ALL PROGRAMMABLE
ANY MEDIA
ANY MACHINE
5G
4K/8K
ANY STANDARD
ANY NETWORK
5G Wireless • SDN/NFV • Video/Vision • ADAS • Industrial IoT • Cloud Computing

XILINX ALL PROGRAMMABLE™

AVNET Memec SILICA An Avnet Company

ECOLE D'ELECTRONIQUE IN2P3 2016

20 Juin 2016

Grégory Donzel (XILINX FAE AVNET MEMEC SILICA)

Agenda

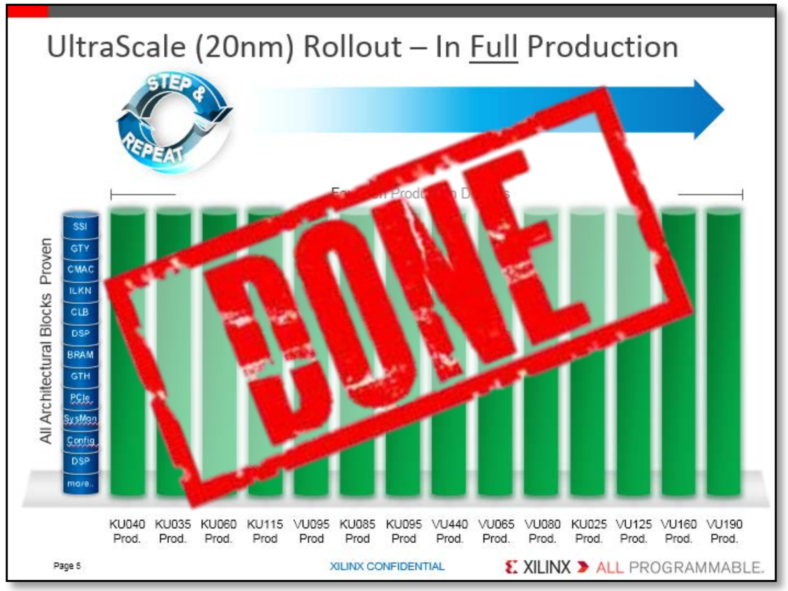
- XILINX Overview
  - A Generation Ahead from 28nm to 16nm
  - Tools and Methodology
- UFDM Guidelines for easier Timing Closure
- Setting Clean Timing Constraints for Predictable Static Timing Analysis
  - How to set "Clean" constraints?
  - Baselining a Design
  - Analyzing through the Design : `report_timing_summary, report_clock_interaction`, `report_cdc…`
- Last Miles Strategy: Tips and Tricks
- What's next?

Agenda

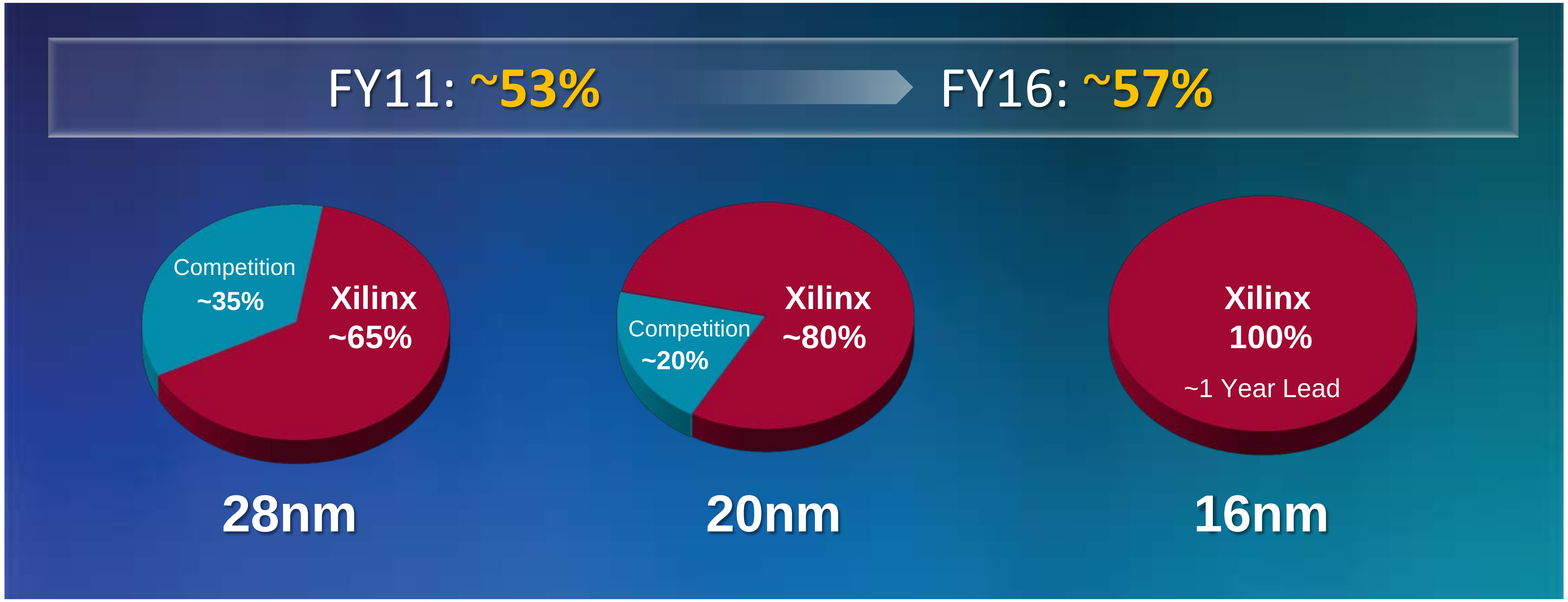
- XILINX Overview
  - A Generation Ahead from 28nm to 16nm
  - Tools and Methodology
- UFDM Guidelines for easier Timing Closure
- Setting Clean Timing Constraints for Predictable Static Timing Analysis
  - How to set “Clean” constraints?
  - Baselining a Design
  - Analyzing through the Design : `report_timing_summary`, `report_clock_interaction`, `report_cdc`…
- Last Miles Strategy: Tips and Tricks
- What’s next?

A Generation Ahead from 28nm to 16nm : All 20nm UltraScale Devices In Volume Production NOW!

- Production rollout of 20nm UltraScale™ FPGAs <u>complete</u>
  - All device, package, speed grade options available NOW
  - Production speedsfile support in Vivado® 2016.1
- High-end uncontested in the market place
- Mid-range ~1.5yr ahead of competition
- Calls to action
  - Leverage our lead at both 20nm and 16nm!!


UltraScale (20nm) Rollout – In Full Production

A Generation Ahead from 28nm to 16nm: *Market Share*



Note: All numbers derived from Altera and Xilinx Only
Source: Public Reports and Xilinx Estimates

Tools and Methodology: Tool Offering

Vivado Analyzer	SysGen For DSP	Vivado HLS	SDSoC
Now free	NL: $995 EF-DSP-PC-NL FL: $1295 EF-DSP-PC-FL	Now free	NL: $995 EF-SDSOC-NL FL: $1395 EF-SDSOC-FL

AVNET Memec SILICA An Avnet Company

XILINX ALL PROGRAMMABLE.

Agenda

- XILINX Overview
  - A Generation Ahead from 28nm to 16nm
  - Tools and Methodology
- UFDM Guidelines for easier Timing Closure
- Setting Clean Timing Constraints for Predictable Static Timing Analysis
  - How to set “Clean” constraints?
  - Baselining a Design
  - Analyzing through the Design : `report_timing_summary`, `report_clock_interaction`, `report_cdc`…
- Last Miles Strategy: Tips and Tricks
- What’s next?

Tools and Methodology: Quick Survey…

Who’s working with ISE?

– Support any FPGA family before 7-Series: The right and only tool to support Spartan-6, Virtex-6, etc…
– Latest supported version: ISE 14.7 (available since October 2013)
– ISE won’t be covered today

Who knows Documentation Navigator?

– Catalog View: Great tool to work with the latest versions of docs
– Design Hub & Checklists
– Can be installed standalone from http://www.xilinx.com/support/documentation-navigation/overview.html
– How to use?
  - http://www.xilinx.com/video/support/how-to-use-document-navigator.html

Who knows what UFDM means?

– UFDM = UltraFast Design Methodology
  - Vivado Design Suite methodology (UG949) – HDL flow (board to closure)
  - Embedded design methodology (UG1046) – Embedded flow (HW + SW)
  - High level design methodology (UG1197) – HLx flow (IPI + HLS)

UltraFAST

UltraFast Design Methodology Guide for the Vivado Design Suite

UG949 (v2014.1) April 2, 2014

UG949 XILINX

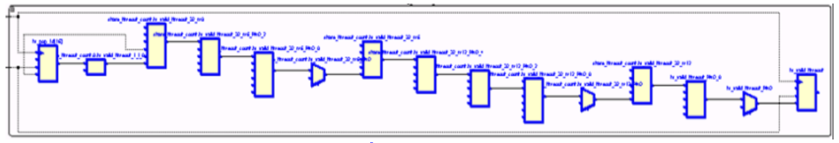
UG1046 XILINX

UG1197 XILINX

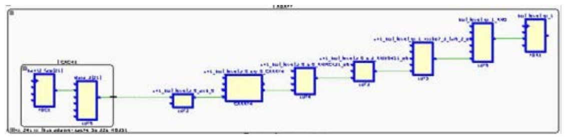
www.xilinx.com/ultrafast

Critical Path could be a Moving Target
Example from a Real Design

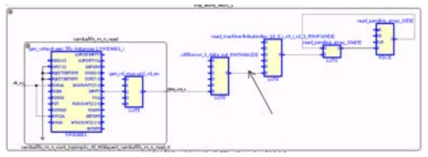
- **Post-synthesis estimates (the real problem)**
  – Worst path: 13 levels of logic


worst path: 4.3ns

- **Post-place**
  – Worst path: 7 levels
  – <u>Paths with 7-13 levels got placed locally</u>


worst path: 4.2ns

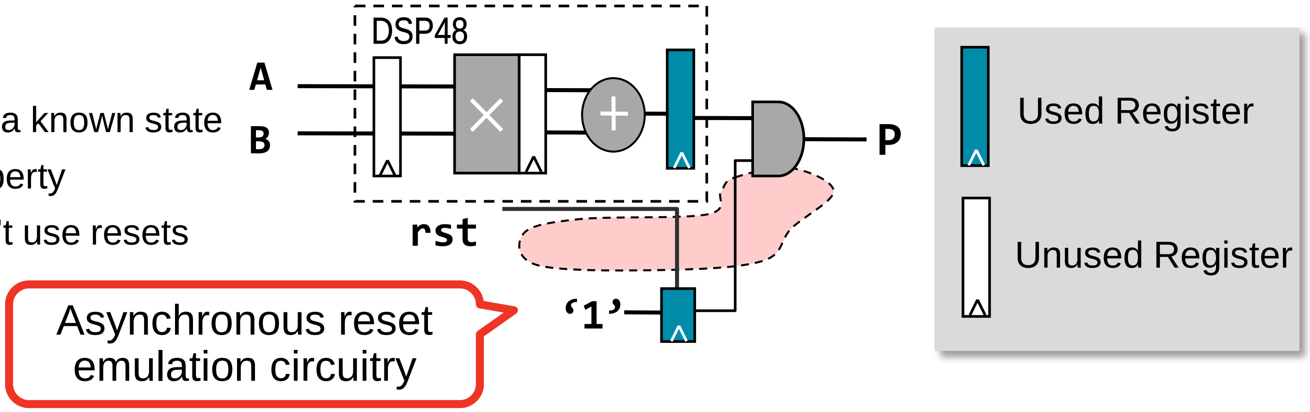
- **Post-route (the side-effect of the real problem)**
  – Worst Path: 4 levels of logic
  – <u>Paths with 5-13 levels got preferred routing</u>


worst path: 4.1ns

Analyze & Fix timing issues at early stages for faster timing convergence

Impact of HDL Coding Style

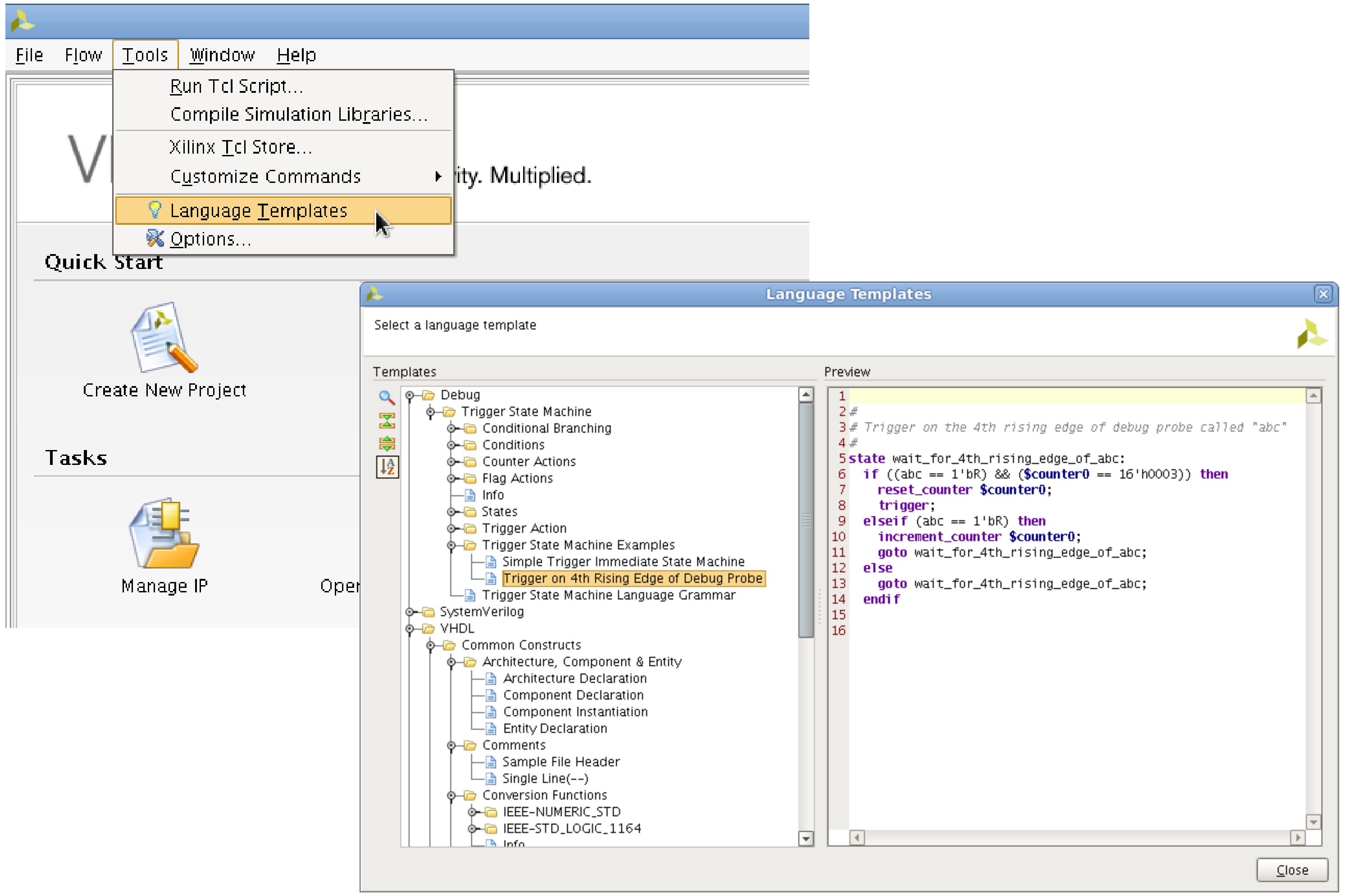
- **Block inference**
  - Follow recommended templates for RAM, DSP, LUTRAM, SRL inference
- **Pipeline your design to reduce levels of logic**
- **Avoid Reset**
  - No reset at all *(if possible)* is best: Xilinx devices boot in a known state
  - Default register value can be controlled via the INIT property
  - Dedicated shifters (SRLs) and RAM memory arrays don't use resets
  - WP272



- **Synchronous resets are preferred**
  - Allow packing of registers into dedicated RAM and DSP blocks
    - Active HIGH rather than active low reset
  - Tools have the option to implement reset in datapath (LUT)
- **Give more freedom to Synthesis**
  - Revisit attributes needed by other synthesis engines or older releases
  - Avoid KEEP, dont_touch, syn_preserve, max_fanout attributes…

Note sure about HDL Coding Style? Use Language Templates

- **Synthesis Templates**
  - BRAM, LUTRAM, ROM, SRL
  - Counter, MULT
  - FSM, Decoder, Encoder
  - …
- **Accessing templates in IDE**
  - Windows → Language Templates
  - Available as a standalone window
    - Tools → Language Templates
    - No project required

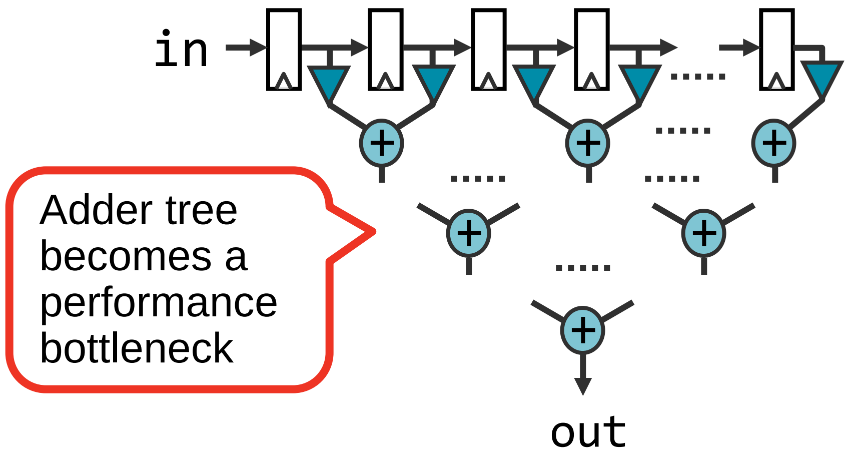


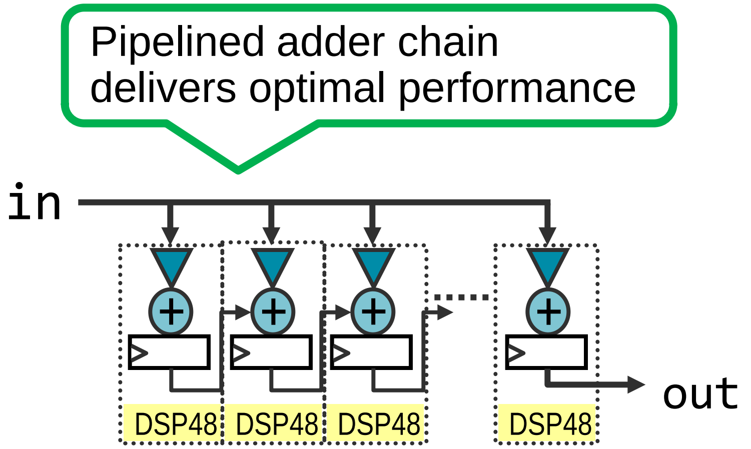
```

#
# Trigger on the 4th rising edge of debug probe called "abc"
#
state wait_for_4th_rising_edge_of_abc:
  if ((abc == 1'bR) && ($counter0 == 16'h0003)) then
    reset_counter $counter0;
    trigger;
  elseif (abc == 1'bR) then
    increment_counter $counter0;
    goto wait_for_4th_rising_edge_of_abc;
  else
    goto wait_for_4th_rising_edge_of_abc;
  endif


```

- **Drag and drop into Vivado text editor to use HDL templates**

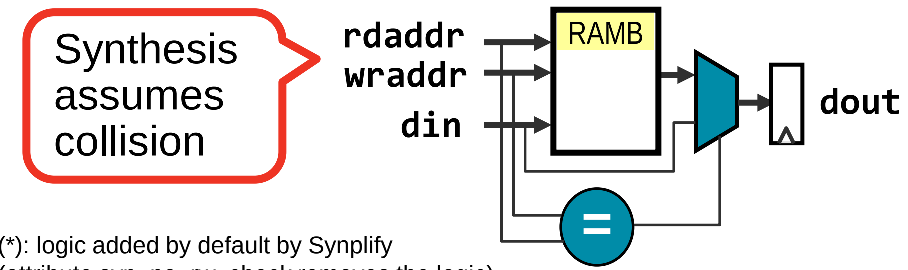
Using Language Templates: Coding to Match the Hardware

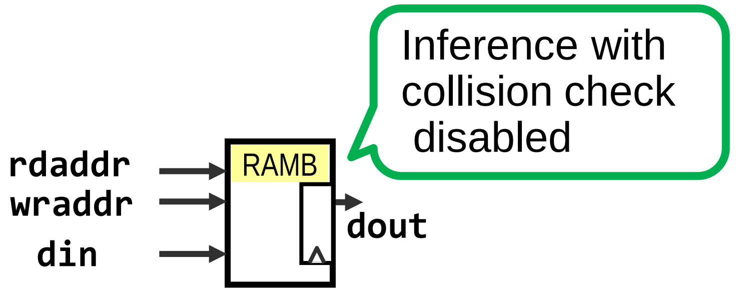
- **Leverage DSP block cascading capabilities**





- **Avoid Block RAM collision avoidance logic**(*)





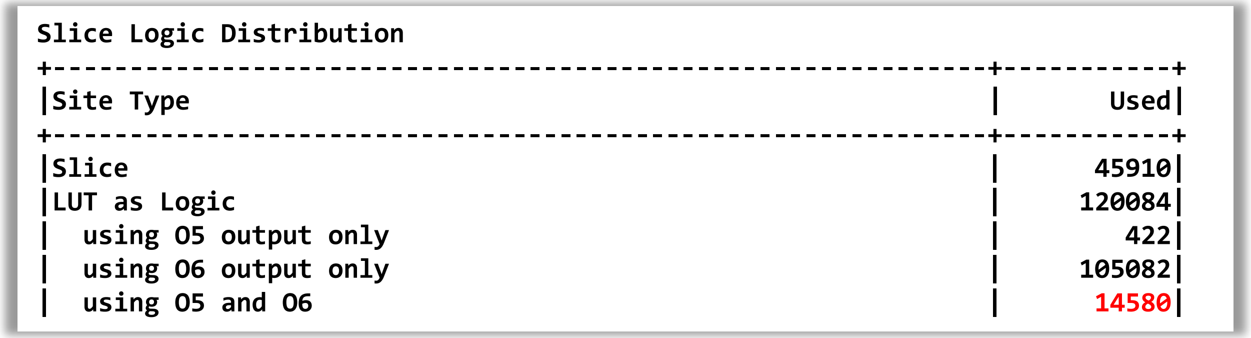
(*): logic added by default by Synplify
(attribute syn_no_rw_check removes the logic)

LUT Combining

- **LUT combining leverages the dual-output LUT (O5/O6)**
  - **Pro**: saves area
  - **Con**: could induce congestion

- **Tools behavior**
  - XST/Synplify combine by default, Vivado Synth has "soft" LC constraints
  - Implementation combines LUT based on utilization in place_design
  - High device or Pblock utilization will see more combined LUTs

- **Use `report_utilization` and look for LUTs with O5 and O6**



```
Slice Logic Distribution
+-------------------------------------------------------------+-----------+
|Site Type                                                    |       Used|
+-------------------------------------------------------------+-----------+
|Slice                                                        |      45910|
|LUT as Logic                                                 |     120084|
|   using O5 output only                                      |        422|
|   using O6 output only                                      |     105082|
|   using O5 and O6                                           |      14580|
```

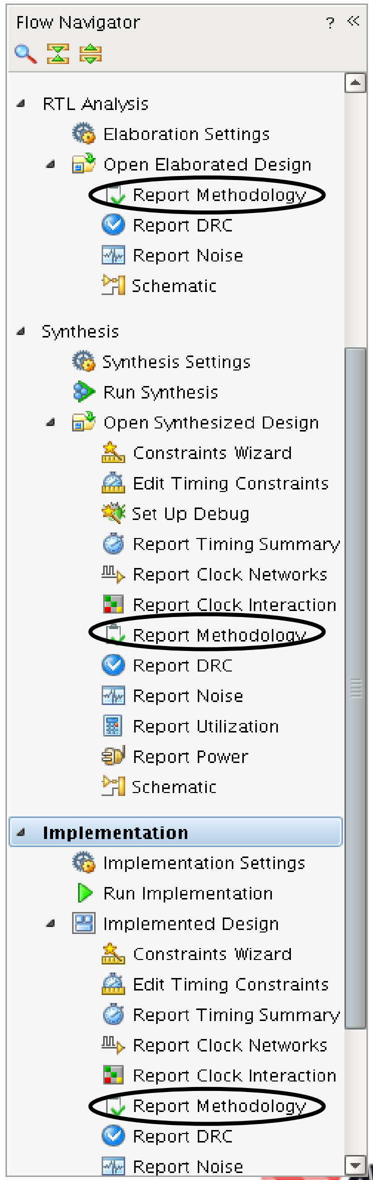
- **Guideline: If >15% of LUT use both O5 and O6, then**
  - Consider turning off LUT combining in synthesis

Gauging Other Design Metrics

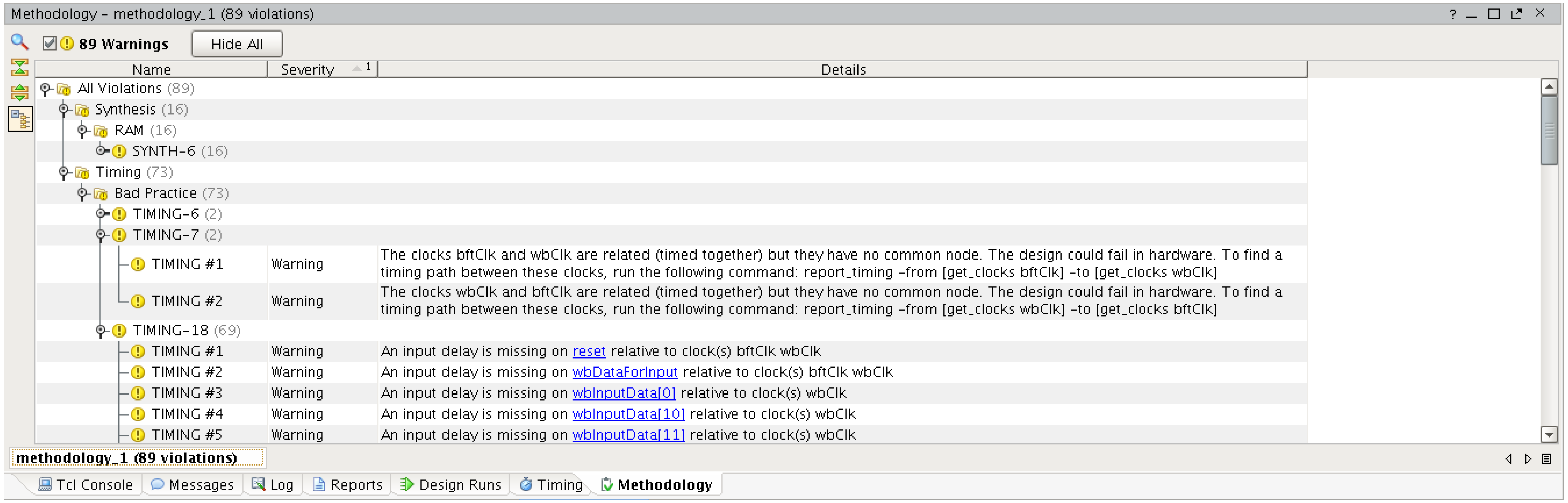
- **report_high_fanout_nets**
  - To reduce fanout on a net use…
    - max_fanout (Vivado synthesis and XST)
    - syn_maxfan (Synplify)
  - Use `phys_opt_design` for timing driven replication

- **report_control_sets**
  - Indicator of possible packing fragmentation and fitting issues
  - Run the `-verbose` option to generate a full list
  - Use Synplify's `syn_reduce_controlset_size` attribute for control
    - Default is 2, set it to 8 to eliminate most lowest fanout control sets

Ultrafast Methodology Checks



- “Report Methodology” added to the Flow Navigator
- New “Methodology” messages tab
- Replaces “Report DRC” methodology rule deck



Name	Severity	Details
All Violations (89)		
Synthesis (16)		
RAM (16)		
SYNTH-6 (16)		
Timing (73)		
Bad Practice (73)		
TIMING-6 (2)		
TIMING-7 (2)		
TIMING #1	Warning	The clocks bftClk and wbClk are related (timed together) but they have no common node. The design could fail in hardware. To find a timing path between these clocks, run the following command: report_timing -from [get_clocks bftClk] -to [get_clocks wbClk]
TIMING #2	Warning	The clocks wbClk and bftClk are related (timed together) but they have no common node. The design could fail in hardware. To find a timing path between these clocks, run the following command: report_timing -from [get_clocks wbClk] -to [get_clocks bftClk]
TIMING-18 (69)		
TIMING #1	Warning	An input delay is missing on reset relative to clock(s) bftClk wbClk
TIMING #2	Warning	An input delay is missing on wbDataForInput relative to clock(s) bftClk wbClk
TIMING #3	Warning	An input delay is missing on wbInputData[0] relative to clock(s) wbClk
TIMING #4	Warning	An input delay is missing on wbInputData[10] relative to clock(s) wbClk
TIMING #5	Warning	An input delay is missing on wbInputData[11] relative to clock(s) wbClk

Review and Resolve Critical Warnings

- **Vivado does not stop for Critical Warnings**
  - Enables fixing many issues at once
  - Bitstream generation will error with unresolved critical warnings
- **Critical warnings are serious design issues**
  - Invalid constraints or XDC syntax errors
  - Netlist or target objects not found or invalid
- **Address these warnings before moving forward**
  - Results of design analysis may be inaccurate
  - Critical Warnings may prevent design success

In a Word…

UG949

Agenda

- XILINX Overview
  - A Generation Ahead from 28nm to 16nm
  - Tools and Methodology
- UFDM Guidelines for easier Timing Closure
- Setting Clean Timing Constraints for Predictable Static Timing Analysis
  - How to set “Clean” constraints?
  - Baselining a Design
  - Analyzing through the Design : `report_timing_summary, report_clock_interaction`, `report_cdc…`
- Last Miles Strategy: Tips and Tricks
- What’s next?

Timing Constraints need to be “clean”

- **When constraints (clock, IO) are missing**
  - The corresponding paths are timed optimistically
  - No violation will be reported but design may not work on HW
- **When path are incorrectly constrained**
  - Runtime and optimization efforts will be spent on the wrong paths
  - Reported timing violations may not result in any issues on HW
- **When constraints create wrong HOLD violations**
  - May result in long runtime and SETUP violations
  - P&R fixes HOLD violations as #1 priority, because:
    - Designs with HOLD violations won’t work on HW
    - Designs with SETUP violations will work, but slower
- **No timing violations**
  - Setup/recovery (max analysis): WNS > 0ns and TNS = 0ns
  - Hold/removal (min analysis): WHS > 0ns and THS = 0ns

“Clean” Constraints for Rapid Timing Closure

- **Prioritize and close 1 step at a time**
- **Converge first at Synthesis (faster, higher impact), then in back-end**
- **Start with the simplest (baseline) constraint:**
  - Internal Fmax (flop-to-flop constraints) which is the problem 9/10 times
  - Define proper clock dependencies
- **Make sure the design & constraints are reasonable**
- **Analyze, get to root cause, then decide how to fix it**
  - Clock path vs. data path vs. interconnect delay vs. logic delay…
  - Add I/O constraints (with Vivado XDC templates) and redo…

Method to Create Good “Clean” Constraints

- **Create Constraints: Four Key Steps**
  1. Create clocks
  2. Define clocks interactions
  3. Set input and output delays
  4. Set timing exceptions

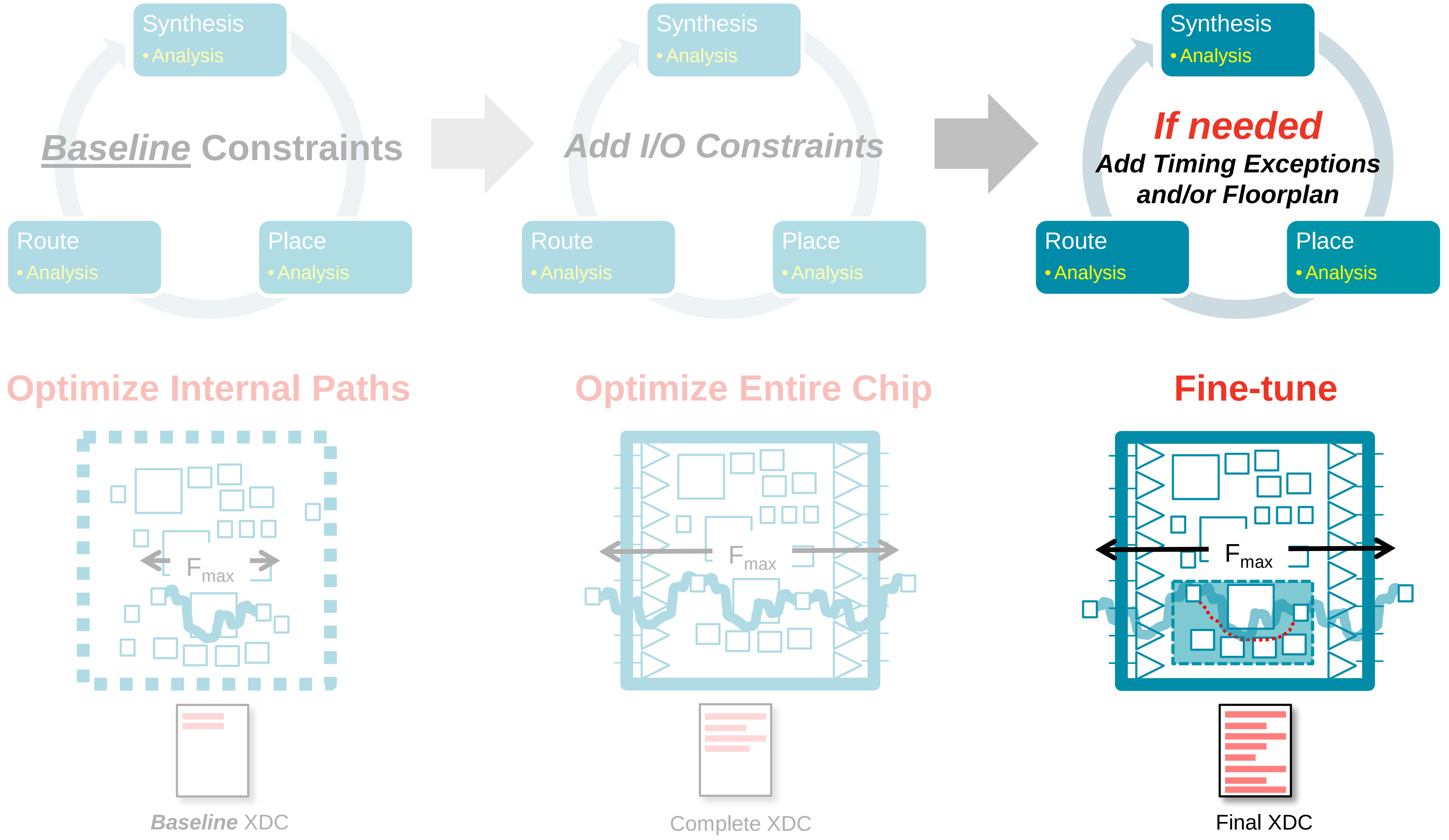
- **Use the Timing Constraints Wizard**

- **Validate Constraints at each step**
  - Monitor unconstrained objects
  - Validate timing

Note: Available via GUI and Tcl

- ✓ `report_timing_summary`
- ✓ `check_timing`
- ✓ `report_clocks` *(Note: Tcl only)*
- ✓ `report_clock_networks`
- ✓ `report_clock_interaction`
- ✓ `report_timing`

In a Drawing: Progressive Approach to Design Closure



Baselining Designs With VIVADO

- **Starting from a fully Synthesized Netlist**

- **2 Baseline Stages:**

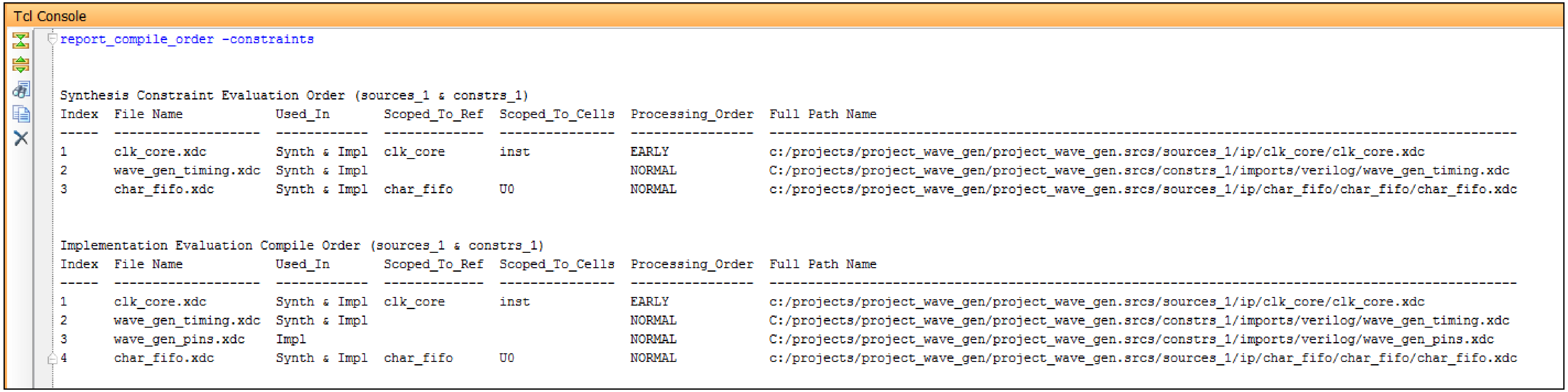
1. Constraint Development
   1.1. Add IP constraints
   1.2. Create clocks
      1.2.1. Use `create_clock`
      1.2.2. Run `report_clocks`
   1.3. Define clocks interactions

2. Implementation with **`report_timing_summary`**

Baseline Stage 1: Constraint Development
Add IP Timing Constraints

Do Not Forget To Include IP Timing Constraints

- Many cores have their own timing constraints that include important exceptions (PCIE, MIG, 2-clock distributed FIFOs…)
- Non-native IP: very easy to drop the IP constraints especially if customer only provides IP as .ngc netlist files
- Native IP: use report_compile_order –constraints to identify all constraint file sources



Tcl Console

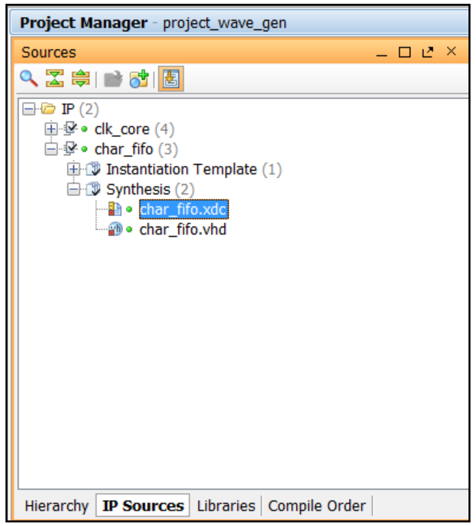
```
report_compile_order -constraints


Synthesis Constraint Evaluation Order (sources_1 & constrs_1)
Index  File Name            Used_In       Scoped_To_Ref  Scoped_To_Cells  Processing_Order  Full Path Name
-----  -------------------  ------------  -------------  ---------------  ----------------  ---------------------------------------------------------------------------------------------
1      clk_core.xdc         Synth & Impl  clk_core       inst             EARLY             c:/projects/project_wave_gen/project_wave_gen.srcs/sources_1/ip/clk_core/clk_core.xdc
2      wave_gen_timing.xdc  Synth & Impl                                  NORMAL            C:/projects/project_wave_gen/project_wave_gen.srcs/constrs_1/imports/verilog/wave_gen_timing.xdc
3      char_fifo.xdc        Synth & Impl  char_fifo      U0               NORMAL            c:/projects/project_wave_gen/project_wave_gen.srcs/sources_1/ip/char_fifo/char_fifo/char_fifo.xdc


Implementation Evaluation Compile Order (sources_1 & constrs_1)
Index  File Name            Used_In       Scoped_To_Ref  Scoped_To_Cells  Processing_Order  Full Path Name
-----  -------------------  ------------  -------------  ---------------  ----------------  ---------------------------------------------------------------------------------------------
1      clk_core.xdc         Synth & Impl  clk_core       inst             EARLY             c:/projects/project_wave_gen/project_wave_gen.srcs/sources_1/ip/clk_core/clk_core.xdc
2      wave_gen_timing.xdc  Synth & Impl                                  NORMAL            C:/projects/project_wave_gen/project_wave_gen.srcs/constrs_1/imports/verilog/wave_gen_timing.xdc
3      wave_gen_pins.xdc    Impl                                          NORMAL            C:/projects/project_wave_gen/project_wave_gen.srcs/constrs_1/imports/verilog/wave_gen_pins.xdc
4      char_fifo.xdc        Synth & Impl  char_fifo      U0               NORMAL            c:/projects/project_wave_gen/project_wave_gen.srcs/sources_1/ip/char_fifo/char_fifo/char_fifo.xdc
```

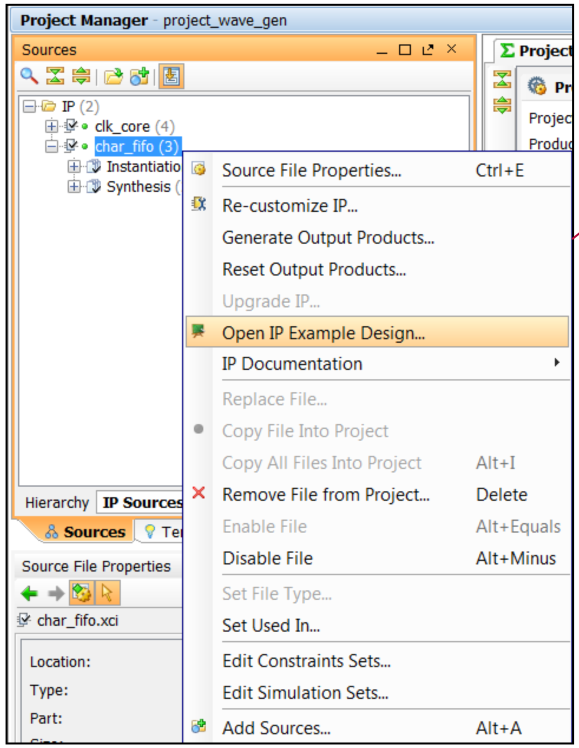
Baseline Stage 1: Constraint Development

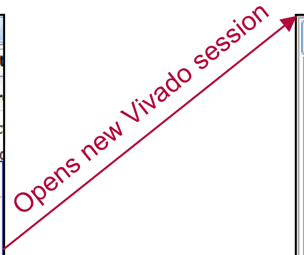
Add IP Timing Constraints

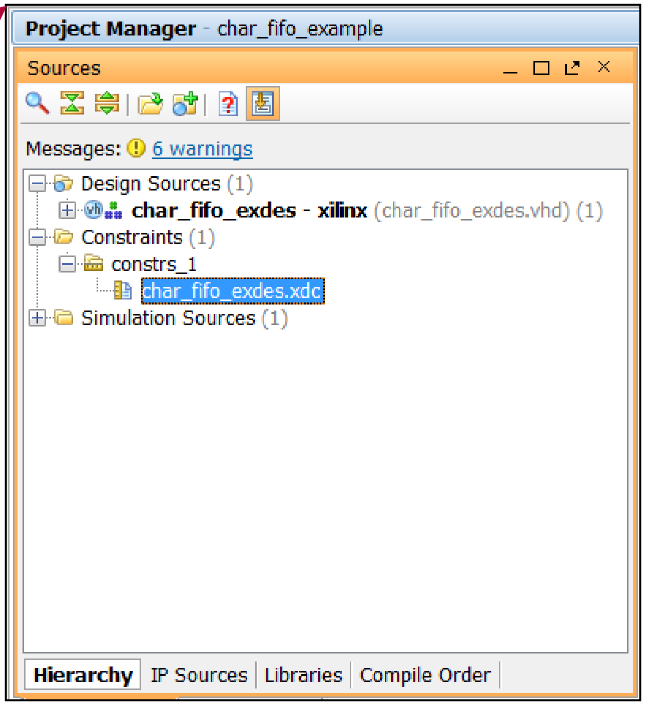
Do Not Forget To Include IP Timing Constraints

– Native IP: review BOTH xdc file that comes with core AND example project xdc for timing exceptions





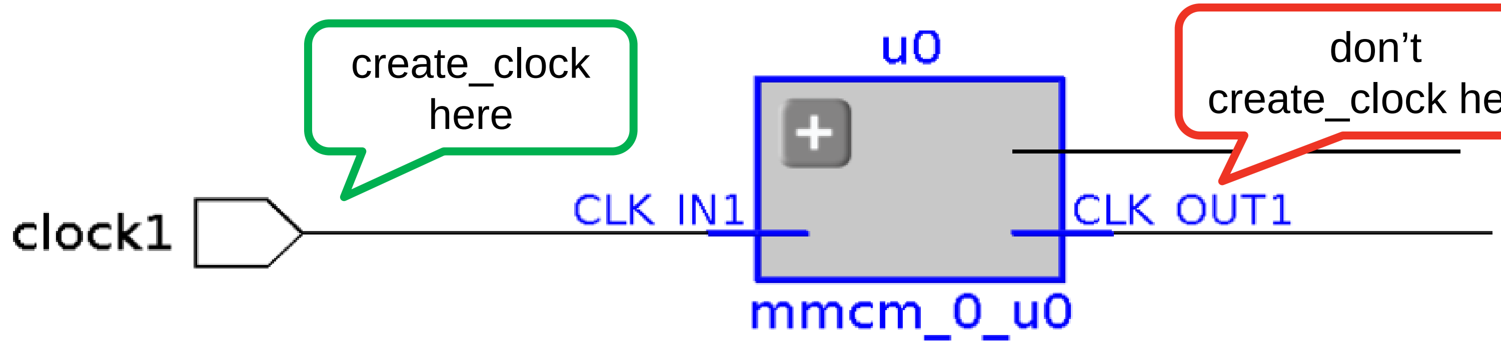




Baseline Stage 1: Constraint Development

Creating Clocks

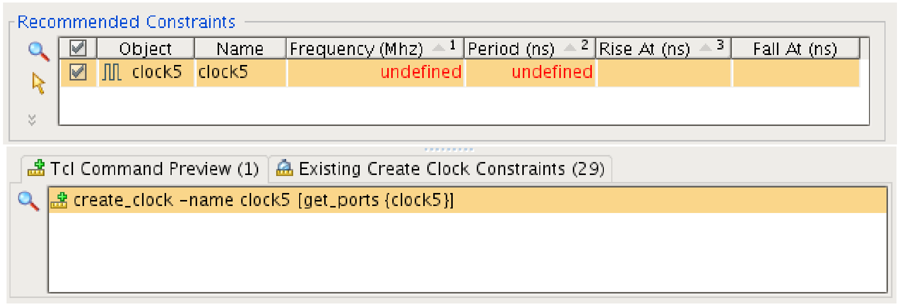
- **Clock Ground Rules…**
- **For SDC-based timers, clocks only exist if you create them**
  - Use `create_clock` for primary clocks
- **Clocks propagate <u>automatically</u> through clocking modules**
  - MMCM and PLL output clocks are automatically generated
  - Gigabit transceivers are not supported. Create them manually.



- **Use `create_generated_clock` for internal clocks *(if needed)***
- **All inter-clock paths are evaluated by default**

Baseline Stage 1: Constraint Development
Creating Clocks

Timing Constraint Wizard!


Recommended Constraints

☑	Object	Name	Frequency (Mhz)	Period (ns)	Rise At (ns)	Fall At (ns)
☑	clock5	clock5	undefined	undefined		

Tcl Command Preview (1) | Existing Create Clock Constraints (29)

```
create_clock -name clock5 [get_ports {clock5}]
```

- **Define primary clocks: `create_clocks`**
  - Create at top level port or GT OUTCLK pins
  - Run the design (synthesis) or open netlist design

- **Verify specified and automatically generated clocks: `report_clocks`**

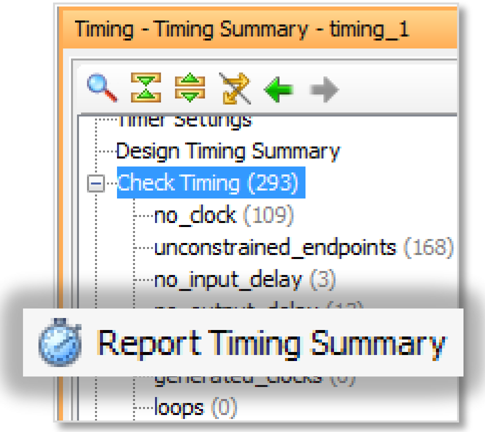
```
Attributes
  P: Propagated
  G: Generated

Clock          Period  Waveform        Attributes  Sources
sys_clk        10.000  {0.000 5.000}   P           {sys_clk}
pll0/clkout0   2.500   {0.000 1.250}   P,G         {pll0/plle2_adv_inst/CLKOUT0}
pll0/clkout1   10.000  {0.000 5.000}   P,G         {pll0/plle2_adv_inst/CLKOUT1}
```

  - To check constraint quality or to identify constraint issues: `check_timing`

- **Define remaining internal clocks: `create_generated_clocks`**
  - Find unconstrained generated clocks in *Check Timing* and *Report Clock Networks* reports

Baseline Stage 1: Constraint Development
Creating Clocks: Clock Constraint Validation Helpers

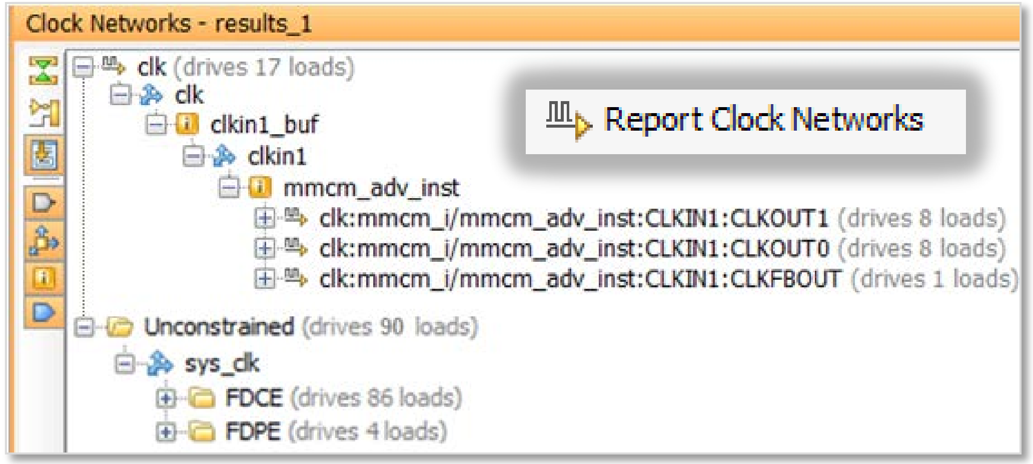
- **Review and monitor unconstrained objects**
  - To Check Progress:
    - `report_clocks`
    - `report_clock_networks`
    - `check_timing`
    - `report_timing_summary`: Check Timing section



Timing - Timing Summary - timing_1

- Timer Settings
- Design Timing Summary
- Check Timing (293)
  - no_clock (109)
  - unconstrained_endpoints (168)
  - no_input_delay (3)
  - generated_clocks (0)
  - loops (0)

- **Avoid Clock Skew**
  - Verify clock network topology
    - `report_clock_networks`
    - Beware of:
      - Gated clocks
      - Unconstrained clocks
      - Related clock from different MMCM

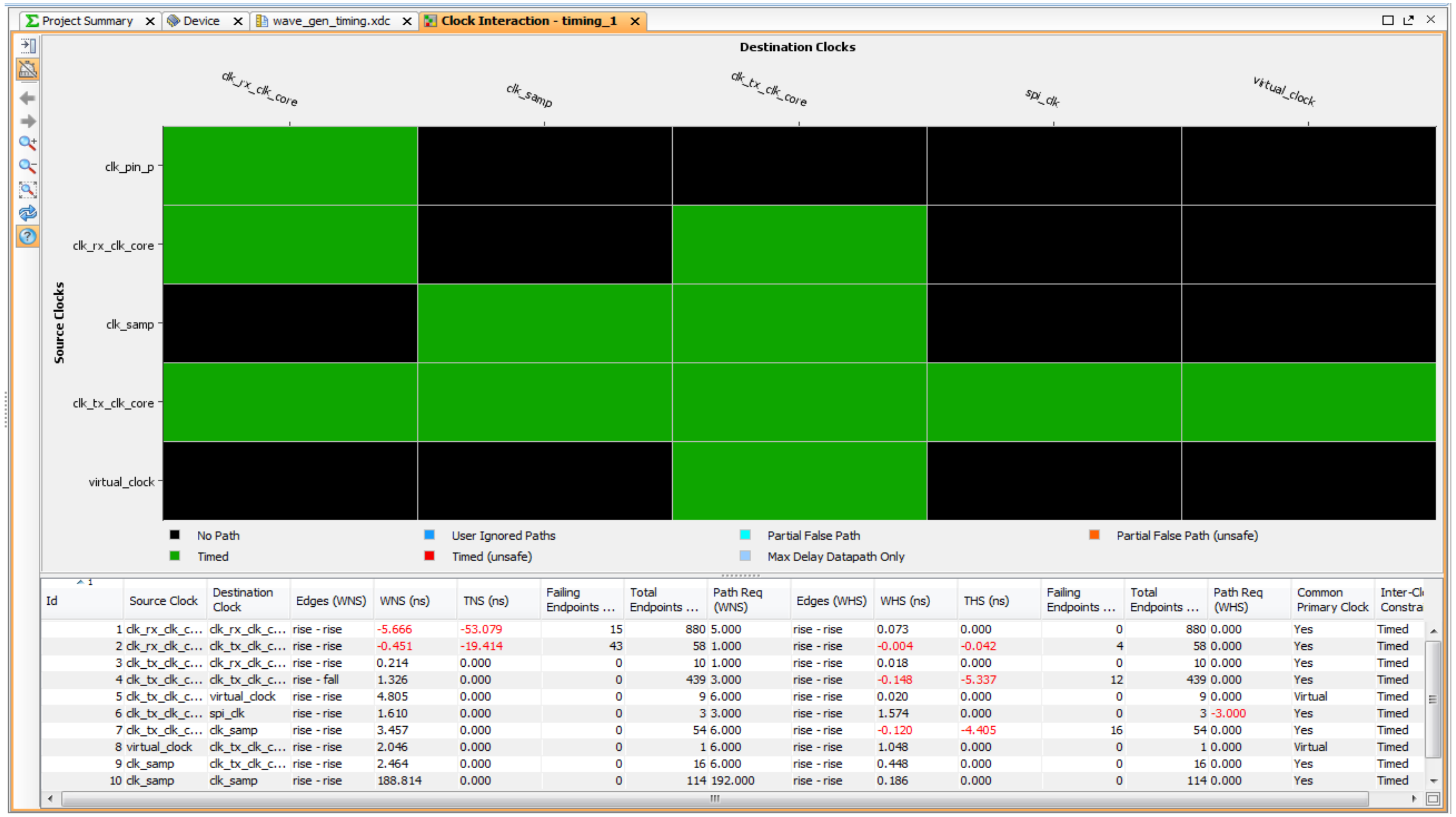


Clock Networks - results_1

```
clk (drives 17 loads)
  clk
    clkin1_buf
      clkin1
        mmcm_adv_inst
          clk:mmcm_i/mmcm_adv_inst:CLKIN1:CLKOUT1 (drives 8 loads)
          clk:mmcm_i/mmcm_adv_inst:CLKIN1:CLKOUT0 (drives 8 loads)
          clk:mmcm_i/mmcm_adv_inst:CLKIN1:CLKFBOUT (drives 1 loads)
Unconstrained (drives 90 loads)
  sys_clk
    FDCE (drives 86 loads)
    FDPE (drives 4 loads)
```

Report Clock Interaction after Creating Clocks in Baseline Stage 1

- Run `report_clock_interaction`



Destination Clocks

Source Clocks	clk_rx_clk_core	clk_samp	clk_tx_clk_core	spi_clk	virtual_clock
clk_pin_p	🟩	⬛	⬛	⬛	⬛
clk_rx_clk_core	🟩	⬛	🟩	⬛	⬛
clk_samp	⬛	🟩	🟩	⬛	⬛
clk_tx_clk_core	🟩	🟩	🟩	🟩	🟩
virtual_clock	⬛	⬛	🟩	⬛	⬛

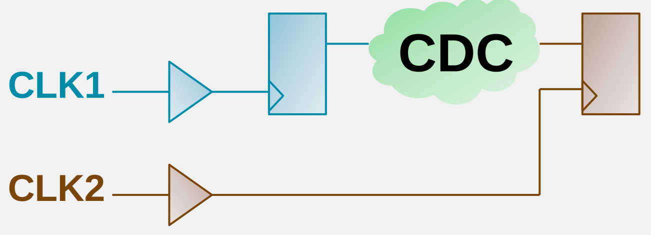
⬛ No Path 🟦 User Ignored Paths 🟦 Partial False Path 🟧 Partial False Path (unsafe)
🟩 Timed 🟥 Timed (unsafe) 🟦 Max Delay Datapath Only

Id	Source Clock	Destination Clock	Edges (WNS)	WNS (ns)	TNS (ns)	Failing Endpoints ...	Total Endpoints ...	Path Req (WNS)	Edges (WHS)	WHS (ns)	THS (ns)	Failing Endpoints ...	Total Endpoints ...	Path Req (WHS)	Common Primary Clock	Inter-Cl Constra
1	clk_rx_clk_c...	clk_rx_clk_c...	rise - rise	-5.666	-53.079	15	880	5.000	rise - rise	0.073	0.000	0	880	0.000	Yes	Timed
2	clk_rx_clk_c...	clk_tx_clk_c...	rise - rise	-0.451	-19.414	43	58	1.000	rise - rise	-0.004	-0.042	4	58	0.000	Yes	Timed
3	clk_tx_clk_c...	clk_rx_clk_c...	rise - rise	0.214	0.000	0	10	1.000	rise - rise	0.018	0.000	0	10	0.000	Yes	Timed
4	clk_tx_clk_c...	clk_tx_clk_c...	rise - fall	1.326	0.000	0	439	3.000	rise - rise	-0.148	-5.337	12	439	0.000	Yes	Timed
5	clk_tx_clk_c...	virtual_clock	rise - rise	4.805	0.000	0	9	6.000	rise - rise	0.020	0.000	0	9	0.000	Virtual	Timed
6	clk_tx_clk_c...	spi_clk	rise - rise	1.610	0.000	0	3	3.000	rise - rise	1.574	0.000	0	3	-3.000	Yes	Timed
7	clk_tx_clk_c...	clk_samp	rise - rise	3.457	0.000	0	54	6.000	rise - rise	-0.120	-4.405	16	54	0.000	Yes	Timed
8	virtual_clock	clk_tx_clk_c...	rise - rise	2.046	0.000	0	1	6.000	rise - rise	1.048	0.000	0	1	0.000	Virtual	Timed
9	clk_samp	clk_tx_clk_c...	rise - rise	2.464	0.000	0	16	6.000	rise - rise	0.448	0.000	0	16	0.000	Yes	Timed
10	clk_samp	clk_samp	rise - rise	188.814	0.000	0	114	192.000	rise - rise	0.186	0.000	0	114	0.000	Yes	Timed

Baseline Stage 1: Constraint Development

Clock Interaction

- **Clock Interaction Ground Rule…**
- **All inter-clock paths are evaluated by default**



CLK1 & CLK2

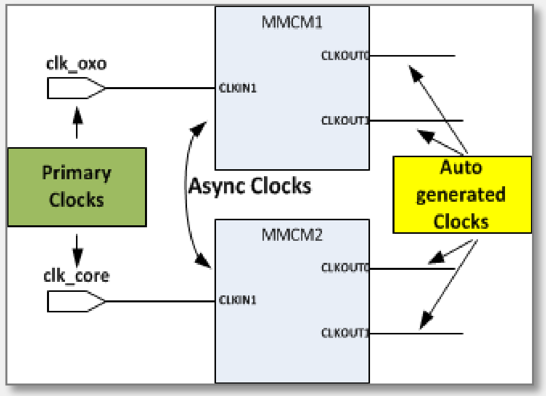
UCF: asynchronous SDC: synchronous

CDC

UCF: ignored SDC: Timed

Use `set_clock_groups` to

- make CLK1 & CLK2 asynchronous
- ignore CDC



```
# primary clocks
create_clock -name clk_oxo -period 10.00 [get_ports
clk_oxo]
create_clock -name clk_core -period 3.33 [get_ports
clk_core]

# set Asynchronous Clock Groups
  set_clock_groups -asynchronous \
   -group [… -include_generated_clocks clk_oxo] \
   -group [… -include_generated_clocks clk_core]
```

Baseline Stage 1: Constraint Development

Clock Interaction

Evaluate the clock interaction

– Use `report_clock_interaction`

BEWARE: All inter-clock paths are constrained by default!

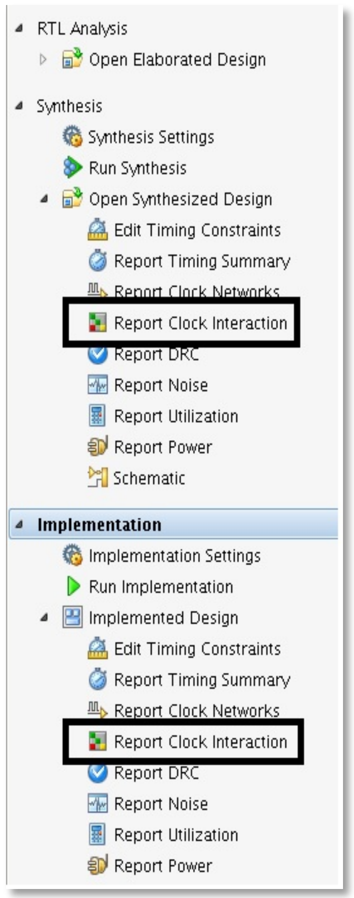
– Mark inter-clock paths (Clock Domain Crossing) as asynchronous

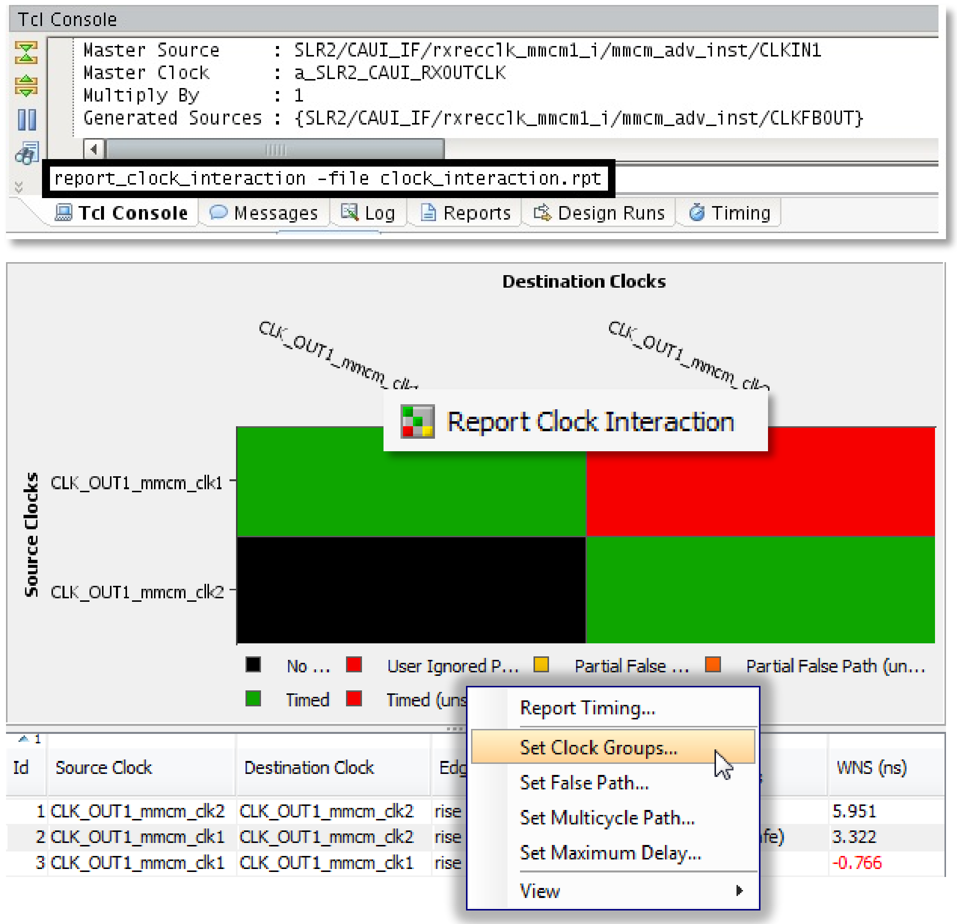
- Make sure you designed proper CDC synchronizers
- Use `set_clock_groups` (preferred method to set_false_path)

BEWARE: This overrides any set_max_delay constraints!

– Do you have unconstrained objects?

- Find out with `check_timing`





```
Master Source     : SLR2/CAUI_IF/rxrecclk_mmcm1_i/mmcm_adv_inst/CLKIN1
Master Clock      : a_SLR2_CAUI_RXOUTCLK
Multiply By       : 1
Generated Sources : {SLR2/CAUI_IF/rxrecclk_mmcm1_i/mmcm_adv_inst/CLKFBOUT}
```

```
report_clock_interaction -file clock_interaction.rpt
```

Id	Source Clock	Destination Clock	Edg		WNS (ns)
1	CLK_OUT1_mmcm_clk2	CLK_OUT1_mmcm_clk2	rise		5.951
2	CLK_OUT1_mmcm_clk1	CLK_OUT1_mmcm_clk2	rise	fe)	3.322
3	CLK_OUT1_mmcm_clk1	CLK_OUT1_mmcm_clk1	rise		-0.766

Baseline Stage 1: Constraint Development
Clock Interaction: Constraining Cross Clock Domains

Run Report Clock Domain Crossings

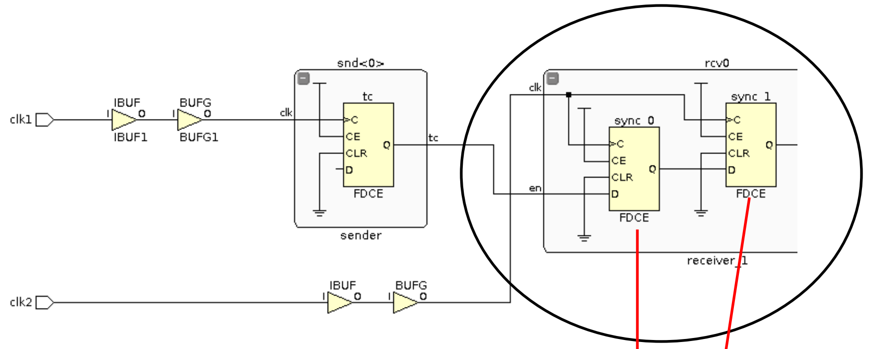
- If two clocks are not related, but paths exist between them, then there **must** be a clock crossing circuit between them.
- `report_cdc`
- Check CDC Topologies

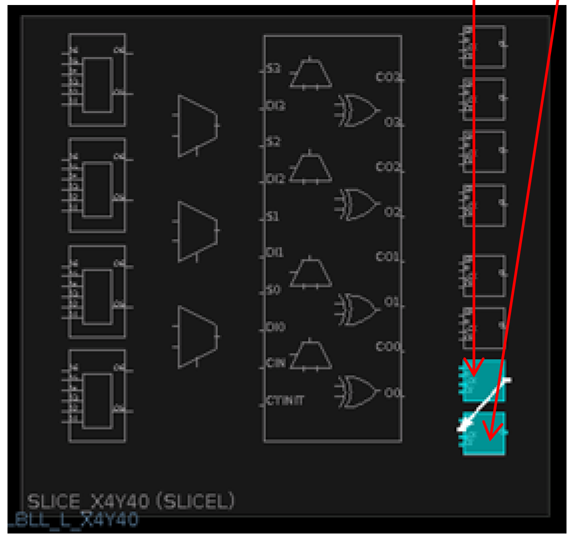
Use appropriate synchronizing techniques

- Asynchronous signals always cause some possibility that the system would fail.
- 2 **or more** register stages, for single bit
- FIFO for buses

Reducing Impact of Metastability and Maximize MTBF

- ASYNC_REG to place synchronizing flops in the same slice for best Mean Time Between Failures (MTBF)
- Usually comes with set_max_delay constraint





```
set_property ASYNC_REG TRUE \
[get_cells [list sync0_reg sync1_reg]]
```

Baseline Stage 1: Constraint Development

Clock Interaction: Constraining for Asynchronous CDC – Single Bit

Ignoring timing paths between individual clocks

set_clock_groups –asynchronous –group {clk1} –group {clk2}

This is equivalent to:

set_false_path –from [get_clocks clk1] –to [get_clocks clk2]

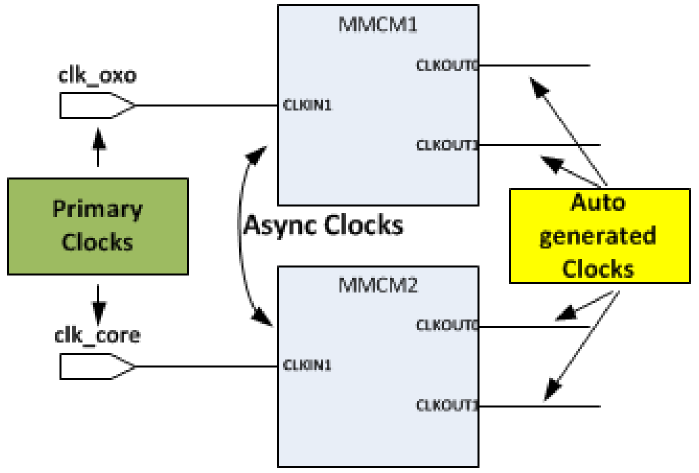
set_false_path –from [get_clocks clk2] –to [get_clocks clk1]

BEWARE: This overrides any set_max_delay constraints!

Ignoring timing paths between groups of clocks

```
# SDC create_clock for the two primary clocks
create_clock -name clk_oxo  -period 10 [get_ports clk_oxo]
create_clock -name clk_core -period 10 [get_ports clk_core]

# Set Asynchronous Clock Groups
set_clock_groups -asynchronous
-group [get_clocks –include_generated_clocks clk_oxo] \
-group [get_clocks –include_generated_clocks clk_core} ]
```

BEWARE: This overrides any set_max_delay constraints!



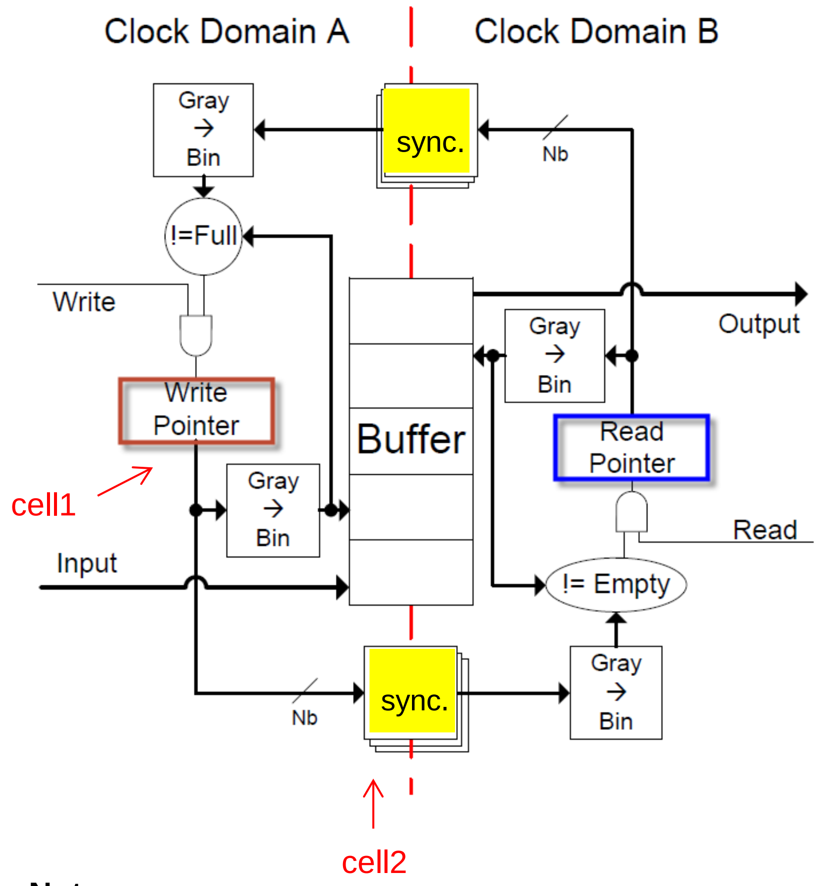
Baseline Stage 1: Constraint Development

Clock Interaction: Constraining for Asynchronous CDC – Bus

- **Use built-in hard FIFO (preferred)**
  - Circuit is designed for async transfers
  - Use set_clock_groups constraint
- **Use fabric Gray coded FIFO transfer**
  - Set timing requirement:

    ```
    set_max_delay $delay \
        –from [get_pins cell1/C]  \
        –to [get_pins cell2/D] \
        –datapath_only
    ```

    (with $delay < clk A period or smaller of the two clock periods)
  - XDC file with set_max_delay constraint auto-generated by the IP Catalog
  - Do not create async clock groups
    - set_clock_groups has higher precedence and would override set_max_delay



Notes:

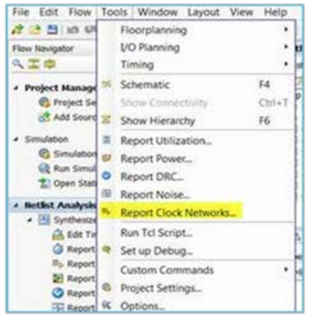
– datapath_only is Xilinx specific (not SDC compliant)

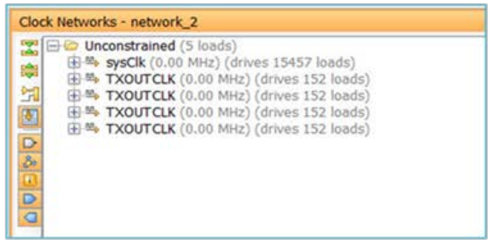
Baseline Stage 1: Constraint Development

Clock Interaction: Final Step

- **Run `report_clock_networks` to ensure that the data path between the clock domains are analysed properly**
  – You want the design to have clean clock lines without logic
    - Tip: Use clock gating option in synthesis to remove LUTs on the clock line

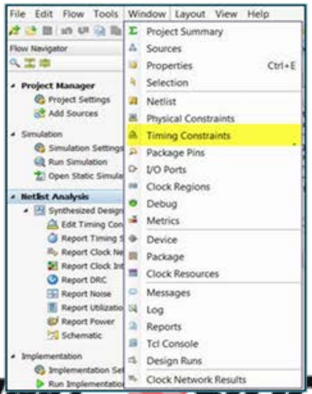
- **`report_clock_network` shows unconstrained networks**

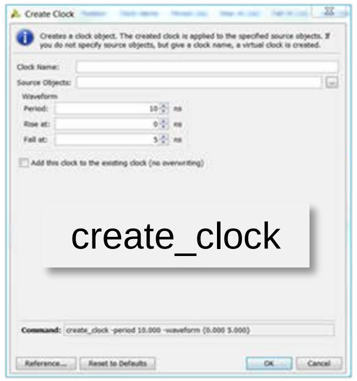




```
Clock Networks - network_2
⊟ Unconstrained (5 loads)
    ⊞ sysClk (0.00 MHz) (drives 15457 loads)
    ⊞ TXOUTCLK (0.00 MHz) (drives 152 loads)
    ⊞ TXOUTCLK (0.00 MHz) (drives 152 loads)
    ⊞ TXOUTCLK (0.00 MHz) (drives 152 loads)
    ⊞ TXOUTCLK (0.00 MHz) (drives 152 loads)
```

Report clock network has unconstrained clocks





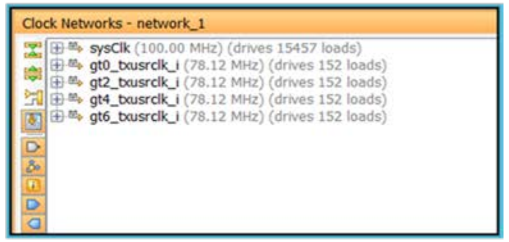
Create Clock

Creates a clock object. The created clock is applied to the specified source objects. If you do not specify source objects, but give a clock name, a virtual clock is created.

Clock Name:
Source Objects:
Waveform
Period: 10 ns
Rise at: 0 ns
Fall at: 5 ns
☐ Add this clock to the existing clock (no overwriting)

Command: `create_clock -period 10.000 -waveform {0.000 5.000}`

create_clock

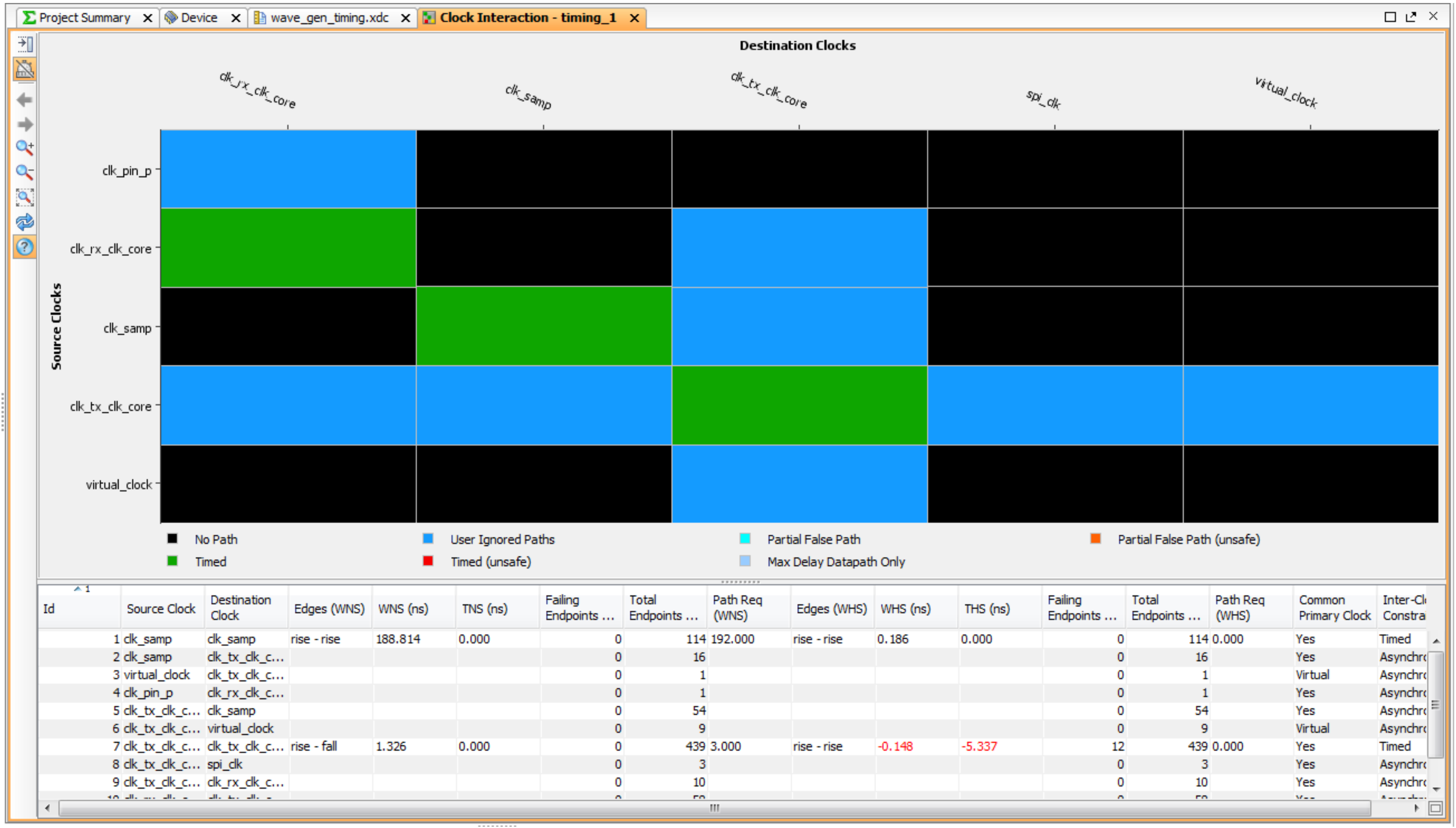


```
Clock Networks - network_1
⊞ sysClk (100.00 MHz) (drives 15457 loads)
⊞ gt0_txusrclk_i (78.12 MHz) (drives 152 loads)
⊞ gt2_txusrclk_i (78.12 MHz) (drives 152 loads)
⊞ gt4_txusrclk_i (78.12 MHz) (drives 152 loads)
⊞ gt6_txusrclk_i (78.12 MHz) (drives 152 loads)
```

Report clock network has **no** unconstrained clocks

Report Clock Interaction after setting set_clock_groups constraints in Baseline Stage 1

- **Run `report_clock_interaction`**

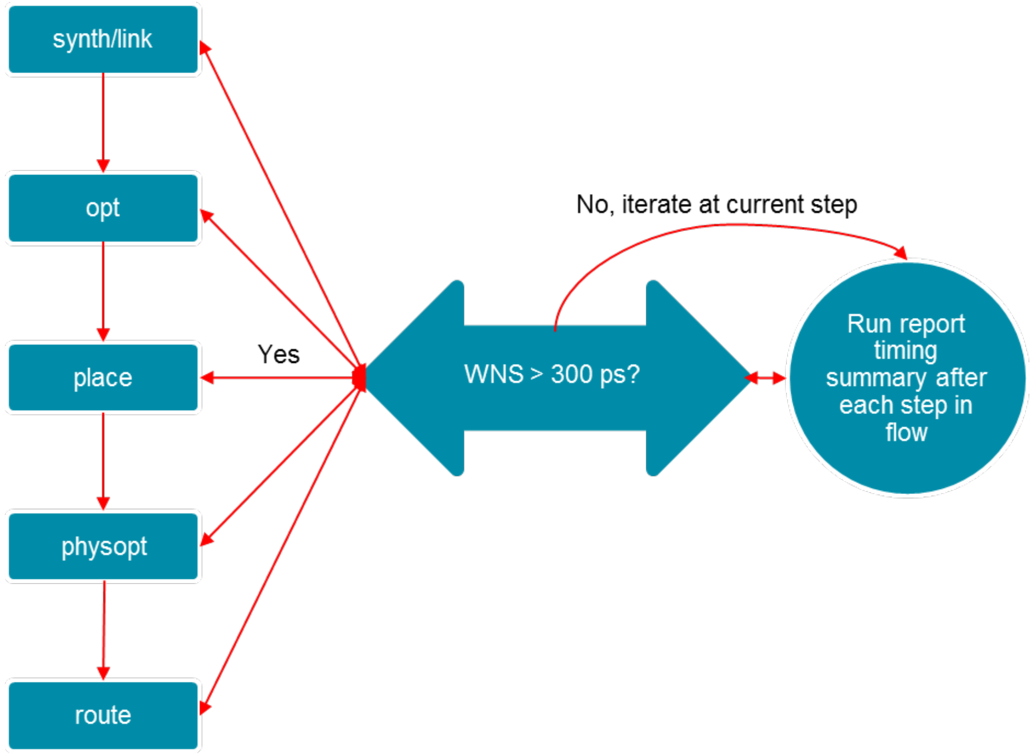


	clk_rx_clk_core	clk_samp	clk_tx_clk_core	spi_clk	virtual_clock
clk_pin_p	🟦	⬛	⬛	⬛	⬛
clk_rx_clk_core	🟩	⬛	🟦	⬛	⬛
clk_samp	⬛	🟩	🟦	⬛	⬛
clk_tx_clk_core	🟦	🟦	🟩	🟦	🟦
virtual_clock	⬛	⬛	🟦	⬛	⬛

Id	Source Clock	Destination Clock	Edges (WNS)	WNS (ns)	TNS (ns)	Failing Endpoints ...	Total Endpoints ...	Path Req (WNS)	Edges (WHS)	WHS (ns)	THS (ns)	Failing Endpoints ...	Total Endpoints ...	Path Req (WHS)	Common Primary Clock	Inter-Cl Constra
1	clk_samp	clk_samp	rise - rise	188.814	0.000	0	114	192.000	rise - rise	0.186	0.000	0	114	0.000	Yes	Timed
2	clk_samp	clk_tx_clk_c...				0	16					0	16		Yes	Asynchr
3	virtual_clock	clk_tx_clk_c...				0	1					0	1		Virtual	Asynchr
4	clk_pin_p	clk_rx_clk_c...				0	1					0	1		Yes	Asynchr
5	clk_tx_clk_c...	clk_samp				0	54					0	54		Yes	Asynchr
6	clk_tx_clk_c...	virtual_clock				0	9					0	9		Virtual	Asynchr
7	clk_tx_clk_c...	clk_tx_clk_c...	rise - fall	1.326	0.000	0	439	3.000	rise - rise	-0.148	-5.337	12	439	0.000	Yes	Timed
8	clk_tx_clk_c...	spi_clk				0	3					0	3		Yes	Asynchr
9	clk_tx_clk_c...	clk_rx_clk_c...				0	10					0	10		Yes	Asynchr

Baseline Stage 2: Implementation with report_timing_summary
WNS < 300ps as a rule of thumb…

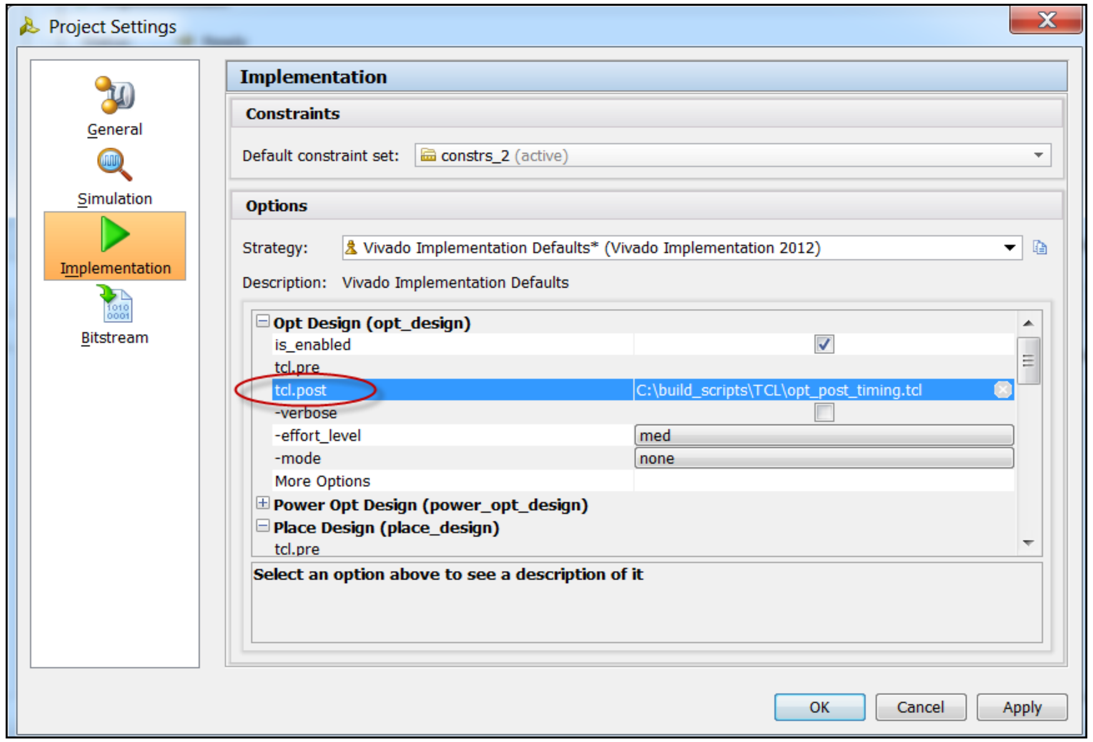
- **Run report_timing_summary after each step (not optional)**
- **Ensure <u>WNS < 300 ps</u>**
- **If TNS is better than -30ns, you can actually proceed to the next step even if the WNS is worse than -300ps**


synth/link
opt
place
physopt
route
Yes
WNS > 300 ps?
No, iterate at current step
Run report timing summary after each step in flow

Baseline Stage 2: Implementation with report_timing_summary

Setting up with report_timing_summary

GUI



Project Settings

Implementation

Constraints

Default constraint set: constrs_2 (active)

Options

Strategy: Vivado Implementation Defaults* (Vivado Implementation 2012)

Description: Vivado Implementation Defaults

Opt Design (opt_design)	
is_enabled	✓
tcl.pre	
tcl.post	C:\build_scripts\TCL\opt_post_timing.tcl
-verbose	
-effort_level	med
-mode	none
More Options	
Power Opt Design (power_opt_design)	
Place Design (place_design)	
tcl.pre	

Select an option above to see a description of it

opt_post_timing.tcl file:

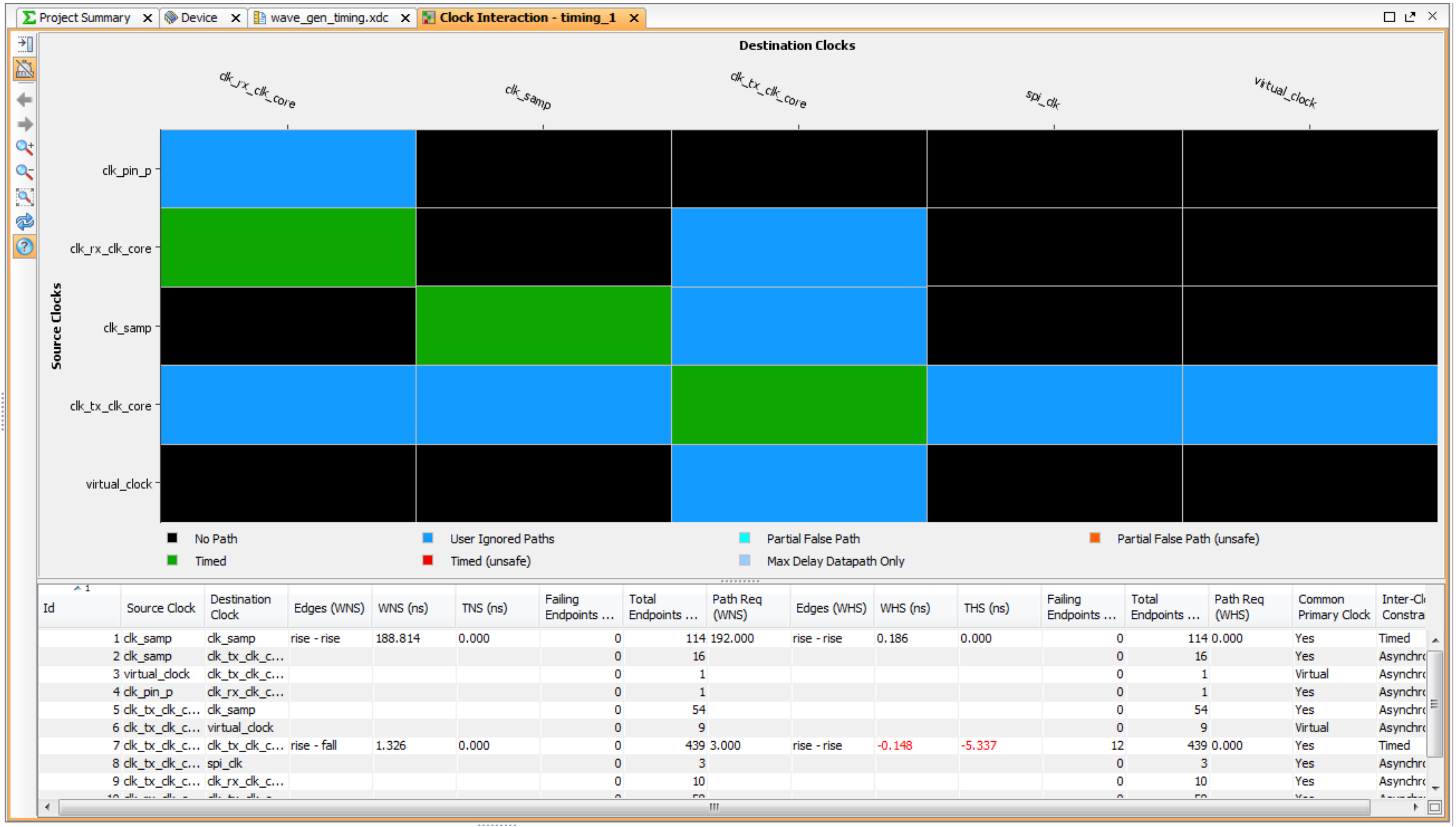
```
report_timing_summary -file opt_timing.rpt
```

Batch

Build.tcl file:

```
link_design -name top -part xc7vx1140tflg1928-2
read_xdc top.xdc

opt_design
report_timing_summary -file opt_timing.rpt
write_checkpoint -force opt.dcp

place_design
report_timing_summary -file place_timing.rpt
write_checkpoint -force place.dcp

phys_opt_design
report_timing_summary -file popt_timing.rpt
write_checkpoint -force popt.dcp

route_design
report_timing_summary –file routed_timing.rpt
write_checkpoint –force routed.dcp
```

Report Clock Interaction after Baseline Stage 2



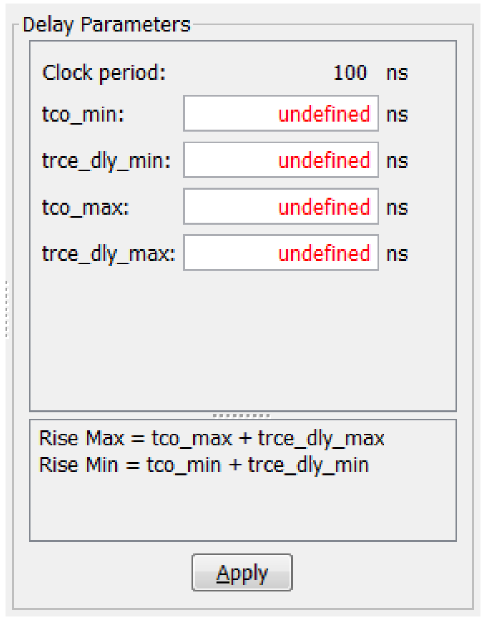
Id	Source Clock	Destination Clock	Edges (WNS)	WNS (ns)	TNS (ns)	Failing Endpoints ...	Total Endpoints ...	Path Req (WNS)	Edges (WHS)	WHS (ns)	THS (ns)	Failing Endpoints ...	Total Endpoints ...	Path Req (WHS)	Common Primary Clock	Inter-Cl Constra
1	clk_samp	clk_samp	rise - rise	188.814	0.000	0	114	192.000	rise - rise	0.186	0.000	0	114	0.000	Yes	Timed
2	clk_samp	clk_tx_clk_c...				0	16					0	16		Yes	Asynchr
3	virtual_clock	clk_tx_clk_c...				0	1					0	1		Virtual	Asynchr
4	clk_pin_p	clk_rx_clk_c...				0	1					0	1		Yes	Asynchr
5	clk_tx_clk_c...	clk_samp				0	54					0	54		Yes	Asynchr
6	clk_tx_clk_c...	virtual_clock				0	9					0	9		Virtual	Asynchr
7	clk_tx_clk_c...	clk_tx_clk_c...	rise - fall	1.326	0.000	0	439	3.000	rise - rise	-0.148	-5.337	12	439	0.000	Yes	Timed
8	clk_tx_clk_c...	spi_clk				0	3					0	3		Yes	Asynchr
9	clk_tx_clk_c...	clk_rx_clk_c...				0	10					0	10		Yes	Asynchr

Setting Input / Output Delays

- **Specify Realistic I/O delays:** `set_input_delay, set_output_delay`
  - – Wrong delay value (e.g., <0 ns) can cause *invalid analysis*
- **Input/Output Delay constraint helpers**
  - – Use the XDC template for constraining input and output interfaces
- **Check Progress:** `check_timing, report_timing_summary, report_timing`

Constraining Inputs

- **Referenced to clock input**
  - Max for setup analysis
  - Min for hold analysis

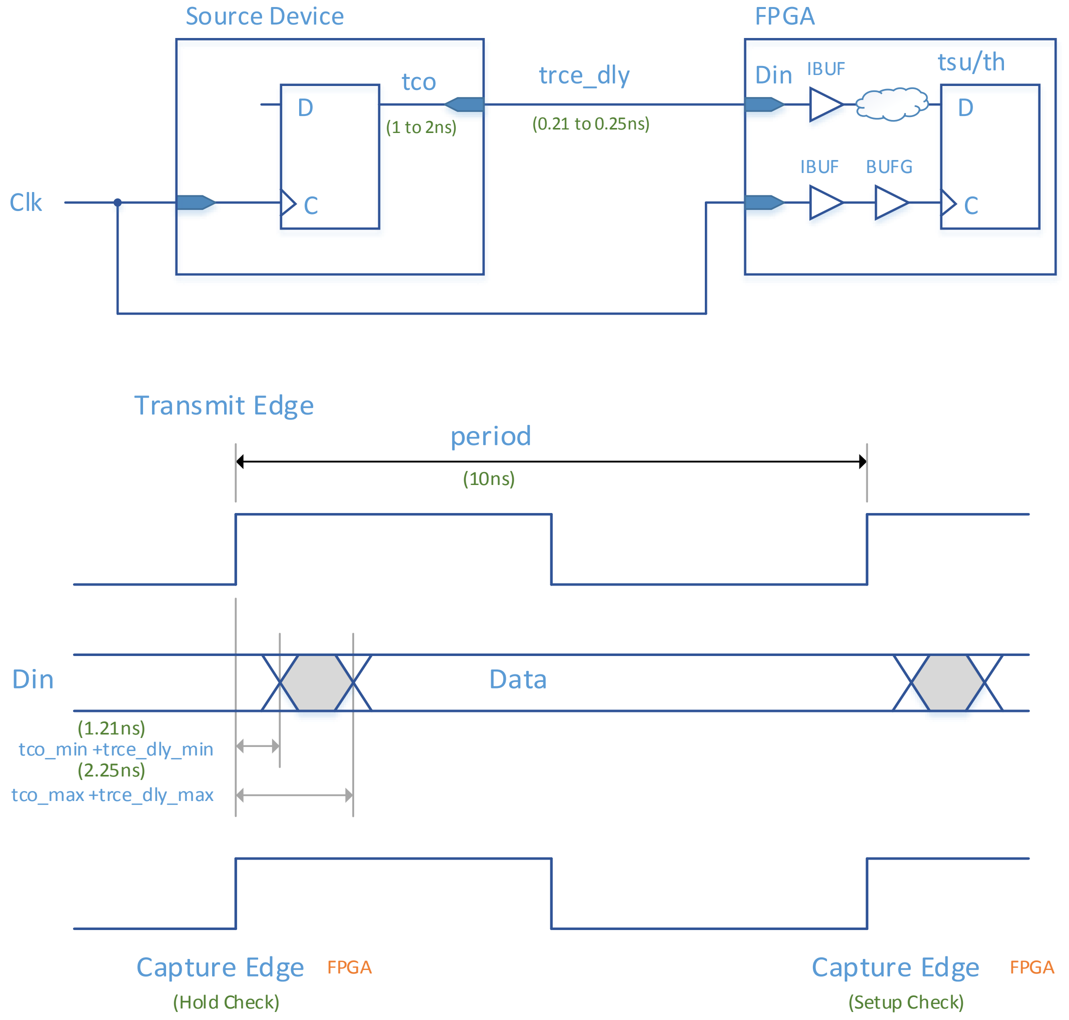


Delay Parameters

Clock period:	100	ns
tco_min:	undefined	ns
trce_dly_min:	undefined	ns
tco_max:	undefined	ns
trce_dly_max:	undefined	ns

Rise Max = tco_max + trce_dly_max
Rise Min = tco_min + trce_dly_min

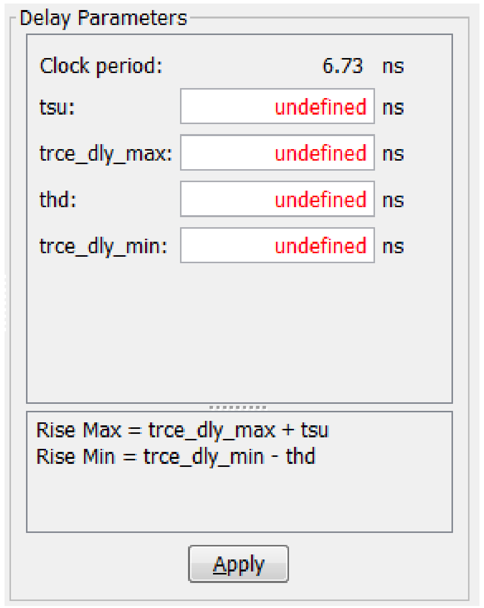
Timing Constraint Wizard!



set_input_delay -clock [get_clocks {Clk}] -min -add_delay 1.21 [get_ports {Din[*]}]
set_input_delay -clock [get_clocks {Clk}] -max -add_delay 2.25 [get_ports {Din[*]}]

Constraining Outputs

- **Referenced to clock input**
  - Max for setup analysis
  - Min for hold analysis

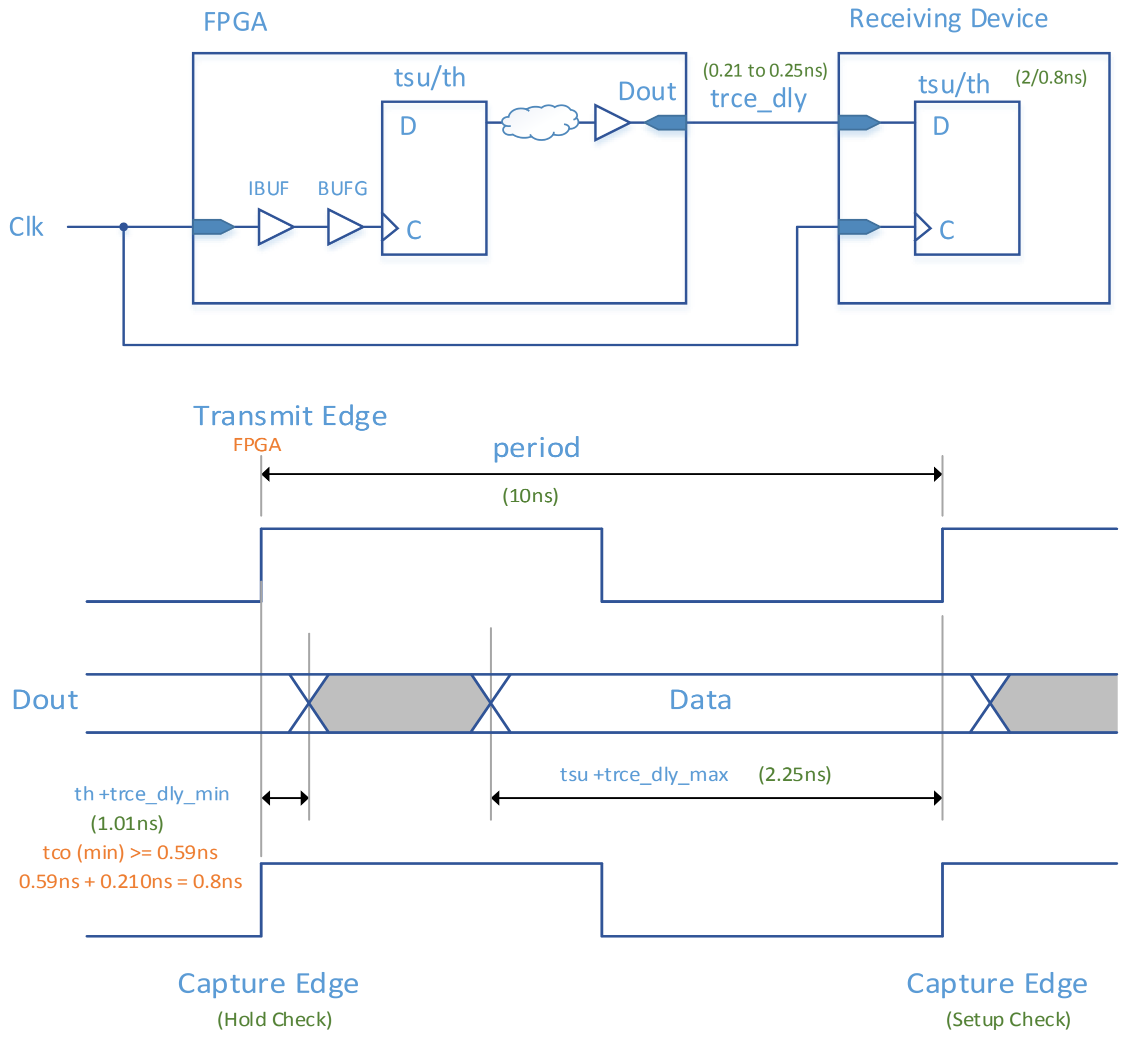


Delay Parameters

Clock period:	6.73	ns
tsu:	undefined	ns
trce_dly_max:	undefined	ns
thd:	undefined	ns
trce_dly_min:	undefined	ns

Rise Max = trce_dly_max + tsu
Rise Min = trce_dly_min - thd

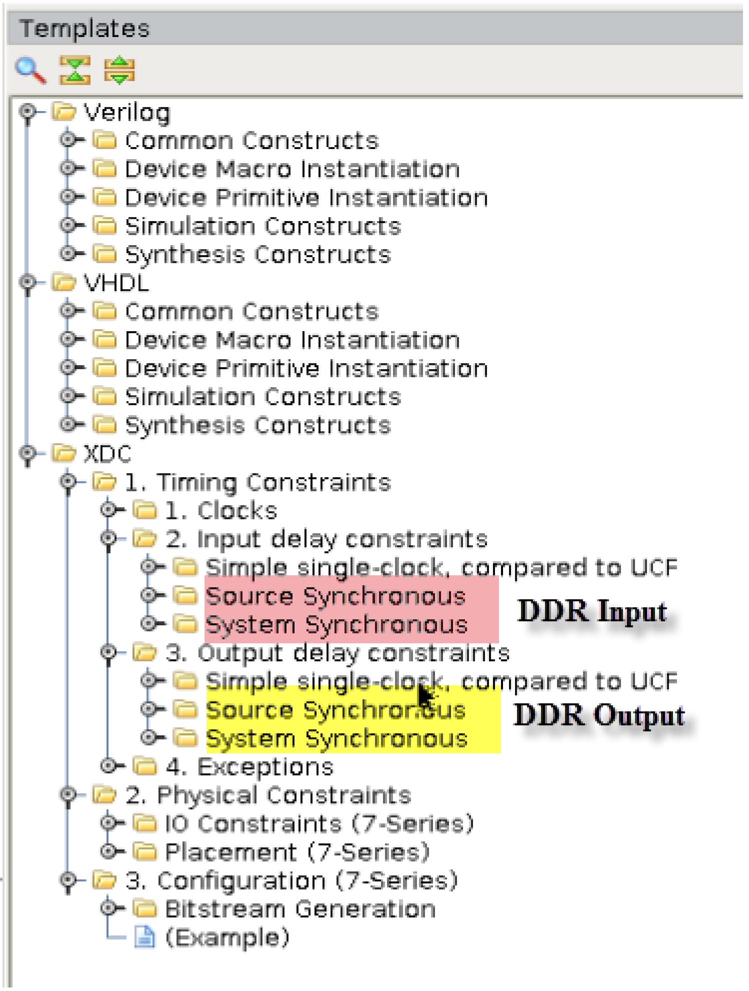
Timing Constraint Wizard!



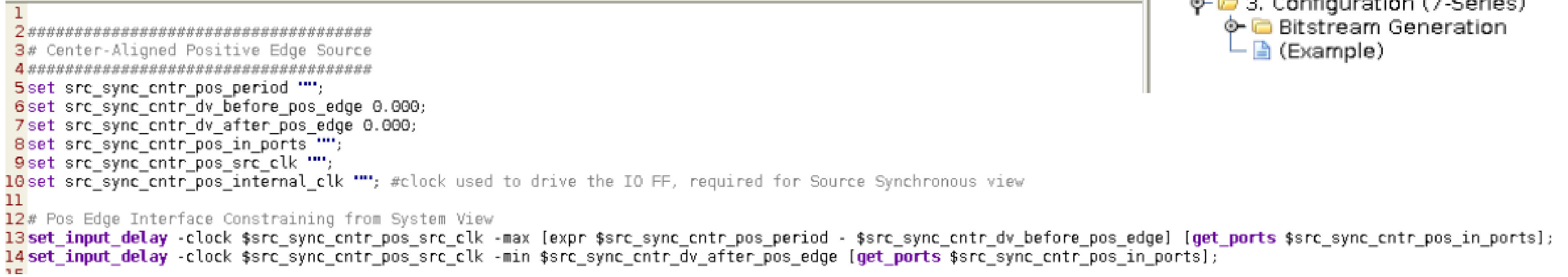
set_output_delay -clock [get_clocks {Clk}] -min -add_delay -0.59 [get_ports {Dout[*]}]
set_output_delay -clock [get_clocks {Clk}] -max -add_delay 2.25 [get_ports {Dout[*]}]

Using Vivado Language Templates
XDC Template

- **Setting Setting Input / Output Delays can be tricky**
- **Accessing templates in IDE**
  - Windows → Language Templates
- **SDR & DDR Templates**
  - Inputs and outputs
  - Source / System synchronous
  - Center / Edge aligned



```
Templates
Verilog
    Common Constructs
    Device Macro Instantiation
    Device Primitive Instantiation
    Simulation Constructs
    Synthesis Constructs
VHDL
    Common Constructs
    Device Macro Instantiation
    Device Primitive Instantiation
    Simulation Constructs
    Synthesis Constructs
XDC
    1. Timing Constraints
        1. Clocks
        2. Input delay constraints
            Simple single-clock, compared to UCF
            Source Synchronous                      DDR Input
            System Synchronous
        3. Output delay constraints
            Simple single-clock, compared to UCF
            Source Synchronous                      DDR Output
            System Synchronous
        4. Exceptions
    2. Physical Constraints
        IO Constraints (7-Series)
        Placement (7-Series)
    3. Configuration (7-Series)
        Bitstream Generation
        (Example)
```



```

#########################################
# Center-Aligned Positive Edge Source
#########################################
set src_sync_cntr_pos_period "";
set src_sync_cntr_dv_before_pos_edge 0.000;
set src_sync_cntr_dv_after_pos_edge 0.000;
set src_sync_cntr_pos_in_ports "";
set src_sync_cntr_pos_src_clk "";
set src_sync_cntr_pos_internal_clk ""; #clock used to drive the IO FF, required for Source Synchronous view

# Pos Edge Interface Constraining from System View
set_input_delay -clock $src_sync_cntr_pos_src_clk -max [expr $src_sync_cntr_pos_period - $src_sync_cntr_dv_before_pos_edge] [get_ports $src_sync_cntr_pos_in_ports];
set_input_delay -clock $src_sync_cntr_pos_src_clk -min $src_sync_cntr_dv_after_pos_edge [get_ports $src_sync_cntr_pos_in_ports];
```

Synthesize and Implement after setting Input / Output Delays

- **Re-synthesize and implement the design to enable the I/O constraints to alter the synthesis and implementation results found from Baseline Stage 2**
  - Note that if your additional timing constraints meet timing after reloading the netlist, you may not need to re-synthesize and re-implement
- **Analyze the design's performance against the performance baseline, you may find some intra-clock paths now fail**

Timing Exceptions: Less is More!

- **Goal – to help timing closure**
  - Adjust unrealistic timing requirements
  - Avoid higher implementation runtimes
  - Be aware of Exception Priority

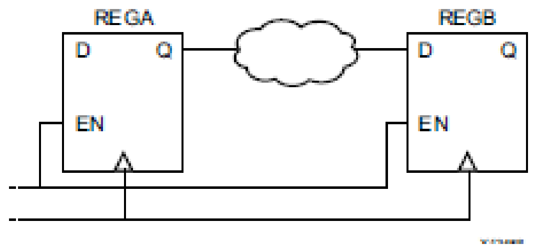
```
set_false_path
set_multicycle_path
set_max_delay
```

- **Exceptions can HURT timing closure**
- **Exception Validation: `report_exceptions, report_drc -ruledeck timing_checks/methodology_checks`**

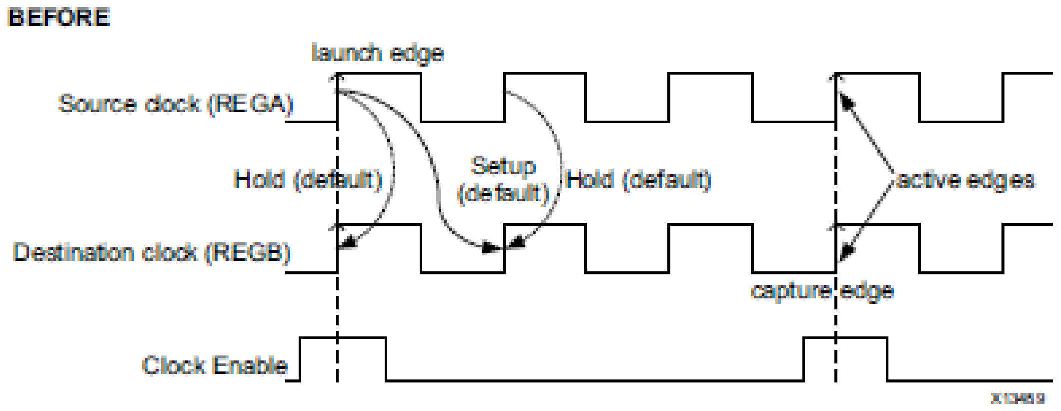
Syntax related	`set_multicycle_path`	– beware about *–hold* (avoid *wrong* hold violations)
	`regexp`	– check you only cover the expected paths
Runtime	`set_false_path -from`	– No runtime impact
	`set_false_path -from … -to …`	– Impact. (due to shared paths)
Conflicts resolution	`set_multicycle_path 3 -from REGA/Q`	*-from wins: it has higher priority vs. -to*
	`set_multicycle_path 2 -to REGB/D`	

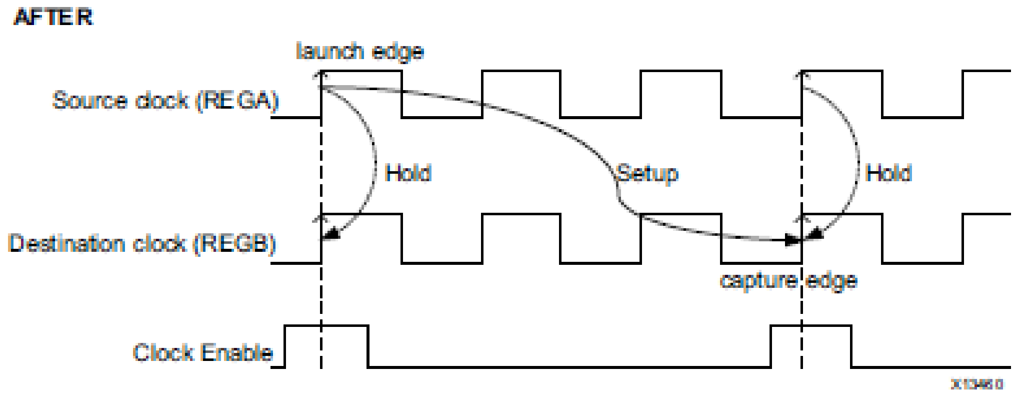
Multicycle Paths

- `set_multicycle_path` **N implies a HOLD check at N-1**
  - E.g.: a multicycle_path of 10 implies a HOLD requirement of 9 cycles!
- **Whenever setup check is changed, hold check is also changed**
- **Guidelines to avoid hurting runtime and SETUP**
  - Add proper circuitry (e.g. clock enable logic)
  - Bring the HOLD requirement back to 0 (reduce by N-1) to avoid incorrect HOLD violations
  - Example: Same clock for both startpoint and endpoint, with a clock enable active every 3 cycles



Enabled Flops with Same Clock Signal





```
BEFORE:    set_multicycle_path -from [get_cells regB] -to [get_cells regC]  3          → setup:3, hold:2
                                                       →
AFTER:     set_multicycle_path -from [get_cells regB] -to [get_cells regC]  3 -setup
           set_multicycle_path -from [get_cells regB] -to [get_cells regC]  2 -hold    → setup:3, hold:1
```

XDC Timing File with Timing Exceptions

- **First, remove the `set_clock_groups -asynchronous` constraint**
  - This is not necessary any more since you want to now properly constraint your design's inter-clock paths
- **Apply timing exceptions to the design (for example)**

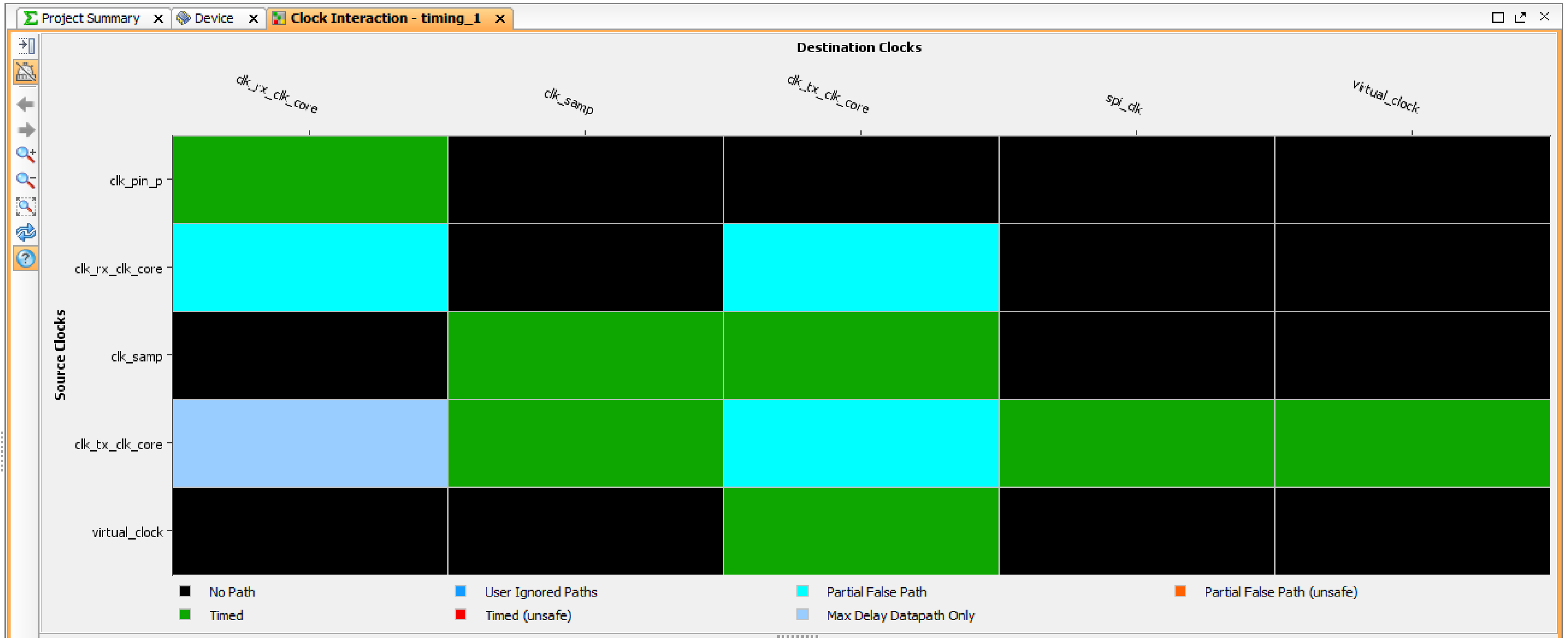
```
set_multicycle_path -from [get_cells \
{cmd_parse_i0/send_resp_data_reg[*]}] -to \
[get_cells {resp_gen_i0/to_bcd_i0/bcd_out_reg[*]}] 2
set_multicycle_path -hold -from [get_cells \
{cmd_parse_i0/send_resp_data_reg[*]}] -to \
[get_cells {resp_gen_i0/to_bcd_i0/bcd_out_reg[*]}] 1
set_false_path -from [get_ports rst_pin]
set_max_delay 5 -from $rx_clk -to $tx_clk
set_max_delay -from [get_cells \
clkx_nsamp_i0/meta_harden_bus_new_i0/signal_meta_reg] -to \ [get_cells
clkx_nsamp_i0/meta_harden_bus_new_i0/signal_dst_reg]\ 2
set_max_delay -from [get_cells \
clkx_pre_i0/meta_harden_bus_new_i0/signal_meta_reg] -to \ [get_cells
clkx_pre_i0/meta_harden_bus_new_i0/signal_dst_reg] 2
```

Synthesize and Implement after setting Timing Exceptions

- **Re-synthesize and implement the design to enable the path-specific constraints to alter the synthesis and implementation results found after setting input/output constraints**
  - Note that if your additional timing constraints meet timing after reloading the netlist, you may not need to re-synthesize and re-implement
    - This is **especially** true if you are only adding multi-cycle and false path constraints

Report Clock Interaction after adding Timing Exceptions

- **This is the final Clock Interaction report generated after it has been completely and properly constrained**
  - From this you can see that some of the paths between clocks do not have any paths-specific constraints (and logically no synchronization circuits)
    - This utility does not anticipate your design's needs; it only tries to help you evaluate your design



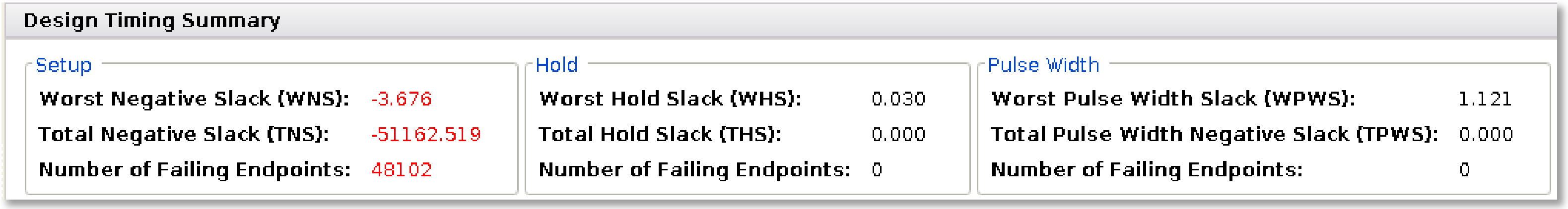
Destination Clocks / Source Clocks

Source Clocks	clk_rx_clk_core	clk_samp	clk_tx_clk_core	spi_clk	virtual_clock
clk_pin_p	🟩	⬛	⬛	⬛	⬛
clk_rx_clk_core	🟦	⬛	🟦	⬛	⬛
clk_samp	⬛	🟩	🟩	⬛	⬛
clk_tx_clk_core	▨	🟩	🟦	🟩	🟩
virtual_clock	⬛	⬛	🟩	⬛	⬛

⬛ No Path	■ User Ignored Paths	🟦 Partial False Path	🟧 Partial False Path (unsafe)
🟩 Timed	🟥 Timed (unsafe)	▨ Max Delay Datapath Only	

Timing Analysis, Reading Reports

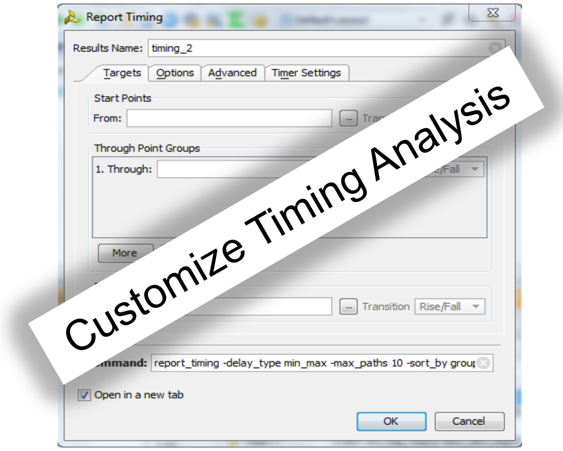
- **`report_timing_summary` – a complete view on the Design Timing**
  - Store results from various commands: `check_timing`, `report_timing`, …



Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	-3.676	Worst Hold Slack (WHS):	0.030	Worst Pulse Width Slack (WPWS):	1.121
Total Negative Slack (TNS):	-51162.519	Total Hold Slack (THS):	0.000	Total Pulse Width Negative Slack (TPWS):	0.000
Number of Failing Endpoints:	48102	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0

- **`report_timing` – interactive STA**
  - Enables to focus on a specific design part
    - One clock domain
    - All paths between two registers
    - All paths going though a specific net



Report Timing

Results Name: timing_2

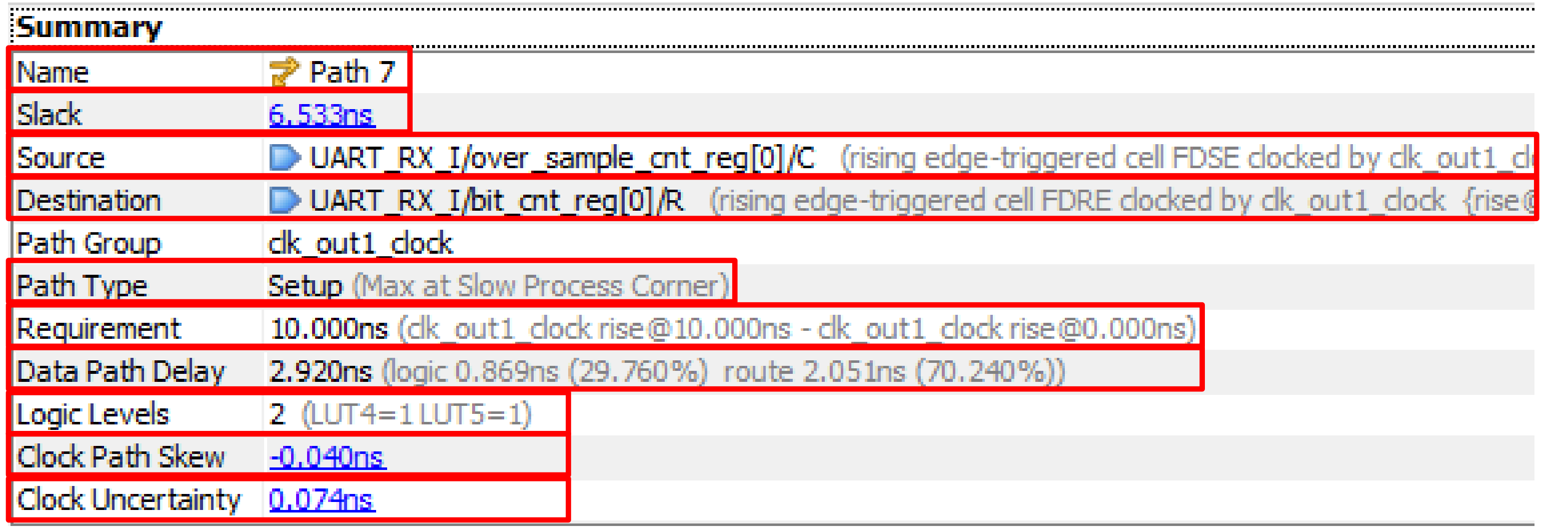
Command: report_timing -delay_type min_max -max_paths 10 -sort_by grou

☑ Open in a new tab

Use them for constraints tuning at each constraints definition step

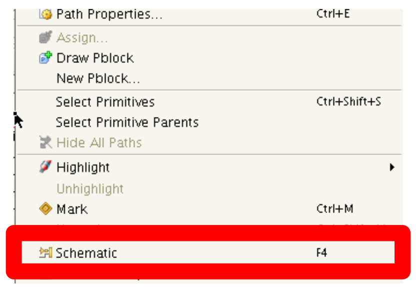
Understanding Timing Reports - Summary

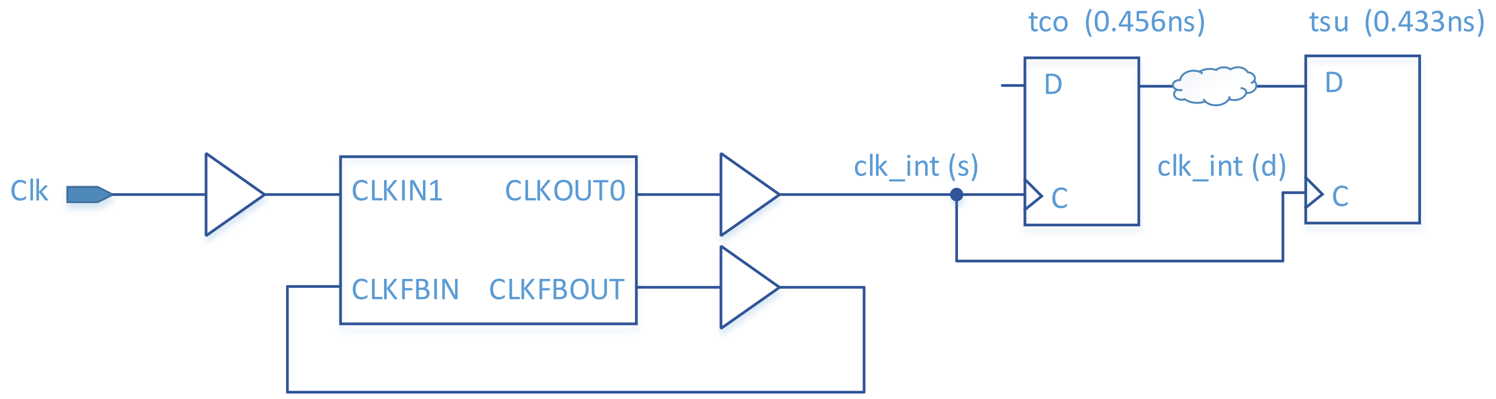
- Path Name
- Slack
- Source
- Destination
- Path Type
- Requirement
- Data Path Delay
- Logic Levels
- Clock Path Skew
- Clock Uncertainty



Summary

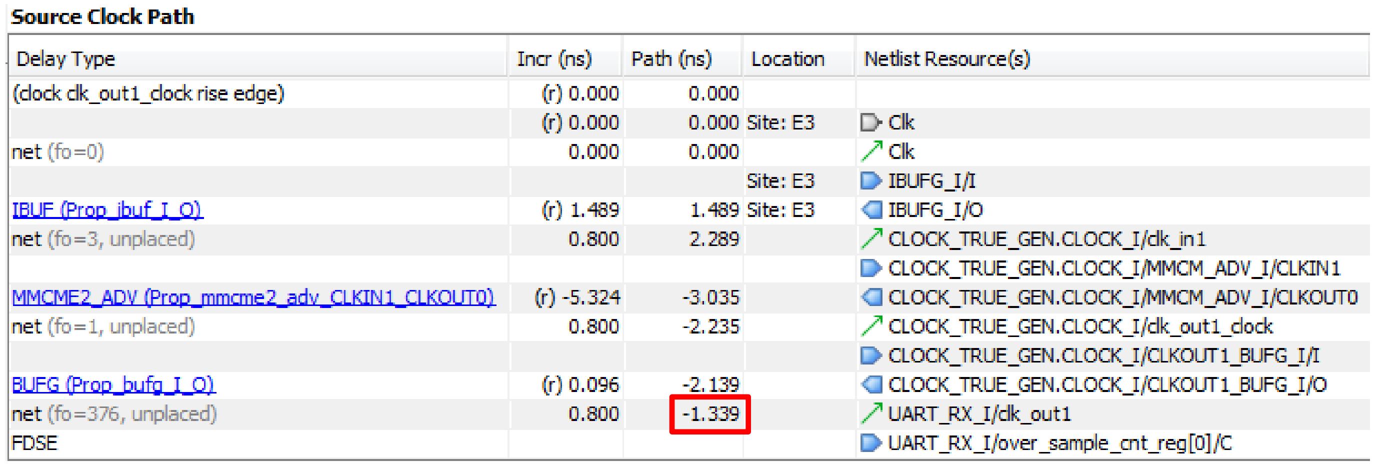
Name	Path 7
Slack	6.533ns
Source	UART_RX_I/over_sample_cnt_reg[0]/C (rising edge-triggered cell FDSE clocked by clk_out1_cl
Destination	UART_RX_I/bit_cnt_reg[0]/R (rising edge-triggered cell FDRE clocked by clk_out1_clock {rise@
Path Group	clk_out1_clock
Path Type	Setup (Max at Slow Process Corner)
Requirement	10.000ns (clk_out1_clock rise@10.000ns - clk_out1_clock rise@0.000ns)
Data Path Delay	2.920ns (logic 0.869ns (29.760%) route 2.051ns (70.240%))
Logic Levels	2 (LUT4=1 LUT5=1)
Clock Path Skew	-0.040ns
Clock Uncertainty	0.074ns





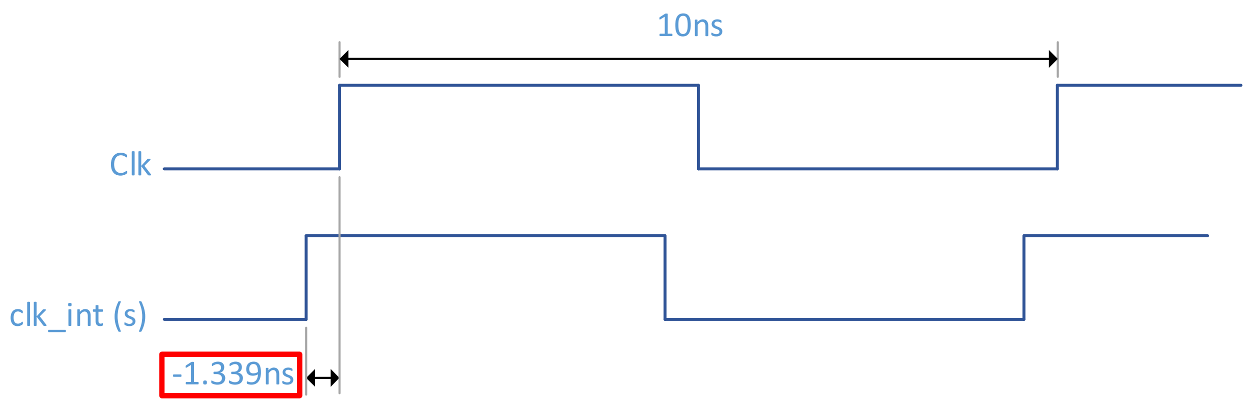
Understanding Timing Reports – Source Clock

- **Delay from the Clk input to source Clock input**



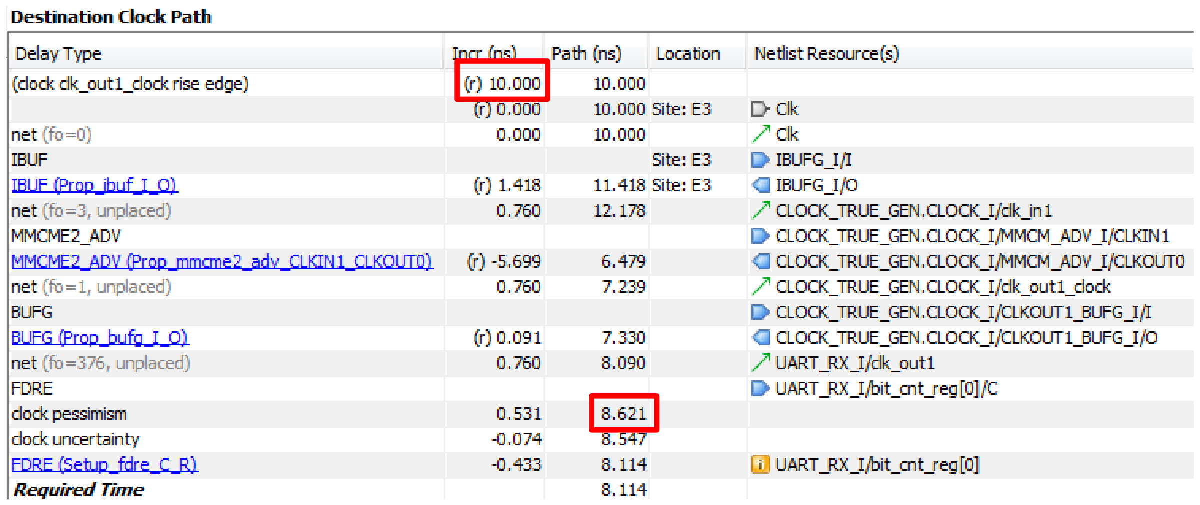
Source Clock Path

Delay Type	Incr (ns)	Path (ns)	Location	Netlist Resource(s)
(clock clk_out1_clock rise edge)	(r) 0.000	0.000		
	(r) 0.000	0.000	Site: E3	Clk
net (fo=0)	0.000	0.000		Clk
			Site: E3	IBUFG_I/I
IBUF (Prop_ibuf_I_O)	(r) 1.489	1.489	Site: E3	IBUFG_I/O
net (fo=3, unplaced)	0.800	2.289		CLOCK_TRUE_GEN.CLOCK_I/clk_in1
				CLOCK_TRUE_GEN.CLOCK_I/MMCM_ADV_I/CLKIN1
MMCME2_ADV (Prop_mmcme2_adv_CLKIN1_CLKOUT0)	(r) -5.324	-3.035		CLOCK_TRUE_GEN.CLOCK_I/MMCM_ADV_I/CLKOUT0
net (fo=1, unplaced)	0.800	-2.235		CLOCK_TRUE_GEN.CLOCK_I/clk_out1_clock
				CLOCK_TRUE_GEN.CLOCK_I/CLKOUT1_BUFG_I/I
BUFG (Prop_bufg_I_O)	(r) 0.096	-2.139		CLOCK_TRUE_GEN.CLOCK_I/CLKOUT1_BUFG_I/O
net (fo=376, unplaced)	0.800	-1.339		UART_RX_I/clk_out1
FDSE				UART_RX_I/over_sample_cnt_reg[0]/C



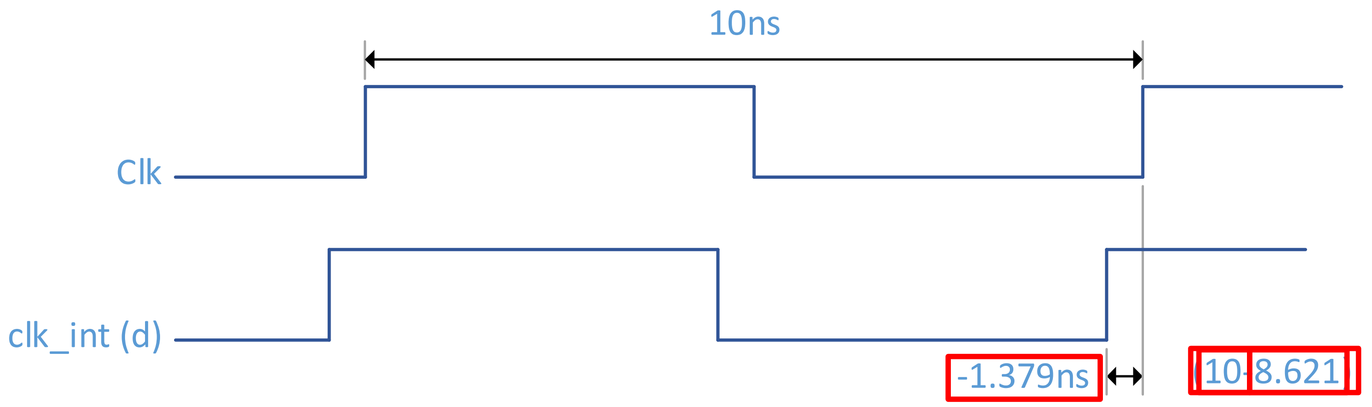
Understanding Timing Reports – Destination Clock

Delay from the Clk input to destination Clock input



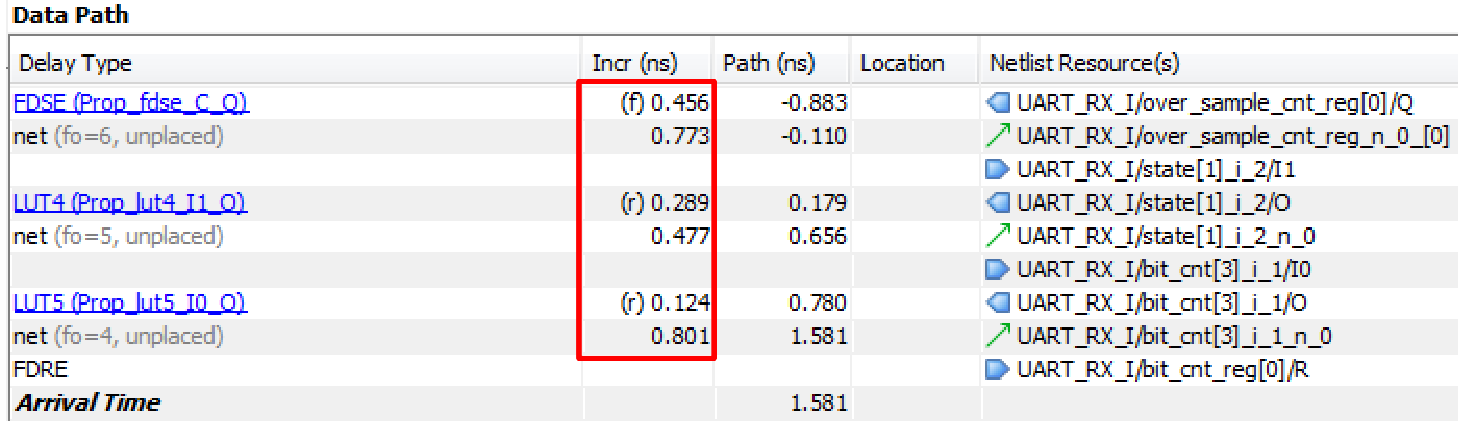
Destination Clock Path

Delay Type	Incr (ns)	Path (ns)	Location	Netlist Resource(s)
(clock clk_out1_clock rise edge)	(r) 10.000	10.000		
	(r) 0.000	10.000	Site: E3	Clk
net (fo=0)	0.000	10.000		Clk
IBUF			Site: E3	IBUFG_I/I
IBUF (Prop_ibuf_I_O)	(r) 1.418	11.418	Site: E3	IBUFG_I/O
net (fo=3, unplaced)	0.760	12.178		CLOCK_TRUE_GEN.CLOCK_I/clk_in1
MMCME2_ADV				CLOCK_TRUE_GEN.CLOCK_I/MMCM_ADV_I/CLKIN1
MMCME2_ADV (Prop_mmcme2_adv_CLKIN1_CLKOUT0)	(r) -5.699	6.479		CLOCK_TRUE_GEN.CLOCK_I/MMCM_ADV_I/CLKOUT0
net (fo=1, unplaced)	0.760	7.239		CLOCK_TRUE_GEN.CLOCK_I/clk_out1_clock
BUFG				CLOCK_TRUE_GEN.CLOCK_I/CLKOUT1_BUFG_I/I
BUFG (Prop_bufg_I_O)	(r) 0.091	7.330		CLOCK_TRUE_GEN.CLOCK_I/CLKOUT1_BUFG_I/O
net (fo=376, unplaced)	0.760	8.090		UART_RX_I/clk_out1
FDRE				UART_RX_I/bit_cnt_reg[0]/C
clock pessimism	0.531	8.621		
clock uncertainty	-0.074	8.547		
FDRE (Setup_fdre_C_R)	-0.433	8.114		UART_RX_I/bit_cnt_reg[0]
Required Time		8.114		



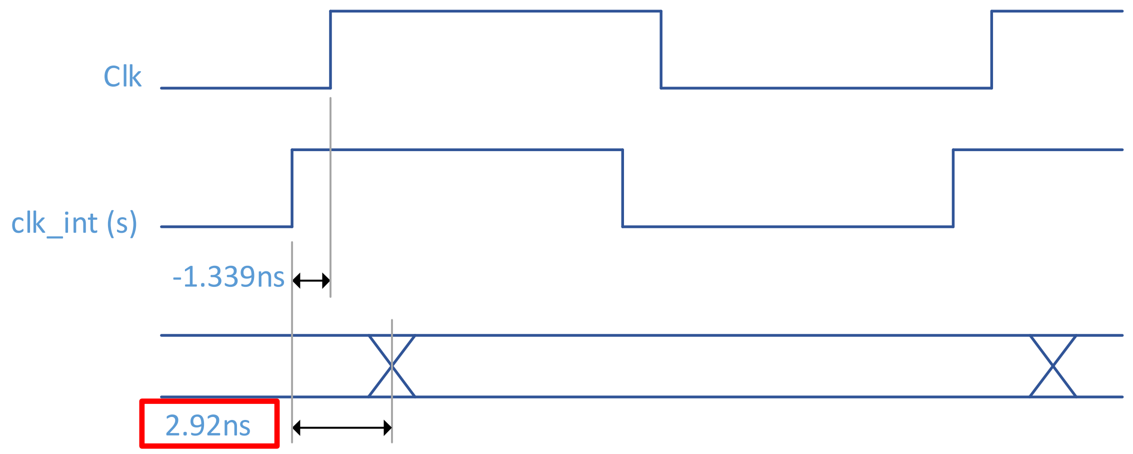
Understanding Timing Reports – Data Path

- **Delay from source FF to destination FF input**



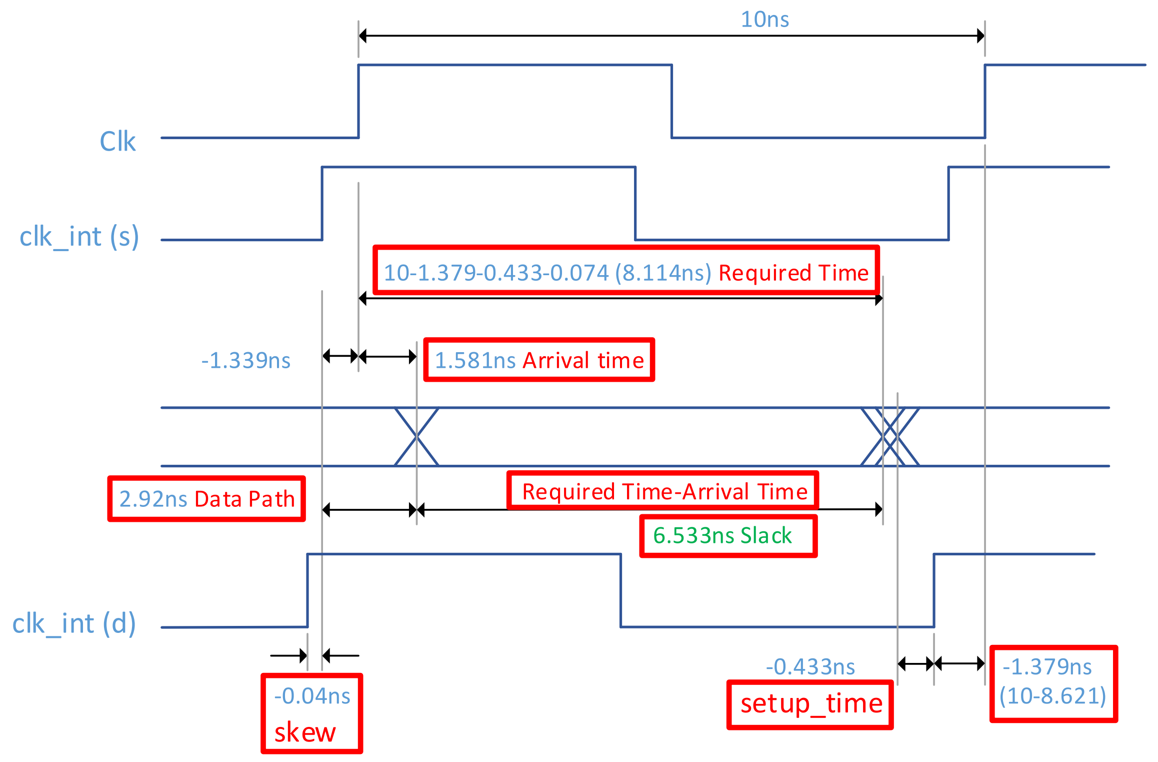
Data Path

Delay Type	Incr (ns)	Path (ns)	Location	Netlist Resource(s)
FDSE (Prop_fdse_C_Q)	(f) 0.456	-0.883		UART_RX_I/over_sample_cnt_reg[0]/Q
net (fo=6, unplaced)	0.773	-0.110		UART_RX_I/over_sample_cnt_reg_n_0_[0]
				UART_RX_I/state[1]_i_2/I1
LUT4 (Prop_lut4_I1_O)	(r) 0.289	0.179		UART_RX_I/state[1]_i_2/O
net (fo=5, unplaced)	0.477	0.656		UART_RX_I/state[1]_i_2_n_0
				UART_RX_I/bit_cnt[3]_i_1/I0
LUT5 (Prop_lut5_I0_O)	(r) 0.124	0.780		UART_RX_I/bit_cnt[3]_i_1/O
net (fo=4, unplaced)	0.801	1.581		UART_RX_I/bit_cnt[3]_i_1_n_0
FDRE				UART_RX_I/bit_cnt_reg[0]/R
Arrival Time		1.581		



Understanding Timing Reports - Slack

- Clock path skew is the difference between source and destination clocks
- Clock uncertainty reduces slack
- Arrival time is data path delay with respect to the Clk input
- Required time is the requirement - clock delay, setup time and clock uncertainty
- Slack is required time - arrival time

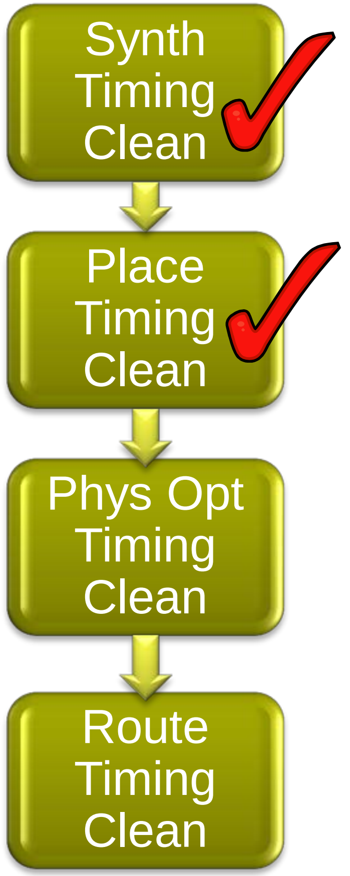


Agenda

- XILINX Overview
  - A Generation Ahead from 28nm to 16nm
  - Tools and Methodology
- UFDM Guidelines for easier Timing Closure
- Setting Clean Timing Constraints for Predictable Static Timing Analysis
  - How to set "Clean" constraints?
  - Baselining a Design
  - Analyzing through the Design : `report_timing_summary, report_clock_interaction`, `report_cdc…`
- Last Miles Strategy: Tips and Tricks
- What's next?

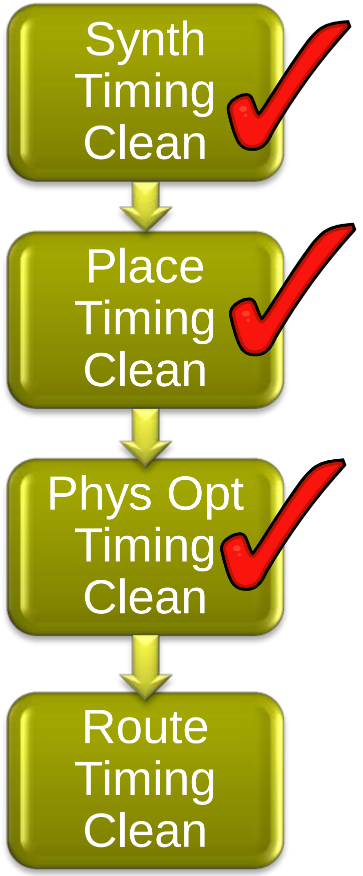
Timing Results Post Place Design

- **Assuming clean timing before place design.**
- **Typical causes of large timing violations:**
  - High fanout nets
  - Bad floorplan and/or bad IO placement
  - Over utilization
  - SLR crossings on SSI devices
- **Can go to phys-opt even if timing is not clean**
  - Reasons for bad WNS can be fixed by phys_opt



Timing Results Post Phys Opt Design

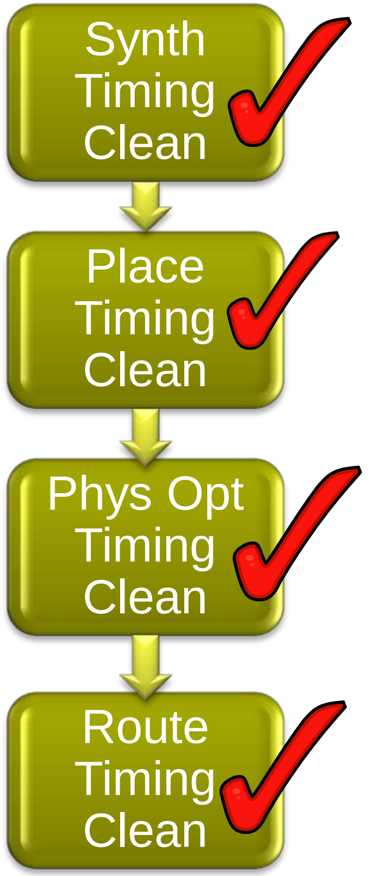
- **Assuming timing clean before phys-opt**
- **Typical causes of large timing violations:**
  - Phys-opt only works on top offenders
    - Try looping with various options
  - High fanout nets driven from LUTs
  - DONT_TOUCH attribute preventing optimizations
  - Replace
  - Retime push FFs in/out of BRAMs/SRLs
- **Once timing is clean WNS better than -300ps**
  - Go to route_design



XILINX ALL PROGRAMMABLE.

Timing Results Post Route Design

- **Assuming timing clean timing before route**
- **Typical causes of large timing violations:**
  - Hold fixing -> run route_design with:
    - set_false_path –hold –from [all_registers]
    - Report timing actual vs estimated
  - Congestion
- **Tips**
  - Overconstrain
  - Incremental placement
  - OOC for sub blocks



Overconstraining

- **Overconstraining works well in some cases**
  - When placer under-estimates routing delays
- **Correlation between routing estimates and actual routing are getting tighter in newer Vivado releases**
  - Post route: report timing with estimates and compare to actual
- **What is a good candidate for overconstraining?**
  - Positive slack in placer, but fails by ~200-300ps in router
  - Small negative slack in placer and router, i.e. ~200-300ps
- **What to overconstrain?**
  - Placer and phys-opt
  - Placer, phys_opt and router

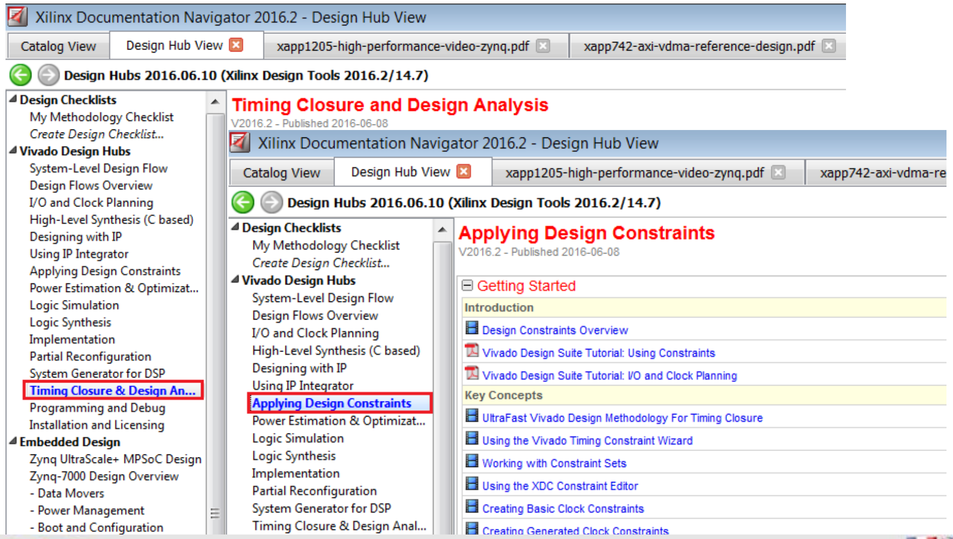
Agenda

- XILINX Overview
  - A Generation Ahead from 28nm to 16nm
  - Tools and Methodology
- UFDM Guidelines for easier Timing Closure
- Setting Clean Timing Constraints for Predictable Static Timing Analysis
  - How to set “Clean” constraints?
  - Baselining a Design
  - Analyzing through the Design : `report_timing_summary`, `report_clock_interaction`, `report_cdc`…
- Last Miles Strategy: Tips and Tricks
- What’s next?

What's Next?

- Documentation Navigator: Design Hub View
  - Applying Design Constraints
  - Timing Closure & Design Analysis

- MVD Training (Ludovic Aubel)
  - ludovic.aubel@mvd-fpga.com
  - Mobile: +33 (0)6 06 45 13 64


Xilinx Documentation Navigator 2016.2 - Design Hub View

Catalog View | Design Hub View | xapp1205-high-performance-video-zynq.pdf | xapp742-axi-vdma-reference-design.pdf

Design Hubs 2016.06.10 (Xilinx Design Tools 2016.2/14.7)

- Design Checklists
  - My Methodology Checklist
  - *Create Design Checklist...*
- Vivado Design Hubs
  - System-Level Design Flow
  - Design Flows Overview
  - I/O and Clock Planning
  - High-Level Synthesis (C based)
  - Designing with IP
  - Using IP Integrator
  - Applying Design Constraints
  - Power Estimation & Optimizat...
  - Logic Simulation
  - Logic Synthesis
  - Implementation
  - Partial Reconfiguration
  - System Generator for DSP
  - Timing Closure & Design An...
  - Programming and Debug
  - Installation and Licensing
- Embedded Design
  - Zynq UltraScale+ MPSoC Design
  - Zynq-7000 Design Overview
  - - Data Movers
  - - Power Management
  - - Boot and Configuration

Timing Closure and Design Analysis
V2016.2 - Published 2016-06-08

Xilinx Documentation Navigator 2016.2 - Design Hub View

Catalog View | Design Hub View | xapp1205-high-performance-video-zynq.pdf | xapp742-axi-vdma-re

Design Hubs 2016.06.10 (Xilinx Design Tools 2016.2/14.7)

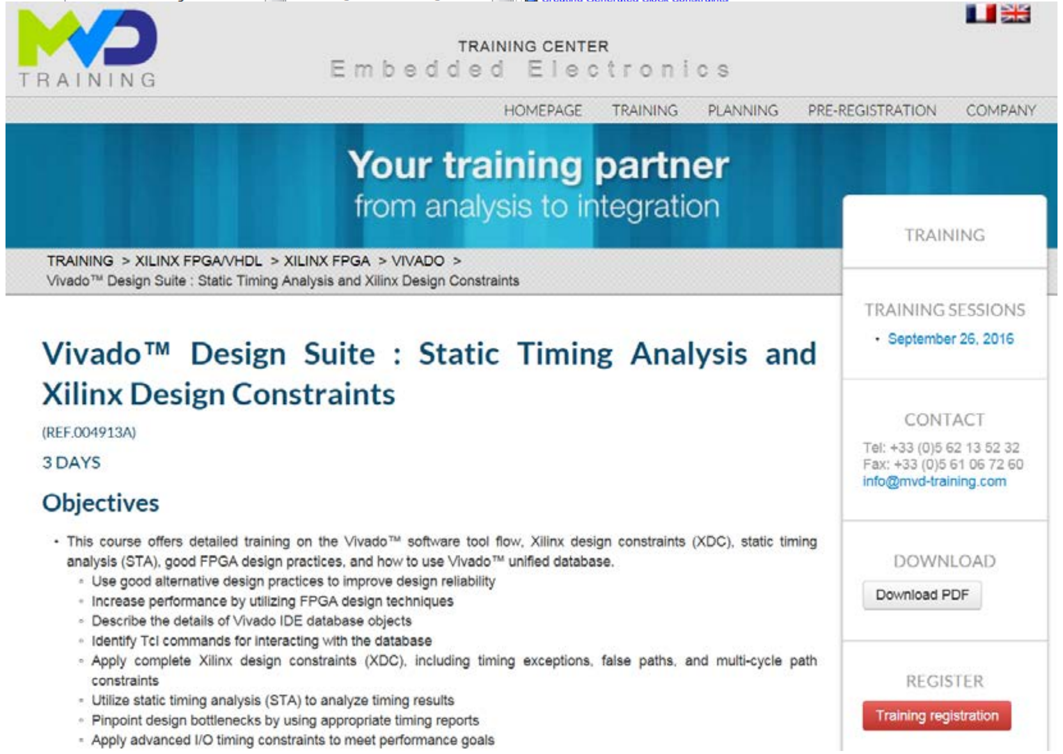
- Design Checklists
  - My Methodology Checklist
  - *Create Design Checklist...*
- Vivado Design Hubs
  - System-Level Design Flow
  - Design Flows Overview
  - I/O and Clock Planning
  - High-Level Synthesis (C based)
  - Designing with IP
  - Using IP Integrator
  - Applying Design Constraints
  - Power Estimation & Optimizat...
  - Logic Simulation
  - Logic Synthesis
  - Implementation
  - Partial Reconfiguration
  - System Generator for DSP
  - Timing Closure & Design Anal...

Applying Design Constraints
V2016.2 - Published 2016-06-08

Getting Started

Introduction
- Design Constraints Overview
- Vivado Design Suite Tutorial: Using Constraints
- Vivado Design Suite Tutorial: I/O and Clock Planning

Key Concepts
- UltraFast Vivado Design Methodology For Timing Closure
- Using the Vivado Timing Constraint Wizard
- Working with Constraint Sets
- Using the XDC Constraint Editor
- Creating Basic Clock Constraints
- Creating Generated Clock Constraints


TRAINING CENTER
Embedded Electronics

HOMEPAGE | TRAINING | PLANNING | PRE-REGISTRATION | COMPANY

Your training partner
from analysis to integration

TRAINING > XILINX FPGA/VHDL > XILINX FPGA > VIVADO >
Vivado™ Design Suite : Static Timing Analysis and Xilinx Design Constraints

Vivado™ Design Suite : Static Timing Analysis and Xilinx Design Constraints

(REF.004913A)

3 DAYS

Objectives

- This course offers detailed training on the Vivado™ software tool flow, Xilinx design constraints (XDC), static timing analysis (STA), good FPGA design practices, and how to use Vivado™ unified database.
  - Use good alternative design practices to improve design reliability
  - Increase performance by utilizing FPGA design techniques
  - Describe the details of Vivado IDE database objects
  - Identify Tcl commands for interacting with the database
  - Apply complete Xilinx design constraints (XDC), including timing exceptions, false paths, and multi-cycle path constraints
  - Utilize static timing analysis (STA) to analyze timing results
  - Pinpoint design bottlenecks by using appropriate timing reports
  - Apply advanced I/O timing constraints to meet performance goals

TRAINING

TRAINING SESSIONS
- September 26, 2016

CONTACT
Tel: +33 (0)5 62 13 52 32
Fax: +33 (0)5 61 06 72 60
info@mvd-training.com

DOWNLOAD
Download PDF

REGISTER
Training registration

XILINX ALL PROGRAMMABLE.

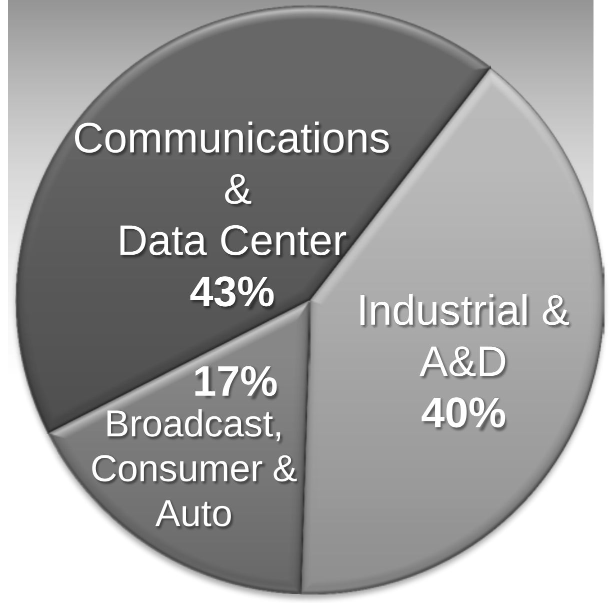
Summary

- UFDM → UG949
  - Use HDL Coding Guidelines
  - Avoid Reset whenever possible: Reset at startup by default!
- Use the Timing Constraints Wizard
  - Timing Constraints Editor available too
- Baseline the design first!
  - **`report_clock_interaction`** → Clocks are related by default in XDC (unlike UCF)
  - Start evaluate Constraints Post-Synthesis before running Implementation
- Timing Exceptions: Less is more!
- `report_timing_summary` – a complete view on the Design Timing

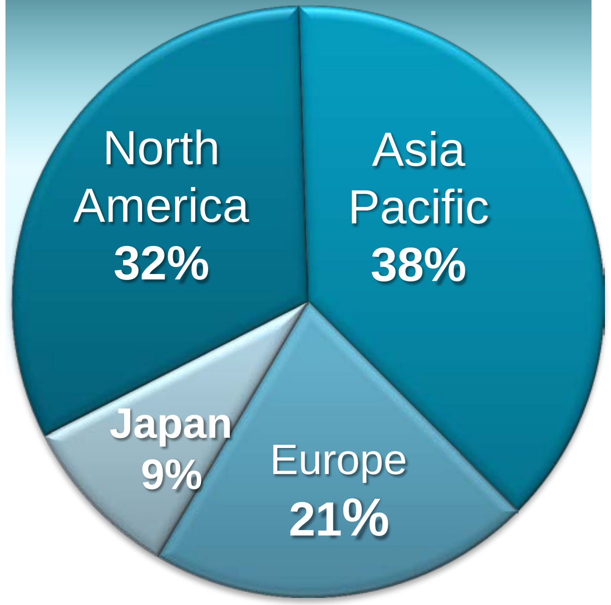
Thank You!

Revenue Breakdown – March 2016

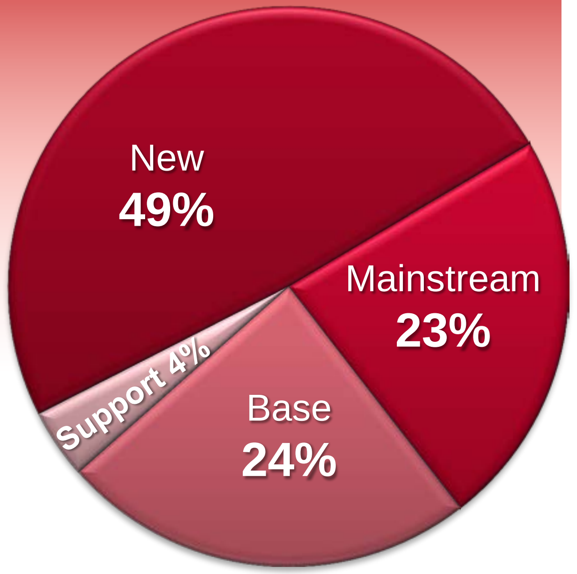
Revenue by End Market



Revenue by Geography

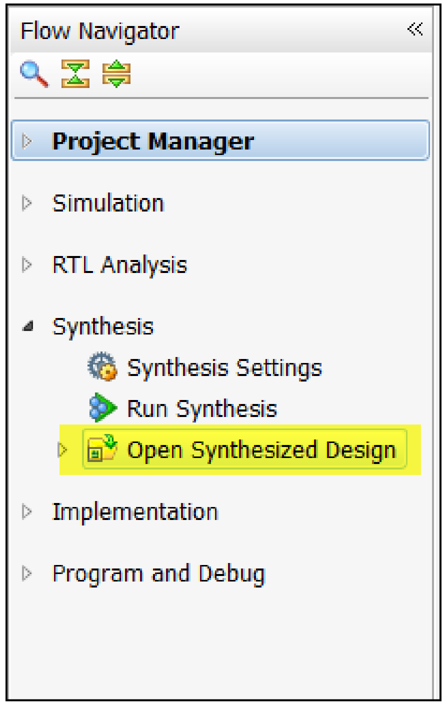


Revenue by Category



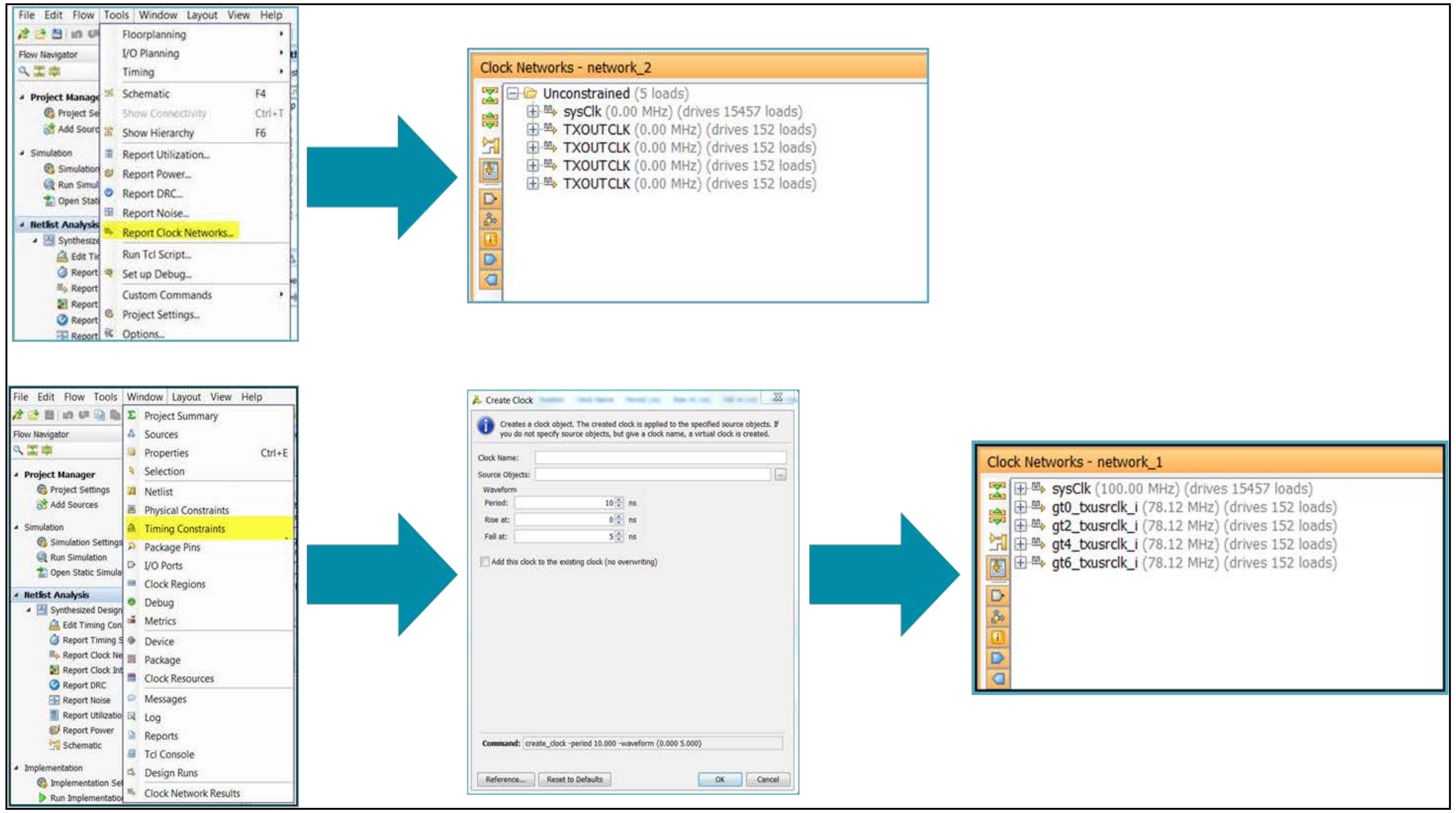
Labs 1 - 2

- **Lab 1: Open and Run synthesis on a project and Review timing summary**
  - Open Lab Project
    - Run Synthesis and Open Synthesized Design
  - Run report_timing_summary
    - Gauge timing after synthesis
- **Lab 2: Post-Synthesis design analysis for identifying constraint issues (clocks)**
  - Run report_clock_networks
    - Identify Missing Clocks
  - Create the missing clocks by using the XDC template
    - Examine if all clocks constrained correctly
  - Run report_clocks in the TCL console
    - Open the ASCII report file to view the clocks in the design



Using report_clock_networks

- **Q. How do I know when I am done constraining clocks?**
  - A. When `report_clock_networks` shows no unconstrained networks



Clock Networks - network_2

```
Unconstrained (5 loads)
  sysClk (0.00 MHz) (drives 15457 loads)
  TXOUTCLK (0.00 MHz) (drives 152 loads)
  TXOUTCLK (0.00 MHz) (drives 152 loads)
  TXOUTCLK (0.00 MHz) (drives 152 loads)
  TXOUTCLK (0.00 MHz) (drives 152 loads)
```

Create Clock

Creates a clock object. The created clock is applied to the specified source objects. If you do not specify source objects, but give a clock name, a virtual clock is created.

Clock Name:
Source Objects:
Waveform
Period: 10 ns
Rise at: 0 ns
Fall at: 5 ns
☐ Add this clock to the existing clock (no overwriting)

Command: `create_clock -period 10.000 -waveform {0.000 5.000}`

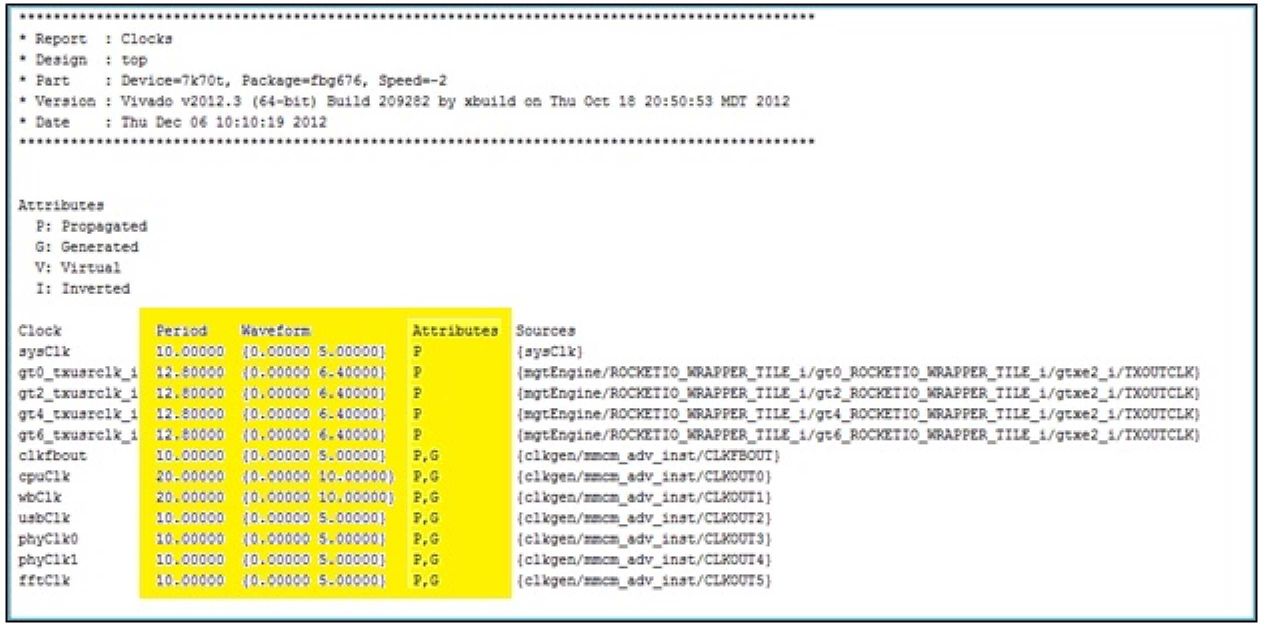
Reference... | Reset to Defaults | OK | Cancel

Clock Networks - network_1

```
sysClk (100.00 MHz) (drives 15457 loads)
gt0_txusrclk_i (78.12 MHz) (drives 152 loads)
gt2_txusrclk_i (78.12 MHz) (drives 152 loads)
gt4_txusrclk_i (78.12 MHz) (drives 152 loads)
gt6_txusrclk_i (78.12 MHz) (drives 152 loads)
```

Using report_clocks

- **Q. How do I make sure my clocks are correct?**
  - A. When `report_clocks` shows period, waveform, and attribute for every clock in the design



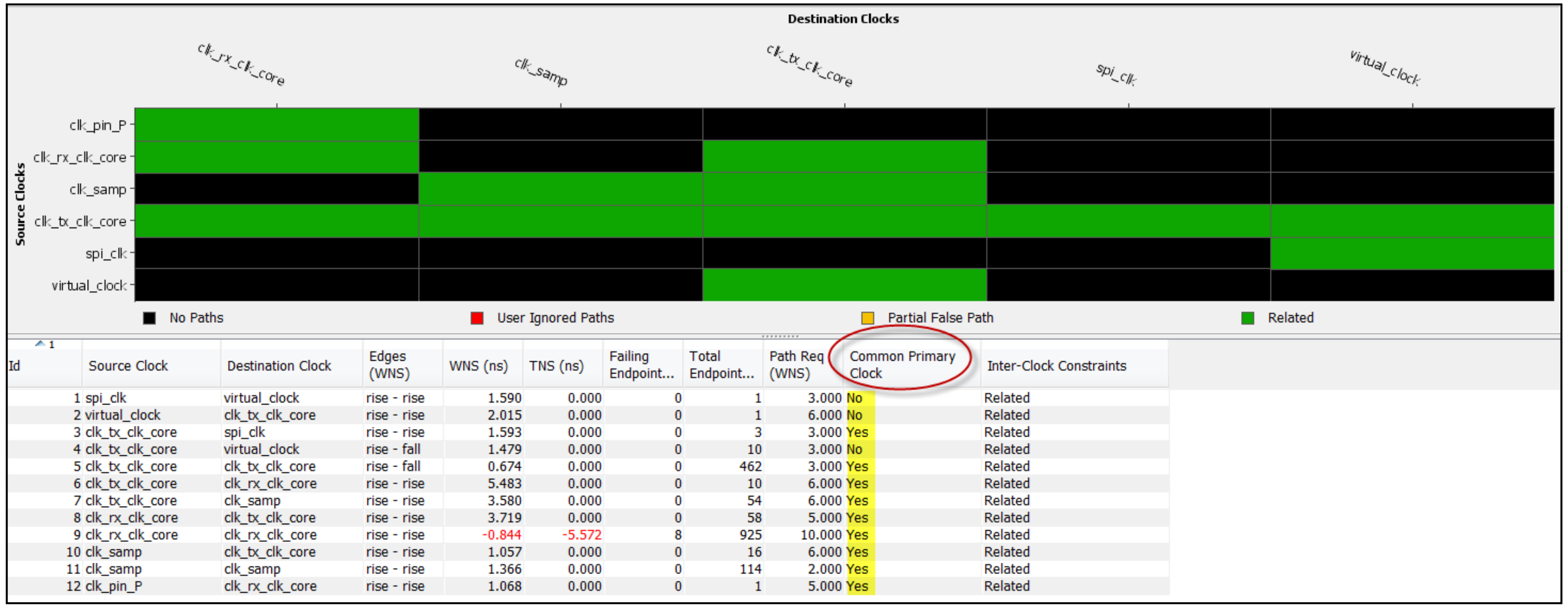
```
*******************************************************************************************
* Report  : Clocks
* Design  : top
* Part    : Device=7k70t, Package=fbg676, Speed=-2
* Version : Vivado v2012.3 (64-bit) Build 209282 by xbuild on Thu Oct 18 20:50:53 MDT 2012
* Date    : Thu Dec 06 10:10:19 2012
*******************************************************************************************


Attributes
  P: Propagated
  G: Generated
  V: Virtual
  I: Inverted

Clock             Period    Waveform              Attributes  Sources
sysClk            10.00000  {0.00000 5.00000}     P           {sysClk}
gt0_txusrclk_i    12.80000  {0.00000 6.40000}     P           {mgtEngine/ROCKETIO_WRAPPER_TILE_i/gt0_ROCKETIO_WRAPPER_TILE_i/gtxe2_i/TXOUTCLK}
gt2_txusrclk_i    12.80000  {0.00000 6.40000}     P           {mgtEngine/ROCKETIO_WRAPPER_TILE_i/gt2_ROCKETIO_WRAPPER_TILE_i/gtxe2_i/TXOUTCLK}
gt4_txusrclk_i    12.80000  {0.00000 6.40000}     P           {mgtEngine/ROCKETIO_WRAPPER_TILE_i/gt4_ROCKETIO_WRAPPER_TILE_i/gtxe2_i/TXOUTCLK}
gt6_txusrclk_i    12.80000  {0.00000 6.40000}     P           {mgtEngine/ROCKETIO_WRAPPER_TILE_i/gt6_ROCKETIO_WRAPPER_TILE_i/gtxe2_i/TXOUTCLK}
clkfbout          10.00000  {0.00000 5.00000}     P,G         {clkgen/mmcm_adv_inst/CLKFBOUT}
cpuClk            20.00000  {0.00000 10.00000}    P,G         {clkgen/mmcm_adv_inst/CLKOUT0}
wbClk             20.00000  {0.00000 10.00000}    P,G         {clkgen/mmcm_adv_inst/CLKOUT1}
usbClk            10.00000  {0.00000 5.00000}     P,G         {clkgen/mmcm_adv_inst/CLKOUT2}
phyClk0           10.00000  {0.00000 5.00000}     P,G         {clkgen/mmcm_adv_inst/CLKOUT3}
phyClk1           10.00000  {0.00000 5.00000}     P,G         {clkgen/mmcm_adv_inst/CLKOUT4}
fftClk            10.00000  {0.00000 5.00000}     P,G         {clkgen/mmcm_adv_inst/CLKOUT5}
```

Labs 3-4

- **Lab 3: Running `check_timing` and `report_clock_interaction`**
  - Run check_timing and review results
    - What issues did you find in this design?
    - Review other areas where check_timing might be useful in your design
  - Running `report_clock_interaction`
    - Analyze the report
    - Identify the column where clock relationships are identified
    - Identify if the path requirements for Setup and Hold
- **Lab 4: Constraining IOs**
  - Run `check_timing` in the TCL console
    - How many inputs are unconstrained
    - How many outputs are unconstrained
    - Use the XDC template to constrain the unconstrained IO ports

Clock Interactions

- **Q. How do I know what clocks should be related?**
  - A. `report_clock_interactions` – sort by Common Primary Clock



Destination Clocks

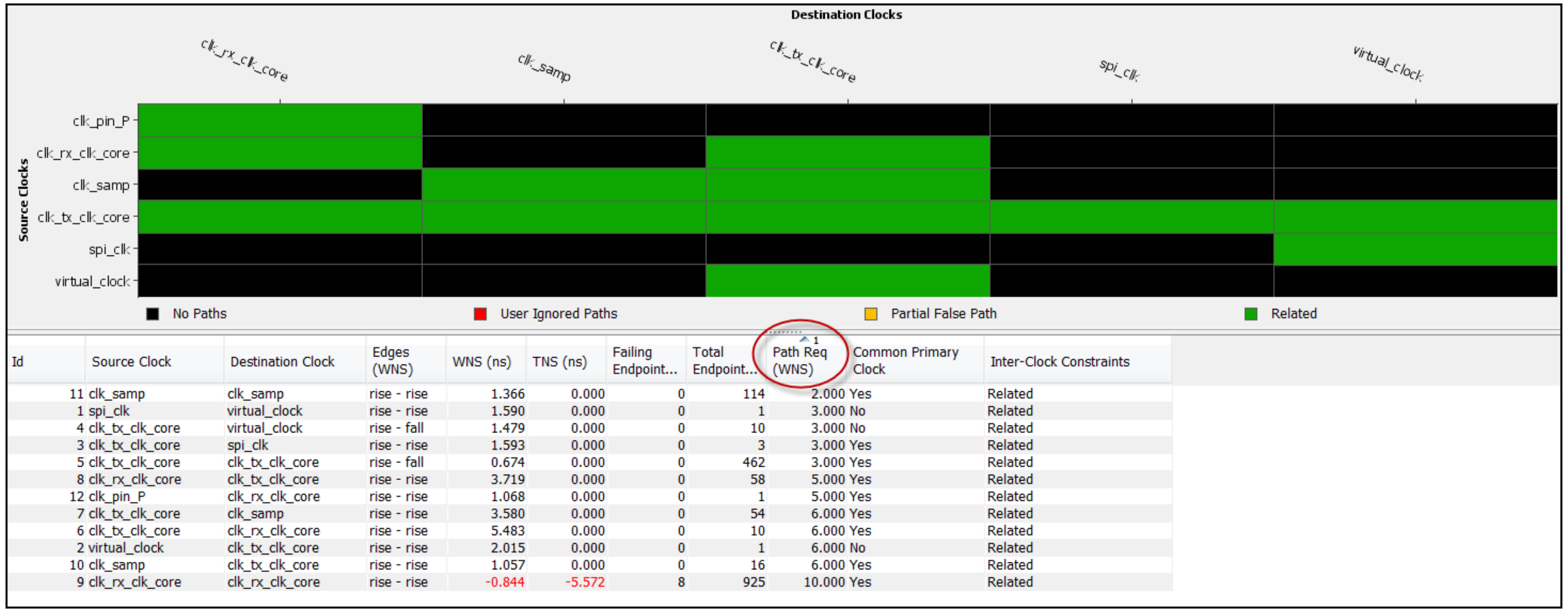
Source Clocks	clk_rx_clk_core	clk_samp	clk_tx_clk_core	spi_clk	virtual_clock
clk_pin_P	🟩	■	■	■	■
clk_rx_clk_core	🟩	■	🟩	■	■
clk_samp	■	🟩	🟩	■	■
clk_tx_clk_core	🟩	🟩	🟩	🟩	🟩
spi_clk	■	■	■	■	🟩
virtual_clock	■	■	🟩	■	■

■ No Paths 🟥 User Ignored Paths 🟨 Partial False Path 🟩 Related

Id	Source Clock	Destination Clock	Edges (WNS)	WNS (ns)	TNS (ns)	Failing Endpoint...	Total Endpoint...	Path Req (WNS)	Common Primary Clock	Inter-Clock Constraints
1	spi_clk	virtual_clock	rise - rise	1.590	0.000	0	1	3.000	No	Related
2	virtual_clock	clk_tx_clk_core	rise - rise	2.015	0.000	0	1	6.000	No	Related
3	clk_tx_clk_core	spi_clk	rise - rise	1.593	0.000	0	3	3.000	Yes	Related
4	clk_tx_clk_core	virtual_clock	rise - fall	1.479	0.000	0	10	3.000	No	Related
5	clk_tx_clk_core	clk_tx_clk_core	rise - fall	0.674	0.000	0	462	3.000	Yes	Related
6	clk_tx_clk_core	clk_rx_clk_core	rise - rise	5.483	0.000	0	10	6.000	Yes	Related
7	clk_tx_clk_core	clk_samp	rise - rise	3.580	0.000	0	54	6.000	Yes	Related
8	clk_rx_clk_core	clk_tx_clk_core	rise - rise	3.719	0.000	0	58	5.000	Yes	Related
9	clk_rx_clk_core	clk_rx_clk_core	rise - rise	-0.844	-5.572	8	925	10.000	Yes	Related
10	clk_samp	clk_tx_clk_core	rise - rise	1.057	0.000	0	16	6.000	Yes	Related
11	clk_samp	clk_samp	rise - rise	1.366	0.000	0	114	2.000	Yes	Related
12	clk_pin_P	clk_rx_clk_core	rise - rise	1.068	0.000	0	1	5.000	Yes	Related

Clock Interactions

- **Q. How do I know if I have unrealistic path requirements?**
  - A. `report_clock_interactions` – sort by Path Req (WNS)



Destination Clocks

Source Clocks	clk_rx_clk_core	clk_samp	clk_tx_clk_core	spi_clk	virtual_clock
clk_pin_P	🟩	⬛	⬛	⬛	⬛
clk_rx_clk_core	🟩	⬛	🟩	⬛	⬛
clk_samp	⬛	🟩	🟩	⬛	⬛
clk_tx_clk_core	🟩	🟩	🟩	🟩	🟩
spi_clk	⬛	⬛	⬛	⬛	🟩
virtual_clock	⬛	⬛	🟩	⬛	⬛

⬛ No Paths 🟥 User Ignored Paths 🟨 Partial False Path 🟩 Related

Id	Source Clock	Destination Clock	Edges (WNS)	WNS (ns)	TNS (ns)	Failing Endpoint...	Total Endpoint...	Path Req (WNS)	Common Primary Clock	Inter-Clock Constraints
11	clk_samp	clk_samp	rise - rise	1.366	0.000	0	114	2.000	Yes	Related
1	spi_clk	virtual_clock	rise - rise	1.590	0.000	0	1	3.000	No	Related
4	clk_tx_clk_core	virtual_clock	rise - fall	1.479	0.000	0	10	3.000	No	Related
3	clk_tx_clk_core	spi_clk	rise - rise	1.593	0.000	0	3	3.000	Yes	Related
5	clk_tx_clk_core	clk_tx_clk_core	rise - fall	0.674	0.000	0	462	3.000	Yes	Related
8	clk_rx_clk_core	clk_tx_clk_core	rise - rise	3.719	0.000	0	58	5.000	Yes	Related
12	clk_pin_P	clk_rx_clk_core	rise - rise	1.068	0.000	0	1	5.000	Yes	Related
7	clk_tx_clk_core	clk_samp	rise - rise	3.580	0.000	0	54	6.000	Yes	Related
6	clk_tx_clk_core	clk_rx_clk_core	rise - rise	5.483	0.000	0	10	6.000	Yes	Related
2	virtual_clock	clk_tx_clk_core	rise - rise	2.015	0.000	0	1	6.000	No	Related
10	clk_samp	clk_tx_clk_core	rise - rise	1.057	0.000	0	16	6.000	Yes	Related
9	clk_rx_clk_core	clk_rx_clk_core	rise - rise	-0.844	-5.572	8	925	10.000	Yes	Related

Lab 5-6

- **Lab 5: Cross-Probing features in Vivado**
  - Schematics, RTL, Device Floorplan, etc.
  - Run report_timing_summary
- **Lab 6: Last Mile of timing closure**
  - Is timing closure achieved?
  - Review timing results
    - Is the design fully constrained?
    - Are the timing constraints too pessimistic?
  - Run report_drc
  - Review all Critical Warning

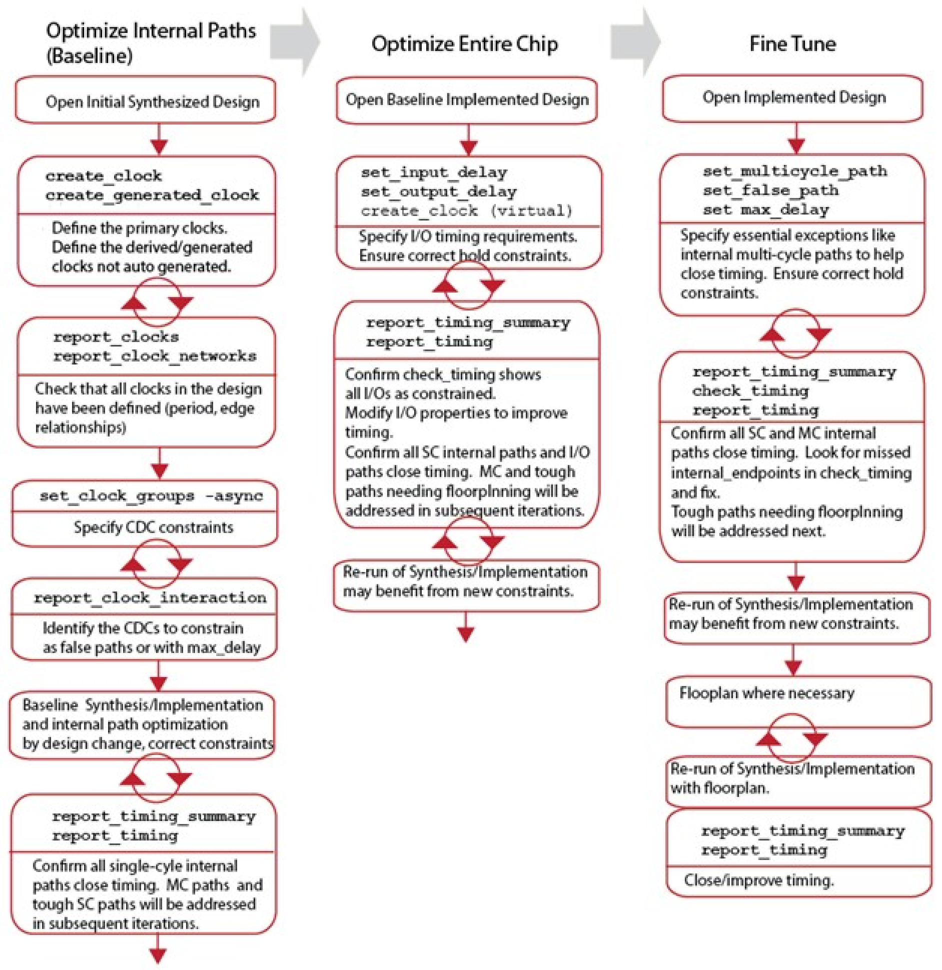
Cross-Probing

- **What is the short-cut key for opening the schematic?**
  - A. F4

© Copyright 2012 Xilinx

Performance Baselining

- This is our most detailed description of performance baselining



Optimize Internal Paths (Baseline)

Open Initial Synthesized Design

```
create_clock
create_generated_clock
```

Define the primary clocks. Define the derived/generated clocks not auto generated.

```
report_clocks
report_clock_networks
```

Check that all clocks in the design have been defined (period, edge relationships)

```
set_clock_groups -async
```

Specify CDC constraints

```
report_clock_interaction
```

Identify the CDCs to constrain as false paths or with max_delay

Baseline Synthesis/Implementation and internal path optimization by design change, correct constraints

```
report_timing_summary
report_timing
```

Confirm all single-cyle internal paths close timing. MC paths and tough SC paths will be addressed in subsequent iterations.

Optimize Entire Chip

Open Baseline Implemented Design

```
set_input_delay
set_output_delay
create_clock (virtual)
```

Specify I/O timing requirements. Ensure correct hold constraints.

```
report_timing_summary
report_timing
```

Confirm check_timing shows all I/Os as constrained. Modify I/O properties to improve timing. Confirm all SC internal paths and I/O paths close timing. MC and tough paths needing floorplnning will be addressed in subsequent iterations.

Re-run of Synthesis/Implementation may benefit from new constraints.

Fine Tune

Open Implemented Design

```
set_multicycle_path
set_false_path
set max_delay
```

Specify essential exceptions like internal multi-cycle paths to help close timing. Ensure correct hold constraints.

```
report_timing_summary
check_timing
report_timing
```

Confirm all SC and MC internal paths close timing. Look for missed internal_endpoints in check_timing and fix. Tough paths needing floorplnning will be addressed next.

Re-run of Synthesis/Implementation may benefit from new constraints.

Flooplan where necessary

Re-run of Synthesis/Implementation with floorplan.

```
report_timing_summary
report_timing
```

Close/improve timing.

Summary

- **Fully integrated design suite increases efficiency**
- **Creating and Validating Timing constraints are easy**
  - Use XDC template for help in creating constraints
  - Use the Vivado Design Software's report commands to debug and fine tune constraints
- **Send any feedback to balacha@xilinx.com**

Follow Xilinx on:

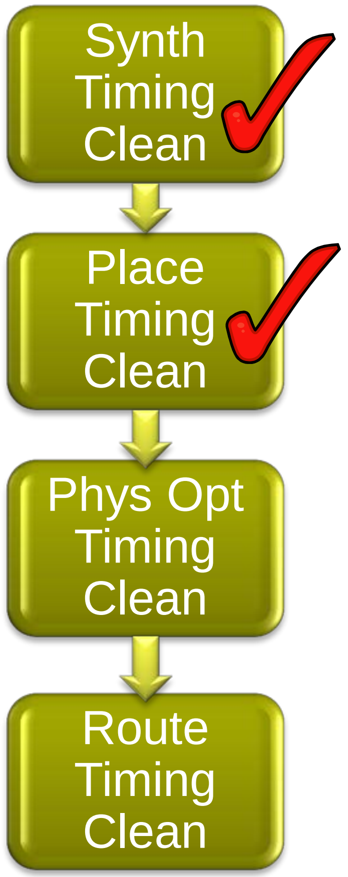
facebook.com/XilinxInc

twitter.com/#!/XilinxInc

youtube.com/XilinxInc

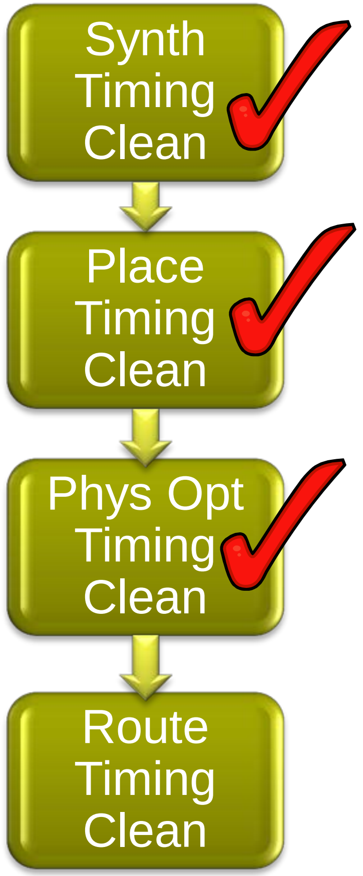
Timing Results Post Place Design

- **Assuming clean timing before place design.**
- **Typical causes of large timing violations:**
  - High fanout nets
  - Bad floorplan and/or bad IO placement
  - Over utilization
  - SLR crossings on SSI devices
- **Can go to phys-opt even if timing is not clean**
  - IFF reasons for bad WNS can be fixed by phys_opt
  - IFF there are not too many issues



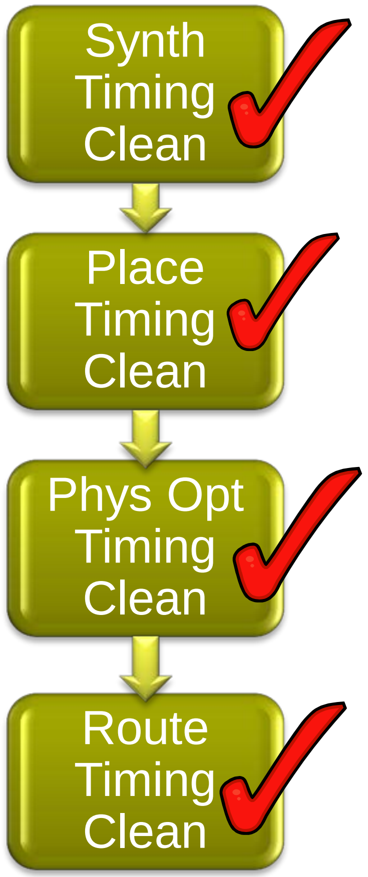
Timing Results Post Phys Opt Design

- **Assuming timing clean before phys-opt**
- **Typical causes of large timing violations:**
  - – Phys-opt only works on top offenders
    - Try looping with various options
  - – High fanout nets driven from LUTs
  - – DONT_TOUCH attribute preventing optimizations
  - – Replace
  - – Retime push FFs in/out of BRAMs/SRLs
- **Once timing is clean WNS better than -300ps**
  - – Go to route_design



Timing Results Post Route Design

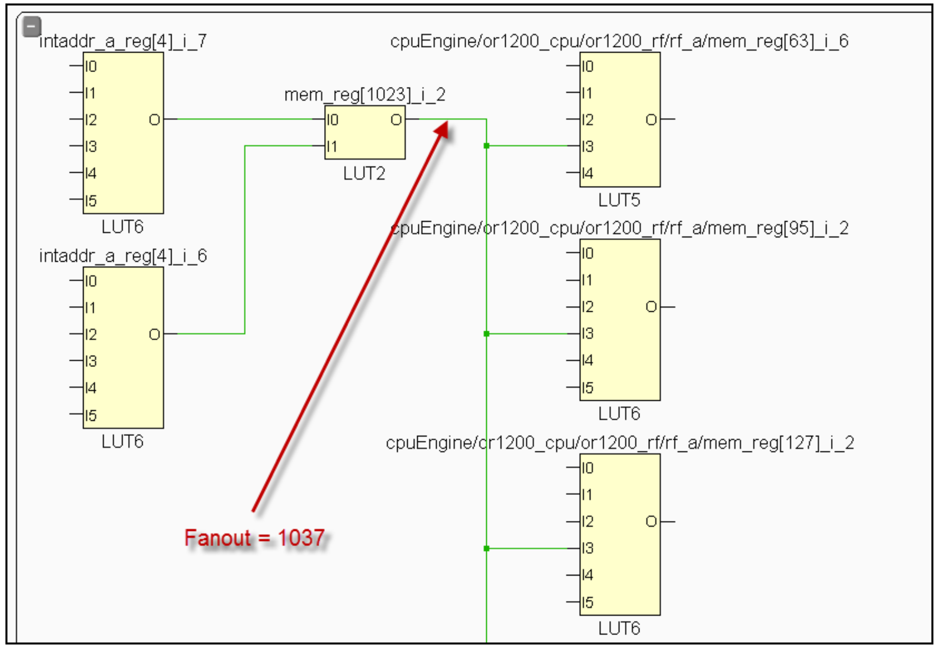
- **Assuming timing clean timing before route**
- **Typical causes of large timing violations:**
  - Hold fixing -> run route_design with:
    - set_false_path –hold –from [all_registers]
    - Report timing actual vs estimated
  - Congestion
- **Tips**
  - Overconstrain
  - Incremental placement
  - OOC for sub blocks

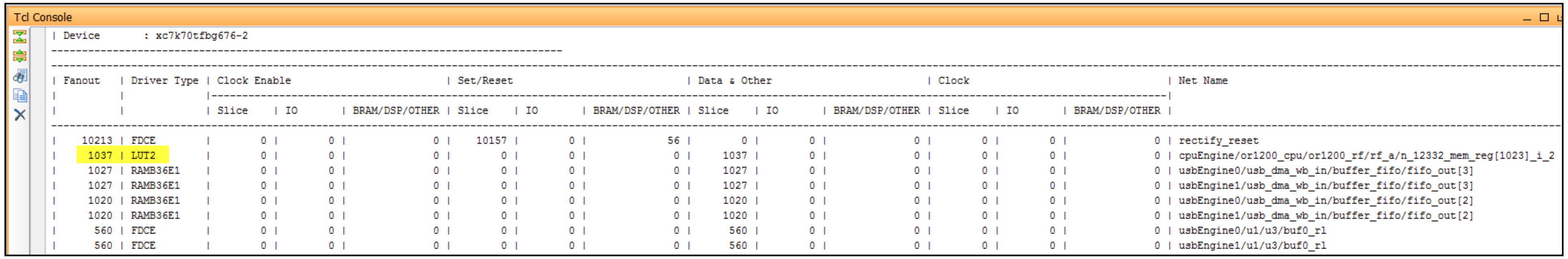


XILINX ALL PROGRAMMABLE.

High Fanout Nets Driven by LUTs

- Recommended to drive high fanout nets from a synchronous start point
- Identify high fanout nets driven by LUTs
  report_high_fanout_nets –load_types –max_nets 100
  - 2012.4 requires <u>placed</u> design
  - 2013.1 hope to be able to do this before placement

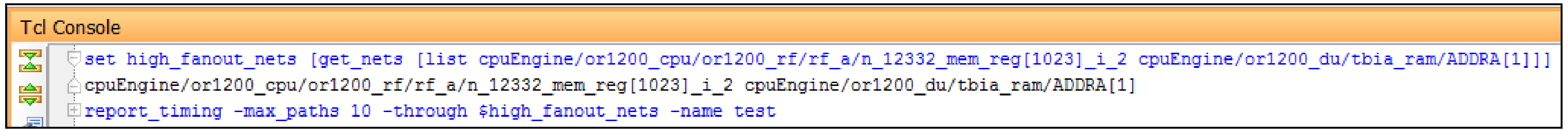

intaddr_a_reg[4]_i_7
I0
I1
I2
I3
I4
I5
O
LUT6
mem_reg[1023]_i_2
I0
I1
O
LUT2
cpuEngine/or1200_cpu/or1200_rf/rf_a/mem_reg[63]_i_6
LUT5
cpuEngine/or1200_cpu/or1200_rf/rf_a/mem_reg[95]_i_2
LUT6
intaddr_a_reg[4]_i_6
LUT6
cpuEngine/or1200_cpu/or1200_rf/rf_a/mem_reg[127]_i_2
LUT6
Fanout = 1037



```
| Device       : xc7k70tfbg676-2
-------------------------------------------------------------------------------

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
| Fanout    | Driver Type | Clock Enable                          | Set/Reset                             | Data & Other                          | Clock                                 | Net Name
|           |             |-----------------------------------------------------------------------------------------------------------------------------------------------------------|
|           |             | Slice     | IO        | BRAM/DSP/OTHER | Slice     | IO        | BRAM/DSP/OTHER | Slice     | IO        | BRAM/DSP/OTHER | Slice     | IO        | BRAM/DSP/OTHER |
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
|     10213 | FDCE        |         0 |         0 |              0 |     10157 |         0 |             56 |         0 |         0 |              0 |         0 |         0 |              0 | rectify_reset
|      1037 | LUT2        |         0 |         0 |              0 |         0 |         0 |              0 |      1037 |         0 |              0 |         0 |         0 |              0 | cpuEngine/or1200_cpu/or1200_rf/rf_a/n_12332_mem_reg[1023]_i_2
|      1027 | RAMB36E1    |         0 |         0 |              0 |         0 |         0 |              0 |      1027 |         0 |              0 |         0 |         0 |              0 | usbEngine0/usb_dma_wb_in/buffer_fifo/fifo_out[3]
|      1027 | RAMB36E1    |         0 |         0 |              0 |         0 |         0 |              0 |      1027 |         0 |              0 |         0 |         0 |              0 | usbEngine1/usb_dma_wb_in/buffer_fifo/fifo_out[3]
|      1020 | RAMB36E1    |         0 |         0 |              0 |         0 |         0 |              0 |      1020 |         0 |              0 |         0 |         0 |              0 | usbEngine0/usb_dma_wb_in/buffer_fifo/fifo_out[2]
|      1020 | RAMB36E1    |         0 |         0 |              0 |         0 |         0 |              0 |      1020 |         0 |              0 |         0 |         0 |              0 | usbEngine1/usb_dma_wb_in/buffer_fifo/fifo_out[2]
|       560 | FDCE        |         0 |         0 |              0 |         0 |         0 |              0 |       560 |         0 |              0 |         0 |         0 |              0 | usbEngine0/u1/u3/buf0_rl
|       560 | FDCE        |         0 |         0 |              0 |         0 |         0 |              0 |       560 |         0 |              0 |         0 |         0 |              0 | usbEngine1/u1/u3/buf0_rl
```

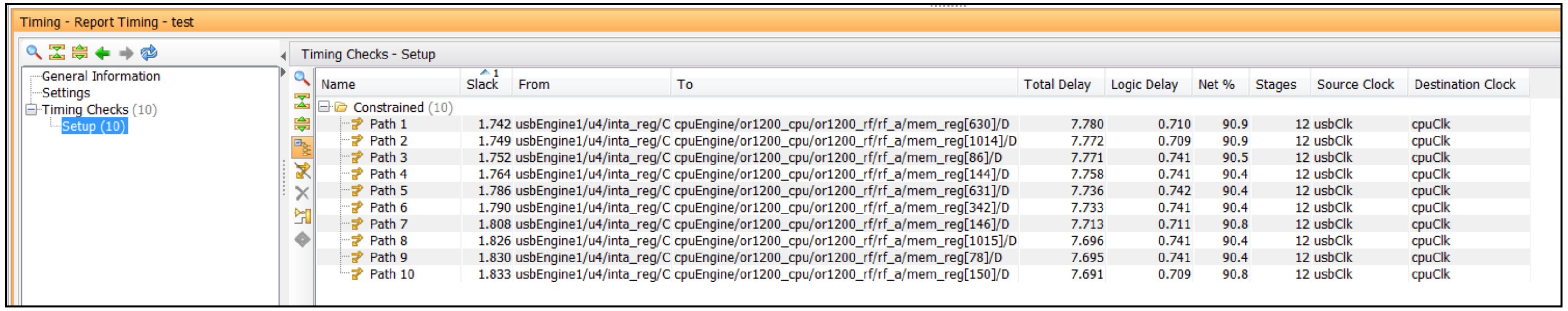
High Fanout Nets Driven by LUTs

- Upon identifying a group of high fanout nets driven by LUTs, use **report_timing –through** to evaluate timing



Tcl Console

```
set high_fanout_nets [get_nets [list cpuEngine/or1200_cpu/or1200_rf/rf_a/n_12332_mem_reg[1023]_i_2 cpuEngine/or1200_du/tbia_ram/ADDRA[1]]]
cpuEngine/or1200_cpu/or1200_rf/rf_a/n_12332_mem_reg[1023]_i_2 cpuEngine/or1200_du/tbia_ram/ADDRA[1]
report_timing -max_paths 10 -through $high_fanout_nets -name test
```



Timing - Report Timing - test

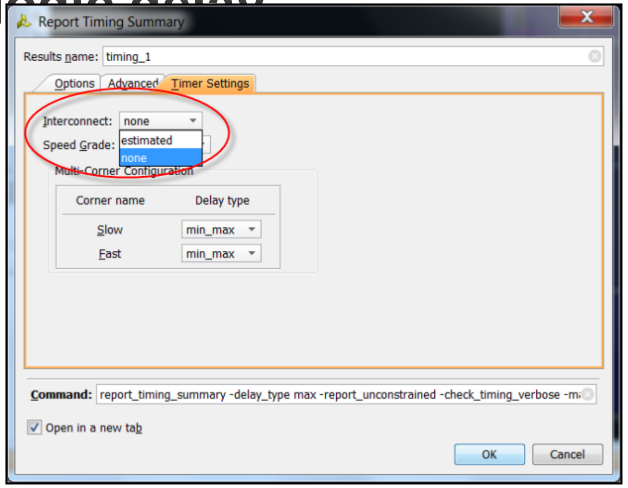
Timing Checks - Setup

Name	Slack	From	To	Total Delay	Logic Delay	Net %	Stages	Source Clock	Destination Clock
Constrained (10)									
Path 1	1.742	usbEngine1/u4/inta_reg/C	cpuEngine/or1200_cpu/or1200_rf/rf_a/mem_reg[630]/D	7.780	0.710	90.9	12	usbClk	cpuClk
Path 2	1.749	usbEngine1/u4/inta_reg/C	cpuEngine/or1200_cpu/or1200_rf/rf_a/mem_reg[1014]/D	7.772	0.709	90.9	12	usbClk	cpuClk
Path 3	1.752	usbEngine1/u4/inta_reg/C	cpuEngine/or1200_cpu/or1200_rf/rf_a/mem_reg[86]/D	7.771	0.741	90.5	12	usbClk	cpuClk
Path 4	1.764	usbEngine1/u4/inta_reg/C	cpuEngine/or1200_cpu/or1200_rf/rf_a/mem_reg[144]/D	7.758	0.741	90.4	12	usbClk	cpuClk
Path 5	1.786	usbEngine1/u4/inta_reg/C	cpuEngine/or1200_cpu/or1200_rf/rf_a/mem_reg[631]/D	7.736	0.742	90.4	12	usbClk	cpuClk
Path 6	1.790	usbEngine1/u4/inta_reg/C	cpuEngine/or1200_cpu/or1200_rf/rf_a/mem_reg[342]/D	7.733	0.741	90.4	12	usbClk	cpuClk
Path 7	1.808	usbEngine1/u4/inta_reg/C	cpuEngine/or1200_cpu/or1200_rf/rf_a/mem_reg[146]/D	7.713	0.711	90.8	12	usbClk	cpuClk
Path 8	1.826	usbEngine1/u4/inta_reg/C	cpuEngine/or1200_cpu/or1200_rf/rf_a/mem_reg[1015]/D	7.696	0.741	90.4	12	usbClk	cpuClk
Path 9	1.830	usbEngine1/u4/inta_reg/C	cpuEngine/or1200_cpu/or1200_rf/rf_a/mem_reg[78]/D	7.695	0.741	90.4	12	usbClk	cpuClk
Path 10	1.833	usbEngine1/u4/inta_reg/C	cpuEngine/or1200_cpu/or1200_rf/rf_a/mem_reg[150]/D	7.691	0.709	90.8	12	usbClk	cpuClk

Long Logic Paths

- Identifying long logic paths is useful to diagnose where difficult QoR challenges exist
- Identifying longest logic paths measured by logic levels is helpful, but doesn't always tell the full story
- Identifying longest paths measured by logic delay without routing

```
set_delay_model -interconnect none
report_timing_summary -max_paths 10
```



Results name: timing_1

Interconnect: none

Speed Grade:

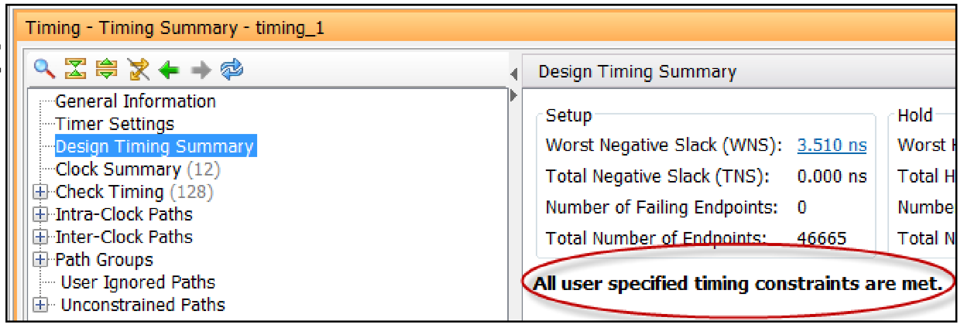
Corner name	Delay type
Slow	min_max
Fast	min_max

Command: report_timing_summary -delay_type max -report_unconstrained -check_timing_verbose -m

✓ Open in a new tab

Long Logic Paths

- **Negative slack in timing report with no routing means timing closure not achievable**
- **Of larger importance is the nature of the timing paths in each clock domain**
- **Most frequent offenders**
  - Paths sourced by unregistered BRAMs
  - Paths sourced by SRL
  - Paths containing unregistered, cascaded DSP blocks
  - Paths with large number of logic levels

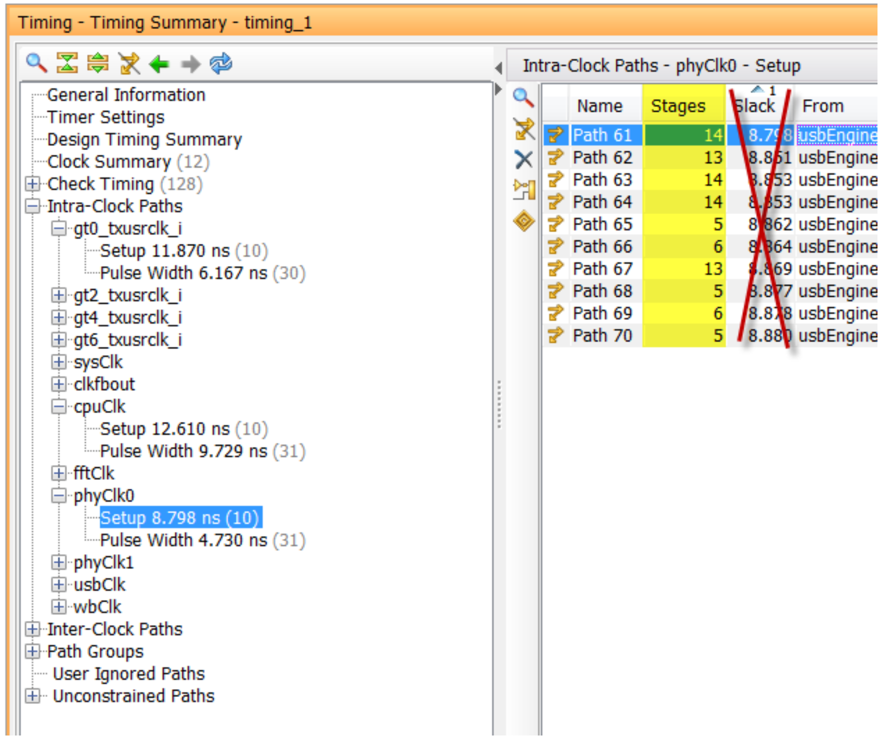

Timing - Timing Summary - timing_1

- General Information
- Timer Settings
- Design Timing Summary
- Clock Summary (12)
- Check Timing (128)
- Intra-Clock Paths
- Inter-Clock Paths
- Path Groups
- User Ignored Paths
- Unconstrained Paths

Design Timing Summary

Setup		Hold
Worst Negative Slack (WNS):	3.510 ns	Worst
Total Negative Slack (TNS):	0.000 ns	Total H
Number of Failing Endpoints:	0	Numbe
Total Number of Endpoints:	46665	Total N

All user specified timing constraints are met.


Timing - Timing Summary - timing_1

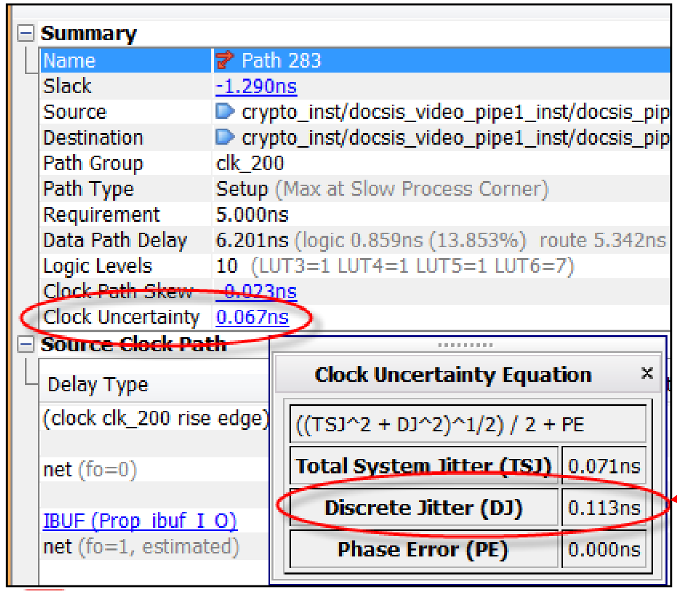
- General Information
- Timer Settings
- Design Timing Summary
- Clock Summary (12)
- Check Timing (128)
- Intra-Clock Paths
  - gt0_txusrclk_i
    - Setup 11.870 ns (10)
    - Pulse Width 6.167 ns (30)
  - gt2_txusrclk_i
  - gt4_txusrclk_i
  - gt6_txusrclk_i
  - sysClk
  - clkfbout
  - cpuClk
    - Setup 12.610 ns (10)
    - Pulse Width 9.729 ns (31)
  - fftClk
  - phyClk0
    - Setup 8.798 ns (10)
    - Pulse Width 4.730 ns (31)
  - phyClk1
  - usbClk
  - wbClk
- Inter-Clock Paths
- Path Groups
- User Ignored Paths
- Unconstrained Paths

Intra-Clock Paths - phyClk0 - Setup

Name	Stages	~~Slack~~	From
Path 61	14	~~8.798~~	usbEngine
Path 62	13	~~8.851~~	usbEngine
Path 63	14	~~8.853~~	usbEngine
Path 64	14	~~8.853~~	usbEngine
Path 65	5	~~8.862~~	usbEngine
Path 66	6	~~8.864~~	usbEngine
Path 67	13	~~8.869~~	usbEngine
Path 68	5	~~8.877~~	usbEngine
Path 69	6	~~8.878~~	usbEngine
Path 70	5	~~8.880~~	usbEngine

The Best Kept Secret To Acquire (almost) Free Timing Margin

- Many Xilinx customers do not pay close attention to jitter when setting up their clocks
- Unintended consequence is that they are leaving timing margin on the table in the form of higher than necessary clock uncertainty
- Clock uncertainty = ((TSJ^2 + DJ^2)^1/2)/2 + PE



Summary

Name	Path 283
Slack	-1.290ns
Source	crypto_inst/docsis_video_pipe1_inst/docsis_pip
Destination	crypto_inst/docsis_video_pipe1_inst/docsis_pip
Path Group	clk_200
Path Type	Setup (Max at Slow Process Corner)
Requirement	5.000ns
Data Path Delay	6.201ns (logic 0.859ns (13.853%) route 5.342ns
Logic Levels	10 (LUT3=1 LUT4=1 LUT5=1 LUT6=7)
Clock Path Skew	-0.023ns
Clock Uncertainty	0.067ns

Source Clock Path

Delay Type

(clock clk_200 rise edge)

net (fo=0)

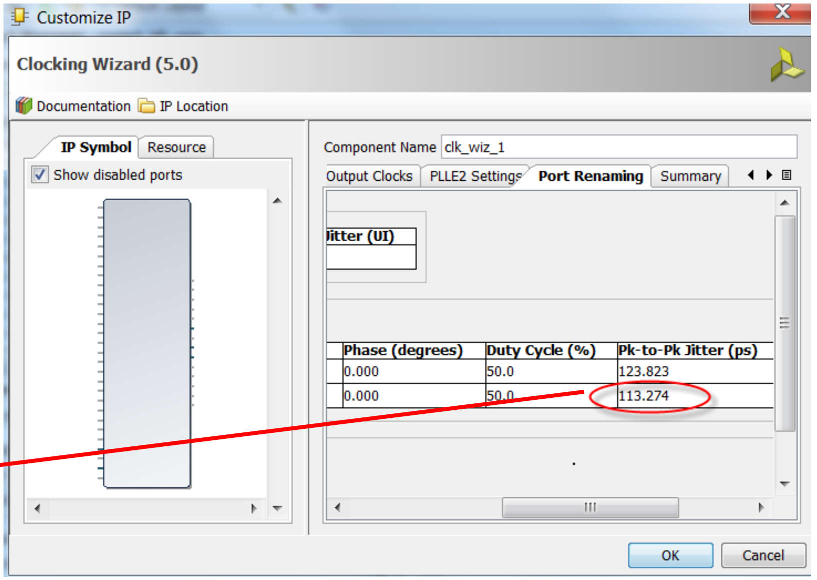
IBUF (Prop_ibuf_I_O)

net (fo=1, estimated)

Clock Uncertainty Equation

((TSJ^2 + DJ^2)^1/2) / 2 + PE

Total System Jitter (TSJ)	0.071ns
Discrete Jitter (DJ)	0.113ns
Phase Error (PE)	0.000ns



Customize IP

Clocking Wizard (5.0)

Documentation IP Location

IP Symbol | Resource

☑ Show disabled ports

Component Name: clk_wiz_1

Output Clocks | PLLE2 Settings | Port Renaming | Summary

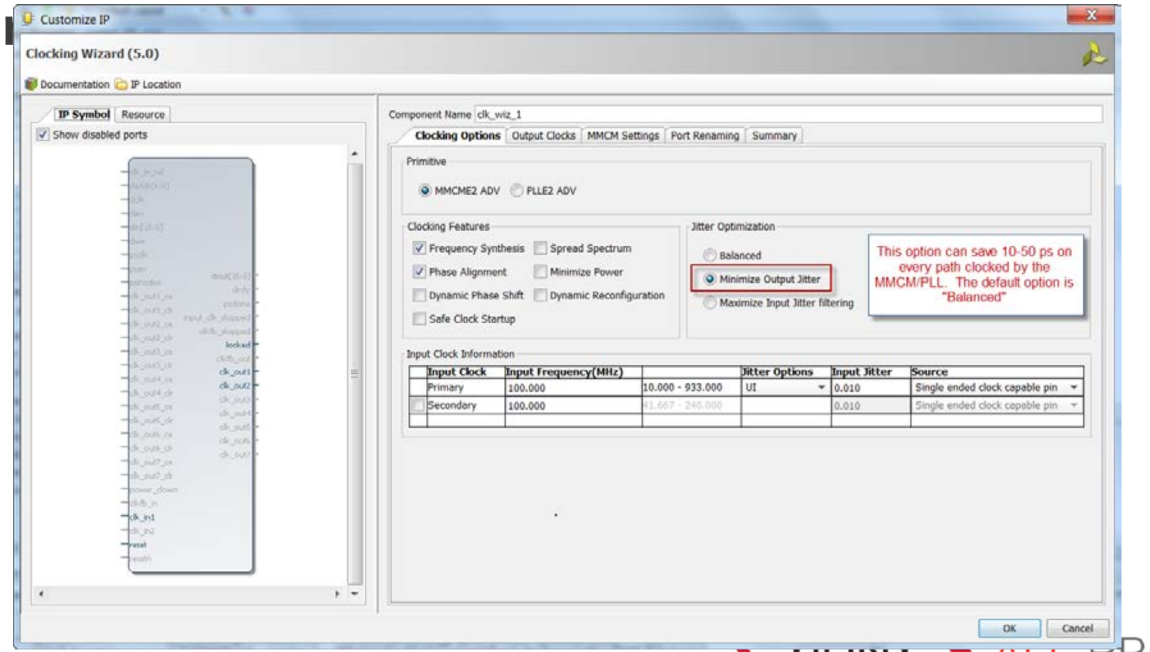
Jitter (UI)

Phase (degrees)	Duty Cycle (%)	Pk-to-Pk Jitter (ps)
0.000	50.0	123.823
0.000	50.0	113.274

OK Cancel

The Best Kept Secret To Acquire (almost) Free Timing Margin

- **Observation: Running VCO as fast as possible often reduces calculated clock uncertainty in order to buy small amount of margin across thousands of paths at the expense of slightly higher power**
- **Recommendataion: Try different options with clock wizard to emulate customer MMCM/PLL configuration to identify best peak-to-peak jitter per**



Component Name clk_wiz_1

Clocking Options | Output Clocks | MMCM Settings | Port Renaming | Summary

Primitive: [✓ MMCME2 ADV] PLLE2 ADV

Clocking Features:
- ✓ Frequency Synthesis
- ✓ Phase Alignment
- Dynamic Phase Shift
- Safe Clock Startup
- Spread Spectrum
- Minimize Power
- Dynamic Reconfiguration

Jitter Optimization: Balanced [✓ Minimize Output Jitter] Maximize Input Jitter filtering

This option can save 10-50 ps on every path clocked by the MMCM/PLL. The default option is "Balanced"

Input Clock Information

Input Clock	Input Frequency(MHz)		Jitter Options	Input Jitter	Source
Primary	100.000	10.000 - 933.000	UI	0.010	Single ended clock capable pin
Secondary	100.000	41.667 - 240.000		0.010	Single ended clock capable pin

The Best Kept Secret To Acquire (almost) Free Timing Margin

- Two real examples & demo

MMCM/PLL	Jitter Opt	VCO (MHz)	Clkout3 Pk-to-pk jitter (ps)	Uncertainty based on 71 ps system jitter (ps)
MMCM	Balanced	1000	113	67
MMCM	Min Out Jitter	1400	90	57
PLL	Min Out Jitter	1800	72	51 ★

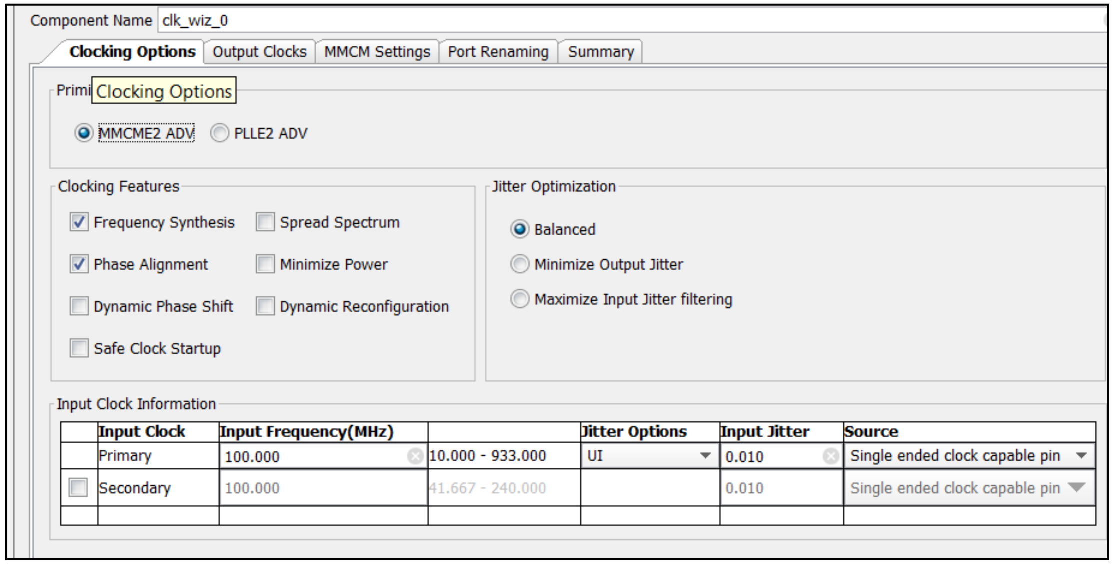
Saved 16 ps per path over thousands of paths!!

MMCM/PLL	Jitter Opt	VCO (MHz)	Clkout3 Pk-to-pk jitter (ps)	Uncertainty based on 71 ps system jitter (ps)
MMCM	Balanced	1000	356	182
MMCM	Min Out Jitter	1250	226	118
PLL	Min Out Jitter	1500	178	95 ★

Saved 87 ps per path on thousands of paths!!

Demo

- **Input clock**
  - 100 MHz
- **Output clocks**
  - 85 MHz
  - 340 MHz
- **Jitter**
  - 85 MHz (242 ps)
  - 340 MHz (200 ps)



Component Name: clk_wiz_0

[✓ MMCME2 ADV] PLLE2 ADV

Clocking Features

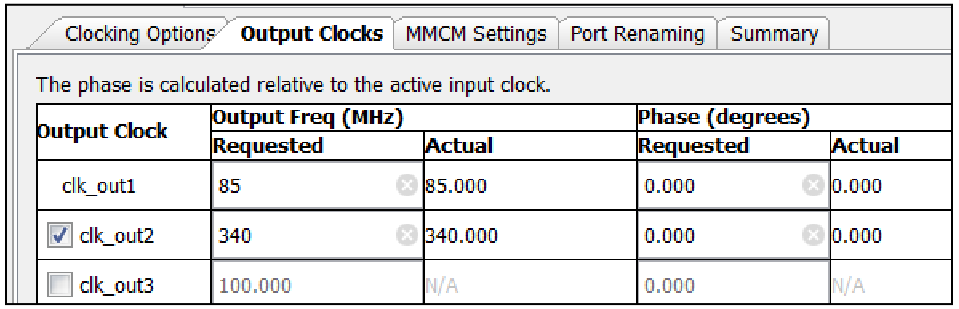
- [✓] Frequency Synthesis
- [ ] Spread Spectrum
- [✓] Phase Alignment
- [ ] Minimize Power
- [ ] Dynamic Phase Shift
- [ ] Dynamic Reconfiguration
- [ ] Safe Clock Startup

Jitter Optimization

[✓ Balanced] Minimize Output Jitter / Maximize Input Jitter filtering

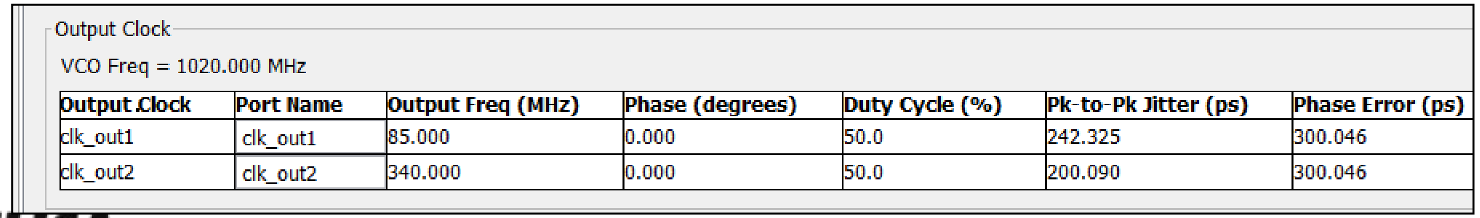
Input Clock Information

	Input Clock	Input Frequency(MHz)		Jitter Options	Input Jitter	Source
	Primary	100.000	10.000 - 933.000	UI	0.010	Single ended clock capable pin
☐	Secondary	100.000	41.667 - 240.000		0.010	Single ended clock capable pin



The phase is calculated relative to the active input clock.

Output Clock	Output Freq (MHz)		Phase (degrees)	
	Requested	Actual	Requested	Actual
clk_out1	85	85.000	0.000	0.000
✓ clk_out2	340	340.000	0.000	0.000
☐ clk_out3	100.000	N/A	0.000	N/A



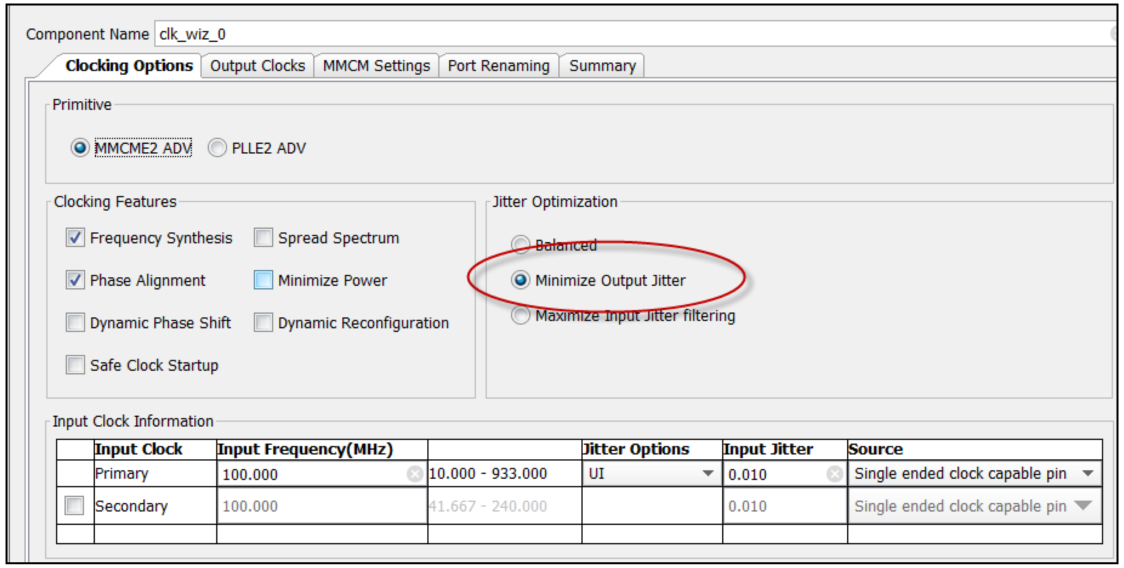
Output Clock

VCO Freq = 1020.000 MHz

Output Clock	Port Name	Output Freq (MHz)	Phase (degrees)	Duty Cycle (%)	Pk-to-Pk Jitter (ps)	Phase Error (ps)
clk_out1	clk_out1	85.000	0.000	50.0	242.325	300.046
clk_out2	clk_out2	340.000	0.000	50.0	200.090	300.046

Demo

- **Change Jitter Opt**
  - Minimize Output Jitter
- **Swap output clocks**
  - Faster clock on clk_out1 (CLKOUT0 of MMCM) allows for use of fractional divider which results in higher VCO
- **Jitter**
  - 85 MHz (113 ps)
  - 340 MHz (88 ps)


Component Name clk_wiz_0

Clocking Options | Output Clocks | MMCM Settings | Port Renaming | Summary

Primitive

● MMCME2 ADV ○ PLLE2 ADV

Clocking Features

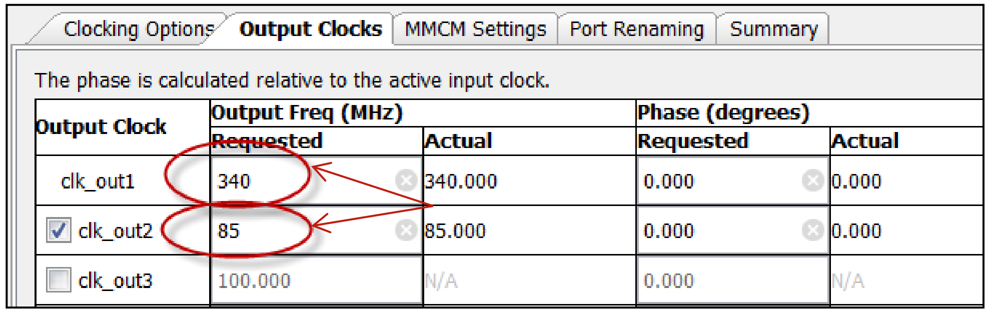
- ✓ Frequency Synthesis
- ☐ Spread Spectrum
- ✓ Phase Alignment
- ☐ Minimize Power
- ☐ Dynamic Phase Shift
- ☐ Dynamic Reconfiguration
- ☐ Safe Clock Startup

Jitter Optimization

- ○ Balanced
- ● Minimize Output Jitter
- ○ Maximize Input Jitter filtering

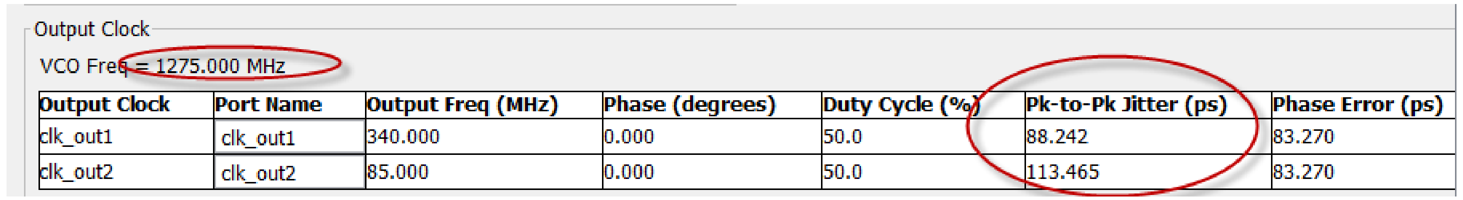
Input Clock Information

	Input Clock	Input Frequency(MHz)		Jitter Options	Input Jitter	Source
	Primary	100.000	10.000 - 933.000	UI	0.010	Single ended clock capable pin
☐	Secondary	100.000	41.667 - 240.000		0.010	Single ended clock capable pin


Clocking Options | Output Clocks | MMCM Settings | Port Renaming | Summary

The phase is calculated relative to the active input clock.

Output Clock	Output Freq (MHz)		Phase (degrees)	
	Requested	Actual	Requested	Actual
clk_out1	340	340.000	0.000	0.000
✓ clk_out2	85	85.000	0.000	0.000
☐ clk_out3	100.000	N/A	0.000	N/A

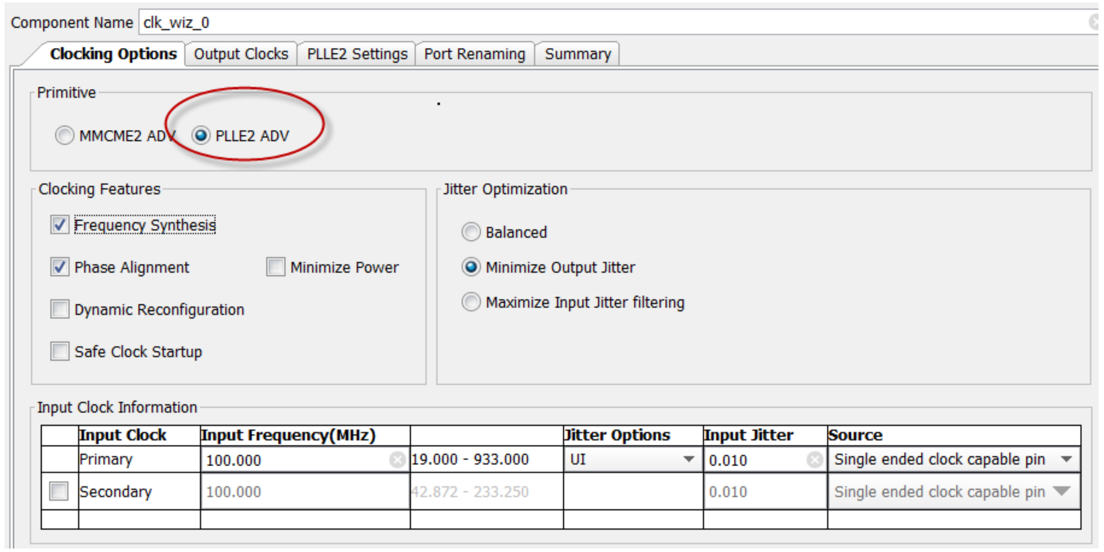

Output Clock

VCO Freq = 1275.000 MHz

Output Clock	Port Name	Output Freq (MHz)	Phase (degrees)	Duty Cycle (%)	Pk-to-Pk Jitter (ps)	Phase Error (ps)
clk_out1	clk_out1	340.000	0.000	50.0	88.242	83.270
clk_out2	clk_out2	85.000	0.000	50.0	113.465	83.270

Demo

- **Change Primitive**
  - Use PLL
- **Same output clocks**



Component Name: clk_wiz_0

Clocking Options | Output Clocks | PLLE2 Settings | Port Renaming | Summary

Primitive

○ MMCME2 ADV [● PLLE2 ADV]

Clocking Features

- ☑ Frequency Synthesis
- ☑ Phase Alignment
- ☐ Minimize Power
- ☐ Dynamic Reconfiguration
- ☐ Safe Clock Startup

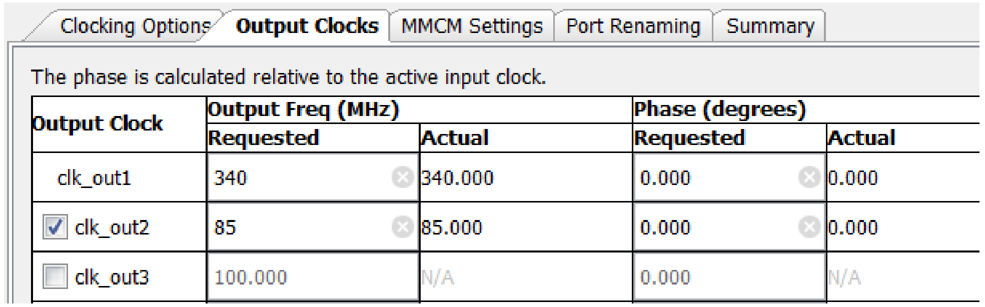
Jitter Optimization

- ○ Balanced
- ● Minimize Output Jitter
- ○ Maximize Input Jitter filtering

Input Clock Information

	Input Clock	Input Frequency(MHz)		Jitter Options	Input Jitter	Source
	Primary	100.000	19.000 - 933.000	UI	0.010	Single ended clock capable pin
☐	Secondary	100.000	42.872 - 233.250		0.010	Single ended clock capable pin

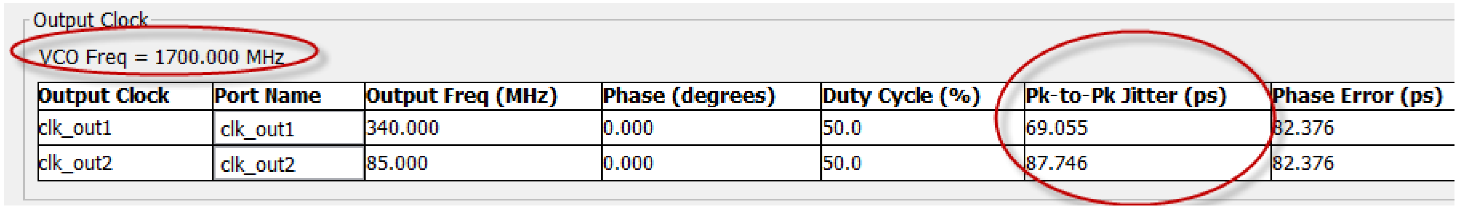
- **Jitter**
  - 85 MHz (87 ps)
  - 340 MHz (69 ps)



Clocking Options | Output Clocks | MMCM Settings | Port Renaming | Summary

The phase is calculated relative to the active input clock.

	Output Freq (MHz)		Phase (degrees)	
Output Clock	Requested	Actual	Requested	Actual
clk_out1	340	340.000	0.000	0.000
☑ clk_out2	85	85.000	0.000	0.000
☐ clk_out3	100.000	N/A	0.000	N/A



Output Clock

VCO Freq = 1700.000 MHz

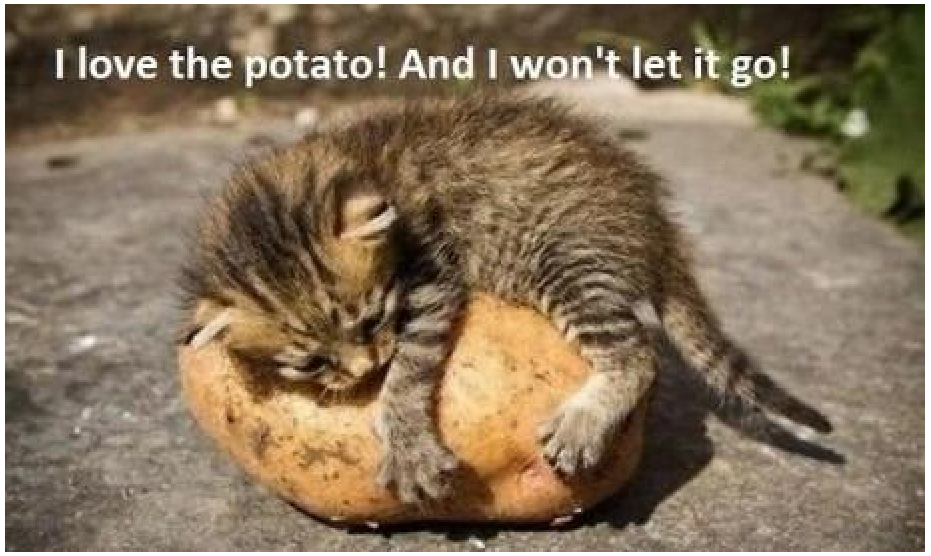
Output Clock	Port Name	Output Freq (MHz)	Phase (degrees)	Duty Cycle (%)	Pk-to-Pk Jitter (ps)	Phase Error (ps)
clk_out1	clk_out1	340.000	0.000	50.0	69.055	82.376
clk_out2	clk_out2	85.000	0.000	50.0	87.746	82.376

Agenda

- New Tricks with the IDE
- The Best Kept Secret to (almost) Free Timing Margin
- **Sweeping Vivado Directives with Tcl**

ISE Tools That Were Tough To Let Go

- **SmartXplorer**
  - Ability to run multiple implementation runs with different tool options on a number of different hosts
- **Cost Tables**
  - Ability to slightly perturb initial random placement to “hopefully” produce a different “slightly better” timing result



Vivado Tool For Running Multiple Builds

- **Directive: for non-project mode**
  - “directs” the behavior of a command to choose a set of algorithms
  - Building blocks for strategies
- **Uses different algorithms**
  - Not random seeds like ISE cost tables
  - More consistent behavior
- **But…….many people still ask for Smartxplorer & Seeds**

Directives >> Cost Tables !

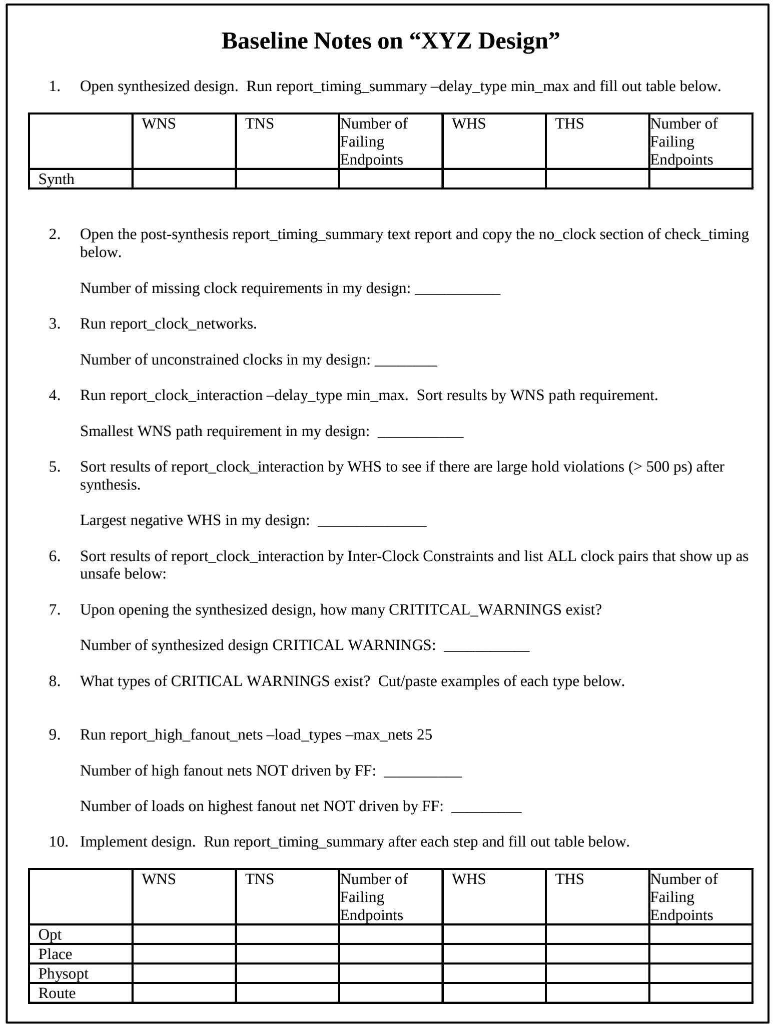
18x7x8 = 1008

Directive Sweeping with Vivado

- **Goal: choose the optimal directive for each implementation step**
- **“Directive Sweeping”**
  - Tcl script opens command shells, creates directories and launches tool for each implementation step
  - Each implementation step uses a unique implementation directive, while keeping the rest of the design constant.
- **After each implementation step, compare timing results of the attempts and choose the best candidate(s) to carry forward to next implementation step**
  - Look for results that are head-and-shoulders above the rest

Directive Sweeping with Vivado

- Baseline the design first to ensure timing constraints are reasonable
- Baseline the design first to ensure timing constraints are reasonable (Yes – this is here on purpose)
- Start point is a linked design or optimized design checkpoint with full timing constraints and no floorplan
- For effective time use, launch all attempts for each implementation step in parallel
  – Requires heavy compute resources



Baseline Notes on "XYZ Design"

1. Open synthesized design. Run report_timing_summary –delay_type min_max and fill out table below.

	WNS	TNS	Number of Failing Endpoints	WHS	THS	Number of Failing Endpoints
Synth						

2. Open the post-synthesis report_timing_summary text report and copy the no_clock section of check_timing below.

   Number of missing clock requirements in my design: ___________

3. Run report_clock_networks.

   Number of unconstrained clocks in my design: ________

4. Run report_clock_interaction –delay_type min_max. Sort results by WNS path requirement.

   Smallest WNS path requirement in my design: ___________

5. Sort results of report_clock_interaction by WHS to see if there are large hold violations (> 500 ps) after synthesis.

   Largest negative WHS in my design: ______________

6. Sort results of report_clock_interaction by Inter-Clock Constraints and list ALL clock pairs that show up as unsafe below:

7. Upon opening the synthesized design, how many CRITITCAL_WARNINGS exist?

   Number of synthesized design CRITICAL WARNINGS: ___________

8. What types of CRITICAL WARNINGS exist? Cut/paste examples of each type below.

9. Run report_high_fanout_nets –load_types –max_nets 25

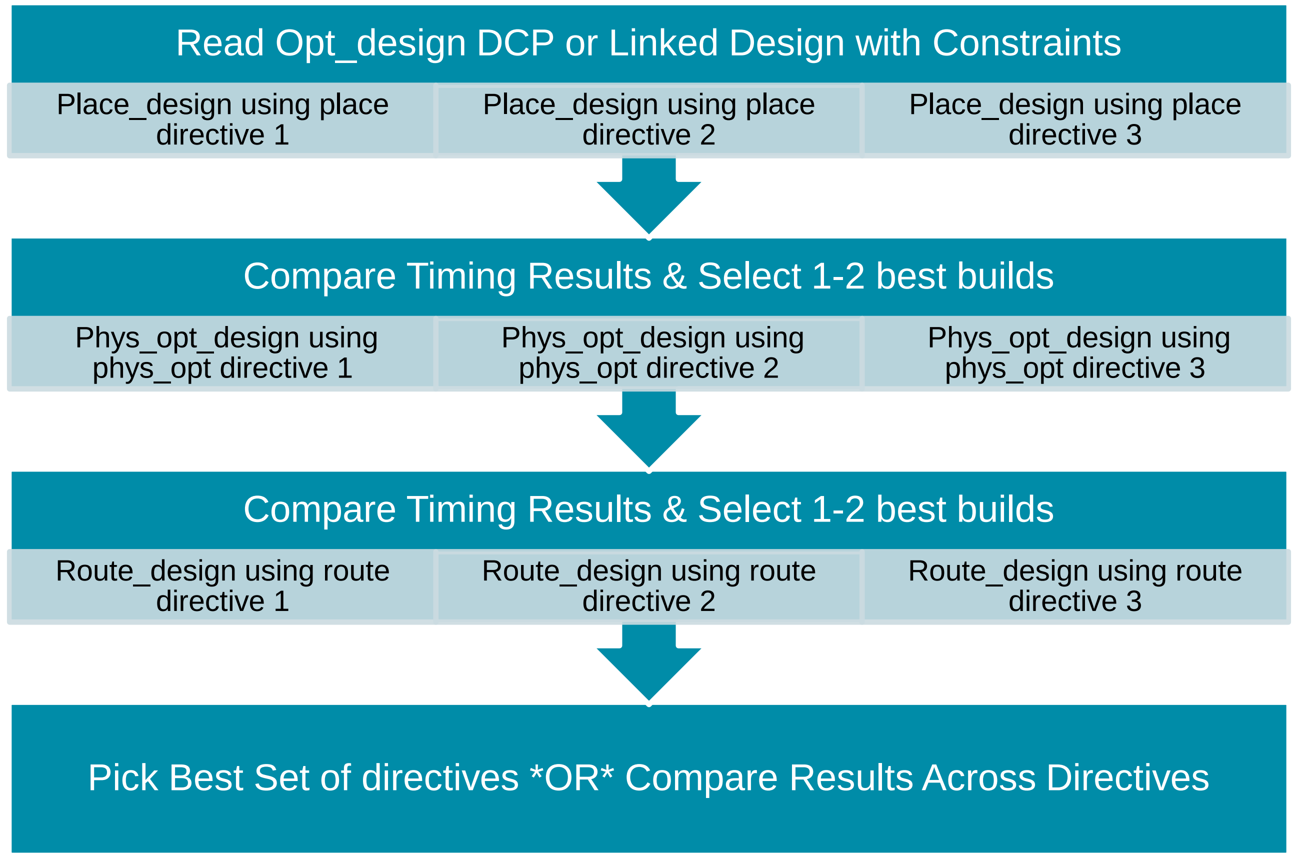
   Number of high fanout nets NOT driven by FF: __________

   Number of loads on highest fanout net NOT driven by FF: _________

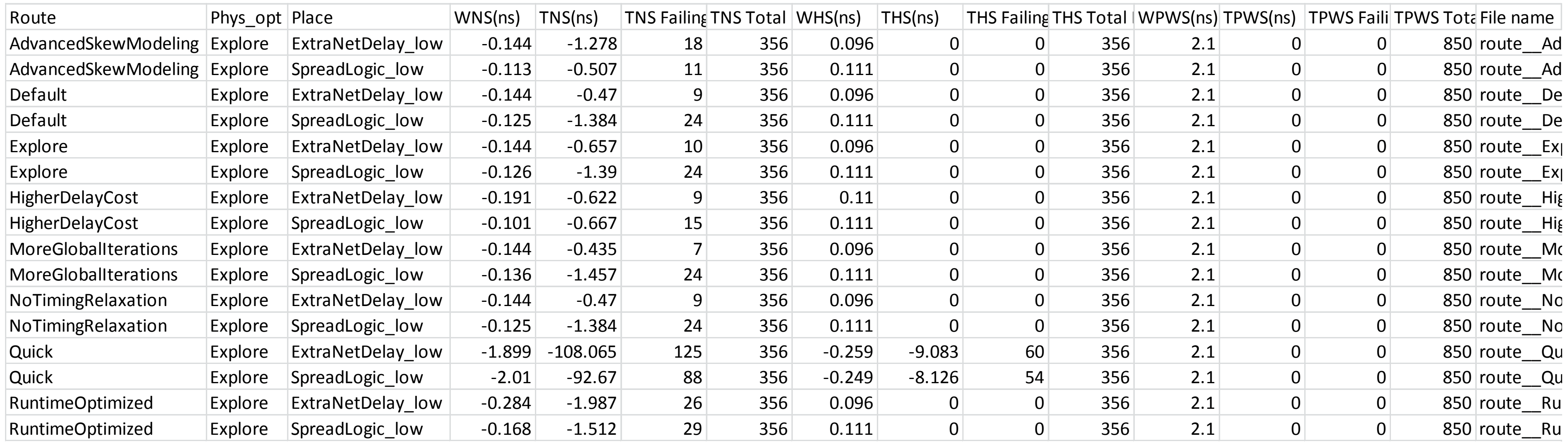
10. Implement design. Run report_timing_summary after each step and fill out table below.

	WNS	TNS	Number of Failing Endpoints	WHS	THS	Number of Failing Endpoints
Opt						
Place						
Physopt						
Route						

Directive Sweeping Flowchart


Read Opt_design DCP or Linked Design with Constraints
Place_design using place directive 1
Place_design using place directive 2
Place_design using place directive 3
Compare Timing Results & Select 1-2 best builds
Phys_opt_design using phys_opt directive 1
Phys_opt_design using phys_opt directive 2
Phys_opt_design using phys_opt directive 3
Compare Timing Results & Select 1-2 best builds
Route_design using route directive 1
Route_design using route directive 2
Route_design using route directive 3
Pick Best Set of directives *OR* Compare Results Across Directives

DesignTimingSummaries.csv



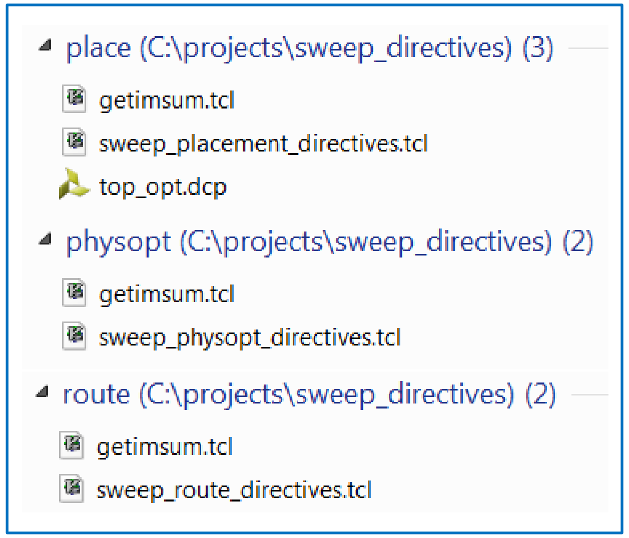
Route	Phys_opt	Place	WNS(ns)	TNS(ns)	TNS Failing	TNS Total	WHS(ns)	THS(ns)	THS Failing	THS Total	WPWS(ns)	TPWS(ns)	TPWS Faili	TPWS Tota	File name
AdvancedSkewModeling	Explore	ExtraNetDelay_low	-0.144	-1.278	18	356	0.096	0	0	356	2.1	0	0	850	route__Ad
AdvancedSkewModeling	Explore	SpreadLogic_low	-0.113	-0.507	11	356	0.111	0	0	356	2.1	0	0	850	route__Ad
Default	Explore	ExtraNetDelay_low	-0.144	-0.47	9	356	0.096	0	0	356	2.1	0	0	850	route__De
Default	Explore	SpreadLogic_low	-0.125	-1.384	24	356	0.111	0	0	356	2.1	0	0	850	route__De
Explore	Explore	ExtraNetDelay_low	-0.144	-0.657	10	356	0.096	0	0	356	2.1	0	0	850	route__Ex
Explore	Explore	SpreadLogic_low	-0.126	-1.39	24	356	0.111	0	0	356	2.1	0	0	850	route__Ex
HigherDelayCost	Explore	ExtraNetDelay_low	-0.191	-0.622	9	356	0.11	0	0	356	2.1	0	0	850	route__Hig
HigherDelayCost	Explore	SpreadLogic_low	-0.101	-0.667	15	356	0.111	0	0	356	2.1	0	0	850	route__Hig
MoreGlobalIterations	Explore	ExtraNetDelay_low	-0.144	-0.435	7	356	0.096	0	0	356	2.1	0	0	850	route__Mo
MoreGlobalIterations	Explore	SpreadLogic_low	-0.136	-1.457	24	356	0.111	0	0	356	2.1	0	0	850	route__Mo
NoTimingRelaxation	Explore	ExtraNetDelay_low	-0.144	-0.47	9	356	0.096	0	0	356	2.1	0	0	850	route__No
NoTimingRelaxation	Explore	SpreadLogic_low	-0.125	-1.384	24	356	0.111	0	0	356	2.1	0	0	850	route__No
Quick	Explore	ExtraNetDelay_low	-1.899	-108.065	125	356	-0.259	-9.083	60	356	2.1	0	0	850	route__Qu
Quick	Explore	SpreadLogic_low	-2.01	-92.67	88	356	-0.249	-8.126	54	356	2.1	0	0	850	route__Qu
RuntimeOptimized	Explore	ExtraNetDelay_low	-0.284	-1.987	26	356	0.096	0	0	356	2.1	0	0	850	route__Ru
RuntimeOptimized	Explore	SpreadLogic_low	-0.168	-1.512	29	356	0.111	0	0	356	2.1	0	0	850	route__Ru

Demo

- **Sweep Directives Demo**
  - Very simple demo (7 files)
  - Easy to demo for customers
  - Laptop
  - Linux
  - Linux with remote hosts (using ssh)

sweep_directives.zip

Directory Structure



place (C:\projects\sweep_directives) (3)
- getimsum.tcl
- sweep_placement_directives.tcl
- top_opt.dcp

physopt (C:\projects\sweep_directives) (2)
- getimsum.tcl
- sweep_physopt_directives.tcl

route (C:\projects\sweep_directives) (2)
- getimsum.tcl
- sweep_route_directives.tcl

Directive Sweeping Results for a Real Design

- Design Type: Comms
- Significant IP: 10G XFI x2, RLDRAM I/F
- Part: xc7v2000t-2L
- Utilization:
  - Slice: 56%
  - FF: 23%
  - BRAM: 52%
- Clock Frequencies: 200 MHz core, 300+ GT (multiple), 500 MHz RDLRAM, 100 MHz uP

Directive Sweeping Results for a Real Design

- **Create an optimized checkpoint (opt_design) that does _*not*_ contain a floorplan**
- **Tcl script opens command terminals to implement design**
- **Tcl script creates directories that match directive names**
- **Tcl script runs variation of the script below in each Vivado shell:**
  - set place_directive <directive_name>
  - read_checkpoint <opt_checkpoint>.dcp
  - place_design -directive $place_directive
  - report_timing_summary -file $place_directive.tmg.rpt
  - write_checkpoint $place_directive.dcp
  - #exit

Directive Sweeping Results for a Real Design

- **Created set of composite timing results with tcl script**
- **Hold time is not considered**
  - as long as post-place WHS < ~500ps

Directive Sweeping --- Place_design Results

Place_design Directive	WNS	TNS	Total Err
Explore	-1.750	-1706.488	974
WLDrivenBlockPlacement	-1.951	-1492.123	764
LateBlockPlacement	-1.614	-877.359	548
ExtraNetDelay_high	**-0.836**	**-29.283**	**35**
ExtraNetDelay_medium	-1.099	-48.552	44
ExtraNetDelay_low	-1.341	-268.127	206
SpreadLogic_high	-1.614	-954.618	591
SpreadLogic_medium	-1.614	-954.618	591
SpreadLogic_low	-1.750	-1706.488	974
ExtraPostPlacementOpt	-1.750	-1706.488	974
SSI_ExtraTimingOpt	-1.152	-283.900	246
SSI_SpreadSLLs	-2.107	-1056.771	501
SSI_BalanceSLLs	-2.005	-1108.340	552
SSI_BalanceSLRs	**-0.886**	**-29.709**	**33**
SSI_HighUtilSLRs	-2.450	-2367.354	966

Directive Sweeping – Phys_opt Results (1)

- The SSI_BalanceSLRs and ExtraNetDelay_high had similar results, and were clearly better than the other attempts
- These two DCPs were chosen for Exploration in phys_opt_design

Placement = SSI_BalanceSLRs

Phys_opt_design Directive	WNS	TNS	Total Err
Explore	-0.409	-3.345	8
AggressiveExplore	-0.409	-2.103	5
AlternateReplication	-0.629	-17.727	28
AggressiveFanoutOpt	-0.629	-10.584	16
AlternateDelayModeling	-0.676	-18.165	26
AddRetime	-0.629	-17.231	27

Directive Sweeping – Phys_opt Results (2)

- The ExtraNetDelay_high and SSI_BalanceSLRs had similar results, and were clearly better than the other attempts
- These two DCPs were chosen for Exploration in phys_opt_design

Placement = ExtraNetDelay_high Placement

Phys_opt_design Directive	WNS	TNS	Total Err
Explore	-0.014	-0.025	2
AggressiveExplore	**0.046**	**0.000**	**0**
AlternateReplication	-0.558	-19.343	35
AggressiveFanoutOpt	-0.558	-15.993	28
AlternateDelayModeling	-0.628	-19.971	32
AddRetime	-0.558	-19.343	34

Directive Sweeping -- Route_design Results

- Phys_opt_design directive AggressiveExplore on placed DCP ExtraNetDelay_high provided the best result
- This DCP was selected to run in route_design

Placement = ExtraNetDelay_high Placement, Phys_opt = AggressiveExplore

Route_design Directive	WNS	TNS	Total Err
Explore	**0.000**	**0.000**	**0**
NoTimingRelaxation	N/A	N/A	N/A
MoreGlobalIterations	N/A	N/A	N/A
HigherDelayCost	N/A	N/A	N/A
AdvancedSkewModeling	N/A	N/A	N/A
RuntimeOptimized	N/A	N/A	N/A

✓ Done!

Closing Timing after Directive Sweeping

- Create pblocks by “Reverse Engineering” the floorplan from the final route. See if this improves run time.
- Incremental design flow to retain placement results

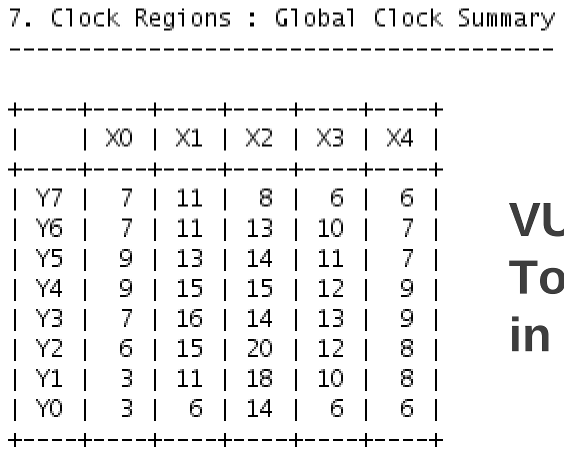
Conclusion

- Directive Sweeping can rapidly unearth the optimal implementation options
- A complex design can achieve timing closure without a floorplan, as was demonstrated here
- A good practice is to apply optimal implementation directives before applying any floorplanning constraints
- Looking at WNS in a vacuum is not enough – need to consider WNS ALONG with TNS and total number of errors
- If total number of errors < 100 at any step, review those errors to see if they can be easily resolved by design change, constraint change or floorplan
- If TNS is better than -30ns, you can proceed to the next step even if the WNS is worse than -300ps

Implementation Analysis and Reporting

- report_clock_utilization overhaul
  - New structure and data for UltraScale and UltraScale+
  - Includes related clock object and constraint info
  - Text-based maps of utilization by clock region

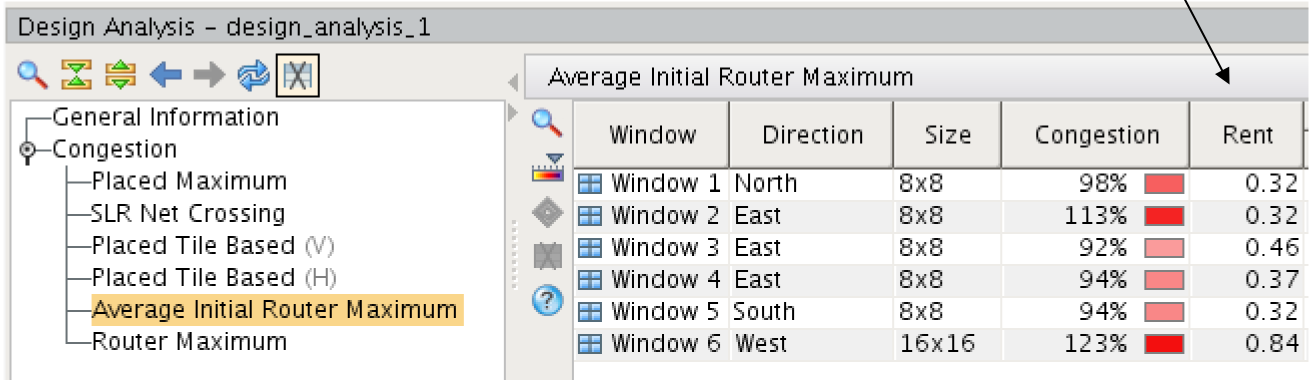
- report_design_analysis improvements
  - Compare estimated without unrouting or loading placement
    - -routed_vs_estimated
  - Get the timing paths from logic levels distribution table:
    - -logic_levels, -end_point_clock, -return_timing_paths
  - Netlist Rent of a placed region
    - -bounding_boxes
  - Average Initial Router congestion
    - Congestion router sees at outset
    - Identifies real problem areas to analyze against the placement



```
7. Clock Regions : Global Clock Summary
---------------------------------------

+----+----+----+----+----+----+
|    | X0 | X1 | X2 | X3 | X4 |
+----+----+----+----+----+----+
| Y7 |  7 | 11 |  8 |  6 |  6 |
| Y6 |  7 | 11 | 13 | 10 |  7 |
| Y5 |  9 | 13 | 14 | 11 |  7 |
| Y4 |  9 | 15 | 15 | 12 |  9 |
| Y3 |  7 | 16 | 14 | 13 |  9 |
| Y2 |  6 | 15 | 20 | 12 |  8 |
| Y1 |  3 | 11 | 18 | 10 |  8 |
| Y0 |  3 |  6 | 14 |  6 |  6 |
+----+----+----+----+----+----+
```

VU095 Example: Total Global Clocks in each Clock Region

Rent of placed regions



Design Analysis - design_analysis_1

- General Information
- Congestion
  - Placed Maximum
  - SLR Net Crossing
  - Placed Tile Based (V)
  - Placed Tile Based (H)
  - Average Initial Router Maximum
  - Router Maximum

Average Initial Router Maximum

Window	Direction	Size	Congestion	Rent
Window 1	North	8x8	98% 🟥	0.32
Window 2	East	8x8	113% 🟥	0.32
Window 3	East	8x8	92% 🟥	0.46
Window 4	East	8x8	94% 🟥	0.37
Window 5	South	8x8	94% 🟥	0.32
Window 6	West	16x16	123% 🟥	0.84