System Connectivity Manager to Constraint Manager User Guide

Product Version 23.1

September 2023

Document Last Updated: June 2020

Contents

Preface

About This User Guide

This guide explains how to use the Cadence logical design tool — System Connectivity Manager — with Constraint Manager for managing electrical constraints. Constraint Manager is tightly integrated with System Connectivity Manager, which is a high-end design tool that supports spreadsheet-based methodology for design capture.

This user guide assumes that you are familiar with the development and design of electronic circuits at the system or board level.

Finding Information in This User Guide

This user guide covers the following topics:

See...	For Information About...
Chapter 1, "Introduction to Constraint Manager"	Introduces Constraint Manager and constraint objects supported in Constraint Manager.
Chapter 2, "Electrical Constraints"	Describes how to capture electrical constraints in Constraint Manager.
Chapter 4, "ECSet in SigXplorer"	Provides an overview of how SigXplorer is used with Constraint Manager to modify design topologies and constraints.
Chapter 5, "Constraints in Hierarchical Designs"	Describes how to manage constraints in a hierarchical design.
Chapter 6, "Restoring Constraints from Definition"	Describes how to restore lower-level constraints from their original definitions.
Chapter 7, "Working with Properties"	Describes how to use Constraint Manager to view and capture properties on design elements, such as nets, components, and pins.

See...	For Information About...
Appendix A, "Pre-16.0 Designs in 16.0 Release"	Describes migration issues when you open an old design in the latest release.

Related Documentation

You can also refer the following documentation to know more about related tools and methodologies:

System Connectivity Manager

- For information about System Connectivity Manager, see *System Connectivity Manager User Guide.*
- For learning System Connectivity Manager, see *System Connectivity Manager Tutorial*.
- For information on the new features in 17.0, see *System Connectivity Manager: What's New in Release 17.0*.

Related Tools and Flows

- For information on the front-to-back flow for PCB design, see *Allegro Front-to-Back User Guide*.
- For information on various PCB design working environments such as a team of designers working on a Design Entry HDL project, implementing FPGAs in designs, working with high-speed constraints, importing IFF files for radio-frequency designs, and reusing existing modules, see *PCB Design Flows*.
- For information on maintaining and modifying the libraries, see the *PCB Librarian Expert User Guide*, and *PCB Librarian User Guide*.
- For information on capturing electrical constraints in Constraint Manager, see the *Allegro Constraint Manager User Guide*.

Typographic and Syntax Conventions

This list describes the syntax conventions used for this user guide:

`literal`	Nonitalic words indicate keywords that you must enter literally. These keywords represent command (function, routine) or option names.
`argument`	Words in italics indicate user-defined arguments for which you must substitute a name or a value.
|	Vertical bars (OR-bars) separate possible choices for a single argument. They take precedence over any other character.
`[ ]`	Brackets denote optional arguments. When used with OR-bars, they enclose a list of choices. You can choose one argument from the list.
`{ }`	Braces are used with OR-bars and enclose a list of choices. You must choose one argument from the list.

2

Differential Pairs in Constraint Manager

Working with Differential Pairs

A differential pair represents a pair of nets or Xnets that have to be routed in a way that the signals passing through them are opposite in sign with respect to the same reference. This ensures that any electromagnetic noise in the circuit is cancelled out.

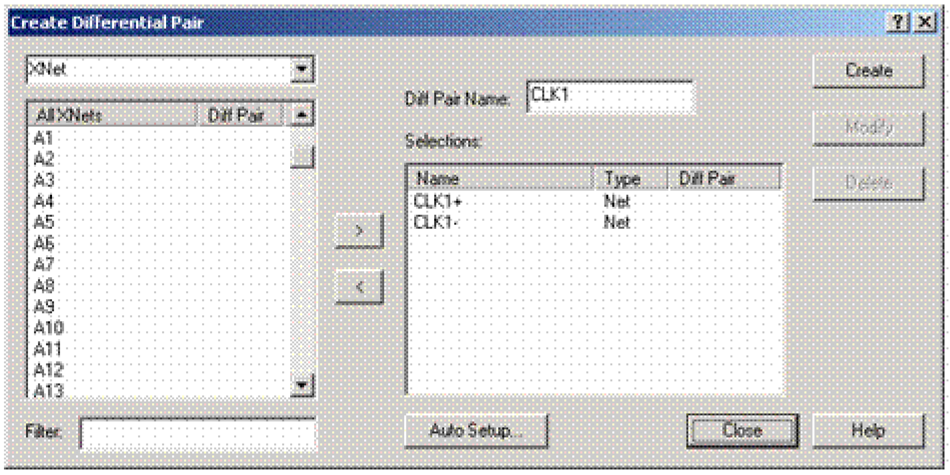
You can create a differential pair in System Connectivity Manager as well as in Constraint Manager. In Constraint manager, you can create differential pair manually or can specify setup option for auto-creation of differential pairs.

Creating Differential Pair in Constraint Manager

You can create a differential pair in any worksheet of Constraint Manager. The procedure for creating a differential pair is explained below. To create a differential pair constituting nets `CLK1+` and `CLK1-`, do the following:

1. In the Constraint Manager spreadsheet, select the nets `CLK1+` and `CLK1-` and right-click.
2. From the pop-up menu, choose *Create – Differential Pair*.

The *Create Differential Pair* dialog box appears. You can see that nets `CLK1+` and `CLK1-` are members of differential pair `CLK1`.



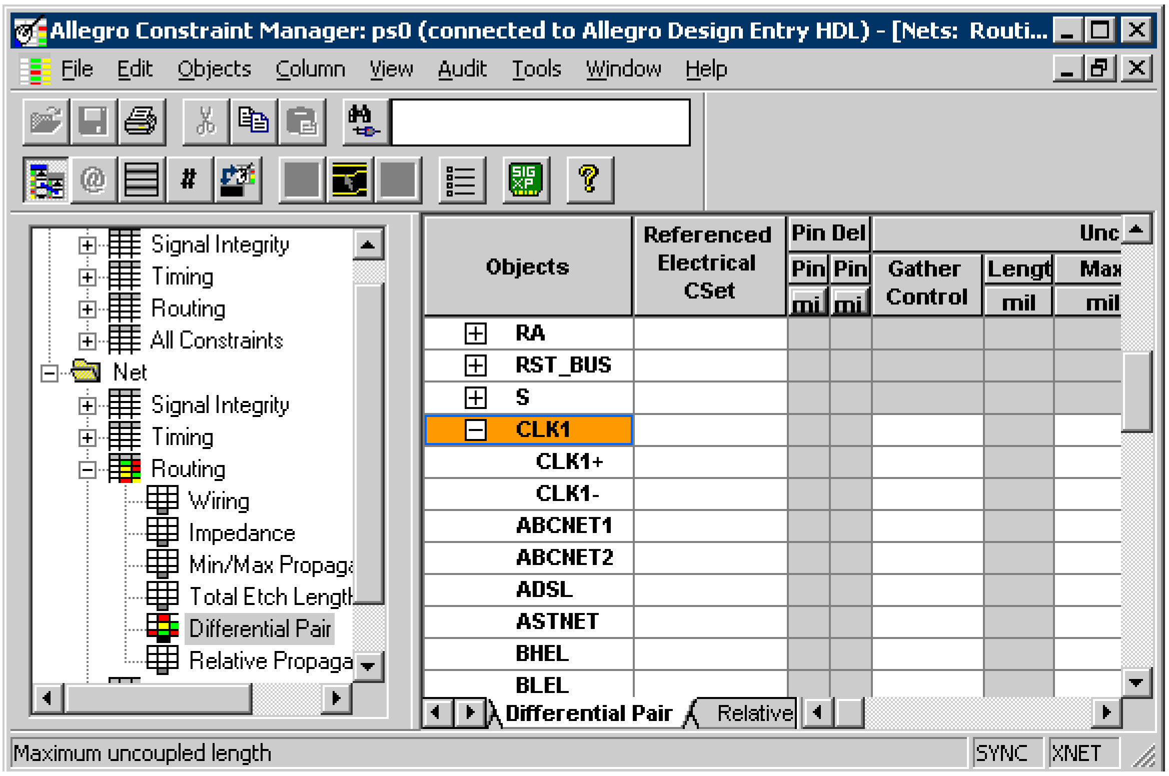
Diff Pair Name: CLK1

Name	Type	Diff Pair
CLK1+	Net	
CLK1-	Net	

3. Click *Create*.

4. Click *Close*.

   The *Create Diff Pair* dialog box closes. You can view the newly created differential pair `CLK1` as follows:

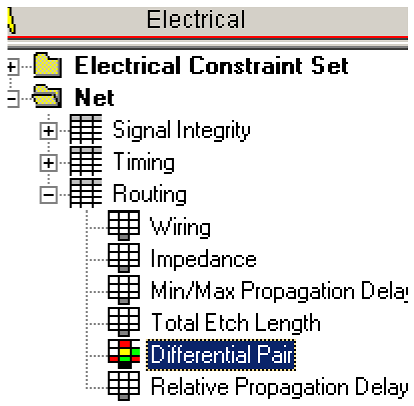


Objects	Referenced Electrical CSet	Pin Del Pin mil	Pin Del Pin mil	Gather Control	Unc Length mil	Unc Max mil
RA						
RST_BUS						
S						
CLK1						
CLK1+						
CLK1-						
ABCNET1						
ABCNET2						
ADSL						
ASTNET						
BHEL						
BLEL						

Note: You can also use the *Auto Setup* option in the *Create Diff Pair* dialog box to automatically create differential pairs for your design based on signal names. For more details on using the *Auto Setup* option, refer to the Allegro Constraint Manager Reference.

Adding Constraints on Differential Pairs

Using Constraint Manager, you can capture constraints on model-defined and user-defined differential pair signals. To capture constraints on a differential pair, use the *Differential Pair* worksheet in the *Routing* workbook accessible from the Electrical tab.



For details on the various constraints that can be captured on differential pairs, refer to the *Allegro® Platform Constraints Reference*.

Differential pair constraints present in the *Differential Pair* worksheet can be applied to only differential pairs and not to individual members of a differential pair. All other constraints present in the Constraint Manager spreadsheet can be applied to differential pairs as well as to members of a differential pair.

Differential Pairs in Hierarchical Blocks

In a logical design, if a block having differential pairs is instantiated multiple times, Constraint Manager ensures that all differential pairs have unique names. Constraint Manager creates unique names for differential pairs by adding `_1`, `_2`, `_3` and so on as suffixes to the differential pair name. This behavior is true for model-defined as well as user-defined differential pair signals.

For example, consider that the logical design for a table block, `blck`, has a differential pair signal, `FXT`, with `FXT+` and `FXT-` as member nets. If you now add two instances of `blck` to you design and then launch Constraint Manager, you will notice that for one of the blocks the differential pair signal is named as `FXT_1`. For this instance, the physical net names for the member nets are `FXT+_1` and `FXT-_1`. For each instance of the block in the design, the suffix value is incremented by 1, as shown in the following table.

	Differential Pair Name	Member Net Names
Parent Block	FXT	FXT+ FXT-
First Instance of the Block	FXT	FXT+ FXT-
Second Instance of the Block	FXT_1	FXT+_1 FXT-_1
Third Instance of the Block	FXT_2	FXT+_2 FXT-_2

In System Connectivity Manager, if you add multiple instances of a table block with differential pair signals, and select either the *Use Suffix* or *Use Prefix* option in the Block Packaging Options dialog box, then the names of the differential pair listed in Constraint Manager also use the suffix or the prefix value specified by you.

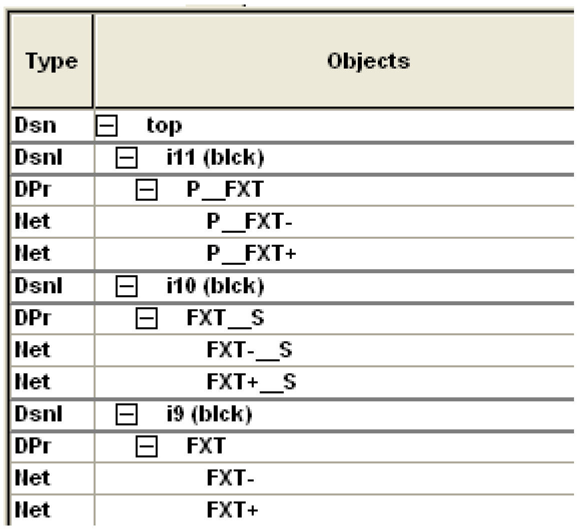
For example, consider that the logical design for a table block, `blck`, has a differential pair signal, `FXT`, with `FXT+` and `FXT-` as member nets. In a design, add three instances of `blck`. For the first instance, select *Use Optimized Packaging* option, in the Block Packaging Options dialog box. For the second instance, select the S option with suffix value as `_S`. For the third instance on `blck`, select the *Use Prefix* option, with prefix value as `P_`.

If you now launch Constraint Manager on the design, the differential pairs names are as listed in the following table.

	Differential Pair Name	Member Net Names
First Instance of the Block (Optimized Packaging)	FXT	FXT+ FXT-

	Differential Pair Name	Member Net Names
Second Instance of the Block	FXT_S	FXT+_S
(Use Suffix)		FXT-_S
Third Instance of the Block	P_FXT	P_FXT+
(Use Prefix)		P_FXT-

The differential pairs listed in the differential pair worksheet are shown in the figure given below.



Type	Objects
Dsn	⊟ top
Dsnl	⊟ i11 (blck)
DPr	⊟ P__FXT
Net	P__FXT-
Net	P__FXT+
Dsnl	⊟ i10 (blck)
DPr	⊟ FXT__S
Net	FXT-__S
Net	FXT+__S
Dsnl	⊟ i9 (blck)
DPr	⊟ FXT
Net	FXT-
Net	FXT+

Renaming a Differential Pair

Constraint Manager provides support for renaming differential pairs. While renaming a differential pair signal in Constraint Manager, following points should be kept in mind.

- On renaming a differential pair signal, the member net names are not modified. Therefore, renaming differential pair signals does not impact the pin-net connectivity in a design.
- Renaming a differential pair signal in Constraint Manager modifies the physical name of the differential pair signal.
- Modifications made because of renaming a differential pair signal in Constraint Manager, are reflected immediately in the Signal List pane in System Connectivity Manager.

- In case of flat designs, renaming a differential pair signal in Constraint Manager, causes the logical name to be updated in the Signal List pane in System Connectivity Manager.
- In case of hierarchical designs that have lower-level blocks with differential pair signals, the behavior after renaming depends on whether the differential pair is renamed in master mode or in context of the root design.
    - Renaming a differential pair signal in Constraint Manager when the block is edited in the master mode.
        - The logical name of the differential pair signal, displayed in SCM, is also updated.
        - Unless you have modified the differential pair signal name for an instance in Constraint Manager, the logical name of the differential pair signal is updated in all instances of the block in the design.
    - Renaming a differential pair signal, when an instance of the block is opened in context of the root design.
        - Only the physical name for the differential pair signal for that instance —listed in the *Phys Name* column in System Connectivity Manager — is modified.

Important

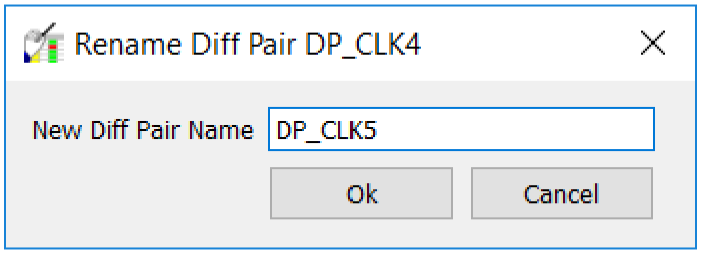
Modifications made to a name of differential pair signal in Constraint Manager, are reflected in the physical net name column of System Connectivity Manager.

To rename differential pairs in Constraint Manager, perform one of the following steps:

1. In the *Objects* column, select a differential pair object (Type `DPr`), then choose *Objects – Rename*.

   Alternatively, you can also right-click and choose *Rename* from the pop-up menu or press F2 after select a differential pair object from the *Objects* column.

   The *Rename Diff Pair* dialog box displays.



New Diff Pair Name: DP_CLK5

2. Enter a name in the *New Diff Pair Names* field or click *Use Default* to specify the new name.

   Note: The *Use Default* button is not display for user-defined differential pairs.

3. Click *OK* to rename the differential pair.

Note: You can also rename a differential pair object from the *Differential Pair Membership* dialog box (*Objects–Group members*).

Renaming Differential Pair Signals in System Connectivity Manager

- Changing the logical name for the differential pair signal in SCM changes the physical name assigned to the differential pair signal for all instances and these are reflected in CM
- Changing the physical name of a differential pair signal in SCM changes the name only for that instance in CM

Deleting a Differential Pair

If required, you can delete a differential pair using one of the following methods.

- In the *Objects* column, select a differential pair object (Type `DPr`), then choose *Objects–Delete*.
- In the *Objects* column, select a differential pair object, right-click and choose *Delete* from the pop-up menu.

Deleting a differential pair only removes the differential pair. The member nets of the differential pair are not removed from the design.

In case of a hierarchical design, with multiple instances of a block, deleting a differential pair from the instance deletes the Differential pair from Constraint Manager, but in System Connectivity Manager, logical differential pair name is still visible in the Signal List Pane. However, the `Phys Name` column in the Signal List Pane in blank.

Important

To know about launching SigXplorer on a differential pair, see Differential Pairs in SigXplorer on page 84.

1

Introduction to Constraint Manager

What is Constraint Manager?

A constraint is a user-defined requirement applied to a net or pin-pair in a design. Electrical constraints (ECs) govern the electrical behavior of a net or pin-pair in a design. For example, you can capture a constraint to define the maximum voltage overshoot tolerated by a net and capture the minimum first switch delay for a driver-receiver pin-pair in your design.

To capture the constraints Cadence provides a tool named Constraint Manager. You can use Constraint Manager with Cadence logic design tools, System Connectivity Manager and Design Entry HDL, to capture and manage electrical constraints as you implement logic. Constraint Manager is well integrated with the logic design tools, therefore, the changes that you make to constraint information in Constraint Manager are displayed in the logic design tools. Similarly, the changes that you make to constraint information in the logic design tools are displayed in Constraint Manager.

Why Constraint Manager

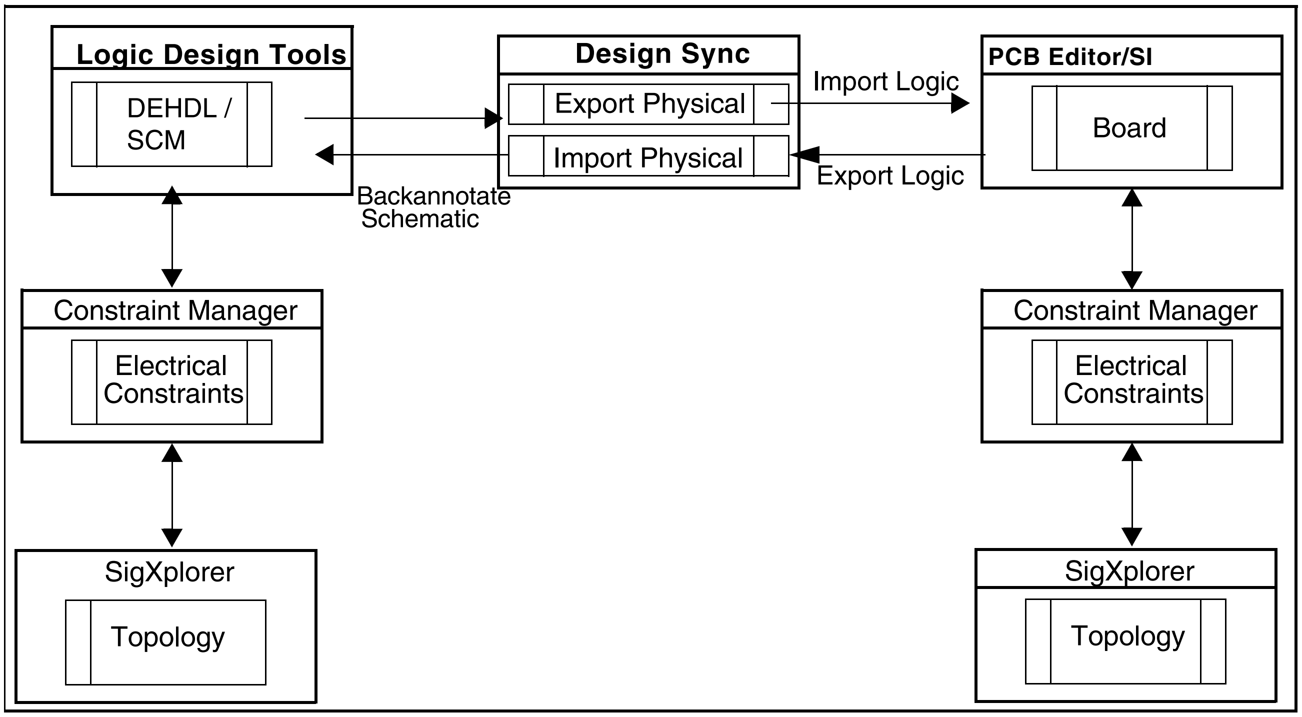
Using Constraint Manager to capture design constraints has the following advantages:

- It provides a spread-sheet based user interface that allows you to quickly capture, modify, and delete constraints.
- It supports syntax checking for all constraints.
- It supports constraint inheritance. The constraints captured on a schematic block are inherited by the design in which the block is instantiated.
- It lets you create Electrical Constraint Sets (ECSets)—a collection of electrical constraints that define a particular design requirement— and assign them to objects on which you want to capture the same set of constraints. For example, you can create an ECSet to define the default timing and noise tolerance for a net. An Electrical Constraint Set applies to an individual net although other nets may contribute to the measure of the constraint (for example, to crosstalk).

Modifying an ECSet automatically applies to all objects on which the ECSet is assigned.

- It lets you generate reports on constraints captured in the logical design and in the board.

Concurrent Capture of Constraints in Logical Design and Board



Constraint Objects

The Allegro platform has a collection of constraints that you can use to control all aspects of the design, including electrical, physical, spacing, and manufacturing parameters. The usual design practice is to capture the electrical constraints during the design capture process. These constraints are then passed to the physical design process. Physical, spacing and manufacturing constraints are imposed during the physical design.

This section provide a brief description of the constraint objects used in Constraint Manager, when it is launched from a design capture tool. In Constraint Manager user interface, the object type is listed in the TYPE column of all the worksheet. Acronyms are used to indicate the object type. Table 1-1 on page 25 lists the acronym used for each object type that is available when you launch Constraint manager from a design capture tool. Placing your cursor over a few of the Constraint Object names noting the context sensitive data tips

Table 1-1

Constraint Object	Indicated on System Connectivity Manager as...
Bus	Bus
Design	Dsn
Design Instance	DsnI
Differential Pair	Dpr
ECSet	ECS
ECSet created Match Group	ECSM
Matched Group	mgrp
Net	Net
Pin Pair	PPr
Xnet	Xnet

For more information on constraint objects listed in this section, see *Allegro Constraint Manager User Guide*.

Constraints

The Allegro platform organizes constraints by domain (Electrical, Physical, Spacing, and Design), depending on the nature of the constraint.

Electrical Constraints

An electrical constraint is a rule that characterizes and constrains the electrical behavior of a net. Impedance and propagation delay are examples of electrical constraints.

Note: Electrical Constraints are available in Constraint Manager launched from design capture tools.

Physical Constraints

A physical constraint is a rule that characterizes and constrains the physical instantiation of a net. Minimum line width and minimum neck width are examples of physical constraints that apply to the connect lines (clines) of a net.

Spacing Constraints

A spacing constraint is a rule that specifies the spacing between two physical objects. These objects may be the physical instantiation of the same net or different nets. Line-to-line spacing is an example of a spacing constraint that applies between two connect lines (clines).

Design Constraints

A *Design* constraint is a rule that applies to the entire physical design. These constraints are independent of the electrical circuit. Soldermask alignment and testpoint spacing are examples of *Design* constraints.

Constraint Sets

The *Electrical*, *Physical* and *Spacing* domains support Constraint Sets (CSets). A CSet is a named, reusable collection of constraint values. CSets are not supported in the *Design* domain.

Physical and Spacing CSets

A Physical CSet consists of one value per layer for each physical constraint. A Spacing CSet consists of one value per layer for each spacing constraint. In all designs, the tool provides one Physical- and one Spacing-CSet, named *DEFAULT,* and each constraint within the CSet has a pre-defined value. You can set the value of any constraint within a Physical- and Spacing-CSet, but you cannot remove or add constraints to the CSet, nor can you delete the CSet.

Electrical CSets

With an Electrical CSet, you define the constraints in the set. There is no pre-defined configuration, nor any pre-defined values. You can delete an Electrical CSet.

Constraint Override

A constraint *override* occurs when an individual constraint value is applied to a member of a constraint object and it overrides the inherited constraint value A constraint override has higher precedence than a CSet.

Important

Throughout the Allegro Platform, the terms *constraint*, *property* and *attribute* are sometimes mistakenly used interchangeably. Consider a *property* or an *attribute* as a mechanism to store supplemental information on an object that serves to augment your design. Consider a *constraint* as rule that is validated by the constraint system.

Net Class

A *Net Class* constraint object lets you group net objects (clines, shapes, pins, and vias) that share common characteristics and require a similar constraint requirement uses, differential pairs, Xnets, and nets.

Net Classes apply to the *Electrical*, *Physical*, and *Spacing* domains and are constrained by

- referencing a CSet

  -or-

- setting overrides directly on the *Net Class* constraint object

Differential Pairs

A *differential pair* constraint object represents a pair of Xnets or nets that are routed differentially. The opposite signals of the pair, which share the same reference, cancel out any electromagnetic noise in the circuit.

Important

The DIFFP_PRIMARY_GAP, MIN_LINE_WIDTH, DIFFP_NECK_GAP and MIN_NECK_WIDTH properties are allowed on both ECSets and PCSets; however, the ECSet value takes precedence over the PCSet value.

The Allegro platform supports two types of differential pairs:

- Model-Defined Differential Pairs

You specify model-defined diff pairs in a device signal model by designating inverting and non-inverting signals of the diff pair. You can uniquely characterize the diff pair by specifying pin parasitics, launch delays, logic thresholds, and buffer delays.

You assign device signal models to components using the PCB editor, APD, or SigXplorer. Constraint Manager then recognizes the model-defined differential pair through its view of the board database.

- User-Defined differential pairs

  You can create user-defined differential pairs directly in Constraint Manager on a net-level object. This affords you more flexibility in renaming differential pair objects and changing differential pair membership, but you forgo the accuracy of model-defined differential pairs.

Match Groups

A match group constraint object is a collection of nets, Xnets, or pin pairs that must all match (in *delay* or *length*) or be relative to a specific *target* within the group.

You can create match groups manually by adding properties directly to Xnets, or by using Constraint Manager to explicitly define pin pairs in an ECSet. Regardless of the method that you choose, Match Groups are resolved down to specific pin pairs.

Scope of Match Group

Scope controls the validation of the match group. You can create a match group with one of the following scopes:

- Local
- Global
- Bus
- Class

Local

Validates only pin pairs within each net (or Xnet) against other pin pairs in the same net (or Xnet) for each member of the match group. With a local scope, all pin pairs occupy a single match group; Constraint Manager compares pin pairs to only pin pairs in the same Xnet.

Global

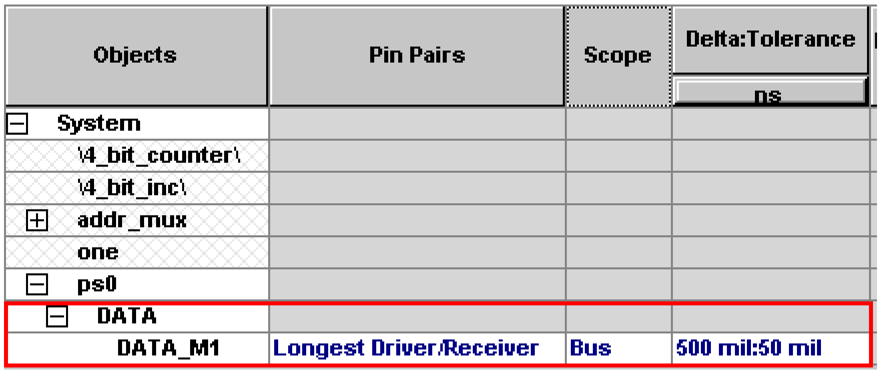
Validates all pin pairs against all other pin pairs in the match group. With a global scope, Constraint Manager validates pin pairs collectively in the (single) match group.

Bus

Generates unique match groups based on the way the target nets are grouped in the design. All pin pairs within the same bus and within the same match group are matched in such match groups. Using Bus scope, a single ECSet can be used to generate multiple unique match groups, thereby, optimizing the number of topologies required to constraint a design.

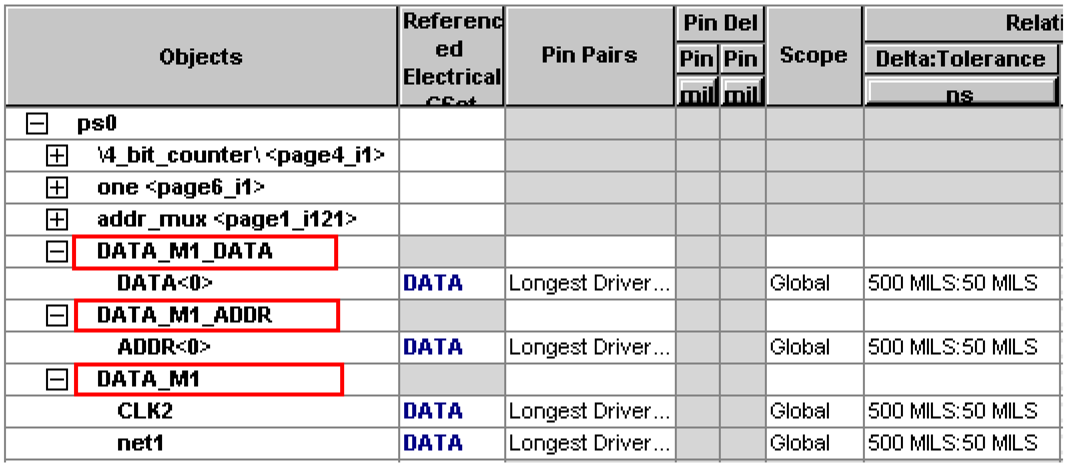
Bus scope is useful in flat designs and replicated blocks where buses (groups of signals) are replicated and the same ECSet needs to be applied to these buses. In the absence of the Bus scope, to constrain these buses you would require multiple identical ECSets that only differ by the match group name.

When you apply an ECSet to a bus, each bus inherits constraints from the ECSet. For each bus referencing an ECSet, a unique match group is created, where each bus member can be matched to the other member nets. The match group is created with a name derived from the ECSet name and the name of the bus to which the net belongs. In case of a non-bus member, Constraint Manager retains the original name of the match group. In the example shown in the figure below, notice the ECSet definition of the ECSet `DATA_M1`



Objects	Pin Pairs	Scope	Delta:Tolerance
			ns
⊟ System			
\4_bit_counter\			
\4_bit_inc\			
⊞ addr_mux			
one			
⊟ ps0			
⊟ DATA			
DATA_M1	Longest Driver/Receiver	Bus	500 mil:50 mil

- If this ECSet, `DATA_M1`, is assigned to a net in the bus `DATA`, then the net will be added to the match group `DATA_M1_DATA`.
- If the net belongs to another bus, let's say `ADDR`, a new match group is created, `DATA_M1_ADDR`.
- If the ECSet is applied to a non-bus member, the net is added to the match group, `DATA_M1`. All the other non-bus members referencing this ECSet will be added to this match group.



Objects	Referenced Electrical CSet	Pin Pairs	Pin Del		Scope	Relati
			Pin	Pin		Delta:Tolerance
			mil	mil		ns
⊟ ps0						
⊞ \4_bit_counter\ <page4_i1>						
⊞ one <page6_i1>						
⊞ addr_mux <page1_i121>						
⊟ DATA_M1_DATA						
DATA<0>	DATA	Longest Driver...			Global	500 MILS:50 MILS
⊟ DATA_M1_ADDR						
ADDR<0>	DATA	Longest Driver...			Global	500 MILS:50 MILS
⊟ DATA_M1						
CLK2	DATA	Longest Driver...			Global	500 MILS:50 MILS
net1	DATA	Longest Driver...			Global	500 MILS:50 MILS

Important

In the above figure, note that the match groups appear under the lower-level blocks to which the bus or the net(s) belongs.

Note: You must specify a bus scope for a match group within an ECSet in either the *ECSet folder* or in SigXplorer. You can then apply the ECSet to a bus or a net in the Relative Propagation Delay worksheet. Although you define Bus scope at the *ECSet* level, when the ECSet is applied to a bus member at the *Net* level, the Scope column indicates *Global*.

Class

Generates unique match groups for each class. Similar to Bus scope, Class scope also optimizes the number of topologies required to constraint a design. However, no other signal from outside the net class can be added to a match group with Class scope.

Why Class Scope?

Match groups created in a lower-level block of a hierarchical design become Global in nature in the context of the top-level design, which means that the nets in a match group can be matched with any other net in the entire design. However, the requirement in most of the designs is that the nets in a match group created in a hierarchical block must match the nets only within that block. For example, in a design you might want to match the nets of the address bus with some control signals and clocks for a specific interface, but not across multiple instances of the same interface block. While the Bus scope works in instances where the bus is contained in a hierarchical block, you cannot add any additional nets to the match group created by the Bus scope.

To address this issue, System Connectivity Manager has the Class scope, which is in essence the same as the Bus scope. Similar to the Bus scope, the Class scope also reduces the number of topologies (ECSets) required to constraint a design.

What is Class Scope?

A Class scope matches all the signals within a class and within the same match group by creating a unique match group name for each class. Class scope is limited to ECSets and is only used during the ECSet mapping process.

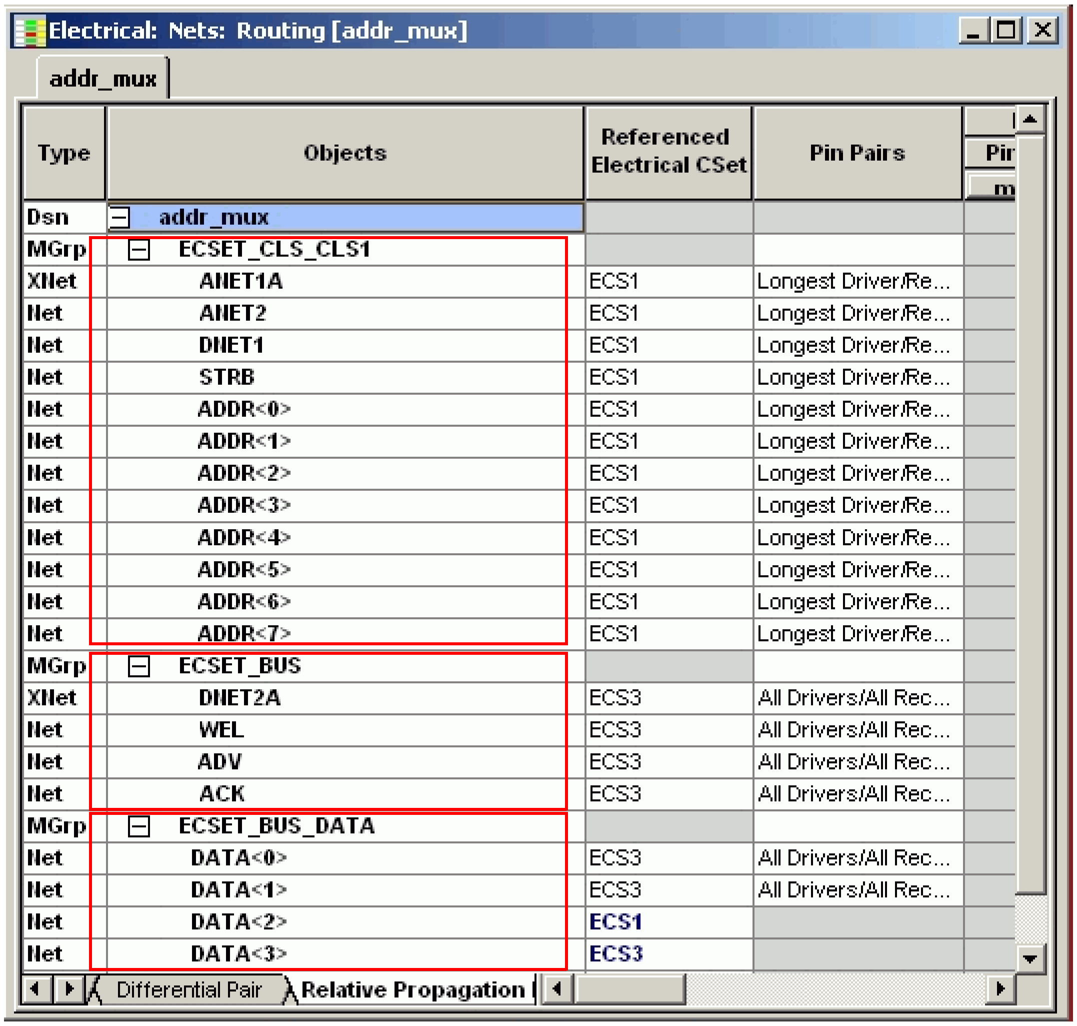
When you apply an ECSet to a class, each class inherits constraints from the ECSet. For each class referencing an ECSet, a unique match group is created, where each member of the class can be matched to the other member nets. The match group is created with a name derived from the ECSet name and the name of the class to which the net belongs.

When to Use Class Scope?

Class scope has more flexibility than Bus scope because a class can include more signals than a bus, which is typically limited to vectored nets or nets that share a common topology. Unlike the Bus scope, the Class scope adds all the selected members, including bus members, to the match group created by the ECSet (with Class scope). When you need the functionality of a Bus scope and also need additional non-bus members in a match group, use the Class scope.

In the example shown in below, an ECSet created match group `ECSET_BUS` with scope defined as BUS is assigned to class, `CLS2` containing an Xnet, a net, and a bus. Two match groups are created, `ECSET_BUS_DATA` containing only the bus members, and `ECSET_BUS` with the remaining members of `CLS2`. On the other hand, when the ECSet created match group `ECSET_CLS` with scope defined as CLASS is assigned to class, `CLS1` containing an Xnet, a net, and a bus, only one match group `ECSET_CLS_CLS1` is created. `ECSET_CLS_CLS1` contains all the members of `CLS1` including the bus `ADDR` and its members.

Figure 1-1 Class Scope



Type	Objects	Referenced Electrical CSet	Pin Pairs	Pir m
Dsn	⊟ addr_mux			
MGrp	⊟ ECSET_CLS_CLS1			
XNet	ANET1A	ECS1	Longest Driver/Re...	
Net	ANET2	ECS1	Longest Driver/Re...	
Net	DNET1	ECS1	Longest Driver/Re...	
Net	STRB	ECS1	Longest Driver/Re...	
Net	ADDR<0>	ECS1	Longest Driver/Re...	
Net	ADDR<1>	ECS1	Longest Driver/Re...	
Net	ADDR<2>	ECS1	Longest Driver/Re...	
Net	ADDR<3>	ECS1	Longest Driver/Re...	
Net	ADDR<4>	ECS1	Longest Driver/Re...	
Net	ADDR<5>	ECS1	Longest Driver/Re...	
Net	ADDR<6>	ECS1	Longest Driver/Re...	
Net	ADDR<7>	ECS1	Longest Driver/Re...	
MGrp	⊟ ECSET_BUS			
XNet	DNET2A	ECS3	All Drivers/All Rec...	
Net	WEL	ECS3	All Drivers/All Rec...	
Net	ADV	ECS3	All Drivers/All Rec...	
Net	ACK	ECS3	All Drivers/All Rec...	
MGrp	⊟ ECSET_BUS_DATA			
Net	DATA<0>	ECS3	All Drivers/All Rec...	
Net	DATA<1>	ECS3	All Drivers/All Rec...	
Net	DATA<2>	**ECS1**		
Net	DATA<3>	**ECS3**		

How to Use Class Scope?

- Creating an ECSet with Class Scope
- ECSets with Class Scope in the Context of a Top-Level Design
- ECSets with Class Scope in Constraint Manager Connected to Allegro PCB Editor

Creating an ECSet with Class Scope

In the procedures that follow, we'll consider an example of a design `ps0`, which contains a hierarchical block `addr_mux`, which in turn contains the following objects:

- Nets - STRB, ADV, ACK, WEL, ANET1A, DNET1

- Xnets - ANET1A, DNET2A
- Differential Pairs: DP1 and DP2
- Buses - ADDR, DATA
- Classes - CLS1, CLS2

To create an ECSet with the Class scope, perform the following steps:

1. Create an ECSet

   a. Launch Constraint Manager on a design.

   b. Select the *Electrical Constraint* folder.

   c. Select the *Routing—Relative Propagation* worksheet.

   d. Choose *Object – Create – Electrical Cset*.

   e. Specify the name of the ECSet as *ECS1*.

   f. Click *OK*.

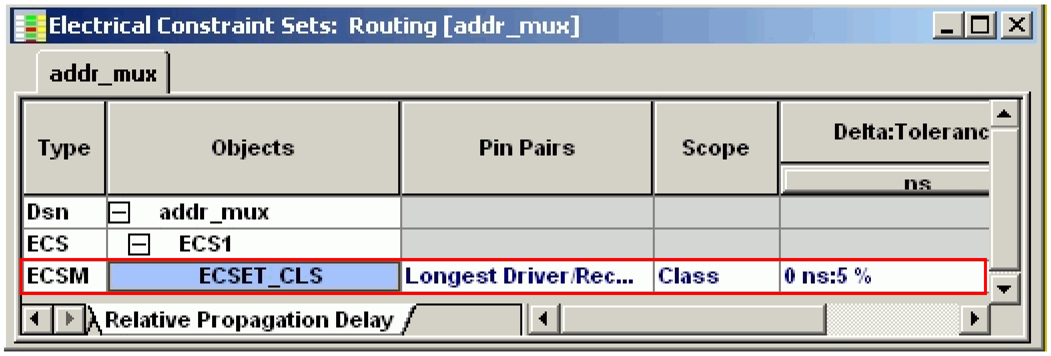
   g. Right-click *ECS1* on the worksheet and choose *Create – Match Group* from the pop-up menu.

   h. Specify *ECSET_CLS* in the Create Electrical Cset Match Group dialog box.

   i. Click *OK*.

   j. Specify *Longest Driver/Receiver* as the pin pair value.

   k. Select *CLASS* as the scope.


Electrical Constraint Sets: Routing [addr_mux]
addr_mux
Relative Propagation Delay

Type	Objects	Pin Pairs	Scope	Delta:Toleranc ns
Dsn	⊟ addr_mux			
ECS	⊟ ECS1			
ECSM	ECSET_CLS	Longest Driver/Rec...	Class	0 ns:5 %

Tip

Alternatively, you can right-click *CLS1* under *Net–Routing* worksheet in the *Electrical Constraints* tab and select SigXplorer from the pop-up menu and then create the ECSet match group in the *Relay Prop Delay* tabbed page of the Set Topology Constraints dialog box.

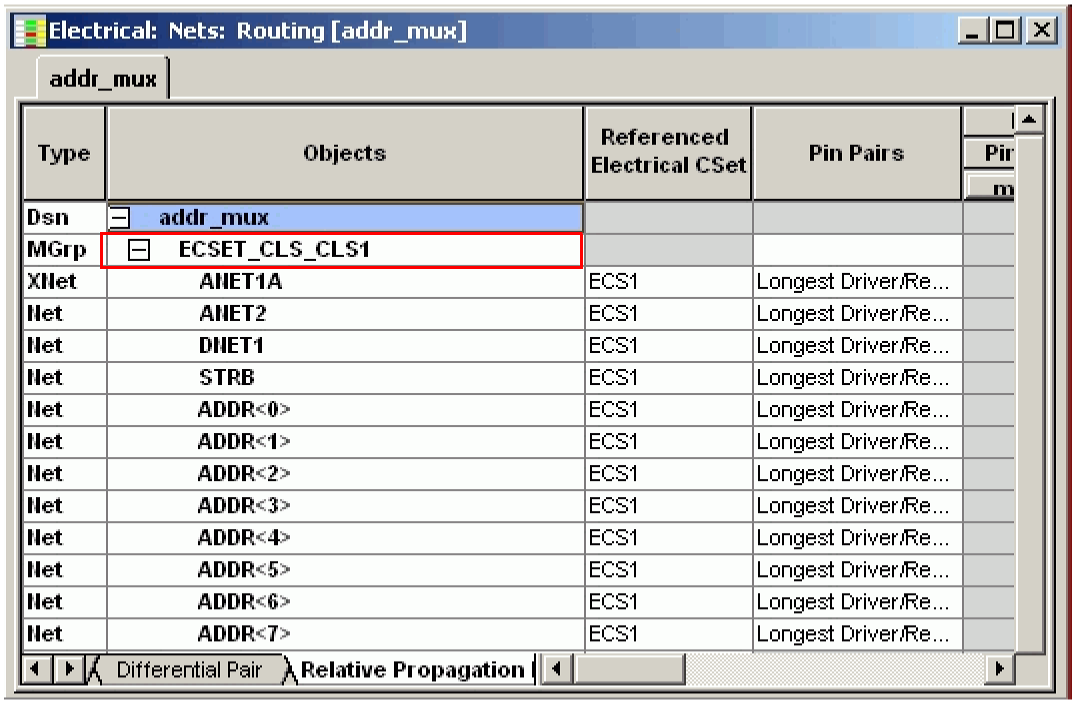
2. Assign the ECSet with Class scope to a class.

Perform the following steps to assign Class scope to a class:

a. Create a class `CLS1` containing a net, an Xnet, a differential pair, and a bus.

b. Right-click the Referenced Electrical Cset column next to CLS1 and select ECS1.

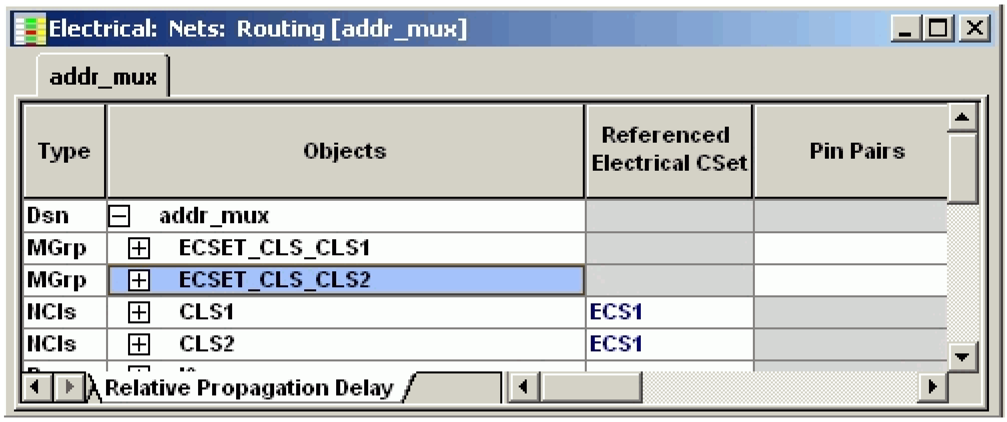
Notice a new match group `ECSET_CLS_CLS1` is created. The naming convention followed is <ECMS>_<Class name>. The newly created match group contains all the objects including the bus as part of a single match group.



Type	Objects	Referenced Electrical CSet	Pin Pairs
Dsn	addr_mux		
MGrp	ECSET_CLS_CLS1		
XNet	ANET1A	ECS1	Longest Driver/Re...
Net	ANET2	ECS1	Longest Driver/Re...
Net	DNET1	ECS1	Longest Driver/Re...
Net	STRB	ECS1	Longest Driver/Re...
Net	ADDR<0>	ECS1	Longest Driver/Re...
Net	ADDR<1>	ECS1	Longest Driver/Re...
Net	ADDR<2>	ECS1	Longest Driver/Re...
Net	ADDR<3>	ECS1	Longest Driver/Re...
Net	ADDR<4>	ECS1	Longest Driver/Re...
Net	ADDR<5>	ECS1	Longest Driver/Re...
Net	ADDR<6>	ECS1	Longest Driver/Re...
Net	ADDR<7>	ECS1	Longest Driver/Re...

c. Now assign the same ECSet to CLS2.

Notice a new match group ECSET_CLS_CLS2 is created.


Electrical: Nets: Routing [addr_mux]
addr_mux
Relative Propagation Delay

Type	Objects	Referenced Electrical CSet	Pin Pairs
Dsn	⊟ addr_mux		
MGrp	⊞ ECSET_CLS_CLS1		
MGrp	⊞ ECSET_CLS_CLS2		
NCls	⊞ CLS1	ECS1	
NCls	⊞ CLS2	ECS1	

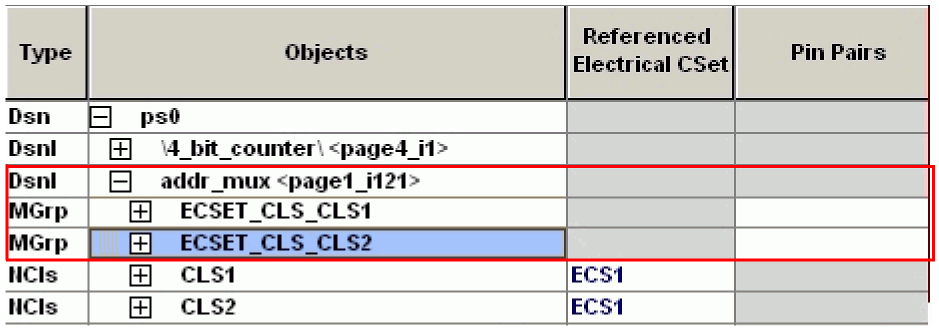
ECSets with Class Scope in the Context of a Top-Level Design

Class scope can also be used to assure that unique match groups are created in the top-level of a hierarchical design especially when replicated blocks are used. Buses and class names are given unique names when lower-level blocks are merged into a top-level design.

Let's now see how these match groups appear in the context of a top-level design.

1. In Project Manager, click *Setup*.
2. Change the root design by specifying the name of the parent (top-level) design in the Design Name field to ps0.
3. Click *OK*.
4. Launch Design Entry HDL.
5. Choose *File – Export Physical*.
6. Click *OK*.
7. Click *No* when prompted to view the log file.
8. Launch Constraint Manager.
9. Select *Routing—Relative Propagation Delay* worksheet in the *Net* folder.

Notice that the match groups `ECSET_CLS_CLS1`and `ECSET_CLS_CLS2` created in the hierarchical block `addr_mux` appear under `addr_mux`.



Type	Objects	Referenced Electrical CSet	Pin Pairs
Dsn	⊟ ps0		
Dsnl	⊞ \4_bit_counter\ <page4_i1>		
Dsnl	⊟ addr_mux <page1_i121>		
MGrp	⊞ ECSET_CLS_CLS1		
MGrp	⊞ ECSET_CLS_CLS2		
NCls	⊞ CLS1	ECS1	
NCls	⊞ CLS2	ECS1	

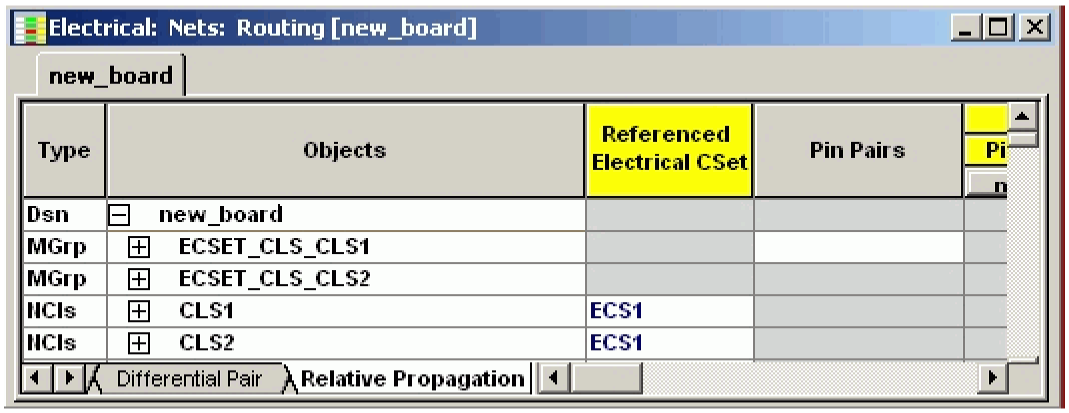
ECSets with Class Scope in Constraint Manager Connected to Allegro PCB Editor

You will now see how match groups created in a lower-level group appear in Constraint Manager connected to Allegro PCB Editor.

1. In Project Manager, click *Layout*.
2. In Allegro PCB Editor, choose *Setup – Constraints – Electrical*.

   Constraint Manager connected to Allegro PCB Editor appears.
3. Click *Routing–Relative Propagation Delay* in the *Net* folder.

   Notice that the match groups created in the lower-level block appear under the parent board.



Electrical: Nets: Routing [new_board]

new_board

Type	Objects	Referenced Electrical CSet	Pin Pairs
Dsn	⊟ new_board		
MGrp	⊞ ECSET_CLS_CLS1		
MGrp	⊞ ECSET_CLS_CLS2		
NCls	⊞ CLS1	ECS1	
NCls	⊞ CLS2	ECS1	

Note: For more information on scopes, refer to the *Working with Constraint Objects* chapter of Allegro Constraint Manager User Guide.

Buses

A *bus* constraint object represents a named collection of Xnets or Nets.

You can use a bus to group functionally similar nets, Xnets, and differential pairs. Constraints captured on a bus are inherited by all members of the bus.

Nets and Xnets

A *net* represents an electrical connection from one pin to another pin (or pins) on the same device or on a different device.

If the path of a net traverses a passive, discrete device (resistor, inductor or capacitor), then each net segment is represented by an individual net entity in the board database. The constraint system, however, interprets these net segments as a contiguous extended net, or an *Xnet*. An Xnet can also traverse connectors and cables in a multi-board configuration. Xnet creation is based on the presence of the SIGNAL_MODEL property on the discrete components.

Pin Pairs

A *pin pair* represents a pair of logically connected pins, often a driver-receiver connection. Pin Pairs may not be directly connected but they must exist on the same net or Xnet.

You use pin pairs to capture specific pin-to-pin constraints for a net or an Xnet. You can also use pin pairs to capture generic pin-to-pin constraints for CSets. Generic pin pairs are used to automatically define net- or Xnet-specific pin pairs when the CSet is referenced.

You may specify pin pairs explicitly (for example, `U1.8 U3.8`), or they can be derived based on the following criteria:

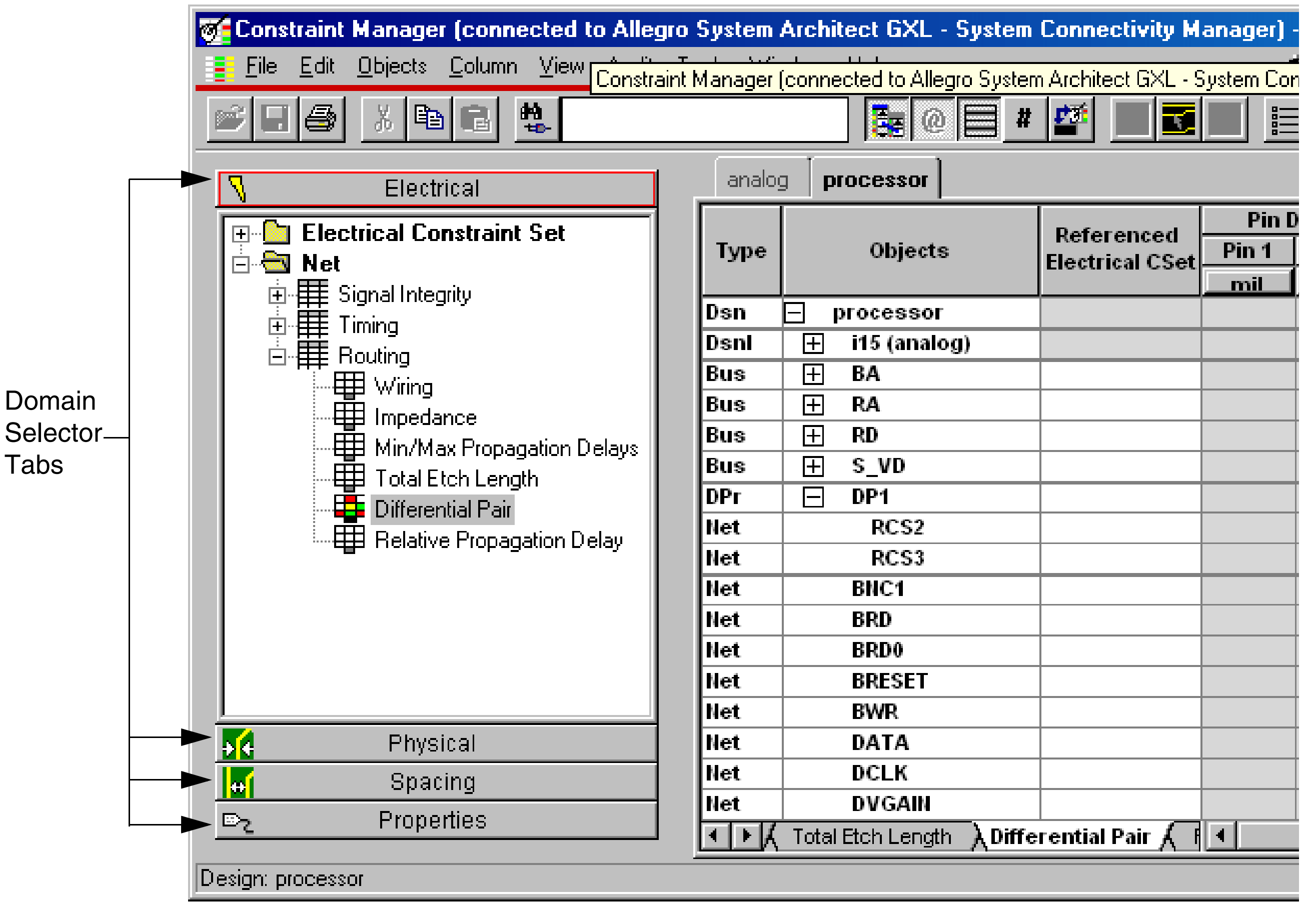
- longest pin pair
- longest driver-receiver pair
- all driver-receiver pairs

Constraint Manager User Interface

Constraint Manager user interface provides you with spreadsheet-based interface that provides a bird's eye view of constraint information in a domain at-a-glance. As shown in Figure 1-2 on page 39, Constraint Manager provides separate tabs for specifying constraints in the *Electrical*, *Physical*, and *Spacing* domains. Each domain has workbooks with property sheets that lists the constraints that can be assigned to the design object. Constraint Manager provides a spreadsheet-based view of properties for all the components, nets and pins in the design. Rows in the spreadsheet view represent objects (components, nets or pins) and columns represent properties.

To know more about workbooks, worksheet, and constraint objects, see the section on Constraint Manager controls in *Allegro Constraint Manager User Guide*.

The Properties Tab provides you with the spreadsheet based interface for capturing general properties of nets in the design.


Domain Selector Tabs
Constraint Manager (connected to Allegro System Architect GXL - System Connectivity Manager) -
File
Edit
Objects
Column
View
Constraint Manager (connected to Allegro System Architect GXL - System Con
Electrical
Electrical Constraint Set
Net
Signal Integrity
Timing
Routing
Wiring
Impedance
Min/Max Propagation Delays
Total Etch Length
Differential Pair
Relative Propagation Delay
Physical
Spacing
Properties
analog
processor
Total Etch Length
Differential Pair
Design: processor

Type	Objects	Referenced Electrical CSet	Pin D
			Pin 1
			mil
Dsn	processor		
Dsnl	i15 (analog)		
Bus	BA		
Bus	RA		
Bus	RD		
Bus	S_VD		
DPr	DP1		
Net	RCS2		
Net	RCS3		
Net	BNC1		
Net	BRD		
Net	BRD0		
Net	BRESET		
Net	BWR		
Net	DATA		
Net	DCLK		
Net	DVGAIN		

Figure 1-2 Constraint Manager User Interface

In any worksheet, the first column of lists the object type. For example, DPr in the Type column indicates that the object in that row is a differential pair.

In case of large designs, you can use Constraint Manager controls, to selectively display only the objects in a worksheet. For more information on viewing selective objects in the Constraint Manager user interface, see the section *Viewing Worksheet Cells and Objects* in *Allegro Constraint Manager User Guide*.

Important

The worksheets displayed in a workbook depend on the tool from which Constraint Manager is invoked. For example, when invoked from Allegro PCB Editor, worksheets that have constraints and properties relevant to the manufacturing process are displayed. These are not displayed when you invoke Constraint Manager from System Connectivity Manager or Design Entry HDL.

Electrical Tab

This tab displays worksheets corresponding to the ECSets and SI/Timing/Routing constraints on nets. Use this domain to capture electrical constraints on nets.

Important

Electrical constraints can only be captured in Constraint Manager invoked from logic design tool.

Physical Tab

When Constraint Manager is invoked from a logic design tool, this tab lists all the net physical classes in the design along with the member nets of each class. You can add new net classes and can also modify the membership of these classes. This information is passed to the Allegro PCB Editor in the front-to-back flow. However, if you have invoked Constraint Manager from a logic design capture tool, you cannot view or apply the constraints to the net classes. This can only be done if Constraint Manager is invoked from Allegro PCB Editor.

Note: If you open a design created in an earlier version of the design tool, the values assigned in the Physical column of the General Properties worksheet are converted to net physical classes. The nets, on which these physical constraints were assigned, are listed as member nets of the net physical classes.

When invoked from Allegro PCB Editor, worksheets for PCSets and Region net classes will also be displayed. These are not supported in the logic design and are therefore, not listed in Constraint Manager invoked from the logic design tools.

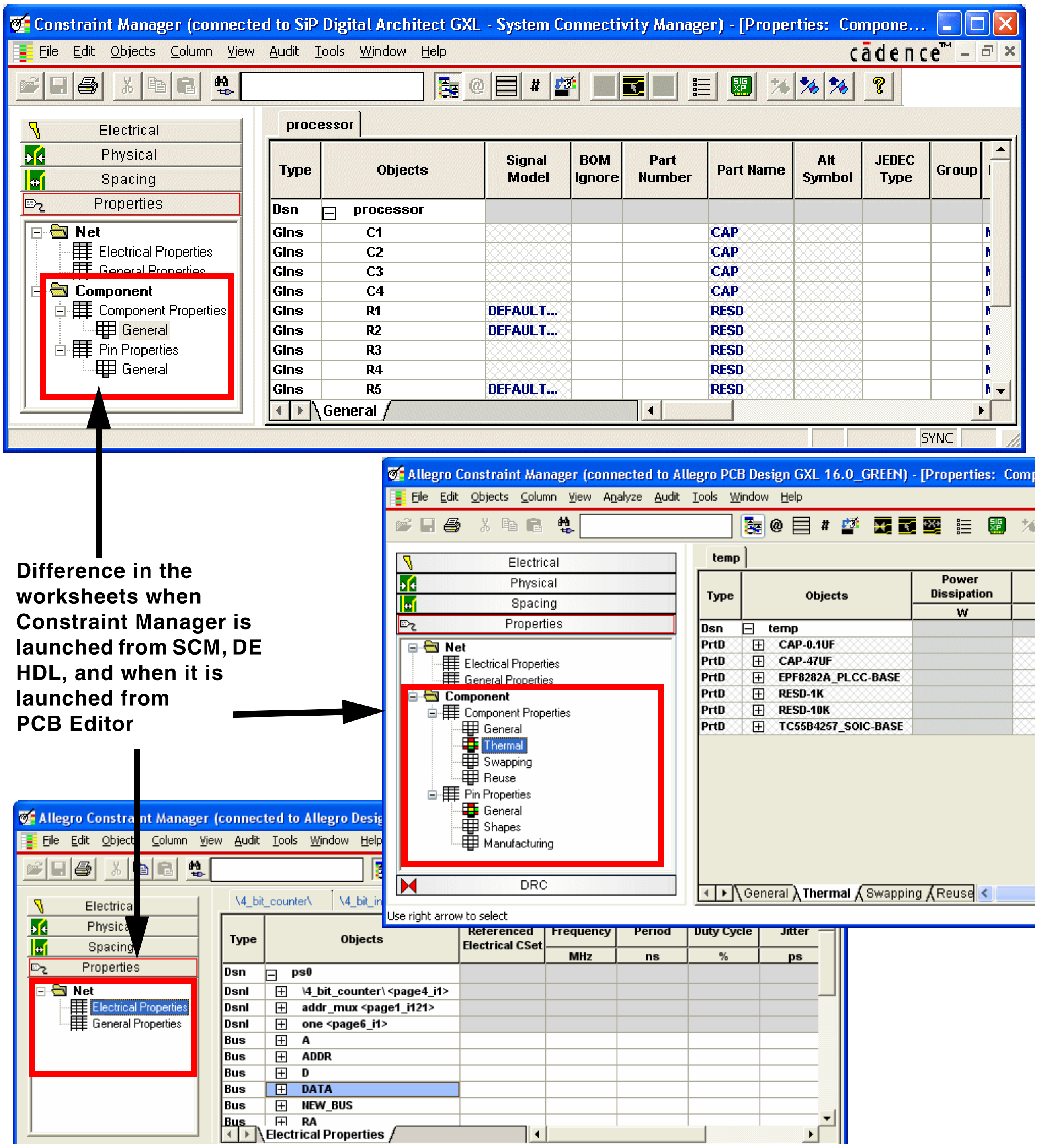
Spacing Tab

When CM is invoked from an logic design tool, this tab lists all the net spacing classes in the design along with the member nets of each class. You can add new net classes and can also modify the membership of these classes. This information is passed to the board in the front-to-back flow. However, if you have invoked Constraint Manager from a logic design capture tool, you cannot apply constraints to the net spacing classes. This can only be done if Constraint Manager is invoked from Allegro PCB Editor.

Note: If you open a design created in an earlier version of the design tool, the values assigned in the *Spacing* column of the *General Properties* worksheet are converted to net physical classes. The nets, on which these physical constraints were assigned, are listed as member nets of the net physical classes.

Properties Tab

When Constraint Manager is invoked from logic design capture tools, this tab displays the Electrical and General properties that can be assigned to the nets using Constraint Manager.



Difference in the worksheets when Constraint Manager is launched from SCM, DE HDL, and when it is launched from PCB Editor

Type	Objects	Signal Model	BOM Ignore	Part Number	Part Name	Alt Symbol	JEDEC Type	Group
Dsn	processor							
Glns	C1				CAP			
Glns	C2				CAP			
Glns	C3				CAP			
Glns	C4				CAP			
Glns	R1	DEFAULT...			RESD			
Glns	R2	DEFAULT...			RESD			
Glns	R3				RESD			
Glns	R4				RESD			
Glns	R5	DEFAULT...			RESD			

Type	Objects	Power Dissipation
		W
Dsn	temp	
PrtD	CAP-0.1UF	
PrtD	CAP-47UF	
PrtD	EPF8282A_PLCC-BASE	
PrtD	RESD-1K	
PrtD	RESD-10K	
PrtD	TC55B4257_SOIC-BASE	

Type	Objects	Referenced Electrical CSet	Frequency	Period	Duty Cycle	Jitter
			MHz	ns	%	ps
Dsn	ps0					
Dsnl	\4_bit_counter\ <page4_i1>					
Dsnl	addr_mux <page1_i121>					
Dsnl	one <page6_i1>					
Bus	A					
Bus	ADDR					
Bus	D					
Bus	DATA					
Bus	NEW_BUS					
Bus	RA					

Actual worksheets displayed in the Properties tab depend on the tool from which Constraint Manager is launched. For example, when invoked from System Connectivity Manager, Constraint Manager can be used to edit properties and constraints on components, pins, and nets. Where as when Constraint Manager is invoked from Design Entry HDL, it can only be used to capture constraints on nets.

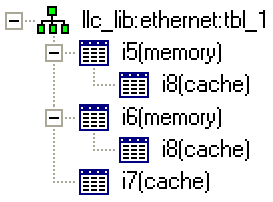
Constraint Manager Views

Constraint Manager provides a spreadsheet-based view of properties for all the components, nets and pins in the design. Rows in the spreadsheet view represent objects (components, nets or pins) and columns represent properties. There are two ways in which you can view information in Constraint Manager.

- Using Single View
- Using Tabbed View

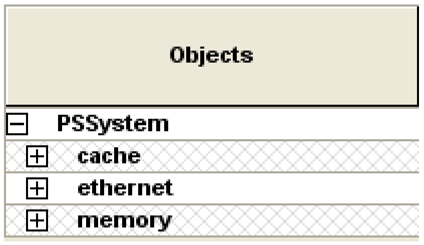
Using Single View

If you have a hierarchical design with the following structure in System Connectivity Manager:

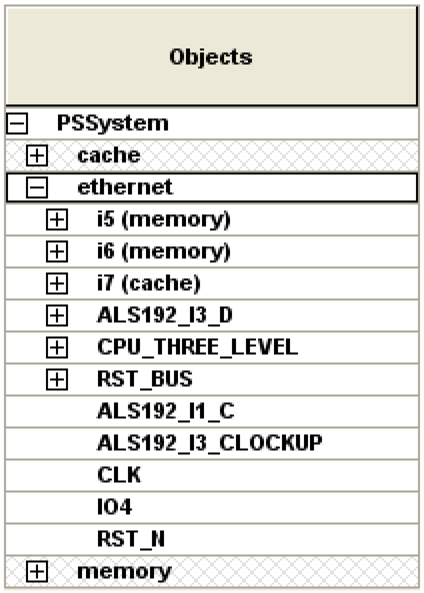


Hierarchical Design

By default, Constraint Manager displays the root design and each block in the hierarchical design as design objects under a system object named `PSSystem` as shown below:

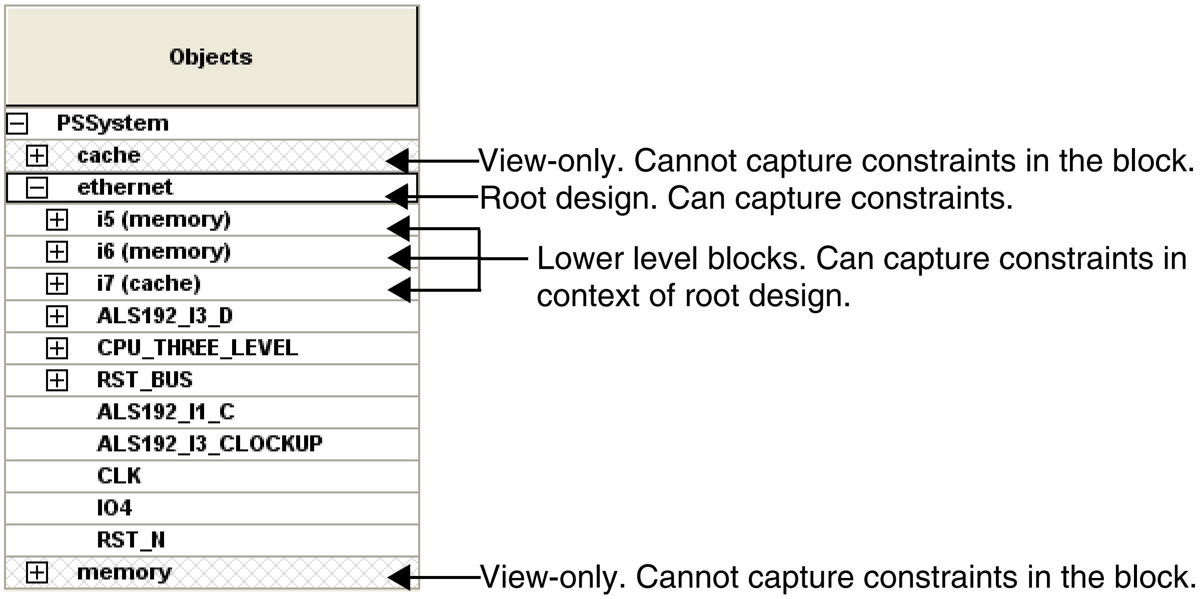


Click the ⊞ icon next to the root design or a block to view the objects under the root design or a block. For example, if you click the ⊞ icon next to the root design `ethernet`, the hierarchical structure of objects under the root design are displayed as shown below.


Objects
⊟ PSSystem
⊞ cache
⊟ ethernet
⊞ i5 (memory)
⊞ i6 (memory)
⊞ i7 (cache)
⊞ ALS192_I3_D
⊞ CPU_THREE_LEVEL
⊞ RST_BUS
ALS192_I1_C
ALS192_I3_CLOCKUP
CLK
IO4
RST_N
⊞ memory

The above figure displays the blocks and nets under the root design named `ethernet`.

Note: PSSystem is not visible if you set Constraint Manager options to display information is tabbed views


Objects
⊟ PSSystem
⊞ cache
⊟ ethernet
⊞ i5 (memory)
⊞ i6 (memory)
⊞ i7 (cache)
⊞ ALS192_I3_D
⊞ CPU_THREE_LEVEL
⊞ RST_BUS
ALS192_I1_C
ALS192_I3_CLOCKUP
CLK
IO4
RST_N
⊞ memory

View-only. Cannot capture constraints in the block.
Root design. Can capture constraints.
Lower level blocks. Can capture constraints in context of root design.
View-only. Cannot capture constraints in the block.

In the above figure, the blocks named `cache` and `memory` are grayed out because the blocks are not set as the root design for the project. You can view the constraints in blocks but cannot make any changes. If you want to assign constraints in the `cache` or `memory` block, you must set the block as the root design for the project.

In the above figure, the design named `ethernet` is not grayed out because it is the root design for the project. Constraint Manager lets you assign constraints only on the root design for the project. You can also assign constraints in the blocks under the root design. However, the constraints you assign in the blocks under the root design are applied in *context of the root design*. For example, if you assign constraints in the lower-level blocks named `memory` and `cache` under the `ethernet` design, the constraints are stored in the property file of the root design `ethernet`. In other words, the constraints will be visible in the `memory` and `cache` blocks only if you set the `ethernet` design as the root design in System Connectivity Manager and view the constraints on the `memory` and `cache` blocks. If you set the `memory` or `cache` block as the root design in System Connectivity Manager and open Constraint Manager, the constraints you added on the on the `memory` and `cache` blocks when the `ethernet` design was set as the root design are not displayed in the `memory` and `cache` blocks. This is because the constraints are applied in context of the root design `ethernet`.

Using Tabbed View

By default, the properties of all the design components are displayed in a single view. This may be difficult to work with for large designs as illustrated in Figure 1-3 on page 46. With all the data related to the lower blocks being displayed in the same worksheet, there is not much space available for the root design data. To overcome this problem, Constraint Manager provides you with an option of displaying data for different design blocks in different tabs, as shown in Figure 1-4 on page 47.

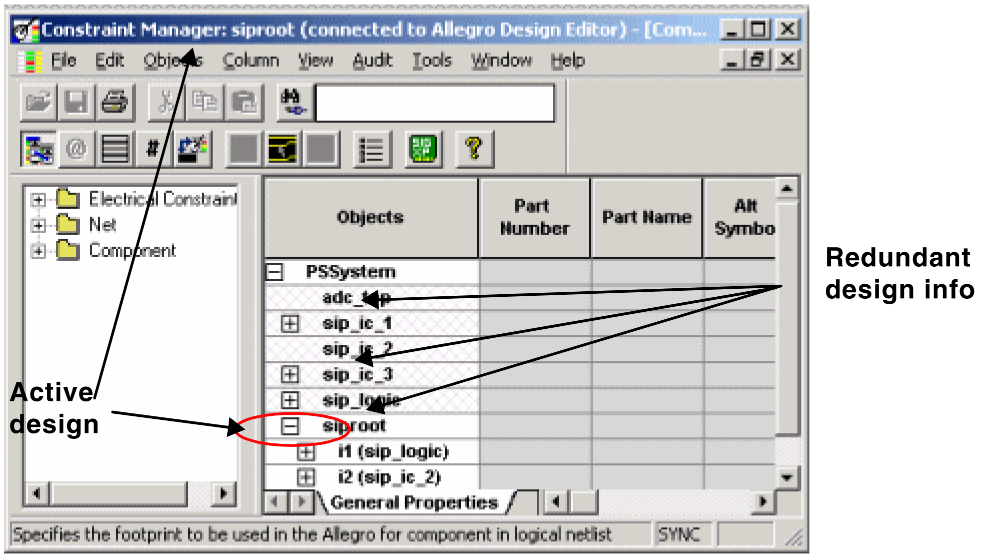

Constraint Manager: siproot (connected to Allegro Design Editor) - [Com...
File Edit Objects Column View Audit Tools Window Help
Electrical Constraint
Net
Component
Objects
Part Number
Part Name
Alt Symbo
PSSystem
adc_top
sip_ic_1
sip_ic_2
sip_ic_3
sip_logic
siproot
i1 (sip_logic)
i2 (sip_ic_2)
General Properties
Specifies the footprint to be used in the Allegro for component in logical netlist
SYNC
Redundant design info
Active design

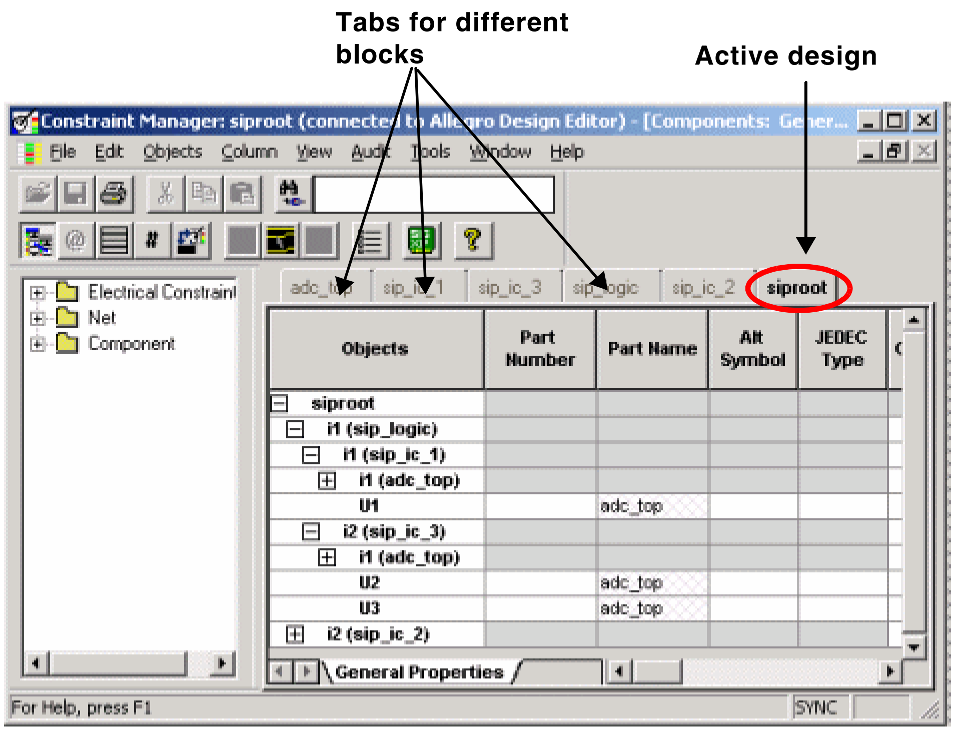
Figure 1-3 Information for all design blocks displayed in the same view.

Displaying Constraints in the Tabbed View

To display the constraint information in the tabbed view, perform the following steps in Constraint Manager:

1. Choose *View – Options*.
2. Select the *Use Tabbed View* check box.
3. Click *OK*.

Tabs are displayed in Constraint Manager as shown in the figure given below. The active tab is the tab corresponding to the root design.


Tabs for different blocks
Active design
Constraint Manager: siproot (connected to Allegro Design Editor) - [Components: Gener...
File Edit Objects Column View Audit Tools Window Help
Electrical Constraint
Net
Component
adc_top
sip_ic_1
sip_ic_3
sip_logic
sip_ic_2
siproot
General Properties
For Help, press F1
SYNC

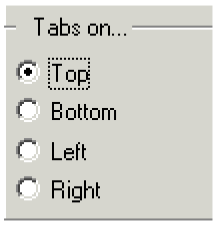
Objects	Part Number	Part Name	Alt Symbol	JEDEC Type
siproot				
i1 (sip_logic)				
i1 (sip_ic_1)				
i1 (adc_top)				
U1		adc_top		
i2 (sip_ic_3)				
i1 (adc_top)				
U2		adc_top		
U3		adc_top		
i2 (sip_ic_2)				

Figure 1-4 Information displayed in tabbed view.

Important

Number of tabs in Constraint Manager will be same as number of blocks in the logical design that can be set as root design.

Selecting the *Use Tabbed View* check box, enables the options used for specifying the position of tabs with respect to workbooks.



Tabs on...
[✓ Top]
Bottom
Left
Right

By default, tabs are displayed at the top of the workbook as shown in Figure 1-4 on page 47. If required, select the appropriate option to display the tabs at any other location.

Synchronizing Tab Selection

Constraint Manager allows you to open multiple workbooks to view different constraints. To ensure that same design is active in all open workbooks, do the following:

1. Choose *View – Options*.
2. Select the *Synchronize Tabs* check box.
3. Click *OK*.

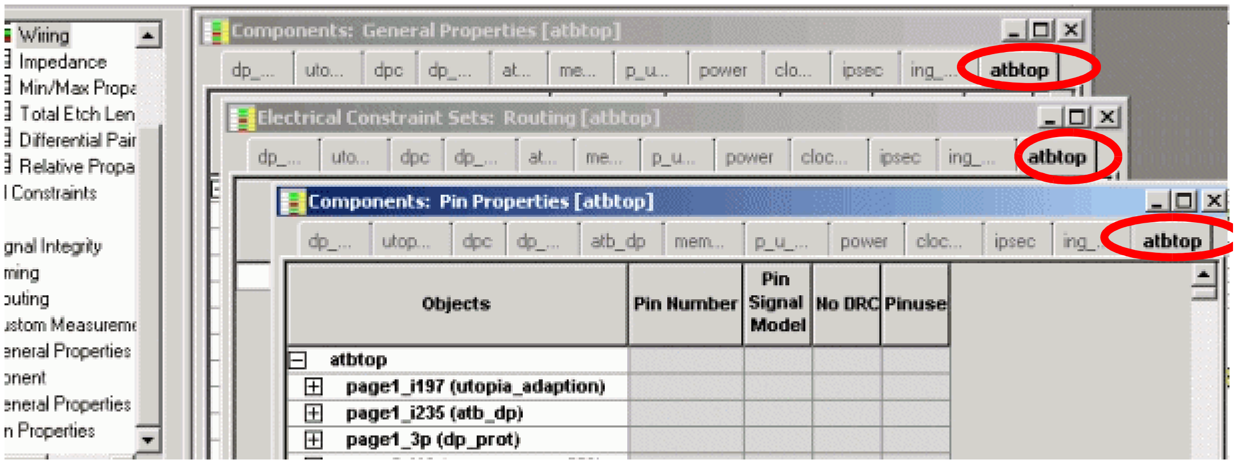

Components: General Properties [atbtop]
dp_... uto... dpc dp_... at... me... p_u... power clo... ipsec ing_... atbtop
Electrical Constraint Sets: Routing [atbtop]
Components: Pin Properties [atbtop]
Objects
Pin Number
Pin Signal Model
No DRC
Pinuse
atbtop
page1_i197 (utopia_adaption)
page1_i235 (atb_dp)
page1_3p (dp_prot)

Figure 1-5 Synchronized Tabs

If the *Synchronize Tabs* check box is not selected, different workbooks show data for different blocks in the design,

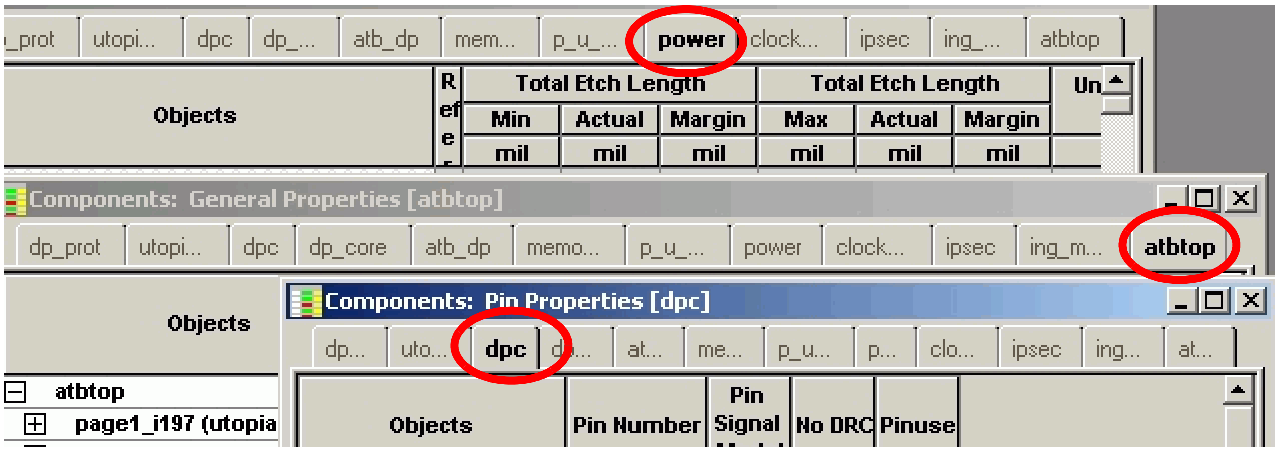

prot utopi... dpc dp... atb_dp mem... p_u_... power clock... ipsec ing_... atbtop
Objects
Total Etch Length
Min Actual Margin Max Actual Margin
mil
Components: General Properties [atbtop]
dp_prot utopi... dpc dp_core atb_dp memo... p_u... power clock... ipsec ing_m... atbtop
Components: Pin Properties [dpc]
dp... uto... dpc d... at... me... p_u... p... clo... ipsec ing... at...
Objects
Pin Number
Pin Signal
No DRC
Pinuse
atbtop
page1_i197 (utopia

Figure 1-6 Non-Synchronized Tabs

Note: If required, you can use the context-sensitive menu to modify the synchronization settings for the current session. To do this, right-click and clear the *Stay Synchronized* command.

Design Capture with Constraint Manager

Figure 1-7 on page 51 depicts the flow of information in the front to back flow. In this flow, Constraint Manager stores information about electrical constraints in a file named `<`*`root_design_name`*`>.dcfx (<`*`root_design_name`*`>.dcf)`. This file contains a snapshot of electrical constraint information in the design.

Note: For Design Entry HDL- Constraint Manager flow, the `<`*`root_design_name`*`>.dcf` file is available in the `constraints` view under the root design.

Front-to-Back Flow

The part of the diagram that covers the data flow from the logic design capture tool, to physical layout tool — Allegro PCB Editor or Allegro SI — is referred to as front -to-back flow. For sharing data with the physical layout tool, design capture tools use the *Export Physical* command. Using Export Physical command generates a set of package files that contain information about electrical constraints and the netlist. The files that are created when you the use the Export Physical command are:

- `pstchip.dat`
- `pstrxprt.dat`
- `pstXnet.dat`
- `pstcmdb.dat`

To know more about the front-to-back flow, see *Allegro Front-to-Back User Guide*.

Back-to-Front Flow

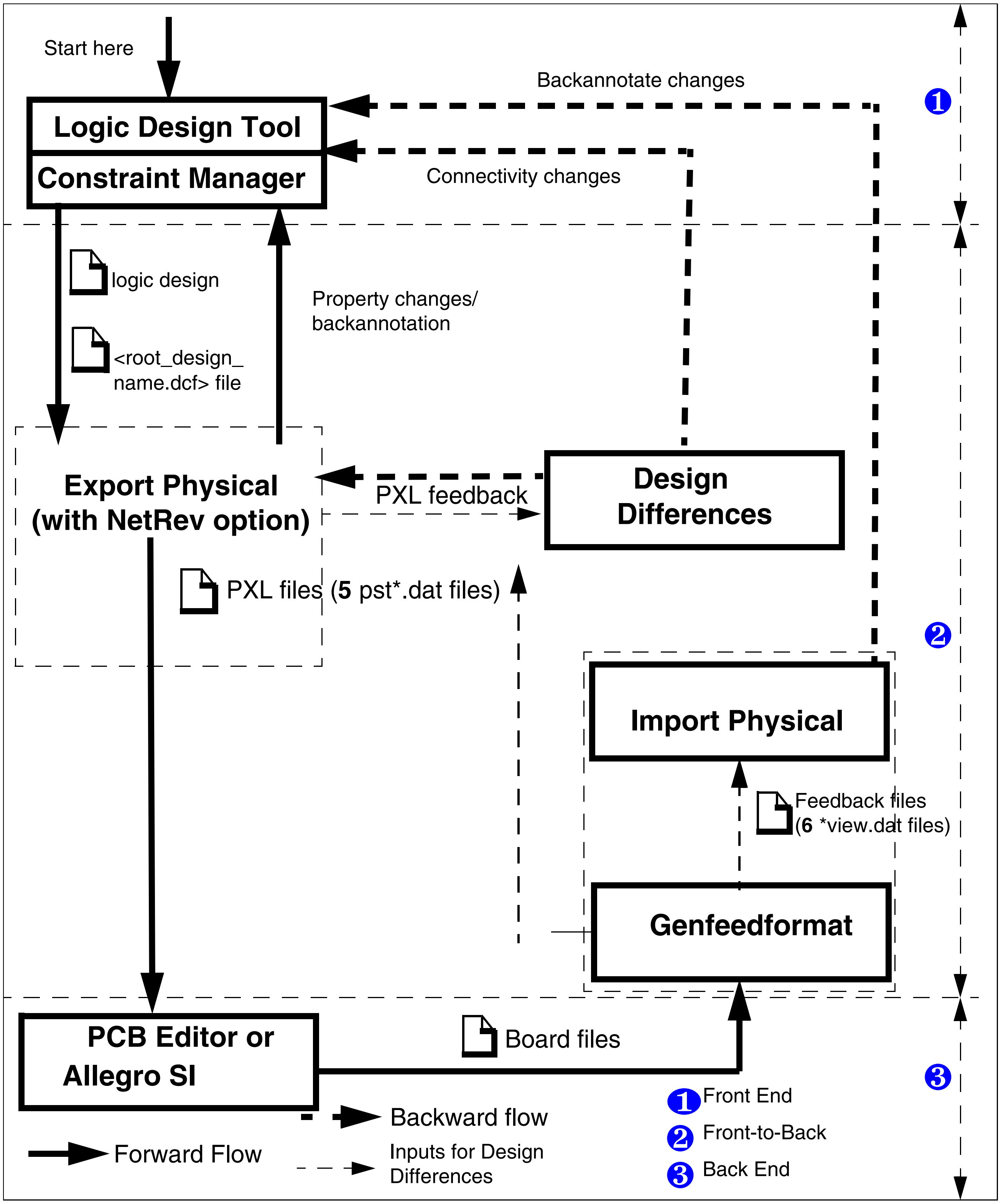
If you have made changes to the design in physical layout tool — Allegro PCB Editor or Allegro SI — the logical design needs to be updated with the changes, to ensure that the designs are synchronized. This flow of constraints and other data from Allegro PCB Editor to logic design tools is referred to as back-to-front flow. To update the logical design with the modifications in the physical layout of the design, use *Import Physical* command from the logic design tools.

The files that are read by the logic design tools, while importing changes communicate component, part, function, pin, and electrical constraint information. The six files that are used to share data in the back-to-front flow are:

- `compview.dat`
- `funcview.dat`
- `netview.dat`
- `pinview.dat`

For more information on the files used in the back-to-front flow, see the section *Back to Front Constraint Flow* in *Allegro Constraint Manager User Guide*.

Figure 1-7 Constraint Manager in Design Capture Flow



Starting Constraint Manager

From SCM

When you use System Connectivity Manager to capture your designs in a spreadsheet-based design environment, all constraints are captured in Constraint Manager. To launch Constraint Manager from SCM, use one of the following methods.

- Choose *Design – Edit Constraints*.
- Right-click in the Hierarchy Viewer, Component List pane, Signal List pane, Signal Connectivity Details pane, or Component Connectivity Details pane, and choose *Properties*.
- Click the toolbar button.

The Constraint Manager window appears. The Constraint Manager title bar displays (connected to System Connectivity Manager). This indicates that Constraint Manager has been started from System Connectivity Manager. By default, Constraint Manager is launched with the Electrical tab selected.

Note: If you want to use Constraint Manager with System Connectivity Manager, you must start Constraint Manager only from System Connectivity Manager. If you start Constraint Manager from Allegro PCB Editor, Allegro PCB SI, or as a standalone application, you cannot use Constraint Manager to manage constraints in System Connectivity Manager.

2

Electrical Constraints

Electrical constraints (ECs) govern the electrical behavior of a net or pin-pair in a design. For example, you can capture a constraint to define the maximum voltage overshoot tolerated by a net and capture the minimum first switch delay for a driver-receiver pin-pair in your design.

You can use Constraint Manager with design capture tools, Design Entry HDL and System Connectivity Manager, to capture and manage electrical constraints as you implement logic. The changes that you make to constraint information in Constraint Manager are displayed in these design capture tools. Similarly, the changes that you make to constraint information in the design capture tools are displayed in Constraint Manager.

Constraint Manager’s user-friendly interface allows you to quickly capture and manage electrical constraints. Constraint Manager validates the constraint information that you enter and passes the information in the correct syntax to the design capture tools.

Note the following when working with constraints in Constraint Manager:

- Nets are displayed in Constraint Manager using physical net (packaged) names.
- If you capture a constraint on a bus (vectored net) or vectored pin, the constraint is applicable to all the bits of the bus or vectored pin. You can also capture a constraint on a bit of a bus or vectored pin. A constraint captured on a bit of a bus or vectored pin overrides a constraint captured on the bus or vectored pin.
- If a net is aliased to another net or nets, only the base net is displayed in Constraint Manager. The base net inherits all the constraints that exist on the nets aliased to it. A constraint you add on a base net also applies to the nets aliased to it.

  For more information on base nets, see *Aliasing Nets* in *System Connectivity Manager User Guide*.
- If two nets having the same constraint are aliased, the constraint value on the base net is applied on both the nets.
- Undo and redo of constraints is not supported in Constraint Manager.

This chapter covers the following sections:

Capturing Electrical Constraints with ECSets

In Constraint Manager, you can capture an electrical constraint on design objects in the following two ways:

- Create an electrical constraint set (ECSet) in Constraint Manager and assign the ECSet to a net in the *Net* worksheets.
- Specify the constraints directly on an object (net or pin-pair) in the *Net* worksheets.

Important

Before you capture electrical constraints in your design, ensure that the design units in the board are the same as the design units (precision) in which you want to capture electrical constraints in Constraint Manager (that you launch from System Connectivity Manager or Design Entry HDL).

- ❑ To set the design units in the board, choose *Setup – Drawing Size* in Allegro to display the Drawing Parameters dialog box. The *User Units* field displays the design units for the board.
- ❑ To set the design units (precision) in Constraint Manager, choose *Tools – Precision* in Constraint Manager.

This section covers the following topics:

Overview to ECSets

An Electrical Constraint Set (ECSet) is a collection of electrical constraints that define a particular design requirement— and assign them to objects on which you want to capture the same set of constraints. For example, you can create an ECSet to define the default timing and noise tolerance for a net.

This way an ECSet can be used to define a generic set of rules applicable to a number of nets. If your design requirement changes at a later point in time, you can edit your constraint and all the objects referencing the ECSet will inherit the changed ECSet automatically. Therefore, using ECSets is a very efficient way of capturing constraints in Constraint Manager.

As design requirements change, you can:

- edit the ECSet constraints. All objects that reference the ECSet will automatically inherit these changes.
- assign a different ECSet, one that reflects a different rule-set, to the object.
- specify override properties on individual objects. Cells with overrides are colored blue.

Note: ECSets can be referenced by any number of objects, such as Buses, Differential Pairs, Xnets, Nets, and Net Classes.

Creating an ECSet

To create an ECSet, perform the following steps:

1. In the Constraint Manager window, click the *Electrical Constraint Set* folder.
2. Click *All Constraints*.
3. Choose *Objects – Create – Electrical CSet*.

   The Create Electrical CSet dialog box appears.
4. Specify a name for the Electrical CSet.
5. Click *OK*.
6. Click *Signal Integrity/Timing/Routing* workbook under All Constraints.

   This creates an empty ECSet. You can now specify constraint parameters in a worksheet cell of the ECSet.
7. Click the entry for the newly created ECSet.

8. Specify the required values in various sections for defining the ECSet. For example, set the values for the Reflection, Switch/Settle Delays, and Single-line Impedance etc.
9. Choose *File – Save*.

   This ECSet is created.

Tip

Alternatively, in the Electrical Constraint Set object folder, click on a workbook, or a worksheet within a workbook, then click on an existing ECSet, and choose *Objects – Create – Electrical CSet.* Make sure you deselect the *copy constraints from* check box.

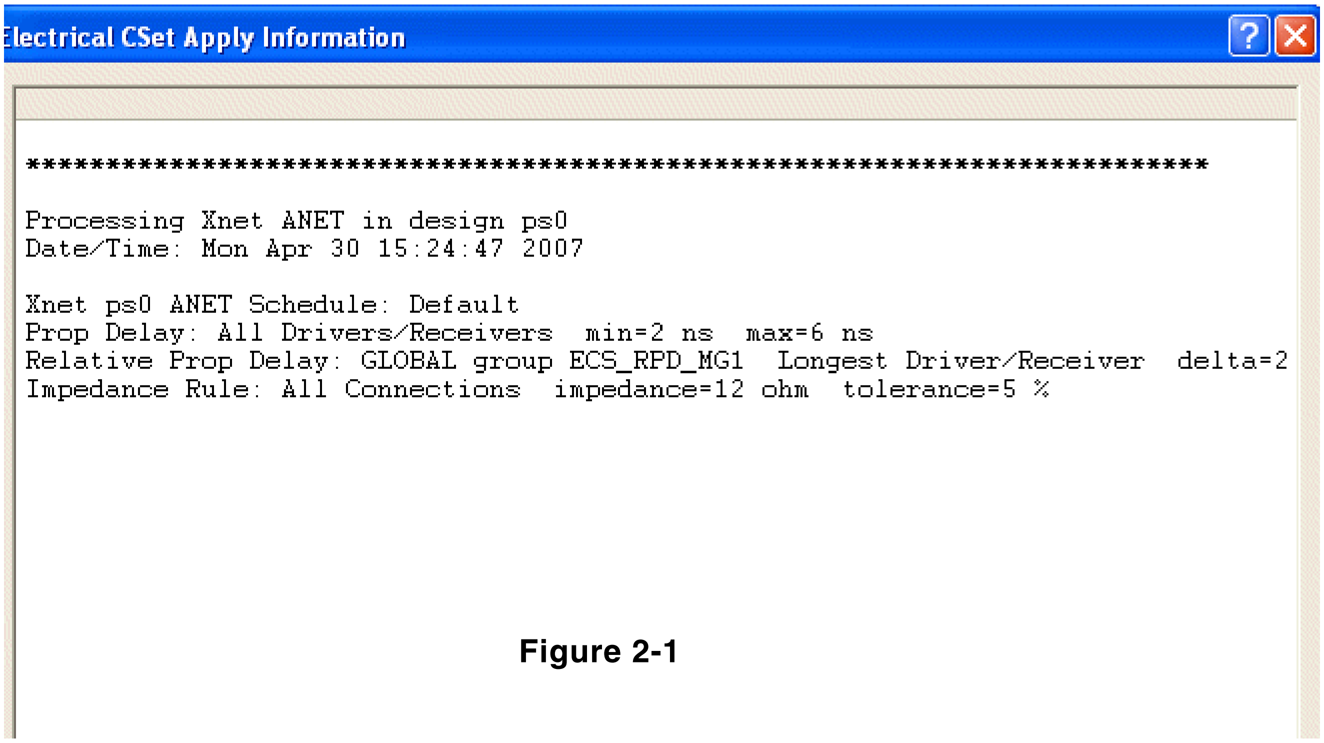
Note: You can also create an ECSet using the SigXplorer tool. For more information, refer to the chapter, Topology Extraction in SigXplorer.

For more information on detailed descriptions of every electrical constraint see, the Electrical Constraints Data Sheets chapter of the *Allegro Constraint Architecture* guide.

Assigning ECSets

You can assign constraints across nets and objects. When you apply electrical constraints defined in an ECSet to another net, the ECSet is first validated against the net and then applied. A success or failure notification is displayed depending on whether the ECSet was assigned or not.

Note: In the releases prior to 15.2, when you applied an ECSet existing on one net to another net, the process would complete without performing any validation and no errors or warnings were flagged in case of a mismatch.



```
****************************************************************************

Processing Xnet ANET in design ps0
Date/Time: Mon Apr 30 15:24:47 2007

Xnet ps0 ANET Schedule: Default
Prop Delay: All Drivers/Receivers  min=2 ns  max=6 ns
Relative Prop Delay: GLOBAL group ECS_RPD_MG1  Longest Driver/Receiver  delta=2
Impedance Rule: All Connections  impedance=12 ohm  tolerance=5 %
```

Figure 2-1

Note: An ECSet is validated only at the time of applying it on a net/Xnet or a net object or when you generate a report using the *Audit – Electrical CSets* menu command.

This topics covers the following tasks:

- Applying an ECSets on a Net or Xnet on page 57
- Applying an ECSets on a Bus on page 59
- Applying an ECSet on a Differential Pair on page 60
- Applying an ECSet to Members of a Match group on page 61

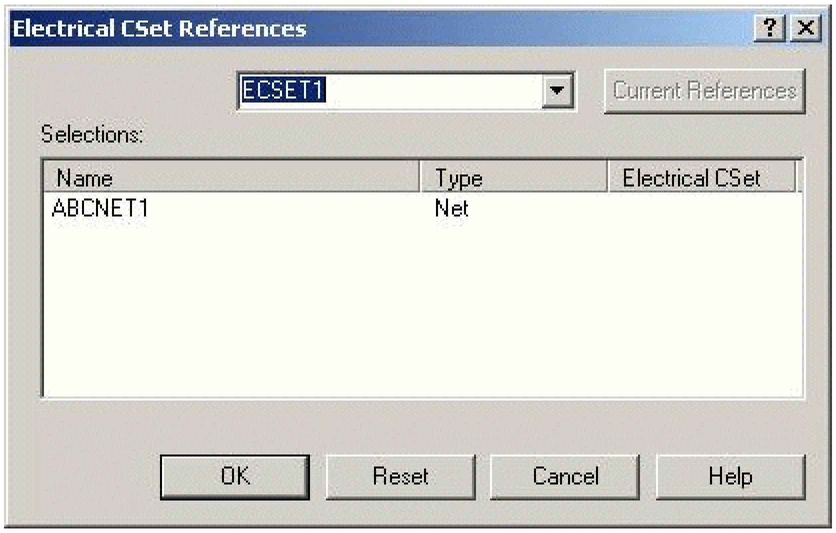
Applying an ECSets on a Net or Xnet

To apply an ECSet on a net or Xnet, do the following:

1. Right-click the net under the *Net* folder and choose the *Constraint Set References* menu from the pop-up menu.

2. Select the ECSet to assign from the drop-down list in the Electrical CSet References dialog box.

3. Click *OK*.



Name	Type	Electrical CSet
ABCNET1	Net	

Figure 2-2

Tip

You can also click the Referenced Electrical CSet column next to the net name and select the ECSet from the drop-down list.

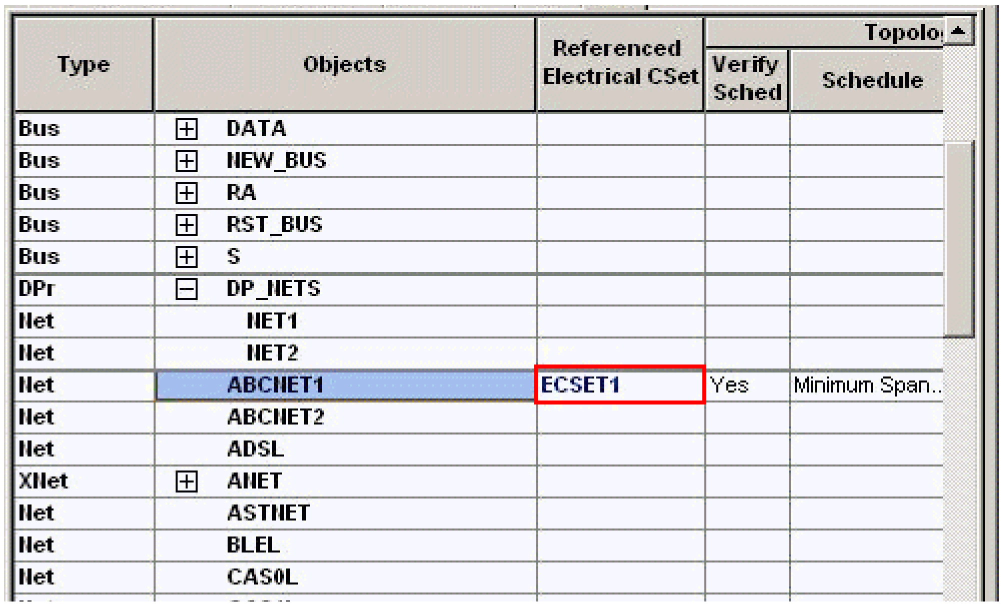
Tip

Alternatively, you can right-click on the ECSet that you created in the Electrical Constraint Set workbook, and choose the *Constraint Set References* command from the pop-up menu. Select the object type from the drop-down list, such as Net, Bus, Differential Pair, and so on. Finally, select the object from the list, click the right arrow and click *OK*.

The Electrical CSet Apply Information message box confirms that the ECSet is attached to the net.

4. Click *Close*.

The associated ECSet name appears in the Referenced Electrical CSet column next to the net name.



Type	Objects	Referenced Electrical CSet	Topolo	
			Verify Sched	Schedule
Bus	⊞ DATA			
Bus	⊞ NEW_BUS			
Bus	⊞ RA			
Bus	⊞ RST_BUS			
Bus	⊞ S			
DPr	⊟ DP_NETS			
Net	NET1			
Net	NET2			
Net	ABCNET1	ECSET1	Yes	Minimum Span..
Net	ABCNET2			
Net	ADSL			
XNet	⊞ ANET			
Net	ASTNET			
Net	BLEL			
Net	CAS0L			

Figure 2-3

Note: You can also create an ECSet in SigXplorer and apply it on the Constraint Manager objects. For more information, refer to the chapter, Topology Extraction in SigXplorer.

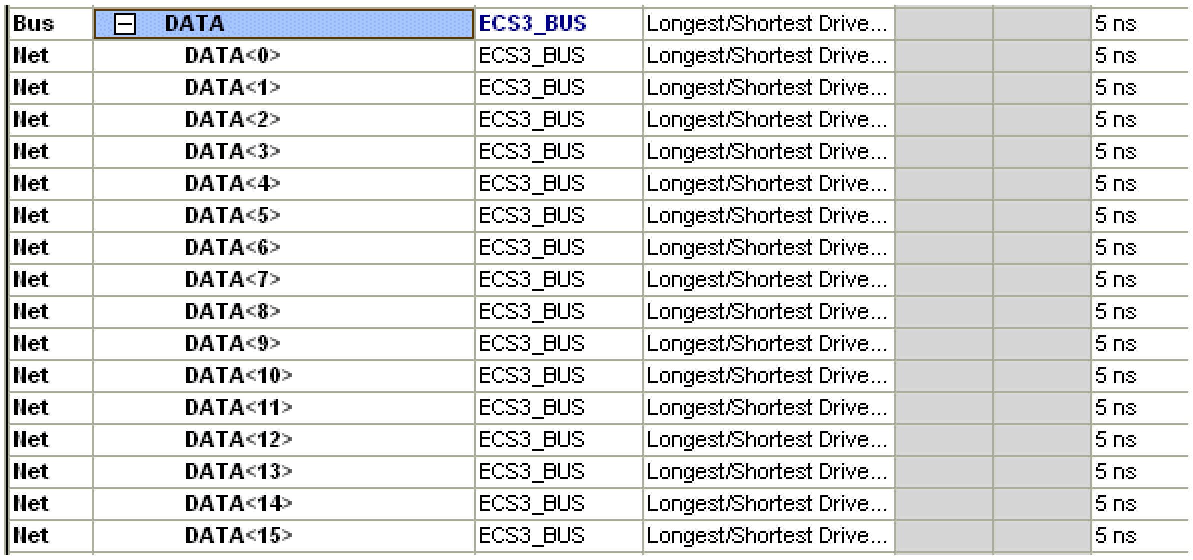
Applying an ECSets on a Bus

Constraints captured or applied on a bus (vectored net) are inherited by all members of the bus. When you associate the ECSet with a bus, all the members (bits) of the bus inherit the constraints defined in the ECSet. For example, when you associate the ECSet with a bus, all members (bits) of the bus inherit the pin-pair constraints defined in the ECSet. You need not explicitly assign constraint to individual bits of a bus. A constraint captured on a bit of a bus overrides a constraint captured on the bus.

To apply an ECSet on a bus, do the following:

1. Right-click the bus name and choose the *Constraint Set References* menu command from the pop-up menu.
2. Select the appropriate ECSet name in the Electrical CSet References dialog box.

The ECSet is assigned to the bus and all its members.



Bus	⊟ DATA	ECS3_BUS	Longest/Shortest Drive...			5 ns
Net	DATA<0>	ECS3_BUS	Longest/Shortest Drive...			5 ns
Net	DATA<1>	ECS3_BUS	Longest/Shortest Drive...			5 ns
Net	DATA<2>	ECS3_BUS	Longest/Shortest Drive...			5 ns
Net	DATA<3>	ECS3_BUS	Longest/Shortest Drive...			5 ns
Net	DATA<4>	ECS3_BUS	Longest/Shortest Drive...			5 ns
Net	DATA<5>	ECS3_BUS	Longest/Shortest Drive...			5 ns
Net	DATA<6>	ECS3_BUS	Longest/Shortest Drive...			5 ns
Net	DATA<7>	ECS3_BUS	Longest/Shortest Drive...			5 ns
Net	DATA<8>	ECS3_BUS	Longest/Shortest Drive...			5 ns
Net	DATA<9>	ECS3_BUS	Longest/Shortest Drive...			5 ns
Net	DATA<10>	ECS3_BUS	Longest/Shortest Drive...			5 ns
Net	DATA<11>	ECS3_BUS	Longest/Shortest Drive...			5 ns
Net	DATA<12>	ECS3_BUS	Longest/Shortest Drive...			5 ns
Net	DATA<13>	ECS3_BUS	Longest/Shortest Drive...			5 ns
Net	DATA<14>	ECS3_BUS	Longest/Shortest Drive...			5 ns
Net	DATA<15>	ECS3_BUS	Longest/Shortest Drive...			5 ns

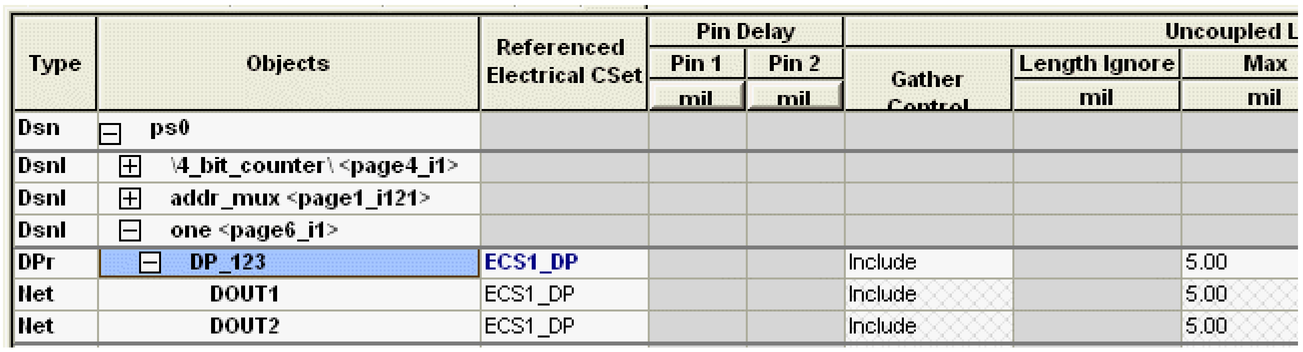
Figure 2-4

For more information on capturing constraints on a bus or a bit of a bus in Constraint Manager, see the *Allegro Constraint Manager User Guide*.

Applying an ECSet on a Differential Pair

You specify differential pair constraints in the Differential Pair worksheet of the Routing workbook. Differential pair constraints present in the *Differential Pair* worksheet can be applied to only differential pairs and not to individual member nets of a differential pair. For details on the various constraints that can be captured on differential pairs, refer to the *Allegro Constraint Architecture Guide*.

Note: All other constraints present in the Constraint Manager spreadsheet can be applied to differential pairs as well as to members of a differential pair.



Type	Objects	Referenced Electrical CSet	Pin Delay		Uncoupled L		
			Pin 1	Pin 2	Gather Control	Length Ignore	Max
			mil	mil		mil	mil
Dsn	⊟ ps0						
Dsnl	⊞ \4_bit_counter\ <page4_i1>						
Dsnl	⊞ addr_mux <page1_i121>						
Dsnl	⊟ one <page6_i1>						
DPr	⊟ DP_123	ECS1_DP			Include		5.00
Net	DOUT1	ECS1_DP			Include		5.00
Net	DOUT2	ECS1_DP			Include		5.00

Figure 2-5

The constraints added on differential pairs in Constraint Manager are saved only in the Constraint Manager database.

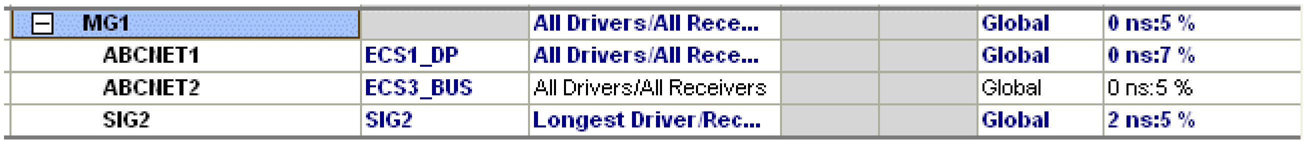
Applying an ECSet to Members of a Match group

You can assign ECSets to members of a match group, but not to a match group.The constraints you assign on a net or Xnet, which is part of a match group, will be reflected on the net or Xnet across match groups.

To assign an ECSet to a net or Xnet:

1. Right-click the net or Xnet and choose *Electrical CSet References* from the pop-up menu.
2. In the Electrical CSet References dialog box, choose the ECSet you want to assign.
3. Click *OK*.

The ECSet is assigned to the selected net/Xnet and it is reflected in the match group to which the Xnet belongs. Each member of the match group can have a different ECSet assigned to it.



⊟ MG1		All Drivers/All Rece...			Global	0 ns:5 %
ABCNET1	ECS1_DP	All Drivers/All Rece...			Global	0 ns:7 %
ABCNET2	ECS3_BUS	All Drivers/All Receivers			Global	0 ns:5 %
SIG2	SIG2	Longest Driver/Rec...			Global	2 ns:5 %

Figure 2-6

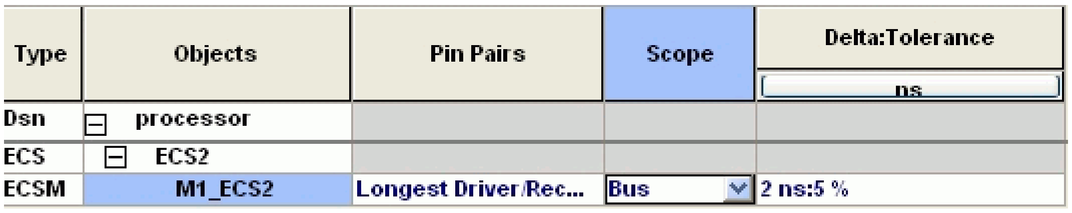
For more information on match groups, refer to the chapter Working with Objects of *Allegro Constraint Manager User Guide*.

Creating a Match Group based on an ECSet in Constraint Manager

When you create an ECSet of the Relative Propagation Delay constraints, you cannot specify constraints value at the time of creating the ECSet rule. To assign specific constraint values, such as pin pairs, scope, and Delta:Tolerance to the ECSet rule, you need to create a match group based on the ECSet. Such match groups are called ECSet match groups.

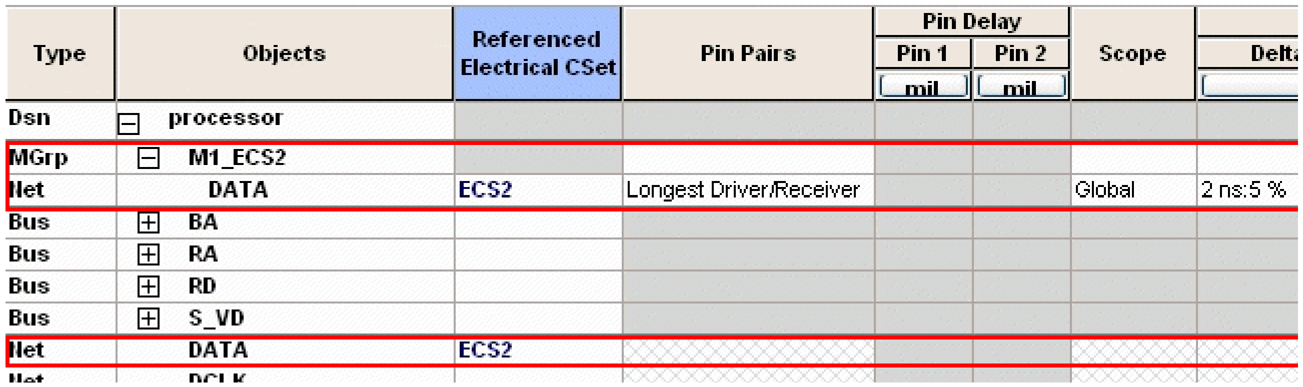
1. In the Electrical Constraint Set workbook, select the *Relative Propagation Delay* worksheet.
2. Choose *Objects – Create – Electrical CSet*.
3. Specify a name for the ECSet. For example, *ECS2*.
4. Deselect the *Copy Constraints From* check box.
5. In the Relative Propagation Delay worksheet, right-click the newly created ECSet, *ECS2*.
6. Choose *Create – Match Group* from the pop-up menu.
7. Specify a name for the match group in the Create Electrical CSet Match Group dialog box. For example, *M1_ECS2*.

   You can now specify specific values for the Relative Propagation Delay constraints in the match group, *M1_ECS2*.



Type	Objects	Pin Pairs	Scope	Delta:Tolerance
				ns
Dsn	⊟ processor			
ECS	⊟ ECS2			
ECSM	M1_ECS2	Longest Driver/Rec...	Bus	2 ns:5 %

When you assign the ECSet, *ECS2*, to a net, for example *DATA*, it will be automatically added to the ECSet match group *M1_ECS2* and inherit the constraint values from the match group.



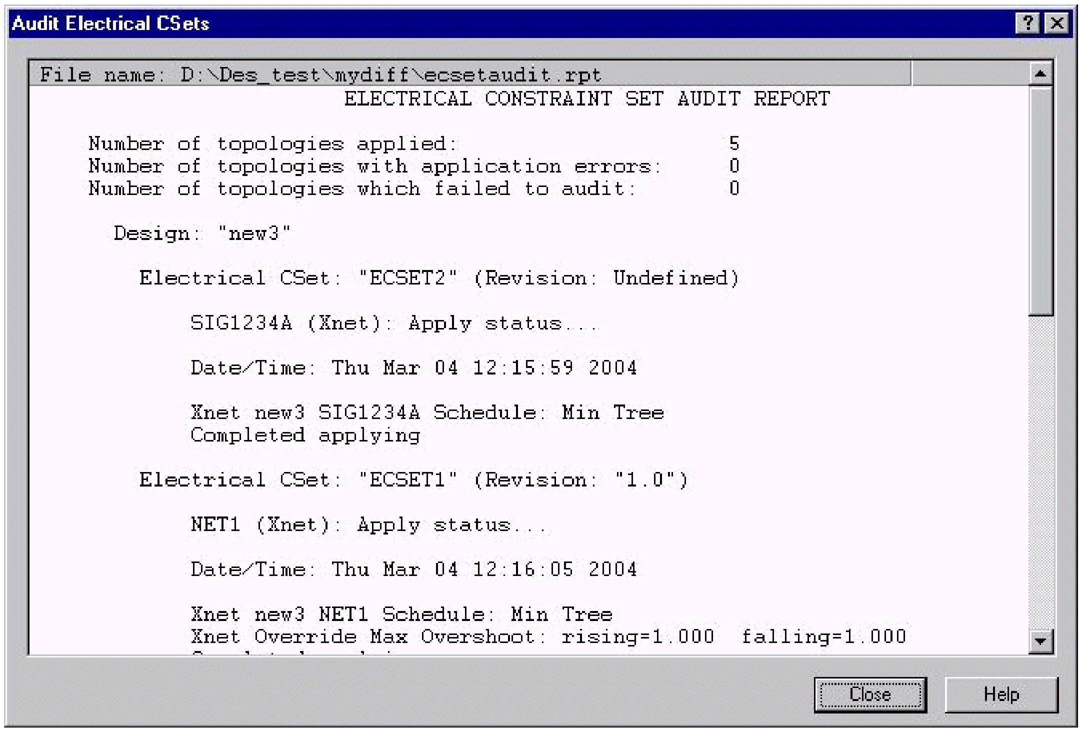
Type	Objects	Referenced Electrical CSet	Pin Pairs	Pin Delay		Scope	Delt
				Pin 1	Pin 2		
				mil	mil		
Dsn	⊟ processor						
MGrp	⊟ M1_ECS2						
Net	DATA	ECS2	Longest Driver/Receiver			Global	2 ns:5 %
Bus	⊞ BA						
Bus	⊞ RA						
Bus	⊞ RD						
Bus	⊞ S_VD						
Net	DATA	ECS2					
Net	DCLK						

Note: You can also create an ECSet generated match group in SigXplorer. For more information, see *Creating ECSet Generated Match Groups in SigXplorer*.

Auditing Electrical Constraint Set

When a topology mismatch occurs between an net/Xnet and its ECSet, the Referenced Electrical CSet cell for the net/Xnet is colored red. In addition, the status bar provides an indication of why the mismatch occurred. However, if you close Constraint Manager, the color coding is lost on subsequent invocation of Constraint Manager. Constraint Manager provides you with the option to audit all the ECSets and regenerate the color coding along with a report with detailed information on the ECSet application or violations.

➤ To generate the report, choose *Audit – Electrical CSet*.



```
File name: D:\Des_test\mydiff\ecsetaudit.rpt
                          ELECTRICAL CONSTRAINT SET AUDIT REPORT

    Number of topologies applied:                    5
    Number of topologies with application errors:    0
    Number of topologies which failed to audit:      0

      Design: "new3"

        Electrical CSet: "ECSET2" (Revision: Undefined)

            SIG1234A (Xnet): Apply status...

            Date/Time: Thu Mar 04 12:15:59 2004

            Xnet new3 SIG1234A Schedule: Min Tree
            Completed applying

        Electrical CSet: "ECSET1" (Revision: "1.0")

            NET1 (Xnet): Apply status...

            Date/Time: Thu Mar 04 12:16:05 2004

            Xnet new3 NET1 Schedule: Min Tree
            Xnet Override Max Overshoot: rising=1.000  falling=1.000
```

You can view the latest validation of the ECSet. As ECSets are stored in the `.dcf` file, topology can be reapplied to all the nets to which the ECSet is assigned.

Working with Electrical Constraints

This section covers the following topics:

- Adding Electrical Constraints on page 64
- Modifying Electrical Constraints on page 66
- Deleting Electrical Constraints on page 66
- Auditing Obsolete Objects on page 67

Adding Electrical Constraints

In addition to creating an ECSet and then assigning it to objects, you can add specific electrical constraints to individual objects in the Constraint Manager spreadsheet.

To add constraints to an object in Constraint Manager, do the following:

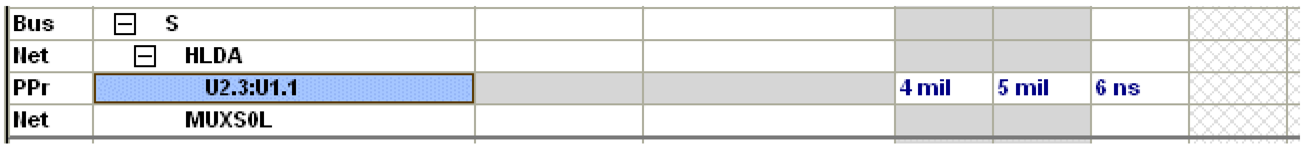
1. Start Constraint Manager from the design capture tool.
2. Click an appropriate cell next to the object to which you want to add the constraint. Add the requisite value for the cell. For example, to specify the Min/Max Propagation Delay constraints under the Routing worksheet, enter the requisite values in the *Min Prop Delay* and *Max Prop Delay* columns for a specific net or Xnet.
3. Choose *File – Save*.

Capturing Constraints on Pin-Pairs

A pin-pair represents a pair of logically connected pins that form a driver-receiver connection. Pin pairs may not be directly connected but they must exist on the same net. You use pin pairs to capture specific pin-to-pin constraints for a net.

1. Open the *Min/Max Propagation Delays* worksheet in Constraint Manager.
2. Right-click on a net and choose the *Create – Pin Pair* menu command from the pop-up menu.
3. Select the driver and receiver pins form the *First Pins* and *Second Pins* columns of the *Create Pin Pair of* <net_name> dialog box.
4. Click *OK*.

   The specified pin-pair is created.
5. Specify appropriate values in the *Pin Delay* and *Prop Delay* cells to specify the `PIN_DELAY and PROPAGATION_DELAY` constraints, respectively.



Bus	⊟ S						
Net	⊟ HLDA						
PPr	U2.3:U1.1			4 mil	5 mil	6 ns	
Net	MUXS0L						

Figure 2-7

Note: For more information on capturing constraints on pin-pairs in Constraint Manager, see the Working with Constraint Objects chapter of *Allegro Constraint Manager User Guide*.

Handling Changes in Pin-Pairs

After you have created pin-pairs and have captured certain constraints on them, packaging information such as the section, pin number, or the reference designator of the component(s) forming the pin-pair might change.

The reference designator of a component can change in the following cases:

- You change the `LOCATION` property assigned to the component during packaging or that you had set previously.
- You change packaging properties like `GROUP` or `ROOM` as a result of which the reference designator changes during packaging.

The pin number of a pin of a component can change in the following cases:

- Change the section of a component
- Swaps the pins of a component

In any of the above cases, the pin-pair information in Constraint Manager becomes outdated. Constraint Manager updates the database with the changed pin-pair when you save the constraints in the Constraint Manager window.

Modifying Electrical Constraints

If you have captured an electrical constraint in Constraint Manager or added an electrical constraint in Design Entry HDL or System Connectivity Manager, you can modify the constraint in Constraint Manager.

To modify constraints in Constraint Manager, do the following:

1. Launch Constraint Manager from your design capture tool.
2. Modify the constraint in Constraint Manager.
3. Choose *File – Save*.

For more information on modifying constraints in Constraint Manager, see the *Allegro Constraint Manager User Guide*.

Deleting Electrical Constraints

If you have captured an electrical constraint in Constraint Manager, you can delete the constraint in Constraint Manager.

To delete constraints in Constraint Manager, do the following:

1. Start Constraint Manager from the design capture tool.
2. Delete the constraint in Constraint Manager.

3. Choose *File – Save*.

For more information on deleting constraints in Constraint Manager, see the *Allegro Constraint Manager User Guide*.

Auditing Obsolete Objects

In a design, a net or pin that has an electrical constraint or a placeholder for a constraint becomes obsolete when you do any of the following:

- Delete the net
- Rename the net
- Rename a net to which the net is aliased or synonymed

Constraint Manager allows you to list the obsolete objects that no longer exist in the design but have electrical constraint information. You can delete the obsolete objects or merge the constraints on the obsolete object to an existing object. For example, if a net `SELECT` that has the `PULSE_PARAM` electrical constraint becomes an obsolete object, you can merge the constraint to an existing net in the schematic called `CLOCK`. The `PULSE_PARAM` electrical constraint will visible on the net `CLOCK` in the schematic.

For more information on auditing obsolete objects, see the *Allegro Constraint Manager User Guide*.

3

Physical and Spacing Constraints

Physical and spacing constraints impact the physical layout of a PCB board or of a SiP, and are therefore, usually captured and modified during the layout design stage by layout engineers. However, stack-up information and some physical and spacing constraints are finalized during the design capture stage itself. For example, physical constraints, such as minimum line width and maximum line width, that have an impact on the manufacturing as well as the functioning of the design. Such constraints are specified at the design stage and can then be modified while creating the physical layout. Similarly, information such as layers to be used by signal groups and trace proximity is also decided at the design stage.

This chapter describes how Constraint Manager, launched from a design capture tool can be used to capture, edit, and view physical and spacing constraints.

The topics covered in this chapter are:

Note: To know about the physical and spacing constraints and corresponding CSets, see Physical Constraints, Spacing Constraints, and Physical and Spacing CSets.

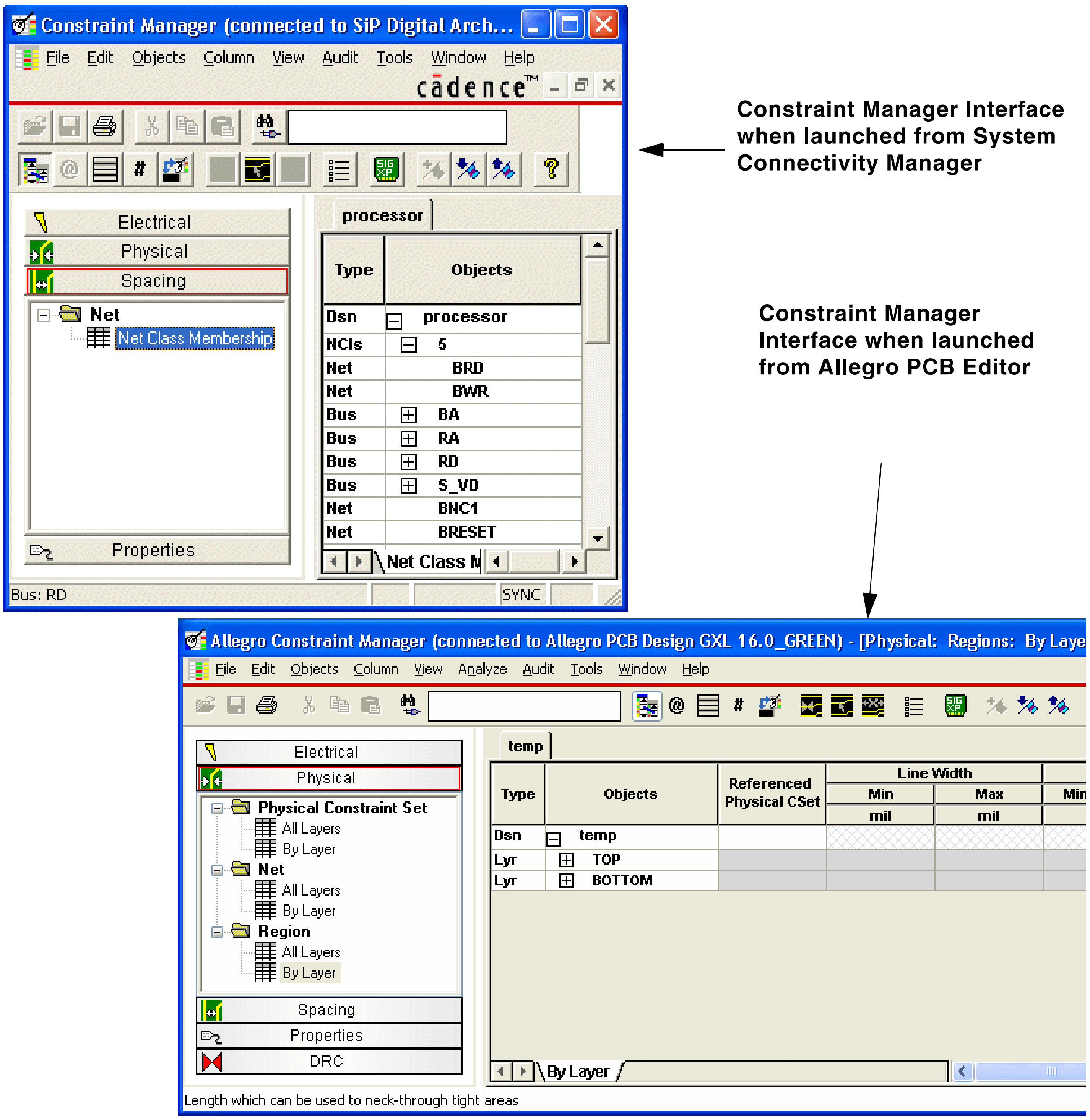
Working with Net Classes

Constraint Manager provides support for using classes to capture constraints. This is possible with constraint objects called net classes in Constraint Manager. A net class is created by grouping constraint objects with common features and similar constraints. Using classes allows you to quickly apply similar constraints on all the objects in the same class using CSets.

Using Net Classes

Depending on whether the net class is in Electrical, Physical, or Spacing domain, you can apply different CSets to each class. The CSets available depend on whether Constraint Manager is launched from a design capture tool or from a board design tool.

For example, when you launch Constraint Manager from a design capture tool, you can create net classes in the Physical and Spacing domain. You can also modify the membership of these classes by adding or removing nets from these classes.


Constraint Manager Interface when launched from System Connectivity Manager
Constraint Manager Interface when launched from Allegro PCB Editor
Constraint Manager (connected to SiP Digital Arch...
File Edit Objects Column View Audit Tools Window Help
cadence
Electrical
Physical
Spacing
Net
Net Class Membership
Properties
processor
Net Class M
SYNC
Allegro Constraint Manager (connected to Allegro PCB Design GXL 16.0_GREEN) - [Physical: Regions: By Laye
File Edit Objects Column View Analyze Audit Tools Window Help
Electrical
Physical
Physical Constraint Set
All Layers
By Layer
Net
All Layers
By Layer
Region
All Layers
By Layer
Spacing
Properties
DRC
temp
By Layer

Type	Objects
Dsn	⊟ processor
NCls	⊟ 5
Net	BRD
Net	BWR
Bus	⊞ BA
Bus	⊞ RA
Bus	⊞ RD
Bus	⊞ S_VD
Net	BNC1
Net	BRESET

Bus: RD

Type	Objects	Referenced Physical CSet	Line Width		
			Min	Max	Min
			mil	mil	
Dsn	⊟ temp				
Lyr	⊞ TOP				
Lyr	⊞ BOTTOM				

Length which can be used to neck-through tight areas

The net classes supported when you launch Constraint Manager from a design capture tool, are:

- Electrical Net Classes
- Physical and Spacing Net Classes

To know about the operations that can be performed on these net classes, see the *Constraint Manager Reference*. To know more about the new constraint objects, see *Constraint Manager User Guide*.

Electrical Net Classes

These are the net classes created for electrical constraints. Net classes created for electrical constraints are considered as local classes and support a logical name, which is then used to instantiate a unique net class in the higher-level design.

Physical and Spacing Net Classes

The physical and spacing net classes are considered global classes. There is only one instance of these classes in the design. Physical and Spacing net classes use physical names for processing lower-level constraints. By using these classes, you can now define physical constraints in hierarchical designs as well.

When Constraint Manager is invoked from a design capture tool, you can only add or modify the membership of the physical and spacing net classes. Applying constraints to these classes is supported only if physical and spacing constraints are enabled for the design. See Enabling Physical and Spacing Constraints.

If capturing physical and spacing constraints in not enabled, invoke Constraint Manager from Allegro PCB Editor and then capture physical and spacing constraints.

Creating Net Classes

To create a new net class in Constraint Manager, complete the following steps.

1. Select a workbook in the domain for which net class is to be created.
2. Choose *Object – Create – Net Class*.
3. In the *Create Net Class* dialog box, specify the name of the net class.
4. Click *OK*.

For more details, see *Constraint Manager Reference*.

Editing Net Class Membership

1. In the worksheet, right-click on the net class.
2. From the pop-up menu, choose *Membership – Net Class*.
3. In the dialog box, add or remove objects from the selected net class.

Capturing Physical and Spacing Constraints

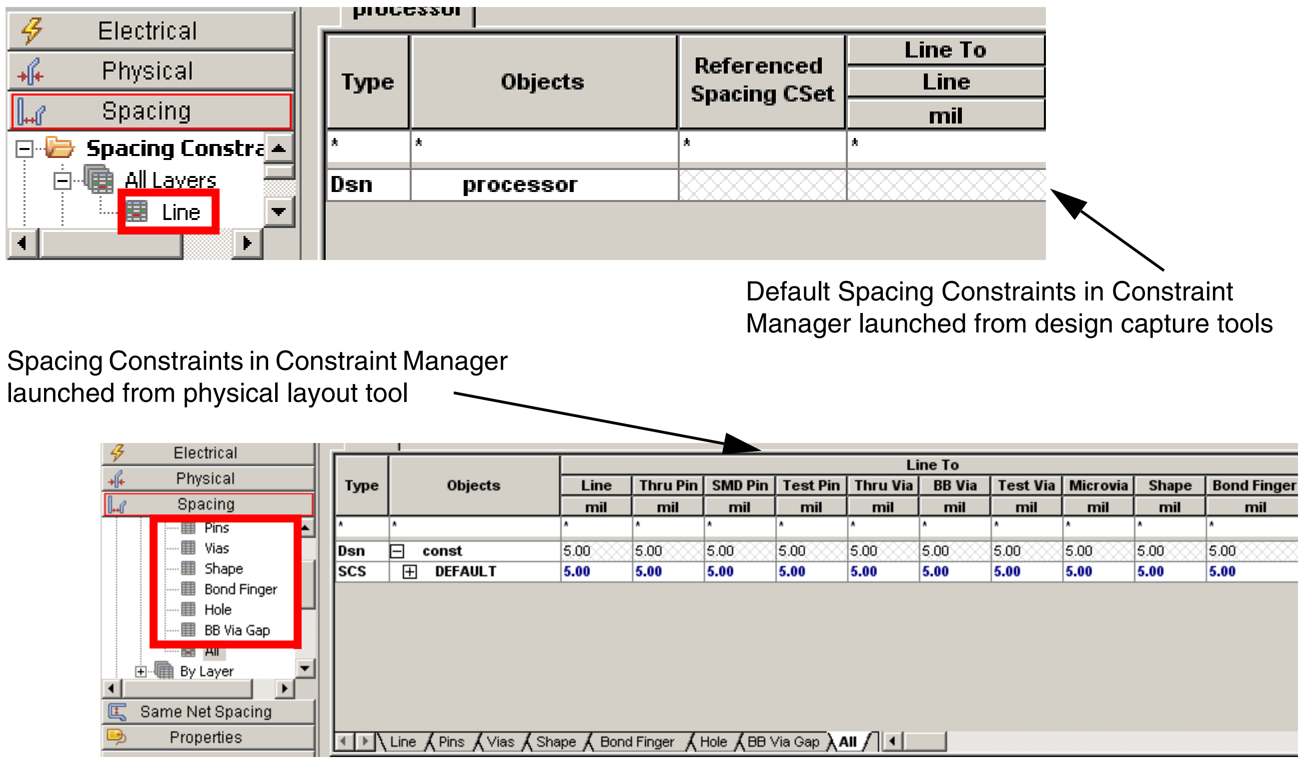
Form this release onwards, Constraint Manager launched from logic design tools can be used to view, capture, or edit the physical and spacing constraints. By default, only those constraints that impact the functioning of design are displayed in Constraint Manager launched from the design capture tool.

The Physical constraints that are visible by default are:

- Minimum and maximum line width
- Minimum Width and the maximum length of the Neck
- Primary gaps and neck gaps for differential pairs
- Allow Etch and Ts

The Spacing constraints that are visible by default are:

- Line-to-line spacing



Type	Objects	Referenced Spacing CSet	Line To Line mil
*	*	*	*
Dsn	processor		

Default Spacing Constraints in Constraint Manager launched from design capture tools

Spacing Constraints in Constraint Manager launched from physical layout tool

Type	Objects	Line To									
		Line	Thru Pin	SMD Pin	Test Pin	Thru Via	BB Via	Test Via	Microvia	Shape	Bond Finger
		mil	mil	mil	mil	mil	mil	mil	mil	mil	mil
*	*	*	*	*	*	*	*	*	*	*	*
Dsn	⊟ const	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00
SCS	⊞ DEFAULT	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00

Modes for Physical and Spacing Constraints

When Constraint Manager is launched from the design capture tools, physical and spacing constraints are available in two modes; Read-only mode and Edit mode.

Important

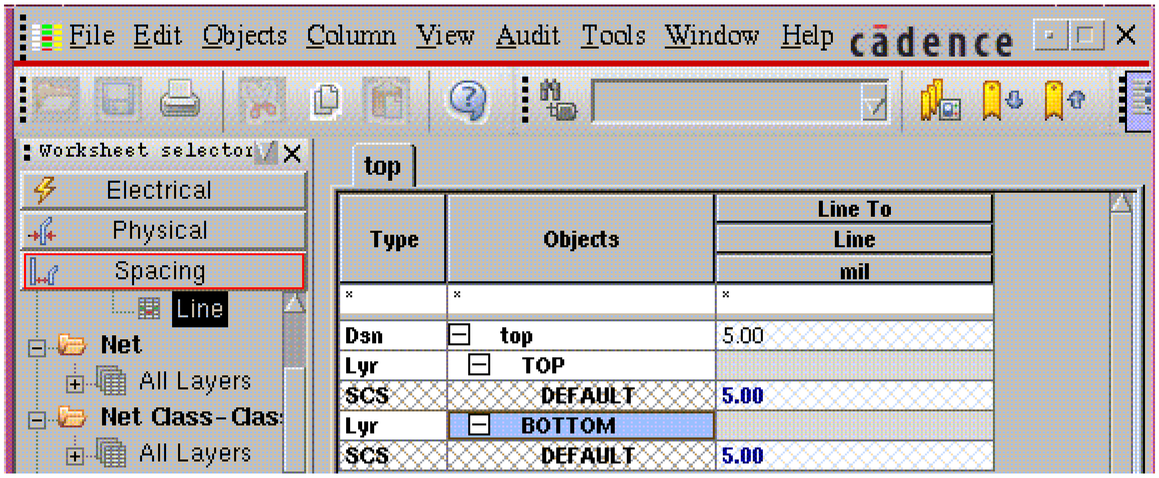
The read-only and edit modes discussed in this section are valid only for the physical and spacing constraints. These are not applicable to the electrical constraints displayed in Constraint Manager.

Read-Only Mode

In the read-only mode, you can only view the physical and spacing constraints in the Constraint Manager. In this mode, edit and delete operations are not supported for physical and spacing constraints.

If you run the importPhysical command in the read-only mode, modifications made to the constraints during the physical design stage are visible in Constraint Manager launched from

SCM, but cannot be modified. Similarly, on running the exportPhysical command, the physical and spacing constraints are not transferred to the physical layout of the design.

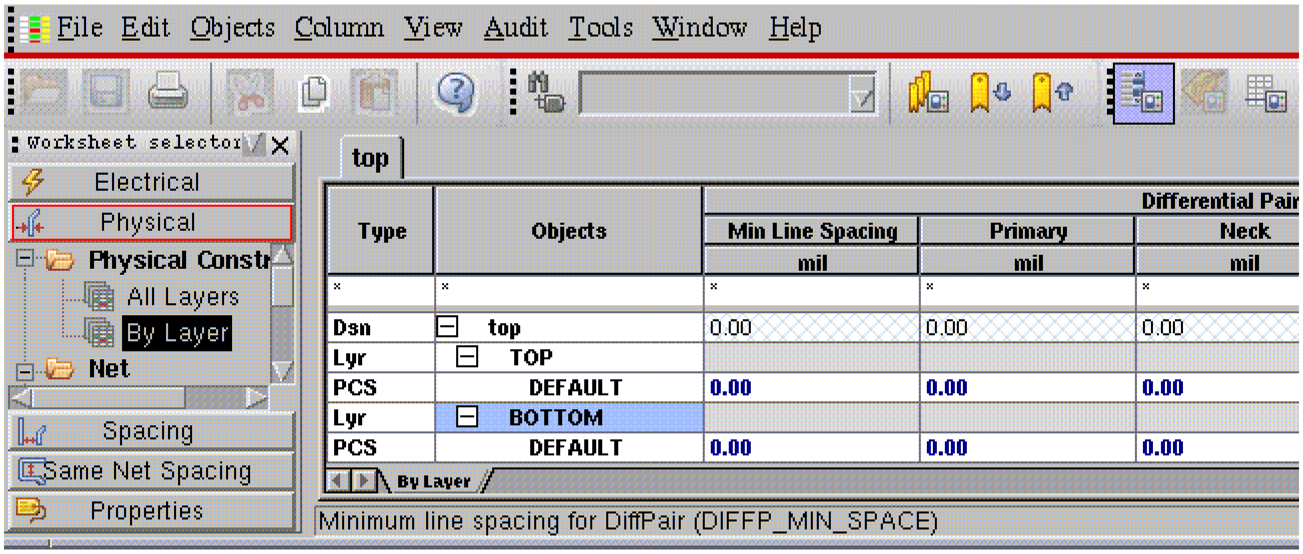


Type	Objects	Line To
		Line
		mil
*	*	*
Dsn	⊟ top	5.00
Lyr	⊟ TOP	
SCS	DEFAULT	5.00
Lyr	⊟ BOTTOM	
SCS	DEFAULT	5.00

Edit Mode

In this mode, you can view the constraints as well as modify them in Constraint Manager.

As you implement design logic in the design capture tool, you can also capture or modify physical and spacing constraints in Constraint Manager. You can add new physical and spacing constraints and also delete and modify the existing constraints. In the Edit mode, you can apply Constraint Sets (CSets) on the objects of physical and spacing worksheets. These constraints can then be passed on to the layout engineer.



Type	Objects	Differential Pair		
		Min Line Spacing	Primary	Neck
		mil	mil	mil
*	*	*	*	*
Dsn	⊟ top	0.00	0.00	0.00
Lyr	⊟ TOP			
PCS	DEFAULT	0.00	0.00	0.00
Lyr	⊟ BOTTOM			
PCS	DEFAULT	0.00	0.00	0.00

Minimum line spacing for DiffPair (DIFFP_MIN_SPACE)

Modifying physical and spacing constraints specified during the layout design stage is also supported.

Important

Layer information cannot be modified. On running Export Physical command to update an existing board, only constraint data is modified. Layers added or deleted during the design capture process are ignored.

Table 3-1 on page 76 lists the possible ways in which Constraint Manager can be launched from a design capture tool and it's impact of information flow.

Table 3-1 Constraint Manager Launched From Design Capture Tool

Capturing Physical & Spacing Constraints...	Constraint Manager Mode	Flow of Physical & Spacing Constraints
Enabled	Read-Only	Constraint information updated in the back-to-front flow.
Enabled	Edit	Constraint information shared in the front-to-back flow as well as in the back-to-front flow.

Enabling Physical and Spacing Constraints

This section describes the steps to be performed to enable capturing or modifying of physical and spacing constraints using Constraint Manager launched from the design capture tool.

Though, you can create physical and spacing net classes and also define their membership, but actual capturing or modifying of physical and spacing constraints using Constraint Manager launched from a design capture is enabled if the following two conditions are satisfied.

- In the *`project`*`.cpm` file, the value of the `EDIT_PHYSICAL_SPACING_CONSTRAINTS` directive must be set to ON.

  The `.cpm` file must have the following lines.

  ```
  START_CONSTRAINT_MGR
  EDIT_PHYSICAL_SPACING_CONSTRAINTS 'ON'
  END_CONSTRAINT_MGR
  ```

- The stackup data for the board file is available in Design Entry HDL. System Connectivity Manager.

  The layer definition or the stackup data has an impact on the physical and spacing constraints. Therefore, these constraints are available in Constraint Manager only if the layer information is available in the design.

  To see how layer information can be made available in design capture tools, see Importing Stackup Data.

Important

In designs created in the 16.2 release and later, physical and spacing constraints are enabled by default. For these designs, the default value of the `EDIT_PHYSICAL_SPACING_CONSTRAINTS` directive is `ON`. However, if you create a new design in System Connectivity Manager and launch Constraint Manager, physical and spacing constraints are not visible till layer information is imported in the design.

Importing Stackup Data

Capturing and editing of physical and spacing constraints is enabled only if the layer or the stackup information is available during the design capture stage. This information can be made available using one of the following methods.

- Importing Physical Data

- Importing Technology File

Importing Physical Data

This method is recommended for designs for which the physical layout has already been created and the logical and physical designs are in sync. In case of such designs, it is recommended that you run Import Physical and update the logical design with changes in the physical design.

Along with layer information, Physical and Spacing CSets are also imported. The information imported in the logical design can be categorized as follows.

- Layer Stackup Changes
- Physical CSet Differences
- Spacing CSet Differences
- Same Net Spacing CSet Differences
- Physical CSet Association Differences
- Spacing CSet Association Differences
- Same Net Spacing CSet Association Differences

Importing Technology File

For new designs, layer information can be included ny importing the technology file that is to be used for creating the physical design.

1. Launch Constraint Manager.
2. Choose *File – Import Technology File*.
3. Specify the technology file to be imported.
4. Select the mode in which the technology file is to be imported.

Tip

If you want to view the modification that will be made before performing the actual import, select the *Report Only* check box.

5. Click *Open*.

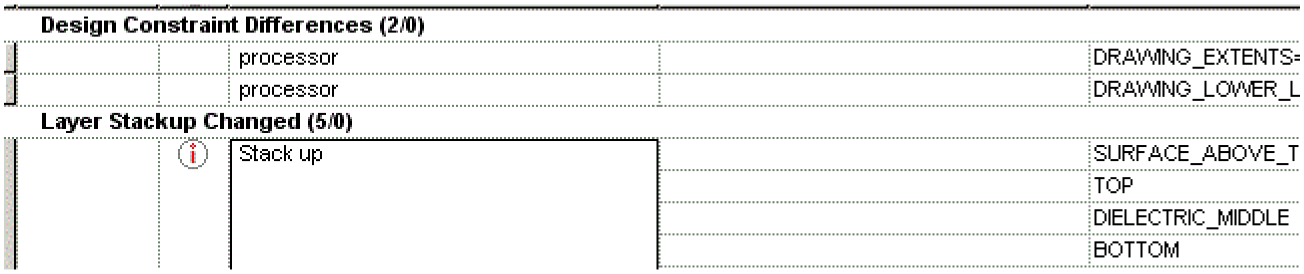
To know more about using technology files, see the *Using Technology and Parameters Files* chapter in the *Defining and Developing Libraries* guide.

Capturing Constraints in Pre-16.2 Designs

To enable physical and spacing constraints in designs that were created in releases prior to 16.2, do the following:

1. Open the `.cpm` file in a text editor, add the `EDIT_PHYSICAL_SPACING_CONSTRAINTS` directive, set its value to ON, save and close the file.
2. Open the design in the design capture tool.
3. Use the import physical command to import layer information.

   If the physical design is in sync with the logical design, following changes are imported in the logical design.



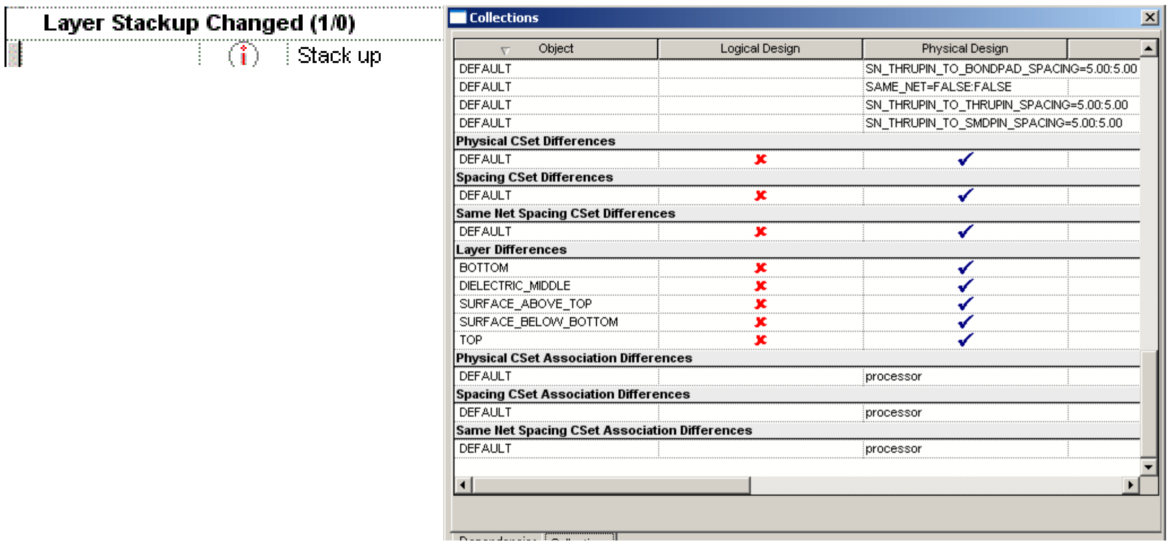
Design Constraint Differences (2/0)

		processor		DRAWING_EXTENTS=
		processor		DRAWING_LOWER_L

Layer Stackup Changed (5/0)

	ⓘ	Stack up		SURFACE_ABOVE_T
				TOP
				DIELECTRIC_MIDDLE
				BOTTOM

   Note that along with the constraint differences, layer stackup changes are also imported. To view the modification details, click ⓘ . The modification details are displayed in the Collections window.



Layer Stackup Changed (1/0)

ⓘ Stack up

Collections

Object	Logical Design	Physical Design
DEFAULT		SN_THRUPIN_TO_BONDPAD_SPACING=5.00:5.00
DEFAULT		SAME_NET=FALSE:FALSE
DEFAULT		SN_THRUPIN_TO_THRUPIN_SPACING=5.00:5.00
DEFAULT		SN_THRUPIN_TO_SMDPIN_SPACING=5.00:5.00
Physical CSet Differences		
DEFAULT	✗	✓
Spacing CSet Differences		
DEFAULT	✗	✓
Same Net Spacing CSet Differences		
DEFAULT	✗	✓
Layer Differences		
BOTTOM	✗	✓
DIELECTRIC_MIDDLE	✗	✓
SURFACE_ABOVE_TOP	✗	✓
SURFACE_BELOW_BOTTOM	✗	✓
TOP	✗	✓
Physical CSet Association Differences		
DEFAULT		processor
Spacing CSet Association Differences		
DEFAULT		processor
Same Net Spacing CSet Association Differences		
DEFAULT		processor

4. To update the logical design with all the changes, select Update All.

The design is updated with the layer stackup data, PCSets and ECSets

Important

For a design, if the precision values in the logical and physical design are different, on running the Import Physical command, this is reported in Visual Design difference window as Design Unit Differences. Updating this ensures that the precision value used in a logical design is in sync with the precision value used in the physical design.

Editing Physical and Spacing Constraints

If the conditions listed in the Enabling Physical and Spacing Constraints section are satisfied, the physical and spacing constraints are visible in Constraint Manager launched from the design capture tool. These constraints can also be edited.

If you now run the Export Physical command to synchronize the logical and the physical design, all the physical and spacing constraints captured or modified in the design capture tool will be available in Constraint Manager launched from the physical layout tool.

Important

Layer information cannot be modified. On running the Export Physical command to update an existing board, only constraint data is modified. Layers added or deleted during the design capture process are ignored.

Editing Constraints in a Hierarchical Design

Which capturing physical and spacing constraints for an hierarchical design, following needs to be taken care of.

- It is recommended that the layer stackup of the lower-level blocks should be same as the layer stackup for the root design.
- In case the layer stackup for lower-level blocks is different from the layer stackup for root design, following rules are applied for merging physical and spacing constraints.
    - Name mapping is used to identify the layers.
    - Addition or deletion of layers is not supported.

- ❑ If a layer is available for a lower-level block but is missing from the stackup of the root design, the constraint information in the layer is ignored.

Switching Modes for Physical and Spacing Constraints

In designs created in 16.2 and later, physical and spacing constraints are enabled by default, and are in the edit mode. To switch Constraint Manager from the edit mode to read-only mode and vice-versa, the *`project.cpm`* file must be manually edited.

Switching From Edit to Read-Only Mode

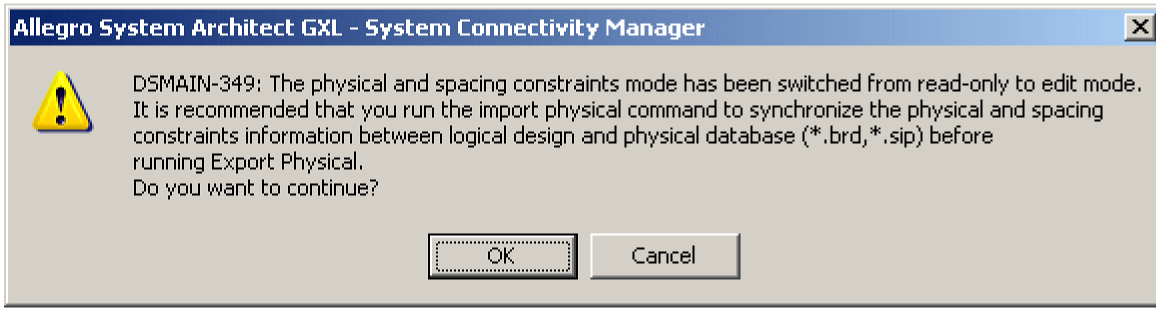
In Read-Only Mode, the physical and spacing constraints information in logical design is not shared with the physical design. Therefore, it is recommended that before you switch from edit mode to read-only mode, run the Export Physical command to ensure that the physical design is updated with the constraint modifications made in the logical design.

To switch from edit mode to read-only mode, edit the *`project.cpm`* file and set the value of the `EDIT_PHYSICAL_SPACING_CONSTRAINTS` to OFF.

Switching From Read-Only to Edit Mode

When you change from read-only mode to edit mode, a physical or a spacing constraint can be modified while making changes to the logical design as well as while modifying the physical design. Depending on whether you export the design changes or import the modifications made to the physical design, there are changes that a actual constraint value is overwritten by a stale value.

To prevent such scenarios, it is recommended that before you switch from the read-only mode to edit mode, use the importPhysical command to update the logical design with the latest constraint values from the physical design. The message displayed when you run Export Physical command for the first time after switching modes, reiterates this recommendation.



DSMAIN-349: The physical and spacing constraints mode has been switched from read-only to edit mode.
It is recommended that you run the import physical command to synchronize the physical and spacing constraints information between logical design and physical database (*.brd,*.sip) before running Export Physical.
Do you want to continue?

Caution

Changing modes of Constraint Manager launched from design capture tool requires editing of the .cpm file. Therefore, frequent switching of modes is not recommended. Instead, it is recommended that the `EDIT_PHYSICAL_SPACING_CONSTRAINTS` ***directive must be specified at the site level in site.cpm and directive locking should be used to prevent frequent changing of modes.***

4

ECSet in SigXplorer

Why SigXplorer?

While capturing a design constraints in Constraint Manager, you can view the entire topology on a object in SigXplorer, which is the tool used to create, modify, simulate, and save prototypes of net topologies. A topology can be defined as a representation of how signals are physically and electrically connected. When you launch SigXplorer on an object in Constraint Manager, the driver-receiver relationship along with its connectivity details are created in SigXplorer. While capturing your designs, if you create a topology that meets the circuit specifications, you can extract the topology in SigXplorer, and save it as a topology template for reuse in future.

For detailed information on topology templates in Constraint Manager, see *ECSets and Topology Templates* in *Allegro Constraint Manager User Guide*.

For topology extraction and constraint modification, you can launch SigXplorer on design object. You can also validate and generate electrical constraint sets (ECSets) in SigXplorer. These ECSets can then be applied on other Xnets or net objects in Constraint Manager and validated. When Constraint Manager is invoked from a design capture tool, you can launch SigXplorer you can launch SigXplorer on following design objects.

- ❑ nets
- ❑ Xnets
- ❑ pin pairs
- ❑ Buses (SigXplorer displays the topology on the first bit)
- ❑ classes
- ❑ differential pairs
- ❑ ECSets

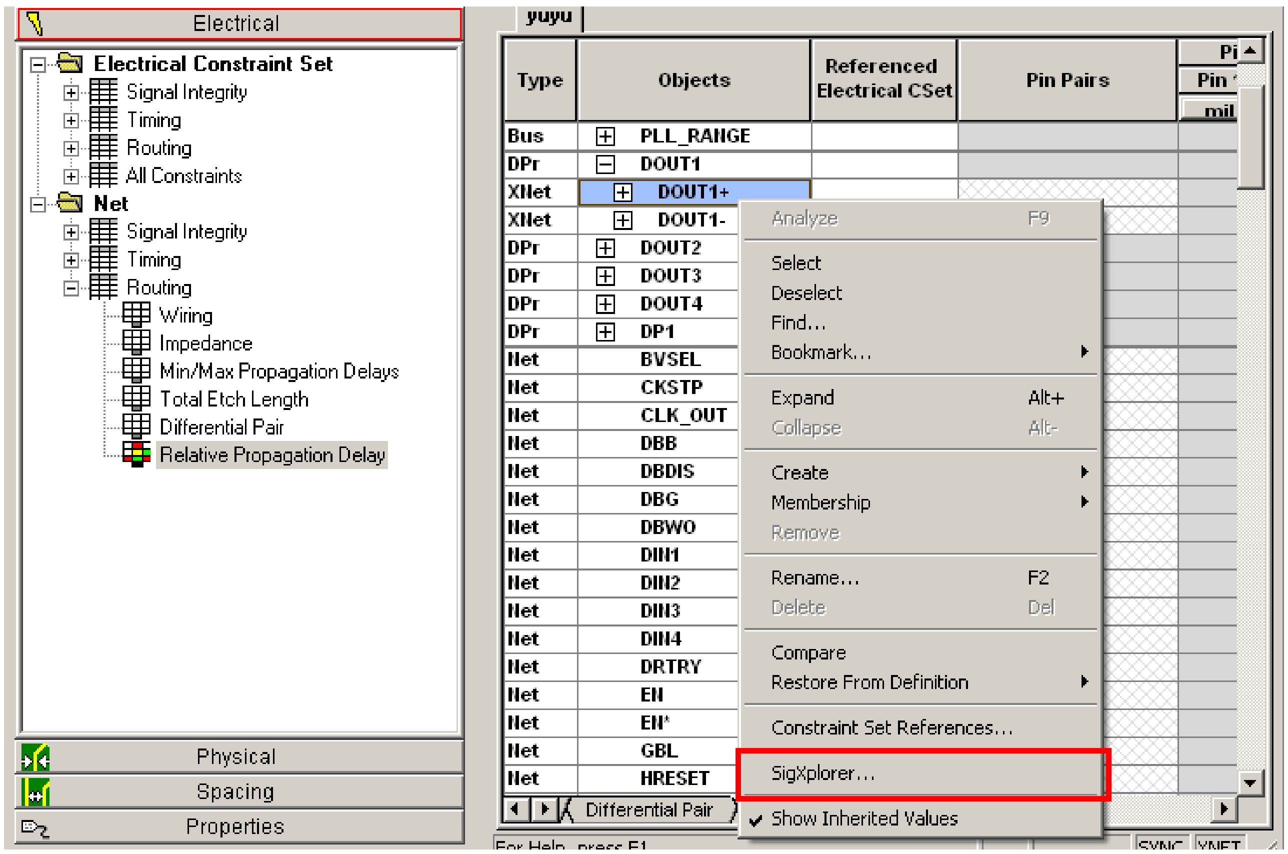
Important

To launch SigXplorer on model-defined differential pair objects, it is recommended that you use a SigXplorer license that is higher than the default license shipped with design capture tools. To change the product suite, set the value of the environment variable SIGXP_TIER to the product number for the high-end license. For example, `SIGXP_TIER = PA5630`

Viewing Topology in SigXplorer

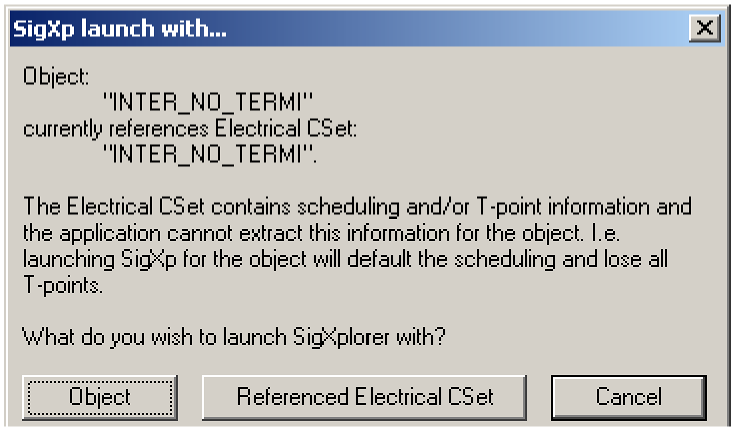
Constraint Manager provides support for viewing the valid models in the SigXplorer canvas. To launch SigXplorer on a design object in Constraint Manager, perform the following steps.

➤ Right-click on the design object and from the pop-up menu choose *SigXplorer*.



Type	Objects	Referenced Electrical CSet	Pin Pairs
Bus	PLL_RANGE		
DPr	DOUT1		
XNet	DOUT1+		
XNet	DOUT1-		
DPr	DOUT2		
DPr	DOUT3		
DPr	DOUT4		
DPr	DP1		
Net	BVSEL		
Net	CKSTP		
Net	CLK_OUT		
Net	DBB		
Net	DBDIS		
Net	DBG		
Net	DBWO		
Net	DIN1		
Net	DIN2		
Net	DIN3		
Net	DIN4		
Net	DRTRY		
Net	EN		
Net	EN*		
Net	GBL		
Net	HRESET		

If you have an ECSet applied to the design object, following dialog box appears.



SigXp launch with...

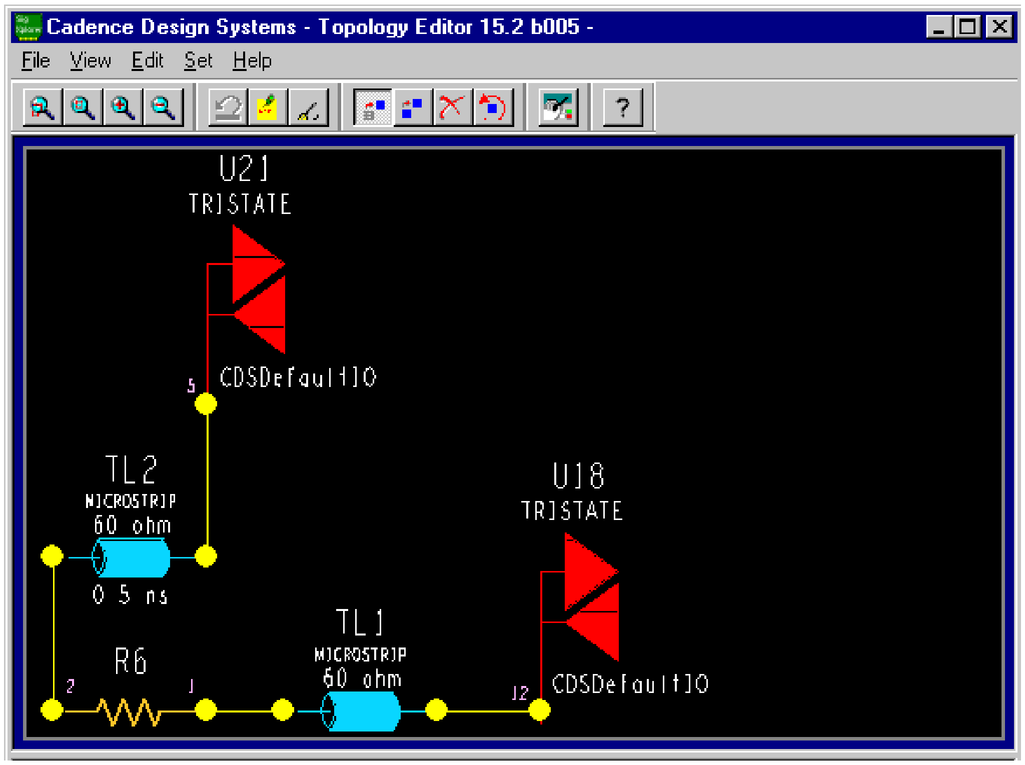
Object:
"INTER_NO_TERMI"
currently references Electrical CSet:
"INTER_NO_TERMI".

The Electrical CSet contains scheduling and/or T-point information and the application cannot extract this information for the object. I.e. launching SigXp for the object will default the scheduling and lose all T-points.

What do you wish to launch SigXplorer with?

- ❑ Select *Object*, if you want to launch SigXplorer on the object.
- ❑ Select *Referenced Electrical Set* if you want to modify the topology and constraints defined in the ECSet.

SigXplorer is invoked and displays the design object and valid models assigned to the devices.



Note: If a discrete device is not assigned a valid signal model, SigXplorer will not launch on the Xnet.

Differential Pairs in SigXplorer

When you launch SigXplorer on a differential pair, the topology depends on whether the differential pair is model-defined, library-defined, or user-defined.

For a model-defined differential pair, both the legs of the differential pair are extracted in SigXplorer. In case of library-defined or user-defined differential-pairs, only one leg of the differential pair is extracted in SigXplorer.

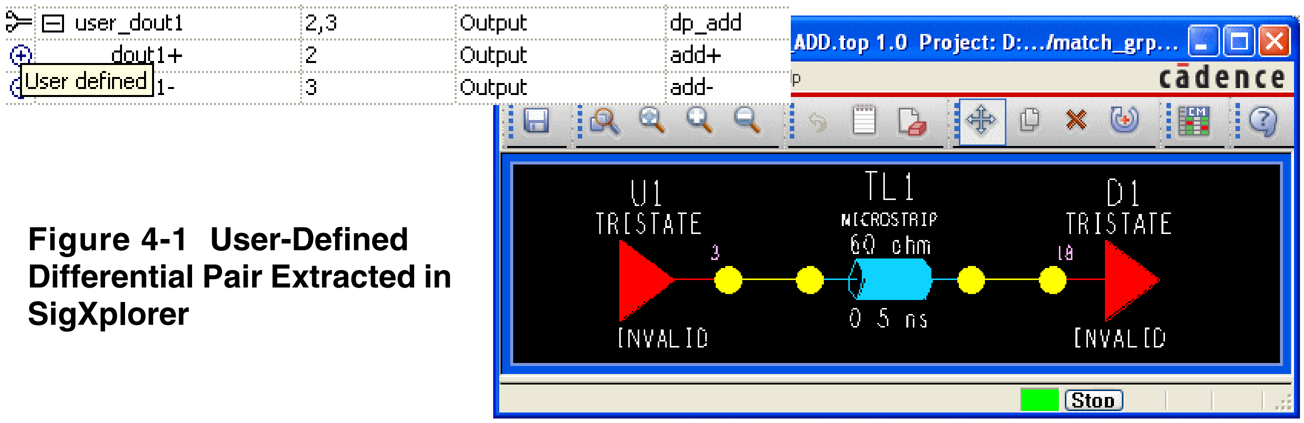


Figure 4-1 User-Defined Differential Pair Extracted in SigXplorer

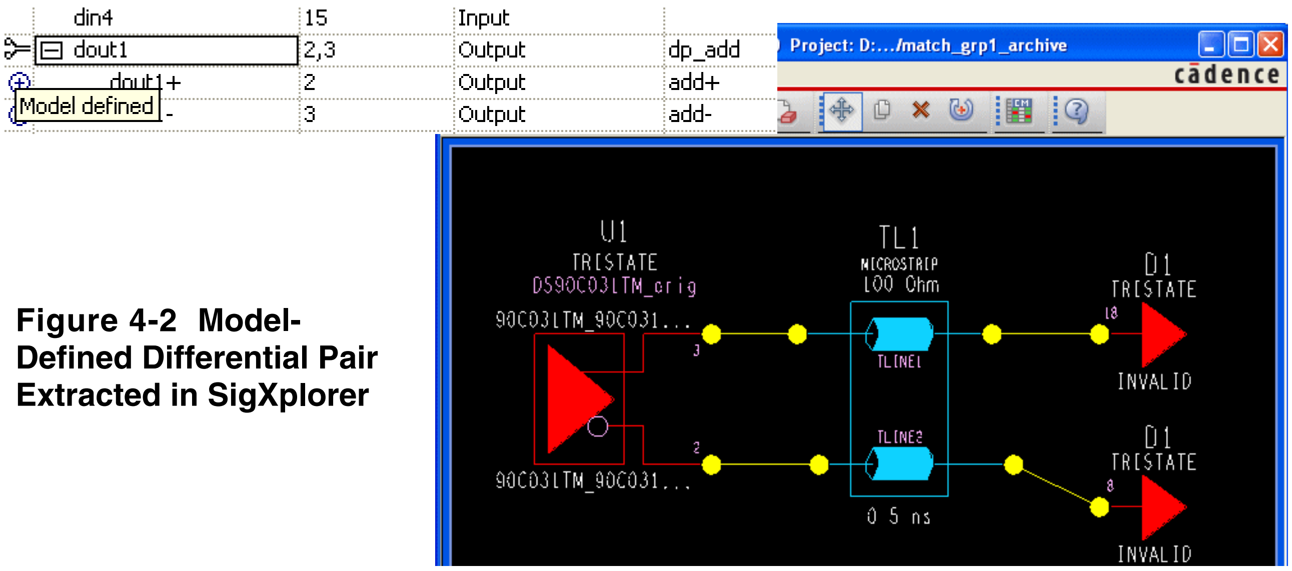


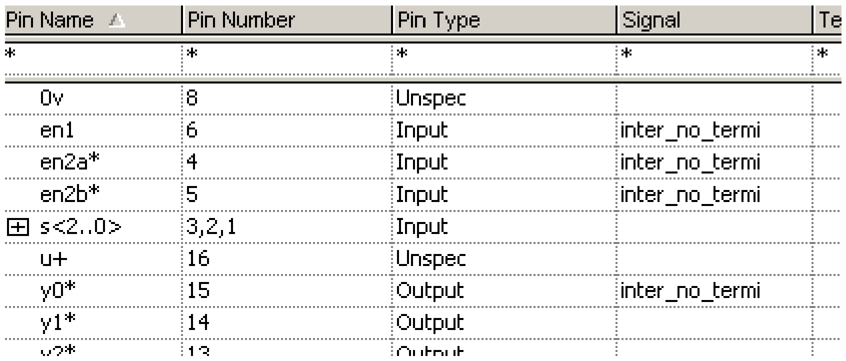
Figure 4-2 Model-Defined Differential Pair Extracted in SigXplorer

Inserting T-Points in SigXplorer

SigXplorer reads electrical constraints on a net or Xnet, which you can modify. You can also create new topology files to capture electrical constraints in SigXplorer.

During the design capture stage, if have a one-to-many connection from a driver to receiver, you can use the SigXplorer to modify the topology, such that all receivers receive the signal in parallel at with the same delay value.

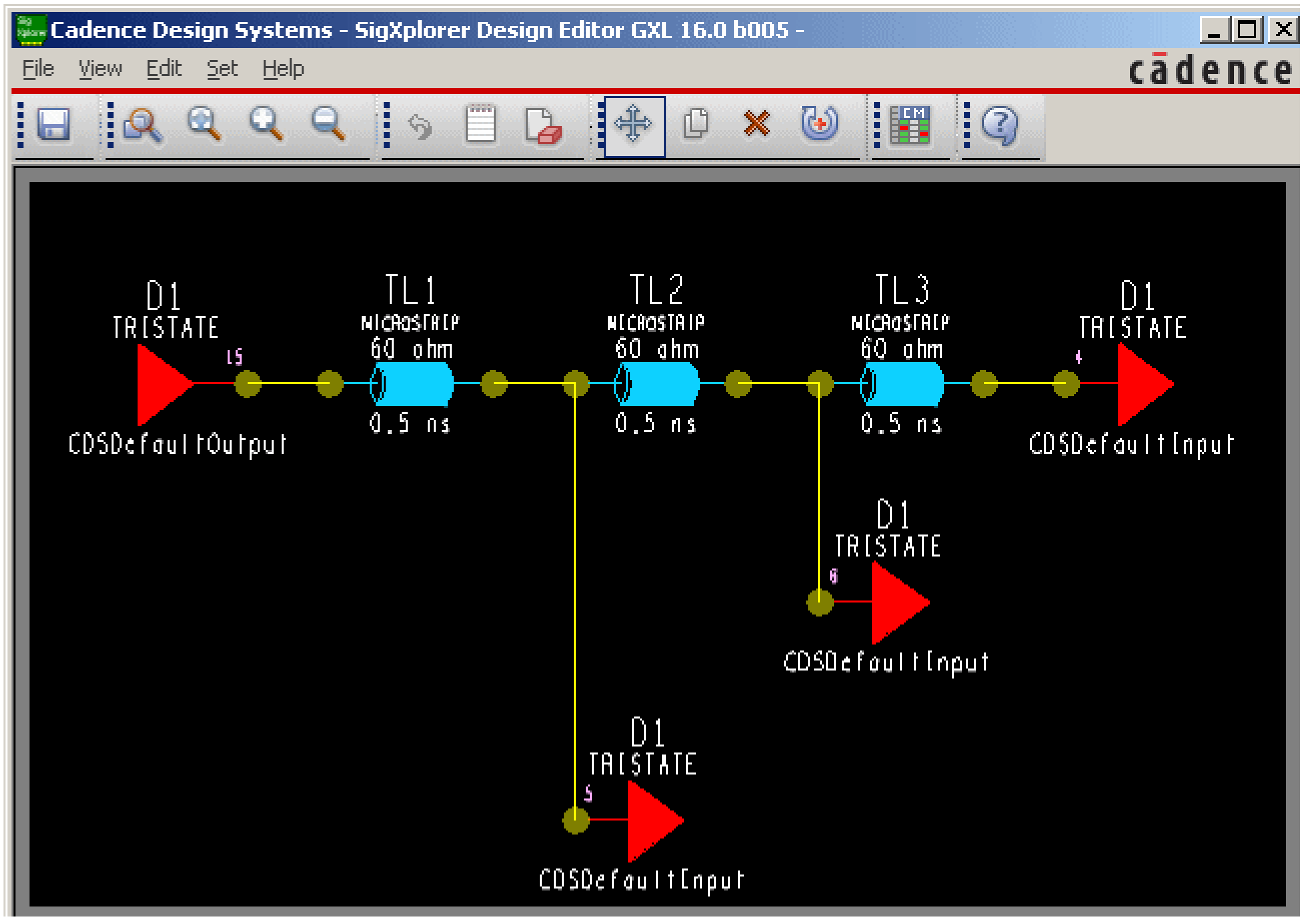
A design that has one driver connected multiple receivers is shown in the figure below.



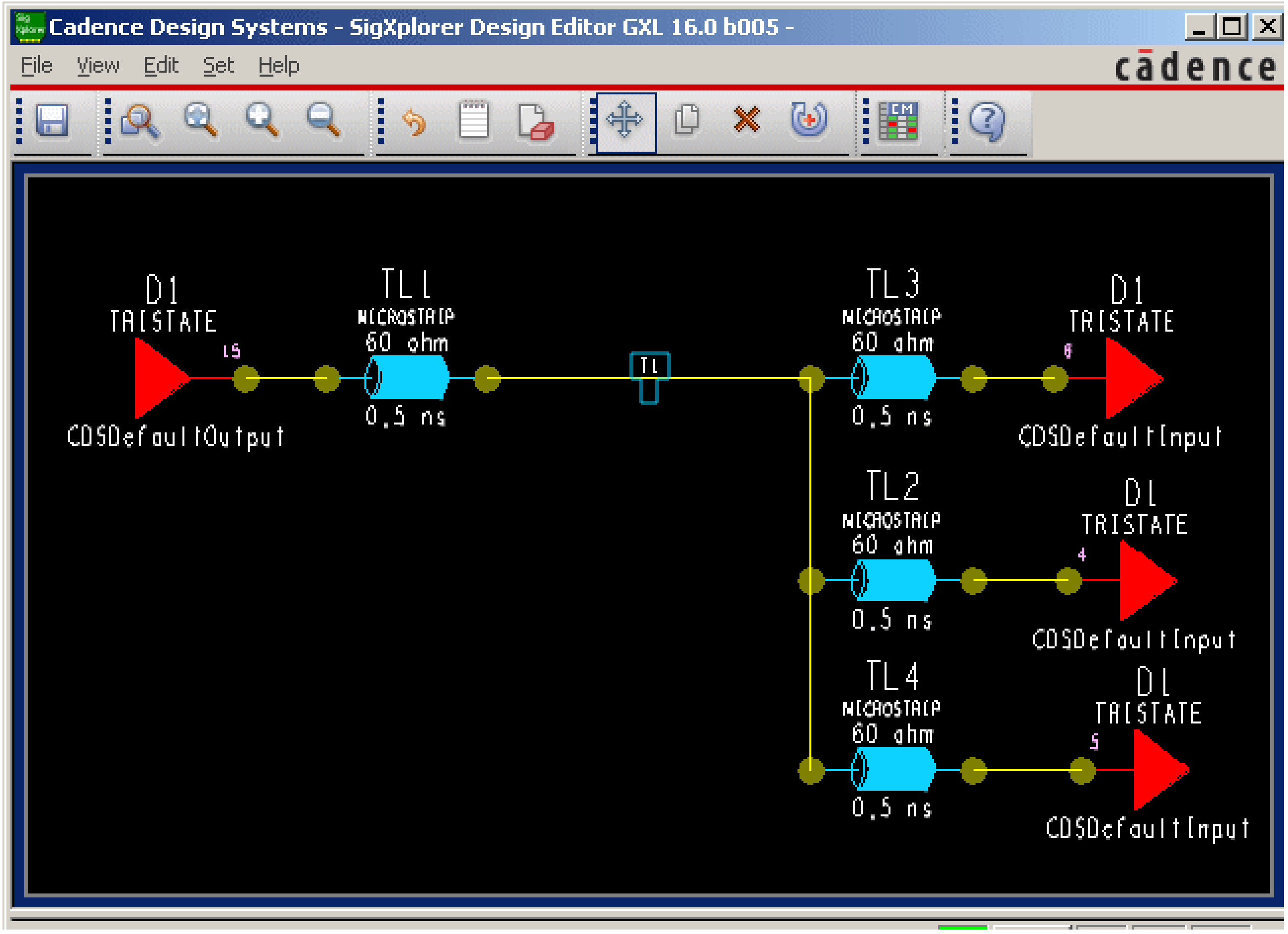
Pin Name	Pin Number	Pin Type	Signal	Te
*	*	*	*	*
0v	8	Unspec		
en1	6	Input	inter_no_termi	
en2a*	4	Input	inter_no_termi	
en2b*	5	Input	inter_no_termi	
⊞ s<2..0>	3,2,1	Input		
u+	16	Unspec		
y0*	15	Output	inter_no_termi	
y1*	14	Output		
y2*	13	Output		

Figure 4-3 Design with One-To-Many Driver to Receiver connection

If you now invoke Constraint Manager and launch SigXplorer on the net, the topology of the net is as shown in the figure below.



If you now modify the topology, such that the final topology is as shown in the figure below.



As you modify the topology, note that a new element, T-point, gets added automatically. The topology modifications are saved as topology (`.top`) file. The advantage of using T-points is that you can specify different separate propagation delay constraints for net segments from driver to t-point and from t-point to each receiver.

If you now update Constraint Manager (see Updating Constraint Manager with Changes in SigXplorer) with these changes, they are imported as an ECSet. Next time when you launch SigXplorer on the ECSet, the topology of the net is same as shown in figure above.

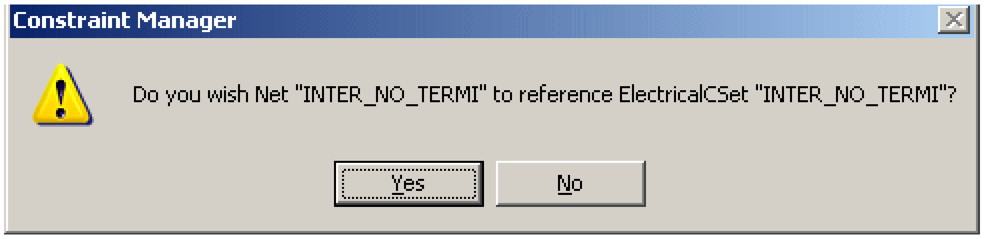
Updating Constraint Manager with Changes in SigXplorer

After you have made changes to design constraints in SigXplorer, you can import these changes to Constraint Manager. All topology changes in SigXplorer are imported in Constraint Manager as ECSets.

To update Constraint Manager with the constraint modifications in SigXplorer, perform the following step.

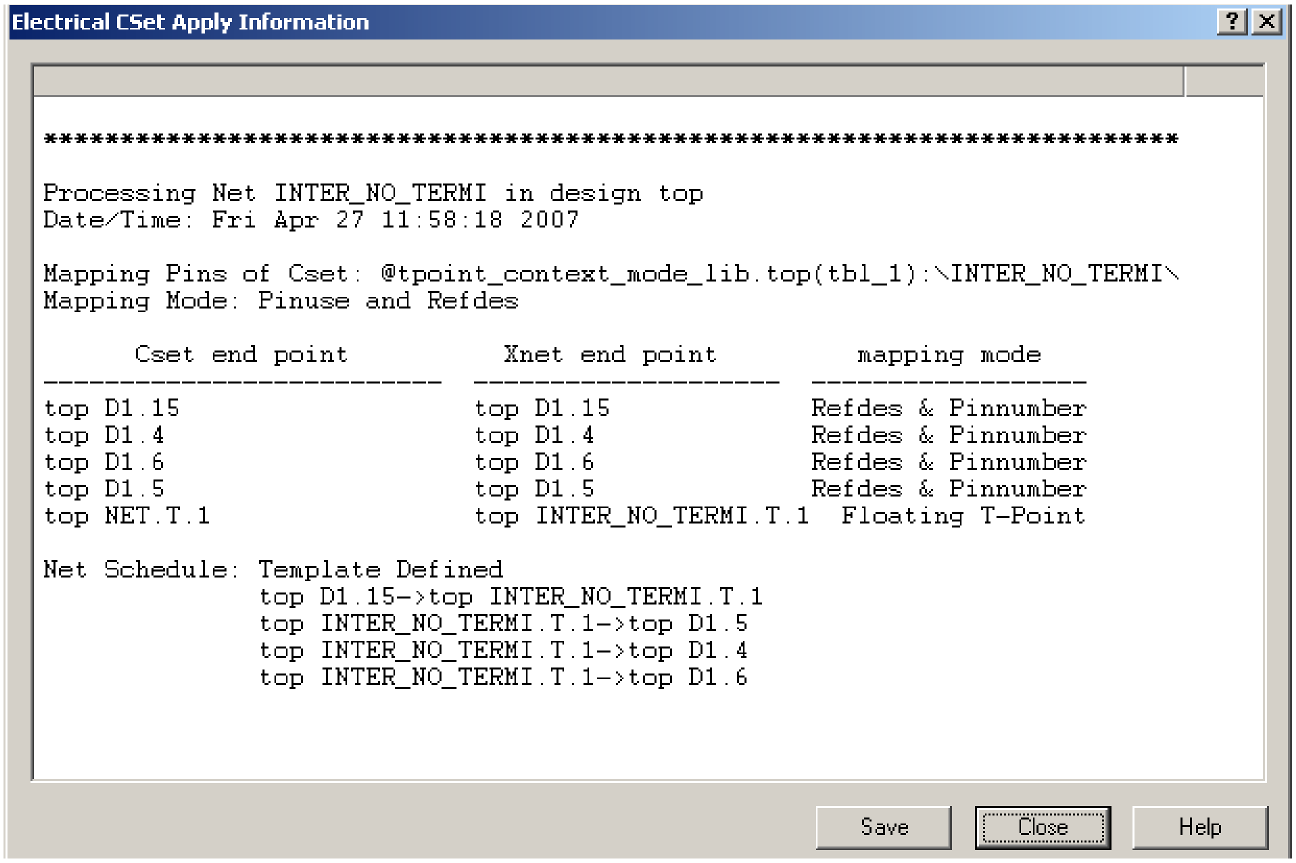
➤ In SigXplorer, choose *File – Update Constraint Manager.*

A new ECSet is created in Constraint Manager. A message box is thrown asking whether the new ECSet should be applied on the object on which SigXplorer was initially launched.



Do you wish Net "INTER_NO_TERMI" to reference ElectricalCSet "INTER_NO_TERMI"?

You might or might not choose to apply ECSet on the object. If you apply the ECSet a message box is displaying the status appears.



```
**************************************************************************

Processing Net INTER_NO_TERMI in design top
Date/Time: Fri Apr 27 11:58:18 2007

Mapping Pins of Cset: @tpoint_context_mode_lib.top(tbl_1):\INTER_NO_TERMI\
Mapping Mode: Pinuse and Refdes

      Cset end point              Xnet end point          mapping mode
--------------------------    --------------------    ------------------
top D1.15                     top D1.15               Refdes & Pinnumber
top D1.4                      top D1.4                Refdes & Pinnumber
top D1.6                      top D1.6                Refdes & Pinnumber
top D1.5                      top D1.5                Refdes & Pinnumber
top NET.T.1                   top INTER_NO_TERMI.T.1  Floating T-Point

Net Schedule: Template Defined
              top D1.15->top INTER_NO_TERMI.T.1
              top INTER_NO_TERMI.T.1->top D1.5
              top INTER_NO_TERMI.T.1->top D1.4
              top INTER_NO_TERMI.T.1->top D1.6
```

Caution

Termination points created in SigXplorer are not backannotated to Constraint Manager or Design Entry HDL.

For more information of SigXplorer, refer to Allegro SI SigXplorer User Guide.

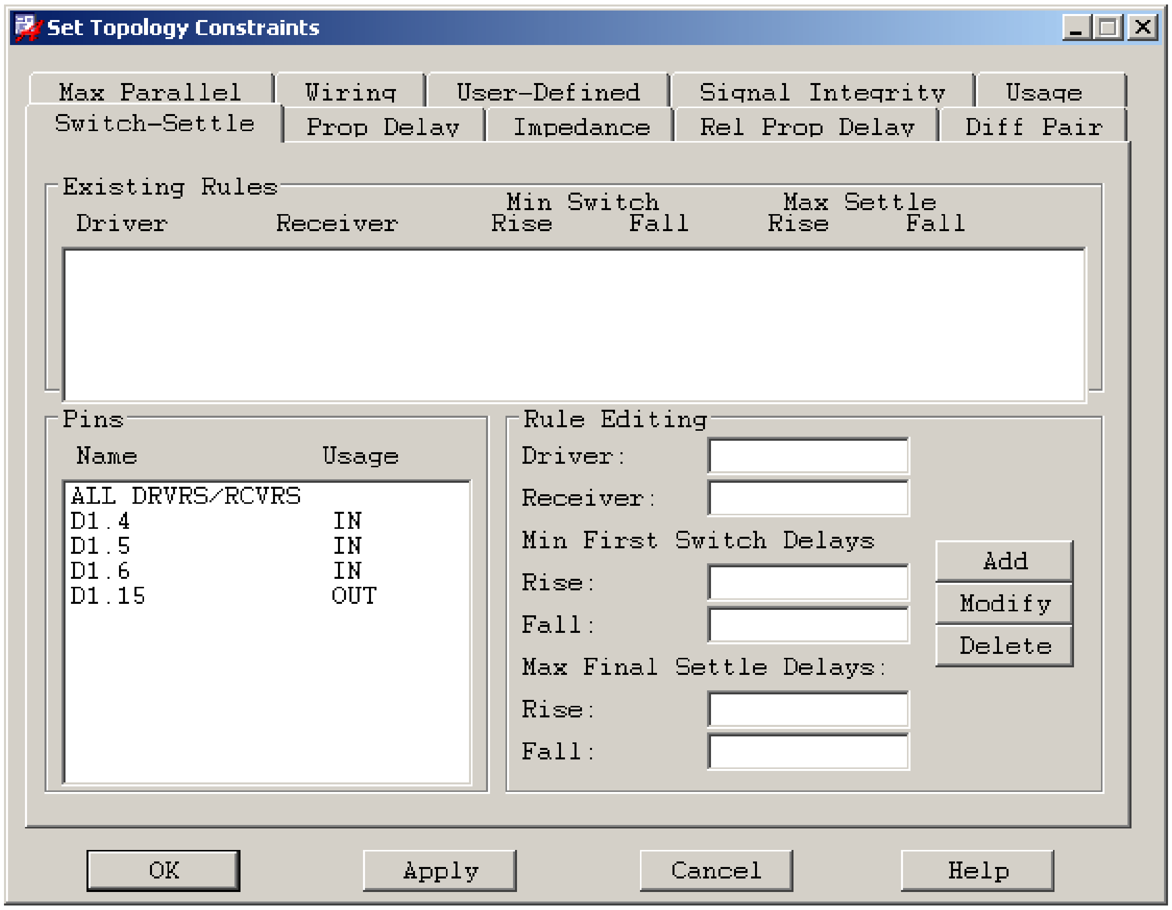
Modifying Electrical Constraints in SigXplorer

Besides modifying the topology, SigXplorer also provides support for modifying constraints on the topology. In SigXplorer, you can generate or modify electrical constraints as a topology file. The constraints added or modified in SigXplorer, as a topology template, is imported in Constraint Manager as an ECSet. Conversely, you can define your constraints in Constraint Manager, as an ECSet, and then export this information to SigXplorer as a topology template.

To add constraint information of an ECSet in SigXplorer, perform the following steps:

1. Launch SigXplorer on ECSet.
2. Choose *Set – Topology Constraints.*

The Set Topology Constraints dialog box is displayed for you. You can use the tabbed pages of this dialog box, to set various constraints such as minimum and maximum propagation delay, impedance, differential pair constraints, and so on.



Name	Usage
ALL DRVRS/RCVRS	
D1.4	IN
D1.5	IN
D1.6	IN
D1.15	OUT

3. Select the required tab.

   For example, to specify the differential pair constraints, select the *Diff Pair* tab.

4. Specify the constraints values.

   For example, in the Diff Pair page, add values in the Line Width text box to specify the line width of a differential pair signal.

5. Click *Apply*.

   If required, select any other tab and specify the constraint values. While adding new constraints in the Switch-Settle, Prop Delay, Impedance, Max Parallel and User Defined pages of the Set Constraint dialog box, you need to specify the constraint details in the Rule Editing section and then click the *Add* button.

Similarly, while modifying constraints, select an entry in the Existing Rules section, modify the constraint values in the Rule Editing section, and then select the *Modify* button.

6. Click *OK*.
7. To save the constraint information as a topology (.top) file, choose *File – Save* or *File – Save As*.

   Note: If you are using Design Entry HDL to capture your design, you can specify pin-pair constraints on unpackaged components. The notation used to identify pins on unpackaged components is, `<page no.>_<location of the pin>.<pin number>`. For example, `1_I5.1` represents pin number 1 of an unpackaged component on page 1 at location I5.

To know more about specifying constraints in SigXplorer, see to *Allegro SI SigXplorer User Guide*.

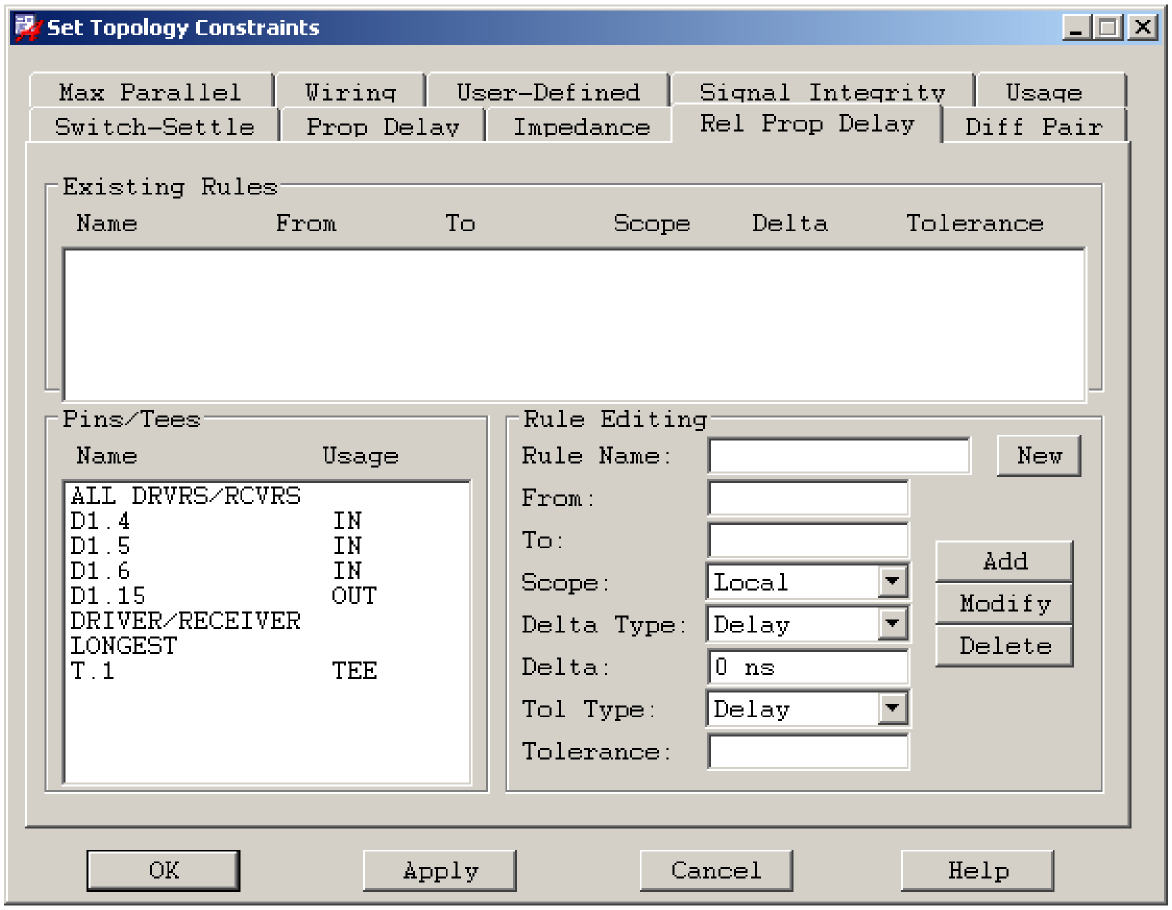
Creating ECSet Generated Match Groups in SigXplorer

While editing constraints in SigXplorer, if you modify or add a rule to specify the relative propagation delay between a pin-pair, and then update Constraint Manager with these changes, a new ECSet and a match group are created in Constraint Manager.

While you modify the relative propagation delay constraints, you can also add new rule that will create ECSet generated match groups, using the following steps.

1. Launch SigXplorer on a design object.
2. Choose *Set – Constraints*.

3. In the Set Topology Constraints window, select the *Rel Prop Delay* tab.



Name	Usage
ALL DRVRS/RCVRS	
D1.4	IN
D1.5	IN
D1.6	IN
D1.15	OUT
DRIVER/RECEIVER	
LONGEST	
T.1	TEE

Scope: Local
Delta Type: Delay
Delta: 0 ns
Tol Type: Delay

4. Click *New*.

The Rule Name field in the Rule Editing section gets populated.

5. To specify the pins between which you want to specify the relative propagation delay, select appropriate pins from the Pin/Tee section for From and To fields.

6. Similarly, specify other values for all other fields in the Rule Editing section and click *Add*.

The relative propagation delay constraint that you have created gets added in the Existing Rules section.

7. Click *OK*.

8. Update Constraint Manager with these modifications.

To view the steps for updating Constraint Manager with modifications in SigXplorer, see Updating Constraint Manager with Changes in SigXplorer.

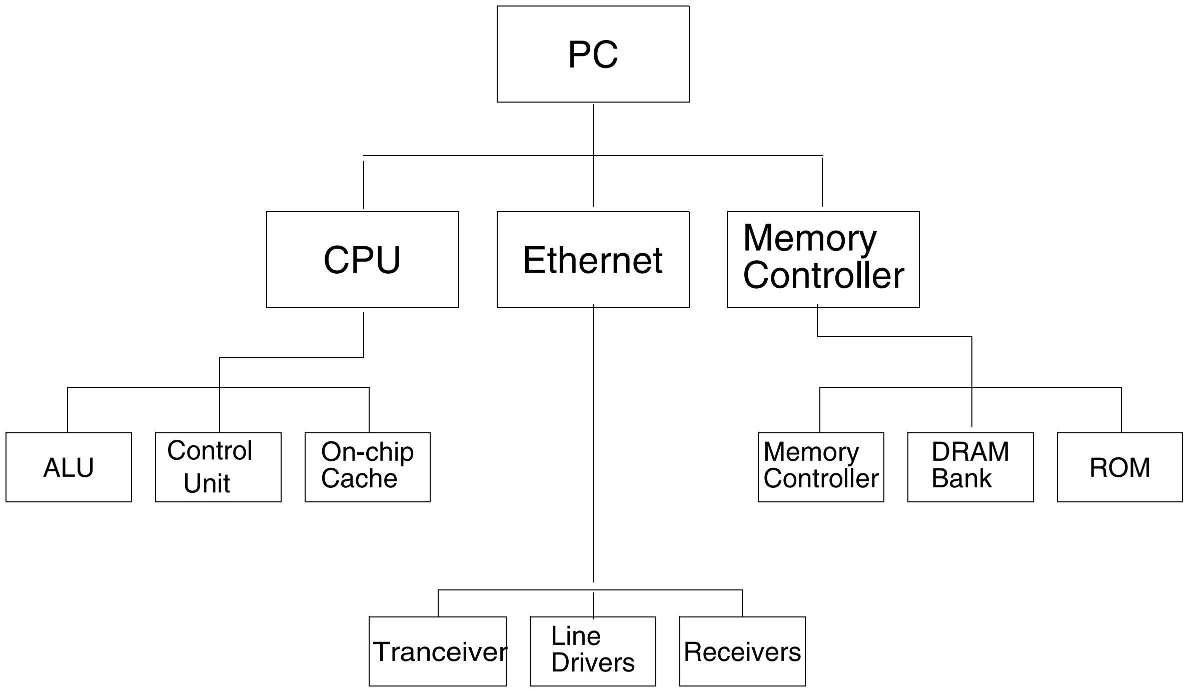
After you have updated Constraint Manager with the topology and constraint modifications, select the Relative Propagation Delay worksheet in the Net folder of the Electrical tab. The pin pair created in the *Rel Prop Delay* page of the Set Constraints dialog box in SigXplorer, is listed. Similarly, you can create multiple pin pairs by adding new rules in SigXplorer.

5

Constraints in Hierarchical Designs

Overview to Hierarchical Designs

A hierarchical design is a large and complex design divided into sub designs (hierarchical blocks). Each of the sub designs can be further divided into sub designs. For example, you may have a hierarchical design called PC that contains sub-designs CPU, Ethernet, and Memory Controller.



The hierarchical design method is typically followed for large and complex designs. These designs are divided into individual modules where each module represents a logic function.

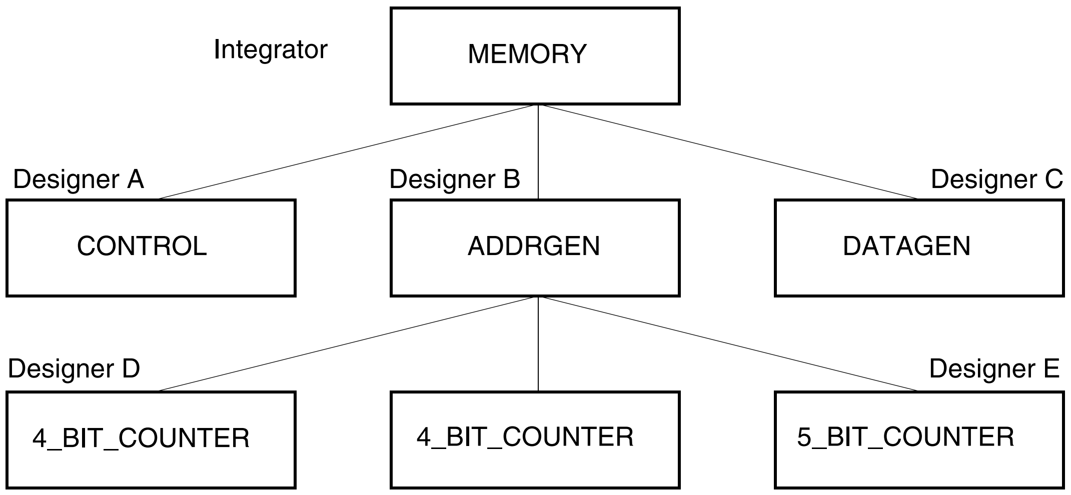
The chapter discusses the following:

- Team Design on page 98

- Assigning Constraints in Hierarchical Designs on page 99
- Constraints on Read-Only Blocks on page 100

Team Design

Teams of designers may be working on a hierarchical design consisting of various blocks. In this team design environment, each designer may be working on a block in the hierarchical design. After all the designers working on the blocks have completed their work, the Integrator (a Team Leader or a designated person) integrates all the blocks into the top-level design.



In the above example of a hierarchical design `MEMORY`, teams of designers work on different blocks of the design and the Integrator integrates all the blocks into the top-level design `MEMORY`.

If a block is instantiated more than once in a design, it is called a reuse or replicated block. For example, `4_BIT_COUNTER` is a replicated block because it is instantiated twice in the schematic for the `ADDRGEN` block.

Cadence recommends the following for managing constraints in a team design environment:

- The designer working on a block must not add electrical constraint properties on a replicated block in the Occurrence Edit mode in Design Entry HDL. This is because such occurrence properties will not be pushed to Constraint Manager. Cadence recommends that only after the Integrator integrates all the lower-level designs or blocks into the top-level design, constraint properties be added on replicated blocks in the Occurrence Edit mode in Design Entry HDL.

For example, the block `4_BIT_COUNTER` is instantiated twice in the schematic for the `ADDRGEN` block. Designer B must not add electrical constraint properties on the `4_BIT_COUNTER` block in the Occurrence Edit mode in Design Entry HDL. After the Integrator has integrated all the lower-level designs into the top-level design `MEMORY`, occurrence properties should be added on the replicated block `4_BIT_COUNTER` in the schematic for the block `ADDRGEN`.

- When the Integrator starts Constraint Manager from Design Entry HDL after integrating all the lower-level designs into the top-level design `MEMORY`, the electrical constraint properties added in the lower-level designs (including the constraint properties added by the Integrator in the Occurrence Edit mode on replicated blocks in the schematic for lower-level designs) are displayed in Constraint Manager.

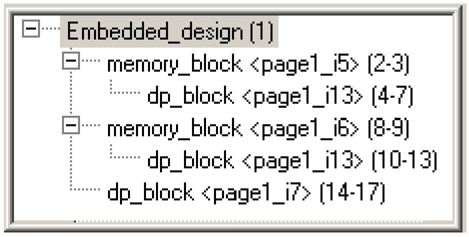
Assigning Constraints in Hierarchical Designs

You can adopt one of the following two methods to assign constraints in hierarchical designs:

1. Assign constraints in the context of the top-level design.

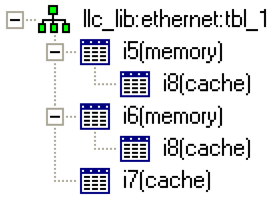
   You can adopt this method if you are not creating the hierarchical design in a team design environment, or if you do not want to create physically reusable blocks. In this method, you assign constraints for all the lower-level designs in the context of the top-level design.

   For example, if you have a hierarchical design similar to the one shown below, you should assign constraints to the lower-level design `memory_block` when the top-level design named `Embedded_design` is set as the root design.



```
Embedded_design (1)
    memory_block <page1_i5> (2-3)
        dp_block <page1_i13> (4-7)
    memory_block <page1_i6> (8-9)
        dp_block <page1_i13> (10-13)
    dp_block <page1_i7> (14-17)
```

Hierarchical Design in Design Entry HDL



```
llc_lib:ethernet:tbl_1
    i5(memory)
        i8(cache)
    i6(memory)
        i8(cache)
    i7(cache)
```

Hierarchical Design in SCM

2. Assign constraints in the individual blocks (sub-designs) in the hierarchical design.

   You can adopt this method if you are creating the hierarchical design in a team design environment, or if you want to create physically reusable blocks.

   In this method, you set the block in which you want to assign constraints as the root design for the project and then assign constraints in the block. When you add the block in a design, the constraints in the block appear as inherited constraints in the design. You can then choose to override the inherited constraints from the block or retain them in the design. The constraints inherited from lower-level designs are referred to as *lower-level constraints*.

For example, if a hierarchical design named `Embedded_design` with two blocks named `memory_block` and `dp_block` is being created in a team design environment, the designers working on the blocks assign constraints in each block when the block is set as the root design. When the owner of the top-level design named `Embedded_design` uses the blocks, the constraints in the blocks appear as inherited constraints in `Embedded_design`. The owner of `Embedded_design` can choose to override the constraints inherited from the blocks or retain them. Constraints can also be assigned on the lower-level design in the context of the top-level design, `Embedded_design`.

Caution

Constraints overridden in the top-level design (for example, `Embedded_design)` ***cannot be pushed down to lower-level designs. If you need to override constraints in all the occurrences of the lower-level design (***`memory_block` or `dp_block)`***, you should make the changes in the lower-level design.***

Note: For more information on how lower-level constraints appear in top-level design and how you can use them effectively, see Working with Constraints in Lower-Level Designs on page 100.

Constraints on Read-Only Blocks

If an electrical constraint or a placeholder for a constraint is present on a read-only block used in a hierarchical design and you modify or delete the constraint in Constraint Manager, errors will be displayed when you backannotate the schematic. This is because you do not have write permissions in the read-only block. To avoid this, add the `NO_BACKANNOTATE=ALL` property on the read-only block in the schematic.

Note: The `NO_BACKANNOTATE=ALL` property applies only to the block on which it is added and not to its child blocks.

Working with Constraints in Lower-Level Designs

Design Entry HDL-Constraint Manager flow includes support for handling constraints in schematic blocks in hierarchical designs. You can capture constraints in a schematic block and later pull the constraints into a top-level design by instantiating the block in the top-level design. These constraints are visible and can be edited in the context of the top-level design.

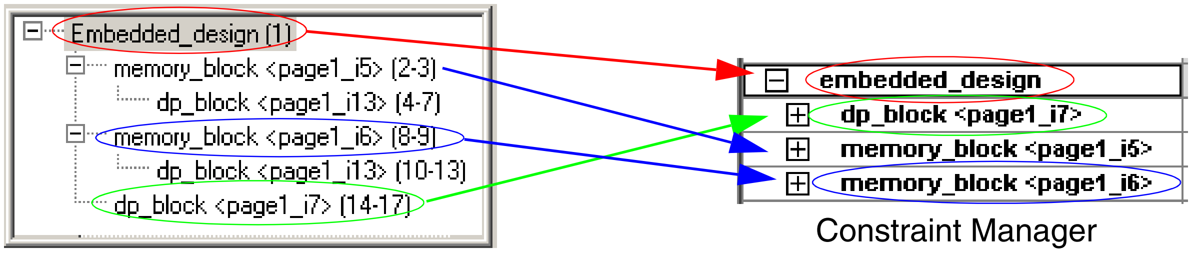
Constraint changes made in a top-level design will override constraints coming from the lower-level design in the context of the top-level design. You can work with lower-level designs to:

- view constraints as defined in the lower-level design in the context of the top-level design.
- make changes that can be reflected in all instances of the lower-level design.
- refresh a higher-level writable block with the current information from a lower-level design.

Note: Prior to release 15.5, data stored in a schematic block in a hierarchical design could not be merged into the top-level (parent) design in which the schematic block was instantiated. For example, constraints captured in the schematic block `dp_block` could not be propagated to the top-level design, `Embedded_Design` where `dp_block` is instantiated as a lower-level design.

Understanding How Hierarchical Designs Appear in Constraint Manager

Constraint Manager displays all lower-level design blocks in a hierarchical design, separately. Constraint Manager treats the root design as the top-level design and displays all the lower-level designs below it.

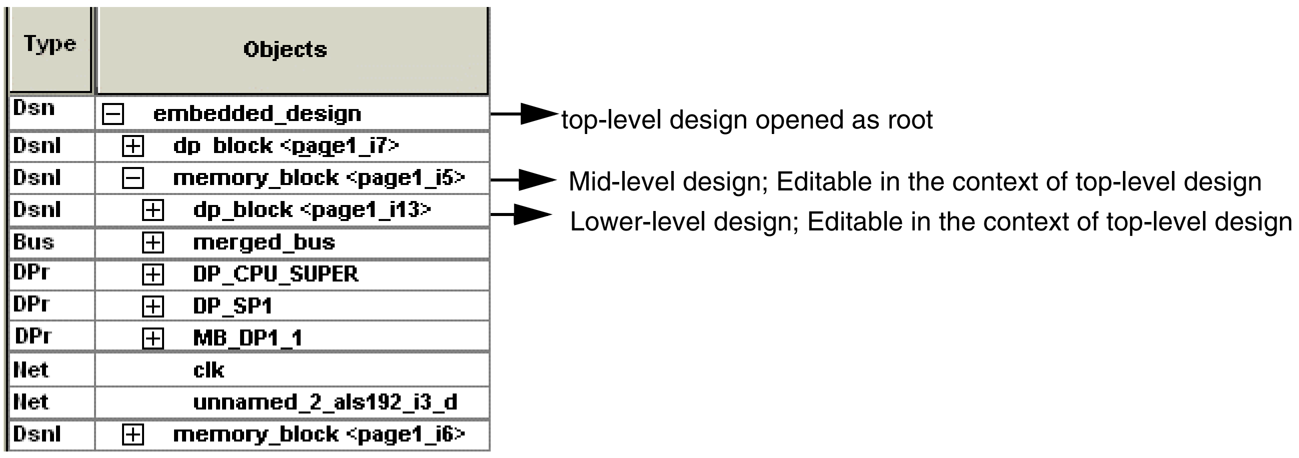

Embedded_design (1)
memory_block <page1_i5> (2-3)
dp_block <page1_i13> (4-7)
memory_block <page1_i6> (8-9)
dp_block <page1_i13> (10-13)
dp_block <page1_i7> (14-17)
embedded_design
dp_block <page1_i7>
memory_block <page1_i5>
memory_block <page1_i6>
Constraint Manager

Design Entry HDL

For instance in the above example, `Embedded_design` is the top-level design and it includes two instances of `memory_block` and one instance of `dp_block`. Constraint Manager displays the design in a hierarchical view.

Note: Before 15.5, Constraint Manager displayed all objects in a hierarchical design in a flat hierarchy.

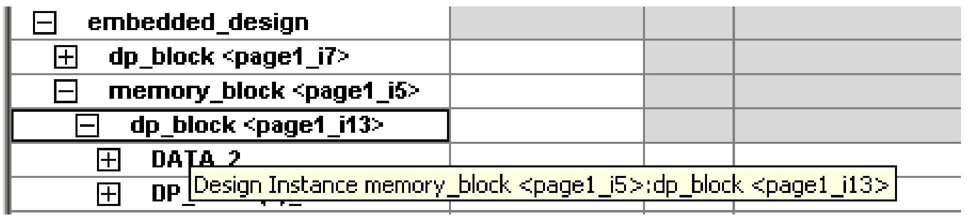
The root design is treated as an instance under `System`. For example, notice that while `Embedded_design` is the root design, it appears as an instance under `System`.


top-level design opened as root
Mid-level design; Editable in the context of top-level design
Lower-level design; Editable in the context of top-level design

Type	Objects
Dsn	⊟ embedded_design
Dsnl	⊞ dp_block <page1_i7>
Dsnl	⊟ memory_block <page1_i5>
Dsnl	⊞ dp_block <page1_i13>
Bus	⊞ merged_bus
DPr	⊞ DP_CPU_SUPER
DPr	⊞ DP_SP1
DPr	⊞ MB_DP1_1
Net	clk
Net	unnamed_2_als192_i3_d
Dsnl	⊞ memory_block <page1_i6>

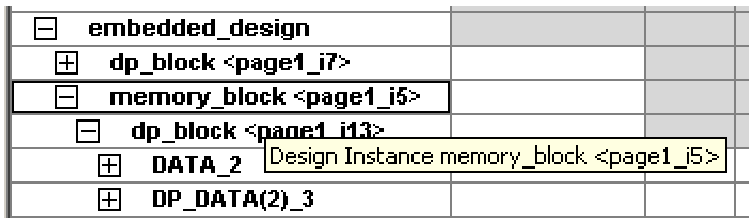
Each design has an associated tooltip and its path details are displayed in the tooltip as well as on the status bar. Lower-level designs included under the top-level design appear with the tooltip displaying “design”. Lower-level designs included within another lower-level design are treated as instances and the tooltip displays “design instance”. The name of the design and its path is also mentioned. Also, the Type column lists the type of the object, such as *Dsn* represents the root design, and *DsnI* represents an instance of a lower-level design.

If a design instance has more blocks, then the tooltip has the syntax `Design instance \<design name>\<Location>: <sub_design name>\<Location>`. For a design with instantiated lower-level designs, the lower-level design names appear before the net names or constraint names.



```
⊟ embedded_design
  ⊞ dp_block <page1_i7>
  ⊟ memory_block <page1_i5>
    ⊟ dp_block <page1_i13>
      ⊞ DATA_2
      ⊞ DP_
```

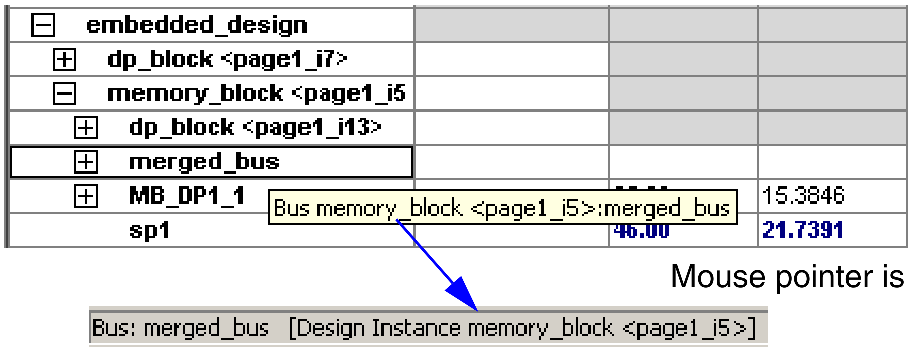
Design Instance memory_block <page1_i5>:dp_block <page1_i13>



```
⊟ embedded_design
  ⊞ dp_block <page1_i7>
  ⊟ memory_block <page1_i5>
    ⊟ dp_block <page1_i13>
      ⊞ DATA_2
      ⊞ DP_DATA(2)_3
```

Design Instance memory_block <page1_i5>

You can find the source information for an object by placing the mouse pointer on the object. For example, the following figure shows the message displayed on the status bar when the mouse pointer is placed on a bus, `merged_bus`.



Bus memory_block <page1_i5>:merged_bus

Bus: merged_bus [Design Instance memory_block <page1_i5>]

Mouse pointer is displayed on `merged_bus`

and the status bar displays object's details.

Note: Prior to release 15.5, you could only view hierarchical designs as flat designs in Constraint Manager.

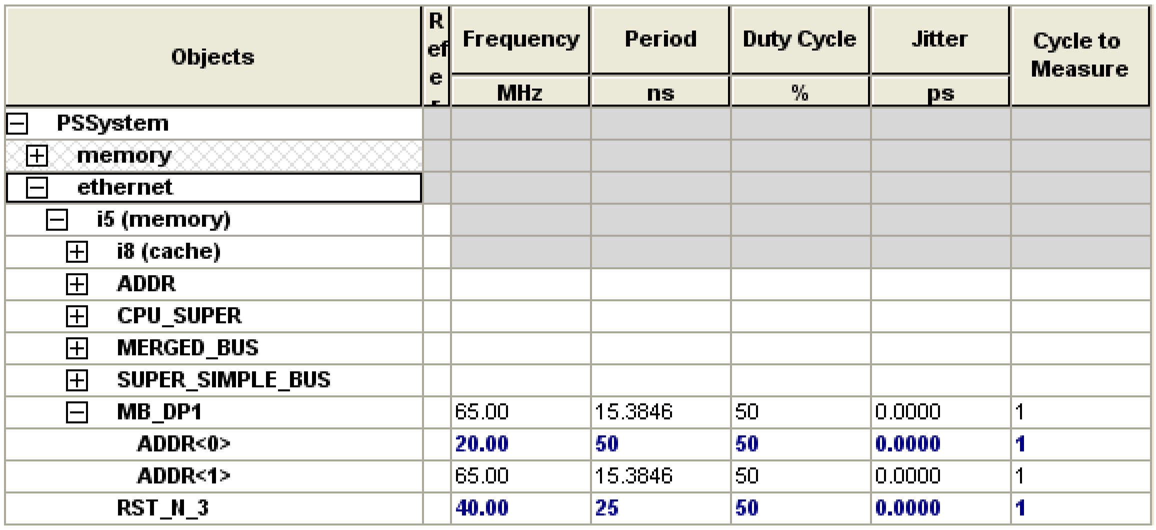
Inherited and Overridden Constraints

- Inherited constraint —It is a property value which is inherited from a parent object. Lower-level constraints are called inherited constraints. By default, a cell which is populated and colored regular black reflects that the value is inherited from a parent object.
- Overridden constraint — Constraints that are modified in the context of the top-level design are called overridden constraints. As such overridden constraints belong to the design specified as the root design. By default, a worksheet cell is colored bold blue if the value it contains is overridden.

You can identify whether a constraint is inherited or overridden by its color. Within a design, constraints may be displayed in blue color or gray color.

- Blue color—Indicates a constraint added on an object in the top-level design or an inherited constraint that is overridden in the top-level design.
- Gray color—Indicates a constraint inherited from a lower-level block or from an ECSet. You can override the inherited constraints in the top-level design.

For example, in the following figure the lower-level block named `memory` has a differential pair named `MB_DP1` with some constraints added on it. When you open the project with `ethernet` as the root design, the constraints on the differential pair named `MB_DP1` in memory block appear in gray color because the constraints are inherited by the ethernet design.



Objects	Refer	Frequency	Period	Duty Cycle	Jitter	Cycle to Measure
		MHz	ns	%	ps	
⊟ PSSystem						
⊞ memory						
⊟ ethernet						
⊟ i5 (memory)						
⊞ i8 (cache)						
⊞ ADDR						
⊞ CPU_SUPER						
⊞ MERGED_BUS						
⊞ SUPER_SIMPLE_BUS						
⊟ MB_DP1		65.00	15.3846	50	0.0000	1
ADDR<0>		**20.00**	**50**	**50**	**0.0000**	**1**
ADDR<1>		65.00	15.3846	50	0.0000	1
RST_N_3		**40.00**	**25**	**50**	**0.0000**	**1**

If you override an inherited constraint in the ethernet design, the overridden constraint appears in blue color. For example the constraint on the net ADDR<0> appears in blue color because it is overridden in the ethernet design. Note that the constraints on the RST_N_3 net appear in blue color because the net belongs to the ethernet design.

Note: If you create an object, such as a differential pair, match group or ECSet, the constraints assigned to the object are inherited by its member nets. If you change the constraints on the member nets, Constraint Manager treats the change as constraint overrides and displays the overrides in blue color.

Rules For Merging Lower-Level Constraints

General Rules

1. If you delete a pin pair, differential pair or match group in a lower-level block (by setting the lower-level block as the root design), the pin pair, differential pair or match group gets deleted in context of the top-level design even if you have overridden the constraints on the pin pair, differential pair or match group in context of the top-level design.

   For example, assume that you have a pin pair in a lower-level block named memory. Override the constraints on the pin pair in a higher level design named ethernet (that is set as the root design). If you now set the memory block as the root design, delete the pin pair and then set the ethernet design as the root design, the pin pair is no longer displayed in context of the `ethernet` design, even though the constraints on the pin pair were overridden in context of the `ethernet` design.

2. When you apply ECSets in the design by running the *File – Update Constraint Manager* command in SigXplorer or the *File – Import – Electrical CSets command in Constraint Manager*, if the ECSet apply results in addition or deletion or terminations (coming from the topology) the addition or deletion of terminations will be done only for the nets in the root design and not for the nets in the lower-level blocks.

Local Nets, Xnets and Buses

Local nets, Xnets, and buses belong to individual designs. To assign constraints to these objects, you need to open the design as the root design.

1. Constraints assigned to local nets, Xnets, and buses in a lower-level design are visible at higher levels. If you change the constraint value for a local net, Xnet or bus in the lower-level design, the constraint changes are visible at the top-level design also.
2. Constraints added in lower-level designs appear as inherited values in the top-level design. You can override these inherited values in the top-level design. If you override an inherited constraint in the top-level design and then change the same constraint in the lower-level design, the overridden constraint value in the top-level design is not impacted. The constraints defined in the top-level design wins.
3. You can set different overrides for different instances of a reused block at different levels of hierarchy.

   For example, assume that a block named `memory_block` has the `MAX_OVERSHOOT` constraint with the value `5100:-610`. If you add two instances of this block in a design named `Embedded_design`, the `MAX_OVERSHOOT` constraint in `memory_block` appears as an inherited constraint in the context of the top-level design, `Embedded_design`. You can override the constraint on each instance of the block by assigning different values. For example, you can override the constraint on the first instance of `memory_`block by specifying the value `5000:-600` and override the constraint on the second instance of the block by specifying the value as `5300:-630`.
4. A constraint coming from a lower-level design when deleted at the top-level design makes the lower-level value stale. Such constraints are not pushed to the top-level design even when changed at the lower-level design. It is recommended that you take caution while deleting or changing values of lower-level constraints in the top-level design as the changes in lower-level constraints are not automatically rolled back to the top-level design. To manually roll back changes, use one of the following two methods:

   a. Manually edit the value to match the value in the lower-level design. When the values are in-sync, any future edits done in the lower-level design will ripple up to the top-level.

   b. Delete and add the block instances again.

5. Changes made to an object in a higher level design are ignored if the object is deleted from lower-level designs. For example, if you create a pin-pair in a lower-level design and assign constraints to it, then the pin-pair definition and constraint values ripple up the hierarchy. You can override the constraint values in higher level design. If you now delete the pin-pair from the lower-level design, then the same is also deleted from the higher level design.

Interface Nets

1. Constraints assigned on interface nets are merged with the base net and appear in the context of the top-level design. If the interface nets have the same constraint with different values, any one of the constraint values will be used.
2. If an interface net in a lower-level design is connected to an Xnet in a higher-level design, the interface net becomes a member of the Xnet.

Global Nets and Buses

1. Constraints assigned to global signals in lower-level designs are rippled up and displayed in the top-level design.

Differential Pairs

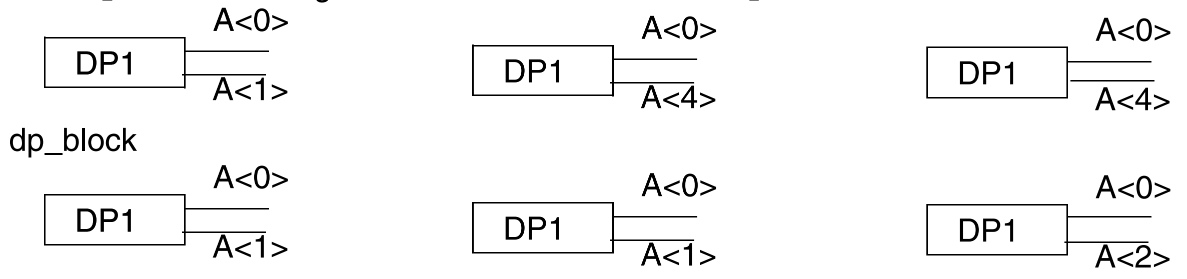
1. Constraints assigned to differential pairs in lower-level designs appear as inherited values in the top-level design.
2. If there are conflicts in differential pairs because the same differential pair name exists across multiple instances of a design imported in a top-level design, then the differential pair names are renamed automatically. The differential pair in the first merging design retains its name while the second differential pair is renamed using a numerical suffix.
3. If you rename a differential pair at any level of a hierarchical design, the new name is displayed in the top-level design.

   For example, assume that the block `dp_block` has a differential pair named `CAC_DP1`. If you set `memory_block` as the root design and rename the differential pair to `CACHE_DP1`, and then set `Embedded_design` as the root design, the differential pair is displayed as `CACHE_DP1` in the design. If you now set `dp_block` as the root design and rename the differential pair to `CAC_DP1`, and then set `Embedded_design` as the root design, the differential pair is displayed as `CAC_DP1`.

Note: Renaming a differential pair does not affect inheritance of constraints from lower-level designs.

4. If you make any changes to constraint values in a differential pair in a lower-level design, even after renaming the differential pair in the top-level design, the changes to constraint values in the lower-level design are propagated to the top-level design.
5. If you have an interface net differential pair which has nets connected from a lower-level design to a top-level design, the differential pair appears in the lower-level design and it shows nets that are available at the top-level design. For example, assume you have an interface net differential pair `DP1`, which contains nets `sp1` and `sp2`, in the design `DP_block`. Assume that these nets are connected in the top-level design `memory_block` to nets `p1` and `p2`. In Constraint Manager, a differential pair `DP1` containing the nets `p1` and `p2` will appear in the (lower-level) design `DP_block`.
6. If you have an inherited interface net differential pair and you rename it in a top-level design, and then rename the differential pair in the lower-level design, the renamed lower-level differential pair is not rippled up to the top-level design.
7. If you change the membership of a differential pair object in the context of the top-level design, and then set the block in which the differential pair exists as the root design and change the membership of the differential pair again, the membership changes you made in the context of the top-level design wins. To understand this, see the example below:

`memory_block` design: It contains an instance of `dp_block`



Step 1
`dp_block` is root
Differential pair ripples to `memory_block`

Step 2
`memory_block` is root
Member net changed in `memory_block`

Step 3
`dp_block` is root
Member net is changed
However, nets in `memory_block` win

Match Groups

1. Constraints assigned to match groups in lower-level designs ripple up in hierarchy and appear as inherited values in the top-level design.
2. A match group assigned in a lower-level design is reflected in the top-level design with the same target net relationship. However, you can change the target net in match groups inherited from lower-level designs in the context of the top-level design. This will assign an overridden value to the original target net.

   For example, assume that you have a match group `MG_RST_BUS` in `memory_block` with `rst_bus<0>`, `rst_bus<1>`, and `rst_bus<3>` as member nets and `rst_bus<0>` as the target net. When you make `embedded_design` as the root design, the match groups in the two instances of `memory_block` appear as `MG_RST_BUS` and `MG_RST_BUS_1`. You can set a different target net in these match groups. For instance, you can make `rst_bus<2>` as the target net in `MG_RST_BUS` match group.
3. If there are conflicts in match groups in case the same match group name exists across multiple instances of a design instantiated in a top-level design, the match groups are renamed automatically. The match group in the first merging design retains its name while the second match group is renamed using a numerical suffix.

   Note: ECSet-generated match groups in a lower-level design are not renamed in the top-level design. as such match groups are considered as global objects. See Special Cases Where Rules for Constraint Merging Are Overridden on page 110 for more details.
4. If you rename a match group in a design, the renamed match group ripples up to higher-level designs.

   For example, assume that the block `dp_block` has a match group named `MG_RST_BUS`. If you set `memory_block` as the root design and rename the match group to `MG_RST`, and then set `Embedded_design` as the root design, the match group is displayed as `MG_RST` in the design. If you now set `dp_ block` as the root design and rename the match group to `MG_RST_LOW`, and then set `Embedded_design` as the root design, the match group is displayed as `MG_RST_LOW` in `Embedded_design`.
5. A match group in a lower-level design is displayed in the context of the top-level design. If you now add a net existing in the top-level design as a member of the match group, the net appears as a member of the match group in the lower-level design. For example, assume that a net `clk` in the top-level design named `Embedded_design` is added as a member of a match group named `MG_RST_BUS` in `memory_block`. The `MG_RST_BUS` shows `clk` as its member in lower-level design (`memory_block`) when `Embedded_design` is set as the root design. The net `clk` is no longer displayed under the root design `Embedded_design`.
6. If you rename a match group rippled up from a lower-level design in a higher-level design and if you again rename the match group in the lower-level design, the renamed lower-

level match group ripples up and overrides the match group name changes done at the top-level.

For example, assume that you have a 3-level design with `embedded_design` at the highest level, `memory_block` instantiated in `embedded_design` and `dp_block` instantiated in `memory_block`. You have a match group named `MG1_DP` in `dp_block`. This match group ripples up and appears in `memory_block` when it is made the root design. If you now rename this match group to `MG1_MEM` and make the `embedded_design` design as root, the match group will ripple up. If you make `dp_block` as the root design and change the match group name from `MG1_DP` to `MG1_DP1`, then the renamed match group name again ripples up the hierarchy. The name changes in mid- and top- levels are ignored and the last changed name wins as it ripples up in hierarchy.

7. If you add a new member net from the top-level design in a match group, which is part of a lower-level design, the new net appears in the match group under the lower-level design. For example if a net named `clk` is part of `embedded_design`, which is the top-level design and is added in the match group, `MG_RST_BUS`. `MG_RST_BUS` shows the net `clk` as a member and continues to appear under the lower-level design (`memory_block`).

Note: In pre-15.5 releases, if there were multiple replicated blocks at one level of hierarchy, you could select nets from different blocks to create differential pairs and match groups. Post release 15.5, you can create differential pairs and match groups for nets from different blocks only in the top-level design. For example, assume that you have a top-level design named `TOP` that contains two instances of design `MID` – `MID1` and `MID2`. Assume that `MID1` contains `net1` and `MID2` contains `net2` and both these nets are part of a differential pair `DP1`. In the previous releases, you could create such differential pairs in `MID` as `TOP` was treated as a flat design for the purpose of constraint-assignment. However, to create such differential pairs now, you would need to make `TOP` as the root design and then create differential pairs. You cannot create such differential pairs across lower-level design `MID`.

ECSet

1. If you make a definition change to an ECSet, it is automatically propagated up in the hierarchy.
2. If you reference an ECSet on an interface net, the constraints in the ECSet are re-applied to the aliased base net in the top-level design. In fact, all ECSet references on all objects can be re-applied at higher-levels.
3. If ECSets with the same name across different blocks have the same topology, the constraints on the ECSets that have the same topology are merged in the context of the top-level design. However, if such ECSets have a different topology, Constraint Manager

retains the name for one of the ECSets. For the other ECSets, it adds a suffix of `_n` where `n` is a number.

For example, if you add `memory_block` that has an ECSet named `ECSet1` with an associated topology in the root design named `Embedded_design`, the ECSet appears as `ECSet1` in the context of `embedded_design`. If you now add an instance of a block named `dp_block` that has an ECSet named `ECSet1` but with a different associated topology, the ECSet on block `dp_block` appears as `ECSet1_1` in the context of `embedded_design`.

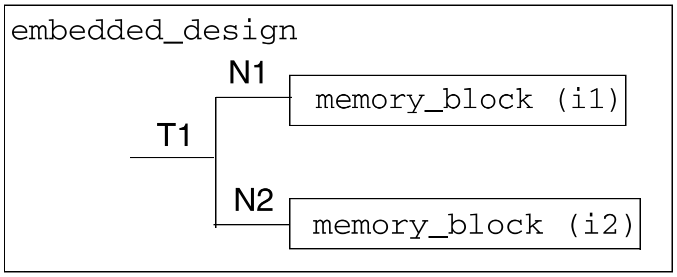
4. If you want ECSets with the same names to be treated differently, rename them to have a unique name in each block. For example, you can append all ECSets from lower-level designs with a unique suffix.

Special Cases Where Rules for Constraint Merging Are Overridden

The following section describes certain situations where Constraint Manager may make a variation in processing the rules described in Rules For Merging Lower-Level Constraints on page 104.

1. If two or more nets having the same constraint with different values are aliased, the constraint on the base net wins. For example, if net `p1` having the `MAX_OVERSHOOT` constraint value of `5100:-610` is aliased to a net `sp1` which has the `MAX_OVERSHOOT` constraint value of `4900:-610`, the `MAX_OVERSHOOT` constraint with the value `5100:-610` will be applied on both the nets because `p1` is the base net.
2. If two or more nets are aliased and if the base net does not have the same constraint that exists on the aliased nets, the constraint on the aliased nets will be merged on the base net. If the aliased nets have the same constraint with different values, the constraint value on one of the aliased nets (on a random basis) will be merged on the base net.
3. Constraint changes made to a pin-pair in a higher-level design are ignored if the pin-pair is deleted from lower-level designs. For example, if you create a pin-pair in a lower-level design and assign constraints to it, the pin-pair and its constraint values ripple up the hierarchy. You can override the constraint values in the higher level design. If you now delete the pin-pair from the lower-level design, then the pin-pair is also deleted from the higher level designs.
4. The `SIGNAL_MODEL` property on components is not read from the OPF of the lower-level design. Therefore, you should always annotate this property on the schematic.
5. If you have a design containing interface nets instantiated multiple times in a top-level design, the top-level design may have a base net that is connected with multiple interface nets. For example, assume that you have a design named `embedded_design`, with two

instances of `memory_block`. Each instance of `memory_block` has interface nets, `NI` and `N2` connected to the net `T1` of `embedded_design`.



Constraint Manager checks whether the top-level net, `T1`, has any constraints defined. If `T1` has constraints defined on it, these constraint values win. If `T1` does not have any constraints defined on it, constraint values assigned to `NI` or `N2` win based on which design is merged first in the top-level design.

6. If you have ECSets with the same names but different constraint definitions across two levels of hierarchy, constraint differences between two ECSets may be ignored while merging at the top-level. For instance, assume that you have a design, `dp_block`, instantiated in `memory_block`. Both `dp_block` and `memory_block` designs have an ECSet named `ECSet1` with different constraint definitions. When `memory_block` is launched as the root design, Constraint Manager checks if the topology file associated with the ECSet is different. Here constraint differences between `dp_block:ECSet1` and `memory_block:ECSet1` may be ignored when the first block is added.

7. If you create a match group or differential pair in the context of the root design, the match group or differential pair is displayed under the root design and not under the lower-level design in which it was added in the context of the root design.

8. A match group or differential pair existing in a lower-level design will continue to be displayed under the lower-level design even if the interface nets that are members of the match group or differential pair are aliased to base nets in the top-level design.

9. If an ECSet exists in a lower-level design, it appears in the context of the top-level design. However, if you set the lower-level design as the root design and delete the ECSet, the ECSet will continue to appear in the context of the top-level design.

   For example, assume that you have an ECSet in the lower-level design `memory_block` added in `embedded_design`. The ECSet is displayed in the context of `embedded_design` (when set as the root design). If you now set `memory_block` as the root design, delete the ECSet, and then set `embedded_design` as the root design, the ECSet continues to be displayed in the context of `embedded_design`.

10. If two or more blocks have ECSets with the same name and if the ECSets do not have an associated topology, the constraints on the ECSets are merged in the top-level design.

11. In pre-15.5 releases, you could backannotate constraints added in lower-level designs in a top-level design by making the top-level design as the root design and running the *Tools – Constraints – Update Schematic* command in Design Entry HDL. Release 15.5 onwards, you cannot backannotate lower-level constraints in the top-level design.

Recommendations for Optimum Handling of Lower-Level Constraints

1. It is recommended that you use ECSets to capture constraints in lower-level blocks.
2. Exercise caution while deleting or changing values of lower-level constraints at top-level design as you cannot rollback the changes in lower-level constraints back to the top-level design.
3. If you have set the top-level design as root, you can override the values of constraints in lower-level designs, but cannot change values of constraints in lower-level designs directly. To make any changes to a lower-level design, make it the root design.
4. It is recommended that you make changes to constraints on interface nets only at the top-level design.
5. Always create ECSets using interface or global objects in the top-level design and not in the lower-level design.

   This is because, if you create an ECSet that has one driver and two receivers in a lower-level design using interface or global nets and then connect the interface or global net to a signal in the top-level design, the ECSet will become invalid in the context of the top-level design because it now has three receivers.
6. If you want to apply the same ECSets across different blocks, it is recommended that you apply them at the top-level design.
7. Add signal models to all the nets, which you want to connect to interface ports.
8. Always clean up obsolete objects (signals and buses) in a lower-level design before merging it in a higher level design.

Handling Constraints in Hierarchical Designs

This section contains the following topics:

- Overriding Constraints in a Hierarchical Design
- Handling Differential Pairs in Hierarchical Designs

Overriding Constraints in a Hierarchical Design

In a hierarchical design, you cannot add constraints in the *Occurrence Edit* mode, which means that you cannot override the constraints on lower-level designs in the schematic. You can only override constraints on a lower-level net in Constraint Manager. The overrides performed on a lower-level design in Constraint Manager cannot be backannotated. However, you can see the winning value for the lower-level block if you descend into the block in the *Expanded* or *Occurrence Edit* mode.

Handling Differential Pairs in Hierarchical Designs

In hierarchical designs, the `DIFFERENTIAL_PAIR` property on a lower-level design is updated in the context of the top-level design. The constraint is not backannotated to the lower-level design on the schematic and remains in Constraint Manager.

Note: You cannot add the `DIFFERENTIAL_PAIR` property in the *Occurrence Edit* mode, which means that you cannot add the property on lower-level nets in the schematic.

Migration requirement

To ensure that the lower-level constraints of a design created in previous releases are read into top-level designs in the current release, save the old design in the current release.

6

Restoring Constraints from Definition

Overview

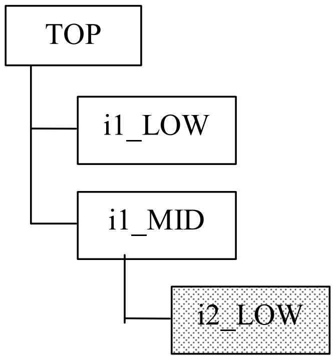
The *restore from definition* functionality enables you to restore constraint values in a hierarchical design from a lower-level (schematic) block. This functionality is helpful when you override constraint values in the context of a top-level design and want to revert to the original value stored in the lower-level block.

The restore from definition functionality restores values for objects from the immediate child block. Therefore, the object whose value is to be restored must be under the active design.

Note: Restore from definition is not supported for Component and Pin properties.

Example

Consider the example of a hierarchical design with TOP, MID, and LOW blocks, each representing its place value in the hierarchy.



In this example, restoring *NET1* in *i2_LOW* restores the value from the design *MID*, which represents the next level of hierarchy. However, if the design *TOP* is active, you cannot restore *i2_LOW* with constraint information directly from *LOW*. If design *TOP* is active, restoring *i2_LOW* would restore the values from design *MID*. To restore *i2_LOW* from its

LOW definition overridden in *MID*, you must open *MID* as the root design and restore the instance of *LOW* in *MID*.

This section covers:

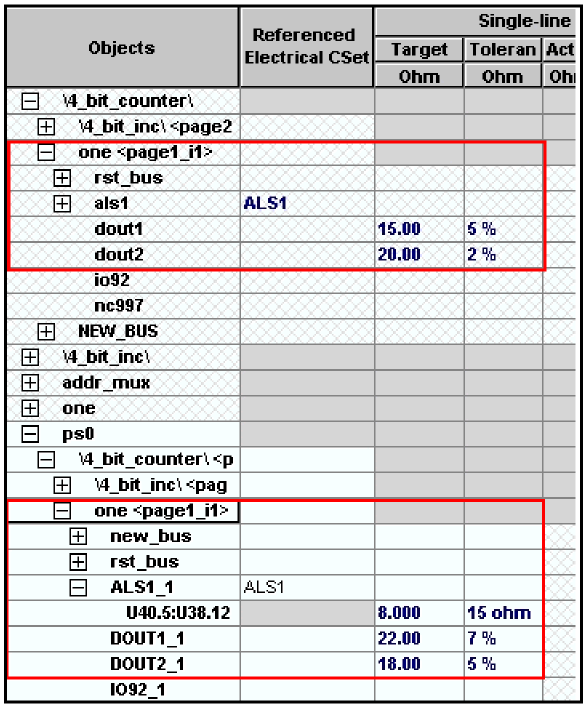
- Restoring a Constraint from its Definition
- Use Models
- Restoring Objects from their Definitions
- Other Cases

Restoring a Constraint from its Definition

Note: All the examples in this section are taken from a design named *ps0*, which includes, among other components and blocks, two instances of a schematic block, *one*.

To restore constraints on a net from its original definition, perform the following steps:

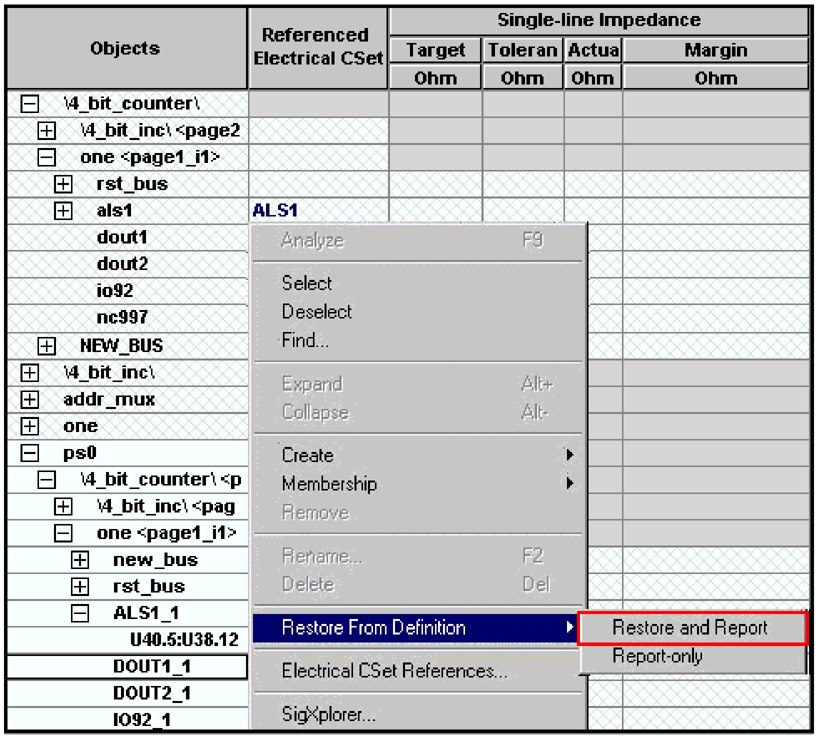
1. Select the appropriate workbook under the Net worksheet in Constraint Manager.



Objects	Referenced Electrical CSet	Single-line		
		Target	Toleran	Act
		Ohm	Ohm	Oh
⊟ \4_bit_counter\				
⊞ \4_bit_inc\ <page2				
⊟ one <page1_i1>				
⊞ rst_bus				
⊞ als1	ALS1			
dout1		15.00	5 %	
dout2		20.00	2 %	
io92				
nc997				
⊞ NEW_BUS				
⊞ \4_bit_inc\				
⊞ addr_mux				
⊞ one				
⊟ ps0				
⊟ \4_bit_counter\ <p				
⊞ \4_bit_inc\ <pag				
⊟ one <page1_i1>				
⊞ new_bus				
⊞ rst_bus				
⊟ ALS1_1	ALS1			
U40.5:U38.12		8.000	15 ohm	
DOUT1_1		22.00	7 %	
DOUT2_1		18.00	5 %	
IO92_1				

Figure 6-1 Lower-level constraints from the block *one* are overridden in the context of the top-level design *ps0*.

2. Right-click on the object or the constraint whose value you want to restore from the child block. In the example shown above, we will restore the constraints on the net *DOUT1* in the block *\4_BIT_COUNTER\ONE* under the design *ps0*.



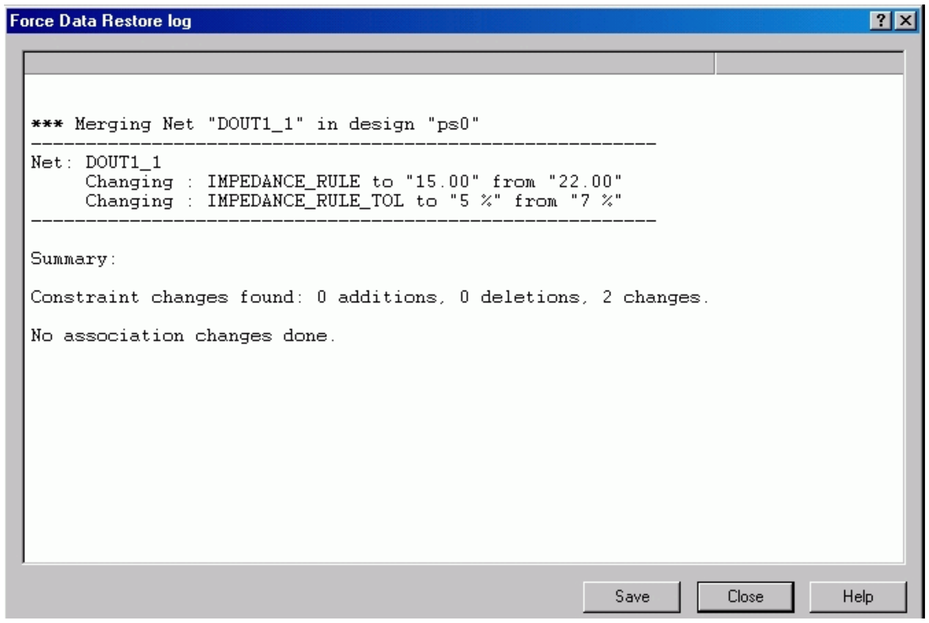
Objects	Referenced Electrical CSet	Single-line Impedance			
		Target	Toleran	Actua	Margin
		Ohm	Ohm	Ohm	Ohm
\4_bit_counter\					
\4_bit_inc\ <page2					
one <page1_i1>					
rst_bus					
als1	ALS1				
dout1					
dout2					
io92					
nc997					
NEW_BUS					
\4_bit_inc\					
addr_mux					
one					
ps0					
\4_bit_counter\ <p					
\4_bit_inc\ <pag					
one <page1_i1>					
new_bus					
rst_bus					
ALS1_1					
U40.5:U38.12					
DOUT1_1					
DOUT2_1					
IO92_1					

Figure 6-2 The Restore From Definition option

3. Choose *Restore From Definition > Restore and Report*.

   Note: If you want to view a report of the differences between the original definition and the overridden values without restoring the values, you can choose *Restore From Definition > Report only*.

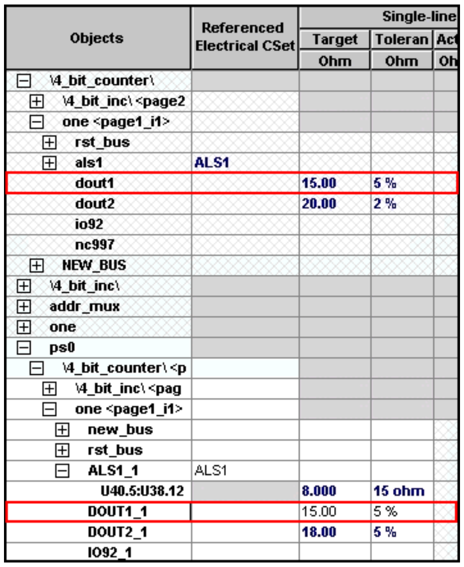
A notification is displayed summarizing the changes that have taken place on restoring the values.



```
*** Merging Net "DOUT1_1" in design "ps0"
----------------------------------------------------------
Net: DOUT1_1
     Changing : IMPEDANCE_RULE to "15.00" from "22.00"
     Changing : IMPEDANCE_RULE_TOL to "5 %" from "7 %"
----------------------------------------------------------

Summary:

Constraint changes found: 0 additions, 0 deletions, 2 changes.

No association changes done.
```

Figure 6-3 The Force Data Restore log window

The constraints on the selected net, *DOUT1*, are restored from the lower-level (child) block, *one*.



Objects	Referenced Electrical CSet	Single-line		
		Target	Toleran	Act
		Ohm	Ohm	Oh
⊟ \4_bit_counter\				
⊞ \4_bit_inc\ <page2				
⊟ one <page1_i1>				
⊞ rst_bus				
⊞ als1	ALS1			
dout1		15.00	5 %	
dout2		20.00	2 %	
io92				
nc997				
⊞ NEW_BUS				
⊞ \4_bit_inc\				
⊞ addr_mux				
⊞ one				
⊟ ps0				
⊟ \4_bit_counter\ <p				
⊞ \4_bit_inc\ <pag				
⊟ one <page1_i1>				
⊞ new_bus				
⊞ rst_bus				
⊟ ALS1_1	ALS1			
U40.5:U38.12		8.000	15 ohm	
DOUT1_1		15.00	5 %	
DOUT2_1		18.00	5 %	
IO92_1				

Figure 6-4 Constraint values are restored from the child block, *one*.

Use Models

Restore All the Constraints on an Object

As shown in the example, performing the restore from definition operation on an object restores all the constraints set on that object.

Restore a Hierarchical Block

Performing the restore from definition operation on a hierarchical block restores all the net level constraints in the block and properties from the child block.

Restore a Specific Constraint in Constraint Manager

You can also restore a specific constraint from the child block.

Interface nets, global buses, or Xnets cannot be restored. However, pin pairs for these objects may be restored.

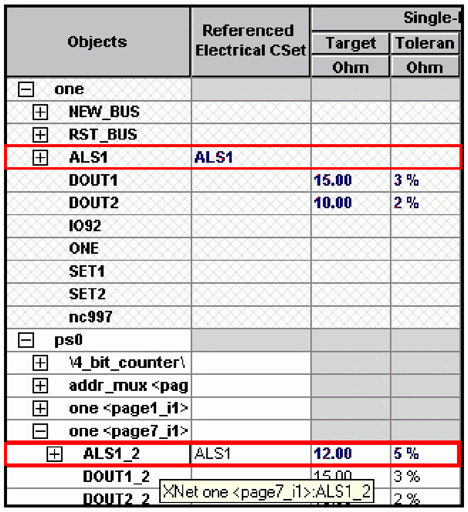
Note: Restoring constraints may cause violation in the Min:Max Propagation delay limits. For example, if the Min:Max Propagation delay limits on a lower-level block is set to 10:20 and in the root-design the constraint definition is overridden, such that Min:Max Propagation delay limits for the nets is set to 3:4. If you now restore the minimum propagation delay from the definition, the Min:Max limits will be modified to an unrealistic figure of 10:4.

Restoring Objects from their Definitions

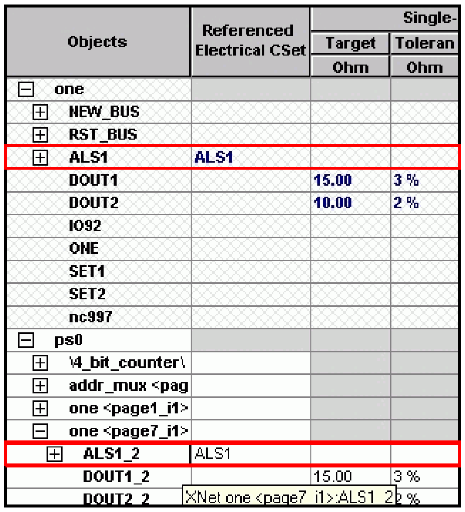
This section shows the effect of performing the restore from definition operation on various objects in Constraint Manager. Xnets/nets, differential pairs, bus-level and user-defined constraints, and pin pair and non-pin pair constraints are restored from their child blocks on restoring from the definition.

- Restoring Xnet Constraints
- Restoring Differential Pair Constraints
- Restoring Constraints in a Hierarchical Block
- Restoring Constraints in a Matched Group
- Restoring Bus-Level Constraints

Restoring Xnet Constraints



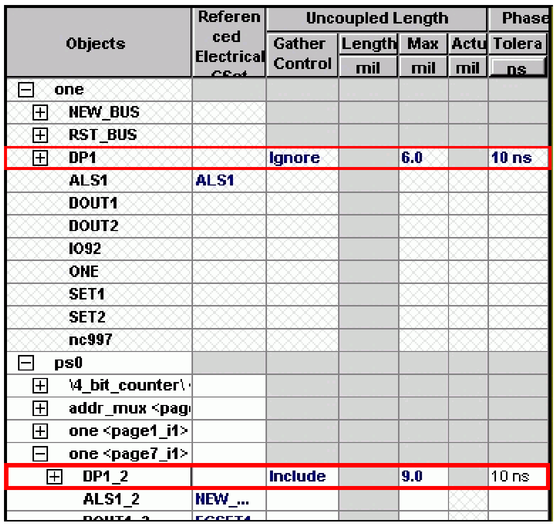
Objects	Referenced Electrical CSet	Single- Target Ohm	Toleran Ohm
⊟ one			
⊞ NEW_BUS			
⊞ RST_BUS			
⊞ ALS1	ALS1		
DOUT1		15.00	3 %
DOUT2		10.00	2 %
IO92			
ONE			
SET1			
SET2			
nc997			
⊟ ps0			
⊞ \4_bit_counter\			
⊞ addr_mux <pag			
⊞ one <page1_i1>			
⊟ one <page7_i1>			
⊞ ALS1_2	ALS1	12.00	5 %
DOUT1_2		15.00	3 %
DOUT2_2	XNet one <page7_i1>:ALS1_2		2 %



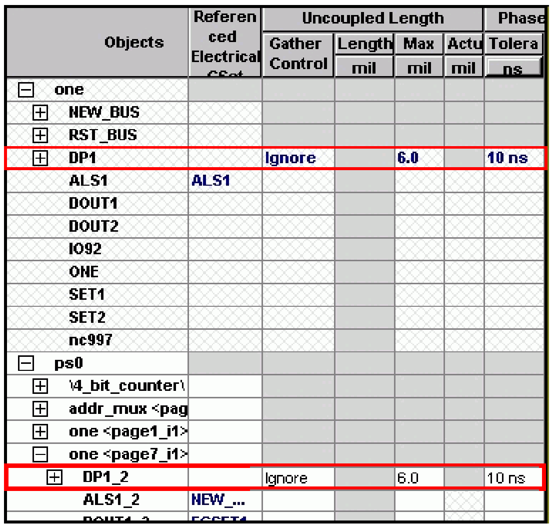
Objects	Referenced Electrical CSet	Single- Target Ohm	Toleran Ohm
⊟ one			
⊞ NEW_BUS			
⊞ RST_BUS			
⊞ ALS1	ALS1		
DOUT1		15.00	3 %
DOUT2		10.00	2 %
IO92			
ONE			
SET1			
SET2			
nc997			
⊟ ps0			
⊞ \4_bit_counter\			
⊞ addr_mux <pag			
⊞ one <page1_i1>			
⊟ one <page7_i1>			
⊞ ALS1_2	ALS1		
DOUT1_2		15.00	3 %
DOUT2_2	XNet one <page7_i1>:ALS1_2		2 %

Figure 6-5 Xnet ALS1_2 before and after restoring the constraints from the child block *one*

Restoring Differential Pair Constraints



Objects	Referen ced Electrical CSet	Uncoupled Length Gather Control	Length mil	Max mil	Actu mil	Phase Tolera ns
⊟ one						
⊞ NEW_BUS						
⊞ RST_BUS						
⊞ DP1		Ignore		6.0		10 ns
ALS1	ALS1					
DOUT1						
DOUT2						
IO92						
ONE						
SET1						
SET2						
nc997						
⊟ ps0						
⊞ \4_bit_counter\						
⊞ addr_mux <pag						
⊞ one <page1_i1>						
⊟ one <page7_i1>						
⊞ DP1_2		Include		9.0		10 ns
ALS1_2	NEW_...					
DOUT1_2	ECSET1					

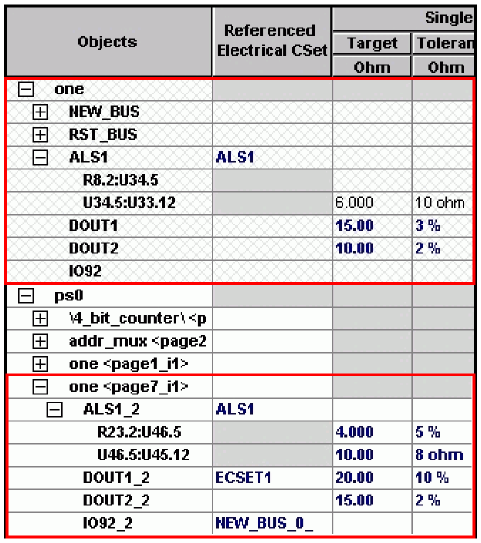


Objects	Referen ced Electrical CSet	Uncoupled Length Gather Control	Length mil	Max mil	Actu mil	Phase Tolera ns
⊟ one						
⊞ NEW_BUS						
⊞ RST_BUS						
⊞ DP1		Ignore		6.0		10 ns
ALS1	ALS1					
DOUT1						
DOUT2						
IO92						
ONE						
SET1						
SET2						
nc997						
⊟ ps0						
⊞ \4_bit_counter\						
⊞ addr_mux <pag						
⊞ one <page1_i1>						
⊟ one <page7_i1>						
⊞ DP1_2		Ignore		6.0		10 ns
ALS1_2	NEW_...					
DOUT1_2	ECSET1					

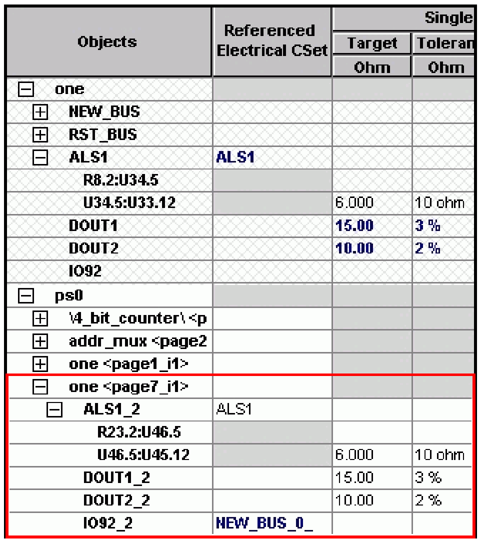
Figure 6-6 Differential pair ***DP1_2*** **before and after restoring the constraints from the child block,** *one*

Restoring Constraints in a Hierarchical Block

When you perform the restore from definition operation on a hierarchical block, all the constraints and the objects are restored. This includes all the objects and constraints added, modified, or deleted in the top-level block.



Objects	Referenced Electrical CSet	Single	
		Target	Toleran
		Ohm	Ohm
one			
NEW_BUS			
RST_BUS			
ALS1	ALS1		
R8.2:U34.5			
U34.5:U33.12		6.000	10 ohm
DOUT1		15.00	3 %
DOUT2		10.00	2 %
IO92			
ps0			
\4_bit_counter\<p			
addr_mux <page2			
one <page1_i1>			
one <page7_i1>			
ALS1_2	ALS1		
R23.2:U46.5		4.000	5 %
U46.5:U45.12		10.00	8 ohm
DOUT1_2	ECSET1	20.00	10 %
DOUT2_2		15.00	2 %
IO92_2	NEW_BUS_0_		



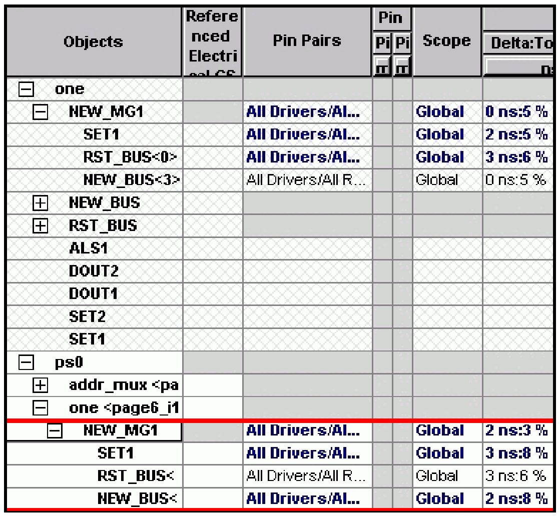
Objects	Referenced Electrical CSet	Single	
		Target	Toleran
		Ohm	Ohm
one			
NEW_BUS			
RST_BUS			
ALS1	ALS1		
R8.2:U34.5			
U34.5:U33.12		6.000	10 ohm
DOUT1		15.00	3 %
DOUT2		10.00	2 %
IO92			
ps0			
\4_bit_counter\<p			
addr_mux <page2			
one <page1_i1>			
one <page7_i1>			
ALS1_2	ALS1		
R23.2:U46.5			
U46.5:U45.12		6.000	10 ohm
DOUT1_2		15.00	3 %
DOUT2_2		10.00	2 %
IO92_2	NEW_BUS_0_		

Figure 6-7 Lower-level block, *one*, before and after restoring the hierarchical block from its definition

Note: When you restore lower-level constraints on a Design Entry HDL block, a number of false pin-pair messages are reported. This is because, restoring lower-level constraints recreates pin-pairs.

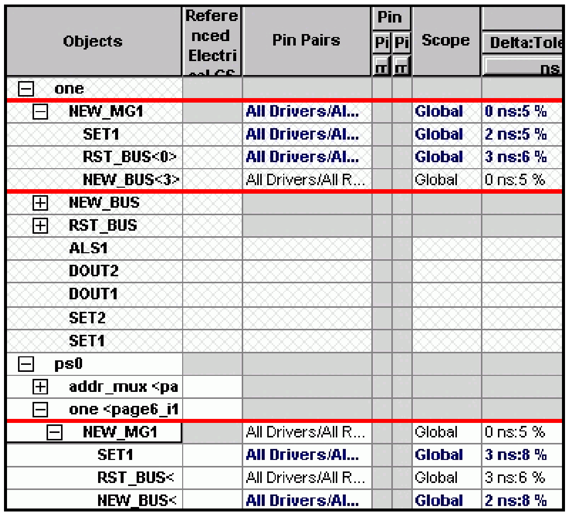
Restoring Constraints in a Matched Group

The Relative Propagation Delay constraint value and the members are restored. In our example, if the matched group member nets were changed in the context of the top-level design, *ps0*, performing the restore operation on the matched group object would restore the original member nets from the child block. However, if there are constraint overrides on the member nets set in the context of *ps0*, the restore operation on the matched group object will preserve the overridden net/Xnet constraints.



Objects	Referenced Electri cal CS	Pin Pairs	Pin Pi	Pin Pi	Scope	Delta:To ns
⊟ one						
⊟ NEW_MG1		All Drivers/Al...			Global	0 ns:5 %
SET1		All Drivers/Al...			Global	2 ns:5 %
RST_BUS<0>		All Drivers/Al...			Global	3 ns:6 %
NEW_BUS<3>		All Drivers/All R...			Global	0 ns:5 %
⊞ NEW_BUS						
⊞ RST_BUS						
ALS1						
DOUT2						
DOUT1						
SET2						
SET1						
⊟ ps0						
⊞ addr_mux <pa						
⊟ one <page6_i1						
⊟ NEW_MG1		All Drivers/Al...			Global	2 ns:3 %
SET1		All Drivers/Al...			Global	3 ns:8 %
RST_BUS<		All Drivers/All R...			Global	3 ns:6 %
NEW_BUS<		All Drivers/Al...			Global	2 ns:8 %

Figure 6-8 Matched group NEW_MG1 with overridden constraints at the matched group- and member net- level

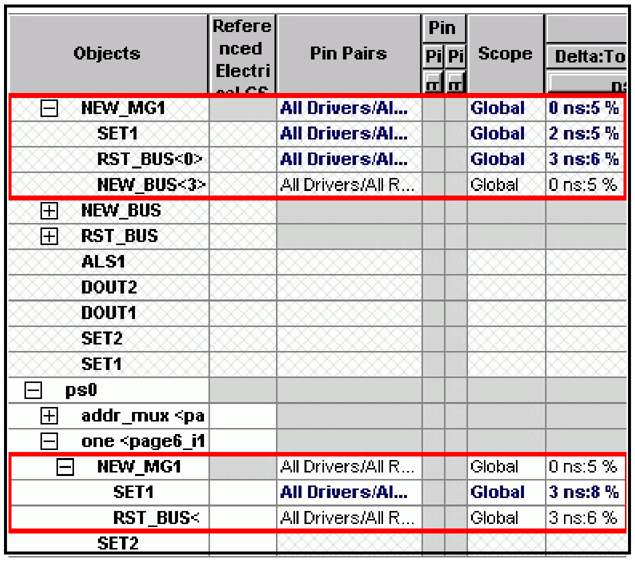


Objects	Referenced Electri cal CS	Pin Pairs	Pin Pi	Pin Pi	Scope	Delta:Tole ns
⊟ one						
⊟ NEW_MG1		All Drivers/Al...			Global	0 ns:5 %
SET1		All Drivers/Al...			Global	2 ns:5 %
RST_BUS<0>		All Drivers/Al...			Global	3 ns:6 %
NEW_BUS<3>		All Drivers/All R...			Global	0 ns:5 %
⊞ NEW_BUS						
⊞ RST_BUS						
ALS1						
DOUT2						
DOUT1						
SET2						
SET1						
⊟ ps0						
⊞ addr_mux <pa						
⊟ one <page6_i1						
⊟ NEW_MG1		All Drivers/All R...			Global	0 ns:5 %
SET1		All Drivers/Al...			Global	3 ns:8 %
RST_BUS<		All Drivers/All R...			Global	3 ns:6 %
NEW_BUS<		All Drivers/Al...			Global	2 ns:8 %

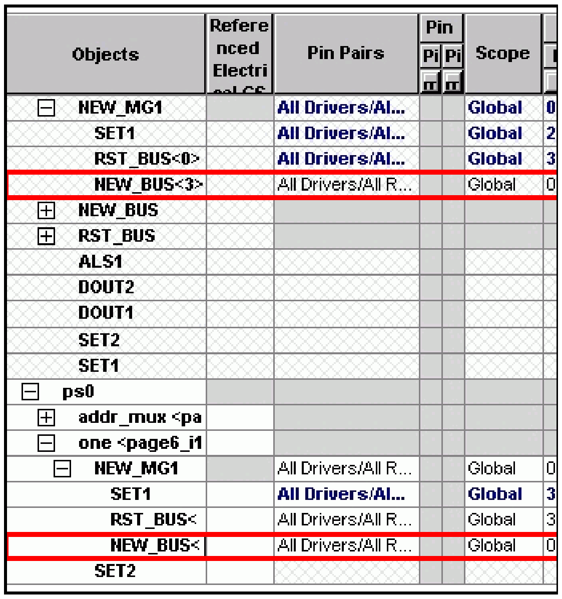
Figure 6-9 Matched group-level constraints of *NEW_MG1* after restoring

Note: If you want to restore a specific object or a member net in a matched group, then select the object and perform the restore operation on the object separately. Remember, restoring the object restores all the constraint information for the object, not just the constraints which are displayed in the active worksheet. If you do not want to restore all the constraints, perform the restore operation on the specific cell(s) that are overridden.

Similarly, if a matched group contains a net/Xnet/pin pair and the object is removed in the context of the top-level design, on running the restore operation, the object is restored in the matched group in the top-level design. For all the newly added objects, the restore should be run explicitly on the objects to ensure that their relative propagation delay constraints reflect what is stored in the definition. If you do not restore the objects, they will inherit the matched group-level values.



Objects	Refere nced Electri	Pin Pairs	Pin Pi	Pi	Scope	Delta:To
NEW_MG1		All Drivers/Al...			Global	0 ns:5 %
SET1		All Drivers/Al...			Global	2 ns:5 %
RST_BUS<0>		All Drivers/Al...			Global	3 ns:6 %
NEW_BUS<3>		All Drivers/All R...			Global	0 ns:5 %
NEW_BUS						
RST_BUS						
ALS1						
DOUT2						
DOUT1						
SET2						
SET1						
ps0						
addr_mux <pa						
one <page6_i1						
NEW_MG1		All Drivers/All R...			Global	0 ns:5 %
SET1		All Drivers/Al...			Global	3 ns:8 %
RST_BUS<		All Drivers/All R...			Global	3 ns:6 %
SET2						

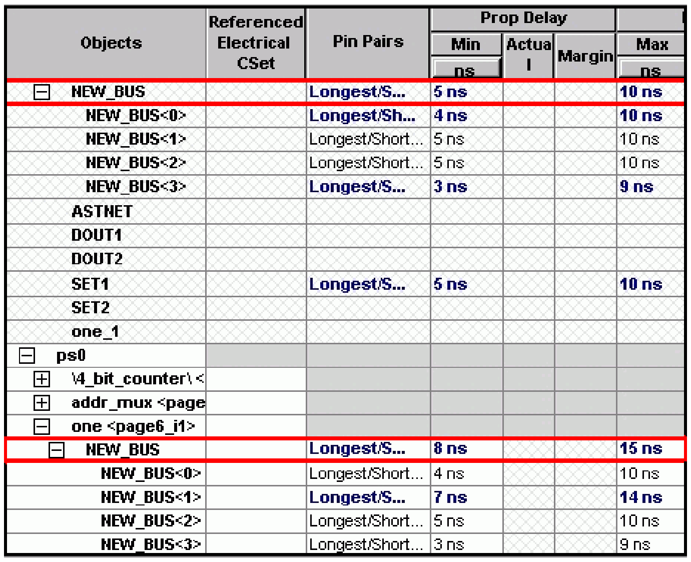


Objects	Refere nced Electri	Pin Pairs	Pin Pi	Pi	Scope	
NEW_MG1		All Drivers/Al...			Global	0
SET1		All Drivers/Al...			Global	2
RST_BUS<0>		All Drivers/Al...			Global	3
NEW_BUS<3>		All Drivers/All R...			Global	0
NEW_BUS						
RST_BUS						
ALS1						
DOUT2						
DOUT1						
SET2						
SET1						
ps0						
addr_mux <pa						
one <page6_i1						
NEW_MG1		All Drivers/All R...			Global	0
SET1		All Drivers/Al...			Global	3
RST_BUS<		All Drivers/All R...			Global	3
NEW_BUS<		All Drivers/All R...			Global	0
SET2						

Figure 6-10 *NEW_BUS<3>***deleted from the matched group** *NEW_MG1* **in the context of the top-level design** *ps0* **is restored after performing the restore operation on** *NEW_MG1*

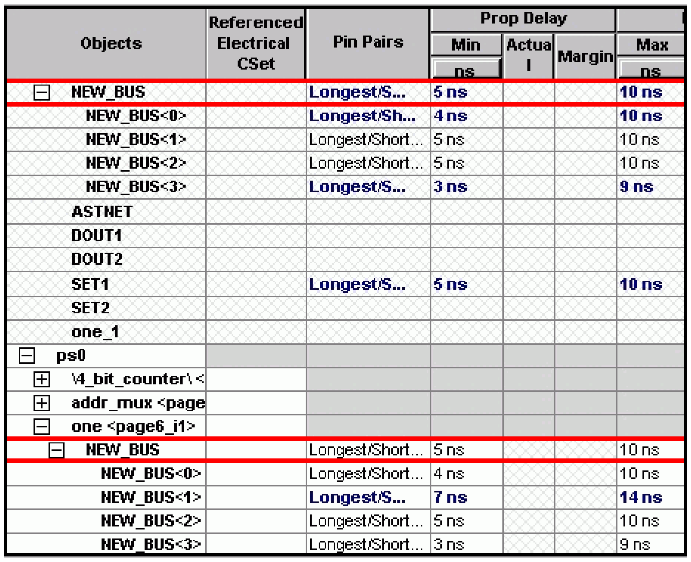
Restoring Bus-Level Constraints

A restore operation performed on a bus restores only bus-level constraints.



Objects	Referenced Electrical CSet	Pin Pairs	Prop Delay			
			Min	Actual	Margin	Max
			ns			ns
⊟ NEW_BUS		Longest/S...	5 ns			10 ns
NEW_BUS<0>		Longest/Sh...	4 ns			10 ns
NEW_BUS<1>		Longest/Short...	5 ns			10 ns
NEW_BUS<2>		Longest/Short...	5 ns			10 ns
NEW_BUS<3>		Longest/S...	3 ns			9 ns
ASTNET						
DOUT1						
DOUT2						
SET1		Longest/S...	5 ns			10 ns
SET2						
one_1						
⊟ ps0						
⊞ \4_bit_counter\<						
⊞ addr_mux <page						
⊟ one <page6_i1>						
⊟ NEW_BUS		Longest/S...	8 ns			15 ns
NEW_BUS<0>		Longest/Short...	4 ns			10 ns
NEW_BUS<1>		Longest/S...	7 ns			14 ns
NEW_BUS<2>		Longest/Short...	5 ns			10 ns
NEW_BUS<3>		Longest/Short...	3 ns			9 ns

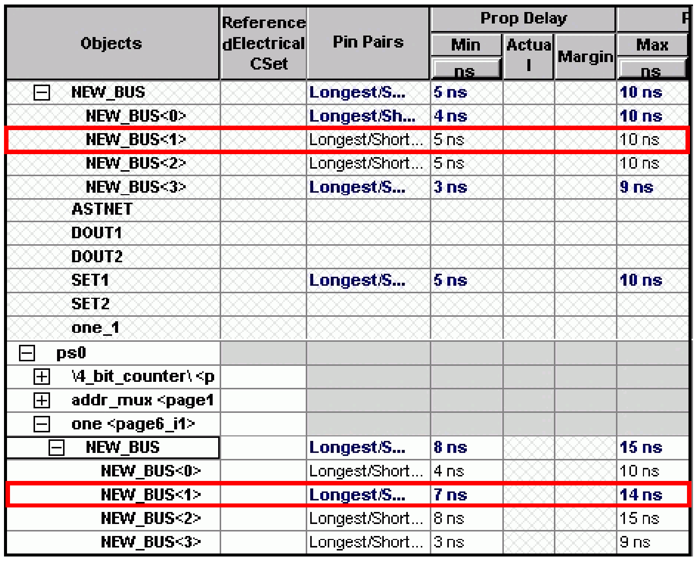
Figure 6-11 Bus-level and member-net level constraints of the bus *NEW_BUS* overridden in *ps0*



Objects	Referenced Electrical CSet	Pin Pairs	Prop Delay			
			Min	Actual	Margin	Max
			ns			ns
⊟ NEW_BUS		Longest/S...	5 ns			10 ns
NEW_BUS<0>		Longest/Sh...	4 ns			10 ns
NEW_BUS<1>		Longest/Short...	5 ns			10 ns
NEW_BUS<2>		Longest/Short...	5 ns			10 ns
NEW_BUS<3>		Longest/S...	3 ns			9 ns
ASTNET						
DOUT1						
DOUT2						
SET1		Longest/S...	5 ns			10 ns
SET2						
one_1						
⊟ ps0						
⊞ \4_bit_counter\<						
⊞ addr_mux <page						
⊟ one <page6_i1>						
⊟ NEW_BUS		Longest/Short...	5 ns			10 ns
NEW_BUS<0>		Longest/Short...	4 ns			10 ns
NEW_BUS<1>		Longest/S...	7 ns			14 ns
NEW_BUS<2>		Longest/Short...	5 ns			10 ns
NEW_BUS<3>		Longest/Short...	3 ns			9 ns

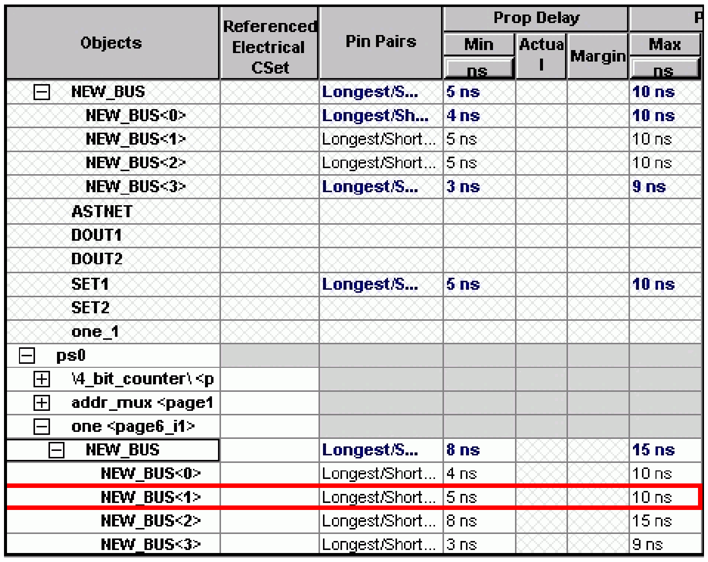
Figure 6-12 Bus-level constraints are restored

Note: To restore constraints on bus bits or member nets, you need to restore the constraints on the member net.



Objects	ReferencedElectrical CSet	Pin Pairs	Prop Delay Min ns	Actual	Margin	Max ns
⊟ NEW_BUS		Longest/S...	5 ns			10 ns
NEW_BUS<0>		Longest/Sh...	4 ns			10 ns
NEW_BUS<1>		Longest/Short...	5 ns			10 ns
NEW_BUS<2>		Longest/Short...	5 ns			10 ns
NEW_BUS<3>		Longest/S...	3 ns			9 ns
ASTNET						
DOUT1						
DOUT2						
SET1		Longest/S...	5 ns			10 ns
SET2						
one_1						
⊟ ps0						
⊞ \4_bit_counter\ <p						
⊞ addr_mux <page1						
⊟ one <page6_i1>						
⊟ NEW_BUS		Longest/S...	8 ns			15 ns
NEW_BUS<0>		Longest/Short...	4 ns			10 ns
NEW_BUS<1>		Longest/S...	7 ns			14 ns
NEW_BUS<2>		Longest/Short...	8 ns			15 ns
NEW_BUS<3>		Longest/Short...	3 ns			9 ns

Figure 6-13 Member-net level constraints overridden in *ps0*



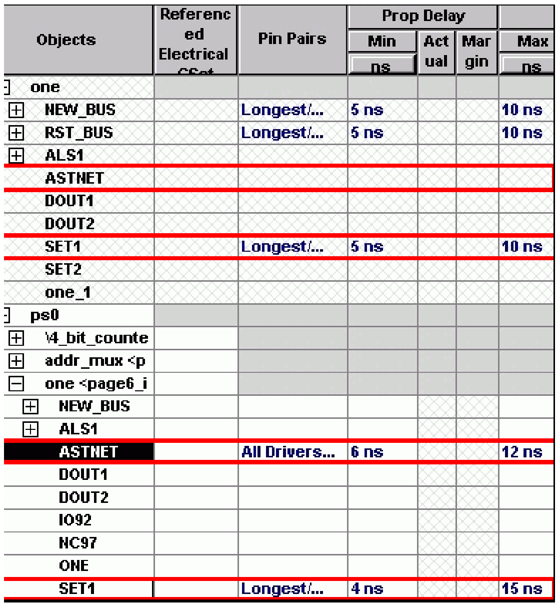
Objects	Referenced Electrical CSet	Pin Pairs	Prop Delay Min ns	Actual	Margin	Max ns
⊟ NEW_BUS		Longest/S...	5 ns			10 ns
NEW_BUS<0>		Longest/Sh...	4 ns			10 ns
NEW_BUS<1>		Longest/Short...	5 ns			10 ns
NEW_BUS<2>		Longest/Short...	5 ns			10 ns
NEW_BUS<3>		Longest/S...	3 ns			9 ns
ASTNET						
DOUT1						
DOUT2						
SET1		Longest/S...	5 ns			10 ns
SET2						
one_1						
⊟ ps0						
⊞ \4_bit_counter\ <p						
⊞ addr_mux <page1						
⊟ one <page6_i1>						
⊟ NEW_BUS		Longest/S...	8 ns			15 ns
NEW_BUS<0>		Longest/Short...	4 ns			10 ns
NEW_BUS<1>		Longest/Short...	5 ns			10 ns
NEW_BUS<2>		Longest/Short...	8 ns			15 ns
NEW_BUS<3>		Longest/Short...	3 ns			9 ns

Figure 6-14 Constraints restored for the member net *NEW_BUS<1>* **after performing the restore operation on it.**

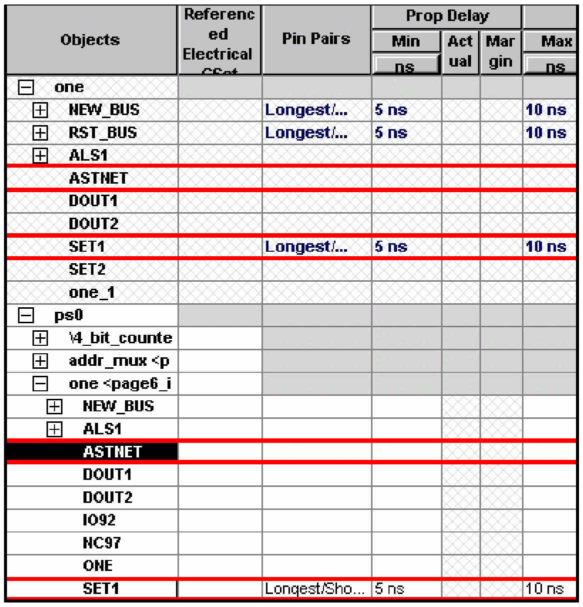
Other Cases

Restoring Multiple Selections

In case of multiple selections, all the selected objects are restored.



Objects	Referenced Electrical	Pin Pairs	Prop Delay			
			Min ns	Actual	Margin	Max ns
one						
NEW_BUS		Longest/...	5 ns			10 ns
RST_BUS		Longest/...	5 ns			10 ns
ALS1						
ASTNET						
DOUT1						
DOUT2						
SET1		Longest/...	5 ns			10 ns
SET2						
one_1						
ps0						
\4_bit_counte						
addr_mux <p						
one <page6_i						
NEW_BUS						
ALS1						
ASTNET		All Drivers...	6 ns			12 ns
DOUT1						
DOUT2						
IO92						
NC97						
ONE						
SET1		Longest/...	4 ns			15 ns

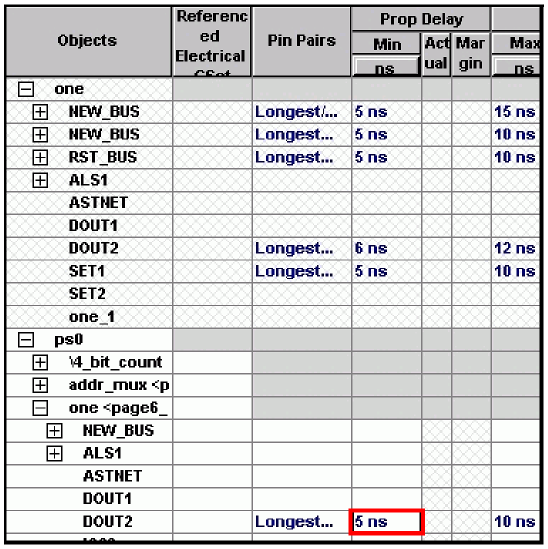


Objects	Referenced Electrical	Pin Pairs	Prop Delay			
			Min ns	Actual	Margin	Max ns
one						
NEW_BUS		Longest/...	5 ns			10 ns
RST_BUS		Longest/...	5 ns			10 ns
ALS1						
ASTNET						
DOUT1						
DOUT2						
SET1		Longest/...	5 ns			10 ns
SET2						
one_1						
ps0						
\4_bit_counte						
addr_mux <p						
one <page6_i						
NEW_BUS						
ALS1						
ASTNET						
DOUT1						
DOUT2						
IO92						
NC97						
ONE						
SET1		Longest/Sho...	5 ns			10 ns

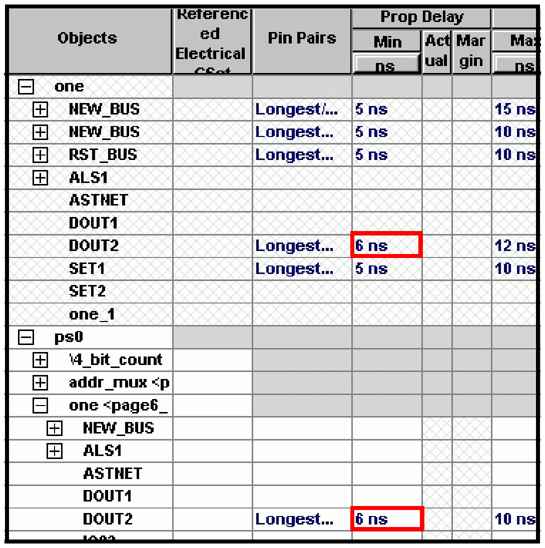
Figure 6-15 Constraints on the nets *ASTNET* and *SET1* restored from the schematic block *one*.

Restoring Specific Constraint Value

To restore a specific constraint value, select the cell and restore the value.



Objects	Referenced Electrical CSet	Pin Pairs	Prop Delay Min ns	Actual	Margin	Max ns
⊟ one						
⊞ NEW_BUS		Longest/...	5 ns			15 ns
⊞ NEW_BUS		Longest...	5 ns			10 ns
⊞ RST_BUS		Longest...	5 ns			10 ns
⊞ ALS1						
ASTNET						
DOUT1						
DOUT2		Longest...	6 ns			12 ns
SET1		Longest...	5 ns			10 ns
SET2						
one_1						
⊟ ps0						
⊞ \4_bit_count						
⊞ addr_mux <p						
⊟ one <page6_						
⊞ NEW_BUS						
⊞ ALS1						
ASTNET						
DOUT1						
DOUT2		Longest...	5 ns			10 ns



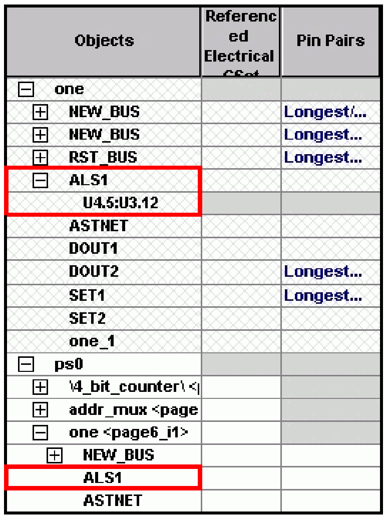
Objects	Referenced Electrical CSet	Pin Pairs	Prop Delay Min ns	Actual	Margin	Max ns
⊟ one						
⊞ NEW_BUS		Longest/...	5 ns			15 ns
⊞ NEW_BUS		Longest...	5 ns			10 ns
⊞ RST_BUS		Longest...	5 ns			10 ns
⊞ ALS1						
ASTNET						
DOUT1						
DOUT2		Longest...	6 ns			12 ns
SET1		Longest...	5 ns			10 ns
SET2						
one_1						
⊟ ps0						
⊞ \4_bit_count						
⊞ addr_mux <p						
⊟ one <page6_						
⊞ NEW_BUS						
⊞ ALS1						
ASTNET						
DOUT1						
DOUT2		Longest...	6 ns			10 ns

Figure 6-16 The Minimum Propagation Delay constraint for net *DOUT2* overridden in *ps0* is restored from the lower-level block *one*.

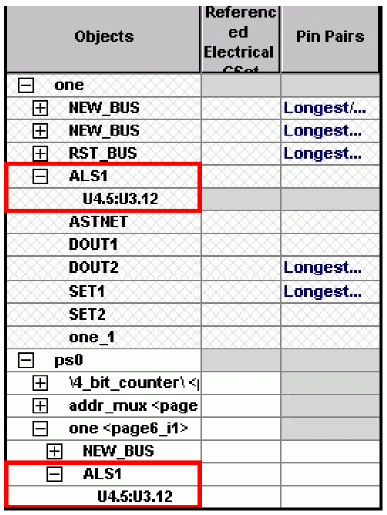
Note: When you perform the restore operation on a specific constraint value in the context of the top-level design, constraint values are pulled from the `.dcf` file of the lower-level block and added to the `OPF` of the top-level design.

Restoring Deleted Objects

Any lower-level object or constraint deleted in the context of the top-level design can be restored.



Objects	Referenced Electrical	Pin Pairs
one		
NEW_BUS		Longest/...
NEW_BUS		Longest...
RST_BUS		Longest...
ALS1		
U4.5:U3.12		
ASTNET		
DOUT1		
DOUT2		Longest...
SET1		Longest...
SET2		
one_1		
ps0		
\4_bit_counter\ <|		
addr_mux <page		
one <page6_i1>		
NEW_BUS		
ALS1		
ASTNET		



Objects	Referenced Electrical	Pin Pairs
one		
NEW_BUS		Longest/...
NEW_BUS		Longest...
RST_BUS		Longest...
ALS1		
U4.5:U3.12		
ASTNET		
DOUT1		
DOUT2		Longest...
SET1		Longest...
SET2		
one_1		
ps0		
\4_bit_counter\ <|		
addr_mux <page		
one <page6_i1>		
NEW_BUS		
ALS1		
U4.5:U3.12		

Figure 6-17 A pin pair, *U33.5:U32.12*, deleted in the context of *ps0* is restored from the lower-level block, *one*, along with constraint values

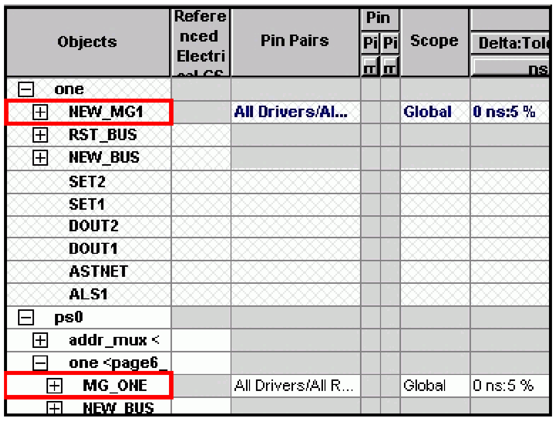
Note: Deleted objects are only re-created by restoring their parent object. When restoring the parent (for example, net *ALS1*), all the constraint information for the parent will also be restored, in addition to the pin pair(s).

The only way to restore deleted matched groups and differential pairs is to restore the block instance from which they originated. As a result, all the constraint information from the block will also be restored.

Restoring Renamed Objects

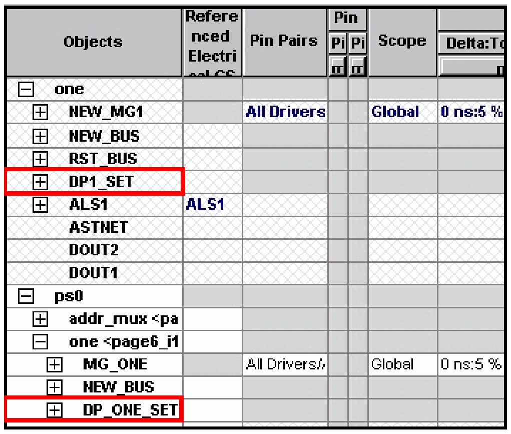
On restoring a matched group or a differential pair renamed in the context of the top-level design, the constraints are restored, but the name is preserved. Also, if you rename a matched group and perform the restore operation on it, you cannot rename it back to the original name.

Note: If you rename an ECSet generated matched group in the ECSet worksheet and restore the ECSet, the name is also restored along with constraints from the child block.



Objects	Referenced Electri	Pin Pairs	Pin Pi	Pi	Scope	Delta:Tol ns
⊟ one						
⊞ NEW_MG1		All Drivers/Al...			Global	0 ns:5 %
⊞ RST_BUS						
⊞ NEW_BUS						
SET2						
SET1						
DOUT2						
DOUT1						
ASTNET						
ALS1						
⊟ ps0						
⊞ addr_mux <						
⊟ one <page6_						
⊞ MG_ONE		All Drivers/All R...			Global	0 ns:5 %
⊞ NEW_BUS						

Figure 6-18 Matched group *NEW_MG1* renamed as *MG_ONE* in the context of *ps0*

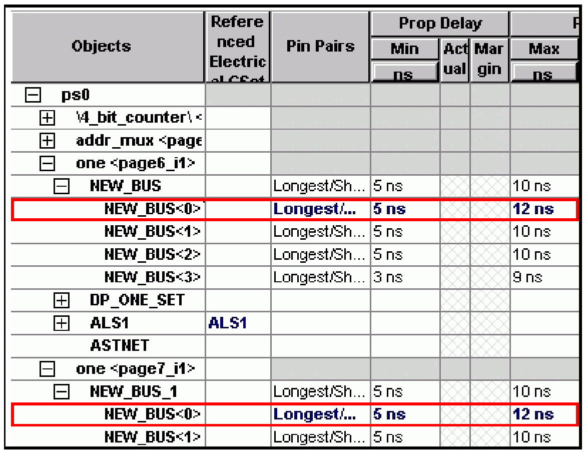


Objects	Referenced Electri	Pin Pairs	Pin Pi	Pi	Scope	Delta:Tc
⊟ one						
⊞ NEW_MG1		All Drivers			Global	0 ns:5 %
⊞ NEW_BUS						
⊞ RST_BUS						
⊞ DP1_SET						
⊞ ALS1	ALS1					
ASTNET						
DOUT2						
DOUT1						
⊟ ps0						
⊞ addr_mux <pa						
⊟ one <page6_i1						
⊞ MG_ONE		All Drivers/			Global	0 ns:5 %
⊞ NEW_BUS						
⊞ DP_ONE_SET						

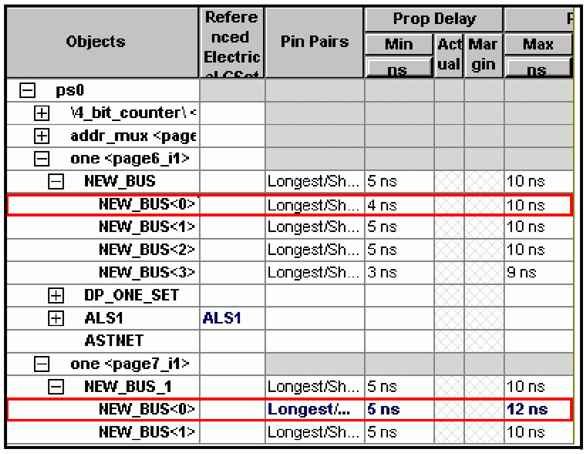
Figure 6-19 Differential pair *DP1_SET* renamed as *DP_ONE_SET* in the context of *ps0*

Restoring in Replicated Blocks

In case of replicated blocks, restoring overridden constraints on one block will restore the values only in that block and not across instances.



Objects	Referenced Electrical CSet	Pin Pairs	Prop Delay Min ns	Actual	Margin	Max ns
⊟ ps0						
⊞ \4_bit_counter\ <						
⊞ addr_mux <page						
⊟ one <page6_i1>						
⊟ NEW_BUS		Longest/Sh...	5 ns			10 ns
NEW_BUS<0>		**Longest/...**	**5 ns**			**12 ns**
NEW_BUS<1>		Longest/Sh...	5 ns			10 ns
NEW_BUS<2>		Longest/Sh...	5 ns			10 ns
NEW_BUS<3>		Longest/Sh...	3 ns			9 ns
⊞ DP_ONE_SET						
⊞ ALS1	ALS1					
ASTNET						
⊟ one <page7_i1>						
⊟ NEW_BUS_1		Longest/Sh...	5 ns			10 ns
NEW_BUS<0>		**Longest/...**	**5 ns**			**12 ns**
NEW_BUS<1>		Longest/Sh...	5 ns			10 ns



Objects	Referenced Electrical CSet	Pin Pairs	Prop Delay Min ns	Actual	Margin	Max ns
⊟ ps0						
⊞ \4_bit_counter\ <						
⊞ addr_mux <page						
⊟ one <page6_i1>						
⊟ NEW_BUS		Longest/Sh...	5 ns			10 ns
NEW_BUS<0>		Longest/Sh...	4 ns			10 ns
NEW_BUS<1>		Longest/Sh...	5 ns			10 ns
NEW_BUS<2>		Longest/Sh...	5 ns			10 ns
NEW_BUS<3>		Longest/Sh...	3 ns			9 ns
⊞ DP_ONE_SET						
⊞ ALS1	ALS1					
ASTNET						
⊟ one <page7_i1>						
⊟ NEW_BUS_1		Longest/Sh...	5 ns			10 ns
NEW_BUS<0>		**Longest/...**	**5 ns**			**12 ns**
NEW_BUS<1>		Longest/Sh...	5 ns			10 ns

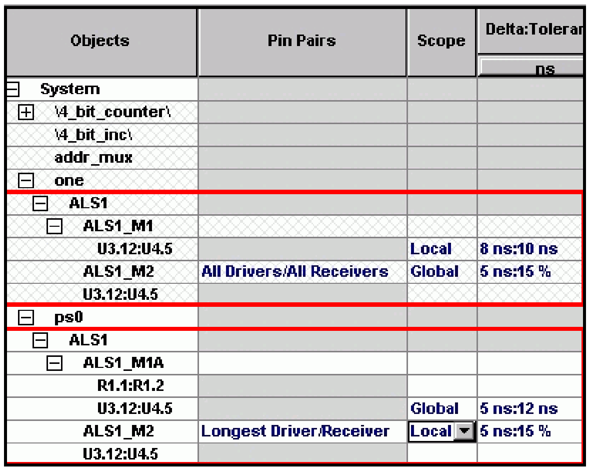
Figure 6-20 Restoring the constraint value of *NEW_BUS<0>* on `page 6` **does not restore the value on** `page 7`

Restoring Electrical Constraints Sets

On restoring an ECSet in the ECSet worksheet, the entire ECSet definition is restored, including the constraints, pin pair, and matched group information. Consider the example shown below. The ECSet *ALS1* is overridden in the context of the top-level design *ps0*:

- Matched group *ALS1_M1* is renamed as *ALS1_M1A*
- A pin pair *R1.1:R1.2* is added to the ECSet

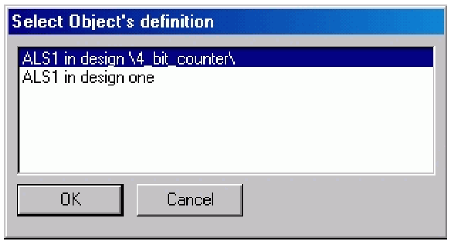
- The `Scope` and `Delta:Tolerance` values are changed for the pin pair *U3.12:U4.5*



Objects	Pin Pairs	Scope	Delta:Toleran
			ns
System			
\4_bit_counter\			
\4_bit_inc\			
addr_mux			
one			
ALS1			
ALS1_M1			
U3.12:U4.5		Local	8 ns:10 ns
ALS1_M2	All Drivers/All Receivers	Global	5 ns:15 %
U3.12:U4.5			
ps0			
ALS1			
ALS1_M1A			
R1.1:R1.2			
U3.12:U4.5		Global	5 ns:12 ns
ALS1_M2	Longest Driver/Receiver	Local	5 ns:15 %
U3.12:U4.5			

Figure 6-21 ECSet definitions overridden in *ps0*

In case of replicated blocks, if a block is instantiated at multiple levels of hierarchy, the Select Object's definition dialog box is displayed. In this dialog box, you choose the level from which you want the definition to be restored. In our example, the lower-level block *one* is instantiated in *ps0*. At the same time *one* is also instantiated in *4_bit_counter*, which is another child block of *ps0*.

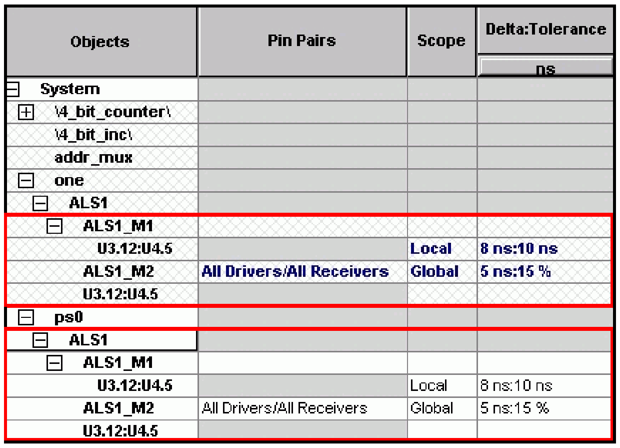


Select Object's definition

ALS1 in design \4_bit_counter\
ALS1 in design one

Figure 6-22 Select Object's definition dialog box

The ECSet definition is restored.



Objects	Pin Pairs	Scope	Delta:Tolerance
			ns
⊟ System			
⊞ \4_bit_counter\			
\4_bit_inc\			
addr_mux			
⊟ one			
⊟ ALS1			
⊟ ALS1_M1			
U3.12:U4.5		Local	8 ns:10 ns
ALS1_M2	All Drivers/All Receivers	Global	5 ns:15 %
U3.12:U4.5			
⊟ ps0			
⊟ ALS1			
⊟ ALS1_M1			
U3.12:U4.5		Local	8 ns:10 ns
ALS1_M2	All Drivers/All Receivers	Global	5 ns:15 %
U3.12:U4.5			

Figure 6-23 ECSet definition restored from the lower-level block *one*.

Note: If you restore a net that is a member of an ECSet-generated matched group and pin pair of that net is a member of another ECSet-generated matched group, restore on net in the matched group also restores the pin pair lying in the other ECSet-generated matched group.

7

Working with Properties

This chapter describes the following sections:

- Using Constraint Manager to Manage Properties on page 133.
- Procedures for Working with Properties in Constraint Manager on page 135.

Refer to the Tools – Customize command in the *Constraint Manager Reference* for more information.

Using Constraint Manager to Manage Properties

Using Constraint Manager you can add, modify, or delete properties for an object. When invoked from System Connectivity Manager, Constraint Manager, can be used to modify the properties of nets, components, and pins. To do this, select the property worksheet for the object in the Properties Tab.

For more information on opening worksheets with property information for objects, see:

- Working with Properties on Nets on page 134
- Working with Properties on Components on page 134
- Working with Properties on Pins on page 134

The object you selected in System Connectivity Manager is highlighted in the property worksheet for the object in Constraint Manager.

Important

If Constraint Manager is launched from Design Entry HDL, only the Net property worksheet is available. Modifying component and pin properties is Constraint Manager is not support for Design Entry HDL- Constraint Manager flow.

Working with Properties on Nets

To work with properties on nets, perform the following steps.

➤ In Constraint Manager, select the *Properties* tab.

➤ Expand the *Net* object folder.

➤ Select the *General Properties* workbook to display the *General Properties* worksheet.

Working with Properties on Components

To work with properties on components, perform the following steps.

➤ Select the *Properties* tab.

➤ Expand the *Component* object folder.

➤ Click the *General Properties* workbook to display the *General Properties* worksheet.

Caution

Component objects folder with Component and Pin properties is available only if you launch Constraint Manager from System Connectivity Manager.

Working with Properties on Pins

To open the worksheet with Pin properties, complete the following steps.

➤ Expand the *Component* object folder and click the *Pin Properties* workbook to display the *Pin Properties* worksheet.

Caution

Component objects folder with Component and Pin properties is available only if you launch Constraint Manager from System Connectivity Manager.

Procedures for Working with Properties in Constraint Manager

The following sections describe how you can work with properties in Constraint Manager:

- Adding Properties in Constraint Manager on page 135
- Editing Properties in Constraint Manager on page 136
- Working with User-Defined Properties in Constraint Manager on page 139
- Sorting Properties Values in Constraint Manager on page 136
- Managing Columns for Properties in Constraint Manager on page 137

Adding Properties in Constraint Manager

To add a property on an object, do the following:

1. Open the property worksheet for the object.
2. Locate the object on which you want to add the property.

   Note: You can perform cross probing from System Connectivity Manager to Constraint Manager to quickly locate objects on which you want to add properties in Constraint Manager.
3. Enter the value of the property in the corresponding cell in the column for the property. For example, if you want to enter the value for the ROOM property, enter the value in the corresponding cell in the ROOM column.

   Note: If the column for the property does not exist in the worksheet, you must add a column for the property in the worksheet. For more information, see Adding a Column for a Property in a Worksheet in Constraint Manager on page 136.

Note the following when working with properties in Constraint Manager:

- Nets are displayed in Constraint Manager using physical (packaged) net names.
- You cannot add properties on a bus (vectored signal) or a vectored pin. You can only add properties on the bits of a bus or on the bits of a vectored pin.
- If a net is aliased to another net or nets, only the base net is displayed in Constraint Manager. The base net inherits all the properties that exist on the nets aliased to it. A property you add on a base net also applies to the nets aliased to it.

Editing Properties in Constraint Manager

You can quickly edit, delete, cut, copy or paste property values.

Note: Undo and redo of properties and constraints is not supported in Constraint Manager.

Tip

You can perform cross probing from System Connectivity Manager to Constraint Manager to quickly locate objects on which you want to edit properties in Constraint Manager.

Sorting Properties Values in Constraint Manager

To sort properties,

➤ Double-click on the column header of the property that you want to sort.

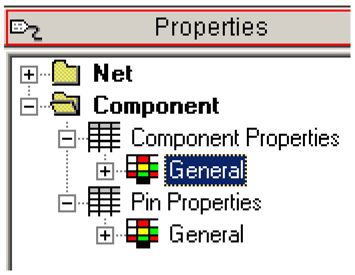
The column is sorted in the ascending or descending order.

Adding a Column for a Property in a Worksheet in Constraint Manager

To be able to add a property on an object, you must add a column for the property in Constraint Manager. For example, if you want to add a property on components in the design, you must add a column for the property.

1. In Constraint Manager, choose *Tools – Customize*.
2. Expand the workbook that contains the property worksheet for the object.

   For example, if you want to add a column for a component property, expand the *Component* object folder in the properties tab, then expand the *Component Properties* workbook to display the *General* worksheet, as shown below:

3. Select the property worksheet for the object, right-click and choose *Add Column*.

The *Add Column* dialog box appears.

4. Depending on whether you want to add a column for pre-defined or user-defined property, do one of the following:
   - ❑ To add a column for a predefined property, from the *Type* drop-down list choose *Pre-defined*. The list of predefined properties are displayed.
   - ❑ To add a column for a user defined property, click the *Type* drop-down list and choose *User-defined*. The list of existing user defined properties are displayed.
5. Select the property for which the column is to be added.
6. Column heading for the column is reflected in the *Name* text field.

   Modify the column name, if required.
7. Click *OK.*

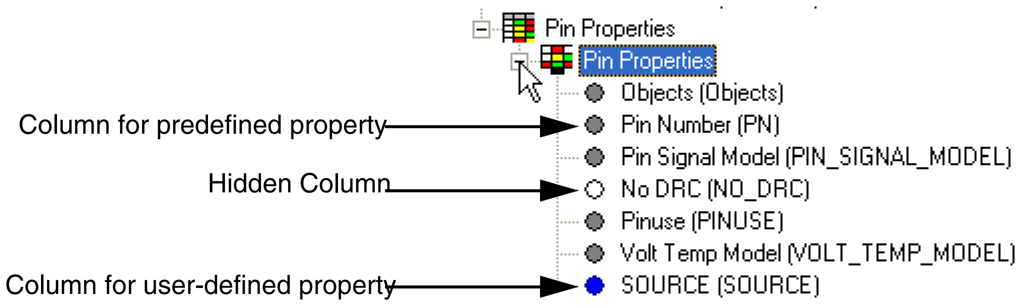
The column for the property is displayed on all the relevant worksheets. For example, if a property can be added on components and on pins, a column for the property is displayed in the *General Properties* worksheet in the *Component* folder and in the *Pin Properties* worksheet.

Managing Columns for Properties in Constraint Manager

You can customize the worksheets in Constraint Manager such that only the required properties are visible in the worksheet. Constraint Manager provides support to hide, show, or delete columns for properties.

1. In Constraint Manager, choose *Tools – Customize*.
2. Expand the worksheet in which you want to hide, show, or delete the column for a property.

For example, if you want to hide the column for a pin property, expand the *Component* object folder, expand the *Pin Properties* workbook, then expand the *Pin Properties* worksheet, as shown below:


Pin Properties
Pin Properties
Objects (Objects)
Column for predefined property
Pin Number (PN)
Pin Signal Model (PIN_SIGNAL_MODEL)
Hidden Column
No DRC (NO_DRC)
Pinuse (PINUSE)
Volt Temp Model (VOLT_TEMP_MODEL)
Column for user-defined property
SOURCE (SOURCE)

In the above figure, the gray circle indicates a column for a predefined property, the circle with white color indicates a hidden column, and the blue circle indicates a column for a user-defined property.

- ❑ To hide a column, select the column for the property, right-click and choose *Visible*.
- ❑ To show a column that is hidden, select the column, right-click and choose *Visible*.
- ❑ To delete a column, select the column, right-click and choose *Delete Column*.

Tip

You can also select the column head for a property in a worksheet, right-click and choose *Hide Column* to hide a column.

Note the following:

- You can only delete the columns for user-defined properties. You cannot delete the columns for predefined properties.

  To know more about user defined properties, see Working with User-Defined Properties in Constraint Manager.

- When you hide or delete the column for a property, the property is not deleted from the objects on which it is added.

Working with User-Defined Properties in Constraint Manager

Constraint Manager lets you define user-defined properties. You can use an user defined property to capture a characteristic of an object.

The following topics provide information on working with user defined properties:

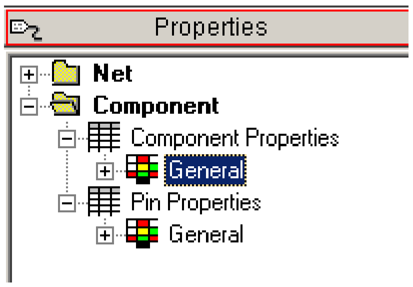
- Defining User-Defined Properties in Constraint Manager on page 139
- Modifying the Definition of User-Defined Properties in Constraint Manager on page 144
- Deleting the Definition of User-Defined Properties in Constraint Manager on page 144

You can also define user-defined properties in System Connectivity Manager.

Defining User-Defined Properties in Constraint Manager

1. In Constraint Manager, choose *Tools – Customize*.
2. Expand the workbook that contains the property worksheet for an object.

   For example, expand the *Component* object folder, then expand the *General Properties* workbook to display the *General* worksheet, as shown below:



3. Select the property worksheet for the object, right-click and choose *Add Column*.

   The *Add Column* dialog box appears.
4. Click the *Type* drop-down list and choose the attribute type as *User-defined*.
5. Click *Create*.

   The *Create Attribute Definition* dialog box appears.
6. In the *Name* field, specify the name of the attribute to be added.

7. From the *Data Type* drop-down list, select the data type to be used for specifying the value of the attribute.
8. In the Objects list box, select the check box next to the objects on which you want to be able to add the property.
   - ❍ Design
   - ❍ Part Instance
   - ❍ Gate Instance
   - ❍ Pin
   - ❍ XNet
   - ❍ Net
   - ❍ Electrical CSet
   - ❍ Diff pair
   - ❍ Bus
   - ❍ Pin Pair
   - ❍ Match Group
   - ❍ Design Instance
   - ❍ Net Class
9. Specify the range of values that are acceptable for the property.

   When you enter a property value, Constraint Manager displays an error message if the value is not within the specified range.
10. Enter a description for the property.
11. Select the *Transfer to/from Physical* check box if you want the property to be transferred between System Connectivity Manager and Allegro PCB Editor along with the netlist when you run the *Export Physical* command.

12. If you have selected the *Part Instance* or *Gate Instance* check box in Step 8, click the *Netlist Options* button and specify the physical netlist options for the property.

Select	If
Create a new physical part for each unique attribute value	You want a new physical part to be created for each instance of a component that has the property with a unique property value. **Note:** If instances of a component have the property with the same value, System Connectivity Manager packages the instances together. However, instances that do not have the property will not be packaged together with instances having the same property value for the property. **Example** If you select this option for a property named MYPROP and if you have an instance of a 74LS00 in your design with the property, MYPROP=ALT1, and another instance of 74LS00 with the property, MYPROP=ALT2, the following two physical parts are created: ■ 74LS00-ALT1 ■ 74LS00-ALT2 If both instances of 74LS00 have the MYPROP=ALT1 property, System Connectivity Manager packages them as a single physical part named 74LS00-ALT1.

Select	If
Package instances with the same attribute value in one physical part	You want component instances having the same property value for the property to be packaged together in one physical part. **Note:** System Connectivity Manager does not package together any instances that have different values for the same property. Also, instances that do not have the property will not be packaged together with instances having the same property value for the property. **Example** If you select this option for a property named MYPROP and if you have two instances of the 74LS00 component in your design with the property, MYPROP=ALT1, Design Editor packages them as a single physical part named 74LS00-ALT1.

Select	If
Package instances with the same attribute value in one physical part. If spare sections exist package sections that do not have.	You want component instances having the same property value for the property to be packaged together in one physical part. However, if spare sections are available, instances without the property will also be packaged together. **Note:** System Connectivity Manager does not package together any instances that have different values for the same property. **Example** If you select this option for a property named MYPROP and if you have four instances of the 74LS00 component with the: ■ Instance i1 having the MYPROP=ALT1 property ■ Instance i2 having the MYPROP=ALT2 property ■ Instance i3 having the MYPROP=ALT1 property, and ■ Instance i4 not having the MYPROP property, then System Connectivity Manager packages instances i1 and i3 together because they have the same property value. The 74LS00 component has four sections. As there are two spare sections in the package, System Connectivity Manager includes the instance i4 also in the same package because i4 does not have the MYPROP property.
None	You do not want to specify any physical netlist options for the property.

13. Click *OK*.

The Add Column dialog box appears displaying the new user defined property.

14. Click *OK*.

The column for the property is displayed on all the relevant worksheets. For example, if a property can be added on components and on pins, a column for the property is displayed in the *General Properties* worksheet in the *Component* folder and in the *Pin Properties* worksheet.

Modifying the Definition of User-Defined Properties in Constraint Manager

Caution

You cannot modify the definition of a predefined property.

To modify the definition of a user-defined property:

1. In Constraint Manager, choose *Tools – Customize*.
2. Expand the workbook that contains the property worksheet for an object.

   For example, expand the *Component* object folder, then expand the *General Properties* workbook to display the *General* worksheet.
3. Select the property worksheet for the object, right-click and choose *Add Column*.

   The *Add Column* dialog box appears.
4. Click the *Type* drop-down list and choose *User-defined*.

   The list of existing user-defined properties are displayed.
5. Select the property you want to modify and click *Edit*.

   The *Edit Attribute Definition* dialog box appears.
6. Modify the user defined property and click *OK*.
7. Click *OK.*

   The *Add Column* dialog box appears.
8. Click *Cancel*.

Deleting the Definition of User-Defined Properties in Constraint Manager

Caution

If you delete the definition for a user-defined property, the property is deleted from all the objects in the design on which it exists.

Note: You cannot delete the definition of predefined properties. If you do not want to see the column for a predefined property, you can hide it. For more information, see Managing Columns for Properties in Constraint Manager on page 137.

To delete the definition of a user-defined property:

1. In Constraint Manager, choose *Tools – Customize*.
2. Expand the workbook that contains the property worksheet for an object.

   For example, expand the *Component* object folder, then expand the *General Properties* workbook to display the *General* worksheet.
3. Select the property worksheet for the object, right-click and choose *Add Column*.

   The *Add Column* dialog box appears.
4. Click the *Type* drop-down list and choose *User-defined*.

   The list of existing user defined properties are displayed.
5. Select the property you want to delete and click *Delete*.
6. If the property is being used, a message appears stating that all instances of the property will be deleted.

   Click Yes.

   The property is deleted from all the objects in the design. The column for the property is also deleted from the relevant worksheets. For example, if a property can be added on components and on pins, the column for the property is deleted from the *General* worksheet in the *Component Properties* workbook and from the *General worksheet* in the *Pin Properties* workbook.

A

Pre-16.0 Designs in 16.0 Release

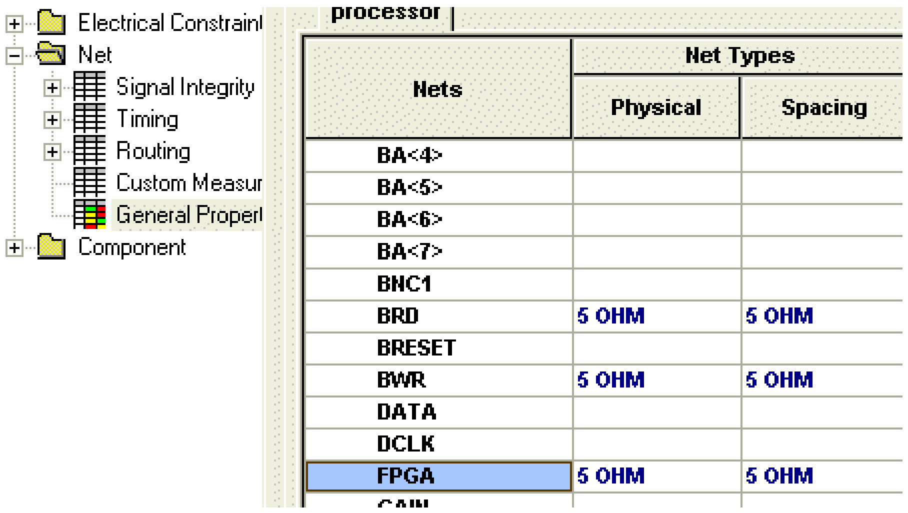
If you open an old design in the 16.0 release, the existing net physical type and net spacing type constraints in the design are converted to corresponding physical and spacing classes in Constraint Manager. The `NET_PHYSICAL_TYPE` and `NET_SPACING_TYPE` values that existed on nets are now listed as classes in Constraint Manager. The nets to which these properties were attached are listed as class members.

Scenario 1

If you have a 15.7 design, for which the logical design is in sync with the board design and you now open the logical design in 16.0 and launch SCM, the following modifications are noticed:

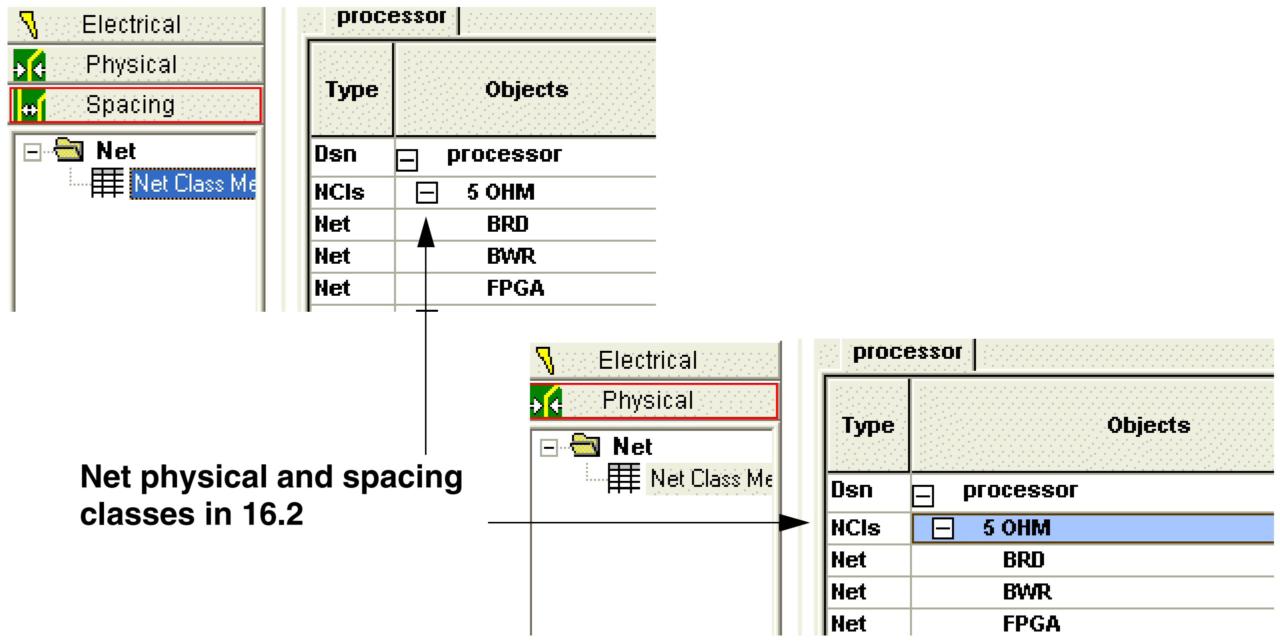
- The values assigned to NET_PHYSICAL_TYPE and NET_SPACING_TYPE properties are converted to net classes listed in the Physical and Spacing domain, respectively.

- All nets with the same value for net physical and spacing constraints are listed as member nets of the same net class object.



Nets	Net Types	
	Physical	Spacing
BA<4>		
BA<5>		
BA<6>		
BA<7>		
BNC1		
BRD	5 OHM	5 OHM
BRESET		
BWR	5 OHM	5 OHM
DATA		
DCLK		
FPGA	5 OHM	5 OHM

Net physical and spacing constraints in release 15.x



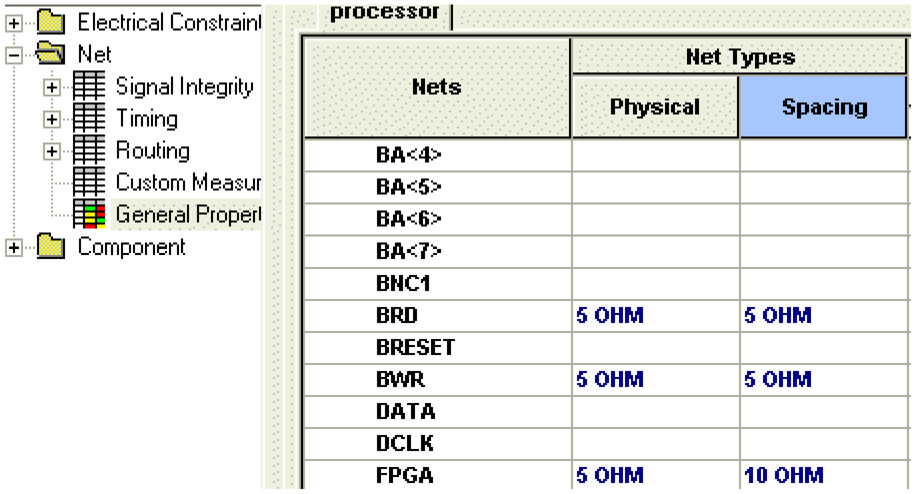
Type	Objects
Dsn	processor
NCls	5 OHM
Net	BRD
Net	BWR
Net	FPGA

Type	Objects
Dsn	processor
NCls	5 OHM
Net	BRD
Net	BWR
Net	FPGA

Net physical and spacing classes in 16.2

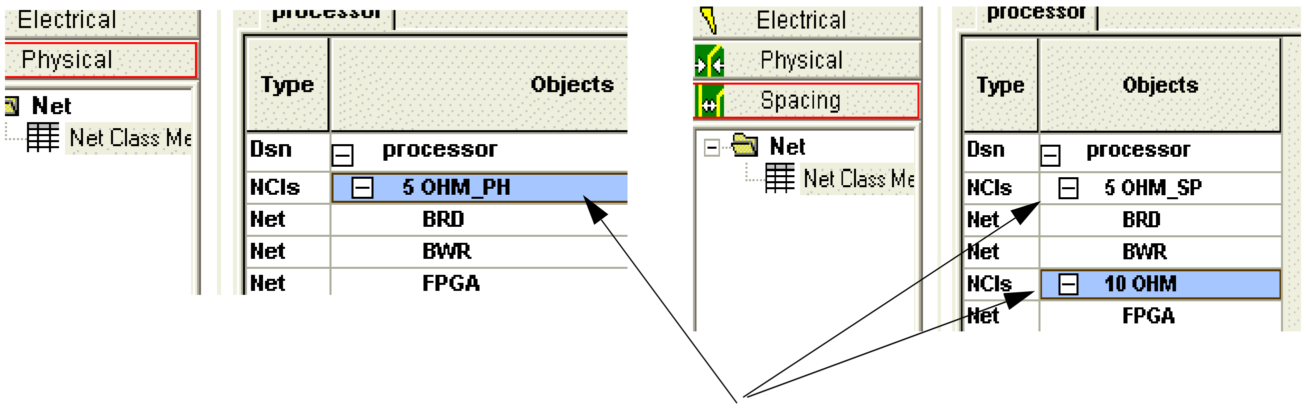
Scenario 2

- By default, Constraint Manager creates the physical and spacing classes with same names. Therefore, in a design if the number of nets with the same physical constraint is different from the number of nets with similar spacing constraint, the net physical class created in the 16.0 release has `_PH` attached as suffix to the net class name. Similarly, the net class created in the spacing domain has `_SP` as suffix.



Nets	Net Types	
	Physical	Spacing
BA<4>		
BA<5>		
BA<6>		
BA<7>		
BNC1		
BRD	5 OHM	5 OHM
BRESET		
BWR	5 OHM	5 OHM
DATA		
DCLK		
FPGA	5 OHM	10 OHM

Separate values for net physical and spacing constraints in 15.x release



Type	Objects
Dsn	processor
NCls	5 OHM_PH
Net	BRD
Net	BWR
Net	FPGA

Type	Objects
Dsn	processor
NCls	5 OHM_SP
Net	BRD
Net	BWR
NCls	10 OHM
Net	FPGA

Net classes in the physical and spacing domain in the 16.0 release

Front-to-back flow

Consider the following situation:

You have a design created in a pre-16.0 release, where the logical design is in sync with the board design. Now open the logical design in System Connectivity Manager, launched from 16.0, and open the board in Allegro PCB Editor launched from 16.0.

If your board design has nets with overridden physical constraints or has regions defined along with physical and spacing constraints, the net classes created for the board design are not in sync with the net classes created for the logical design. Additional classes are created in Allegro PCB Editor causing the logic design to be out of sync with the board. To ensure that the additional classes are available in front-end tools, it is recommended that you synchronize the designs by running back-to-front flow, using the *Import Physical* command. After running the back-to-front flow, you can use the *Export Physical* command to synchronize the physical designs with changes in the logical design.

B

Glossary

Overshoot

Overshoot is the maximum voltage swing above the input voltage. It specifies the acceptable voltage limits of logic families.

Noise Margin

Noise margin is the voltage difference between the maximum voltage dip and the active high threshold or between the maximum voltage dip and the active low threshold.

Jitter

Jitter is the deviation in pulse width of a clock cycle, keeping the clock cycle same.

Sensitive Edge

Sensitive edge signals are those that drive receivers by their edge thresholds. A typical example of a sensitive edge signal is a clock signal.

First Incident Switch

First incident switching is the switching voltage of sufficient amplitude at the initial rise of a signal which is sufficient to drive receivers.

Propagation Delay

Propagation delay is the summation of all calculated transmission line delays along the shortest path between two points. The default unit for propagation delay is ns.

Settle Time

Settle time is the time required for a ringing signal to stabilize to within a specified range of the final value.

Minimum First Switch Time

Specifies the maximum transmission line wire delay plus distortions differing from the nominal driver rise-or-fall time seen in the receiver rise-or-fall.

Duty Cycle

The portion of the time the pulse stimuli is held in the high state as a fraction of the entire pulse period. A value of 0.5 represents equal high and low portions of the cycle period.

Simultaneous Switching Noise

When a number of drivers switch simultaneously in a digital system, a sudden change in current occurs through the power and ground connections to the die. Because of the parasitic inductance that exists in this path, any current change produces a temporary fluctuation in the power and ground voltages as seen by the die. This is typically referred to as Simultaneous Switching Noise (SSN), or Ground Bounce. Simultaneous switching noise can cause noise at the output of non-switching drivers. This noise will then propagate to loads on the net and potentially cause false switching.

Impedance

Impedance is the ratio of input voltage to input current for a transmission line (Z0 = V/I). When a source sends a signal down a line, this is the impedance it must drive. The Source will not see a change in its loading Impedance until 2*TD, where TD is the delay of the line.

Reflection

A reflection on a transmission line is an echo. A portion of the signal power (voltage and current) transmitted down the line goes into the load, and a portion is reflected. Reflections are prevented if the load and the line have the same impedance.

Setup Time

The time for which a digital signal A must be stable and unchanging prior to another digital signal of interest B.

Setup time is most often associated with activity of digital signals immediately before a clock event when these signals must be stable and ready as necessary inputs to clocked circuits, especially latches.

Hold Time

The time for which a digital signal A must be stable and unchanging following the change of another digital signal of interest B.

This parameter is very important with synchronous state machines employing feedback logic that can change as a result of the clock.

Clock Skew

Clock skew is the difference in arrival time between clock and data at a logic gate.

Differential Pair

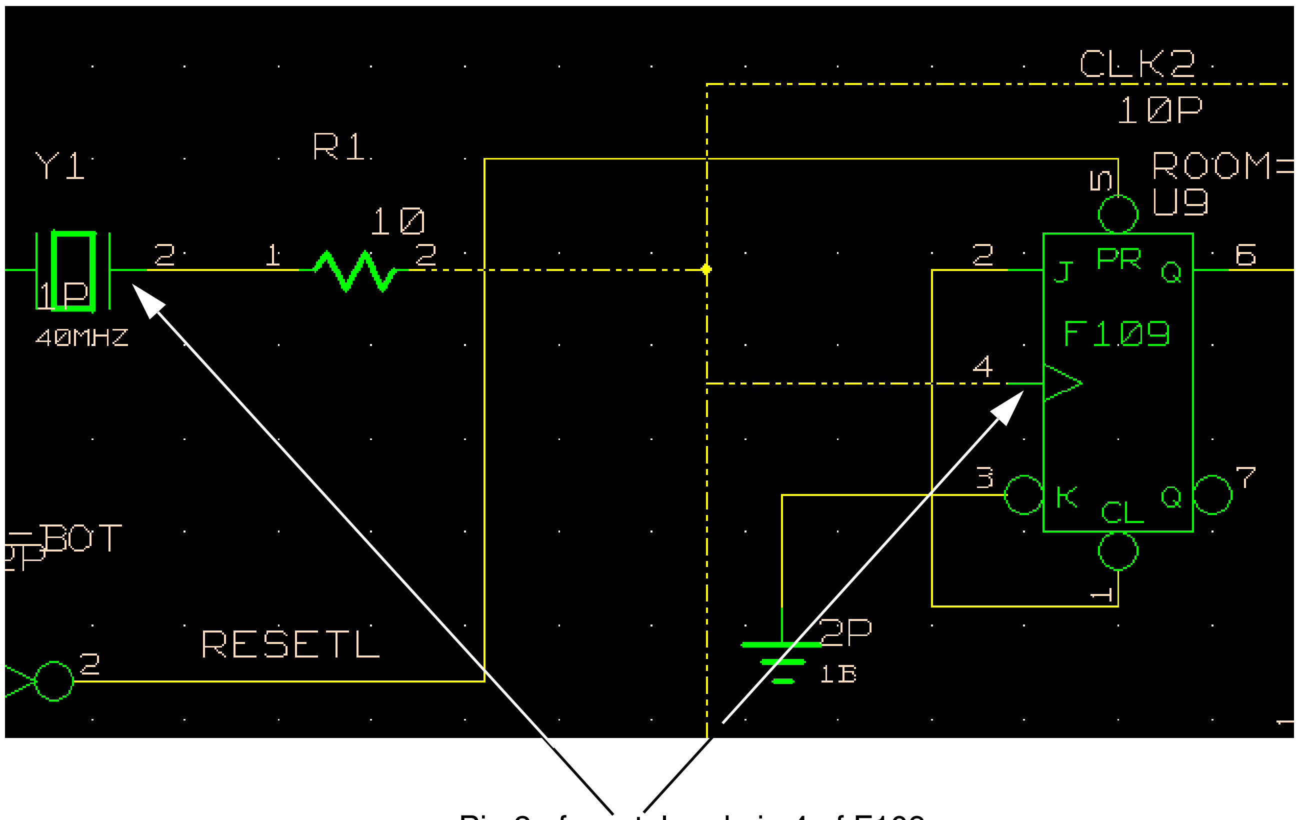
A differential pair represents a pair of nets or Xnets that will be routed in a way that the signals passing through them are opposite in sign with respect to the same reference. This ensures that any electromagnetic noise in the circuit is cancelled out.

Pin-Pair

A pin-pair represents a pair of logically connected pins, often a driver-receiver connection. Pin-pairs may not be directly connected but they must be on the same net or Xnet. A pin-pair for a net connecting component 1 and component 2 is represented as follows:

```
reference designator of component 1.pin number : reference designator of component
     2.pin number
```

For example in the following figure, a pin-pair for net CLK2 is:



Pin 2 of crystal and pin 4 of F109
form a pin-pair for net CLK2.

You can specify a pin-pair explicitly, or it can be derived based on the length of the physical net between the pins forming the pin-pair. The length of the net is determined when the board for the schematic is placed and routed in PCB Editor. Accordingly, the pin-pairs are categorized as follows:

- **longest/shortest pin-pair**

  Out of all the possible pairs of pins that a net connects, the longest pin-pair is the one between whose pins the length of the connecting net is maximum. Similarly, the shortest pin-pair is the one between whose pins the length of the connecting net is minimum.

- **longest/shortest driver-receiver**

  Out of all the pairs of pins for a net where one pin outputs the signal (driver) on the net and the other takes input (receiver), the longest driver-receiver is the one between whose pins the length of the connecting net is maximum. Similarly, the shortest driver-receiver is the one between whose pins the length of the connecting net is minimum.

- **all driver-receiver pin-pairs**

  This refers to all possible pairs of pins for a net such that one pin outputs the signal (driver) on the net and the other takes input (receiver).

Electrical Constraint Set

An electrical constraint set (ECSet) is a collection of constraints and their default values. An ECSet reflects a particular design requirement. You can capture any or all electrical constraints in an ECSet.

ECSets reside in the Electrical Constraint Set object folder. You can create ECSets for signal integrity, timing, and routing constraints.

Primary Gap

Indicates the ideal edge-to-edge spacing between the pair of nets in the differential pair that should be maintained for the entire length of the pair.

Index