Compass Timing Analyzer Engine Timing Constraints

Chapter1

Introduction

1.1 Definitions

The following definitions specify the upper-bounds and lower bounds of the COMPAS timing constraints.

maxAllowedDrive specifies the maximum allowed drive value specified to be 100 ns/pf.

maxAllowedLoadInPf specifies the maximum allowed load value specified to be 100 pf.

maxAllowedLoadInFf specifies the maximum allowed load value specified to be 100000 ff.

maxAllowedCap specifies the maximum allowed wire cap value specified to be 100 pf.

maxAllowedRCDelay specifies the maximum allowed ramp delay value specified to be 100 ns.

maxAllowedPeriod specifies the maximum allowed clock period value specified to be 1000 ns.

maxAllowedInputDelay specifies the maximum allowed input delay value specified to be 1000 ns.

maxAllowedInputDelay specifies the maximum allowed output delay value specified to be 1000 ns.

maxAllowedArrivalTime specifies the maximum allowed arrival time value specified to be 1000 ns.

maxAllowedRequiredTime specifies the maximum allowed required time value specified to be 1000 ns.

minAllowedVddLevel specifies the minimum vdd level and is 4V.

maxAllowedVddLevel specifies the minimum vdd level and is 6V.

maxAllowedTemperature specifies the maximum allowed temp in the set temp command and is 125 celsius.

minAllowedTemperature specifies the minimum allowed temp in the set temp command and is -55 celsius.

Chapter2

General Description

2.1 Functional Summary

2.1.1 Introduction

In this section, we basically review the meaning of a timing constraint and then briefly introduce the syntax and the semantic of the COMPASS timing constraint constructs. This section is illustrated with a complete example and with some guide lines to explain to the reader how to set timing constraints using COMPASS constructs.

2.1.2 Definition

A timing contraint is a way to specify a delay between nodes in a design. The specified delay can be a minimum delay or a maximum delay. When the specified delay is a minimum delay we speak about minDelay timing constraint or lower-bounded timing constraint. When the specified delay is a maximum delay we speak about maxDelay timing constraint or upper-bounded timing constraint.

More accurately, a maxDelay timing constraint is composed of:

- 1. a set of input nodes;
- 2. a set of output nodes;
- 3. for each input node, a signal maximum arrival time, that is the latest time at which the signal is available on that node;
- 4. for each output node, a signal maximum required time, that is the latest time at which the signal should be available on that node;

And a minDelay timing constraint is composed of:

- 1. a set of input nodes;
- 2. a set of output nodes;
- 3. for each input node, a signal minimum arrival time, that is the earliest time at which the signal is available on that node;
- 4. for each output node, a signal minimum required time, that is the earliest time at which the signal should be available on that node;

Timing constraints are used for at least the two following different reasons:

- 1. for checking that a design meets some performance requirements;

- 2. for guiding performance driven optimization tools;

Checking a timing constraint consists in propagating timing constraint input node arrival time along the design and comparing the value of the propagated time and the required time on the output node. For maxDelay timing constraint, we define the “slack” as the difference between the required time and the arrival time. For minDelay timing constraint, we define the “slack” as the difference between the arrival time and the required time. A design is said to be violating a timing constraint when at least one of the timing constraint output node slack is negative. The worst path of a timing constraint is the path ending on the timing constraint output node whose slack is the smallest. Guiding a performance driven optimization tool consists in providing it with the user design and timing constraints and asking it to optimize the design in order to meet those timing constraints.

2.1.3 Origin of the Timing Constraints

There are a lot of origin for timing constraints. In the following we just list some of these:

- 1. the environment world of a design is usually forcing some arrival times on external input connector and some required times on some output connectors;
- 2. the clock frequency and phase is usually forcing some arrival time on sequential element data output and some required time on some sequential element data input connector;
- 3. a desired clock latency is forcing some arrival time (resp. some required time) on the clock network input node (resp. output nodes).

2.1.4 Clocked Designs

2.1.4.1 Introduction

The goal of this section is to remind the reader some of the basis of the clock-network and to define some terms that will be re-used later in that specification document. In the following figure (see Fig 2.1), where there exists some combinational paths between the flip-flop ff1 data output connector to the flip-flop ff2 data input connector, let’s use the following naming convention:

-- ff1 will be called the **master** flip-flop;

-- ff2 will be called the **slave** flip-flop;

-- ck1 will be called the **master** clock;

-- ck2 will be called the **slave** clock;

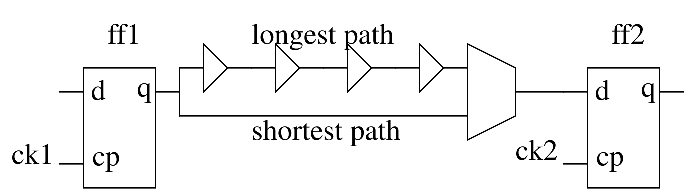

Fig1.1. A typical clocked design.

-- the shortest combinational path between ff1 data output connector to the ff2 data input connector will be called **shortest path**; This path is really composed of the ff1 ck1-to-q timing plusthe combinational path minus the ff2 hold time.

-- the longest combinational path between ff1 data output connector to the ff2 data input connector will be called the **longest path**; This path really is composed of the ff1 ck1-to-q timing plus the combinational path plus the ff2 setup time;

2.1.4.2 Definitions

Let's call:

-- a **launch edge** an active edge of the master clock;

-- a **latch edge** an active edge of the slave clock;

2.1.4.3 Setup check

The **setup check** checks **setup relations** between pair of flip-flops; a setup relation tells that the correct data signal value is stable on the slave flip-flop in time to be properly latched. A setup relation thus determines one **setup launch edge** among the list of the launch edges and one **setup latch edge** among the lsit of latch edges. In turns, the difference between the setup latch edge and setup launch edge implyes a **maximum delay** constraint for the longest path starting on a master flop and ending on a slave flop. In the case of single-frequency design (see fig. 1.2.a) , the setup relation is simple: It tells that the longest path between a pair of flip-flop clocked on the same clock should be smaller than the clock period. In the case of the multi-frequency design (see fig 1.2.b) , such as the one presented on figure 1.1, there may be more than one setup relation between the same pair of flip-flops.

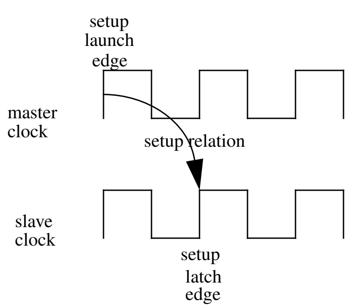

Fig. 1.2.a: Master clock = slave clock

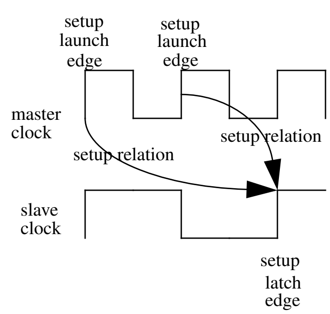

Fig. 1.2.b: Master clock is different from slave clock.

In that case, the setup check first looks for the smallest difference between the setup latch edges and the setup launch edges. It then checks that the longest path is smaller than that difference.

2.1.4.4 Hold Check

To guarantee good design behavior, **hold relations** are checked around all the different setup relations detected by the setup check as follows:

-- a data fired by the master clock active edge that follows the setup launch edge must not be latched by the setup latch edge;

-- a data fired by the setup launch edge must not be latched by the slave clock active edge that preceedes the setup launch edge;

The most restrictive hold relation allows to define the **hold launch edge** and the **hold latch edge** respectively as the master clock active edge and the slave clock active edge creating the hold relation. In turns, the difference between the hold latch edge and the hold launch edge implies a **minimum delay** constraint for the shortest path starting on a master flop and ending on a slave flop. Fig. 1.3.a shows the hold relations on a single frequency design. Fig. 1.3b. presents the hold relation on a multi-frequency design.

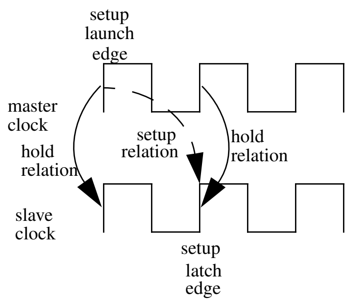

Fig. 1.3.a: Master clock = slave clock

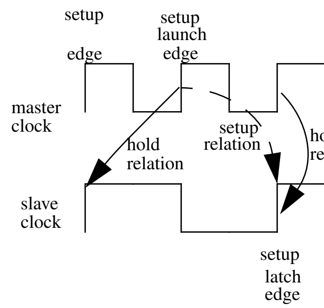

Fig. 1.3.b: Master clock is different from slave clock.

2.1.4.5 Multi-cycle clock

Sometimes, for some pairs of flip-flops, the designer wants to relax the default maximum and minimum delay value automatically detected by the default setup and hold checkers.

Multi-cyling clock is a general mechanism which allows you to escape the default detected setup edge and let's you re-define your own edge. More accurately, multi-cycling a clock allows you:

-- to specify a **setup multiplier** which usually is the number of slave active edges you want to add to the default setup latch edge for setup checking; if you do not multi-cycle the clock, the default number is obviously one. In summary, setting a setup multiplier of “n” pushes forward the setup latch edge by an amount of “n - 1” slave clock cycles. Note that the setup multiplier can also be specified as the number of master active edges you want to remove from the default setup launch edge. In this case, setting a setup multiplier of “n” consists in pushing backward the setup launch edge by an amount of “n-1” master clock cycles.

-- to specify a **hold multiplier** which usually is the number of master active edges you want to add to the default detected hold setup edge. The default number is zero, whatever the slave clock is multicycled or not. In summary, setting a hold multiplier of “n” consists in pushing forward the setup launch edge by a quantity of “n” master clock cycle. Note that the hold multiplier can also be specified as the number of slave active edges you want to remove from the setup latch edge; In this later case, setting a a hold multiplier of “n” consists in pushing backward the setup lacth edge by a quantity of “n” slave clock cycle.

Fig 1.4.a presents a single frequency design with a setup multiplier of 2 and a hold multiplier of zero on the slave flop. Fig 1.4.b presents a multi - frequency design with s setup multiplier of 2 and a hold multiplier of zeroon the slave clock.

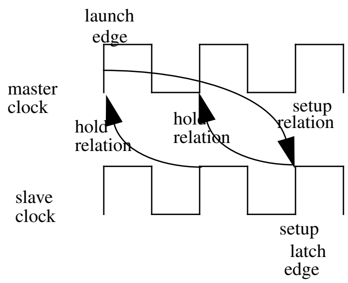

Fig. 1.4.a: Master clock = slave clock
setup multiplier = 2, hold multi = 0

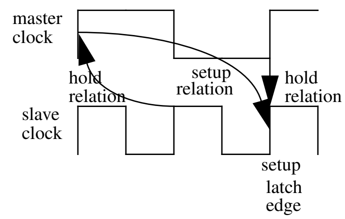

Fig. 1.4.b: Master clock is different from slave clock
setup multiplier = 2, hold multi = 0.

In that example, we clearly see that while we under-constrain the maximum delay constraint introuced by the setup relation, we did create a real minimum delay constraint which was not previously existing. In case the previously defined shortest path is smaller than that minimum delay, the hold checked will report an error, while a synthesis tool will try to fix it by slowing down the shortest path. But often, you know that some flops will be disabled. For instance, in case of the single-frequency design, if both the master and the slave flop are disabled on the seond active edge, you can tell it to the hold checker by specifying a hold multiplier of one. (see figure 1.5.a). In the case of the multi-frequency design, if you know that the slave flop is disabled on the second active edge of the slave clock, you may also tell it to the hold checker by setting a hold multiplier of one.

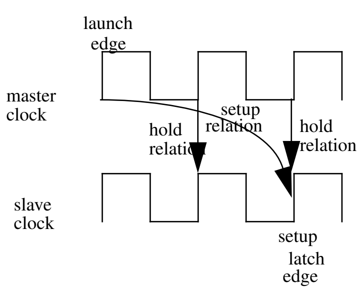

Fig. 1.4.a: Master clock = slave clock
setup multiplier = 2, hold muli = 1

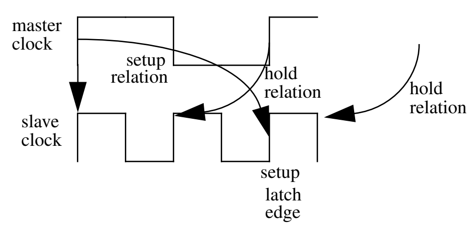

Fig. 1.4.b: Master clock is different from slave clock
setup multiplier = 2, hold multi = 1

2.1.5 Specifying the Timing Constraints

The trivial way to specify a timing constraint is to fill four lists:

- 1. a list of input node names;
- 2. a list of input node arrival times;
- 3. a list of output node names;
- 4. a list of output node required times;

This is acceptable for specifying external arrival and required times on small number of connector design. As soon as the design contains hundreds of connectors, wild cards are requested to match in one statement a complete list of node. In case the user wants to set a constraint coming from a clock declaration, this is no more acceptable for the two following reasons:

- 1. the list of nodes to specify is "design-specific", that is the list of nodes contains
  the names of the sequential element clocked on the clok; It is unlikely that the user will be able to specify the complete list of all these seq. element without omitting one.
- 2. the list of nodes to specify is "technology specific", that is the list of nodes
  contains the name of the data connector of the sequential elements, something as, "q", "qn", "d", "da", "db", "s".

Clearly enough there is room for some special constructs in order to help designer to easily specify timing constraints.

2.1.6 Introduction to the COMPASS Timing Constraints

This is an introduction to the COMPASS timing constraints. For a complete description of the syntax, semantics limitations, we advise the reader to spend hours on the "Specific Requirements" chapter of this document. ALL the timing constraints are undependent. A timing constraint can never relax another timing constraint: the worst one (the tightest one) always win.

Within COMPASS tools, there are four main constructs to specify timing constraints:

- 1. the path cluster constraint;
- 2. the clock constraint;
- 3. the grouped clock constraint;
- 4. the multi-cycle clock constraint;

2.1.7 The Path Cluster Constraint

A path cluster really matches the timing constraint definition here-above presented.

Basically to specify a path cluster, you have to:

- 1. specify a path cluster name;
- 2. specify the input nodes belonging to the path cluster together with their arrival time;
- 3. specify the output nodes belonging to the path cluster together with their required time;

For instance if you want to create a path cluster timing constraint between the input nodes "in1", "in2", "in3" and the output nodes "out1", "out2" and "out3" you will do something as:

```
set pathCluster pc1
set arrivalTime -max 0 in1 in2 in3 -pathCluster pc1
set requiredTime -max 10 out1 out2 -pathCluster pc1
```

```
set requiredTime -max 8 out3 -pathCluster pc1
```

2.1.8 The Clock Constraint

A clock constraint on a design node specifies a timing constraint whose input nodes are the data output connectors of sequential elements clocked on the specified node and whose output nodes are the data input connectors of sequential elements clocked on the specified node. To create a clock constraint you have to declare a clock together with its attribute (period, waveform, latency,skew,...) on a design node. An example of a clock constraint is:

```
set clock clk -period 10
```

This example creates a timing constraint whose:

- 1. input nodes are the data output connectors of the sequential element clocked by the clock clk;
- 2. input node maximum and minimum arrival times are the clock-to-dataOutput propagation time of these sequential element;
- 3. output nodes are the data input connectors of the sequential element clocked by the clock clk;
- 4. output maximum required times are the specified values (in this case 10ns) minus the setup time of these sequential element data input connectors;
- 5. output minimum required times are the hold time these sequential element data input connectors;

Very often, combinational blocks starting on external input connectors and ending on sequential element data input connectors are also subject to a clock timing constraints. As soon as you create a clock constraint as here-above explained, you can easily COMPLETE this timing constraint with some new input node by specifying a clock relative external delay on some external input connectors. An example is

```
set clock clk -period 10
set externalDelay -in in1 1 -clock clk
```

The clock declaration creates exactly the same timing constraint as here-above explained. The clock-relative external delay COMPLETES this timing constraint with:

- 1. one input node which is the external connector "in1";
- 2. an arrival time on this input node which is 1ns;

Very often, combinational blocks ending on external output connectors and starting on sequential element data output connectors are also subject to a clock timing constraints. As soon as you create a clock constraint as here-above explained, you can easily COMPLETE this timing constraint with some new output node by specifying a clock relative external delay on some external output connectors. An example is

```
set clock clk -period 10
set externalDelay -out out1 3 -clock clk
```

The clock declaration creates exactly the same timing constraint as here-above explained. The clock-relative external delay COMPLETES this timing constraint with:

- 1. one output node which is the external connector "out1";
- 2. a required time on this output node which is 10 ns minus 3ns - 7ns;

2.1.9 Grouping Clock Constraints

Clock timing constraint input nodes and output nodes are data input and output connectors of sequential element clocked on that clock. If your design has more than one clock, creating the different clocks does not automatically constrain paths starting on one FF/input connector clocked on one clock and ending ona FF/output connector clocked on another clock. If you want to constrain such paths, you have to create the diffeent clocks, and then you have to “group” these clocks. You group two clocks by using the following command:

```
set clock clkMaster -period masterPeriod
set clock clkSlave -period slavePeriod
set groupClock gr1 -master clkMaster -slave clkSlave
```

(You can group clocks in both way)

Grouping two clock timing constraints will really create a new timing constraint whose:

- 1. input nodes will either be data output connectors of SE clocked on the master clock or external input connector for which an external input delay has been specified relative to the master clock;
- 2. output nodes will either be data input connectors of SE clocked on slave-Clock or external output connector for which an external output delay has been specified on the slave clock;
- 3. the arrival time and required times are computed according to the setup and hold relations determined as explained in section 2.1.4

If you are not satisfied with the default setup and hold reletions automatically deletcted by the setup and hold check, you can overwrite those by specifying “setup multiplier” and “hold multiplier” togetehr with the group clock constraint.

2.1.10 The Multi-Cycle Constraint

In the previous section we presented how we can complete a clock timing constraint by

specifying external delays or by grouping clock together. In this section we explain how we can remove some output
nodes from a clock timing constraint or from a group clock timing constraints in order to create a new timing constraint.
Assuming that in your design some of the flops are delimiting logic which must run at the clock frequency (for instance an adder) and that some other flops are delimiting logic that can run at half the clock frequency (for instance a multiplier), you can first create a clock with the appropriate period. This will constrain the adder and the multiplier to the same timing constraint. You can remove the mulitplier output nodes from the clock timing constraint and create with them a new timing constraint with a double period as required time by using a multi-cycle timing constraint:

```
set clock clk -period 10
set multicycle mc1 -setup 2 -clock clk
set multicycle mc1 -from ff1.dataoutput() ff2.dataoutput()
set MultiCycle mc1 -to ff3.datainput()  ff4.datainput()
```

This will remove the output nodes ff3.datainput() and ff4.datainput() from the clock constraint and create a new timing constraint whose:

- 1. input nodes are the multiCycle start point (ff1.dataoutput() and ff2.dataoutput())
- 2. input node arrival time are still clock-to-dataOutut sequential element propagation time;
- 3. output nodes are the multicycle end point (ff3.datainput() and ff4.datainput());
- 4. output node required time is the twice the period minus the sequential element setup time;

Note that:

- 1. this will remove the multicycle end nodes from the clock constraint; This is necessary, because otherwise these end nodes would still be optimized within the clock constraint, yielding thus the multicycle constraint useless.
- 2. this will NOT remove the multicycle start nodes from the clock constraint; that is the multicycle start nodes will belong to BOTH the multicycle constraint and the clock constraint;
- 3. you can install the same clock output node in more than one multicycle constraint;

The following example re-constrain a pair of flip-flop from a group-clock:

```
set clock clk1 -period 10
```

```
set clock clk2 -period 20
set group clk1_clk2 -master clk1 -slave clk2
set multicycle mc1 -setup 2 -groupclock clk1_clk2
set MultiCycle mc1 -from ff1.dataoutput() ff2.dataoutput()
set MultiCycle mc1 -to ff3.datainput()  ff4.datainput()
```

Note that in the previous example, ff1 has to be clocked on clk1 and ff2 on clk2, otherwise nothing will be constrained.

2.1.11 A Complete Example

Let's take the following example (see Fig. 1.6) and let's name the data blocks as:

- combinational block "block1" type a combinational block starting on external input connectors, and ending on external output connectors;
- combinational block "block2" type a combinational block starting on external input connectors, and ending on SE data input;
- combinational block "block3" type a combinational block starting on SE data output and end ending on SE data input, clocked on the SAME clock;
- combinational block "block4" type a combinational block starting on SE data output and end ending on external output connectors;
- combinational block "block5" type a combinational block starting on SE data output and end ending on SE data input, clocked on different clocks;
- combinational block "block124" type a combinational block starting on both SE data output and external input connectors and ending on both SE data input and external output connectors;
- control block "clock" type the block composed of the clock network;
- control block "reset" type the block composed of the reset logic;

```
    +--------------------------------------------------------+
    |                                                        |
    |   +-----------------------------------------------+    |
 I1 +---+                                               +----+ O0
    |   |     COMBINATIONAL BLOCK1                      +----+ O1
 I2 +---+                                               |    |
    |   +-----------------------------------------------+    |
    |                                                        |
    |         CONTROL BLOCK RESET                            |
  R +-------------------+--------------------+               |
    |                U1 |                 U2 |               |
    |  +---------+ +---+-+  +---------+ +---+-+  +---------+ |
 I3-+--+ COMBIN  | |   R |  | COMBIN  | |   R |  | COMBIN  +-+ O2
 I4 +--+        +--+D   Q+--+        +--+D   Q+--+         | |
 I5 +--+ BLOCK2  | |     |  | BLOCK3  | |     |  | BLOCK4  +-+ O3
    |  +---------+ |   QN+  +---------+ |     |  +---------+ |
 CLK+-------+------+CP   |        |-----+CP   |              |
    |       |      +-----+        |     +-----+              |
    |       \---------------------/                          |
    |         CONTROL BLOCK CLOCK                            |
    |                                                        |
    |  +---------------------------------------------------+ |
 I6 +--+                                                   | |
    |  |      COMBINATIONAL BLOCK124                       +-+ O4
 I7 +--+                                                   | |
    |  |                                                   | |
    |  |        +--------------------------------+         | |
    |  |        |    U3                  U4      |         | |
    |  +        |  +-----+  +---------+ +-----+  |         | |
 I8-+--+        | |     |  | COMBIN  | |     |  |         +-+ O5
    |  |        +--+D   Q+--+        +--+D   Q+--+         | |
 I9 +--+        | |     |  | BLOCK5  | |     |  |         +-+ O6
    |  +---------+ |   QN+  +---------+ |     |  +---------+ |
CLK2+-------+------+CP   |        |-----+CP   |              |
    |              +-----+        |     +-----+              |
CLK3+-----------------------------/                          |
    |          CONTROL BLOCK CLOCK                           |
    |                                                        |
    +--------------------------------------------------------+
```

Fig. 1.6 A typical example.

2.1.11.1 Optimizing the control block.

The control blocks can only be constrained through the usage of "pathCluster" constraint. In the case of the "reset" type block, one would have something as:

```
set pathCluster pc1
set arrivalTime -max 5 R -pathCluster pc1
```

```
set requiredTime -max 12 u1.R u2.R  -pathCluster pc1
```

The data blocks can be constrained through the "pathCluster" constraint or through the "clock" constraint. In the following, we re-view how this would be done for the different types of blocks presented on figure 1.

2.1.11.2 Optimizing Block3-type Data Blocks

Block3-type data blocks can be constrained through the "clock" constraint by "setting" a clock on a node in the monotone combinational transitive fanin of the SEs delimiting the block. In this case, this would be done as follows:

```
set clock clk -period 20
```

Block3-type data blocks can be constrained through the "pathCluster" constraint by:

- 1. creating a pathCLuster
- 2. setting arrivalTime on data output connectors of SE delimiting the block;
- 3. setting requiredTime on data input connector of SE delimiting the block;

In this case, it would be something as:

```
set pathCluster pc1
setarrivalTimeallSE(clk).dataOutput()-addClockPath0-pathClusterpc1
set requiredTime allSE(clk).dataInput() -setup 20 -pathCluster pc1
```

2.1.11.3 Optimizing Block2-type Data Blocks

Block2-type data blocks can be constrained through the "clock" constraint by:

- 1. setting a clock on a node in the monotone combinational transitive fanin of the SEs delimiting the block.
- 2. setting clock dependent external input delay on the external input connectors delimiting the block.

In this case, it would be something as:

```
set clock clk -period 20
set externalDelay -in -max 0 I3 I4 I5 -clock clk
```

NOTE that as you had to specify the a clock, this will ALSO constrain a block3-type block delimited by SEs in the monotone transitive fanout of the clock node.

Block2-type data blocks can be constrained through the "pathCluster"constraint by:

- 1. creating a pathCLuster
- 2. setting arrivalTime on external input connectors delimiting the block;
- 3. setting requiredTime on data input connector of SE delimiting the block;

In this case, it would be something as:

```
set pathCluster pc1
set arrivalTime I3 I4 I5 0  -pathCluster pc1
set requiredTime 20 allSE(clk).dataInput() -setup -pathCluster pc1
```

2.1.11.4 Optimizing Block4-type Data Blocks

Block4-type data blocks can be constrained through the "clock" constraint by:

- 1. setting a clock on a node in the monotone combinational transitive fanin of the SEs delimiting the block.
- 2. setting clock dependent external output delay on the external output connectors delimiting the block.

In this case, it would be something as:

```
set clock clk -period 20
set externalDelay -out -max 12 O2 O3 -clock clk
```

NOTE that as you had to specify the a clock, this will ALSO constrain a block3-type block delimited by SEs in the monotone transitive fanout of the clock node.

Block4-type data blocks can be constrained through the "pathCluster" constraint by:

- 1. creating a pathCLuster
- 2. setting requiredTime on external output connectors delimiting the block;
- 3. setting arrivalTime on data output connector of SE delimiting the block;

In this case, it would be something as:

```
set pathCluster pc1
set arrivalTime 0 allSE(clk).dataOutput -addClockPath -pathCluster pc1
set requiredTime 8 O2 O3  -pathCluster pc1
```

2.1.11.5 Block124-type Data Blocks

Block124-type data blocks can be constrained through the "clock" constraint by:

- 1. setting a clock on a node in the monotone combinational transitive fanin of the SEs delimiting the block.
- 2. setting clock dependent external output delay on the external output connectors delimiting the block.
- 3. setting clock dependent external input delay on the external input connectors delimiting the block.

In this case, it would be something as:

```
set clock clk2 -period 20
set externalDelay -out -max 12   O4 O5 O6 -clock clk2
set externalDelay -in -max 0   I8 I9 I7 I6 -clock clk2
```

Note that as you had to specify a clock, this will ALSO constrain a block3-type block delimited by SEs in the monotone transitive fanout of the clock node;

Block4-type data blocks can be constrained through the "pathCluster" constraint by:

- 1. creating a pathCLuster
- 2. setting requiredTime on external output connectors delimiting the block;
- 3. setting arrivalTime on data output connector of SE delimiting the block;
- 4. setting arrivalTime on external input connectors delimiting the block;
- 5. setting requiredTime on data input connector of SE delimiting the block;

In this case, it would be something as:

```
set pathCluster pc1
set arrivalTime 0 -addClockPath allSE(clk2).dataOutput -pathCluster pc1
set requiredTime 8 O2 O3  -pathCluster pc1
set requiredTime 20 -setup allSE(clk2).dataInput  -pathCluster pc1
set arrivalTime 0 I8 I9 I7 I6  -pathCluster pc1
```

2.1.11.6 Optimizing Block1-type Data Blocks

Block1-type data blocks can be constrained through the "clock" constraint by one of the two following methods:

- 1. re-use a "real" existing clock;
- 2. set a "virtual" clock; by virtual we mean that the specified clockName node does not exist;

If you want to re-use a "real" clock (already set), you have to:

- 1. set clock dependent external output delay on the external output connectors delimiting the block.
- 2. set clock dependent external input delay on the external input connectors delimiting the block.

In this case, it would be something as:

```
set clock clk -period 20
set externalDelay -out -max 14  O1 -clock clk
set externalDelay -in -max 0   I1 I2 -clock clk
```

Note that as you had to specify a clock, this will ALSO constrain a block3-type block delimited by SEs in the monotone transitive fanout of the clock node;
If you want to use a virtual clock, you have to:

- 1. create a virtual clock;
- 2. set clock dependent external output delay on the external output connectors delimiting the block.
- 3. set clock dependent external input delay on the external input connectors delimiting the block.

In this case, it would be something as:

```
set clock vclk -period 20
set externalDelay -out -max 14  O1  -clock vclk
set externalDelay -in -max 0   I1 I2  -clock vclk
```

Note that as the "vclk" node does not exist, it is certainly not in the monotone transitive fanin of any SE, and thus no block3-type block will be accidentaly constrained.

Block1-type data blocks can be constrained through the "pathCluster" constraint by:

- 1. creating a pathCLuster
- 2. setting requiredTime on external output connectors delimiting the block;
- 3. setting arrivalTime on external input connectors delimiting the block;

In this case, it would be something as:

```
set pathCluster pc1
```

```
set requiredTime 6 O1 O0 -pathCluster pc1
set arrivalTime 0 I1 I2 -pathCluster pc1
```

2.1.12 About Bidirectional Connectors

Assuming that inthe here-above design, the connector "O0" is bidectional, you can specify both type of delay or both type of time on it but on two separate timing constraints. For instance, you can do the following:

```
set pathCluster pc1
set arrivalTime 0 I1 I2  -pathCluster pc1
set requiredTime 6 O1 O0  -pathCluster pc1
set pathCluster pc2
set arrivalTime 2 O0  -pathCluster pc2
set requiredTime 6 O1  -pathCluster pc2
```

2.1.13 Multi-Phase Design or Optimizing Block5-type Data Block

Block5-type data block can be constrained through grouping multiple clock constraints as follows:

- 1. setting clocks on nodes in the monotone combinational transitive fanin of the SEs delimiting the block;
- 2. grouping these clocks.

In the present case, it could be something as:

```
set clock clk2 -period <P2> -waveForm (RE2,FE2)
set clock clk3 -period <P3> -waveForm (RE3,FE3)
set groupClock gr1 -master clk2 -slave clk3
set groupClock gr2 -master clk3 -slave clk2
```

2.1.14 Some Guidelines to Help the Designer to Specify Timing Constraints

Here are some guidelines to properly use the COMPASS timing constraints:

- 1. use pathCluster for "control" block such as reset logic block or clock block because it is impossible to constrain these blocks with clock constraints;
- 2. use pathCluster for purely combinational design if you reall do not have any clock specification in mind;
- 3. use clock constraint as soon as you have clock specification for your design;
- 4. use virtual clock when the block of logic you are optimizing is coming from a big design and is thus virtually surrounded by flops clocked on a specified clock;

2.2 How Does CTAE Fit Into Any Other Potential Client

CTAE has a complete procedural interface. You provide him with a design under a pseudo-flat cmp data structure [4], you give him constraints, you ask him a timing analysis, you ask him to trace paths, you ask him any specific information about the timing on one node and so on.

2.2.1 Definitions

Roughly speaking, CTAE is composed of two main things:

- a list of delay computers;
- a tracer/checker

A **delay computer** is an engine in charge of computing all the pin-to-pin timings of all the instances in a design. According to the type of leaf cell library you are using, you may have one or many delay computer available. When there are more than one delay computer available, the user has the possibility to select any of them. A delay computer is allergic to:

-- so-called environment parametrs such as the drive on the input connectors, or the temperature, the VDD level;

-- some design specific parameters such the wire load on some connectors.

As a matter of fact, each time some of these parameters are changed, the pin-to-pin timing of the ocmplete design has to be re-computed.

A **tracer/checker** is an engine in charge of propagating the signal timing along the design and to check that some specified timing constraints are met or not met; A tracer/checker uses thus as information:

-- gate pin-to-pin timing computed by the delay computer;

-- timing constraints specified by the user and which should be checked;

-- some parameters such as the drives on the input connectors.

2.2.2 General architecture

The general architecture of the CTAE is presented on the following figure (see Fig. 1.7)

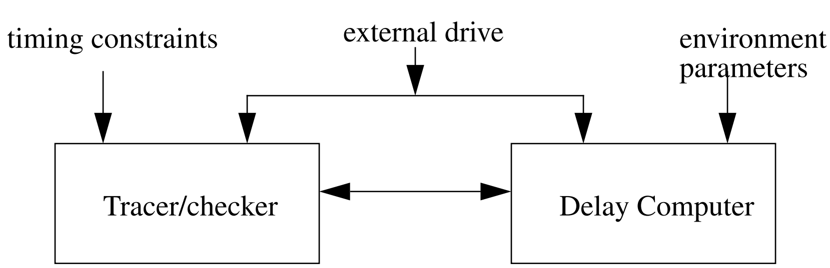

Fig. 1.7: CTAE Overview.

Chapter3

Specifying Displaying and Modifying4

The "set argument" command will be used to specify something. The "show argument" command will be used to display something; when the number of lines to be displayed is bigger than 32, the user will be prompted for the number of lines he does want to really see. This following command lets you modify that limit:

```
set LINELIMIT <n> | unlimited
```

The following command lets you view the actual limit:

```
show LINELIMIT
```

Something specified through the use of the "set" command will be either:

- 1. overwritten by re-using the "set" command;
- 2. removed by using the clear command;

The current AS way of displaying and clearing constraints will be modified to allow constraint overwritting. Also, all commands (except the timing constraint commands) will accept as optional argument a name of cell to tell AS that the the pointed element is not to be reached from the top level cell of your design (default) but the specified cell.

CTAE automatically sends message when you invoke him with bad information, such a non-existing connector name. But we advise the ctae client to filter as much as possible these bad information.

4.1 CTAE Environment Parameters SubMenu

The following parameters will be available in a sub menu and will be set before loading anything (vhdl/nls/...)

- 1. derating, temperature, vddlevel, process;
- 2. delay computer selectiont;

- 3. value of predcap in case the fdm has a internal predcap model;
- 4. the drive on the external input connector;

Also, you can use the environment sub-menu to load an environment file "[env]design" where are specified the previoulsy listed parameters.

4.1.1 Setting Derating Factor

You can set derating factor with the following command:

```
set deratingFactor [-calculated] | [-direct <value>]
```

In case you set it to direct value you can specify the derating factor value as a real positive smaller than **maxAllowedDeratingFactor**.
In case you set it to - calculated, you will have the possibility to specify the different paramaters:

- 1. temperature;
- 2. process;
- 3. Vdd level.

The temperature is set with the following command:

```
set temperature <value>
```

where <value> is a real belonging to [minAllowedTemperature, **maxAllowedTemperature**]. The default is 69.9 celsius.
The process is set with the following command:

```
set process <slow/typical/fast>
```

where the default is slow. The Vdd level is set with teh following command:

```
set vddLevel <value>
```

where <value> is a real positive belonging to [**minAllowedVddLevel**, **maxAllowedVddLevel**]. The default is 4.8 v.

4.1.2 Selecting the Delay Computer

Some libraries have only a prop/ramp timing in the pmd and do not have any delay computer. Some libraries only have a delay computer and do not have any prop/ramp delay in the pmd. Finally, some libraries do have both. The following command lets you specify the delay computer you want to use

```
set timingModel <timingModelName>
```

4.1.3 Setting the drive

You can set the drive in the environment menu with the help of the following commands:

```
set driveRise <value> <connName>
set driveFall <value> connName
```

4.1.4 Setting the predcap

In case the delay computer has its own linear predcap model, you can set the delay computer predcap with the help of the following command:
set predcap <inccap> <defcap>

4.2 Setting the Tag

You set a tag with the following command:

```
set tag <logicTag> [<padTag>]
```

By default the first tag on your search path will be proposed to you.

4.3 Connector Types

Most of CTAE settings accept "connectors" as argument. There are different types of connectors, and each setting only accepts some specific types of connectors as argument. Here is an exhaustive list of all the connector types:

- 1. TopCell External Input Connectors (EIC)
- 2. TopCell External Output Connectors (EOC)
- 3. Sequential Element Data Input Connectors (SEDIC)
- 4. Sequential Element Clock Connectors (SECC)
- 5. Sequential Element Reset/Set Connectors (SERC)
- 6. Sequential Element Data Output Connectors (SEDOC)
- 7. Broken Path Input Connectors (BPIC)
- 8. Broken Path Output Connectors (BPOC)
- 9. Black Box Input Connectors (BBIC)

- 10. Black Box Output Connectors (BBOC)

Topcell bidir connector are belonging to both (EIC) and (EOC);

4.4 Automatic Aliases

A lot of CTAE command accepts as argument a node name. These node can be in the deep hierarchy, and thus can have very long name when addressed by their hierarchical name. As within CTAE all cell should be isolated, in some case it will be easier to address a node by its name and the cell to which it belongs rather than by its hierarchical name. This is why we introduce here what we call a hierarchical instance automatic alias, which is nothing else than the name of the cell the instance is an instance of preeceded by a "$". More formally for hierarchical cell, "$cellName" is an alias of the hierarchical name of an instance as soon as this instance is an instance of the cell "cellName". As an example, assuming that the sting "u1.u2.u4.u7.u9" is the name of an instance of the cell "subcell4", then "$subcell4" is an alias of the string "u1.u2.u4.u7.u9".
Node name are then constructed by suffixing the node name to the name of the cell alias; As an example, the node clk of the same instance is accessed through "u1.u2.u4.u7.u9.clk" or "$subcell4.clk".

4.5 Wild Card

The "*" wild card is accepted in almost all commands and are digested by the CTAE kernel. When refering to connectors, wild cards are expanded to connectors matching the wild card AND having the right type.
Some command argument are expecting only one node. In this case, if the user is using wild card, CTAE will reject the setting as soon as the wild card string can be matched by more than one node. A message will be sent to the user to alert him. This message will give a list of succesfull pattern matches.

4.6 Macros

4.6.1   Introduction

Macros are a way to directly points to a group of design component respecting some criteria, such as "all external input connectors" or "all sequential element". Macros were already existing in the "old" QTV; for instance you could point to all external inputs with the ".inputs" macro. You could point to all sequential element with the ".se" macro. Using "." is error-prone because it is already used as hierarchy delimiter or as a connector delimiter. Indeed, assuming that we have a ".dataInput" macro to point to all the data input of a gate, one would have to write the following to point to all data input of all sequential element:

```
.allse..dataInput
```

while:

```
.allse.dataInput
```

would only match the SE connector whose name really is "dataInput".

We thus propose to name macro with a string appended with braces. Something as "macro()". A direct advantage of working like that is that we can easily extand the syntax by adding some optional argument between the braces in order to restrict the meaning of the macro.

4.6.2 Refering to All Sequential Element

All sequential element can be touched with the following keyword:

```
allSE([<clockNodeName>])
```

where [<clockNodeName>] is optional and would restrict the list to the SE in the transitive fanout of the "clockNodeName". No wild cards are allowed for "clockNodeName".

4.6.3 Refering to All FLip-FLops

All flip-flop can be touched with the following keyword:

```
allff([<clockNodeName>])
```

where [<clockNodeName>] is optional and would restrict the list to the FF in the transitive fanout of the "clockNodeName". No wild cards are allowed for "clockNodeName".

4.6.4 Refering to All Latches

All latch can be touched with the following keyword:

```
alllatch([<clockNodeName>])
```

where [<clockNodeName>] is optional and would restrict the list to the Latches in the transitive fanout of the "clockNodeName". No wild cards are allowed for "clockNodeName".

4.6.5 Refering to Sequential Element Connectors

SE connectors can be reached with the following keyword:

```
dataInput()
clock()
dataOutput()
setReset()
testMode()
scanIn()
scanOut()
scanClock()
```

Note that :

- 1. the connector attributes are coming from the pmd and the pmd system;
- 2. some connector may be matched by more than one macro;

4.6.6 Refering to External Connectors

All external input connectors can be reached with the following macro:

```
allInputs([<clockNodeName>])
```

where [<clockNodeName>] is optional and would restrict the list to the EICs in the transitive fanin of SE clocked on monotone transitive fanout of "clockNodeName". No wild cards are allowed for "clockNodeName".

All external output connectors can be reached with the following macro:

```
allOutputs([<clockNodeName>])
```

where [<clockNodeName>] is optional and would restrict the list to the EOCs in the transitive fanout of SE clocked on monotone transitive fanout of "clockNodeName". No wild cards are allowed for "clockNodeName".

4.6.7 Displaying List of available Macros

The list of availbale macros can be displayed by using the folloiwng command:

```
show macro
```

4.6.8 Example of Use of Macros

Here some example of use of macro:

```
set combinationalPath allSe() -in set() -out dataOutput()
set pathcluster pc1
set arrivalTime -max 0 allinputs() -pathcluster pc1
set requiredtime -max 10 alloutputs() -pathcluster pc1
```

4.7 Legal Node Names

The legal node names are:

1. stringWithWildCards;
2. macro;
3. macro.stringWithWildCards;
4. macro.macro;
5. alias.stringWithWildCards;
6. alias.allInputs;
7. alias.allOutputs;

4.7.1 Viewing Nodes Matching a Legal Node Name

You can view the list of nodes matching a legal node name by using the following command:

```
show node <legalNodeNames>
```

4.7.2 Viewing Synonyms of Nodes Matching a Legal Node Name

You can view synonyms of nodes matching a legal node name by using the following command:

```
show synonyms <legalNodeName>
```

4.8 Legal Instance Names

The legal instance names are:

1. stringWithWildCards;
2. macro;
3. alias.stringWithWildCards;

4.8.1 Viewing Instances Matching a Legal Instance Name

You can view the list of instances matching a legal node name by using the following command:

```
show instance <legalInstanceNames>
```

4.9 External Input Connector Drives

4.9.1 Specifying the Drives of the External Input Connector

You can set the drive on the external input connectors through the environment sub-menu or directly to the tracer. The way you specify them does not matter. They are anyway store din the same unique data strcuture.

By default, the external drives are considered to be zero ns/pf. The main reasons for this are:

1. it helps when doing blind benchmarks;
2. it is the default in QTV.

The value of the drives of the external input connectors is specified with the following command:

```
set DRIVE [-rising|-falling] (<ns/pf> | <expression>) <EICList>
```

where:

- 1. wild cards are allowed;
- 2. <ns/pf> is a drive value specified as an integer or as a real;
- 3. the minimum allowed drive value is zero ns/pf;
- 4. the maximum allowed drive value is maxAllowedDrive;
- 5. <expression> is "n * driveOf(leafCellName)" where n is an integer;
- 6. you can only specify drive for external input connectors ;

An example of use is:

```
CTAEClient> set drive 12 a[1]
  setting drive 12 on connector a[1]
CTAEClient> set drive 3 in*
  setting drive 3 on connectors in1 in2 in3 in4
CTAEClient>
```

where subcella is a black box cell.

4.9.2 Displaying the Drives of the External Input Connector

You can display external connector drive by using the following command:

```
show DRIVE <EICList>
```

Where wild card are allowed. An example of use is:

```
CTAEClient> show drive a[1]
  connector name  rising drive falling drive
      a[1]            12           12
CTAEClient>
```

you can overwrite a connector drive by re-entering its value:

```
CTAEClient> set drive 12 a
CTAEClient> set drive 5 a[1]
CTAEClient> set drive 14 a[2]
CTAEClient> show drive a
```

```
  connector name  rising drive falling drive
      a[1]            5           5
      a[2]           14          14
      a[3]           12          12
      a[4]           12          12
CTAEClient>
```

4.9.3 Clearing the Drives of the External Input Connector

Use the "clear drive" comamnd to clear a drive on a external connector. The syntax is:

```
clear DRIVE [-rising | -falling] <EICList>
```

where:

- 1. wild cards are allowed;
- 2. -rising and -falling are optional. If you specify one of these then only th corresponding drive is cleared. Otherwise both drive values are cleared;
- 3. clearing a drive value means re-setting it to its default value which is 0 ns/pf;

An example is:

```
CTAEClient> clear drive a[2]
```

As the default drive is zero ns/pf, this is strictly equivalent to:

```
CTAEClient> set drive 0 a[2]
```

4.10 Specifying the External Load on External Output Connector

Default external loads are considered to be zero pf. The value of the external load on external connector is specified with the following command:

```
set LOAD <pf> <EOCList >
```

where:

1. wild cards are allowed.
2. <pf> specifies the value of the load, as an integer or as a real;

3. the minimum load value is zero pf;
4. the maximum load value is maxAllowedLoadInPf pf or maxAllowedLoadInFF ff;
5. You can only specify load for external output connectors ;

An example of use is:

```
CTAEClient> set load 2 a
CTAEClient>
```

The default load value is 0pf. The main reasons for that are:

1. it helps when doing blind benchmarks;
2. it is the default in QTV.

The load is displayed by using the following command:

```
show LOAD <EICList>
```

where wild cards are allowed. Reseting a load is overwriting the previous value. You can erase a connector load by either using the "clear load" command or by using the "set load 0 <connName> " command. The syntax of the clear load command is:

clear LOAD <EOCList >

where wild cards are allowed.

4.11 Specifying the External Load on External Input Connector

Default external input loads are considered to be zero pf. The value of the external input load on external connector is specified with the following command:

```
set INPUTLOAD <pf> <EOCList >
```

where:

1. wild cards are allowed.
2. <pf> specifies the value of the inputload, as an integer or as a real;
3. the minimum inputload value is zero pf;
4. the maximum inputload value is maxAllowedLoadInPf pf or maxAllowedLoadInFF ff;

5. You can only specify inputload for external input connectors ; You can NOT specify an inputload on a bidir connector: On such connector, you can only specify an external output load with the help of the “set load “ command.

An example of use is:

```
CTAEClient> set inputload 2 a
CTAEClient>
```

The default inputload value is 0pf. The main reasons for that are:

1. it helps when doing blind benchmarks;
2. it is the default in QTV.

The load is displayed by using the following command:

```
show INPUTLOAD <EICList>
```

where wild cards are allowed. Reseting an inputload is overwriting the previous value. You can erase a connector inputload by either using the "clear inputload" command or by using the "set inputload 0 <connName> " command. The syntax of the clear inputload command is:

clear INPUTLOAD <EOCList >

where wild cards are allowed.

4.12 Overwriting the Default Linear Wiring Capacitance Model

4.12.1 Introduction

By default, CTAE will use as wireCap the values computed by the delay computer. These wireCap values areoverwritten in the following situtations

The value computed from the linear model is overwritten in the following situation:

1. when the user load a [pst]file;
2. when the user load a physical netlist ([nle]file);
3. when the user specifies it with the "set wirecap" command;
4. when the user sets the model of wire to linear model;

4.12.2 Loading a [pst]file

When a [pst]file is loaded together with a netlist, the pst-specified capacitances will be used in place of the one computed from the linear model.

4.12.3 Loading a Physical Netlist

When loading netlists having capacitance already specified on their node, these wiring capacitances will be used rather than the ones computed from the linear model. When loading hierarchical designs composed of logical and physical netlists, ctae will use the physical wiring capacitance where they are set and will use the predictive model where the physical capacitance are not set.
For a hierarchical hybrid net (part is coming from an NLE and part is coming from an NLS), CTAE will use the NLE value for the part of the net in the NLD hierarchy and the predCap model for the part of the net in the NLS hierarchy.

4.12.4 Directly Specifying Node Wire Capacitance.

To correctly address the problem of specifying a node capacitance, we first need to have a look at the following figure (see Fig. 2) which presents a typical testcase:

```
+-----------------------------------------------------------------+
|                                                            |
|      u1                  u2                  u3          |
|   +---------+         +------------+        +---------+    |
|   |         |         |            |        |         |    |
|   |         |         |            |        |         |    |
|   |         |         |            |        |         |    |
|   |  u10    |         |            |        |    u9   |    |
|   |  |\ net1|  net2   |    net3    | net4   |net5|\   |    |
|   |  | +----+---------+------------+--------+----+ +  |    |
|   |  |/z    |out    in|            |out   in|   i|/   |    |
|   |         |         |            |        |         |    |
|   |         |         |            |        |         |    |
|   |         |         |            |        |         |    |
|   +---------+         +------------+        +---------+    |
|                                                            |
+-----------------------------------------------------------------+
```

Fig. 2. Hierarchical wire and hierarchical node.

The set {net1,net2,net3,net4,net5} specifies the same equipotentiel, but as long as the design has to be considered as hierarchical, this set also specifies five different pieces of

the same net. In a context of "physical reoptimization", by assumption all leaf cells have been placed, and the physical tool considers the set {net1,net2,net3,net4,net5} to be one unique net; in that case, the wiring capacitance value specified in a [pst]file produced by the COMPASS chipCompiler will represent the total wiring capacitance of the complete net. In the context of "physical optimization" [5], by assumption only physical hierarchical cells have been placed, and the the physical tool considers the net "net2" to be different from the net "net4"; in that case, COMPASS chipCompiler will specify one wiring capacitance value for the net "net2" and another one for the net "net4". As a matter of fact, when specifying a node wire cap, one needs to be able to tell if the specified value represents the global net cap or only one piece of that net.
This will be done with the following command:

```
set wirecap <pf> <nodeList> [-local [-internal | -external]]
```

where:

1. <nodeList> is any node, wild cards are accepted;
2. <pf> psecifies the value of the wire cap as an integer or as a real;
3. the maximum value for <pf> is maxAllowedCap;
4. the minimum value for <pf> is zero pf;
5. [-local] is optional. When set it tells that the specified value only represents the wire cap of the part of the net directly specified;
6. [-internal] is optional. when the specified nodeName is a hierarchical instance pin name, it tells that the specified value applies for the part of the net inside the hierarchical instance.
7. [-externall] is optional. when the specified nodeName is a hierarchical instance pin name, it tells that the specified value applies for the part of the net outsidethe hierarchical instance.

Going back to the previous figure, this means that:

1. "set wireCap 1 u1.u10.z" specifies the wire cap of the global net;
2. "set wireCap 1 u3.u9.i" specifies the wire cap of the global net;
3. "set wireCap 1 u1.out -local" specifies the wire cap of the part "net2" of the total net;
4. "set wireCap 1 u3.in -local" specifies the wire cap of the part "net4" of the total net;
5. "set wireCap 1 u1.u10.z -local" specifies the wire cap of the part "net1" of theglobal net;
6. “set wirecap 1 u2.in -local -external” specifies the wirecap of the part of the net externally connected to u2.in.
7. “set wirecap 1 2.in -local -internal” specifies the wirecap of the part of the net internally connected to u2.in.

Once you specified a cap on part of hierarchical net, you are no more able to set a cap on the global net, you have to continue to specify one by one the cap on the other parts of the net. In other words, the following is not allowed:

```
CTAEClient> set wireCap 1 u3.in -local
CTAEClient> set wireCap 1 u1.u10.z
 Error: you already specified a local cap on the part "u3.in" of that net.
CTAEClient>
```

It is not necessary to specify all the parts of the net. For the part where the wireCap has not been specified, the predictive model will be sued.

4.12.5 Setting PredCap Model to lineal model

The following command set the model of wire cap used by CTAE:

```
set MODELOFWIRE <linear|fdm>
```

The default is FDM. If you let on FDM then the FDM wire capacitance will be used. Otherwise, the one used with the one spcified by this linear model.

4.12.6 Setting Linear PredCAp Values

If you set the modelOfWire to “linear” mode, then the values of this global model are specified with the following command:

```
SET PREDCAPACITANCE {<incCap> <defCap>} | <predfile>
```

These values will be used through all the hierarchy. The ctae will also be able to use a black-box predcap model when this one belongs to the delay computer. In this case, the delay computer will directly annotate the netslist with the wire cap, and the CTAE will use them rather than the predictive model.

4.12.7 Displaying a Node Wire Capacitance

You can display node wire capacitance by using the following command:

```
show wireCap <nodeList> [-local]
```

This will let you see the node cap values used by ctae (set by any of the previous means). CTAE will also let you know if the value is computed from the predictive model or if the value has been set (through "set wirecap command", or pst or nle). [-local] is optional: if you use it, then it will just use the local partof the net.

Wild cards are allowed. An example of use is:

```
CTAECLIENT> set wirecap a 2
CTAECLIENT> set wirecap b 0
CTAECLIENT> set wirecap d 1 -local
CTAECLIENT> show wirecap a b c d
  node name   wire cap     total cap    status
    a            2            2.6       set
    b            0            0.8       set
    c            .87          1.6       predictive model
    d            1.5          2.5       set and predictive
CTAECLIENT>
```

4.13 Specifying Displaying and Clearing RC Delay

4.13.1 Specifying RC Delay

RC delay can be sent to the common engine through the following command:

```
set RCDELAY [-rising | -falling] <ns> <instName.inputConnNameList>
```

Where:

1. the value specifies the "delta-delay" the delta-delay for a fanout pin is the difference between the actual arrival time taking account RCs and the arrival time you would have if all resistances were ignored and you just had a lumped capacitance on the net.
2. <ns> specifies the value of the rcdelay as an integer or as a real;
3. the maximum rcdelay is maxAllowedRCDelay;
4. the minimum rcdelay is zero ns;

These rcdelay are automatically sent to the common engine when [nle]design are loaded.

4.13.2 Displaying RC Delay

RC delay can be displayed by using the following command:

```
show RCDELAY <instName.connName>
```

Wild cards are allowed. "*.*" would match all the wire in your design. An example of use is:

```
CTAECLIENT> show rcdelay u17.a  u17.b
  instance.connector   rising rc delay   falling rc delay
     u17.a                .6                 .4
     u17.b                .4                 .9
CTAECLIENT>
```

4.13.3 Clearing RC Delay

You can clear a rc delay by using the folllowing command:

```
clear rcdelay <instName.connName list>
```

Wild cards are allowed. "*.*" would match all the wire in your design. An example of use is:

```
CTAECLIENT> show rcdelay u17.a  u17.b
  instance.connector   rising rc delay   falling rc delay
     u17.a                .6                 .4
     u17.b                .4                 .9
CTAECLIENT> clear rcdelay u17.a
CTAECLIENT> show rcdelay u17.a  u17.b
  instance.connector   rising rc delay   falling rc delay
     u17.a                .0                 .0
     u17.b                .4                 .9
CTAECLIENT>
```

This is strictly equivalent to directly setting the rc delay to zero ns by using the "set rcdelay" command:

```
CTAECLIENT> show rcdelay u17.a  u17.b
  instance.connector   rising rc delay   falling rc delay
     u17.a                .6                 .4
     u17.b                .4                 .9
CTAECLIENT> set rcdelay u17.a 0
CTAECLIENT> show rcdelay u17.a  u17.b
  instance.connector   rising rc delay   falling rc delay
     u17.a                .0                 .0
     u17.b                .4                 .9
CTAECLIENT>
```

4.14 Computation of the Total Node Capacitance

The total node capacitance is computed as:

```
nodeCap = nodeWireCap + SumOfNodeConnectorCap
```

The nodeWireCap is either specified as explained here-under or computed from the linear model as follows:

nodeWireCap := defCap + numberOfNodeConnector * incCap

where numberOfNodeConnector is the total number of connector (external and internal leaf cell connector and internal black box connector) of the node.
The SumOfNodeConnectorCap is the sum of the capacitance of each connector connected to the node. For internal leaf cell connectors, this capacitance is read from the pmd's , and for external connectors, this capacitance is the value specified by the user through the "set load" or the set inputload command.

4.15 Invoking the Delay Computer

The delay computer is invoked automatically when you want a path trace (see the report timing command). The first thing it checks is the avaibility of the pin-to-pin timings. If these pin-to-pin timings are available, then it does not performs any delay computation. On the opposite is the pin-to-pin timing are not available, then it computes them. You can get the pin-to-pin timing status by using the following command:

show TIMINGSTATUS

You can invoke the delay computer from the user interface by sing the following command:

PREPROCESS

You can clear the pin-to-pin timing by using the following command:

clear timing.

Let's recall that the pin-to-pin timing are automatically cleared as soon as you modify a paramater that may affect one of these pin-to-pin timing (for instance a wirecap, an external drive, the temperature.)

4.16 Timing Constraints

As presented in the introduction, there are four ways to specify timing constraints:

1. define a PATHCLUSTER together with its input arrival times and its output required times;
2. define a CLOCK together with its external input and output delays;
3. define a multi-cycle timing constraint;
4. grouping clock and multicycle timing constraints.

4.16.1 Path Cluster Timing Constraint

To specify path cluster timing constraint, you have to:

1. create a pathcluster;
2. define arrival time on path cluster input node. This specifies both the fact that the node belongs to the pathCluster and the value of the signal arrival time an that node;
3. define required time on path cluster output node. This specifies both the fact that the node belongs to the pathCluster and the value of the signal required time an that node;

Defining arrival time without specifying a cluster is forbidden and flagged as an error.
Defining required time without specifying a cluster is forbidden and flagged as an error.
Defining a pathcluster and some arrival does not create any timing constraints.
Defining a pathcluster and some required does not create any timing constraints.
You really have to specify the pathcluster, some arrival time AND some required time to really constrain some paths.
Each time you will set/clear anything related to a pathCluster constraint, if the results of your setting/clearing makes that there are no more arrivalTime or requiredTime specific to that constraint, a message telling that the pathcluster does not constrain anything will be sent.

4.16.1.1 Defining a Path Cluster

You define a path cluster by uisng the following command:

```
set PATHCLUSTER <pathClusterName>
```

pathClusterName can not be a specified clock name.

An example of use is:

```
CTAEClient> set pathcluster pc1
CTAEClient> set pathcluster pc1
  Warning: pathcluster "pc1" is already existing
CTAEClient> set clock ck -period 10
CTAEClient> set pathcluster ck
  Error: clock ck is already specified
CTAEClient>
```

4.16.1.2 Defining Path Cluster Input Node Arrival Time

Path cluster input node arrival time are specified with the following command:

```
SET ARRIVALTIME [-rising|-falling] [-min|-max] <ns>
  <EICList SEDOCList BPOCList BBOCList> [-addClockPath [<clockName>]]
       [-pathCluster <pathClusterName>]
```

where:

- <ns> specifies a maximum arrival time value as an integer or a real; maximum allowed arrival time is maxAllowedArrivalTime; minimum allowed arrival time is minus maxAllowedArrivalTime;
- [-rising|-falling] specifies that the <ns> delay value is the rising or the falling one. If neither rising nor falling is specified than both the rising and the falling are set to <ns>
- [-max|-min] specifies that the <ns> delay value is the minimum or the maximum one. If neither maximum nor minimum is specified than ONLY the maximum is set to <ns>
- <EICList SEDOCList BPOCList BBOCList> specifies the connector type for which the specified arrival time apply; wild cards are allowed ;
- [-pathCluster <pathClusterName>] specifies the path cluster name. This is optional. If you do not specify it, the default path clsuter will be used in case it has been sepcified. Otherwise, an error message will be sent;
- [-addClockPath [<clockName>]] is optional. It tells that the specified arrivalTime should be completed by the clock to dataOutput time in case the node is driven by a SE data output connector. "<clockName>" is necessary when the SE has more than one clock. No wild card are allowed.

You first have to create a path cluster before defining a path cluster arrival time. The BPOCs and BBOCs are accepted according to the already set black boxes and broken paths. For latches, the SEDOCs are accepted if the latch is in locked mode.

ArrivalTime are displayed with the following command:

```
show ARRIVALTIME <EIC> | <SEDOC> | <BPOC> | <BBOC>
```

where wild cards are allowed. Arrival times are modified by re-using the "set ARRIVALTIME" command and are removed by using the following command:

```
clear ARRIVALTIME <EICList SEDOCList BPOCList BBOCList> [-falling|-
rising] [-min | -max] [-pathCluster <pathClusterName>] [-addClockPath]
```

where wild cards are allowed.

When clearing arrival time , if you just specify the "<EICs SEDOCs BPOCs BBOCs>" then you will clear any arrival time set on these nodes; if you also specify the "pathCluster" then you will just clear the arrival time on these nodes related to that pathCluster. An example of use is:

```
CTAEClient> set arrivalTime -max 4 a[1] -pathCluster pc1
  Error: "pc1" pathCluster not created
CTAEClient> set pathCluster pc1
CTAEClient> set arrivalTime -max a[1] -pathCluster pc1
CTAEClient> show arrivalTime a[1]
  name          pc1 min   pc1 max
  a[1] rising   ------     4
       falling  ------     4



CTAEClient> clear pathCluster  pc1 -force
CTAEClient> show arrivalTime a[1] -pathCluster pc1
  Error: no pathCluster "pc1" defined
CTAEClient> show arrivalTime a[1]
  Error: no arrivalTime time set on node "a[1]"
```

The specified arrival time specifies the time at which the signal arrives on the node. The time at which the signal is set up on the node of the design is thus computed as:

```
arrivalTime + drive * nodeLoad
```

4.16.1.3 Defining Path Cluster Output Node Required Time

Path cluster output node required time are specified with the following command:

```
SET requiredTime [-rising|-falling] [-min|-max] <ns>
           <EOCList SEDICList SECCList SERCList BBICList BPICList>
```

```
[-setup] [-hold] [-pathCluster <pathClusterName>]
```

where:

- <ns> specifies a maximum required time value as an integer or a real positive or negative; maximum allowed required time is maxAllowedRequiredTime; minimum allowed required time is minus maxAllowedRequiredTime;
- [-rising|-falling] specifies that the <ns> delay value is the rising or the falling one. If neither rising nor falling is specified than both the rising and the falling are set to <ns>
- [-max|-min] specifies that the <ns> delay value is the minimum or the maximum one. If neither maximum nor minimum is specified than ONLY the maximum is set to <ns>
- <EOCList SEDICList SECCList SERCList BBICList BPICList> specifies the output connector or internal node for which the specified required time applies; wild cards are allowed ;
- [-pathCluster <pathClusterName>] specifies the path cluster name; This is optional. If you do not specify it, the default path clsuter will be used in case it has been sepcified. Otherwise, an error message will be sent;
- [-setup] specifies that upper-bounded timing constraints should take into account the setup time of the end point.
- [-hold] specifies that lower-bounded timing constraints should take into account the hold time of the end point.

A maximum required time on an output connector is thus defined to be the time by which any change in this output must be complete.
A minimum required time on an output connector is thus defined to be the time before which this output must not change
You first have to create a path cluster before defining a path cluster required time.
The BBICs and BPICs are accepted according to the already set black boxes and broken paths. For latches, The SEDICs are accepted as long as the latch is in locked mode. The SECCs (resp. SERCs) are accepted when the SE clock-to-dataOutput paths (resp. set/reset-to-dataoutput paths) are not set as combinational paths.
RequiredTime are displayed with the following command:

```
show REQUIREDTIME <EOC> | <SEDIC> | <SECC> | <SERC> | <BBIC> | <BPIC>
```

Required times are modified by re-using the "set REQUIREDTIME" command and are removed by using the following command:

```
clear REQUIREDTIME <EOCList SEDICList SECCList SERCList BBICList
BPICList> [-min | -max] [-rising | -falling]
[-pathCluster <pathClusterName>] [-setup] [-hold]
```

When clearing required time , if you just specify the "<EOCs SEDICs SECCs SERCs BBICs BPICs>" then you will clear any required time set on these nodes; if you also specify the "pathCluster" then you will just clear the required time on these nodes related to that pathCluster. An example of use is:

```
CTAEClient> set requiredTime -max 4  o[1] -pathCluster pc1
  Error: "pc1" pathCluster not created
CTAEClient> set pathCluster pc1
CTAEClient> set requiredTime -max 4  o[1] -pathCluster pc1
CTAEClient> show requiredTime o[1]
  name          pc1 min   pc1 max
  o[1] rising   ------     4
       falling  ------     4
CTAEClient> clear pathCluster  pc1 -force
CTAEClient> show requiredTime pc1 -pathCluster o[1]
  Error: no pathCluster "pc1" defined
CTAEClient> show requiredTime o[1]
  Error: no requiredTime time set on node "o[1]"
CTAEClient>
```

4.16.1.4 Viewing a Path Cluster Constraint

You can view a complete pathCluster constraint by using the following command:

```
show constraint <pathClusterName>
```

An example of use is:

```
CTAEClient> set pathCluster pc1
CTAEClient> show constraint foo
  Error: "foo" clock or pathCluster not defined
CTAEClient> set requiredTime -max 4 o[1] -pathCluster pc1
CTAEClient> set arrivalTime -max a[1] -pathCluster pc1
CTAEClient> set arrivalTime -max 5 a[2] -pathCluster pc1
CTAEClient> show constraint pc1
  constraint type: pathCluster
  input connectors:
    name  rMax fMax rMin fMin



    a[1]  3    3    ---- ----
    a[2]  5    5    ---- ----
  output connectors:
    name rMax fMax rMin fMin
    o[1] 4    4    ---- ----
```

```
CTAEClient>
```

4.16.1.5 Clearing a Path Cluster Constraint

You can erase a pathCluster by using the following command:

```
clear PATHCLUSTER <pathClusterName> [-force]
```

This will totally erase the PATHCLUSTER declaration as well as all the arrival times and the required times associated to that cluster as soon as the -force option is set. Otherwise, in case there are arrival time/ required time retaled to that cluster, you will get a message and nothing will be cleared.

```
CTAEClient> set pathCluster pc1
CTAEClient> show constraint foo
  Error: "foo" clock or pathCluster not defined
CTAEClient> set requiredTime -max 4 o[1] -pathCluster pc1
CTAEClient> set arrivalTime -max 3 a[1] -pathCluster pc1
CTAEClient> set arrivalTime -max 5 a[2] -pathCluster pc1
CTAEClient> show constraint pc1
  constraint type: pathCluster
  input connectors:
    name  rMax fMax rMin fMin
    a[1]  3    3    ---- ----
    a[2]  5    5    ---- ----
  output connectors:
    name rMax fMax rMin fMin
    o[1] 4    4    ---- ----
CTAEClient> clear pathCluster pc1 -force
CTAEClient> show constraint pc1
  Error: "pc1" clock or pathCluster not defined
CTAEClient>
```

4.16.2 Clock Timing Constraint

To specify clock-derived timing constraint, you have to:

1. create a clock: this will constrain path starting and ending on SE in the monotone combinational transitive fanout of the created clock; This means that ctae will cross combinational logic as long as the clock network is monotone. (In orther words, XOR behavior is forbidden in the clock network). Non valid clock will be flagged by the CTAE.
2. define clock dependent external input delays. This is optional. This will constrain

path starting on the so defined external input connector and ending on SE in the monotone combinational transitive fanout of the created clock;

3. define clock dependent external output delays. This is optional. This will constrain path starting on SE in the monotone combinational transitive fanout of the created clock and ending on the so defined external output connector;

4.16.2.1 Creating Clocks

Clocks are created with the following command:

```
set clock <clockName> -period <ns> [-waveForm
(<risingEdge>,<fallingEdge>)]
    [-actualLatency] | [[-latency [<ns>] | [[-risingLatency] <ns>]
     [-fallingLatency <ns>]]] [-skew  <ns>]
```

where:

1. "<clockName>" is the name of the clock signal. This clock signal is either an external connector or an internal net or a non existing node. In this last case, a message is sent to recall the user that it is a virtual clock; No wild card are allowed.
2. "-period <ns>" specifies the clock period in nanoseconds. It is a real and the unit of resolution is the picosecond;
3. the maximum allowed value for a clock period is maxAllowedPeriod;
4. the minimum allowed value for a clock period is 1 ns;
5. -waveForm (<risingEdge>,<fallingEdge>) specifies the clock rising and falling edges. This waveform specification is optional. By default, the clock is assumed to be high the first half period and low the second half period. Rising and falling values should be different, and none of them should be bigger than clock period. Otherwise ctae immediately flags it as an error and rejects it.
6. "[-actualLatency]" specifies that the timing engine should take into account the clock network timing when computing arrival time (resp. required time) on SE data output (resp. SE data input). By default the timing engine does not take the actual clock network timing into account when computing these arrival and required time. This clock attribute is optional and uncompatible with the next one.
7. "[[-latency [<ns>] | [[-rising] <ns>] [-falling <ns>]]]" specifies the latency to be taken into account by the engine when the actual latency should be ignored. If you just specify one value in nanoseconds, both rising and falling latency are assumed to have this value. In this case, only the actual phase is taken into account. If you just specify the rising value (resp. the falling value) the falling value (resp the rising value) is considered to be zero ns. This clock attribute is optional and uncompatible with the previous one. By default, when the engine does not take into account the actual latency, it considers that the latency is zero ns unless you specified its value

with this attribute.

8. If neither the "-actualLatency" option nor the "latency" value is specified, then the clock is considered to be "ideal".
9. "[-skew [<ns>] is the expected post-layout skew budget you allocate. By default the skew budget is considered to be zero ns. The skew value is limited to the clock period. More you allocate skew is, more the upper-bounded constrained blocks will be over-optimized and more lower-bounded constrained blocks will contain buffer chain to fix hold time violations.

The following figure (see Fig. 3) presents a clock and how to declare it with our new syntax.

```
Reference point
 for any clock

   /|\
    |    +--------+           +--------+
    |    |        |           |        |
 ---+----+        +-----------+        +-----------
    |    5        13     18
    |
    |<---> risingEdge
    |<-----------> fallingEdge
    |<------------------> period
```

set clock clk -period 18 -rising 5 -falling 13

Fig. 3. A clock declaration example

The clock edge computation for setup check in a non-inverting clock network are:

```
risingEdge  = clockRisingEdge + risingLatency - Skew
fallingEdge = clockFallingEdge + fallingLatency - Skew
```

The clock edge computation for setup check in a inverting clock network are:

```
risingEdge  = clockFallingEdge + risingLatency - Skew
fallingEdge = clockRisingEdge + fallingLatency - Skew
```

The clock edge computation for hold check in a non-inverting clock network are:

```
risingEdge  = clockRisingEdge + risingLatency + Skew
fallingEdge = clockFallingEdge + fallingLatency + Skew
```

The clock edge computation for hold check in a inverting clock network are:

```
risingEdge  = clockFallingEdge + risingLatency + Skew
fallingEdge = clockRisingEdge + fallingLatency + Skew
```

Clock are modified by re-using the "set clock" command and are displayed by using the following command

```
show CLOCK <clockNameList>
```

where wild cards are allowed. An example of use is:

```
CTAEClient> set clock ck -period 20 -actuallatency
CTAEClient> show clock ck
  name  period  rising falling latency skew
   ck   20      0      10      actual  0
CTAEClient> set clock ck -period 30 -actuallatency
CTAEClient> show clock ck
  name  period  rising falling latency skew
   ck   30      0      15      actual  0
CTAEClient> set clock ck -period 10 -latency -rising 1 -falling .2
CTAEClient> show clock ck
  name  period  rising falling latency  rising  falling  skew
   ck   10      0      5       specified  1       .2       0
CTAEClient>
```

4.16.2.2 Sequential Element Hit by a Declared Clock

The SE hit by a declared clock are the SE on the "MONOTONE COMBINATIONAL TRANSITIVE FANOUT" of a declared clock. The list of this SE are displayed by the "show clockTree" command.

4.16.2.3 Defining External Input Delays

External Input delays are specified with the following command:

```
SET EXTERNALDELAY [-in] [-rising|-falling] [-max | -min] <ns>
               <EICList> [-fallingClock] -clock <clockName>
```

where:

- [-in] specifies that it is a input delay. It is optionalfor uni-directional connector and necessary for bi-directional connector.<ns> specifies a maximum input delay on a input connector, used for the maximum delay check and the setup check;

- <ns> is an integer or a real negative or positive smaller than the period of the-clock; the maximum allowed input delay is maxAllowedInputDelay; the minimum allowed input delay is minus maxAllowedInputDelay;
- [-rising|-falling] specifies that the <ns> delay value is the rising or the falling one. If neither rising nor falling is specified, then both the rising and the falling are set to <ns>
- [-max|-min] specifies that the <ns> delay value is the minimum or the maximum one. If neither maximum nor minimum is specified than both the maximum and the minimum are set to <ns>
- <EICList> specifies the external input connector for which the specified input delays apply; wild cards are allowed; "*" matches all EICs.
- -clock <clockName> tells that the specified input delay are clock relative. Wild cards are not allowed.
- [-fallingClock] specifies that the min and max value are relative to the falling edge of the clock. (rising by default)

You first have to create a clock before defining a external input delay.

externalInputDelays are displayed with the following command:

```
show externalDelay -in <EIC>
```

where wild cards are allowed. The externalInputDelays are modified by re-using the "set externaldelay -in" command and are removed by using the following command:

```
clear externalDelay -in <EICList> [-rising | -falling]
    [-fallingClock | -risingClock][-min | -max] [-clock <clockName>]
```

If only -fallingClock is specified, then only the external delay related to the falling clock edge is deleted. If only -risingClock is specified, then only the external delay related to the rising clock edge is deleted. If none of these is specified, then both are deleted.

An example of use is:

```
CTAEClient> set externalDelay -in -max 4 a[1] -clock ck
  Error: "ck" clock not created
CTAEClient> set clock ck -period 20
CTAEClient> set externalDelay -in -max 4 a[1] -clock ck
CTAEClient> show externalDelay -in a[1]
  name          ck min   ck max
  a[1] rising   ------     4
       falling  ------     4
CTAEClient> show externalDelay -in a[1]
  name          ck min   ck max
```

```
  a[1] rising   ------     4
       falling  ------     4
CTAEClient> clear clock ck -force
CTAEClient> show externalDelay -in a[1]
  no clock defined
CTAEClient>
```

The specified external input delay specifies the time at which the signal arrives on the design. The time at which the signal is set up on the first net of the design is thus computed as:

```
externalInputDelay + drive * nodeLoad
```

4.16.2.4 Defining Clock Dependent Output Delays

Using this common timing analyzis engine, output delays are representing the time used outside the design from an a design output connector to the next flop clocked on the same clock. Output delays on external output connectors are specified with the following command:

```
set EXTERNALDELAY -out  [-rising|-falling] [-min|-max] <ns> <EOCList>
                         [-fallingClock] -clock <clockName>
```

where:

- -out specifies that it is an external OUTPUT delay; This is optional for uni-directional connector and necessary for bi-directional connector.
- <ns> specifies a maximum output delay on a output connector, used for the maximum delay check and the setup check;
- <ns> is an integer or a real negative or positive; the maximum allowed output delay is maxAllowedOutputDelay; the minimum allowed output delay is minus maxAllowedOutputDelay;
- [-rising|-falling] specifies that the <ns> delay value is the rising or the falling one. If neither rising nor falling is specified, then both the rising and the falling are set to <ns>
- [-max|-min] specifies that the <ns> delay value is the minimum or the maximum one. If neither maximum nor minimum is specified than both the maximum and the minimum are set to <ns>
- <EOCList> specifies the external output connector for which the specified input delays apply; wild cards are allowed;
- -clock <clockName> tells that the specified output delay are clock relative. Wild cards are not allowed.

- [-fallingClock] specifies that the min and max value are relative to the falling edge of the clock. (rising by default)

You first have to create a clock before defining a external output delay.

You first have to create a clock before defining a external output delay.
externalOutputDelays are displayed with the following command:

```
show externalDelay -out <EOC>
```

where wild cards are allowed.

The externalOutputDelays are modified by re-using the "set externalDelay -out" command and are removed by using the the following command:

```
clear externalDelay -out <EOCList> [-rising | -falling]
    [-fallingClock | -risingClock] [-min | -max] [-clock <clockName>]
```

If only -fallingClock is specified, then only the external delay related to the falling clock edge is deleted. If only -risingClock is specified, then only the external delay related to the rising clock edge is deleted. If none of these is specified, then both are deleted.
An example of use is:

```
CTAEClient> set externaldelay -out -max 4 a[1] -clock ck
  Error: "ck" clock not created
CTAEClient> set clock ck -period 20
CTAEClient> set externalDelay -out -max 4 a[1] -clock ck
CTAEClient> show externalDelay -out a[1]
  name          ck min   ck max
  a[1] rising   ------     4
       falling  ------     4
CTAEClient> show externalDelay -out a[1]
  name          ck min   ck max
  a[1] rising   ------     4
       falling  ------     4
CTAEClient> clear clock ck -force
CTAEClient> show externalDelay -out a[1]
  no clock defined
CTAEClient>
```

4.16.2.5 Physical Meaning of the External Delays

The following figure (see Fig. 4) helps to understand the physical meaning of the external delays:

```
                 +----------------------------+
                 |                            |
                 |  +-----------------------+ |
              I1 +--+                       | |
                 |  |            B1         +-+ O1
              I2 +--+                       | |
                 |  +-----------------------+ |
                 |                            |
    ff1          |        U1         U2       |          ff2
   +---+  +--+ I3| +--+ +---+ +--+ +---+ +--+ |O2 +--+  +---+
   |   |  |B6+---+-+B2| |   | |B3| |   | |B4+-+---+B5|  |   |
   +D Q+--+  | I4| |  +-+D Q+-+  +-+D Q+-+  | |   |  +--+D Q+
   |   |  |  +---+-+  | |   | |  | |   | |  +-+---+  |  |   |
   | QN+  +--+   | +--+ | QN+ +--+ |   | +--+ |O3 +--+  |   |
 +-+CP |   +-----+-+----+CP |  +---+CP |      |     +---+CP |
 | +---+   |     | |    +---+  |   +---+      |     |   +---+
 |         |     | \-----------/              |     |
 |         |     |                            |     |
 |         |     +----------------------------+     |
 |         |                                        |
 +---------+----------------------------------------+
 |
CLK
```

Fig. 4. A design together with its environment.

External delay are always relative to a clock and a edge.
An external input connector external input delay represents the delay of the path starting on a SE in the transitive fanin of the connector and clocked on the specified clock and ending on the external input connector.
A Maximum external input delay represents the maximum time at which a signal is available on an external input connector. This delay is relative to the rising edge of the clock by default (except if -fallingClock is specified together with the external input delay specification).
In the here above example, assuming that the ff1 clock to dataOutput time is 1.2 ns and that the maximum pin-to-pin timing of block b6 is 2ns, then maximum external input delay is computed as 1.2 + 2ns and specified as:

```
 clk |
     |    +--------+
     |    |        |
     +----+        +------
     |
     |    <--->
           3.2
```

```
set externalDelay -in -max 3.2 I3 -clock clk
```

A Minimum external input delay represents the minimum time at which a signal is available on an external input connector.

In the here above example, assuming that the ff1 clock to dataOutput time is 1.2 ns and that the minimum pin-to-pin timing of block b6 is 0.3, then minimum external input delay is computed as 1.2 + 0.3 and specified as:

```
 clk |
     |    +--------+
     |    |        |
     +----+        +------
     |
     |    |->
          1.5
```

```
set externalDelay -in -min 1.5 I3 -clock clk
```

An external output delay represents the delay of the path starting on an output connector and ending on external SE clocked on the specified clock and on the transitive fanout of this output connector.

Maximum External Output Delays represent the amount of time a signal on an external output connector is required before a clock edge. This amount of time is computed as the sum of the longest path starting on the external output connector and ending on a external SE in the transitive fanout of the connector and the setup time of this SE. This delay is relative to the rising edge of the clock by default (except if -fallingClock is specified together with the external input delay specification).

In the here-above figure, assuming that that ff2 setup time is 1 ns and that B5 external block maximum input-to-output timing is 3 ns, the maximum external output delay on the O2 external output connector is computed as the sum of 3ns + 1ns and thus specified as follows:

```
 clk |
     |      +--------+
     |      |        |
     +------+        +------
     |
     | <---->
        4
```

```
set externalDelay -out -max 4 O2 -clock clk
```

Assuming that the specified "CLK" clock period is 20 ns, this constraint really specifies that the required time on the O2 connector is 20ns - 4ns = 16ns.

Minimum External Output Delays represent the amount of time a signal on an external output connector should be stable before a clock edge. This amount of time is computed as the difference between the shortest path starting on the external output connector and ending on a external SE in the transitive fanout of the connector and the hold time of this SE. If this shortest path is 0 then the minimum external output delay is negative. The way to understand this negative hold time is thus: the signal should be stable minus holdTimeValuebefore clock edge, that plus holdTimeValue after the clock edge.

In the here-above figure, assuming that that ff2 hold time is 1.5 ns and that B5 external block minimum input-to-output timing is .5 ns, the minimum external output delay on the O2 external output connector is computed as the sum of .5ns - 1.5ns and thus specified as follows:

```
 clk |
     |      +--------+
     |      |        |
     +------+        +------
     |
     |    <-|--->
          .5 -1.5
```

```
set externalDelay -out -min -1 O2 -clock clk
```

This really specifies that the signal on the O2 connector should be stable 1 ns after the clock edge (minus one ns before the clock edge).

4.16.2.6 Relation Between External Delays and Arrival/Required Times

Clock constraint is one type of constraint. pathcluster is another type. If you want to translate a clock constraint into a pathcluster, you need to translate the complete clock construct (with all its information clock period - phase) into a pathcluster. Then as soon as it is translated, there are no more clock and thus no more clock cycle, but the information is in the value given to arrival and requiredtime.if you have something as:

```
set clock clk -period 60
set externaldelay -in inconn -max 10
```

```
set externaldelay -in inconn -min 7
set externaldelay -out outconn -max 21
set externaldelay -out outconn -min 8
```

you would translate it into the following pathcluster:

```
set pathcluster pc1
set arrival inconn -max 10          # external input delay
set arrival inconn -min 7           # external input delay
set required outconn -max 39    # clock period - external output delay
set required outconn -min -8        # - external output delay
```

in other words, to achieve the translation, you take into account the clock period/rising edge and so on. In both case there are no min delay violations. On the other hand if you have something as:

```
set clock clk -period 60
set externaldelay -in inconn -max 10
set externaldelay -in inconn -min 0
set externaldelay -out outconn -max 21
set externaldelay -out outconn -min -2
```

you would translate it into the following pathcluster:

```
set pathcluster pc1
set arrival inconn -max 10
set arrival inconn -min 0
set required outconn  -max 39
set required outconn -min 2
```

here in both case, you have a 2ns mindelay violation. In both case you have to add a 2ns delay chain to fix it.

4.16.2.7 Viewing a Clock Timing Constraint

You can display a complete clock timing constraint by using the following command:

```
show constraint <clockName>
```

An example of use is:

```
CTAEClient> set clock ck -period 20
CTAEClient> set externalDelay -out -max 4 o[1] -clock ck
CTAEClient> set externalDelay  -in -min -2 a[2] -clock ck
CTAEClient> set externalDelay  -in -max 6 a[2] -clock ck
```

```
CTAEClient> set externalDelay -in -max 3 a[1] -clock ck
CTAEClient> show constraint foo
  Error: "foo" constraint not defined
CTAEClient> show constraint ck
  constraint type: clock
  clock period: 20
  input connectors:
    name  rMax fMax rMin fMin
    a[1]  3    3    ---- ----
    a[2]  6    6    -2   -2
  output connectors:
    name rMax fMax rMin fMin
    o[1] 4    4    ---- ----
CTAEClient>
```

4.16.2.8 Clearing a Clock Timing Constraint

You can erase a clock by using the following command:

```
clear CLOCK <clockName> -force
```

In case external delay related to that clock have been specified, the user will have a message telling that some delay are specified, except if the -force option is used. An example is:

```
CTAEClient> set clock ck -period 20
CTAEClient> set externalDelay -out -max 4 o[1] -clock ck
CTAEClient> set externalDelay  -in -min -2 a[2] -clock ck
CTAEClient> set externalDelay -in -max 6  a[2] -clock ck
CTAEClient> set externalDelay -in -max 3 a[1] -clock ck
CTAEClient> show constraint foo
  Error: "foo" constraint not defined
CTAEClient> show constraint ck
  constraint type: clock
  clock period: 20
  input connectors:
    name  rMax fMax rMin fMin
    a[1]  3    3    ---- ----
    a[2]  6    6    -2   -2
  output connectors:
    name rMax fMax rMin fMin
    o[1] 4    4    ---- ----
CTAEClient> clear clock ck -force
CTAEClient> show constraint ck
  Error: "ck" constraint not defined
CTAEClient> show externalDelay -in a[1] -clock ck
  Error: "ck" constraint not defined
CTAEClient> show externalDelay -in a[1]
```

```
  Error: no external input delay defined on connector "a[1]"
CTAEClient>
```

4.16.3 Grouping Clock Timing Constraints

Clock timing constraint input nodes and output nodes are data input and output connectors of sequential element clocked on that clock. If your design has more than one clock, creating the different clocks does not automatically constrain paths starting on one FF/input connector clocked on one clock and ending ona FF/output connector clocked on another clock. If you want to constrain such paths, you have to create the diffeent clocks, and then you have to “group” these clocks.

To specify a clock group you have to do the three following things:

1. create a master clock with the help of the set clock command;;

2. create a slave clock with the help of the set clock command

3. Create a clock group with the help set group command

An example of use is:

```
set clock ck1 -period 10
set clock ck2 -period 20
set groupClock gr1 -master ck1 -slave ck2
```

where both clocks should have been declared, otherwise you will get an error message. Grouping two clock timing constraints will really create a new timing constraint whose:

1. input nodes will be data output connectors of SE clocked on the master clock;
2. output nodes will be data input connectors of SE clocked on slave clocks;
3. the arrival time and required times are computed as follows:

This really means that if you only specify the master clock (resp. you only specify some slave clocks) you just specify timing constraint input nodes (resp. you just specify timing constraint output nodes) and thus you do NOT really create a complete timing constraint.

4.16.3.1 Specifying a Clock Group

You create or overwrite a clock group with the following command:

```
set groupClock <clockGroupName>
                  -master <masterClockName>
                  -slave <slaveClockName>
                  [-setup <value> [-start]] | [-hold <value> [-end]]
```

where:

1. "clockGroupName" is the name of the clock group and should be different from any already specified timing constraint name; This is obsolete. If you do not specify it, it is created on the fly, and a message is sent.
2. –master <masterClockName> allows you to specify the groupClock master when creating the group clock. masterClockName should already been creating
3. –slave <slaveClockName> It allows you to specify the groupClock slave when creating the group clock.
4. -setup <value> lets you specify a setup mulitplier in case you do want the default "setup launch edge" and "setup latch edge" automatically detected by the setup checker (see section 2.1.4.3). If you just specify "-setup <value>", then the setup latch edge will be moved forward by "value" slave clock cycle. If you also specify the optional "-start" then the setup launch edge will be moved backward of "value" master clock cycles.
5. -hold <value> lets you specify a hold mulitplier in case you do want the default "hold launch edge" and "hold latch edge" automatically detected by the hold checker (see section 2.1.4.4). If you just specify "-hold <value>", then the hold launch edge will be moved forward by "value" master clock cycle. If you also specify the optional "-end" then the hold latch edge will be moved backward of "value" master clock cycles.
6. -setup and -hold are exclusive: you can not set both of them in one shot: you have to re-edit the clock group if you want to specify both of them.

An example of use is:

CTAEClient> set clock ck1 -period 10

CTAEClient> set clock ck2 -period 20

CTAEClient> set group gr1 -master ck1 -slave ck2 -setup 2

CTAEClient> set group gr1 -hold 1

Note that the secong time we invoked "set group" our goal was just to re-edit the already declared "gr1" group in order to specify its hold multiplier. We thus did not have to re-specify the the master or the slave. If by the way we would have re-specified them, they would have been over-written. Setup and hold multiplier can be-resetted to their default value by doing:

CTAEClient> set group gr1 -master ck1 -slave ck2 -setup 1

CTAEClient> set group gr1 -hold 0

This is totally equivalent of just creating the group clokc as follows:

CTAEClient> set group gr1 -master ck1 -slave ck2

4.16.3.2 Viewing Clock Group

You can view clock groups by using the following command:

```
show groupClock <groupClockName>
```

where wild cards are allowed. As clock group is really a timing constraint, you can also display it with the help of the following command:

```
show constraint <groupClockName>
```

An example of use is:

```
CTAEClient> set groupClock gr1 -master clk1 -slave clk2
CTAEClient> show groupClock gr1
  groupCLockName: gr1
  masterClock: clk1
   slaveCLock: clk2
CTAEClient>
```

4.16.3.3 Clearing Clock Group

You can clear a clock group by using the following command:

```
clear groupClock <groupClockName> [-force]
```

where:

1. `<groupClockName>` specifies the name of the group.Wild cards are allowed. "*" will match all clock group.
2. [-force] is optional; if -force is specified, this will totally delete all the group related entries. If not specified, then it will only clear the group if no master and no slave related to that group has been specified.

4.16.4 Auto-grouping Clocks

The following command:

```
set autogroup [true|false]
```

let's you the possibility to have CTAE automatically create all the possible groups each time you add a new clock. The default is false.

4.16.5 Multi-Cycle Path

In the previous section we presented how we can complete a clock timing constraint by specifying external delays. In this section we explain how we can remove some output nodes from a clock timing constraint or from a group timing constraint in order to create a new timing constraint.

Assuming that in your design some of the flops are delimiting logic which must run at the clock frequency (for instance an adder) and that some other flops are delimiting logic that can run at half the clock frequency (for instance a multiplier), you can first create a clock with the appropriate period. This will constrain the adder and the multiplier to the same timing constraint. You can remove the mulitplier output nodes from the clock timing constraint and create with them a new timing constraint with a double period as required time by using a multi-cycle timing constraint. To create a multi-cycle constraint you have to:

1. specify a multicycle name together with a mutliplier;
2. specify the multicycle inputs;
3. specify the multicycle outputs;

An example is:

```
set clock clk -period 10
set multicycle mc1 -setup 2 -clock clk
set Multicycle mc1 -from ff1.dataoutput() ff2.dataoutput()
set Multicycle mc1 -to ff3.datainput()  ff4.datainput()
```

This will remove the output nodes ff3.datainput() and ff4.datainput() from the clock constraint and create a new timing constraint whose:

1. input nodes are the multiCycle start point (ff1.dataoutput() and ff2.dataoutput())
2. input node arrival time are still clock-to-dataOutut sequential element propagation time;
3. output nodes are the multicycle end point (ff3.datainput() and ff4.datainput());
4. output node required time is the twice the period minus the sequential element setup time;

Note that:

1. this will remove the multicycle end nodes from the clock constraint; This is necessary, because otherwise these end nodes would still be optimized within the clock constraint, yielding thus the multicycle constraint useless.
2. this will NOT remove the multicycle start nodes from the clock constraint; that is the multicycle start nodes will belong to BOTH the multicycle constraint and the clock constraint;
3. you can install the same clock output node in more than one multicycle constraint;

4.16.5.1 Creating a Multicycle

You can create a multicycle with the following command:

```
set MULTICYCLE [-setup <value> [-start]] | [-hold <value> [-end]]
               -clock <clockName> | -groupClock <groupClockName>
               [-from <SEOutputNodeList inConnList>]
               [-to <SEInputputNodeList outConnList>]
                <multiCycleName>
```

where:

1. -clock <clockName> is the name of the multicycled clock; This is one exclusive with the -groupClock <groupClockName>

2. `groupClock <clockName>` is the name of the group clock. This one is mutualy exclusive with the -clock <clockName>

3. -setup <value> lets you specify a setup mulitplier in case you do want the default "setup launch edge" and "setup latch edge" automatically detected by the setup checker (see section 2.1.4.3). If you just specify "-setup <value>", then the setup latch edge will be moved forward by "value" slave clock cycle. If you also specify the optional "-start" then the setup launch edge will be moved backward of "value" master clock cycles.

4. -hold <value> lets you specify a hold mulitplier in case you do want the default "hold launch edge" and "hold latch edge" automatically detected by the hold checker (see section 2.1.4.4). If you just specify "-hold <value>", then the hold launch edge will be moved forward by "value" master clock cycle. If you also specify the optional "-end" then the hold latch edge will be moved backward of "value" master clock cycles.

5. -setup and -hold are exclusive: you can not set both of them in one shot: you have to re-edit the clock group if you want to specify both of them.

6. `[-from <SEOutputNodeList inConnList>]` is optional.It allows you to declare some multicycle start node together with the multi-cycle declaration.

7. `[-to <SEInputputNodeList outConnList>]` is optional.It allows you to declare some multicycle end node together with the multi-cycle declaration.

You can re-edit the multi-cycle by re-entenring the command. Then anything you would re-specify would just overwrite/complete what was already there:

```
CTAEClient> set multicycle mc1 -setup 3 -clock clk
  Error: "clk" clock not existing
CTAEClient> set clock clk -period 20
CTAEClient> set multicycle mc1 -setup 3 -clock clk
CTAEClient> set multicycle mc1 -setup 2 -clock clk
CTAEClient> set clock clk2 -period 10
CTAEClient> set multicycle mc1 -setup 2 -clock clk2
  Error: multicycle "mc1" already defined for clock "clk"
    You first have to delete the multicycle "mc1" with the clear command
CTAEClient>
```

4.16.5.2 Displaying MultiCycle

You can display multicycle with the folowing command:

```
show MULTICYCLE <multiCycleName>
```

An example of use is:

```
CTAEClient> set clock clk -period 20
CTAEClient> set multicycle mc1 -setup 3 -clock clk
CTAEClient> set multicycle mc1 -from ff1.q* ff2.q*
CTAEClient> set multicycle mc1 -to ff1.d ff2.d
CTAEClient> set multicycle mc1 -to ff3.d
CTAEClient> set externalDelay  -out -max 5 out1 -clock clk
CTAEClient> set multicycle mc1 -to out1
CTAEClient> show multicycle mc1
  multicycle: mc1
  clock: clk
  multiplier: 3
CTAEClient>
```

4.16.5.3 Displaying MultiCycle Constraints

You can display multicycle constraint by using the following command:

```
show constraint <multiCycleName>
```

An example of use is:

```
CTAEClient> set clock clk -period 20
CTAEClient> set multicycle mc1 -setup 3 -clock clk
CTAEClient> set multicycle mc1 -from ff1.q* ff2.q*
CTAEClient> set multicycle mc1 -to ff1.d ff2.d
CTAEClient> set multicycle mc1 -to ff3.d
CTAEClient> set externalDelay  -out -max 5 out1 -clock clk
CTAEClient> set multicycle mc1 -to out1
CTAEClient> show constraint mc1
multicycle: mc1
  clock: clk
  multiplier: 3
-to : ff3.d ff1.d ff2.d out1
-from : ff1.q ff2.q ff2.qn ff1.qn
CTAEClient>
```

4.16.5.4 Clearing MultiCycle

You can clear multicycle by using the following command:

```
clear MULTICYCLE <multicycleName> [-from <node>] [-to <node>]
```

where:

1. <multicycleName> accepts wild cards;
2. -from and -to are optional. If do not specify them, then the complete multicycle is removed from the list of constraint. If you specify -from and -to, then only the node belonging to -from -to are removed from the multi-cycles matching <multicycleName> .

4.16.6 Reviewing the "Show Constraint Command"

As stated in the functional summary of this spec (see section "Introduction to COMPASS Timing Constraints" in chapter 2), we provide the user with four types of timing constraints, namely:

1. the path cluster constraint;
2. the clock constraint;
3. the multi-cycle clock constraint;
4. the grouped clock constraint;

As presented through this chapter all these constraint has a specific name. As presented through this chapter of the spec, the "show constraint" command is the general command to display complete timing constraints, with all their settings. The syntax of this command is thus really:

```
show constraint <constraintNameList>
```

where constraintNameList will match any already specified constraint name. The following command will thus display all the timing constraints:

```
show constraint *
```

Together with the timing constraint is also displayed the thrupaths if specified related to that timing constraint.

4.17 Controlling the Path Tracing

The user can control the path tracing with the help of the following concepts:

1. black box;
2. broken path;

3. thru path;
4. bidir propagation;
5. combinational path;
6. transparent latch;

4.17.1 Black Box

Black box is a new feature presented in the ctae_as.fmk [2] specification document. Refer to that spec to understand that feature. In a nutshell a black box is a cell through which timing is not propagated. You usually use black box when:

1. some parts of your design are not yet designed and are thus really empty;
2. you want to focuss on only one part of your design without having to propagate timing on other parts;

4.17.2 Breaking Paths

4.17.2.1 Introduction

You usually break path to cut combinational loop.

4.17.2.2 Setting Broken Paths

Pin-to-pin timing paths can be broken with the following command:

```
set BREAK (<instName> -in <inConnList> -out <outConnList>) | [auto]
```

where:

1. <instName> specifies the name of the instance that you want to break; wild cards are allowed;
2. -in <inConnList> specifies the instance input connectors of the timing arcs you want to break; wild cards are accepted;
3. -out <outConnList> specifies the instance output connectors of the timing arcs you want to break; wild cards are accepted;
4. the command will break any timing arcs starting on a connector specified in the <inConnList> and ending on a connector specified in the <outConnList>;
5. <auto> invoke the automatic combinational loop breaker;

An example of use is:

```
CTAEClient> set break  u4 -in a -out z
CTAEClient>
```

4.17.2.3 Displaying Broken Paths

All broken can be displayed by the following command:

```
show BREAK
```

An example of use is:

```
CTAEClient> set break auto
CTAEClient> show break
  Broken Paths
  u4.a --> u4.Z
  u22.i --> u22.zn
CTAEClient>
```

4.17.2.4 Clearing Broken Paths

Broken paths can be cleared with the following command:

```
clear BREAK <instName> -in <inConnNames> -out <outConnNames>
      [-force]
```

where:

1. <instName> specifies the name of the instance that you want to clear break; wild cards are allowed;
2. -in <inConnList> specifies the instance input connectors of the timing arcs you want to clear break; wild cards are accepted;
3. -out <outConnList> specifies the instance output connectors of the timing arcs you want to break; wild cards are accepted;
4. the command will clear any broken timing arcs starting on a connector specified in the <inConnList> and ending on a connector specified in the <outConnList>;
5. [-force] is used to really clear it even if some arrival/required time are specified on the connector of the broken path.

An example of use is:

```
CTAECLIENT> set break u4 -in a -out z
CTAECLIENT> show break
```

```
  Broken Paths
  u4.a --> u4.Z
CTAEClient> clear break u4 -in a -out z
CTAECLIENT> show break
  No Broken Paths
CTAECLIENT>
```

4.17.2.5 Combinational Loops

Combinational loops are automatically broken by the CTAE when tracing paths. The loops are broken by internally using the "set break" command. The list of CTAE broken paths can be seen by using the "show break" command.

4.17.3 Restricting Paths for a Specific Timing Constraint

4.17.3.1 Introduction

Sometimes, input and output nodes are not sufficient to correctly specify a timing constraint: For instance, in the case of fig. 9, the user would like to specify a different clock frequency for the multiplier than for the adder. The only way to do it is to specify that

- the constraint constraining the multiplier has to “pass thru” the path u1.i2 --> u1.z (and thus NOT pass thru the path u1.i1 --> u1.z)
- the constraint constraining the adder has to “pass thru” the path u1.i1 --> u1.z (and thus NOT pass thru the path u1.i2 --> u1.z)

A thru path is composed of:

1. a set of “fromNode”;
2. a set of “toNode”;
3. a constraint name;

and is defined as:

In other words, a timing arc driving a toNode is broken for the specified constraint as soon as it is not in the transitive fanout of a fromNode.

4.17.3.2 Setting Thru Paths

You can set “thru” path with the help of the following command:

```
set thru [-from <fromNodeNameList>] [-to <todeNameList> ]
               -constraint <constraintName>
```

```
                [-name <thruName>]
```

where:

- `<fromNodeNameList>` specifies a list of nodes. Wild cards are allowed. This is optional.
- `<todeNameList>` specifies a list of nodes. Wild cards are allowed. This is optional.
- `-constraint <constraintName>` specifies the constraint for which the “thru” path applies. The `<constraintName>` should already been declared otherwise you will have an error message. This is optional. You are forced to specify it when you create the thru path. You can omit it when you edit the thru path to add or clear some fromNode or toNode. If you re-specify it with a new constraint name, the old constraint name will be overwritten.
- `[-name <thruName>]` is optional. If you do not specify it, it is created on the fly, and a message is sent to you. The reason of having this “thruName” is that it allows you to add (resp. clear) some fromNode or toNode to a thru path (resp. from a trhu path).

You can specify only fromNodes or only toNodes by using the following command:

```
set FROMTHRU fromNodeNameList -name <thruName>
set TOTHRU todeNameList -name <thruName>
```

The semantic of the “thru” path is described by the following algorithm:

```
for each toNode belonging to toNodeNameList begin
  for each timingArc driving toNode begin
    mark all node in transitive fanin of toNode
    if no fromNode in fromNodeNameList is marked then begin
      break timingArc for the current constraint
    end
  end
end
```

In other words, a timing arc driving a toNode is broken for the specified constraint as soon as it is not in the transitive fanout of a fromNode.
An example of use is:

```
CTAEClient> set thru -from u1.i1 -to u1.z -constraint clk
CTAEClient> set thru -to a[*] -to b[*] -constraint cl2
```

4.17.3.3 Viewing Thru Paths

You can view thru path by using the following command:

show thru [-thruName <thruName>] | [-constraint <constraintName>]

where:

- if you do not specify anything, you will get the list of all the thru paths.
- if you do specify a timing constraint, you will get only the list of thru path related to that constraint.
- if you do specify the thruName, then only that thruName will be displayed.

4.17.3.4 Clearing Thru Paths

You can clear a thru path with the help of the following command:

```
clear thru [-from <fromNodeNameList>] [-to <toNodeNameList>]
                [-thruName <thruName>]
```

where:

- `<fromNodeNameList>` specifies a list of nodes. Wild cards are allowed.
- `<todeNameList>` specifies a list of nodes. Wild cards are allowed.
- `[-thruName <thruName>]` specifies the name of the "thruPath". The `<thruName>` should already been declared otherwise you will have an error message. If you do not specify the <thruName> then it will apply for all the thru paths.

4.17.4 Propagating Through Bidir Pad

By default, time is not propagated through bidir pad. You can specify the tool to propagate time through bidir by using the following command:

```
set PROPAGATETHROUGHBIDIR <EOCs> <true|flase>
```

The following example (see Fig. 5) illustrates the behavior of that command:

```
                         +--------+
                         |        |
in1 >--------------------+        |
                         |        +---> out
out2<--------------------+        |
                         |        |
                         +--------+
```

Fig. 5. Propagating through Bidir Connectors.

If the constraints are the following:

```
set pathcluster pc1
set arrivalTime -max 0 in1 -pathcluster pc1
set requiredtime -max 10 out2 -pathcluster pc1
set pathcluster pc2
set arrivalTime -max 0 out -pathcluster pc2
set requiredtime -max 20 out2 -pathcluster pc2
```

Then NO PATH would be seen for in1 --> out2. A path would be seen for the constraint out --> out2. If the constarints are:

```
set pathcluster pc1
set arrivalTime -max 0 in1 -pathcluster pc1
set requiredtime -max 10 out2 -pathcluster pc1
set pathcluster pc2
set arrivalTime -max 0 out -pathcluster pc2
set requiredtime -max 20 out2 -pathcluster pc2
set propagateThroughBidir out true
```

Then paths would be seen for both constraints.

You can view bidir pads where ctae will propagate time by using the following command:

```
show propagateThroughBidir
```

4.17.5 Setting SE Clock Path and Reset Path As Combinational Path

By default, timing won't be propagated through SE reset paths. By SE clock path we mean the clock-to-dataOutput timing path of a flip-flop and the enable-to-dataOutput timing path of a latch. You can set such paths as combinational paths by using the following command:

```
set COMBINATIONALPATH <SENames>-in <SERC SECC> -out <SEDOC> [-force]
```

where wild cards and macros are allowed . And where <SENames>is a real SE name, allSe(), allLatch(), allFF(). SERC and SECC and SEDOC are either an instance connector name or a connector macro. [-force] is necessary to first clear any constraint hitting the

SE end point. Next section will explain why and how to use this command.

You can view COMBINATIONALPATH by using the following command:

```
show COMBINATIONALPATH
```

You can clear them by using the foolowing command:

```
clear COMBINATIONALPATH <SENames> -in <SERC SECC> -out <SEDOC>
```

4.17.6 Reseting Time Through SE Clock Path

By SE clock path we mean the clock-to-dataOutput timing path of a flip-flop and the enable-to-dataOutput timing path of a latch. The rules about propagating timing through these paths are the following ones:

1. clock path of SE in a transitive fanout of a declared clock are seen as "clocked path" by constraints derived from that clock declaration. That is the arrival time on the SE data ouput connector is computed as the latency on the SE clock connector + the propagation time through the SE clock path.
2. for pathcluster constraint, clock paths NOT SPECIFICALLY SET AS COMBINATIONAL PATH with the help of the "set COMBINATIONALPATH" command are also always seen as "clock paths". As soon as the these paths are set as combinational path, the arrival time on the SE output data connector is computed as the worst arrival time on the SE clock connector + the propagation time though the SE clock path.

The following figure will help to understand these concepts with different experiments (see Fig. 6).

```
                            u1
                         +--------+
                         |        |
in3+---------------------+d      q+--------------------------+out
               U2        |        |
               |\        |        |
 ck+-----------+a\       |        |
      +---+    | z+------+cp      |
in1+--+i o+----+b/       |        |
in2+--+10n|    |/        +--------+
      +---+
       U3
```

Fig. 6. A typical example

With the following set of constraints:

```
set pathCluster pc1
set combinationalPath u1 -in cp -out q
set arrivalTime -max 0 allInputs() -pathCluster pc1
set requiredtime  -max 10 allOutputs() -pathCluster pc1
set clock ck -period 20
set externalDelay -out -max 5 out -clock ck
```

The constraint end points arrival and required time would be:

pathCluster constraint			clock + outdelay constraint		
node	arrival	required	node	arrival	required
ck	0		u1.cp	0	
in1	0				
in2	0				
out		10	out		20 - 5 = 15

Indeed, the SE clock path is seen:

1. as a "combinational path" by the pathCluster constraint;
2. as a "clocked path" by the constraint derived from the clock declaration as this SE is on the combinational monotone transitive fanout of at least one declared clock.

During the optimization, this constraint would imply the following sub-constraints on the two combinational blocks B1 and B2.

	pathCluster constraint			clock + outdelay constraint		
	node	arrival	required	node	arrival	required
B 1	ck	0				
	in1	0				
	in2	0				
	u1.cp		10-(cp->q)			
B 2	u1.q	input->u1.q		u1.q	u1.cp->u1.q	
	out		10	out		20 - 5 = 15

```
+---+----------------------------+-----------------------------------+
```

With the following set of constraints:

```
set pathCluster pc1
set arrivalTime -max 0 allInputs() -pathCluster pc1
set requiredtime  -max 10 alloutputs() -pathCluster pc1
set combinationalPath u1 -in cp -out q
set clock ck -period 20 -actualLatency
set externalDelay -out -max 5 out -clock ck
```

The constraint end points arrival and required time would be:

```
+----------------------------+-----------------------------------+
|pathCluster constraint      |       clock + outdelay constraint |
+----------------------------+-----------------------------------+
|node    arrival   required  |       node    arrival   required  |
+----------------------------+-----------------------------------+
|ck      0                   |       u1.cp   ck-->u2.z           |
|in1     0                   |                                   |
|in2     0                   |                                   |
|out               10        |       out               20 - 5 = 15|
+----------------------------+-----------------------------------+
```

During the optimization, this constraint would imply the following sub-constraints on the two combinational blocks B1 and B2.

```
    +----------------------------+-----------------------------------+
    |pathCluster constraint      |       clock + outdelay constraint |
    +----------------------------+-----------------------------------+
    |node    arrival   required  |       node    arrival   required  |
+---+----------------------------+-----------------------------------+
|   |ck      0                   |                                   |
| B |in1     0                   |                                   |
| 1 |in2     0                   |                                   |
|   |u1.cp             10-(cp->q)|                                   |
+---+----------------------------+-----------------------------------+
|   |u1.q    input->u1.q         |       u1.q    ck->u1.q            |
| B |out               10        |       out               20 - 5 = 15|
| 2 |                            |                                   |
+---+----------------------------+-----------------------------------+
```

With the following set of constraints:

```
set pathCluster pc1
set arrivalTime -max 0 allInputs() -pathCluster pc1
set requiredtime  -max 10 allOutputs() -pathCluster pc1
set combinationalPath u1 -in cp -out q
set clock ck -period 20 -Latency 2
```

```
set externalDelay -out -max 5 out -clock ck
```

The constraint end points arrival and required time would be:

```
+---------------------------+----------------------------------+
|pathCluster constraint     |       clock + outdelay constraint |
+---------------------------+----------------------------------+
|node    arrival   required |       node    arrival   required |
+---------------------------+----------------------------------+
|ck      0                  |       u1.cp   2                  |
|in1     0                  |                                  |
|in2     0                  |                                  |
|out               10       |       out               20 - 5 = 15|
+---------------------------+----------------------------------+
```

During the optimization, this constraint would imply the following sub-constraints on the two combinational blocks B1 and B2.

```
    +---------------------------+----------------------------------+
    |pathCluster constraint     |       clock + outdelay constraint |
    +---------------------------+----------------------------------+
    |node    arrival   required |       node    arrival   required |
+---+---------------------------+----------------------------------+
|   |ck      0                  |                                  |
| B |in1     0                  |                                  |
| 1 |in2     0                  |                                  |
|   |u1.cp             10-(cp->q)|                                 |
+---+---------------------------+----------------------------------+
|   |u1.q    input->u1.q        |       u1.q    2 + u1.cp->u1.q    |
| B |out               10       |       out               20 - 5 = 15|
| 2 |                           |                                  |
+---+---------------------------+----------------------------------+
```

If now, we remove the combinationalPath setting to get the following set of constraints:

```
set pathCluster pc1
set arrivalTime -max 0 allInputs() -pathCluster pc1
set requiredtime  -max 10 allOutputs() -pathCluster pc1
set clock ck -period 20 -Latency 2
set externalDelay -out -max 5 out -clock ck
```

The constraint end points arrival and required time would be:

```
+---------------------------+----------------------------------+
|pathCluster constraint     |       clock + outdelay constraint |
+---------------------------+----------------------------------+
|node    arrival   required |       node    arrival   required |
+---------------------------+----------------------------------+
|ck      0                  |       u1.cp   2                  |
```

```
|in1     0                  |                                  |
|in2     0                  |                                  |
|out               10       |       out               20 - 5 = 15|
+---------------------------+----------------------------------+
```

During the optimization, this constraint would imply the following sub-constraints on the two combinational blocks B1 and B2.

```
    +---------------------------+----------------------------------+
    |pathCluster constraint     |       clock + outdelay constraint  |
    +---------------------------+----------------------------------+
    |node    arrival   required |       node    arrival   required   |
+---+---------------------------+----------------------------------+
|   |ck      0                  |                                  |
| B |in1     0                  |                                  |
| 1 |in2     0                  |                                  |
|   |u1.cp                      |                                  |
+---+---------------------------+----------------------------------+
|   |u1.q                       |       u1.q    2 + u1.cp->u1.q    |
| B |out               10       |       out               20 - 5 = 15|
| 2 |                           |                                  |
+---+---------------------------+----------------------------------+
```

In this case the pathCluster command would not constrain any thing, as the path ck/in1/in2 --> out is not a combinational path. And furthermore, no path would be reported for the pathCluster command as no path really exist between

A last example shows how to optimize all the blocks with only pathCluster constraints:

```
set pathCluster pc1
set arrivalTime -max 0 allInputs() -pathCluster pc1
set arrivalTime -max 0 allse().dataOutput() -addClockPath -pathCluster
pc1
set requiredtime  -max 5 allOutputs() allSe().clock -pathCluster pc1
set requiredtime -max 10 allse().dataInput() -setup -pathCluster pc1
```

The constraint end points arrival and required time would be:

```
+---------------------------+
|pathCluster constraint     |
+---------------------------+
|node    arrival   required |
+---------------------------+
|ck      0                  |
|in1     0                  |
|in2     0                  |
|in3     0                  |
|out               5       |
|u1.q u1.cp->u1.q          |
|u1.cp             5       |
```

```
|u1.d              10 - setup|
+----------------------------+
```

During the optimization, this constraint would imply the following sub-constraints on the two combinational blocks B1 and B2.

```
    +----------------------------+
    |pathCluster constraint      |
    +----------------------------+
    |node    arrival   required  |
+---+----------------------------+
|   |ck      0                   |
| B |in1     0                   |
| 1 |in2     0                   |
|   |u1.cp             5         |
+---+----------------------------+
|   |u1.q 0+u1.cp->u1.q          |
| B |out            5         |
| 2 |                            |
+---+----------------------------+
|B3 |in3     0                   |
|   |u1.d              10 -setup |
+---+----------------------------+
```

4.18 The SKN Example

This section explains how to solve the SKN example with multi-cycle or with path cluster constraints. The SKN original example is composed of one adder and one multiplier surrounded by flops clocked on the same clock "clk" (see Fig. 7). In this example, the multiplier is twice slower than the adder.

```
       +------+    +----------------+       +------+
       |      |    |                |       |      |
       +d    q+-+--+                +-------+d    q+
       |      | |  |  A D D E R     |       |      |
       | FF1  | |  |                |       | FF2  |
clk-+--+cp    | |+-+                |    +--+cp    |
    |  |      | || |                |    |  |      |
    |  +------+ || +----------------+    |  +------+
    |           ||                       |
    +------------------------------------+
    |           ||
    |  +------+ || +----------------+       +------+
    |  |      | || |                |       |      |
    |  +d    q+--+-+                +-------+d    q+
    |  |      | |  |  M U L T I     |       |      |
    |  | FF3  | |  |                |       | FF4  |
    +--+cp    | +--+  P L I E R     |    +--+cp    |
    |  |      |    |                |    |  |      |
    |  +------+    +----------------+    |  +------+
    |                                    |
    +------------------------------------+
```

Fig. 7. The SKN original example.

The correct way to specify the constraint is to use a multicycle construct as follows:

```
set clock clk -period 10
set multicycle mc1 -setup 2 -clock clk
set multicycle mc1 -from ff3.q*
set multicycle mc1 -to ff4.d
```

This will create the following constraint:

constraint name	input nodes	output nodes
clk	ff1.q ff3.q	ff2.d
mc1	ff1.q ff1.q	ff4.d

Note that you can also correctly optimize this design with a pathcluster as follows. This is nevertheless less practical than using the clock and multi-cycle construct.

```
set pathcluster pc1
set arrival ff1.q ff3.q 0 -addclockpath -pathcluster pc1
set required ff2.d 10 -setup  -pathcluster pc1
set pathcluster pc2
set arrival ff1.q ff3.q 0 -addclockpath -pathcluster pc2
set required ff4.d 20 -setup  -pathcluster pc2
```

Fig.8 presents the revisited SKN example where adder and multiplier outputs are driving the data input connectors of the same mux-flip-flop.

```
       +-------+    +-----------------+
       |       |    |                 |
       +d     q+-+--+                 +------+
       |       | |  |  A D D E R      |      |
       | FF1   | |  |                 |      |
clk-+--+cp     | |+-+                 |      |
    |  |       | || |                 |      |
    |  +-------+ || +-----------------+      |
    |            ||                          |
    |            ||                          |
    |            ||                          |
    |  +-------+ || +-----------------+      | +-------+
    |  |       | || |                 |    +-+da       |
    |  +d     q+--+-+                 |      |        q+
    |  |       | |  |  M U L T I      +------+db       |
    |  | FF3   | |  |                 |      | FF4     |
    +--+cp     | +--+  P L I E R      |   +--+cp       |
    |  |       |    |                 |   |  |         |
    |  +-------+    +-----------------+   |  +-------+
    |                                     |
    +-------------------------------------+
```

Fig. 8. The SKN revisited example.

In this case, using the multi-cycle timing constraint, you have to do it like this:

```
set clock clk -period 10
set multicycle mc1 -setup 2 -clock clk
set multicycle mc1 -from ff1.q ff1.qn ff3.q ff3.qn
set multicycle mc1 -to ff4.db
```

This will create the following constraint:

constraint name	input nodes	output nodes
clk	ff1.q	ff4.da

```
|                   |  ff3.q       |                 |
+-------------------+-----------------+-----------------+
|  mc1              |  ff1.q       | ff4.db          |
|                   |  ff1.q       |                 |
+-------------------+-----------------+-----------------+
```

Note that you can also correctly optimize this design with a pathcluster as follows. This is nevertheless less practical then using the clock and multi-cycle construct.

```
set pathcluster pc1
set arrival ff1.q ff3.q 0 -addclockpath -pathcluster pc1
set required ff4.da -setup  -pathcluster pc1
set pathcluster pc2
set arrival ff1.q ff3.q 0 -addclockpath -pathcluster pc2
set required ff4.db 20 -setup  -pathcluster pc2
```

Fig.9 presents the revisited SKN example where adder and multiplier outputs are driving the data input connectors of the same multiplexer.

```
       +------+    +----------------+
       |      |    |                |
       +d    q+-+--+                +--+
       |      | |  |  A D D E R     |  |
       | FF1  | |  |                |  |
clk-+--+cp    | |+-+                |  |
    |  |      | || |                |  |
    |  +------+ || +----------------+  |
    |           ||                     |
    |           ||                     |
    |           ||                     |
    |  +------+ || +----------------+  | +---+
    |  |      | || |                |  | |    \    +----+ +------+
    |  +d    q+--+-+                |  +-+i1   \   |    | |      |
    |  |      | |  |  M U L T I     |    |     Z+--+    +-+d    q+
    |  | FF3  | |  |                |    |  u1  |  |    | |      |
    +--+cp    | +--+  P L I E R     +----+i2   /   +----+ | FF4  |
    |  |      |    |                |    |    /        +--+cp    |
    |  +------+    +----------------+    +---+         |  |      |
    |                                                  |  +------+
    +--------------------------------------------------+
```

Fig. 9. The SKN re-revisited example.

In this case, multiplier (or the adder) path is existing according to the value of the selection bit of the multiplexer. Clearly enough we need a way to give that information to constraint constructs. This is done through the use of the constraint specific broken path in the following way:

1) set clock clk -period 10
2) set mutlicycle mc1 -setup 2 -clock clk
3) set multi mc1 -from ff1.q ff3.q
4) set multi mc1 -to ff4.d
5) set thru -from u1.i2 -to u1.z -constraint mc1
6) set mutlicycle mc2 -setup 1 -clock clk
7) set multi mc2 -from ff1.q ff3.q
8) set multi mc2 -to ff4.d
9) set thru-from u1.i1 -to u1.z -constraint mc2

The explanation are the following ones:

1. first (step 1) we create the clock; this is constraining all the logic between the input node ff1.q, ff3.q and the output node ff4.d to 10 ns.
2. second (step 2, 3 and 4) we create the multicycle for the multiplier;
3. third (step 5) we "thru" the path u1.i2->u1.z to do not constrain the adder with the multiplier multicycle;
4. the multiplier multicycle is unfortunately removing the ff4.d output node from the clock timing constraint (see the definition of the multicycle end points). The direct consequence is that the adder is no more constrained: basically it is driven by the clock constraint input nodes (ff1.q and ff3.q) but does not have anymore output nodes. The fourth phase (step 6 7 and 8) thus consists in creating a multi-cycle for the adder. Note that the multiplier factor of that multicycle is obviously "1";
5. The adder multicycle is unfortunately re-constraining the complete combinational block composed of bith the adder and the multiplier with a one clock period constraint. This is thus killing the effect of the multiplier multicycle; We thus need a fifth phase (step 9) which consists in "thruing" the path u1.i1->u1.z for the multiplier constrain: in this way, the adder multicycle constraint does not constrain the multiplier.

This will create the following constraint:

constraint name	input nodes	output nodes	thru
clk	ff1.q ff3.q		
mc1	ff1.q ff1.q	ff4.d	i2 -> z (u1)

mc2	ff1.q ff1.q	ff4.d	i1 -> z (u2)

You can also constrain this design with a path cluster constraint:

1) set pathcluster pc1
2) set arrival 0 -addclockpath ff1.q ff3.q -pathcluster pc1
3) set required 10 ff4.d -setup -pathcluster pc1
4) set thru -in u1.i1 -out u1.z -constraint pc1
5) set pathcluster pc2
6) set arrival 0 -addclockpath ff1.q ff3.q -pathcluster pc2
7) set required 20 ff4.d -setup -pathcluster pc2
8) set thru -in u1.i2 -out u1.z -constraint pc2

Fig.10 presents the revisited SKN example where adder and multiplier outputs are driving the data input connectors of the same multiplexer whose output directly drives a design external connector.

```
       +------+    +----------------+
       |      |    |                |
       +d    q+--+--+                +--+
       |      |  |  |  A D D E R     |  |
       | FF1  |  |  |                |  |
clk-+--+cp    |  |+-+                |  |
    |  |      |  || |                |  |
    |  +------+  || +----------------+  |
    |            ||                     |
    |            ||                     |
    |            ||                     |
    |  +------+  || +----------------+  | +---+
    |  |      |  || |                |  | |    \
    |  +d    q+--+-+                 |  +-+i1   \
    |  |      |  |  |  M U L T I     |    |      Z+--+ out1
    |  | FF3  |  |  |                |    |  u1  |
    +--+cp    |  +--+  P L I E R     +----+i2   /
       |      |     |                |    |    /
       +------+     +----------------+    +---+
```

Fig. 10. The SKN re-re-revisited example.

This is a variant of previous testcase. Here we really have to specify external delay on the external output connector "out". Using the multicycle construct, we do it in the following way:

1) set clock clk -period 10
2) set externalDelay -out 5 out1
3) set mutlicycle mc1 -setup 2 -clock clk
4) set multi mc1 -from ff1.q ff3.q
5) set multi mc1 -to out1
6) set thru -from u1.i2 -to u1.z -constraint mc1
7) set mutlicycle mc2 -setup 1 -clock clk
8) set multi mc2 -from ff1.q ff3.q
9) set multi mc2 -to out1
10) set thru -from u1.i1 -to u1.z -constraint mc2

The explanation are the following ones:

1. first (step 1) we create the clock; this is not constraining anything in that design.(no flipflop clock on clk in the transitive fanout of the logic).
2. second step (setp 2) we create an external output delay on connector "out1"; this is constraining the complete logic block;
3. third (step 3, 4 and 5) we create the multicycle for the multiplier;
4. fourth (step 6) we thru the path u1.i2->u1.z to do not constrain the adder with the multiplier multicycle;
5. the multiplier multicycle is unfortunately removing the out1 output node from the clock timing constraint (see the definition of the multicycle end points). The direct consequence is that the adder is no more constrained: basically it is driven by the clock constraint input nodes (ff1.q and ff3.q) but does not have anymore output nodes. The fifth phase (step 7, 8 and 9) thus consists in creating a multi-cycle for the adder. Note that the multiplier factor of that multicycle is obviously "1";
6. The adder multicycle is unfortunately re-constraining the complete combinational block composed of both the adder and the multiplier with a one clock period constraint. This is thus killing the effect of the multiplier multicycle; We thus need a sixth phase (step 10) which consists in thruing the path u1.i1->u1.z for the multiplier constrain: in this way, the adder multicycle constraint does not constrain the multiplier.

This will create the following constraint:

constraint name	input nodes	output nodes	thru
clk	ff1.q ff3.q		

mc1	ff1.q	out1	i2 -> z
	ff1.q		(u1)
mc2	ff1.q	out1	i1 -> z
	ff1.q		(u2)

You can also constrain this design with a path cluster constraint:

1) set pathcluster pc1
2) set arrival 0 -addclockpath ff1.q ff3.q -pathcluster pc1
3) set required 5 out1 -pathcluster pc1
4) set thru -from u1.i1 -to u1.z -constraint pc1
5) set pathcluster pc2
6) set arrival 0 -addclockpath ff1.q ff3.q -pathcluster pc2
7) set required 15 out1 -setup -pathcluster pc2
8) set thru -from u1.i2 -to u1.z -constraint pc2

Fig.11 presents the revisited SKN example where adder and multiplier outputs are driving the data input connectors of the same multiplexer whose output directly drives a design external connector and where adder and multiplier input are driven by external input connectors:

```
           +-----------------+
           |                 |
in1  -+--+                 +--+
      |  |  A D D E R      |  |
      |  |                 |  |
      |+-+                 |  |
      || |                 |  |
      || +-----------------+  |
      ||                      |
      ||                      |
      ||                      |
      || +-----------------+  | +---+
      || |                 |  | |    \
in2  --+-+                 |  +-+i1   \
      |  |  M U L T I      |    |      Z+--+ out1
      |  |                 |    |  u1  |
      +--+  P L I E R      +----+i2   /
         |                 |    |    /
         +-----------------+    +---+
```

Fig. 11. The SKN re-re-re-revisited example.

This is a variant of previous testcase. Here we really have to specify external delay on the external output connector "out" AND external input delay on the external input connector. Using the multicycle construact, we do it in the following way:

1) set clock clk -period 10
2) set externalDelay -out 5 out1
3) set externalDelay -in 0 in1
4) set externalDelay -in 0 in2
5) set mutlicycle mc1 -setup 2 -clock clk
6) set multi mc1 -from in1 in2
7) set multi mc1 -to out1
8) set thru -from u1.i2 -to u1.z -constraint mc1
9) set mutlicycle mc2 -setup 1 -clock clk
10) set multi mc1 -from in1 in2
11) set multi mc2 -to out1
12) set thru -from u1.i1 -to u1.z -constraint mc2

The explanation are the following ones:

1. first (step 1) we create the clock; this is not constraining anything in that design.(no flipflop clock on clk in the transitive fanout of the logic).
2. second step (step 2, 3 and 4) we create an external delay on connector "out1" "in1" and "in2"; this is constraining the complete logic block;
3. third (step 5, 6 and 7) we create the multicycle for the multiplier;
4. fourth (step 8) we thru the path u1.i2->u1.z to do not constrain the adder with the multiplier multicycle;
5. the multiplier multicycle is unfortunately removing the out1 output node from the clock timing constraint (see the definition of the multicycle end points). The direct consequence is that the adder is no more constrained: basically it is driven by the clock constraint input nodes (n1 in2) but does not have anymore output nodes. The fifth phase (step 9, 10 and 11) thus consists in creating a multi-cycle for the adder. Note that the multiplier factor of that multicycle is obviously "1";
6. The adder multicycle is unfortunately re-constraining the complete combinational block composed of bith the adder and the multiplier with a one clock period constraint. This is thus killing the effect of the multiplier multicycle; We thus need a sixth phase (step 12) which consists in thruingthe path u1.i1->u1.z for the multiplier constrain: in this way, the adder multicycle constraint does not constrain the multiplier.

This will create the following constraint:

```
+---------------------+--------------+--------------+---------+
```

constraint name	input nodes	output nodes	thru
clk	in1 in2		
mc1	in1 in2	out1	i2 -> z (u1)
mc2	in1 in2	out1	i1 -> z (u2)

You can also constrain this design with a path cluster constraint:

1) set pathcluster pc1
2) set arrival 0 in1 in2 -pathcluster pc1
3) set required 5 out1 -pathcluster pc1
4) set thru -from u1.i1 -to u1.z -constraint pc1
5) set pathcluster pc2
6) set arrival 0 in1 in2 -pathcluster pc2
7) set required 15 out1 -setup -pathcluster pc2
8) set thru -from u1.i2 -to u1.z -constraint pc2

4.19 Specifying THRU Paths From HDL

Starting from the following VHDL description:

```
entity dp is
   port (sel : in bit;
         a,b:  in bit_vector(7 downto 0);
         c:    out bit_vector(15 downto 0)
        );
end dp;

architecture dp of dp is
begin
  c <= extend(a,16) +  extend(b,16) when sel = '0' else a * b;
end dp;
```

which corresponds to the implementation offig. 11, If you want to specify the fact that the multiplier is constrained to 2 * clock period while the adder is constrained to just one clock period, you are in trouble: you have to know the name of the multiplexer instances together with its connector name. Note that it is even worse than that because at the end of the synthesis phase, the "multiplexage" operation is implemented with simple gate (and gates) not with mux gate! And even worse, you have to specify something for each BIT!

Clearly enough, we could be helped if the net driving the multiplexer were named!. The only way to get this is to slightly modify your vhdl description as follows:

```
entity dp is
   port (sel : in bit;
         a,b:  in bit_vector(7 downto 0);
         c:    out bit_vector(15 downto 0)
        );
end dp;

architecture dp of dp is
  signal t1,t2: bit_vector(15 downto 0);
begin
  t1 <= extend(a,16) +  extend(b,16);
  t2 <= a * b;
  c <= t1 when sel = '0' else t2;
end dp;
```

This will force the synthesizer to name the input nodes of the multiplexer with the names “t1” and “t2”. You now constrain your design as follows, without having to worry about the multiplexer names.

```
1)  set clock clk -period 10
2)  set externalDelay -out 5 c
3)  set externalDelay -in  0 a
4)  set externalDelay -in  0 b

5)  set mutlicycle mc1 -setup 2 -clock clk
6)  set multi mc1 -from a b
7)  set multi mc1 -to c
8)  set thru -from t2 -to c -constraint mc1

9)  set mutlicycle mc2 -setup 1 -clock clk
10) set multi mc2 -from a b
11) set multi mc2 -to c
12) set thru -from t1 -to  c -constraint mc2
```

4.19.1 Coming from Verilog module

Coming from a Verilog module, you can play the same trick. Indeed, the following Verilog description corresponding to the implementation of Fig. 11:

```
module test (SEL,A,B,C);
   input SEL;
   input [7:0] A,B;
   output [7:0] C;
   assign     C = (sel == 0) ? A + B :
                                  A - B;
```

```
endmodule
```

can be re-written as:

```
module test (SEL,A,B,C);
   input SEL;
   input [7:0] A,B;
   output [7:0] C;
   wire [7:0] t1,t2;
   assign     t1 = A + B;
   assign     t2 = A - B;
   assign     C = (sel == 0) ? t1:
                                t2;
endmodule
```

4.20 An Example Mixing Multi-Cycle Clocks and Clock Group

In the previous section, when presenting how to specify master clocks as well as save clocks, it has been written that these clocks could be either real declared clock or multi-cycle clock names. This section presents a complete example where clock groups are mixed with multi cycle clocks (see Fig. 11).

```
   +-----------------------------------------------------------+
   |      /---------------------------------------\           |
   |      |                                       |           |
   |      |       U1                   U2         |           |
   |      |      +-----+  +--------+  +-----+     |           |
   |      |      |   R |  | COMBIN |  |   R |     |           |
   |      |      +D   Q+--+        +--+D   Q+--   |           |
   |      |      |     |  | BLOCK12|  |     |     |           |
   |      |      |   QN+  |        |  |     |     |           |
CK2+-------+------+CP   |  |        | ++CP   |     |           |
   |              +-----+  |        | |+-----+     |           |
   |                       |        | +------------+           |
   |               U3      |        |   U4                     |
   |              +-----+  |        |  +-----+                 |
   |              |   R |  |        |  |   R |                 |
   |              +D   Q+--+        +--+D   Q+--               |
   |              |     |  |        |  |     |                 |
   |              |   QN+  +--------+  |     |                 |
   |       +------+CP   |        +-----+CP   |                 |
   |       |      +-----+        |     +-----+                 |
   |       +---------------------/                             |
   |       |       U5                   U6                     |
   |       |      +-----+  +--------+  +-----+                 |
```

```
    |        |       |   R |   | COMBIN |  |   R |                |
    |        |      +D   Q+--+          +--+D   Q+--              |
    |        |       |     |   | BLOCK1 |  |     |                |
    |        |       |   QN+  +---------+  |     |                |
CK1+--------+------+CP    |          |-----+CP    |                |
    |        |      +-----+          |     +-----+                |
    |        \---------------------/                              |
    |                                                             |
    |                                                             |
    +-------------------------------------------------------------+

    Fig. 11. Mixing multi-cycle clock and clcok group.
```

In this example, if the designer only creates the two clocks as follows:

```
CTAEClient> set clock ck1 -period 10
CTAEClient> set clock ck2 -period 20
```

It will automatically create the following constraint input and output nodes:

constraint name	input nodes	output nodes
ck1	u5.q u3.q u5.qn u3.qn	u4.d u6.d
ck2	u1.q u1.qn	u2.d

Assuming that the combinational block12 is very slow compared to combinational block1, it is then necessary to create a multi-cycle constraint like the following ones:

```
CTAEClient> set clock ck1 -period 10
CTAEClient> set clock ck2 -period 20
CTAEClient> set mutlicycle mc1 -setup 2 -clock ck1
CTAEClient> set multi mc1 -from u3.q*
CTAEClient> set multi mc1 -to u4.d
```

This will automatically create the following constraint input and output nodes:

constraint name	input nodes	output nodes
ck1	u5.q u5.qn u3.q u3.qn	u6.d

mc1	u3.q u3.qn	u4.d
ck2	u1.q u1.qn	u2.d

The problem is that now path starting on u3.q and ending on u2.d are still not constrained. The way to properly do it is to create a clock groups between mc1 and ck2 as follows:

```
CTAEClient> set clock ck1 -period 10
CTAEClient> set clock ck2 -period 20
CTAEClient> set mutlicycle mc1 -setup 2 -clock ck1
CTAEClient> set multi mc1 -from u3.q*
CTAEClient> set multi mc1 -to u4.d
CTAEClient> set groupClock gr1 -master ck2 -slave ck1
CTAEClient> set groupClock gr2 -master ck1 -slave ck2
```

This will automatically create the following constraint input and output nodes:

constraint name	input nodes	output nodes
ck1	u5.q u5.qn u3.q u3.qn	u6.d
mc1	u3.q u3.qn	u4.d
ck2	u1.q u1.qn	u2.d
gr1	u1.q u1.qn	u4.d
gr2	u3.q u3.qn	u2.d

4.21 Power of the Set Thru Command

The following example (see fig. 11 bis) illustrates the power of the set thru command:

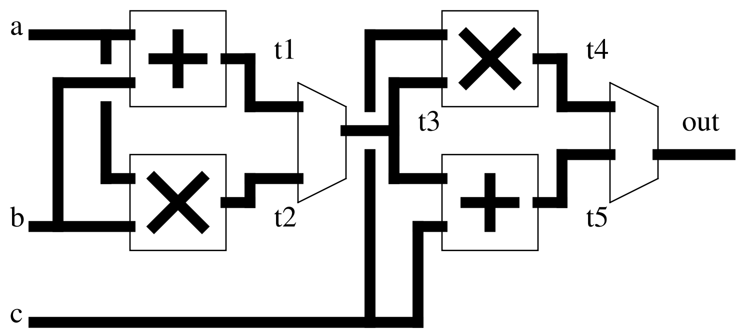

Fig. 11bis. A complex example.

Assuming that the adder needs one clock cycle and the multiplier needs two clock cycles, the correct way to specify this design is:

```
set clock clk -period 10
set externalDelay -out 0 out
set externalDelay -in  0 allInputs()

set mutlicycle mc1 -setup 2 -clock clk -from allInputs() -to out
set thru -from t1 -to t3 -constraint mc1
set thru -from t5 -to out -constraint mc1

set mutlicycle mc2 -setup 3 -clock clk -from allInputs() -to out
set thru -from t2 -to t3 -constraint mc2
set thru -from t5 -to out -constraint mc2

set mutlicycle mc3 -setup 3 -clock clk -from allInputs() -to out
set thru -from t1 -to t3 -constraint mc3
set thru -from t4 -to out -constraint mc3

set mutlicycle mc4 -setup 4-clock clk -from allInputs() -to out
set thru -from t2 -to t3 -constraint mc4
set thru -from t4 -to out -constraint mc4
```

The following example shows how to take the advantage of the “set thru” fom node list (see Fig. 11.ter).

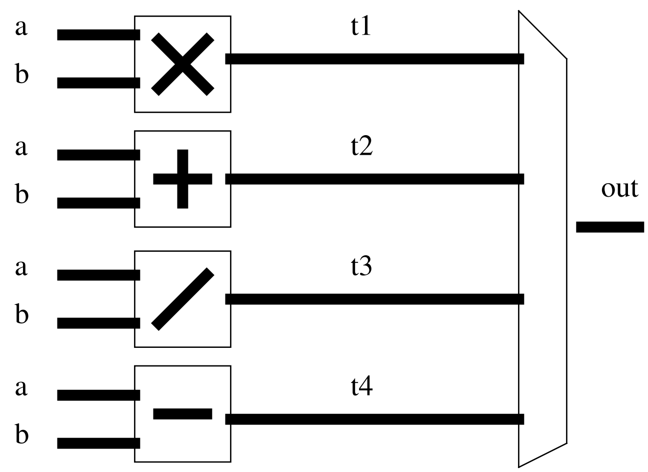

Fig. 11ter. Another complex example.

In this design, both the adder and the substracter are constrained to one clock cycle, while both the multipler and the “divider” are constrained to three clock cycle. The correct way to optimize this design is:

```
set clock clk -period 10
set externalDelay -out 0 out
set externalDelay -in  0 allInputs()

set mutlicycle -name mc1 -mult 3 -clock clk -start allInputs() -end out
set thru -from t1 t3 -to out-constraint mc1

set mutlicycle -name mc2 -mult 1 -clock clk -start allInputs() -end out
set thru -from t2 t4 -to out -constraint mc2
```

4.22 Reporting Timings

Timing reports are obtained by using the following command:

```
REPORT TIMING [-help] [-constraint <constraintName>] [-tracePath]
[-nbPath <pathNumberForEachConstraint> | | -allViolatingPaths] [-
delayType <max|min>]
```

where:

1. <constraintNameList> is the list of name of constraints; wild card are not accepted.
2. -max | -min specifies if you want just the upper bound or just the lower bound timing constraints, or both if you do not specify any of those;
3. [-tracePath] specifies that you do want a trace of the paths or just a final global summary;
4. [-number <n>] specifies the number of worst timings you want to see for each constraints. The default is 1. It applies to both the number of lines you want to see in the summary and the number of trace you want to see in case you want to see the trace;
5. [-allpaths] specifies that you want a report for all the trace of the selcted constraints.
6. [-allViolatingPaths] that you just want a trace for violating paths.

Examples of use are:

```
CTAEClient> set clock clk -period 10
CTAEClient> set pathCluster pc1
CTAEClient> set arrivalTime -max 0 allInputs() -pathCluster pc1
CTAEClient> set requiredTime -max 11 allOutputs() -pathCluster pc1
CTAEClient> report TIMINGS  -max

################################################################
#                  T I M I N G    R E P O R T                  #
################################################################
|                                                              |
|   top cell "F16W8DA_P"                          derating: 1.00 |
|==============================================================|
| Type       | Name                | Required| Initial |  Current |
|-----------+-------------------|---------+---------+----------|
|pathCluster| pc1             | 11.00 ns| 56.23 ns|  17.06 ns| violated +6.06



|-----------+-------------------+---------+---------+----------|
|clock      | clk             | 10.00 ns| 22.22 ns|  13.05 ns| violated +3.05
|-----------+-------------------+---------+---------+----------|

CTAEClient> report TIMINGS -max -trace

################################################################
#               P A T H C L U S T E R                          #
################################################################
| name: pc1                                                    |
| input connectors:                                            |
|   name      rMax  fMax   rMin   fMin                         |
|   in1       0     0      ----   ----                         |
|   in2       0     0      ----   ----                         |
|   ...       0     0      ----   ----                         |
| input connectors:                                            |
|   name      rMax  fMax   rMin   fMin                         |
```

```
|   out1      11    11     ----   ----                          |
|   out2      11    11     ----   ----                          |
|   ...       11    11     ----   ----                          |
|---------------------------------------------------------------|
| first worst timing:                                           |
|                                                               |
|instance Name                                                  |
|input Pin -----> output Pin    trans total      cap    cell    |
|---------------------------     --- -----       ----   ----    |
|                  --> in1      F    .99         .28   top      |
|i_1006      I --> Z            F   2.57        2.28   ni01d5   |
||wr_cntr_ff_b3 CDN --> QN      R   5.51         .96   dfctnb   |
|i_905       A1 --> ZN          F   6.52         .64   nd02d1   |
|i_904       I --> ZN           R  10.61        2.28   in01d1   |
|i_920       A1 --> ZN          F  12.14         .38   ao06d1   |
|i_911       A3 --> ZN          R  13.13         .45   nd04d1   |
|i_910       B1 --> ZN          F  13.72         .63   fn01d2   |
|i_909       I --> Z            F  14.97        1.32   ni01d4   |
|i_945       B1 --> ZN          R  15.53         .40   fn01d2   |
|i_943       A2 --> ZN          F  17.06         .35   nd03d1   |
|               --> out2        F  17.06         .35   top      |
|---------------------------------------------------------------|

################################################################
#                           C L O C K                          #
################################################################
| name: clk                                                     |



| period: 10
| input connectors:                                             |
|   name      rMax  fMax   rMin   fMin                          |
| input connectors:                                             |
|   name      rMax  fMax   rMin   fMin                          |
|---------------------------------------------------------------|
| first worst timing:                                           |
|                                                               |
|instance Name                                                  |
|input Pin -----> output Pin    trans total      cap    cell    |
|---------------------------     --- -----       ----   ----    |
|ff1         CP --> Q           F   1.5          .28   dfctnb   |
|i_1006      I --> Z            F   2.57        2.28   ni01d0   |
|i_905       A1 --> ZN          F   6.52         .64   nd02d1   |
|i_904       I --> ZN           R  10.61        2.28   in01d1   |
|i_920       A1 --> ZN          F  12.14         .38   ao06d1   |
|ff3         SETUP              F  13.05               dfctnb   |
|---------------------------------------------------------------|

################################################################
#                  T I M I N G   R E P O R T                   #
################################################################
|                                                               |
|   top cell "F16W8DA_P"                        derating: 1.00 |
```

Type	Name	Required	Initial	Current
pathCluster	pc1	11.00 ns	56.23 ns	17.06 ns	violated +6.06
clock	clk	10.00 ns	22.22 ns	13.05 ns	violated +3.05

4.23 Do We Need "False Path" or "Stable Pin" for Timing Constraints

4.23.1 Stable Pins

Stable pins are a way for timing verifier to specify which external input connector should be considered as stable when tracing paths. Working with timing constraints, a external input connector is automatically considered as stable as soon as it does not belongs to any constraints. You can view with the "show" command all the input connectors affected by a timing constraints. If accidentaly one input connector is belonging to a constraint while it should not (for instance you used a wild card to specify some arrivalTime, and this wild card accidentaly matched the reset input connector that you do not want to constrain), you just remove it from the list with the help of the "clear" command.

4.24 Overwritting Mechanism

4.24.1 Introduction

The overwriting mechanism has been introduced because it has been requested by numerous users. The direct conclusion is that now the order in which you specify different constraints is very important. This order does affect

1. 1) the value of a parameter;
2. 2) the argument type accepted by the pathCluster constraint;

4.24.1.1 Affecting the Value of a Parameter

This is easy to understand: if you re-specify the value of a parameter, this new value will be used rather than the previous one. For instance, if you do the following:

```
set drive 5 allInputs()
set drive 3 in1
```

the final value of the drive of the "in1" input connector is "3". As a matter of fact, the previous set of commands is not at all equivalent to the following one:

```
set drive 3 in1
set drive 5 allInputs()
```

where the final value of the drive of the connector "in1" is now "5".

4.24.1.2 Affecting the Argument Type Accepted by the PathCluster Constraint

This one is more tedious. As explained in the sections presenting the pathCLuster, you can or can NOT specify arrivalTime (resp. requiredTime) on some internal connector according to the following exhaustive list of status:

1. broken path
2. black box
3. transparent latch
4. reset or clock paths seen as combinational paths

You thus need to really undertand how the tool is working to be sure that it will do what you want him to do.

4.24.2   Broken Paths

4.24.2.1 Introduction

Broken path are introducing BPIC and BPOC (resp. Broken Path Input Connector and Broken Path Output Connector) that can be set as argument to the "set arrivalTime" command. To clarify what happens according to the sequence you use to set patch cluster and broken paths, let's take the following design (see Fig. 13).

```
          +-----------------------------------------------+
          |                                               |
          |                                               |
          |                path1                          |
     in1 -+-----------------------------------------------+- out1
          |                                               |
          |               pathU5                          |
          |                 U5                            |
          |     path21      |\          path22            |
     in2 -+-----------------+ +---------------------------+- out2
          |                i|/ z                          |
          |                                               |
          |                 path3                         |
     in3 -+-----------------------------------------------+- out3
          |                                               |
          |                                               |
          |                                               |
          +-----------------------------------------------+
```

Fig. 13. Example where we will add a broken path.

4.24.2.2 Not Setting Broken Paths

If we just specify a global cluster using the "*" wild card such as:

```
set pathCluster pc1
set arrivalTime -max 0 * -pathcluster pc1
set requiredTime -max 10 * -pathcluster pc1
```

Then the arrivalTime (resp. requiredTime) "*" wild card will match the external input connector "in1", "in2" and "in3" (resp. the external output connector "out1", "out2" and "out3"). And the different paths will be constrained as follows:

pathName	arrivalTime	requiredTime
path1	0 (in1)	10 (out1)
path3	0 (in3)	10 (out3)
path21 + pathu5 + path22	0 (in2)	10 (out2)

4.24.2.3 Setting Broken Paths Before the Path Cluster

If we set a broken path before the path cluster as follows:

```
set break u5 -in i -out z
set pathCluster pc1
set arrivalTime -max 0 * -pathcluster pc1
set requiredTime -max 10 * -pathcluster pc1
```

Then the arrivalTime (resp. requiredTime) "*" wild card will match the external input connector "in1", "in2" and "in3" and the BPOC "u5.z" (resp. the external output connector "out1", "out2" and "out3" and the BPIC "u5.i"). And the different paths will be constrained as follows:

pathName	arrivalTime	requiredTime
path1	0 (in1)	10 (out1)
path3	0 (in3)	10 (out3)
path21	0 (in2)	10 (u5.i)
path22	0 (u5.z)	10 (out2)

4.24.2.4 Setting the Broken Path After the Path Cluster

If we set a broken path after the path cluster as follows:

```
set pathCluster pc1
set arrivalTime -max 0 * -pathcluster pc1
set requiredTime -max 10 * -pathcluster pc1
set break u5 -in i -out z
```

Then the arrivalTime (resp. requiredTime) "*" wild card will match the external input connector "in1", "in2" and "in3" (resp. the external output connector "out1", "out2" and "out3"). And the different paths will be constrained as follows:

pathName	arrivalTime	requiredTime
path1	0 (in1)	10 (out1)
path3	0 (in3)	10 (out3)
path21	0 (in2)	----(u5.i)
path22	---(u5.z)	10 (out2)

That is, path22 and path21 WON'T be optimized.

4.24.2.5 Clearing the Broken Path

If you do clear a broken path whose connectors are are affected by a path cluster, you will get a message except if you use the -force option.

```
CTAEClient> set  break u5.i --> u5.z
CTAEClient> set pathCluster pc1
CTAEClient> set arrivalTime -max 0 * -pathcluster pc1
CTAEClient> set requiredTime -max 10 * -pathcluster pc1
CTAEClient> clear break u5.i --> u5.z
   Error: .....
CTAEClient> clear arrivalTime u5.z -pathcluster pc1
CTAEClient> clear requiredTime u5.i -pathcluster pc1
CTAEClient> clear break u5.i --> u5.z
  Clearing broken path u5.i --> u5.z
CTAEClient>
```

4.24.3 Black Box

4.24.3.1 Introduction

Black Box are introducing BBIC and BBOC (resp. Black Box Input Connector and Black Box Output Connector) that can be set as argument to the "set arrivalTime" command. To clarify what happens according to the sequence you use to set patch cluster and BlackBox paths, let's take the following design (see Fig. 14).

```
        +-----------------------------------------------+
        |                                               |
        |                                               |
        |                path1                          |
   in1 -+-----------------------------------------------+- out1
        |                                               |
        |               pathU5                          |
        |                 U5 (subcella)                 |
        |     path21    +------+      path22            |
   in2 -+---------------+i    z+------------------------+- out2
        |               +------+                        |
        |                                               |
        |                 path3                         |
   in3 -+-----------------------------------------------+- out3
        |                                               |
        |                                               |
        |                                               |
        +-----------------------------------------------+
```

Fig. 14. Example where we will add a black box path.

4.24.3.2 Not Setting Black Box

If we just specify a global cluster using the "*" wild card such as:

```
set pathCluster pc1
set arrivalTime -max 0 * -pathcluster pc1
set requiredTime -max 10 * -pathcluster pc1
```

Then the arrivalTime (resp. requiredTime) "*" wild card will match the external input connector "in1", "in2" and "in3" (resp. the external output connector "out1", "out2" and "out3"). And the different paths will be constrained as follows:

pathName	arrivalTime	requiredTime
path1	0 (in1)	10 (out1)
path3	0 (in3)	10 (out3)
path21	0 (in2)	10 - (pathu5 + path22) (u5.i)
pathu5	path21 (u5.i)	10 - path22 (u5.z)

```
path22        path21 + pathU5 (u5.z)   10  (out2)
```

4.24.3.3 Setting Black Box Before the Path Cluster

If we set a Black Box before the path cluster as follows:

```
set blackBox subcella
set pathCluster pc1
set arrivalTime -max 0 * -pathcluster pc1
set requiredTime -max 10 * -pathcluster pc1
```

Then the arrivalTime (resp. requiredTime) "*" wild card will match the external input connector "in1", "in2" and "in3" and the BBOC "u5.z" (resp. the external output connector "out1", "out2" and "out3" and the BBIC "u5.i"). And the different paths will be constrained as follows:

pathName	arrivalTime	requiredTime
path1	0 (in1)	10 (out1)
path3	0 (in3)	10 (out3)
path21	0 (in2)	10 (u5.i)
path22	0 (u5.z)	10 (out2)

4.24.3.4 Setting the Black Box After the Path Cluster

If we set a black box after the path cluster as follows:

```
set pathCluster pc1
set arrivalTime -max 0 * -pathcluster pc1
set requiredTime -max 10 * -pathcluster pc1
set blackBox subcella
```

Then the arrivalTime (resp. requiredTime) "*" wild card will match the external input connector "in1", "in2" and "in3" (resp. the external output connector "out1", "out2" and "out3"). And the different paths will be constrained as follows:

pathName	arrivalTime	requiredTime
path1	0 (in1)	10 (out1)
path3	0 (in3)	10 (out3)
path21	0 (in2)	----(u5.i)
path22	---(u5.z)	10 (out2)

That is, path22 and path21 WON'T be optimized.

4.24.3.5 Clearing the Black Box

If you clear a black box whose connectors are affected by a path cluster, you will get a message except if you use the -force option.

```
CTAEClient> set blackBox subcella
CTAEClient> set pathCluster pc1
CTAEClient> set arrivalTime -max 0 * -pathcluster pc1
CTAEClient> set requiredTime -max 10 * -pathcluster pc1
CTAEClient> clear blackBox subcella
   Error ...
CTAEClient> clear requiredTime u5.i -pathcluster pc1
CTAEClient> clear arrivalTime u5.z -pathcluster pc1
CTAEClient> clear blackBox subcella
  Clearing blackBox subcella
CTAEClient>
```

4.24.4 Setting Combinational Paths

4.24.4.1 Introduction

Combinational latches are removing SECC and SERC (resp. Sequential Element Clock Connector and Sequential Element Reset Connector) that can be set as argument to the "set arrivalTime" command. To clarify what happens according to the sequence you use to set patch cluster and combinational paths, let's take the following design (see Fig. 16).

```
     +------------------------------------------------+
     |                                                |
     |                                                |
     |                path1                           |
in1 -+------------------------------------------------+- out1
     |                                                |
     |               pathU5                           |
     |                 U5                             |
     |     path21    +------+      path22             |
in2 -+---------------+d    q+-------------------------+- out2
     |               |      |                         |
clk -+---------------+e     |                         |
     |     path23    +------+                         |
     |                                                |
     |                 path3                          |
in3 -+------------------------------------------------+- out3
     |                                                |
     |                                                |
     +------------------------------------------------+
```

Fig. 16. Example where we will set a combinational path.

4.24.4.2 Not Setting the Combinational Path

If we just specify a global cluster using the "*" wild card such as:

```
set pathCluster pc1
set arrivalTime -max 0 * -pathcluster pc1
set requiredTime -max 10 * -pathcluster pc1
```

Then the arrivalTime (resp. requiredTime) "*" wild card will match the external input connector "in1", "in2" and "in3" and the SEDOC "u5.q" (resp. the external output connector "out1", "out2" and "out3" and the SECC "U5.e" and the SEDIC "U5.d"). And the different paths will be constrained as follows:

pathName	arrivalTime	requiredTime
path1	0 (in1)	10 (out1)
path3	0 (in3)	10 (out3)
path23	0 (clk)	10 (u5.e)
path21	0 (in2)	10 (u5.d)
path22	0 (u5.q)	10 (out2)

4.24.4.3 Setting Combinational Path Before the Path Cluster

If we set a combinational path before the path cluster as follows:

```
set combinationalPath  u5 -in e -out q
set pathCluster pc1
set arrivalTime -max 0 * -pathcluster pc1
set requiredTime -max 10 * -pathcluster pc1
clear required u5.d
clear arrival u5.q
```

Then the arrivalTime (resp. requiredTime) "*" wild card will match the external input connector "in1", "in2" and "in3" (resp. the external output connector "out1", "out2" and "out3" and the SEDIC "u5.d"). And the different paths will be constrained as follows:

pathName	arrivalTime	requiredTime
path1	0 (in1)	10 (out1)
path3	0 (in3)	10 (out3)
path23	0 (clk)	10 - (path22 + pathu) (u5.e)
path22	path23 + pathu (u5.q)	10 (out2)
path21	0 (in2)	10 (u5.d)

4.24.4.4 Setting the Combinational Path After the Path Cluster

If you set a combinational Path whose connectors are affected by a path cluster constraint you will get a message except if the -force option is used.

```
CTAEClient> set pathCluster pc1
CTAEClient> set arrivalTime -max 0 * -pathcluster pc1
CTAEClient> set requiredTime -max 10 * -pathcluster pc1
CTAEClient> set combinationalPath  u5 -in e -out q
CTAEClient> clear requiredTime u5.e -pathcluster pc1
CTAEClient> clear arrivalTime u5.q -pathcluster pc1
CTAEClient> set combinationalPath  u5 -in e -out q
  Setting combinationalPath u5.e --> u5.q
CTAEClient>
```

4.24.4.5 Clearing the Combinational Path

If you clear a Combinational Path after the path cluster as follows:

```
set combinationalPath  u5 -in e -out q
set pathCluster pc1
set arrivalTime -max 0 * -pathcluster pc1
set requiredTime -max 10 * -pathcluster pc1
clear required u.d
clear combinationalPath  u5.e --> u5.q
```

Then the arrivalTime (resp. requiredTime) "*" wild card will match the external input connector "in1", "in2" and "in3" (resp. the external output connector "out1", "out2" and "out3" and the SEDIC "u5.d"). And the different paths will be constrained as follows:

```
   pathName     arrivalTime              requiredTime
   path1        0  (in1)                 10  (out1)
   path3        0  (in3)                 10  (out3)
   path23       0  (clk)                 --- (u5.e)
   path22       --- (u5.z)               10  (out2)
   path21       0 (in2)                  10 (u5.d)
```

4.24.5 Conclusion: Always Checking

The overwriting is helpful and really lets you do what you want. On the other hand it might be sometimes tedious to use at the very beginning. This is why we advise the user to always use the "show" command in case of doubt; this command will let know him the real status of any parameter or constraint.