Using Timing Constraints

Preface

About This Manual

This manual describes how to specify timing constraints.

Before using this manual, you should be familiar with the operations that are common to all Xilinx's software tools: how to bring up the system, select a tool for use, specify operations, and manage design data. These topics are covered in the *Development System Reference Guide*.

Conventions

Typographical

This manual uses the following conventions. An example illustrates each convention.

- `Courier font` indicates messages, prompts, and program files that the system displays.

  ```
  speed grade: -100
  ```

- **`Courier bold`** indicates literal commands that you enter in a syntactical statement.

  `rpt_del_net=`

  `Courier bold` also indicates commands that you select from a menu.

  `File` → **`Open`**

- *Italic font* denotes the following items.

  - Variables in a syntax statement for which you must supply values

    `edif2ngd` *design_name*

  - References to other manuals

    See the *Development System Reference Guide* for more information.

  - Emphasis in text

    If a wire is drawn so that it overlaps the pin of a symbol, the two nets are *not* connected.

- Square brackets “[ ]” indicate an optional entry or parameter. However, in bus specifications, such as bus [7:0], they are required.

  `edif2ngd` [*option_name*] *design_name*

  Square brackets also enclose footnotes in tables that are printed out as hardcopy in DynaText®.

- Braces “{ }” enclose a list of items from which you choose one or more.

  `lowpwr ={on|off}`

- A vertical bar “ | ” separates items in a list of choices.

  `symbol` *editor_name* `[bus|pins]`

- A vertical ellipsis indicates repetitive material that has been omitted.

  ```
  IOB #1: Name = QOUT’
  IOB #2: Name = CLKIN’
  .
  .
  .
  ```

- A horizontal ellipsis “. . .” indicates that an item can be repeated one or more times.

  `allow block` *block_name loc1 loc2* `...` *locn;*

Contents

Using Timing Constraints

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

- XC3000A / L
- XC3100A / L
- XC4000E / L
- XC4000EX / XL / XV
- XC5200
- Spartan

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

- "Timing Requirements and Xilinx Software" section
- "Timing Specifications" section
- "Additional Timing Constraints" section
- "Constraints Priority" section
- "Syntax Summary" section
- "Specialized Support for Synopsys" section

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, PAR maps, places, and routes your design based on these requirements.

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

Timing Specifications

Timing Specifications specify timing requirements for your design.

The following sections describe how you use timing specifications.

- How to enter timing specifications
- Specifying Groups in timing specifications
- Defining timing specifications

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 program (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 uppercase 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.

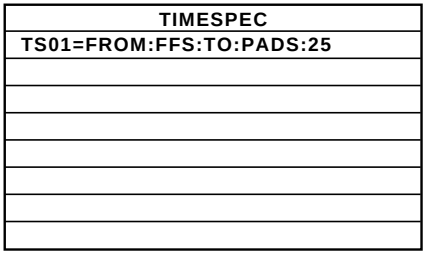


Figure 0-1 TIMESPEC Primitive

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:

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

where **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 uncacceptable keywords are: From, To, fRoM, tO.

Note: The Mentor netlist writer program (ENWRITE) converts all property names to lowercase letters, and the Xilinx netlist reader EDIF2NGD then converts the property names to uppercase letters. To ensure references from one constraint to another are processed correctly, a **TS***identifier* name should contain only uppercase 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 uppercase letters. For example, the TSID1 in the second constraint below must be entered in uppercase 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 TIMESPEC Syntax" section. More detailed forms of the attribute are also described in that section.

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 TIMESPEC Syntax" section.

Specifying Groups in TS Attributes

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
LATCHES	CLB or IOB latches only; not latches built from function generators
PADS	Input/output pads
RAMS	For architectures with RAMS

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:

```
TS01=FROM:FFS:TO:FFS:30
TS02=FROM:LATCHES:TO:LATCHES:25
TS03=FROM:PADS:TO:RAMS:70
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* **])**

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

Note: In cases where a BLKNM or HBLKNM constraint is used, the pad name will not reflect the net name placed between the external pad symbol and the IO primitive(s).

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:

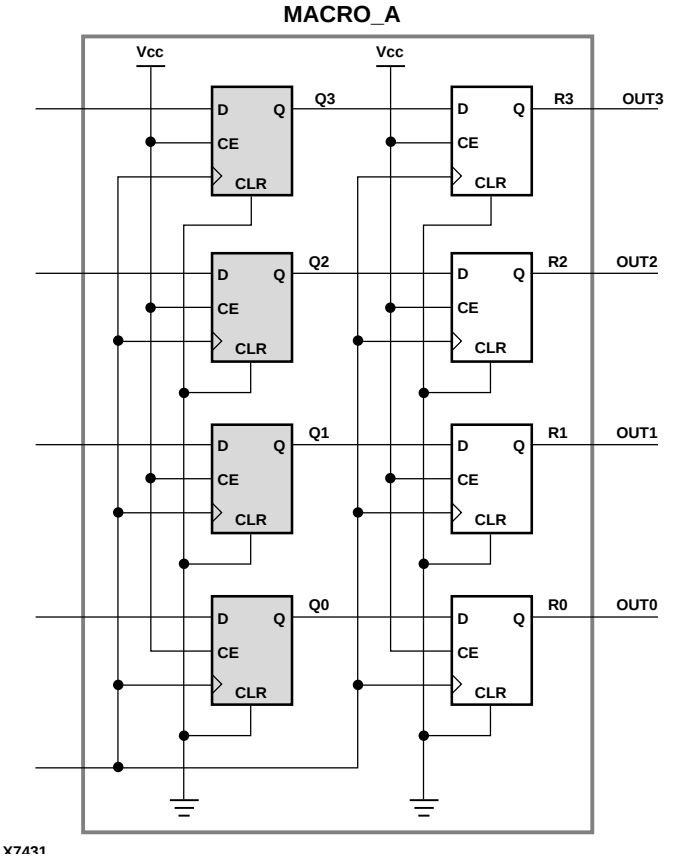


Figure 0-2 Using Qualifiers with Predefined Groups

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 flag 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 using the following syntax:

TNM=*identifier*

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

Warning: 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
CLBNM	H_SET	RLOC_RANGE
CMOS	INIT	SCHNM
CYMODE	INTERNAL	SLOW
DECODE	LIBVER	SYSTEM
DEF	LOC	TNM
DIVIDE1_BY	LOWPWR	TRIM
DIVIDE2_BY	MAP	TS
DOUBLE	MEDFAST	TTL
EQN	MEDSLOW	TYPE
FAST	MINIM	USE_RLOC
		U_SET

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:

TNM=*predefined_group***:***identifier*

where *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, 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. This mechanism is called forward tracing.

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:

```
TNM=RAMS:FIFO_CORE
```

A predefined 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.

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.

Placing TNMs on Primitive Symbols

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

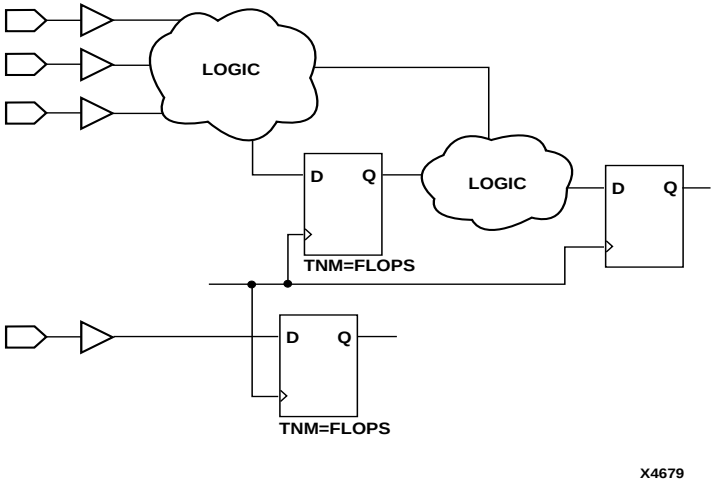


Figure 0-3 TNM on Primitive Symbols

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, load pin, or macro load pin. If you want to assign more than one identifier to the same symbol, include all identifiers on the right side of the equal sign (=) separated by a semicolon (;), as follows:

```
TNM=joe;fred
```

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:

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

If you want to place an identifier on more than one symbol type, separate each symbol type and identifier with a semicolon (;) as illustrated by the following examples:

```
TNM=FFS:FLOPS;PADS:OPADS
TNM=RAMS:MEMS;LATCHES:INLATS
```

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" and all RAMs are

tagged with the TNM ‘‘MEM.” By tagging the macro symbol, you do not have to tag each underlying symbol individually.

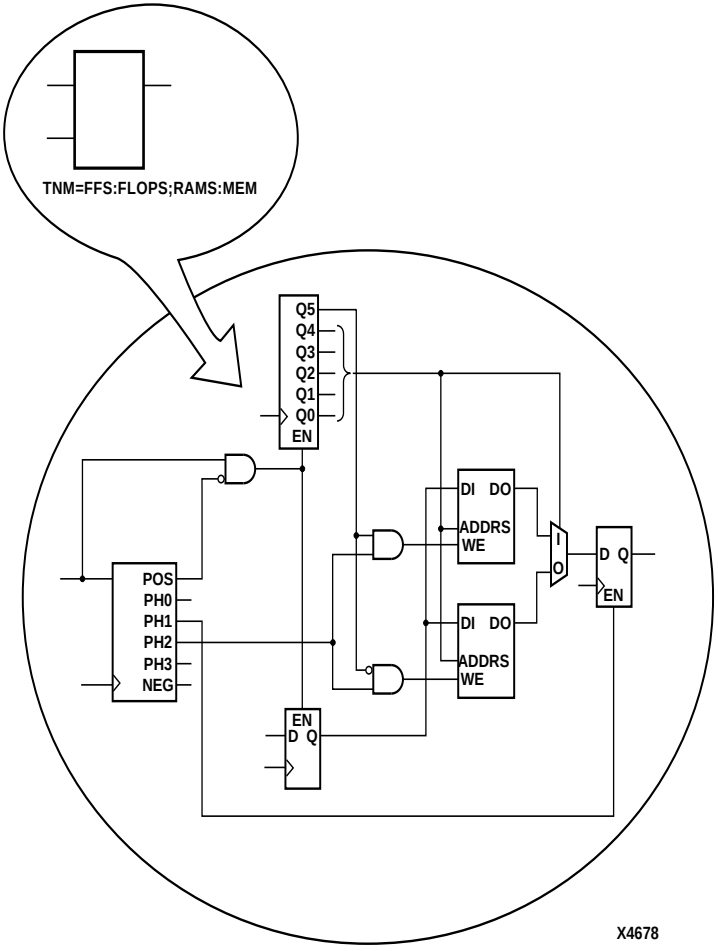


Figure 0-4 TNM on Macro Symbol

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 load 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.

Placing a TNM on a net is equivalent to placing that TNM attribute on every load pin of the net. Use pin TNM attributes when you need finer control.

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

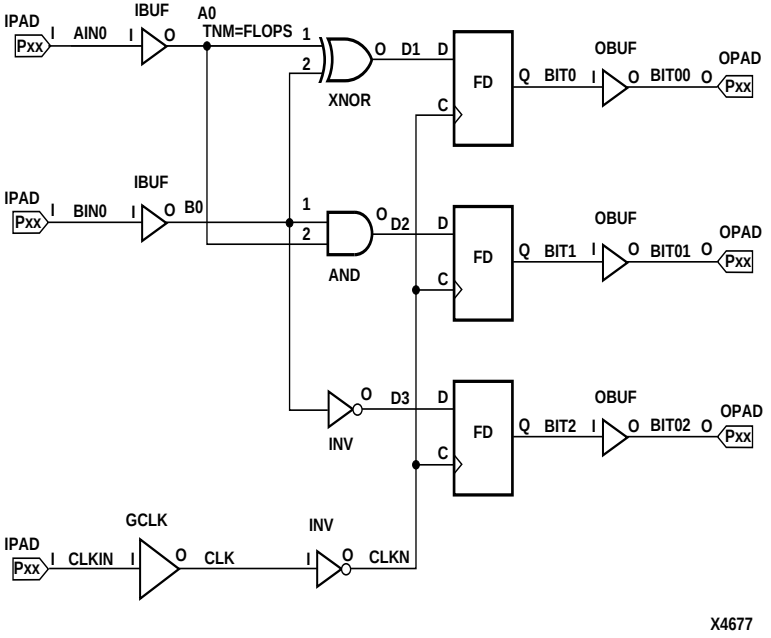


Figure 0-5 TNM on Net Used to Group Flip-Flops

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:

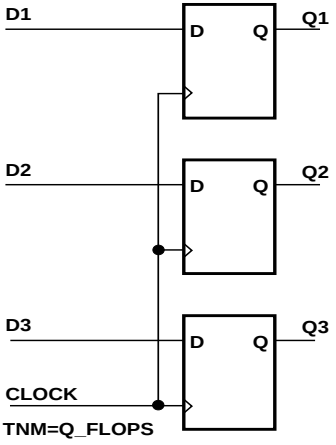


Figure 0-6 TNM on Clock Pin Used to Group Flip-Flops

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

For example:

```
TNM=FFS:data
TNM=RAMS:fifo
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 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 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:

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

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

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

- Group names should contain only uppercase 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 uppercase letters. For example, the GROUP3 in the first constraint below must be entered in uppercase letters to match the GROUP3 in the second constraint.

```
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.

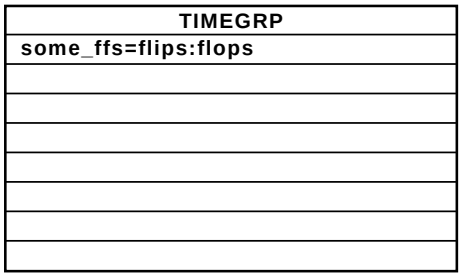


Figure 0-7 TIMEGRP Primitive

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 TNMs 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:

```
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.

```
many_ffs=ffs1:ffs2
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:

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.

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.

group1=RISING:ffs
group2=RISING:*ffs_group*
group3=FALLING:ffs
group4=FALLING:*ffs_group*

where *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 TIMESPEC 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.

```
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:

```
lowgroup=TRANSLO:latchgroup
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:

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

where *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 unless you placed a BLKNM or HBLKNM on the pad in the schematic; in this case, you should specify its value instead.

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

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

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

```
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":

```
some_latches=latches($1I3/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:

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:

```
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”

Timing Points

There are situations where a particular point or set of points in your design need 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:

TPSYNC = *identifier*

where *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 a 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.

  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.
- Attached to a primitive pin, TPSYNC flags the primitive's input as a potential source or destination for timing specifications.
- Attached to a primitive symbol, TPSYNC identifies the output(s) of that element as a potential source or destination for timing specifications.

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.

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*

where *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.

Timing specifications using TPTHRU groups are described in the "Defining Intermediate Points on a Path" section

Basic TIMESPEC 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*

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

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 following examples illustrate the use of From-To TS attributes:

```
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
```

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.

Defining Intermediate Points on a Path

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:

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

- *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 or predefined groups.
- *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.

Ignoring Selected Paths

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.

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:

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.

To specify that a path should be ignored only when the path appears in certain TIMESPECS, use the following form of the TIG attribute:

TIG = TS*identifier*

where *identifier* indicates the TIMESPEC for which paths through the object should be ignored. If a path in another TIMESPEC passes through the object, this path is still available for timing analysis.

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:

```
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.

Note: Previous versions of the Xilinx Development System used an IGNORE in the following syntax to specify ignored nets:

TS*identifier***=IGNORE**

This syntax is supported in this release, but it is not the recommended method of ignoring nets. Use the TIG attribute instead.

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 or KHZ for kilohertz

As an alternate way of specifying time delay, you can specify one time delay in terms of another. This method is described in the next section.

Specifying a TS Attribute 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.

TS*identifier***=***specification***:***reference_TS_attribute***[*|/]***number*

where *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 or IGNORE.

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.

```
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.

```
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.

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

Setting TIMESPEC Priorities

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

normal_timespec_syntax : **PRIORITY :** *integer*

where *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.

See the "Constraints Priority" section for more details.

Defining a Clock Period

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 paths D0 to CLB1.R and EN to CLB2.EC (if the reset/enable were synchronous with respect to the clock).

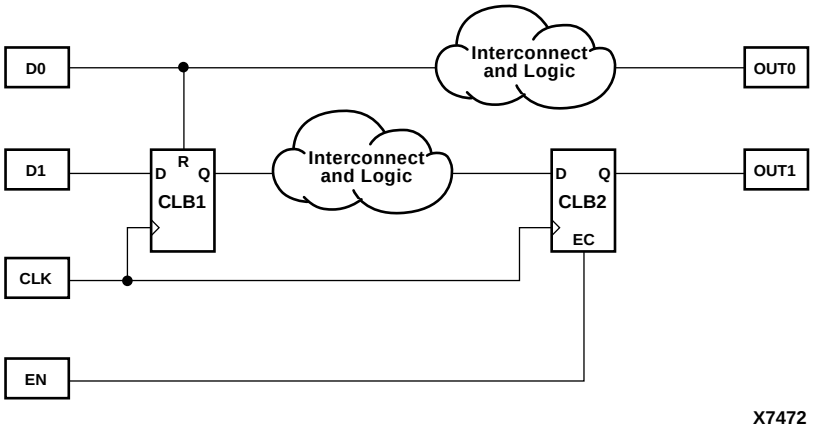


Figure 0-8 Paths for PERIOD Constraint

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:

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

where *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. There are no rules about not tracing through certain elements. 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.

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

where *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, and *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 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.

Example

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

```
TS_master=PERIOD: master_clk: 50: HIGH: 30
```

This period constraint applies to the net sys_clk, and defines a clock period of 50 nanoseconds, with a 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.

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 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.

Example

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

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

Sample Schematic Using TIMESPECs

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.

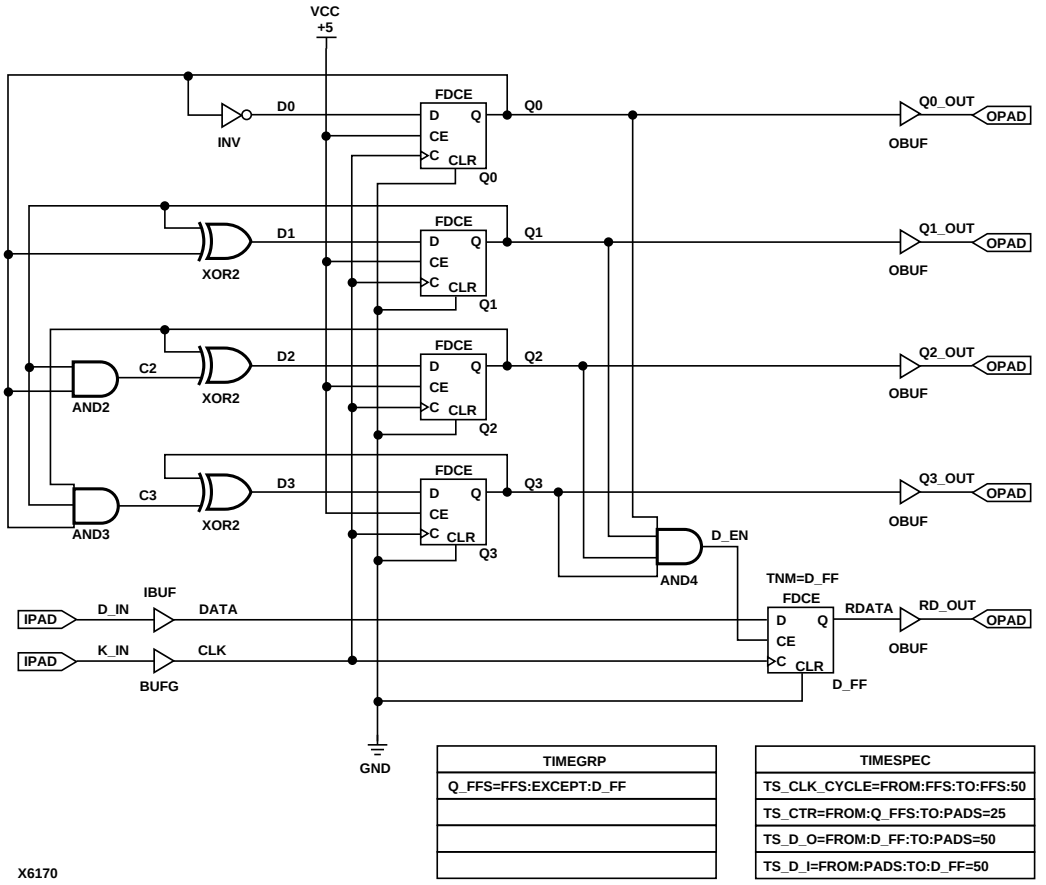


Figure 0-9 Example of Using TNMs and TIMEGRPs in Your Schematic

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.

Additional Timing Constraints

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

- Net skew control
- Net delay control
- Path tracing control
- The DROP_SPEC constraint

Controlling Net Skew

Skew is the difference between the minimum and 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:

MAXSKEW=*allowable_skew*

where *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.

Controlling Net Delay

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

MAXDELAY=*allowable_delay*

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

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*;

where *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 "Standard Block Delay Symbols for Path Tracing" 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 0-1 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

Table 0-1 Standard Block Delay Symbols for Path Tracing

io_o_i	IO output to input propagation delay. Disabled for tristated IOBs.	Enabled
io_o_pad	IO output to pad propagation delay.	Enabled

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. Syntax is as follows.

DROP_SPEC = *identifier*

where *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 "Setting TIMESPEC Priorities" 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 load pin.

Flag Type	TNM Attribute Syntax	Where Applied
Net Symbol Pin Macro	**TNM=***identifier* **TNM=***predefined_group***:***identifier*	Net, Symbol, Pin, Macro

TIMEGRP Attributes

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

Group Type	TIMEGRP Attribute Syntax
Combine	*new_group***=***group1***:***group2* **[:***group3* **. . .]**
Exclude	*new_group***=***group1***[:***group2* **. . .]:EXCEPT:***group3***[:***group4* **...]**
Clock Edge (flip-flops)	*new_group***=RISING:***group1* *new_group***=FALLING:***group1*

Group Type	TIMEGRP Attribute Syntax
Gate Edge (latches)	*new_group*=**TRANSHI:***group1* *new_group*=**TRANSLO:***group1*
Pattern Matching	*new_group*=*predefined_group*:(*name_qualifier1*[:*name_qualifier2* . . .])

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	**TS***id*=**FROM:***source_group*:**TO:***dest_group*:*delay*
Ignore	**TS***id*=**IGNORE**
Through point	**TS***id*=**FROM:***source_group*:**THRU** :*thru_point*[:**THRU:** *thru_point*]**TO:***dest_group*:*delay*
Linked specification	**TS***id*=**FROM:***source_group*:**TO:***dest_group*:*another_TSid* [*|/]*number*
Clock period	**TS***id*=**PERIOD:***TNM_reference period*:{**HIGH|LOW**}:[*high_or_low_time*]
Derived clocks	**TS***id*=**PERIOD:***TNM_reference*:*another_PERIOD_identifier* [/|*]*number*{**HIGH|LOW**} [*high_or_low_time*]
TS attribute priority	*normal_timespec_syntax*:**PRIORITY:***integer*

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

Attribute Syntax	Where Applied	How Used
TPTHRU=*identifier*	Net, symbol, pin, macro	In through point TS attribute
TPSYNC=*identifier*	Net, symbol, pin, macro	As group in TS attribute
TIG **TIG=***identifier*	Net, pin	Prevents timing analysis

Attribute Syntax	Where Applied	How Used
DROP_SPEC=*identifier* (Constraints file only)	N / A	Prevents timing analysis for **TS***identifier*

Additional Timing Constraints

The following table lists additional timing constraints.

Attribute Syntax	Where Applied	How Used
PERIOD:*period***{HIGH|LOW}** **[***high_or_low_time***]**	Net, pin	Specifies register clock period
MAXSKEW=*allowable_skew*	Net	Specifies net skew
MAXDELAY=*allowable_delay*	Net	Specifies net delay

Specialized Support for Synopsys

The Xilinx Development System contains support for Synopsys timing constraints in UCF, NCF, and PCF files but not for schematics.

Timing Specification Offsets

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:

- 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.

Offsets support the translation of Synopsys set_arrival, set_input_delay, and set_output_delay constraints. These constraints are only used for pad-related nets and cannot be used to extend the arrival time specification method to the internal nets in a design.

There are two forms of the offset specification. The first method is a logical form that can be placed in a UCF or NCF file in a NET

netname constraint type record. The constraint has the following syntax:

OFFSET = (IN|OUT):*offset_time [units]***:{BEFORE|AFTER}:***clk_net*

where *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 variable name *clk_net* is the fully hierarchical net name of the clock net between its pad and its input buffer.

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*. For example, **OFFSET IN 20 ns BEFORE clk1** dictates that all inputs will have data present at least 20 ns before the triggering edge of clk1.

The second form is a physical form used in a PCF file that uses blocks (comps) instead of nets. The syntax is:

[COMP : *iob_name***] OFFSET = {IN | OUT}** *offset_time* **[***units* **] (BEFORE |AFTER) COMP** *clk_iob_name*

where *iob_name* is an optional field that defines the block name of the IOB that the offset refers to. If this field is omitted, 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.

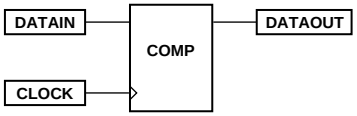
clk_iob_name is the block name of the clock IOB.

In cases where all offset specifications are defined without the optional clock net, they are all relative to each other. In cases where an offset has clock specified, it is only defined relative to the other offset specifications that reference the same clock net.

The relationship between offsets and the XDELAY margins is that a margin on an input is equivalent to an OFFSET:IN:BEFORE style offset, and a margin on an output equivalent to an OFFSET:OUT:AFTER style offset. The difference between margins and offsets is that margins were not actually used for timing specification, only timing reporting.

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

In the following figure, two IOBs (DATAIN and DATAOUT) have an external timing relationship with the reference input CLOCK. The sequential element (COMP) is common to both IOBs (DATAIN and DATAOUT).



X7435

The following offset constraint examples are applied to the figure above. The constraints, as they would appear in the PCF file, are in boldface type.

Examples

Example 1

```
COMP DATAIN OFFSET=IN 20.0 ns BEFORE COMP CLOCK;
```

This constraint indicates that the data will be present on the DATAIN pin at least 20 ns before the triggering edge of the clock net. Paths from DATAIN to COMP and from CLOCK to COMP are implicitly enumerated.

To ensure that the timing requirements are met, the timing analysis software would verify 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. In the following formulas, t represents delay time.

$MAX(^{t}data) + MAX(^{t}setup) - MIN(^{t}clk) \leq 20\ ns$

where ^{t}data is the delay from DATAIN to COMP, ^{t}setup is the setup time for the register input relative to the clock, and ^{t}clk is the delay from CLOCK to COMP.

Example 2

```
COMP DATAIN OFFSET=IN 30.0 ns AFTER COMP CLOCK;
```

This constraint indicates that the data will arrive at the pin of the device (COMP) no more than 30 ns after the triggering edge of the clock. Again, paths from DATAIN to COMP and from CLOCK to COMP are implicitly enumerated. The path DATAIN to COMP would contain the setup time for the COMP data input relative to the CLOCK input.

Verification would be almost identical to Example 1, except that the offset margin (30.0 ns) would be 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 constraint must be used to specify the period for the reference path. This is required only for offset **OUT** constraints with the **BEFORE** keyword or offset **IN** with the **AFTER** keyword.

Period $-^{t}$(DATAIN to COMP) $-^{t}$(setup) + ^{t}(CLOCK to COMP) $\geq$30 ns

Example 3

```
COMP DATAOUT OFFSET=OUT 15.0 ns BEFORE COMP CLOCK;
```

This constraint states that the data clocked to DATAOUT must leave the FPGA 15 ns before the next triggering edge of the clock. Paths from COMP to DATAOUT and from CLOCK to COMP are implicitly enumerated. The path COMP to DATAOUT would include the CLOCK-to-Q delay (component delay). The data clocked to DATAOUT will leave the FPGA 15.0 ns before the next clock input.

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

As in example 2, a PERIOD or FREQUENCY constraint must be used to specify the period for the reference path. This is required only for

offset **OUT** constraints with the **BEFORE** keyword or offset **IN** with the **AFTER** keyword.

Period $-^{t}$(CLOCK to COMP) $-^{t}$(COMP to DATAOUT) $\geq$ 15 ns

Example 4

```
COMP DATAOUT OFFSET=OUT 35.0 ns AFTER COMP CLOCK;
```

This constraint calls for the data to leave the FPGA 35 ns after the present clock input. As in Example 3, data paths from COMP to DATAOUT and from CLOCK to COMP are enumerated. The path COMP to DATAOUT would include the CLOCK-to-Q delay (component delay).

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

^{t}(CLOCK to COMP) + ^{t}(COMP to DATAOUT) $\leq$ 35 ns

Ignoring Paths

The Synopsys set_false_path constraint defines a path that should be ignored for timing purposes. These equivalent Xilinx constraint constructs are based on the FROM:TO style described in the “Basic TIMESPEC Syntax” section and the FROM:THRU:TO style syntax described in the “Defining Intermediate Points on a Path” section. These constraints require the definition of source and destination groups using the grouping mechanisms described in the “Specifying Groups in TS Attributes” section and their syntax is as follows:

TS*identifier***=FROM:***source_group***{:THRU:***thru_point***}:TO** *dest_group***:TIG**

Index

W

X