skew* [*units*]**;**

allowable_skew is the timing requirement.

The default units for *allowable_skew* are nanoseconds, but the timing number can be followed by ps, ns, us, ms, GHz, MHz, or KHz to indicate the intended units.

skew_item is one of the following,

- NET "*net_name*"
- TIMEGRP "*group_name*" (PCF only)
- ALLCLOCKNETS (PCF only)

Note: TIMEGRP and ALLCLOCKNETS are supported in PCF files only.

It is important to understand exactly what MAXSKEW defines. Consider the following example.

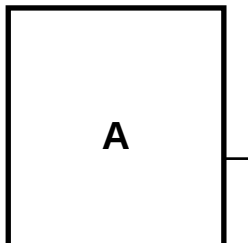


Figure 4-20 MAXSKEW

In the preceding diagram, for t_a(1,2), 1 ns is the minimum delay and 2 ns is the maximum delay for the Register A clock. For t_b(2,4), 2 ns is the minimum delay and 4 ns is the maximum delay for the Register B clock. MAXSKEW defines the maximum of t_b minus the maximum of t_a, that is, 4-2=2. Since the data delay is greater than MAXSKEW (DD is 2.5 while MAXSKEW is 2), no race condition occurs. However, MAXSKEW does not account for the circumstance where one of the registers is operating at minimum delay (for example, t_a=1) while a second register is operating at maximum delay (for example, t_b=4). Under those conditions, the skew is 3 ns (t_b - t_a= 3). Since the data delay (DD = 2.5) is less than the skew, a race condition exists.

Controlling Net Delay (MAXDELAY)

You can control the maximum allowable delay on a net by attaching the MAXDELAY attribute directly to the net. The UCF syntax is as follows.

NET *net_name* **MAXDELAY=***path_value* [**PRIORITY** *integer*];

TS*identifier***=MAXDELAY=** *path path_value* [**PRIORITY** *integer*];

path **MAXDELAY=***path_value* [**PRIORITY** *integer*];

net_delay_item **MAXDELAY**=*delay_time* [*units*] [**PRIORITY** *integer*];

path is one of the following,

- PATH *"path_name"*
- ALLPATHS
- FROM *group_item* THRU *group_item1... group_itemn*

- FROM *group_item* THRU *group_item1... group_itemn* TO *group_item*
- THRU *group_item1... group_itemn* TO *group_item*

path_value is one of the following,

- *delay_time [units]*

  units defaults to nanoseconds, but the delay time number can be followed by ps, ns, us, or ms (picoseconds, nanoseconds, microseconds, or milliseconds) to specify the units

- *frequency units*

  units can be specified as GHz, MHz, or KHz (gigahertz, megahertz, or kilohertz*)*

- TS*identifier* [{ / |*} *real_number*]

net_delay_item is one of the following.

- NET "*net_name*"
- TIMEGRP "*group_name*"
- ALLCLOCKNETS

Controlling Path Tracing

Path tracing controls allows you to enable or disable specific paths within device components (for example, CLBs and IOBs) for timing analysis. *These constraints can only be entered in a PCF file*; they cannot be applied during design entry or in a UCF or NCF file.

This constraint can be applied at a global or group scope. The path tracing syntax is as follows.

[**TIMEGRP** *predefined_group*] {**ENABLE** | **DISABLE**} **=** *symbol***;**

symbol is a component delay symbol, and *predefined_group* (which is optional) represents the name of a previously-defined time group. If there is no TIMEGRP *predefined_group* qualifier, the path tracing control applies to all logic cells in the design.

The *symbol*, which is case-insensitive, can be either of the following.

- A standard component delay symbol name (for example, reg_sr_q or tbuf_i_o, as described in the following table).

There is a one-to-many correspondence between these symbol names and data book symbol names, and the data book symbols to which each standard block delay signal applies varies from one device family to another.

- A component delay specified in the *Xilinx Programmable Logic Data Book* (for example, T_{ILO} (entered as TILO) or T_{CCK} (entered as TCCK)).

The following table describes the standard block delay symbols.

Table 4-1 Standard Block Delay Symbols for Path Tracing

Symbol	Path Type	Default
reg_sr_q	Set/Reset to output propagation delay	Disabled
lat_d_q	Data to output transparent latch delay	Disabled
ram_d_o	RAM data to output propagation delay	Disabled
ram_we_o	RAM write enable to output propagation delay	Enabled
tbuf_t_o	TBUF tristate to output propagation delay	Enabled
tbuf_i_o	TBUF input to output propagation delay	Enabled
io_pad_i	IO pad to input propagation delay	Enabled
io_t_pad	IO tristate to pad propagation delay	Enabled
io_o_i	IO output to input propagation delay. Disabled for tristated IOBs.	Enabled
io_o_pad	IO output to pad propagation delay	Enabled

The IOB configuration for Virtex is somewhat different than the IOB configuration for other architectures. See the following figure.

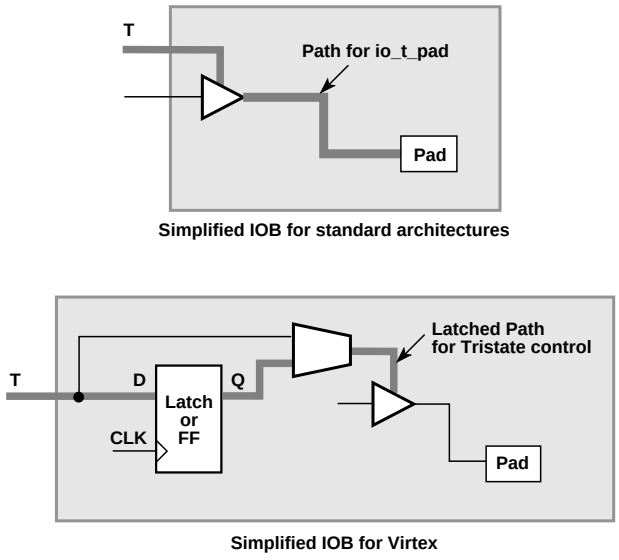


Figure 4-21 Simplified IOB Configurations and io_t_pad

For the Virtex IOB, there is no default path. If a latch is used (latch mode), then io_t_pad controls the D to Q path through the latch. By default D to Q is enabled which is different than other internal latches. The clock to Q of the latch is not disabled by io_t_pad.

If a register is used instead of a latch, the clock to Q of the register is not disabled by io_t_pad.

Path Tracing Examples

The PCF file constraint below prevents timing analysis on any path that includes the I to O delay on a TBUF component. The constraint applies to all TBUF components in the design.

```
DISABLE = "tbuf_i_o";
```

The PCF file constraint below disables the I to O delay on the TBUF components in the group mygroup, if applicable.

```
TIMEGRP "mygroup" DISABLE = "tbuf_i_o";
```

The PCF file constraint below disables the T_{ILO} databook component delay in the group mygroup, if applicable.

```
TIMEGRP "mygroup" DISABLE = "TILO";
```

The delay symbol names in the *Xilinx Programmable Logic Data Book* do not always agree with the delay names reported in TRACE (the Xilinx timing analyzer). To ensure your path tracing constraints are processed correctly and to allow your constraints to be portable from one device to another, use the delay names reported by TRACE instead of the databook names.

You can control path tracing for a single instance by creating a group containing only the instance, then specifying this group in a path tracing constraint.

The DROP_SPEC Constraint

A constraint specified in a UCF constraints file takes precedence over one with the same name in the input design. This allows you to redefine or modify constraints without having to edit the input design. The DROP_SPEC constraint allows you to specify that a timing constraint defined in the input design should be dropped from the analysis. The UCF syntax is as follows.

TS *identifier* **= DROP_SPEC**

identifier is the identifier name used with another timing specification. This constraint can be used when new specifications defined in a constraints file do not directly override all specifications defined in the input design, and some of these input design specifications need to be dropped.

While this timing command is not expected to be used much in an input netlist (or NCF file), it is not illegal. If defined in an input design this attribute must be attached to a TIMESPEC primitive.

Constraints Priority

In some cases, two timing specifications cover the same path. For cases where the two timing specifications on the path are mutually exclusive, the following constraint rules apply.

- Priority depends on the file in which the constraint appears. A constraint in a file accessed later in the design flow replaces a constraint in a file accessed earlier in the design flow. Priority is as follows (first listed is the highest priority, last listed is the lowest).
  - Constraints in a Physical Constraints File (PCF)
  - Constraints in a User Constraints File (UCF)
  - Constraints in a Netlist Constraints File (NCF)
  - Attributes in a schematic
- If two timing specifications cover the same path, the priority is as follows (first listed is the highest priority, last listed is the lowest).
  - Timing Ignore (TIG)
  - FROM:THRU:TO specifications
  - FROM:TO specifications
  - PERIOD specifications
  - ALLPATHS type specifications (in PCF file only).
- FROM:THRU:TO or FROM:TO statements have a priority order that depends on the type of source and destination groups included in a statement. The priority is as follows (first listed is the highest priority, last listed is the lowest).
  - Both the source group and the destination group are user-defined groups
  - Either the source group or the destination group is a predefined group
  - Both the source group and the destination group are predefined groups

Net delay and Net skew specifications are analyzed independently of path delay analysis and do not interfere with one another.

If two constraints are in the same category, the user-defined priority described in the“Using the PRIORITY Keyword” section is used to determine which constraint takes precedence.

Syntax Summary

The following sections summarize the syntax for timing constraints.

TNM Attributes

The following table lists the syntax used when creating TNMs, which you enter directly on the primitive symbol, macro symbol, net, or driver pin.

TNM Attribute Syntax	Where Applied
Schematic syntax: **TNM=***identifier* **TNM=***predefined_group identifier* UCF syntax: {**NET** | **PIN** | **INSTANCE**} *name* **TNM=***identifier* {**NET** | **PIN** | **INSTANCE**} *name* **TNM=***predefined_group***:***identifier***;**	Net, Symbol, Pin, Macro

TIMEGRP Attributes

The following table lists the syntax used with the TIMEGRP primitive.

Group Type	TIMEGRP Attribute Syntax
Combine	Schematic syntax in TIMEGRP primitive: *new_group*=*group1 group2* [*group3* . . .] UCF syntax: **TIMEGRP** *new_group*=*group1*:*group2* [*group3* . . .];
Exclude	Schematic syntax in TIMEGRP primitive: *new_group*=*group1*[:*group2* . . .] **EXCEPT** *group3*[*group4* ...] UCF syntax: **TIMEGRP** *new_group*=*group1*[:*group2* . . .] **EXCEPT** *group3*[*group4* ...];
Clock Edge (flip-flops)	Schematic syntax in TIMEGRP primitive: *new_group*=**RISING** *group1* *new_group*=**FALLING** *group1* UCF syntax: **TIMEGRP** *new_group*=**RISING** *group1*; **TIMEGRP** *new_group*=**FALLING** *group1*;
Gate Edge (latches)	Schematic syntax in TIMEGRP primitive: *new_group*=**TRANSHI** *group1* *new_group*=**TRANSLO** *group1* UCF syntax: **TIMEGRP** *new_group*=**TRANSHI** *group1*; **TIMEGRP** *new_group*=**TRANSLO** *group1*;

Group Type	TIMEGRP Attribute Syntax
Pattern Matching	Schematic syntax in TIMEGRP primitive: *new_group*=*predefined_group* **(***name_qualifier1*[*name_qualifier2* . . .]**)** UCF syntax: **TIMEGRP** *new_group*=*predefined_group* **(***name_qualifier1*[*name_qualifier2* . . .]**);**
Net-specific OFFSETs	Schematic syntax when attached to a net: **OFFSET =** {**IN**|**OUT**} *offset_time* [*units*] {**BEFORE**|**AFTER**} *clk_name* [**TIMEGRP** *group_name*] UCF syntax: **NET** *name* **OFFSET =** {**IN**|**OUT**} *offset_time* [*units*] {**BEFORE**|**AFTER**} *clk_name* [**TIMEGRP** *group_name*]**;** PCF syntax: **COMP "***iob_name***" OFFSET =** {**IN**|**OUT**} *offset_time* [*units*] {**BEFORE**|**AFTER**} **COMP** "*clk_iob_name*" [**TIMEGRP** "*group_name*"]**;**

TIMESPEC Attributes

The following table lists the syntax used for parameters that define TS attributes, which reside in the TIMESPEC primitive or appear in UCF or NCF files.

Spec Type	TS Attribute Syntax
Basic From-To	Schematic syntax in TIMESPEC primitive: **TS***id***=FROM** *source_group* **TO** *dest_group delay* **TS***id***=FROM** *source_group delay* **TS***id***=TO** *dest_group delay* UCF syntax: **TIMESPEC TS***id***=FROM:***source_group* **TO** *dest_group delay***;** **TIMESPEC TS***id***=FROM** *source_group delay***;** **TIMESPEC TS***id***=TO** *dest_group delay***;**
Ignore	Schematic syntax in TIMESPEC primitive: **TS***id***=IGNORE** UCF syntax: **TIMESPEC TS***id***=IGNORE;**
Through point	Schematic syntax in TIMESPEC primitive: **TS***id***=FROM** *source_group* **THRU** *thru_point*[**THRU** *thru_point*] **TO** *dest_group delay* **TS***id***=FROM** *source_group* **THRU** *thru_point*[**THRU** *thru_point*] *delay* **TS***id***=THRU** *thru_point*[**THRU** *thru_point*] **TO** *dest_group delay* UCF syntax: **TIMESPEC TS***id***=FROM***source_group***:THRU** *thru_point*[**THRU** *thru_point*] **TO** *dest_group delay***;** **TIMESPEC TS***id***=FROM** *source_group* **THRU** *thru_point*[**THRU** *thru_point*] *delay***;** **TIMESPEC TS***id***=THRU** *thru_point*[**THRU** *thru_point*] **TO** *dest_group delay***;**

Spec Type	TS Attribute Syntax
Linked specification	Schematic syntax in TIMESPEC primitive: **TS***id***=FROM** *source_group* **TO** *dest_group another_TSid* [***** | **/**]*number* **TS***id***=FROM** *source_group another_TSid* [***** | **/**]*number* **TS***id***=TO** *dest_group another_TSid*[***** | **/**]*number* UCF syntax: **TIMESPEC TS***id***=FROM** *source_group* **TO** *dest_group another_TSid* [***** | **/**]*number***;** **TIMESPEC TS***id***=FROM** *source_group another_TSid* [***** | **/**]*number***;** **TIMESPEC TS***id***=TO** *dest_group another_TSid*[***** | **/**]*number***;**
Clock period	Schematic syntax in TIMESPEC primitive: **TS***id***=PERIOD** *TNM_reference period* {**HIGH** | **LOW**} [*high_or_low_time*] UCF syntax: **TIMESPEC TS***id***=PERIOD** *TNM_reference period***:**{**HIGH** | **LOW**} [*high_or_low_time*]**;**
Derived clocks	Schematic syntax in TIMESPEC primitive: **TS***id***=PERIOD** *TNM_reference another_PERIOD_identifier* [**/** | *****]*number*{**HIGH** | **LOW**} [*high_or_low_time*] UCF syntax: **TIMESPEC TS***id***=PERIOD:***TNM_reference another_PERIOD_identifier* [**/** | *****]*number*{**HIGH** | **LOW**} [*high_or_low_time*]**;**

Spec Type	TS Attribute Syntax
TS attribute priority	*normal_timespec_syntax* **PRIORITY** *integer*
Group OFFSETs	Schematic syntax in TIMESPEC primitive: **TS***identifier***=TIMEGRP** *name* **OFFSET=** {**IN**|**OUT**} *offset_time* [*units*] {**BEFORE**|**AFTER**} *clk_name* [**TIMEGRP** *group_name*] The UCF and PCF syntax do not require the **TS***identifier*. UCF syntax: [**TIMEGRP** *name*] **OFFSET=** {**IN**|**OUT**} *offset_time* [*units*] {**BEFORE**|**AFTER**} *clk_name* [**TIMEGRP** *group_name*]**;** PCF syntax: [**TIMEGRP** *name*] **OFFSET=** {**IN**|**OUT**} *offset_time* [*units*] {**BEFORE**|**AFTER**} **COMP** *clk_iob_name* [**TIMEGRP** *group_name*]**;**

The following table lists additional attributes or constraints that are used in or affect TS attributes.

Attribute Syntax	Where Applied	How Used
Schematic syntax on net, pin, symbol, or macro: **TPTHRU=***identifier* UCF syntax: {**NET** | **PIN** | **INSTANCE**} *name* **TPTHRU=***identifier***;**	Net, symbol, pin, macro	In through point TS attribute
Schematic syntax on net, pin, symbol, or macro: **TPSYNC=***identifier* UCF syntax: {**NET** | **PIN** | **INSTANCE**} *name* **TPSYNC=***identifier***;**	Net, symbol, pin, macro	As group in TS attribute

Attribute Syntax	Where Applied	How Used
Schematic syntax on net or pin: **TIG** **TIG=***identifier* UCF syntax: {**NET** | **PIN**} *name* **TIG;** {**NET** | **PIN**} *name* **TIG=***identifier***;**	Net, pin	Prevents timing analysis
TS*identifier***=DROP_SPEC;** (Constraints file only)	N / A	Prevents timing analysis for **TS***identifier*

Other Constraints

The following table lists additional timing constraints.

Attribute Syntax	Where Applied	How Used
Schematic syntax on net or pin: **PERIOD** *period* {**HIGH** | **LOW**} [*high_or_low_time*] UCF syntax: {**NET** | **PIN**} *name* **PERIOD** *period* {**HIGH** | **LOW**} [*high_or_low_time*]**;**	Nets, pins	Specifies register clock period

Attribute Syntax	Where Applied	How Used
Schematic syntax: **MAXSKEW=***allowable_skew* UCF syntax: **NET** *name* **MAXSKEW=***allowable_skew***;** PCF Syntax: {**NET** | **TIMEGRP** | **ALLCLOCKNETS**} *name* **MAXSKEW=***allowable_skew***;**	Nets, timegroups, ALLCLOCKNETS	Specifies skew

Attribute Syntax	Where Applied	How Used
Schematic syntax: **MAXDELAY=** *path_value* [**PRIORITY** *integer*] UCF syntax: **NET** *net_name* **MAXDELAY=** *path_value* [**PRIORITY** *integer*]; PCF syntax: **TS***identifier*=**MAXDELAY** *path path_value* [**PRIORITY** *integer*]; {**NET** | **TIMEGRP** | **ALLCLOCKNETS**} *name* **MAXDELAY=** *path_value* [**PRIORITY** *integer*]**;** **PATH** *path_name* **MAXDELAY=** *path_value* [**PRIORITY** *integer*]**;** **ALLPATHS MAXDELAY=** *path_value* [**PRIORITY** *integer*]**;** **FROM** *group_item* **THRU** *group_item1... group_itemn* **MAXDELAY=** *path_value* [**PRIORITY** *integer*]**;** **FROM** *group_item* **THRU** *group_item1... group_itemn* **TO** *group_item* **MAXDELAY=** *path_value* [**PRIORITY** *integer*]**;** **THRU** *group_item1... group_itemn* **TO** *group_item* **MAXDELAY=** *path_value* [**PRIORITY** *integer*]**;**	Nets, Paths, FROM:THRU, FROM:THRU:TO, THRU:TO	Specifies delay