cluster\_constraint -help

#

# [cluster\_constraint, usage]

# cluster\_constraint accept\_overflow [-type\_overflow <TYPE\_OVERFLOW>]+ [-reserve]

#

#

# \* COMMAND DESCRIPTION:

# ---------------------

# Let AC trade logic for cut reduction, ie it will accept to map more logic

# if the cut can be improved. Values are percentage.

# The overflow resources are taken from a 'reserved' area (cf. figure below).

# FPGA resource mapping for one resource from 0 to 100%, a resource can be

# either lut, ramlut, reg, bram or dsp:

# 0 max\_fill(\*) upper\_limit(\*\*) 100%

# +----------------|-------------|--------------------+

# <- allocatable-> <- reserve -> <- P& R resources ->

# (1) (2) (3)

# (1) area for design allocation

# (2) reserved area for overflow

# (3) forbidden area only for FPGA P&R and estimation error

# (\*) filling rate constraints.

# (\*\*) maximum filling rate accepted if constraints adjustement is necessary,

# cf. command 'cluster\_constraint\_adv set\_max\_fill\_limit'.

# By default, the overflow is specified as a percentage of area (1), which is

# the allocatable area. If the option '-reserve' is specified, the overflow

# is specified by means of a percentage of the 'reserve' area, i.e. area (2).

# There exists 2 ways to compute overflow variation:

# 'standard\_mode' is based on variation of area (1),

# 'reserve\_overflow\_mode' is based on variation of area (2).

# By default 'reserve\_overflow\_mode' is used (cluster\_constraint\_adv enable

# -reserve\_overflow).

# For backward compatibility, if values given by the accept\_overflow command

# do not match the overflow computation mode, they are automatically

# converted to obtain a similar amount of permitted overflow resources.

#

# \* OPTIONAL ARGUMENTS:

# -------------------

#

# -type\_overflow <TYPE\_OVERFLOW>

# type is either reg, lut, ramlut, bram or dsp and TYPE is the corresponding

# percentage overflow allowed. Default are respectively 5, 10, 10, 0, 0

# (allocatable area).

# Type: <TYPE\_OVERFLOW> is integer.

#

# -reserve

# specify if the values are percentage of 'reserve area' or 'allocatable

# area'.

# Type: switch.

#

# \* COMMAND USAGE EXAMPLES:

# -----------------------

#

# - Example 1:

#

# - cluster\_constraint accept\_overflow -reg 10 -lut 10

# Will accept 10% overflow on both LUTs and REGs. If filling constraint

# on registers is 60%, there will be 124416 available registers on one

# FPGA (V6 family) plus 10% (12441) additional registers if a better cut

# can be found.

#

#

# [cluster\_constraint, usage(1/3)]

# cluster\_constraint atomic [-fnmatch] [-module <MODULE>] [-module\_content <MODULE\_CONTENT>] [-flow\_selection]

#

#

# \* COMMAND DESCRIPTION:

# ---------------------

# Provides several ways to specify modules that will not be splitted across

# FPGAs. One must make sure that any module with all its instrumentation

# logic that is declared as atomic can fit on one FPGA.

#

# \* OPTIONAL ARGUMENTS:

# -------------------

#

# -fnmatch

# If this option is used, the other parameters are interpreted as pattern.

# Any module whose name matches the pattern will be selected.

# Type: switch.

#

# -module <MODULE>

# The specified module will not be splitted.

# Type: <MODULE> is a string.

#

# -module\_content <MODULE\_CONTENT>

# Any module containing an instance of the specified module will not be

# splitted.

# Type: <MODULE\_CONTENT> is a string.

#

# -flow\_selection

# <flow\_selection> is a flag that is either zcorebuild or ztopbuild. When

# <flow\_selection> is specified on the command description it is possible to

# specify which tool will process the command. If nothing is specified the

# command is sent to both zTopBuild and zCoreBuild.

# Type: switch.

#

# \* COMMAND USAGE EXAMPLES:

# -----------------------

#

# - Example 1:

#

# - cluster\_constraint atomic -module dummy

# Will not split any module whose name is 'dummy'

#

# - Example 2:

#

# - cluster\_constraint atomic -module dummy\* -fnmatch

# Will not split any module whose name starts with 'dummy'

#

# - Example 3:

#

# - cluster\_constraint atomic -module\_content dummy

# Will not split any module that contains an instance of module 'dummy'

#

#

# [cluster\_constraint, usage(2/3)]

# cluster\_constraint merge\_blocks [-path\_list <PATH\_LIST>] [-flow\_selection]

#

#

# \* COMMAND DESCRIPTION:

# ---------------------

# The auto-clustering will keep the specified blocks on the same fpga. The

# command can handle multiple statements sharing a common block, in this case

# all the blocks will be merged.

# This command handle hierarchy overlapping in the following way:

# If an instance of the hierarchy is touched by several merge\_block commands,

# the merge\_block command that references the closest of its ancestor will be

# selected.

#

# \* OPTIONAL ARGUMENTS:

# -------------------

#

# -path\_list <PATH\_LIST>

# PATH\_LIST is a list of instance paths to be grouped together. This group

# must fit on one FPGA.

# Type: <PATH\_LIST> is a list.

#

# -flow\_selection

# <flow\_selection> is a flag that is either zcorebuild or ztopbuild. When

# <flow\_selection> is specified on the command description it is possible to

# specify which tool will process the command. If nothing is specified the

# command is sent to both zTopBuild and zCoreBuild.

# Type: switch.

#

# \* COMMAND USAGE EXAMPLES:

# -----------------------

#

# - Example 1:

#

# - cluster\_constraint merge\_blocks -path\_list { a.b.c a.d.e a.f }

# - cluster\_constraint merge\_blocks -path\_list { a.b.g a.f a.x }

# Fusion of merge\_block commands : the 2 commands will keep together all

# the specified blocks because they share 'a.f'.

#

# - Example 2:

#

# - (1) cluster\_constraint merge\_block -path\_list { a.b.c a.d.e }

# - (2) cluster\_constraint merge\_block -path\_list { a.b }

#

# Hierarchy overlapping: the two commands will generate 2 partitions:

# - blocks 'a.b.c' and 'a.d.e' will stay together (command 1),

# - all instances except 'c' of block 'a.b' will stay together (command

# 2).

#

#

# [cluster\_constraint, usage(3/3)]

# cluster\_constraint add\_group <NAME> [-exclusive] [-type\_maxfill <TYPE\_MAXFILL>] [-max\_fpga <MAX\_FPGA>] [-min\_fpga <MIN\_FPGA>] [-nozdelay]

# [-type\_overflow <TYPE\_OVERFLOW>] [-path\_list <PATH\_LIST>] [-flow\_selection]

#

#

# \* COMMAND DESCRIPTION:

# ---------------------

# Defines a logic group of instances that may require the resources of more

# than one FPGA. Any instance with a hierarchical path that matches at least

# one pattern specified in the 'path\_list' option will be attached to this

# group. One can define multiple groups with several commands.

# Any instance that does not belong to a group is attached to the default

# group. Unlike instances that belongs to a group, any instance that belongs

# to the default group has no mapping restriction.

# if '-exclusive' option is specified, blocks belonging to the default group

# cannot be mapped on this core.

# if '-nozdelay' option is specified, zDelay instrumentation cost will be

# discarded from this core.

# It is possible to create a real partitionning of the design by means of

# group definitions if one declares a group at the top of the design. Thus,

# there will be no mixing of instances attached to groups and instances

# attached to the default group.

# If an instance has a path that matches several patterns, the longest (most

# selective) pattern is chosen. The pattern syntax command has limited

# support of regular expressions. In particular, the asterisk \* is a pattern

# that shall match any string, including the null string. Pattern matching is

# case insensitive.

# Usual constraints can be attached to any group. For example, if the

# max\_fpga option is set, the group will be mapped at most on the specified

# number of FPGAs.

#

# \* MANDATORY ARGUMENTS:

# --------------------

#

# <NAME>

# assigns a name to the group of instances.

# Type: <NAME> is a string.

#

# \* OPTIONAL ARGUMENTS:

# -------------------

#

# -exclusive

# no blocks belonging to the default group can be mapped on this core, core

# is isolated.

# Type: switch.

#

# -type\_maxfill <TYPE\_MAXFILL>

# type\_maxfill is either max\_reg, max\_lut, max\_ramlut, max\_bram or max\_dsp

# and TYPE\_MAXFILL is the corresponding filling rate.

# Type: <TYPE\_MAXFILL> is integer.

#

# -max\_fpga <MAX\_FPGA>

# constraint that specifies the maximum number of FPGAs allowed for this

# group

# Type: <MAX\_FPGA> is an integer.

#

# -min\_fpga <MIN\_FPGA>

# reserves at least this number of FPGA for the group.

# Type: <MIN\_FPGA> is an integer.

#

# -nozdelay

# zDelay instrumenting cost will be discarded in this core, use -exclusive to

# improve zDelay estimation accuracy .

# Type: switch.

#

# -type\_overflow <TYPE\_OVERFLOW>

# type\_overflow is either ovf\_reg, ovf\_lut, ovf\_ramlut, ovf\_bram or ovf\_dsp

# and TYPE is the corresponding percentage overflow allowed. Default are

# respectively 5, 10, 10, 0, 0.

# Type: <TYPE\_OVERFLOW> is integer.

#

# -path\_list <PATH\_LIST>

# list of instances and/or patterns of instance paths. If no path\_list is

# given,the group is seen as 'default group' and any instance that is not

# attached to another group will be attached to this one. If 'path\_list' is

# empty and 'max\_fpga=0', no FPGA will be allocated for this default group.

# Type: <PATH\_LIST> is a list.

#

# -flow\_selection

# <flow\_selection> is a flag that is either zcorebuild or ztopbuild. When

# <flow\_selection> is specified on the command description it is possible to

# specify which tool will process the command. If nothing is specified the

# command is sent to both zTopBuild and zCoreBuild.

# Type: switch.

#

# \* COMMAND USAGE EXAMPLES:

# -----------------------

#

# - Example 1:

#

# - cluster\_constraint add\_group everything\_else -path\_list { top\_design }

# - cluster\_constraint add\_group I1 -path\_list { top\_design.inst1 }

# - cluster\_constraint add\_group I2 -path\_list { top\_design.inst2 }

# 3 non overlapping logical groups are created. Group I1 will contain

# 'inst1', group I2 will contain 'inst2', and everything else will be

# attached to group 'everything\_else'

#

# - Example 2:

#

# - cluster\_constraint add\_group app1 -path\_list { top.gg\* top.dummy.dd\* }

# - cluster\_constraint add\_group app2 -path\_list { top.aa\* }

# The 2 groups of instances 'app1' and 'app2' will be mapped on 2

# distinct sets of FPGAs. 'app1' will keep together all the instances of

# 'top' whose name starts with 'gg' and all the instances in top.dummy

# whose name starts with dd. 'app2' will keep together all the instances

# of top whose name starts with 'aa'. All the remaining instances will be

# mapped either with 'app1', 'app2' or on another FPGA.

#