Insight Analyzer

Customizing Fanout Checks for Clocked Circuits

Preliminary Information
Insight Analyzer v5.20

Problem Statement

User wants to run standard Insight Fanout checks, but give particular threshold values to “clocked” nodes.

User does not want to write a custom fanout check.

There are many ways to do this, using Insight Analyzer. Here, we present three potential solutions.

Solution Overview

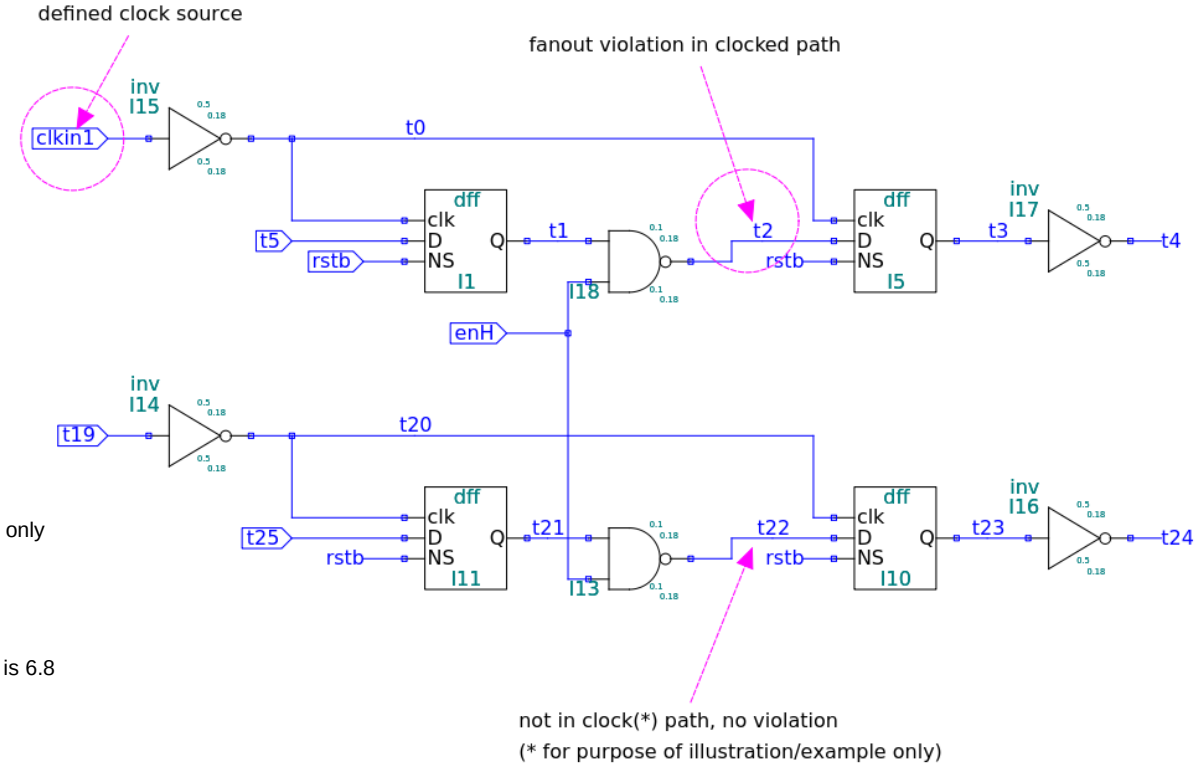
Insight Analyzer built in Fanout checks already support virtually all topologies and corner cases. So it's best to leverage the existing checks.

We do this by applying a specific Fanout threshold of failure to nets of interest: The "clocked" nodes.

1) Define what we mean by "clocked node".
2) Plant the definition in circuit (seed), have it propagate.
3) Use propagation to carry the specific fanout threshold.
4) After seeding, run standard Fanout checks.

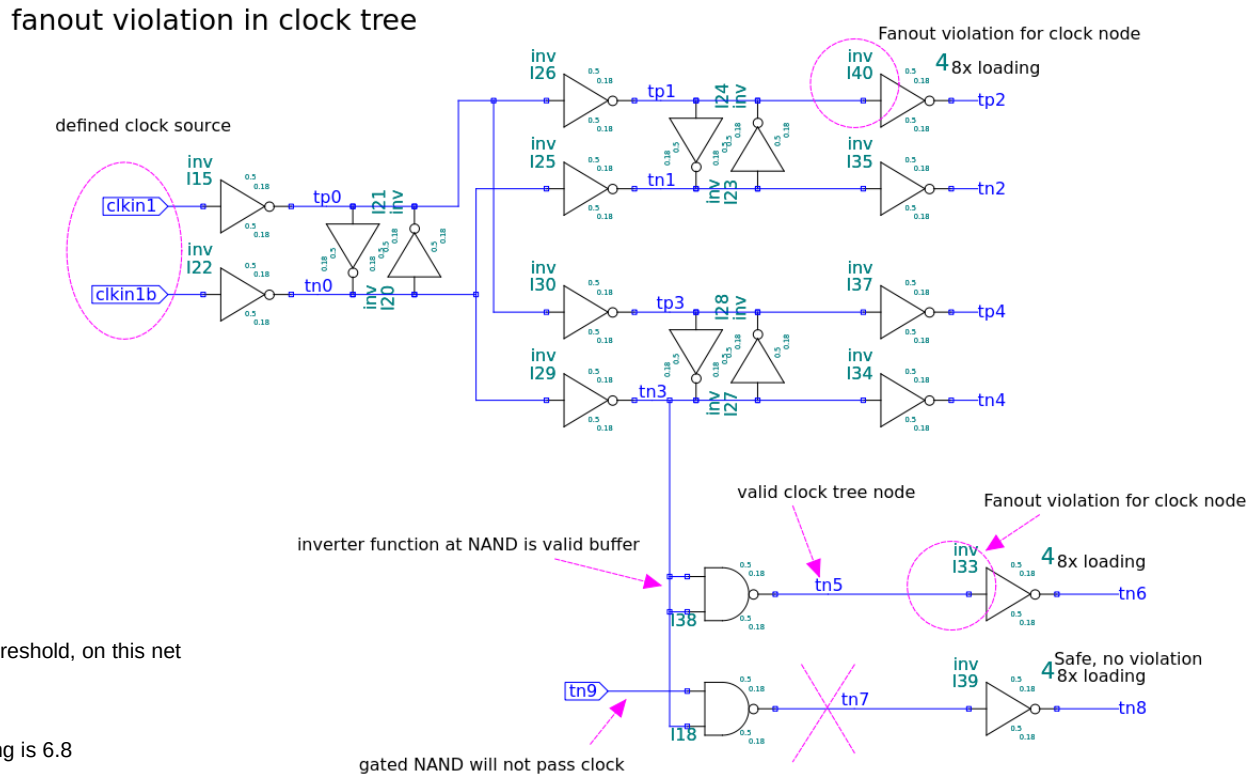
Solution #1: Defined Clocked Paths

```
# Fanout check: Apply particular thresholds based on clocked signals, different threshold per net
#
# Define what is a "clocked net"?
# In this example, we choose to define a clock source.
# From the clock source, all impacted signals downstream
# are considered "clocked net". Basically, anything in
# the path of clocked logic. We will use a tighter fanout
# threshold on such nets. Combinatorial logic will fall
# back to a higher fanout threshold.

::cdb::setupDevices {
   { "NCH"          NMOS  { {-Cg 1} {-DefL 1} } }
   { "PCH"          PMOS  { {-Cg 1} {-DefL 1} } }
}
devparams apply -override

pwrdefs define "" "VDD" "" 1.00
pwrdefs define "" "VSS" "" 0.00
pwrdefs apply -override

csrc load "top.spi" -caseSensitive

# Select one of the top IO as our clock source.
# From this clock source, send down a low fanout threshold for later checking
cscan net "/-clkin1" cones -pushDownstream {
   if { [ cdb net %n getCell ] eq "top" } {; # for simple demo, top schematic nets only
      cdb net %n addAttr "FanoutThreshold" 3.1; # override the global setting
   }
}

# Now, run the standard Fanout checks:
plugin configure "Fanout" -doLoadOver 1 -specLoadOver 6.8; # global setting is 6.8
plugin run "Fanout" -save "fanout.rpt"
```



Solution #2: Clock Tree

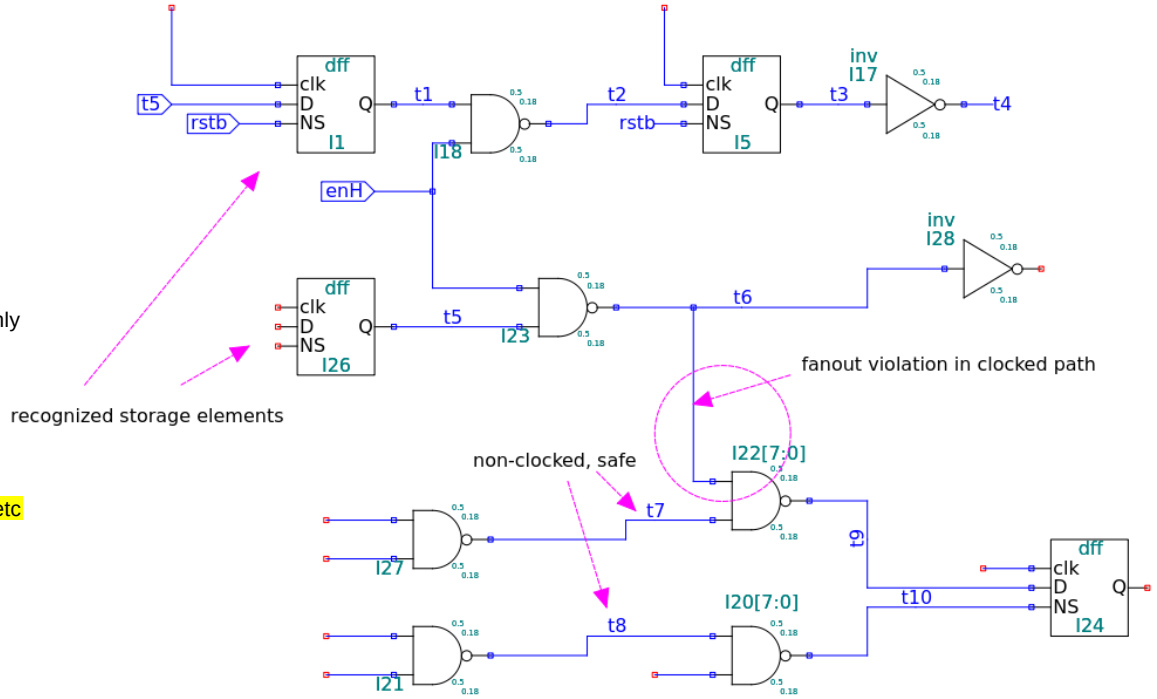
```
# Fanout check: Apply particular thresholds based on clock trees, different from other nets
#
# Define what is a "clock tree"?
# In this example, we choose to define a clock source and
# have Insight Analyzer propagate the clock through proper
# tree stuctures. Propagation will go through normal buffers,
# proper level shifters/isolation, but will stop at
# conditional gating such as CMOS NAND where other inputs
# are present.

::cdb::setupDevices {
   { "NCH"          NMOS  { {-Cg 1} {-DefL 1} } }
   { "PCH"          PMOS  { {-Cg 1} {-DefL 1} } }
}
devparams apply -override

pwrdefs define "" "VDD" "" 1.00
pwrdefs define "" "VSS" "" 0.00
pwrdefs apply -override

csrc load "top.spi" -caseSensitive

# Define this top IO as our clock source.
# From this clock source, propagation will happen immediately.
cdb ckTree create "/-clkin1"; # create clock and trigger propagation

# From the propagated clock, pre-tag clock nodes for fanout checking later
# This will apply ONLY to those clock nodes that are directly part of
# the clock tree.
foreach clkSegment [ cdb ckTree listNets "/-clkin1" ] {
   set netName [ lindex $clkSegment 1 ]
   cdb net $netName addAttr "FanoutThreshold" 2.3; # override the global threshold, on this net
}

# Now, run the standard Fanout checks:
plugin configure "Fanout" -doLoadOver 1 -specLoadOver 6.8; # global setting is 6.8
plugin run "Fanout" -save "fanout.rpt"
```



Solution #3: Recognized Storage

```
# Fanout check: Apply particular thresholds based on clocked signals, different threshold per net
#
# Define what is a "clocked signal"?
# In this example, we choose to leverage recognized storage nodes.
# From the storage nodes, propagate downstream logic. All impacted
# signals downstream are considered to have tigher fanout threshold.

::cdb::setupDevices {
  { "NCH"          NMOS  { {-Cg 1} {-DefL 1} } }
  { "PCH"          PMOS  { {-Cg 1} {-DefL 1} } }
}
devparams apply -override

pwrdefs define "" "VDD" "" 1.00
pwrdefs define "" "VSS" "" 0.00
pwrdefs apply -override

csrc load "top.spi" -caseSensitive

proc tagDownstreamFromLatch { n } {
  cscan net $n cones -pushDownstream {
    if { [ cdb net %n getCell ] eq "top" } {; # for simple demo, top schematic nets only
      cdb net %n addAttr "FanoutThreshold" 6.0; # override the global setting
    }
  }
}

# Find all storage nodes, inside recognized latches
cscan nets {
  if {[ cdb net %n isLatchNode ]} {; # we came to a recognized latch/storage/DFF/etc
    set latchInfo [ cdb net %n getLatchInfo ]
    set outNode [ dict get $latchInfo "output" ]; # get the output
    tagDownstreamFromLatch $outNode; # from the output, go downstream
  }
}

# Now, run the standard Fanout checks:
plugin configure "Fanout" -doLoadOver 1 -specLoadOver 14; # global setting is 14
plugin run "Fanout" -save "fanout.rpt"
```

fanout per recognized latch/flop paths



Alternate Solution

The previous solutions are based on setting a different threshold of failure for Fanout. The drawback to those solutions are: End user will see a flat report, without distinction of “clocked” vs non-clocked.

Here is an outline of an alternate solution, an extension that can modify a Fanout report.

1) Define and propagate clocks, as in previous solutions.
2) Install a “report qualifier” procedure in the Fanout check.
3) Run Fanout checks. All results will be sent to the report qualifier, one by one.
4) Per violation, query the circuit directly for any info, then re-write the violation for end user. “On a clock net”, etc