AN 919: Improving Quality of Results with Design Assistant

Contents

Improving Quality of Results with Design Assistant

Optimizing your design's source code is typically the first and most effective technique for improving the quality of results. Design Assistant is a convenient tool that allows you to identify potential issues earlier. Design Assistant runs targeted sanity checks and provides guidance at each stage, thereby reducing the total number of iterations for design closure.

Related Information

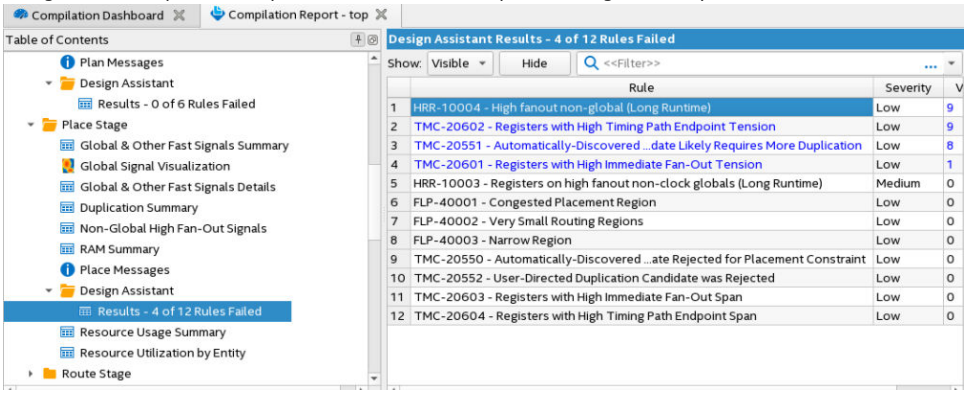
Intel Quartus Prime Pro Edition User Guide: Design Recommendations

About Design Assistant

Design Assistant conveniently identifies potential design issues, including circuit functionality and timing performance. Identifying and fixing issues early in the design cycle gives fewer and faster iterations. You save time running a full compilation if you solve issues in synthesis or in plan and place fitter stages. Intel® Quartus® Prime Pro Edition provides more DRC rules and in more stages. Design Assistant allows the flexibility to choose which rules to run, at what compile stages, and to select and filter rules of interest.

Figure 1. Design Assistant Report

Design Assistant provides reports in each of the compilation stages when you run it.



Some of the issues that Design Assistant can detect include:

- Incorrect SDCs
- Metastability because of clock and reset domain crossings
- Excessive logic levels

- High fan-out nets that can cause congestion
- Potential problems with the design's power-on strategy
- Retiming Restrictions that prevent the Hyper-Retimer from making optimizations

For faster iterations, run the rules on early-compilation snapshots. For fewer iterations, run lots of rules on the snapshots you have to catch and address all kinds of issues.

About the Design Assistant Design Example

Common user errors can result in on-board failures. This design shows those mistakes and how you catch them with Design Assistant.

The design contains two modules, each in a different clock domain. The design contains high fan-out signals, some congestion, and is missing Synopsis design constraints (SDCs) and proper clock domain crossing. By fixing these issues and adding some hyper-pipelining, the design can meet timing.

Download the design (`an919.zip`) from www.intel.com.

The design example consists of the following directories:

- `base` – the original design
- `intermediate` – the original design and fixes for SDC, CDC, RDC
- `final` – the intermediate design and additional optimizations

Each directory has the following files:

- `top.qpf` – project file
- `top.qsf` – Intel Quartus Prime settings file
- `top.sv` – top-level Verilog HDL file
- `top_clk1.sv` – submodule on one of the two clock domains
- `top_clk2.sv` – submodule on the other clock domain
- `byte_enabled_simple_dual_port_ram.v` – dual-port RAM
- `top.out.sdc` – SDC file

The `final` RTL directory also contains `reset_release.ip` and `clock_control.ip` files.

Related Information

an919.zip

Fixing SDC and CDC Rules in the Design Assistant Design Example

Refer to the intermediate design RTL and SDCs for this section. Compile the design in the `intermediate` directory to see the effects of all the changes you make in this section.

1. Open the design in Intel Quartus Prime Pro Edition: click **File > Open Project** and open the `base/top.qpf` project file from the `base` directory.
2. Under **Design Assistant Rule Settings**, turn on **Enable Design Assistant execution during compilation**.
3. Compile the design.
4. Analyze the results and fix high priority rules first.

Figure 2. Results in Compilation Mode

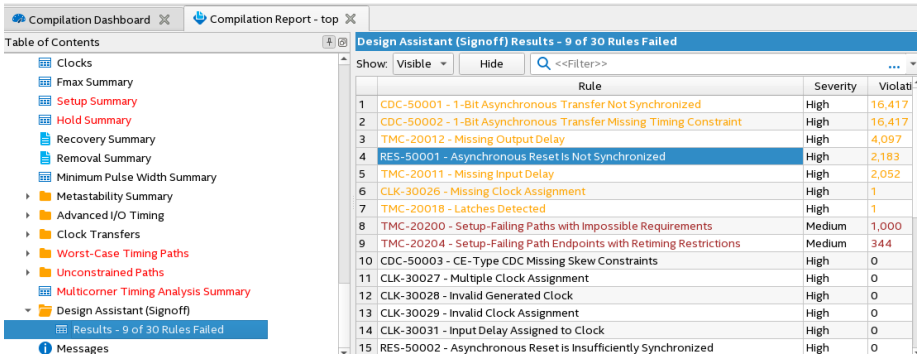
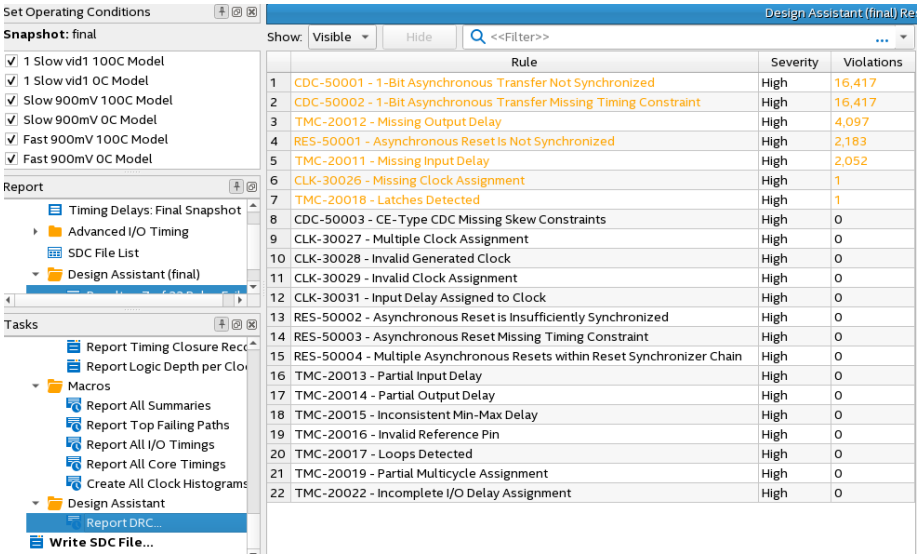


Figure 3. Results in Analysis Mode in Timing Analyzer



5. Update the SDC file and the RTL to fix missing input and output delay constraints for accurate timing results:
   a. Add missing `set_input_delay` SDCs to fix TMC-20011.

```
set_input_delay -add_delay  -clock [get_clocks {clk}]  0.500 [get_ports
{idat[*]}] -reference_pin [get_pins -no_duplicates {u_top_clk1|wen_q|
clk}]
set_input_delay -add_delay  -clock [get_clocks {clk2}]  0.500 [get_ports
{sel[*]}] -reference_pin [get_pins -no_duplicates {u_top_clk2|en0q|clk}]
set_input_delay -add_delay  -clock [get_clocks {clk}] -min 0.100
[get_ports {idat[*]}] -reference_pin [get_pins -no_duplicates
{u_top_clk1|wen_q|clk}]
```

```
set_input_delay -add_delay  -clock [get_clocks {clk2}] -min 0.100
[get_ports {sel[*]}] -reference_pin [get_pins -no_duplicates {u_top_clk2|
en0q|clk}]
```

b. Add missing `set_output_delay` SDCs to fix TMC-20012.

```
set_output_delay -add_delay  -clock [get_clocks {clk}]  -max 0.500
[get_ports {sum[*]}] -reference_pin [get_pins -no_duplicates {u_top_clk1|
wen_q|clk}]
set_output_delay -add_delay  -clock [get_clocks {clk}]  -max 0.500
[get_ports {latch_out}] -reference_pin [get_pins -no_duplicates
{u_top_clk1|wen_q|clk}]
set_output_delay -add_delay  -clock [get_clocks {clk2}] -max 0.500
[get_ports {final_dat[*]}] -reference_pin [get_pins -no_duplicates
{u_top_clk2|en0q|clk}]
set_output_delay -add_delay  -clock [get_clocks {clk}]  -min 0.100
[get_ports {sum[*]}] -reference_pin [get_pins -no_duplicates {u_top_clk1|
wen_q|clk}]
set_output_delay -add_delay  -clock [get_clocks {clk}]  -min 0.100
[get_ports {latch_out}] -reference_pin [get_pins -no_duplicates
{u_top_clk1|wen_q|clk}]
set_output_delay -add_delay  -clock [get_clocks {clk2}] -min 0.100
[get_ports {final_dat[*]}] -reference_pin [get_pins -no_duplicates
{u_top_clk2|en0q|clk}]
```

Refer to *Timing Closure Rules* for more information about these and other related rules.

6. Add reset synchronizers to fix RES-50001.
   a. In `top.sv`, add reset synchronization logic and feed their outputs to their respective clock domain to prevent metastability.

Figure 4. RES-50001

```
logic srst_clk1, srst_clk2;
logic rst_clk1_q1, rst_clk1_q2, rst_clk1_q3;
logic rst_clk2_q1, rst_clk2_q2, rst_clk1_q3;

// active high reset synchronizer
always@(posedge clk1 or posedge rst)
begin
  if(rst) begin
    rst_clk1_q1 <= 1'b1;
    rst_clk1_q2 <= 1'b1;
  end else begin
    rst_clk1_q1 <= 1'b0;
    rst_clk1_q2 <= rst_clk1_q1;
  end
end

always@(posedge clk2 or posedge rst)
begin
  if(rst) begin
    rst_clk2_q1 <= 1'b1;
    rst_clk2_q2 <= 1'b1;
  end else begin
    rst_clk2_q1 <= 1'b0;
    rst_clk2_q2 <= rst_clk2_q1;
  end
end

assign srst_clk1 = rst_clk1_q2;
assign srst_clk2 = rst_clk2_q2;
```

b. Constrain the new reset synchronizers or you see violations on RES-50003 or you may see violations on RES-50003 on subsequent compilations. Add false paths to the `clrn` pins of the reset synchronizer.

```
set_false_path -from [get_ports rst] -to [get_pins rst_clk1_q1|clrn]
set_false_path -from [get_ports rst] -to [get_pins rst_clk1_q2|clrn]
set_false_path -from [get_ports rst] -to [get_pins rst_clk2_q1|clrn]
set_false_path -from [get_ports rst] -to [get_pins rst_clk2_q2|clrn]
```

7. Add synchronization logic for data going between clk1 and clk2 and vice versa by changing `top.sv` and `top_clk2.sv`. The following code shows the CDC for going from clk1 to clk2.

```
// top.sv
// generate a pulse in one clock domain
always@(posedge clk1)
  en_q <= en;

// detect the positive edge of the pulse and change edges
// this edge crosses the clock domain
always@(posedge clk1 or posedge srst_clk1)
begin
  if(srst_clk1)
    en_pulse_det0q <= 1'b0;
  else if(en & ~en_q)
    en_pulse_det0q <= !en_pulse_det0q;
end

// send en_pulse_det0q into the en input of top_clk2.sv
// top_clk2.sv:
  logic en2_pulse;
  logic en0q, en1q, en2q;

// synchronize the detected edge into the clk2 domain and use it
// to generate a pulse
  always@(posedge clk2) begin
    en0q <= en;
    en1q <= en0q;
    en2q <= en1q;
    en2_pulse <= en2q ^ en1q;
  end

// use the pulse to enable data capture
always@(posedge clk2) begin
  if(en2_pulse)
    idat_q <= idat;
```

   a. Add SDCs to constrain the CDC correctly. Rules CDC-50002 and CDC-50003 detect missing SDCs.

```
set_clock_groups -asynchronous -group {clk} -group {clk2}

# clk1 to clk2 paths
for {set i 0} {$i < 8 } {incr i} {
set_net_delay -from u_top_clk1|
[$i].u_byte_enabled_simple_dual_port_ram_out|* -to u_top_clk2|idat_q[$i]
[*] -max \
   -get_value_from_clock_period dst_clock_period -value_multiplier 0.8
set_max_skew -from u_top_clk1|
[$i].u_byte_enabled_simple_dual_port_ram_out|*  -to u_top_clk2|idat_q[$i]
[*] \
   -get_skew_value_from_clock_period src_clock_period -
skew_value_multiplier 0.8
}

# clk2 to clk1 paths
set_net_delay -from u_top_clk2|math_q[*] -to math2_clk1[*] -max \
   -get_value_from_clock_period dst_clock_period -value_multiplier 0.8
set_max_skew -from u_top_clk2|math_q[*] -to math2_clk1[*] \
   -get_skew_value_from_clock_period src_clock_period -
skew_value_multiplier 0.8
```

   Refer to *Clock Domain Crossing and Reset Domain Crossing Rule* for more information about these and other related rules.

8. Remove inferred latch in RTL. Rule TMC-20018 catches the latch. Rule CLK-30026 catches that the design is using the en signal as a clock for the inferred latch.

```
// TMC-20018 inferred latch; CLK-30026 flags the en signal as a clock
logic inferred_lat;
always@(*) begin
  if(en)
    inferred_lat <= math1[0];
end
```

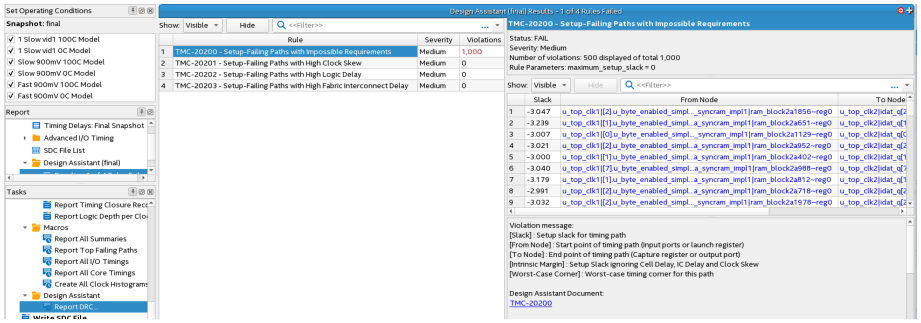
   The fix is:

```
// latch
logic inferred_lat;
always@(posedge clk1) begin
  if(en)
    inferred_lat <= math1[0];
end
```

9. In the base design, check for any TMC-20200 and TMC-20201 rule failures, which may be due to incorrect SDCs and improper CDCs. However, the fixes made in the previous steps should resolve them in this example design.

   Refer to *Intrinsic Margin* for more information on these and other related rules.

Figure 5. TMC-20200 Rule Violations



Related Information

- Reducing High Fan-out Nets on page 19
- Timing Closure Rules on page 16
- Clock Domain Crossing and Reset Domain Crossing Rules on page 17
- Intrinsic Margin Rules on page 18

Fixing Other Faults in the Design Assistant Design Example

When you correctly constrain the design and add CDCs, you see fewer rule violations, but the design still needs further optimizations. For this section, refer to the final RTL, which contains many optimizations. Compile the design in the `final` directory to see the effects of all the changes in this section.

Depending on seed variation, results may differ compared to the screenshots in this section.

1. Open the design in Intel Quartus Prime Pro Edition: click **File > Open Project** and open the `top.qpf` project file from the `intermediate` directory.
2. Under **Design Assistant Rule Settings**, turn on **Enable Design Assistant execution during compilation**.
3. Compile the design.
4. Analyze the Design Assistant results.
   The report shows fewer rule violations. The missing input delay is on an asynchronous pin and is safe.

Figure 6. Design Assistant (Signoff) Results

Design Assistant (Signoff) Results - 2 of 30 Rules Failed

Show: Visible | Hide | <<Filter>>

	Rule	Severity	Violations
1	TMC-20011 - Missing Input Delay	High	1
2	TMC-20204 - Setup-Failing Path Endpoints with Retiming Restrictions	Medium	889
3	CDC-50001 - 1-Bit Asynchronous Transfer Not Synchronized	High	0
4	CDC-50002 - 1-Bit Asynchronous Transfer Missing Timing Constraint	High	0
5	CDC-50003 - CE-Type CDC Missing Skew Constraints	High	0
6	CLK-30026 - Missing Clock Assignment	High	0

5. Observe the failing rules during the fitter plan stage. These failing rules point to high fan-out and congestion issues.

Figure 7. Fitter Plan Stage Failing Rules

Design Assistant Results - 4 of 12 Rules Failed

Show: Visible | Hide | <<Filter>>

	Rule	Severity	Violati
1	HRR-10004 - High fanout non-global (Long Runtime)	Low	9
2	TMC-20551 - Automatically-Discovered ...date Likely Requires More Duplication	Low	9
3	TMC-20602 - Registers with High Timing Path Endpoint Tension	Low	9
4	TMC-20601 - Registers with High Immediate Fan-Out Tension	Low	1
5	HRR-10003 - Registers on high fanout non-clock globals (Long Runtime)	Medium	0
6	FLP-40001 - Congested Placement Region	Low	0

6. Reduce the high fan-out nets in the design (refer to *Reducing High Fan-out Nets*). Design Assistant can identify high fan-out nets that are candidates for duplication.

Figure 8. TMC-20601 Registers with High Immediate Fan-out

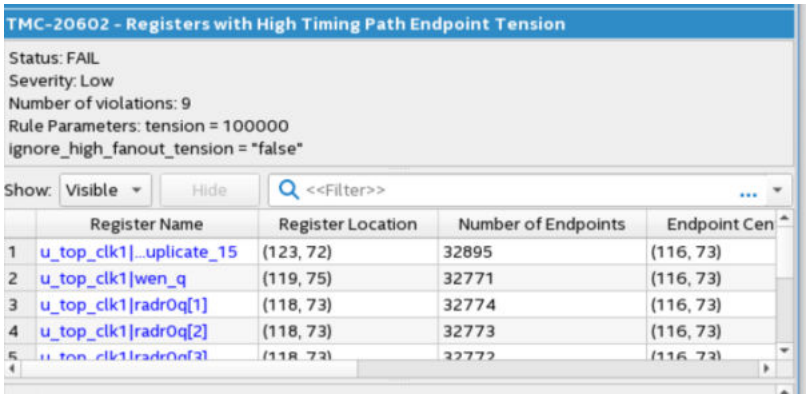
TMC-20601 - Registers with High Immediate Fan-Out Tension

Status: FAIL
Severity: Low
Number of violations: 1
Rule Parameters: tension = 100000

Show: Visible | Hide | <<Filter>>

	Register Name	Register Location	Number of Fan-Outs	Fan-Out Centroid	A
1	u_top_clk1|wen_q	(119, 75)	32771	(116, 73)	45

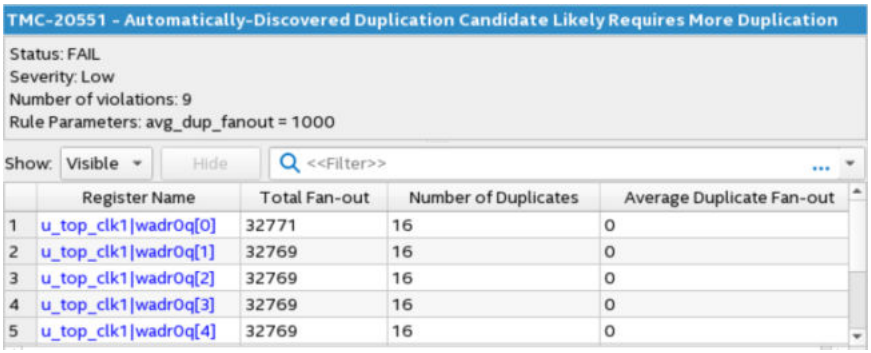
Figure 9. TMC-20602 Registers with High Timing Path Endpoint



a. Divide the memory into stripes to allow for duplication of control signals.

Figure 10. Memory Control Signals

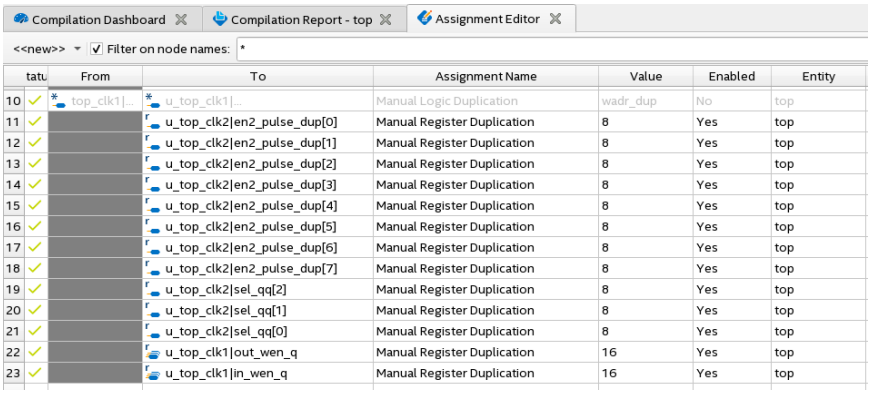
TMC-20551 identifies some control signals that may need further duplication. It is the write address of a large memory.



b. Explicitly duplicate registers in the RTL.

c. Apply the **Manual Register Duplication** assignments on some of these high fan-out registers.

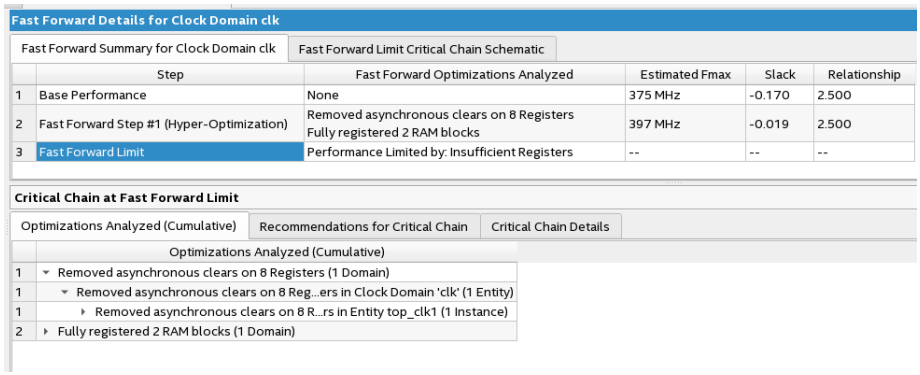
Figure 11. Manual Register Duplication Assignments



d. Duplicate the synchronous reset signal to reduce fan-out.

7. Observe rule TMC-20204 shows the design setup-failing endpoints with retiming restrictions. Fast-forward compilation can provide recommendations.

Figure 12. Fast-forward Compilation



Refer to *Hyper-Timing Restrictions* for rules that indicate where hyper-retiming is limited. In the design, the timing endpoints are in M20Ks, which is a block that you cannot retime.

The final RTL removes asynchronous resets and creates a synchronous reset tree. It also adds additional pipeline stages throughout the design including the inputs to memories and on the synchronous reset tree.

8. Remove initial power-up conditions.

   The design has a signal that didn't reset but controls some other logic, which can lead to functional issues. Refer to the *Initial Power-Up Conditions Rules* for rules that detect initial conditions in the design.

   RES-30132 detects registers that may not be reset. Review the results to see if design reset the signal and determine if it should be reset.

Figure 13. RES-30132

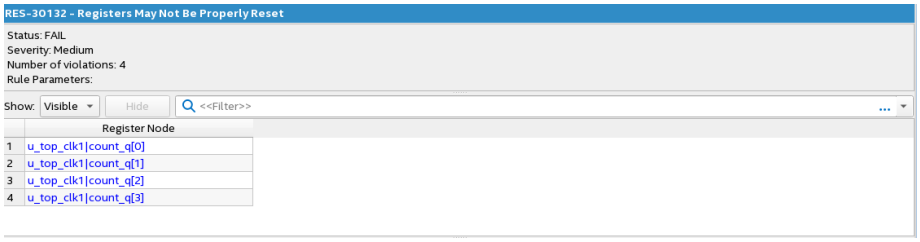


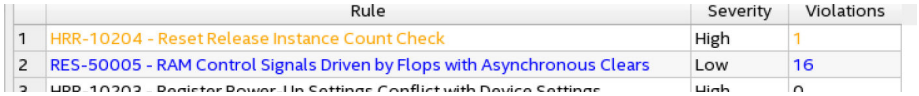
Figure 14. Example of Register to Reset

Can start at any value and should be reset.

```
always@(posedge clk)
begin
  count_q <= count_q + 1'd1;
  if(count_q == 4'hf)
    wen_q <= 1'b1;
  else
    wen_q <= 1'b0;
end
```

9. Add a Reset Release IP.

Figure 15. Reset Release



	Rule	Severity	Violations
1	HRR-10204 - Reset Release Instance Count Check	High	1
2	RES-50005 - RAM Control Signals Driven by Flops with Asynchronous Clears	Low	16
3	HRR-10203 - Register Power-Up Settings Conflict with Device Settings	High	0

The output of the Reset Release IP can gate internal clocks and resets to prevent race conditions. Refer to *Reset Release IP* for more information. The final RTL includes an instance of the reset release IP.

Refer to *Reset Release IP* for more information.

Related Information

- Intrinsic Margin Rules on page 18
- Hyper-Retimer Restrictions on page 24
- Initial Power-Up Conditions Rules on page 25
- Reset Release Intel FPGA IP on page 26

Document Revision History for AN 919: Improving Quality of Results with Design Assistant

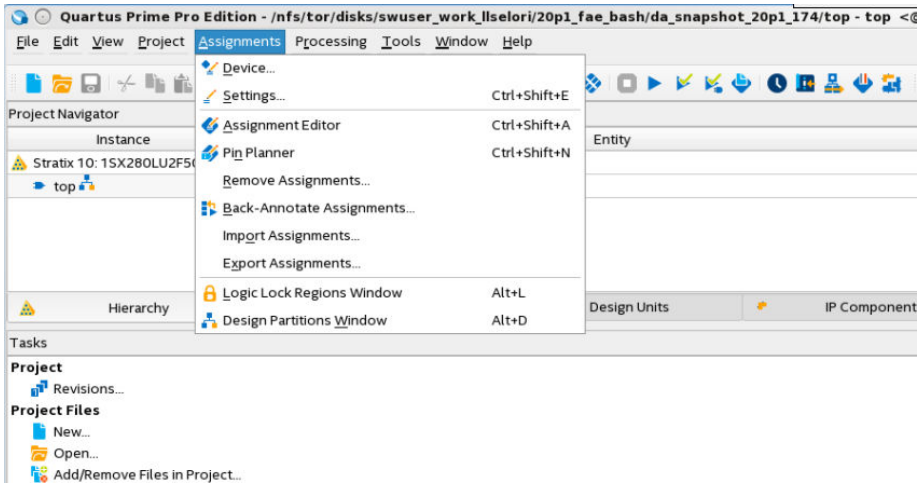
Document Version	Changes
2024.02.15	• Updated the product family name to Intel Agilex 7. • Corrected CDC-50003 rule in *Clock Domain Crossing and Reset Domain Crossing Rules*.
2021.05.04	Added link to `an919.zip`
2020.09.30	Initial release.

Appendix A: Running Design Assistant

Running Design Assistant

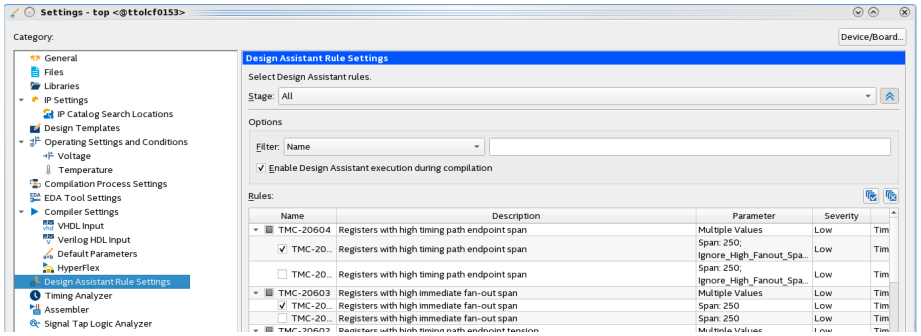
1. Click **Assignment ➤ Settings**.

Figure 16. Settings



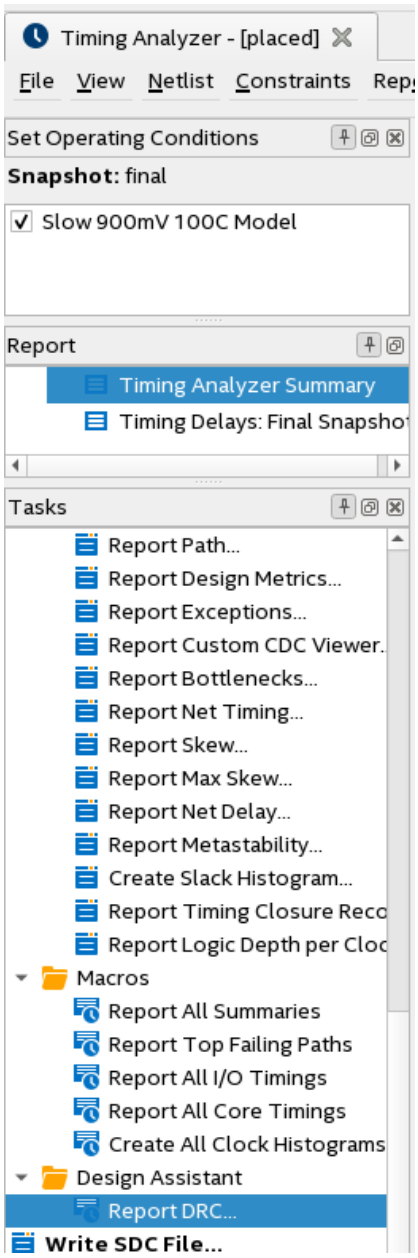
2. Go to **Design Assistant Rule Settings**, and turn on **Enable Design Assistant execution during compilation**.

Figure 17. Design Assistant Rules



3. To run Design Assistant under analysis mode, open Timing Analyzer and click **Report DRC**

Figure 18. Report DRC



4. By default, Design Assistant reports 500 violations, to see more violations, increment that number:

   ```
   set_global_assignment -name
   DESIGN_ASSISTANT_MAX_VIOLATIONS_PER_RULE <number>
   ```

Appendix B: Design Assistant Rules

Timing Closure Rules

Run on the planned snapshot and detect paths with extreme negative slacks. The rules catch any SDC issues that create excessive timing requirements as early as possible.Look at these rules to save a lot of compilation time on seeds that never pass timing.

- TMC-20001 – Timing Paths with Hold Slack Exceeding threshold
- TMC-20002 – Timing Paths with Removal Slack exceeding threshold
- TMC-20004 – Timing Paths with Setup Slack exceeding Threshold
- TMC-20005 – Timing Paths with Recovery Slack Exceeding Threshold

Table 1. High-Priority Timing Closure Rules

Design Assistant runs these timing closure rules on the final netlist, which helps identify incorrect SDC assignments, latches, or combinational loops in the design. These rules are in timing analyzer under Report DRC.

Rule	Description	Possible course of action
TMC-20022	Incomplete I/O delay assignment	Add the missing options to the delay assignment or modify the clock source
TMC-20019	Partial multicycle assignment	Verify that each setup multicycle assignment has a corresponding hold multicycle assignment and vice versa
TMC-20016	Invalid reference pin	Modify the -reference_pin option of the delay assignment to be the direct fan-out of the clock that you specify in the same assignment
TMC-20015	Inconsistent min/max delay	Modify the delay values to ensure that the min delay does not exceed the max delay
TMC-20014	Partial output delay	Verify that output delays have the rise-min, fall-min, rise-max, and fall-max specification
TMC-20013	Partial input delay	Verify that input delays have the rise-min, fall-min, rise-max, and fall-max specification
TMC-20012	Missing output delay	Verify that every output port has an output delay assignment
TMC-20011	Missing input delay	Verify that every input port has an input delay assignment
TMC-20052	Checks for inferred latches in synthesis	Remove any unintended inferred latches from the design
TMC-20018	Latches detected	Remove any unintended inferred latches from the design
TMC-20017	Loops detected	Remove the loops for your design.

Related Information

AN 903: Accelerating Timing Closure in Intel Quartus Prime Pro Edition

Clock Domain Crossing and Reset Domain Crossing Rules

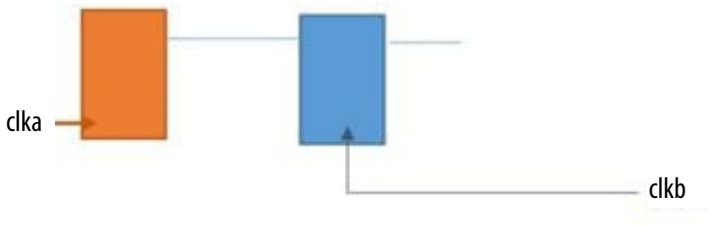
Incorrectly crossing clock-domains can result in functional failures and can be difficult to trace and debug. Additionally, incorrectly setting constraints when crossing clock-domains can result in long run times and impossible timing results. The following CDC rules detect if the datapath signals are going through correct synchronization logic and are being properly constrained.

CDC-50001 – single-bit asynchronous transfer is not synchronized

This rule checks that a single-bit asynchronous transfer has the proper synchronization circuitry.

Figure 19. Unsynchronized 1-bit Asynchronous Transfer

To prevent a CDC-50001 violation, the blue register in the following figure must be followed by at least one other register also latched by `clkb`.



In the example, the output of the blue register should feed another blue register to better protect against metastability.

CDC-50002 – single-bit asynchronous transfer is missing timing constraint

To prevent Intel Quartus Prime Pro Edition from analyzing the paths between clock domains, relax setup and hold requirements on this path. Use a `set_false_path`, `set_clock_groups` (asynchronous), or a large `set_max_delay` and a large negative `set_min_delay`.

CDC-50003 – CE-Type CDC Transfer with Insufficient Constraints

Violations of this rule identify a CDC transfer synchronized by DFFE (D Flip-Flop with enable) registers with insufficient timing constraints. Without proper constraints, all bits of such a topology may not latch on the same clock cycle.

Apply an instance assignment of `Synchronization Register Chain Length = 1` on the head of a CE-type CDC to prevent downstream registers from being treated as a synchronizer chain. If the transfer forms a multibit bus, apply a `set_max_skew` constraint on the bits of the bus to ensure that all bits latch on the same clock cycle. The value of the skew constraint must be equal to or lower than either the source or destination clock period, whichever is lower, by setting the `set_max_skew` constraint.

Also, for all CDC widths, apply a `set_net_delay` constraint or a `set_data_delay` constraint on the bits of the transfer to limit their allowable delay. The value of the net delay constraint or data delay constraint must be equal to or lower than the destination clock period, by setting the `set_net_delay` constraint.

RES-50001 and RES-50002 Reset Synchronization Rules

Deassert asynchronous resets synchronously, to prevent metastability. The following reset domain crossing rules help detect improper reset synchronization.

- RES-50001 – asynchronous reset is not synchronized
- RES-50002 – asynchronous reset is insufficiently synchronized

Example code for a reset synchronizer:

```
module safe_reset_sync (input external reset,
 input clock,
 output internal_reset);
logic q1, q2;
always@(posedge clock or negedge external_reset)
begin
 if(external_reset == 1'b0) begin
 q1 <= 1'b0;
 q2 <= 1'b0;
 end else begin
 q1 <= 1'b1;
 q2 <= q1;
 end

assign external_reset = q2;
endmodule
```

The output of this module can drive the resets of registers. The design needs constraints to cut the timing path from the asynchronous reset to the synchronizer reset pins.

RES-50003 – asynchronous reset is missing timing constraint

Similar to rule CDC-50002, the reset synchronizer should have timing constraints to prevent Timing Analyzer from analyzing these paths. Use `set_false_path`, `set_clock_groups` (asynchronous), or a `set_max_delay` larger than the latch clock period on transfer from the asynchronous reset source to the registers' `async` reset pins.

RES-50004 – multiple asynchronous resets within reset synchronizer chain

Ensure that all asynchronous resets in a reset synchronizer chain have a common source.

Related Information

- CDC-50001: 1-Bit Asynchronous Transfer Not Synchronized
- Intel Quartus Prime Pro Edition Help

Intrinsic Margin Rules

Identifies paths with impossible requirements. Any given datapath in the design has an intrinsic margin that doesn't depend on place and route but which the design's RTL and SDCs determine. The clock relationship between launch and latch clocks, clock uncertainty, and by `uTsu` and `Tco` time drive these intrinsic margin rules.

The Intrinsic margin rules help diagnose why a path may be failing setup timing. For example, too much clock skew, routing delay, or too many logic levels. Timing violations might not violate these rules, but if they do, they provide guidance on what you can do to fix them.

Table 2. Intrinsic Margin Rules

Rule	Description	Possible course of action
TMC-20200	Setup-Failing Paths with Impossible Requirements These paths have a tight clock relationship. Large differences in uTsu, uTco, and, or significant clock source uncertainty cause failures before any additional delay is added.	Fix SDC constraints. Ensure hard blocks are registered. Investigate clock sources.
TMC-20201	Setup-Failing Paths with High Clock Skew These paths have such high clock skew that they fail without any contribution from the datapath.	Apply clock region assignments. Redesign CDCs.
TMC-20202	Setup-Failing Paths with High Cell and Local Interconnect Delay These paths have such high cell delay that they fail without any contribution from the clock network or interconnect.	Reduce logic levels. Unblock retiming optimizations.
TMC-20203	Setup-Failing Paths with High Fabric Interconnect Delay The path has such high interconnect delay that it fails timing without any contribution from the clock network or datapath logic.	Reduce congestion. Apply floorplanning. Restructure RTL to ensure tighter packing.

Combinatorial Logic Levels

The number of combinatorial logic between registers can increase the path delay and limit f_{MAX}. To reach higher clock speeds, reduce the levels of combinatorial logic.

To reduce the levels of combinatorial logic:

- Add additional pipeline stages. With additional pipelining resources, the retimer can reduce the logic levels by balancing the register chain.
- Look for design optimizations that can reduce logic levels. For example, include precomputing values, or computing more in parallel.

Rule TMC-20010 detects paths levels above a given threshold on the worst timing paths.

Reducing High Fan-out Nets

High fan-out nets are difficult to place optimally and can reduce the fMAX of your design. They often lead to congestion and can result in long-path and short-path imbalances that limits retiming in critical chains. While these signals may not be critical, they can span large distances and warp the optimization of other paths around them.

1. Ensure you duplicate high fan-out driver registers:

- Use the DUPLICATE_REGISTER and DUPLICATE_HIERARCHY_DEPTH assignments for automated solutions or
- Edit the RTL to create duplicate copies.
- If you edit the RTL, apply the `preserve_syn_only` attribute to the duplicate registers, and assign the duplicates to individual instances in the fan-out hierarchy.

The compiler automatically promotes recognized high fan-out nets to the global clock network. It also makes a higher optimization effort during place and route stages to duplicate registers.

2. Set Intel Quartus Prime to automatically create duplicate register trees based on estimated physical proximity or based on hierarchy.
   a. Apply the following QSF assignment to duplicate registers based on estimated physical proximity:

      ```
      set_instance_assignment -name DUPLICATE_REGISTER -to
      <register name> <num duplicates>
      ```

      Where `register_name` is the name of the register being duplicated and `num_duplicates` is the number of duplicates to create, including the original.

      Intel Quartus Prime creates duplicates during the fitter stage and assigns fan-outs based on early estimates of physical distance from their destinations.

   b. Apply the following QSF assignment to perform duplications using hierarchy information:

      ```
      set_instance_assignment -name DUPLICATE_HIERARCHY_DEPTH -to
      <register_name> <level_number>
      ```

      Where:

      - Register name is the last register in the chain that fans out to multiple hierarchies
      - Level number is the number of registers in the chain to duplicate.

3. Typically, set up a synchronous reset tree. Resets are often a significant source of high fan-out nets. The registers in the chain must satisfy the following conditions to be included in duplication:
   a. Only another register must feed registers.
   b. Combinatorial logic must not feed registers.
   c. Registers must not be part of a synchronizer chain.
   d. Registers must not have any secondary signals.
   e. Registers must not have a `preserve` attribute or a PRESERVE_REGISTER assignment.
   f. All registers in the chain except the last one must have only one fan-out.

Duplication Rules

The following rules detect an issue with hierarchical tree duplication:

- TMC-20500 – Hierarchical Tree Duplication was Shallower Than Possible
- TMC-20501 – Hierarchical Tree Duplication was Shallower than Requested

Refer to the *Intel Quartus Prime Help* for more information about these rules.

If you are familiar with the design, you can make duplication decisions. Duplicate the register directly in the RTL, using `dont_merge` or `preserve_syn_only` to prevent synthesis from optimizing them away. For example, duplicating high fan-out broadcast signals by module.

The following are high fan-out rules:

- HRR-10115 – Nets with fan out exceeding threshold
- HRR-10003 – Registers with high fan-out non-globals
- TMC-20051 – High fan-out net drives RAM control signals.

To reduce fan-out on a RAM control signal, divide the memory into smaller chunks and send each duplicated control signal to its own memory. This technique works on wide memories.

Figure 20. A 256x2048 Memory with Large Fan-out on a Write Enable

Figure 21. Duplicated Write Enable and Split Memory to Reduce Fan-out

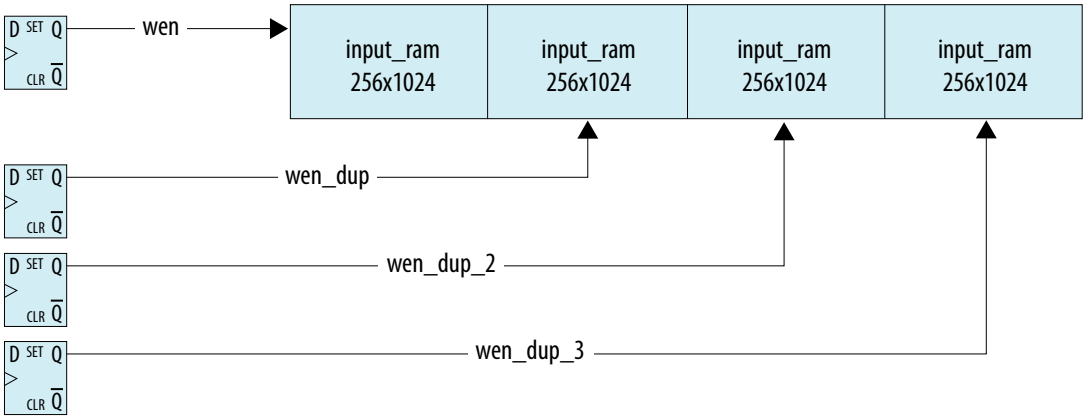


Table 3. Duplication Rejection Rules

The following rules report if the tool rejects duplication or if the design requires further duplication. The tool can reject duplication, for a placement constraint or for other connectivity reasons. You can identify issues with duplication in the planned netlist.

Rule	Description
TMC-20550 – Duplicate Candidate Rejected for Placement Constraint	Registers with a tight placement constraint such as logic lock region, clock region, or location assignments cannot be automatically duplicated. Relax the constraint to include the register's fan-outs or use the duplication techniques.
TMC-20551 – Automatically Discovered Duplication Candidates Likely Requires More Duplication	Intel Quartus Prime duplicates candidate registers automatically. If fan out is still large, apply further duplication techniques to reduce fan-out further.

Rule TMC-20552 detects rejected duplication candidates. Rejection reasons include:

- Registers drive global signals or clock signals.
- Registers have timing assignments or exceptions applied to them.
- Registers have a `preserve` attribute or a PRESERVE_REGISTER assignment.
- Registers are marked as `don't touch`.
- Registers drive or are driven by other partitions.

Figure 22. Duplication Summary

The fitter place stage produces a duplication summary, which shows duplication candidates and their duplication status.

Fitter Duplication Summary

Show: Visible | Hide | <<Filter>>

	Node Name	Candidate Reason	Duplication Status	Total fan-out	Number of Duplicates	Average Du
1	u_top_clk1|wadr0q[7]	Auto: High Fan-out	Rejected: Synchronization register	32769	1	32769.00
2	u_top_clk1|wadr0q[6]	Auto: High Fan-out	Rejected: Synchronization register	32769	1	32769.00
3	u_top_clk1|wadr0q[5]	Auto: High Fan-out	Rejected: Synchronization register	32769	1	32769.00
4	u_top_clk1|wadr0q[4]	Auto: High Fan-out	Rejected: Synchronization register	32769	1	32769.00
5	u_top_clk1|wadr0q[3]	Auto: High Fan-out	Rejected: Synchronization register	32769	1	32769.00
6	u_top_clk1|wadr0q[2]	Auto: High Fan-out	Rejected: Synchronization register	32769	1	32769.00
7	u_top_clk1|wadr0q[1]	Auto: High Fan-out	Rejected: Synchronization register	32769	1	32769.00
8	u_top_clk1|wadr0q[0]	Auto: High Fan-out	Rejected: Synchronization register	32769	1	32769.00
9	u_top_clk1|radr0q[7]	Auto: High Fan-out	Rejected: Synchronization register	32769	1	32769.00
10	u_top_clk1|radr0q[6]	Auto: High Fan-out	Rejected: Synchronization register	32769	1	32769.00
11	u_top_clk1|radr0q[5]	Auto: High Fan-out	Rejected: Synchronization register	32769	1	32769.00
12	u_top_clk1|radr0q[4]	Auto: High Fan-out	Rejected: Synchronization register	32769	1	32769.00
13	u_top_clk1|radr0q[3]	Auto: High Fan-out	Rejected: Synchronization register	32769	1	32769.00
14	u_top_clk1|radr0q[2]	Auto: High Fan-out	Rejected: Synchronization register	32769	1	32769.00

Related Information

- TMC-20500 – Hierarchical Tree Duplication was Shallower Than Possible
- Intel Quartus Prime Pro Edition Help

Tension and Span Rules

The tension and span rules help identify areas of the design that may be difficult for the fitter to place, even if they are meeting timing.

These rules identify registers with sinks that are pulling it in various directions, making it difficult to find an optimal placement.

Registers that fail these rules are good candidates for register duplication. They are low priority rules. Your design can still be meeting timing even if these rules are violating, but they help identify the areas of the design that can contribute to congestion. When looking for places for further optimization, these are good candidates.

Figure 23. Tension

Tension is the sum over each sink (either immediate fanout or timing endpoint, depending on the rule) of the distance of the sink from the centroid of all the sinks.

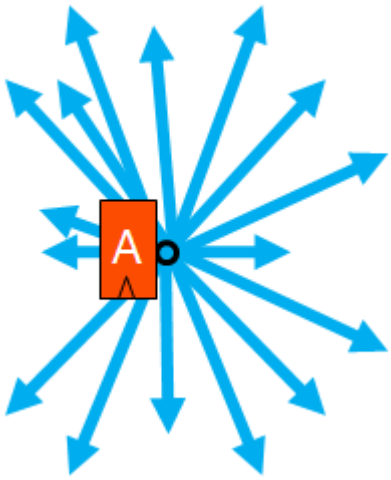
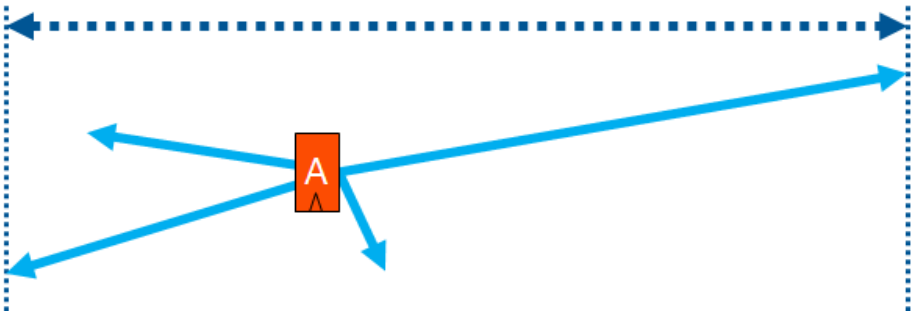


Figure 24. Span

Span is the maximum one-dimensional delta between the left-bottom-most sink with the right-top-most sink.

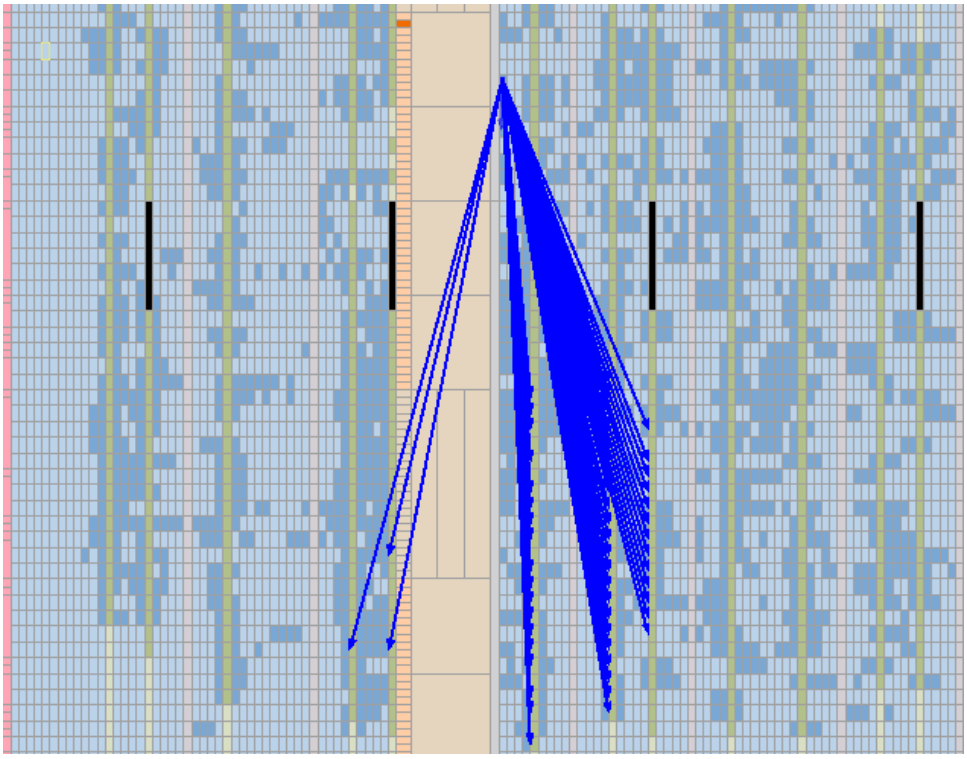


The rules are:

- TMC-20601 – Registers with High Fan-out Tension
- TMC-20602 – Registers with High Endpoint Tension
- TMC-20603 – Registers with High Fan-out Span
- TMC-20604 – Registers with High Endpoint Span

Duplicating these signals where possible allows Intel Quartus Prime to find a better solution.

Figure 25. High Tension Fanout Signal



Hyper-Retimer Restrictions

The Hyper-Retimer balances register chains by retiming ALM registers into Hyper-Registers in the routing fabric.

The Design Assistant rules that identify retiming restrictions prevent the Hyper-Retimer from making optimizations in Intel Agilex™ 7 and Intel Stratix 10 devices, thus limiting performance. Such restrictions include asynchronous resets, high fan-out nets, timing exceptions, the preserve register attributes, and initial conditions. The Design Assistant rules are:

- TMC-20204 – nodes with retiming restrictions that may restrict retiming
- TMC-20205 – nodes with initial conditions that may restrict retiming
- HRR-10003 – registers with high fan-out non-globals

Note: TMC-20204 and TMC-20205 only look at the worst setup-failing paths in the design. They do not report on retiming restrictions or initial conditions in off-critical logic.

Fast forward compile identifies the critical chain and points out the next steps to take to remove the retiming restrictions, if possible. For more information about hyper-retiming restrictions, refer to the *Intel Hyperflex Architecture High-Performance Design Handbook*.

Related Information

Intel Hyperflex Architecture High-Performance Design Handbook

Initial Power-Up Conditions Rules

The initial condition of the design at power-up represents the state of the design at clock cycle 0. This state is transitional rather than functional because the design cannot return to this condition.

Table 4. Initial Conditions Rules

The following Design Assistant rules identify initial conditions in the design and provide you guidance.

Rule	Description	Possible course of action
HRR-10203	Catches registers whose initial conditions drop because of IGNORE_REGISTER_POWER_UP_INITIALIZATION ON	Verify that the design is still functionally correct without these initial conditions.
RES-30132	Registers might not be properly reset	Check to see if you need these registers.
RES-30133	Catches memories that may have spurious writes because of initial conditions	Check to see that these registers are reset.
HRR-10201	Power-up don't care synthesis setting might prevent retiming	Remove power-up don't care setting
TMC-20205	Setup-failing path endpoints with explicit power-up states that might restrict retiming	Use QSFs to ignore initial conditions Remove initial conditions and use reset.

In the Design Assistant example design, `count_q` is not reset, which triggers rule RES-30132. Do not assume that Intel Quartus Prime sets `count_q` to 0 at power-up.

```
 always@(posedge clk)
begin
 count_q <= count_q + 1'd1;
 if(count_q == 4'hf)
 wen_q <= 1'b1;
 else
 wen_q <= 1'b0;
 end
```

Applying a reset to `count_q` so that it is set to 0 at reset fixes the violation.

Intel Agilex 7 and Intel Stratix 10 devices don't power up uniformly across all sectors. They power-up sector by sector. Some parts of the device may have their initial conditions before other parts of the device, which can lead to race conditions or spurious writes to memory during power-up.

Related Information

About the Design Assistant Design Example on page 4

Removing Initial Power-Up Conditions

Intel recommends you do not use initial power-up conditions in your RTL. Always reset the design into a known state.

1. Explicitly specify initial conditions in the RTL:

```
reg q = 1'b1; // q has a default value of '1' always@(posedge clk) begin q
<= d; end
```

Registers without resets may also have initial conditions. By default, Intel Quartus Prime determines their FF power-up values automatically.

2. Turn this setting off and force Intel Quartus Prime to set all uninitialized registers to 0 at power up by using the following global assignment:

   ```
   set_global_assignment ALLOW_POWER_UP_DONT_CARE OFF
   ```

   Generally, Intel does not recommend this setting because it can hinder register retiming.

3. Instead of using a global assignment, target areas of the design to drop automatically generated initial conditions with:

   ```
   set_instance_assignment
   IGNORE_REGISTER_POWER_UP_INITIALIZATION ON -to <instance name>
   ```

Reset Release Intel FPGA IP

Generates a signal to indicate when the device configuration has finished. It is then safe to release reset throughout the device.

Designs can use this IP to gate clocks and write enables or synchronize resets. The Reset Release solves race conditions and spurious writes during power-up. Intel Stratix 10 and Intel Agilex 7 devices require one instance of the Reset Release IP in the design. HRR-10204 is run during synthesis and checks that there is exactly one instance of the Reset Release IP in the design.

Related Information

Including the Reset Release Intel FPGA IP in Your Design