NetFPGA Summer Course

Presented by:

Noa Zilberman

Yury Audzevich

Technion

August 2 – August 6, 2015

http://NetFPGA.org

Section I: General Overview

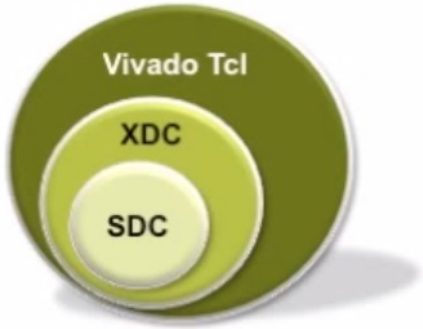
Constraints Methodology

Design constraints define the requirements that must be met by the compilation flow in order for the design to be functional on the board

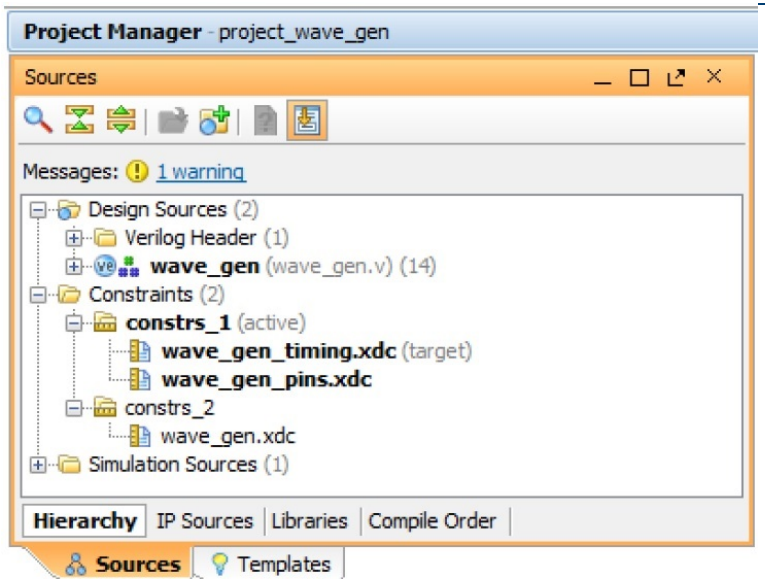
- **Over-constraining and under-constraining is bad, so use reasonable constraints that correspond to your requirements**

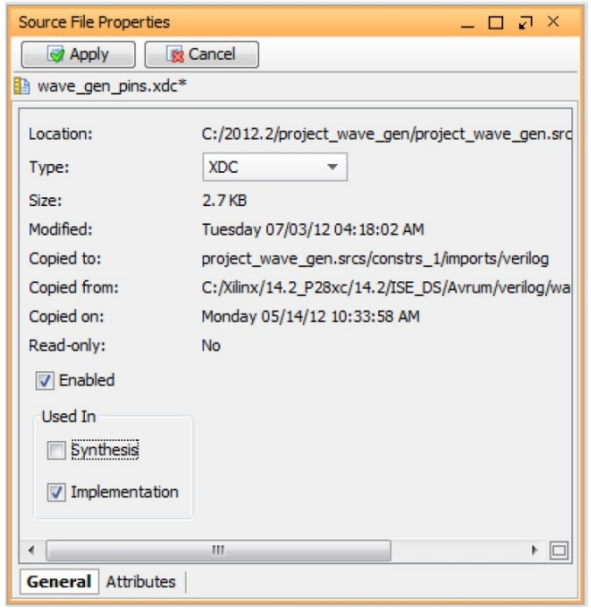
- **Xilinx provides new Xilinx Design Constraint (XDC) file -- quite different from previously used User Constraints File (UCF)**

- **Single or multiple XDC files in a design might serve a different purpose**



Xilinx Design Constraint file





XDC constraints are a combination of:

- **Synopsys Design Constraints format (SDC)**
- **Xilinx centric extensions**
- **Tcl-compatible for advanced scripting**

XDC constraints have the following properties:

- **follow the Tcl semantic,**
- **interpreted like any other Tcl command,**
- **read in and parsed sequentially.**

You can use constraints for:

- **Synthesis and/or Implementation**

Options are specified in file properties or via tcl :

set_property used_in_synthesis false [get_files wave_gen_pins.xdc]

set_property used_in_implementation true [get_files wave_gen_pins.xdc]

XDC File Order

The constraint files are loaded in **the same sequence as the way they are listed**

To change order either drag and drop or reorder using:

```
reorder_files -fileset constrs_1 -before [get_files
wave_gen_timing.xdc] \
[get_files wave_gen_pins.xdc]
```

IPs:

If you use the native IPs, their XDC files are loaded after your files

You cannot change the IP XDC files order, but you can disable them and re-apply constraints in your XDC files

Common pitfalls

Missing constraints:

- The corresponding paths are not optimized for timing
- No violation will be reported but design may not work on HW

Incorrect constraints:

- Runtime and optimization efforts will be spent on the wrong paths
- Reported timing violations may not result in any issues on HW

Unreasonable hold requirements:

- 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

Key to creating XDC constraints

Organize your constraints in the following sequence:

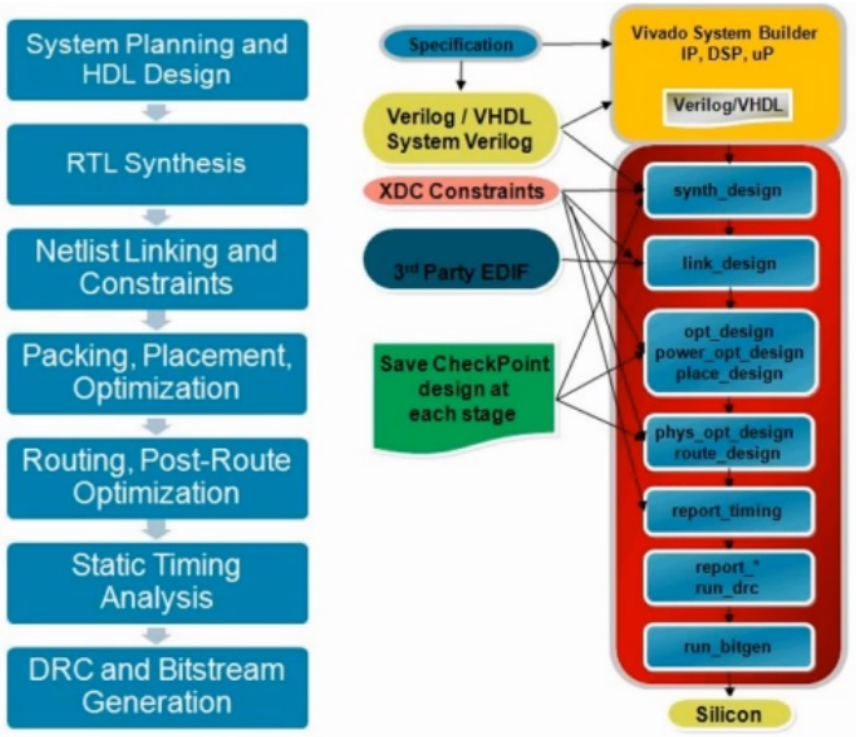
1)
```
## Timing Assertions Section
# Primary clocks
# Virtual clocks
# Generated clocks
# Clock Groups
# Input and output delay constraints
```

2)
```
## Timing Exceptions Section
# False Paths
# Max Delay / Min Delay
# Multicycle Paths
# Case Analysis
# Disable Timing
```

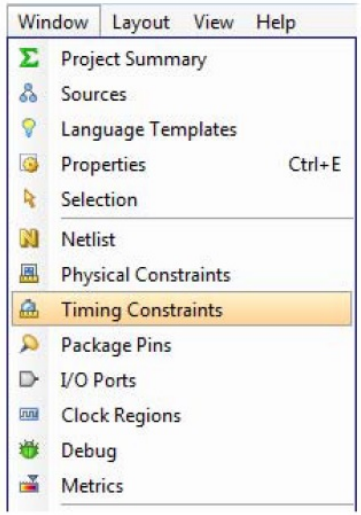
3)
```
## Physical Constraints Section
```

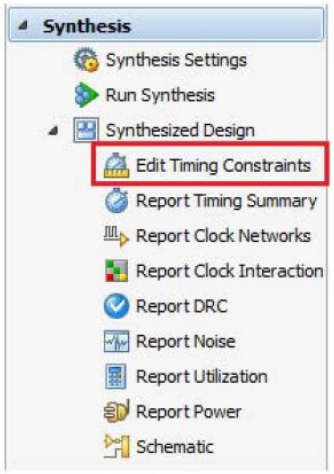
VALIDATE constraints at each step:

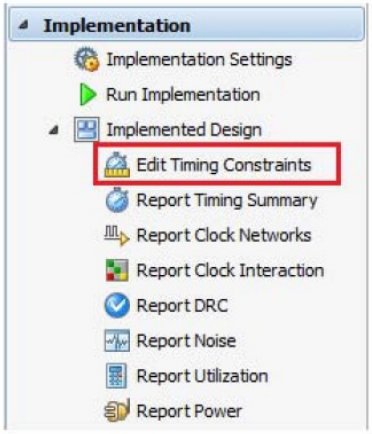
- Constrain it, check reports
- Validate timing



Constraining the design







Constraints include: **Timing constraints**, Pin assignments, Placement constraints (floorplanning), Properties and Attributes.

Syntax of commonly used XDC commands can be checked through:

- Help pages in tcl command line
- XDC Templates (accessed through UI)

Start with an Elaborated Design:

fix timing at early stages -- debug and optimize your RTL

Synthesis Constraints

Vivado IDE synthesis engine transforms the **RTL description into technology mapped netlist**

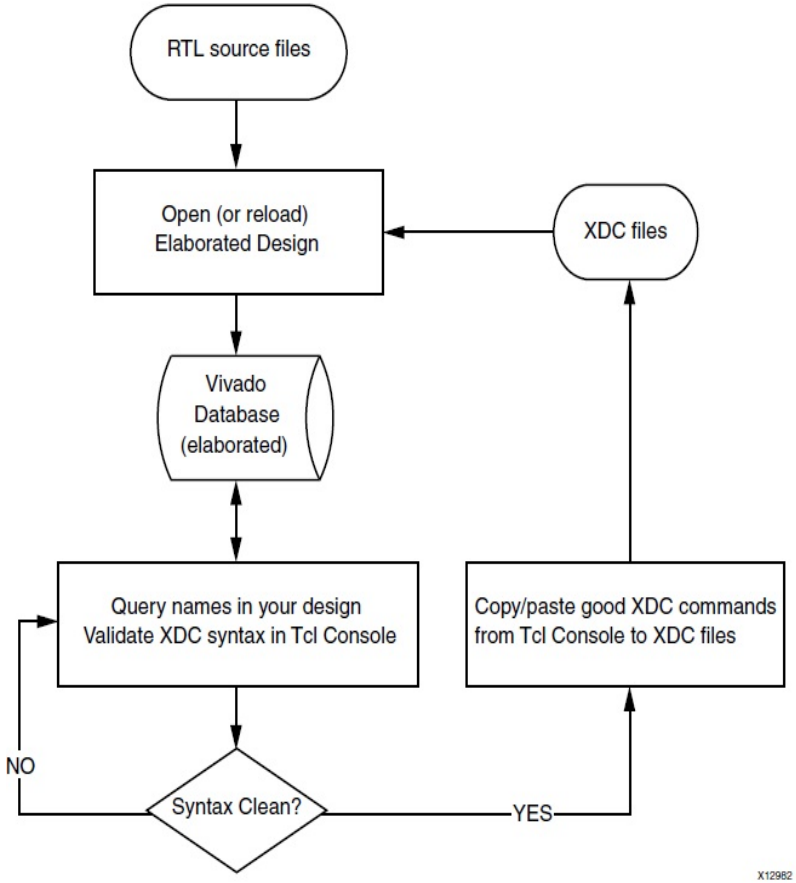
With synth design net delay modelling is not very accurate; synth netlist should either meet timing or fail by a small amount before starting implementation.

There are three categories of constraints for synthesis:

- **RTL Attributes**
  - directives written in the RTL files (MARK_DEBUG, etc.)

- **Timing Constraints (XDC)**
  - the following have real impact on synthesis
    - create_clock
    - create_generated_clock
    - set_input_delay
    - set_output_delay
    - set_clock_groups
    - set_false_path
    - set_max_delay
    - set_multicycle_path

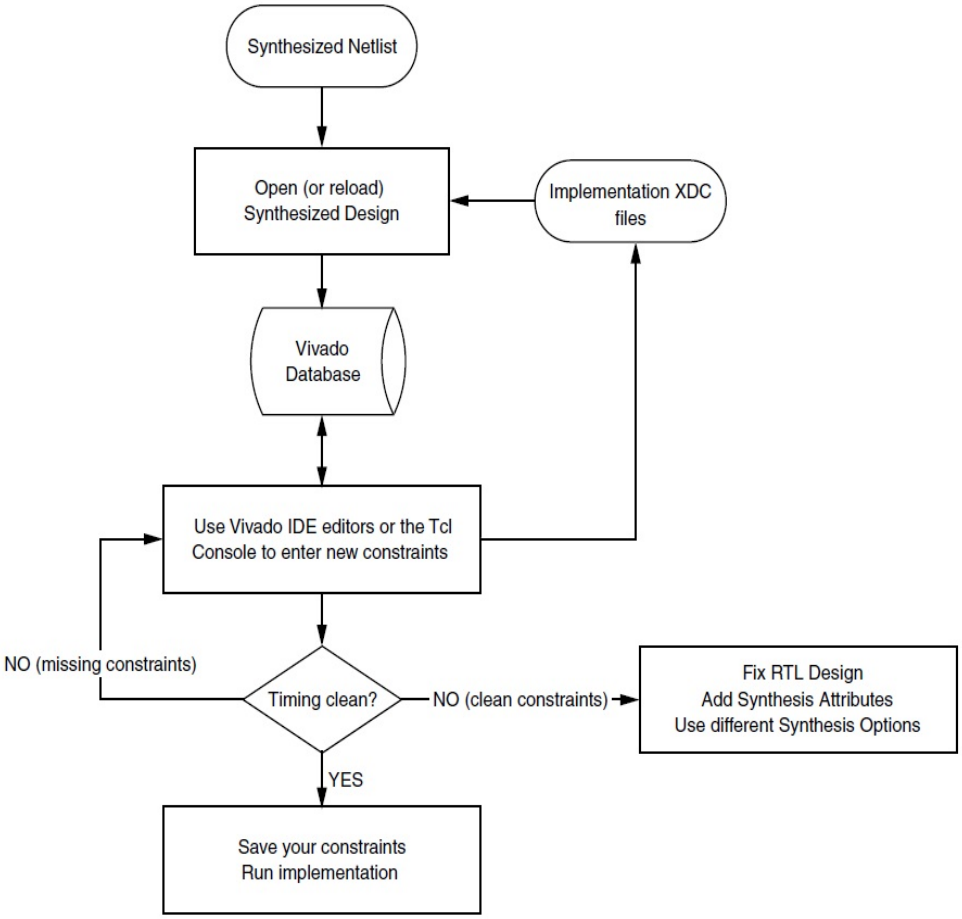
- **Physical and Configuration Constraints**
  - ignored by synthesis algorithms



Implementation Constraints

Synthesized netlist allows running timing analysis:

- Correct the timing constraints and save them to an implementation-only XDC file.
- Add missing constraints, such as asynchronous and exclusive clock groups.
- Add timing exceptions, such as multicycle paths and max delay constraints.
- Identify large violations due to long paths in the design and correct the RTL description.



Section II: Static Timing Analysis

Static Timing Analysis (STA)

A design netlist is an interconnected set of ports, cells and nets

- **The functionality of a design is determined by RTL code (verilog, vhdl, etc.) and *verified by simulation tools***
- **The *quality of your RTL* determines how easy timing will be met**
- **The performance of a design is determined by the delays of cells that compromise the design (STA)**
- ***Static timing analysis* doesn't check the functionality of the components but rather *performance* of components**

STA Goals

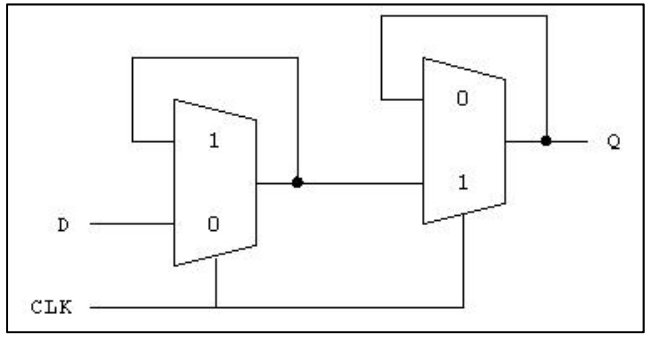
Many FPGA processes are timing driven:

- Synthesis for circuit construction
- Placer for optimal cells locations
- Router for choosing routing elements

Constraints are used to determine the desired performance goals

STA reports whether the design will provide the desired performance through reports

- **Have you heard of Setup/Hold requirements for a single FF?**



… not quite the same as *Setup and Hold path delays* that STA is using

Component delays

Each component has delays to perform it function:

- LUT has propagation delay from it's ins to outs
- Net has delay from driver to receiver
- FF required stable data for a certain time around sampling point

Delays are also dependent of environment factors. These are determined and characterized by Xilinx during device design.

Timing is extracted over the operating range of the device:

- Process (different speed grades)
- Voltage (min $\rightarrow$ max)
- Temperature (min $\rightarrow$ max)

Range delays are extracted at various process corners (STA):

- Slow process corner: slow process, lowest voltage, highest temperature
- Fast process corner: fastest process, highest voltage, lowest temperature

Static Timing Path

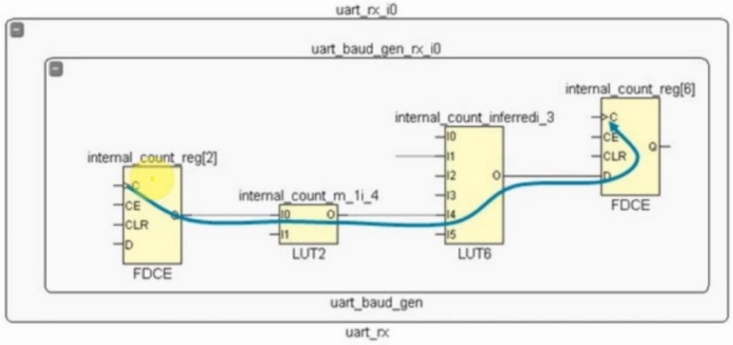
- ***A static timing path*** is a path that starts at a clock element
- Propagates through any # combinatorial elements and nets
- Ends at clocking element

Vivado's synthesis, place and route tool does STA of all paths both fast and slow corners

Source clock delay – starting top level clock port and ending at the launch FF

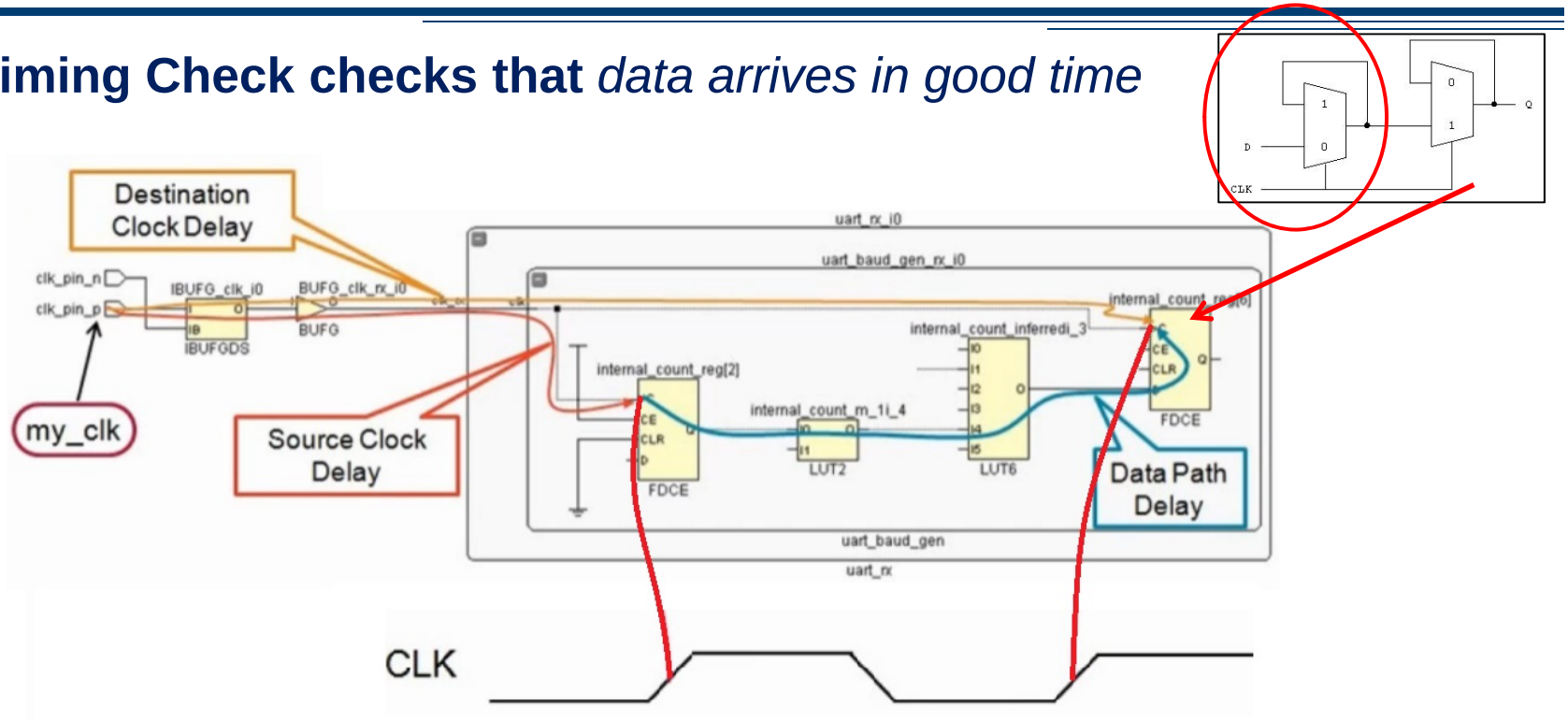
Data path delay – delay to the capturing FF

Destination clock delay – there might be a difference bw these two FFs



Setup check

Setup Timing Check checks that *data arrives in good time*



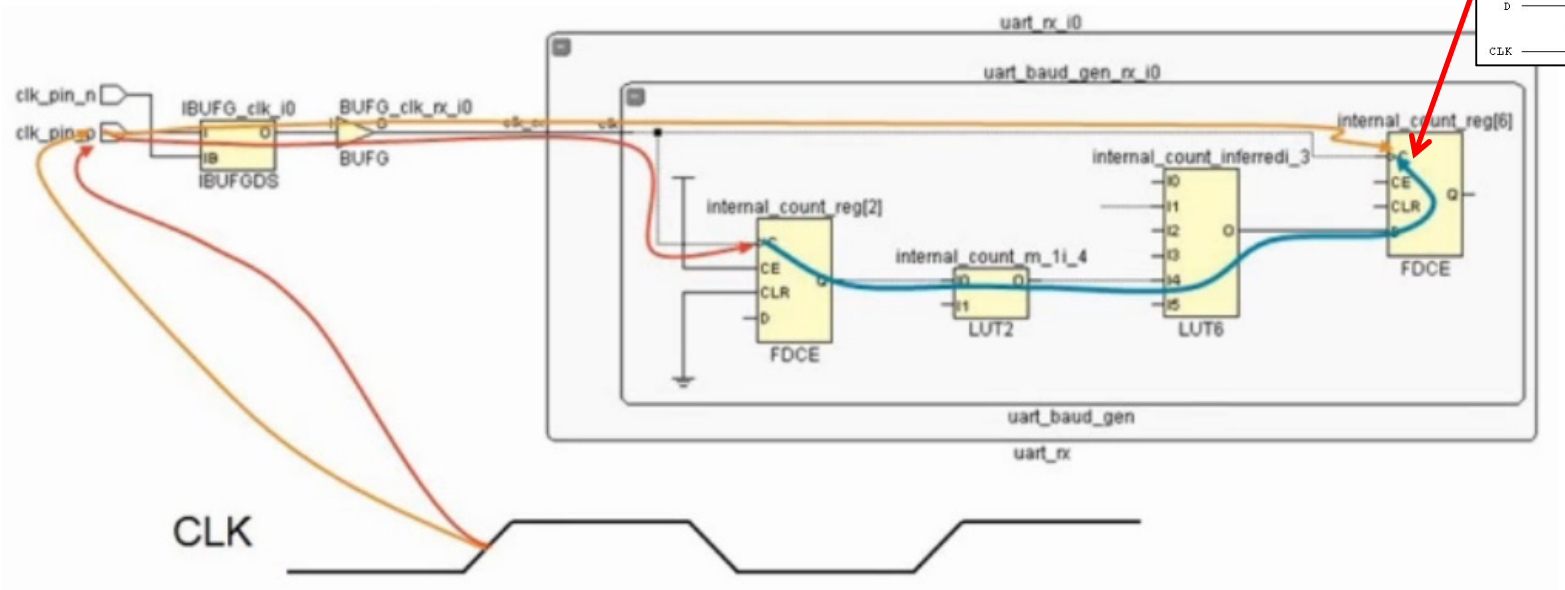
Checks that change in a clocked element has time to propagate to other clocked elements before the next clock event

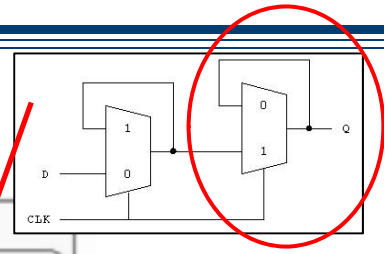
Simple case – same domain & only data path is considered:

$$T(D1\_CLK) + T(FF1_{(Clk->Q)}) + T(Comb) < T(CLK_{period}) - T(FF2_{(setup)}) - T(SU) + T(D2\_CLK)$$

Hold check

Hold time checks that *data doesn't arrive too quickly*





Checks DATA isn't caught at destination FF at the same clock as the clock that launched it at launch FF

Simple case – same domain & only data path is considered:

$$T(D1\_CLK) + T(FF1_{(Clk->Q)}) + T(Comb) > T(FF2_{(hold)}) + T(D2\_CLK) + T(HU)$$

Section III: Timing constraints in Vivado

Method to create good constraints

Create clocks and define clock interactions:

- 4 step rule

Setup Input and Output delays

- Try not creating wrong HOLD violations

Set timing exceptions

- Less is more – let Vivado do magic for you
- Try not creating wrong HOLD violations

Use report commands to validate each step

Clocks in the design

CLKs are periodic signals with:

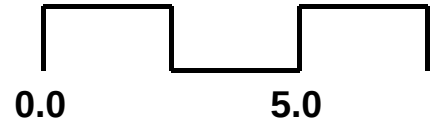
- **1) period – time from rising edge to the next rising edge**
- **2) Duty cycle – high to low ratio of the clock**
- **3) Jitter – variation of period from nominal**
- **4) Phase – position of the rising edge**

Clocks are created with create_clock Tcl command:

- **create_clock –name <name> -period <period> <objects>**
- **<objects> are the list of pins, ports, or nets to which attach the clock,**

Example:

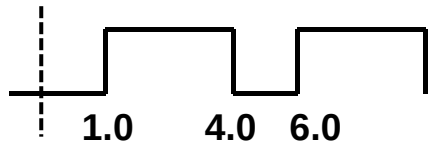
create_clock –name sys_clk –period 5.0 [get_ports clk_in]



Clocks with phase offsets and different duty cycles can be created using "waveform" option:

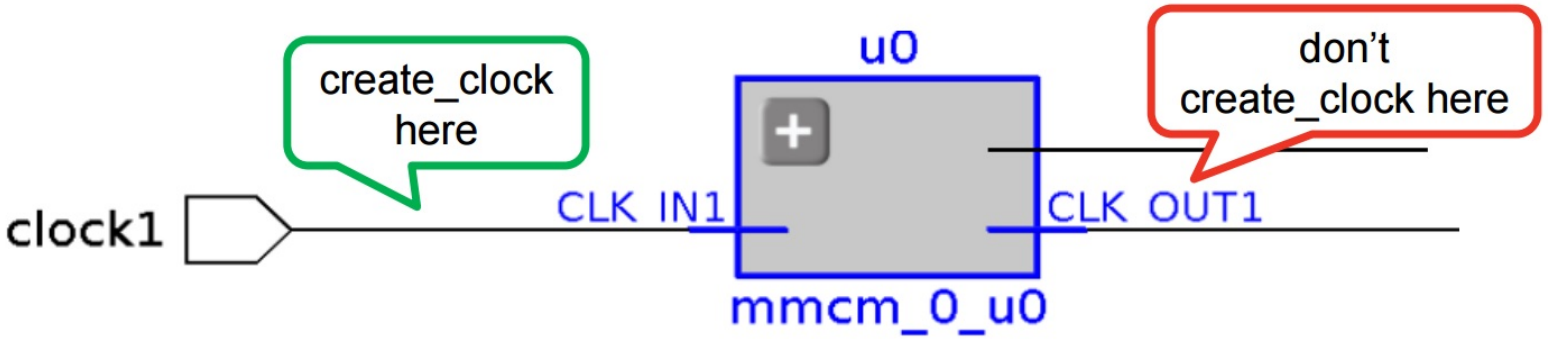
- waveform <edges> - list of numbers representing times of successive edges

**create_clock –name sys_clk1 –period 5.0 –waveform {1.0 4.0} **
[get_ports clk_in1]



Clock rules

- Clock only exist when you create them
- Clocks propagate automatically through clocking modules
  - MMCM/PLL/BUFR clock clocks are automatically generated
  - Transceiver clocks are not supported – create them manually



- Use create_generated_clocks for internal clocks (if needed)
  - Note that timing analysis will be performed using originating primary clock
- ALL inter-clock path are evaluated by default

4 Steps for creating clocks

BEWARE: In Vivado all clocks are related unless you specifically say that they are not!

- **Step 1**
  - Use create_clock for all primary clocks on top level ports
  - Run the synthesis or open netlist design
- **Step 2**
  - Run report_clocks
  - Study the report to verify period, phase and propagation
  - Apply corrections to your constraints if needed

```
Attributes
  P: Propagated
  G: Generated

Clock           Period  Waveform        Attributes  Sources
sys_clk         10.000  {0.000 5.000}   P           {sys_clk}
pll0/clkfbout   10.000  {0.000 5.000}   P,G         {pll0/plle2_adv_inst/CLKFBOUT}
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}
```

Output of report_clocks

4 Steps for creating clocks (cont.)

- **Step 3**

  - **Evaluate the clock interaction using report_clock_interaction**

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

  - **Unconstraint inter-clock paths (Clock Domain Crossing) as needed:**
    - **Make sure you designed proper CDC synchronizers**
    - **Use set_clock_groups (preferred method to set_false_path)**
    - **use report_cdc command in Vivado 2015**

  - **Do you have unconstrained objects?**
    - **Find out with check_timing**

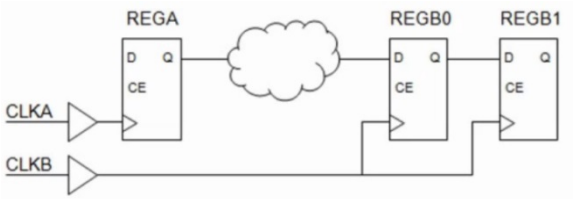
- **Step 4**

  - **Run report_clock_networks**
  - **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**

Constraining clock crossing domains

Use appropriate synchronizing techniques

- 2 or more register synchronizers, for single bit
- Asynchronous FIFOs for buses



Maximize Mean Time Between Failures (MTBF)

- Use ASYNC_REG to place synchronizing flops in the same slice

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

Set the tool to ignore timing paths between individual clocks

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

This is equivalent to:

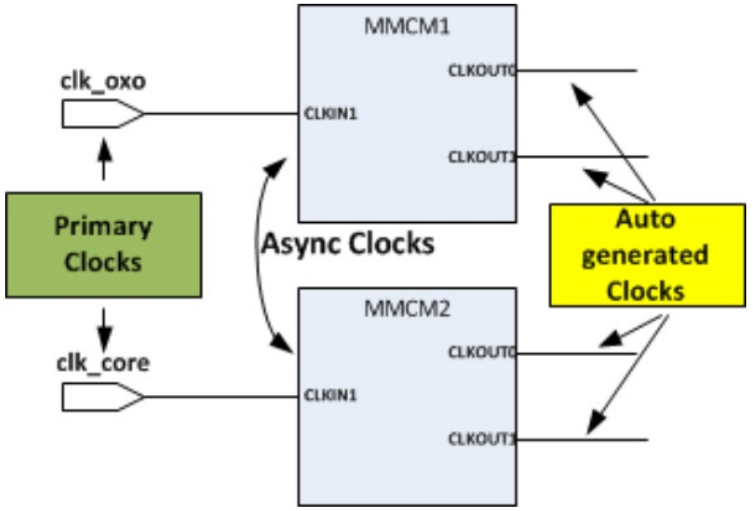
```
set_false_path –from [get_clocks clk1] –to [get_clocks clk2]
set_false_path –from [get_clocks clk2] –to [get_clocks clk1]
```

Asynchronous CDC

Ignoring timing paths between groups of clocks

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} ]
```

Setting Input/Output delay

Constraints should be developed in the following order:

1) **Baseline constraints – Optimize Internal Paths first**
2) **Add I/O constraints – Optimize entire chip**
3) **Add timing exceptions and Floorplan – Fine-tuning step**

set_input_delay (check options):

a) Data propagation from external chip to input package pin of FPGA device, and b) Relative reference board clock

set_output_delay (setup requirement of external source):

a) Data propagating from the output package pin of FPGA device through the board to another device and, b) relative ref. board clock

- **Use set_input_delay and set_output_delay for realistic delays**
- **Wrong delay value (e.g. 0 ns) can cause wrong HOLD violations**

Timing exceptions

- **are needed when the logic behaves in a way that is not timed correctly by default:**

- **set_multicycle_path** - # clock cycles required to propagate data from the start to the end of a path.
- **set_false_path** - logic path in the design that should not be analysed.
- **set_max_delay, set_min_delay** - overrides the default setup and hold constraints with user specified max & min delays.
- **set_case_analysis -** restricts certain signals being propagated through the design.

Timing report

```
report_timing
INFO: [Timing 38-91] UpdateTimingParams: Speed grade: -3, Delay Type: max.
INFO: [Timing 38-191] Multithreading enabled for timing update using a maximum of 8 CPUs
INFO: [Timing 38-78] ReportTimingParams: -max_paths 1 -nworst 1 -delay_type max -sort_by slack.
Copyright 1986-2014 Xilinx, Inc. All Rights Reserved.
------------------------------------------------------------------------------------
| Tool Version      : Vivado v.2014.4 (lin64) Build 1071353 Tue Nov 18 16:47:07 MST 2014
| Date              : Tue Jul 28 20:19:21 2015
| Host              : godzilla running 64-bit Ubuntu 14.04.2 LTS
| Command           : report_timing
| Design            : reference_nic_wrapper
| Device            : 7vx690t-ffg1761
| Speed File        : -3  PRODUCTION 1.11 2014-09-11
| Temperature Grade : C
------------------------------------------------------------------------------------

Timing Report

Slack (MET) :             0.317ns  (required time - arrival time)
  Source:                 reference_nic_i/nf_sume_dma/nf_riffa_dma_0/inst/riffa/riffa_inst/rRst_reg/C
                            (rising edge-triggered cell FDRE clocked by userclk1  {rise@0.000ns fall@2.000ns period=4.000ns})
  Destination:            reference_nic_i/nf_sume_dma/nf_riffa_dma_0/inst/riffa_axis_attachment/riffa_to_axis_conv/tuser_reg[75]/R
                            (rising edge-triggered cell FDRE clocked by userclk1  {rise@0.000ns fall@2.000ns period=4.000ns})
  Path Group:             userclk1
  Path Type:              Setup (Max at Slow Process Corner)
  Requirement:            4.000ns  (userclk1 rise@4.000ns - userclk1 rise@0.000ns)
  Data Path Delay:        3.141ns  (logic 0.232ns (7.387%)  route 2.909ns (92.613%))
  Logic Levels:           0
  Clock Path Skew:        -0.107ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    3.197ns = ( 7.197 - 4.000 )
    Source Clock Delay      (SCD):    3.465ns
    Clock Pessimism Removal (CPR):    0.161ns
  Clock Uncertainty:      0.065ns  ((TSJ^2 + DJ^2)^1/2) / 2 + PE
    Total System Jitter     (TSJ):    0.071ns
    Discrete Jitter          (DJ):    0.108ns
    Phase Error              (PE):    0.000ns

    Location             Delay type                Incr(ns)  Path(ns)    Netlist Resource(s)
  -------------------------------------------------------------------    -------------------
                       (clock userclk1 rise edge)
                                                      0.000     0.000 r
    GTHE2_CHANNEL_X1Y23  GTHE2_CHANNEL                0.000     0.000 r  reference_nic_i/nf_sume_dma/pcie3_7x_1/inst/gt_top_i/pipe_wrapper_i/pipe_lane[0].gt_wrapper_i/gth_channel.gthe2_channel_i/TXOUTCLK
                       net (fo=1, routed)             0.975     0.975    reference_nic_i/nf_sume_dma/pcie3_7x_1/inst/gt_top_i/pipe_wrapper_i/pipe_clock_int.pipe_clock_i/pipe_txoutclk_out
    MMCME2_ADV_X1Y5      MMCME2_ADV (Prop_mmcme2_adv_CLKIN1_CLKOUT2)
                                                      0.069     1.044 r  reference_nic_i/nf_sume_dma/pcie3_7x_1/inst/gt_top_i/pipe_wrapper_i/pipe_clock_int.pipe_clock_i/mmcm_i/CLKOUT2
                       net (fo=1, routed)             1.146     2.190    reference_nic_i/nf_sume_dma/pcie3_7x_1/inst/gt_top_i/pipe_wrapper_i/pipe_clock_int.pipe_clock_i/userclk1
```

- **Report Summary**

  Contains info about design, device, tool version, data and time of report

- **Path summary**

  Summarizes timing information for the path: timing is met (Slack), source and destination, clock used, setup and hold check (requirements), number of level of logic, skew and uncertainty

Timing report (cont.)

- **Source clock delay**

  Delays of clock network: edge of the SRC clock, through clock network, until clk pin of launch FF

- **Data path delay**

  Delay: clock pin of launch FF, plus combinational delay until D input of the capturing FF

The above 2 are accumulated for slack calculation

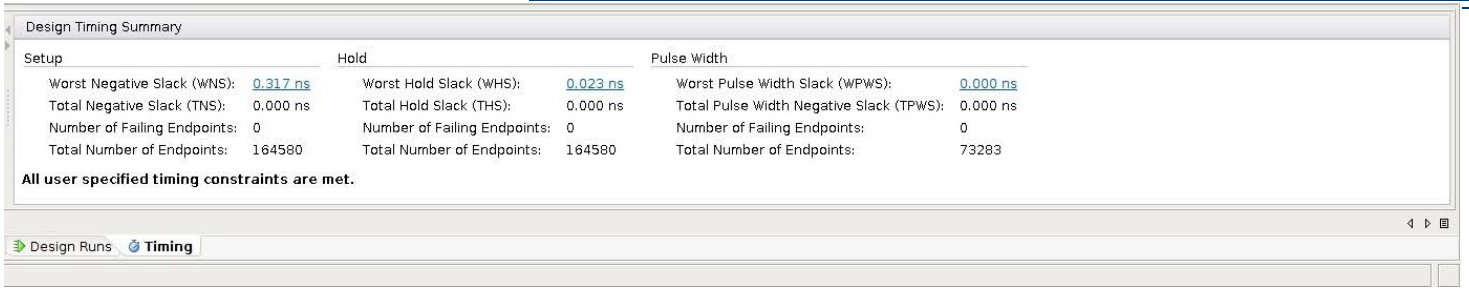
- **Destination Clock delay**

  Propagation from destination clk to the clk pin of destination clocked element

- **Slack calculation**

  Subtracts the arrival time (end of Data Path section) from the required time (end of Destination Clock section)

Timing command summary



- **Create and validate clocks:**
  - **check_timing: for missing clocks and IO constraints**
  - **report_clocks: check frequency and phase**
  - **report_clock_networks: possible clock root**
- **Validate clock groups:**
  - **report_clock_interaction**
- **Validate I/O delays**
  - **report_timing –from [input_port] –setup/-hold**
  - **report_timing –to [output_port] –setup/-hold**
- **Add exceptions if necessary**
  - **Validate using report_timing**

Section III: Integrated Logic Analyzer

Debugging the design

- **RTL-level design simulation**
  - ✓ **Visibility of the entire design; ability to quickly iterate through debug cycle**
  - x **Difficulty of simulating larger designs in a reasonable amount of time**
- **Post-implemented design simulation**
  - ✓ **Debugging the post-implemented timing-accurate model for the design**
  - x **Long run-times and system model accuracy**
- **In-system debugging**
  - ✓ **Debugging of post-implemented design on an FPGA device**
  - ✓ **Debugging actual system environment at system speeds**
  - x **Lower visibility of debug signals**
  - x **Longer design/implementation/debug iterations & hard close timing**

Integrated Logic Analyzer

- **1. Probing phase: Identifying what signals in your design you want to probe and how you want to probe them**
  - **Identifying what signals or nets you want to probe**
  - **Deciding how you want to add debug cores to your design**
- **2. Implementation phase: Implementing the design that includes the additional debug IP that is attached to the probed nets**
  - **The debug core hub must be implemented prior to running the PL & RT.**
- **3. Analysis phase: Interacting with the debug IP contained in the design to debug and verify functional issues**
  - **Connecting to the Hardware Target and Programming the FPGA Device**
  - **Setting up the ILA Core to Take a Measurement**
  - **Viewing ILA Cores in the Debug Probes Window**
  - **Using Basic Trigger Mode**
  - **Viewing ILA Probe Data in the Waveform Viewer**

Inserting ILA cores

- **Either *Manually* add the debug IP component instances through the source code, or**

- **Allow Vivado tool to *automatically insert* the debug cores into your post-synthesis netlist**

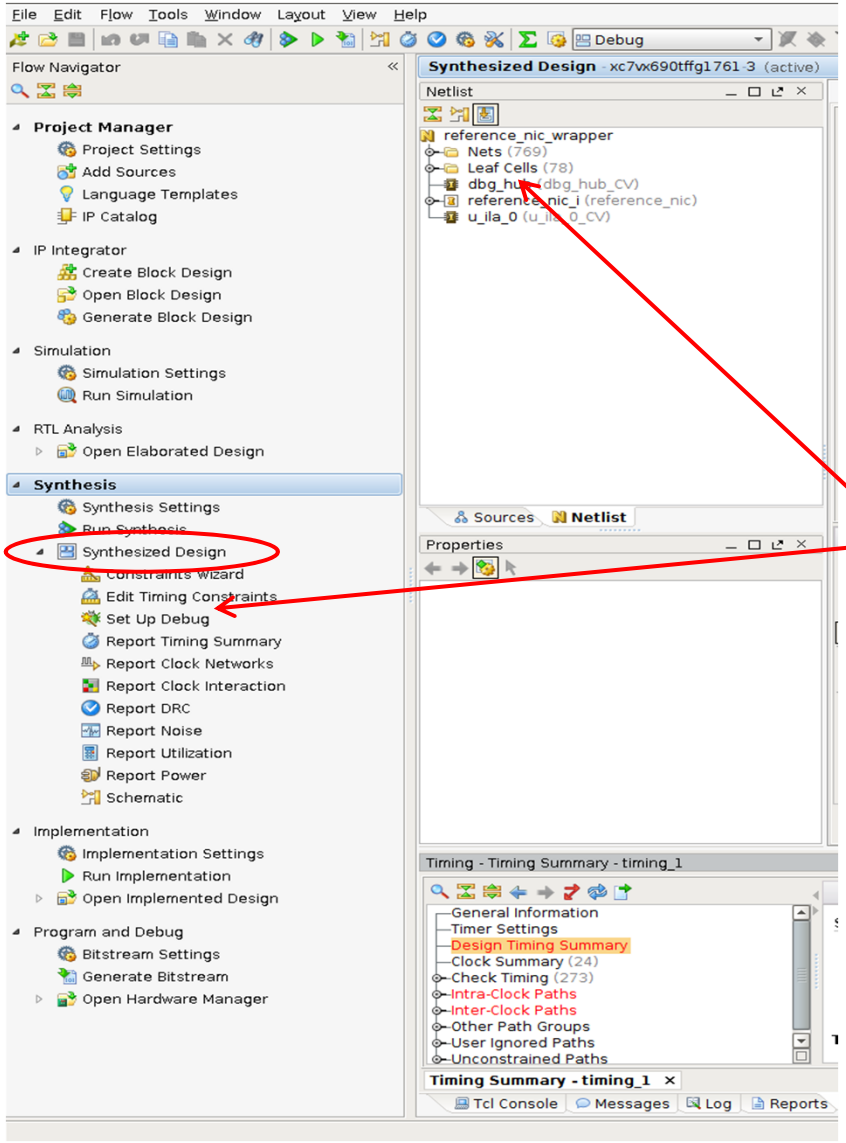
The first approach is more straight forward:

- **Start with Identifying signals for debugging at the HDL source level prior to synthesis**

(* mark_debug = "true" *) wire [7:0] char_fifo_dout; -- Verilog example

- **Once design is synthesized use Set up Debug wizard for core assignment and configuration**

Inserting ILA cores (cont.)

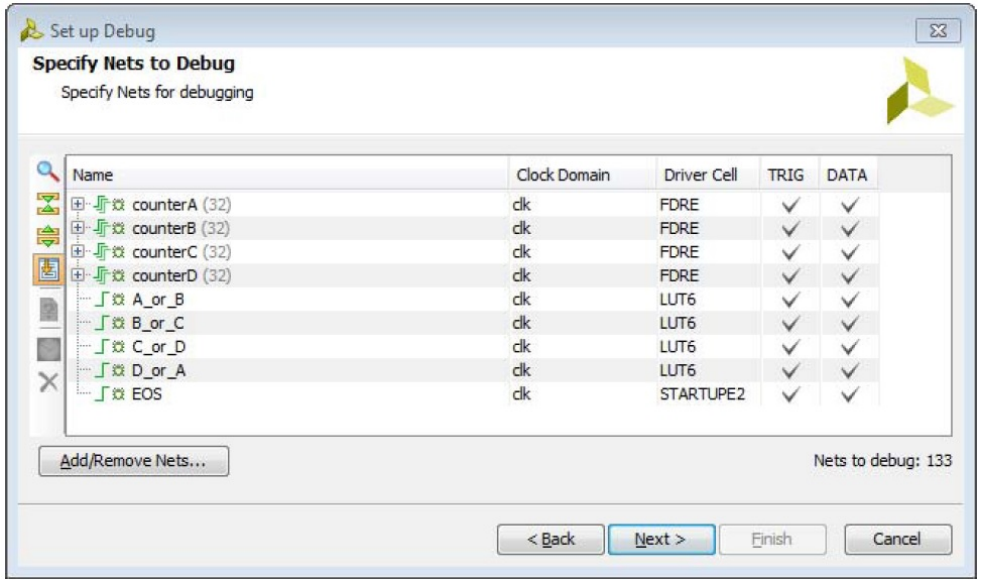


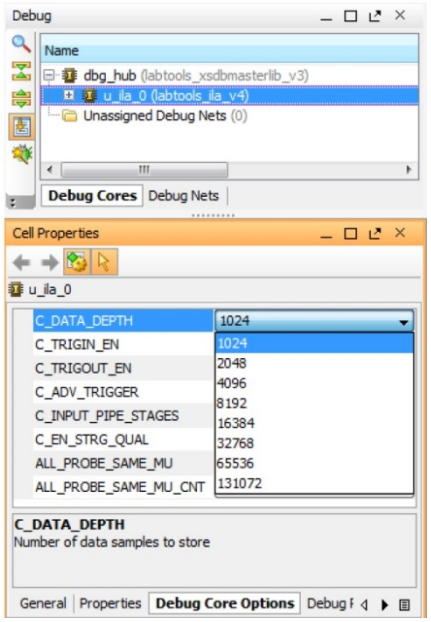
You can insert it from GUI as well:

- **Synthesize your design first**
- **Open synthesized design**
- **Set up debug**
- **The core can be seen in the Netlist folder**

Inserting Debug Cores

Open synthesized design and Insert Debug cores from the list of Unassigned nets.





The **Set up Debug wizard** automatically selects clock domains

The properties of each **core can be customized** using GUI or manually

The appropriate code will be **inserted automatically** into XDC file

Inserting Debug Cores (cont.)

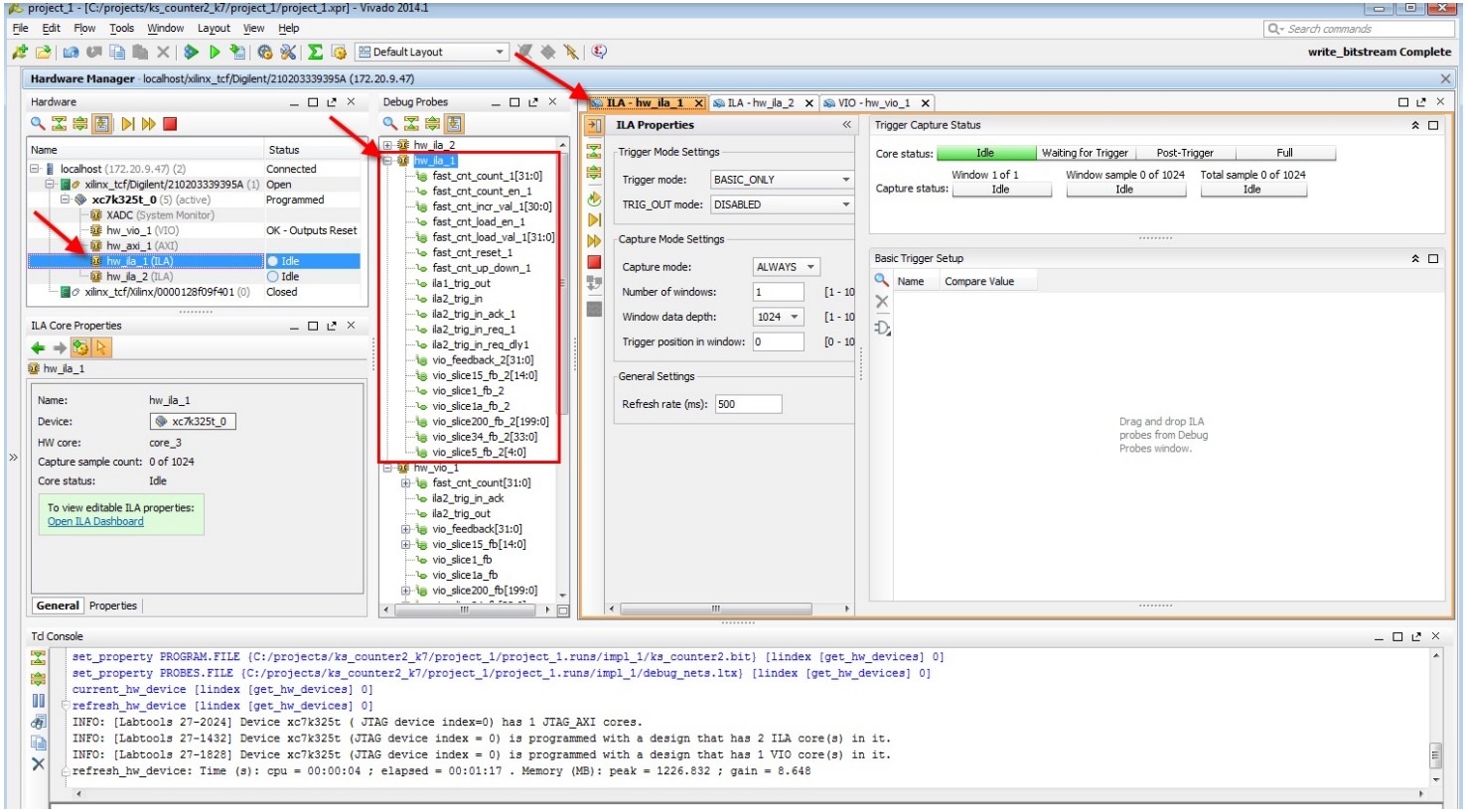
- **XDC Commands can be also used to Insert Debug Cores**

```
create_debug_core u_ila_0 ila
set_property C_DATA_DEPTH 1024 [get_debug_cores u_ila_0]
set_property C_TRIGIN_EN false [get_debug_cores u_ila_0]
set_property C_TRIGOUT_EN false [get_debug_cores u_ila_0]
set_property C_ADV_TRIGGER false [get_debug_cores u_ila_0]
…
```

- **Saving constraints may cause the synthesis and implementation to go out-of-date;**
- **you do not need to re-synthesize the design since the debug XDC constraints are only used during implementation**
- **Check Xil UG908 for advanced debugging capabilities and IBERT**

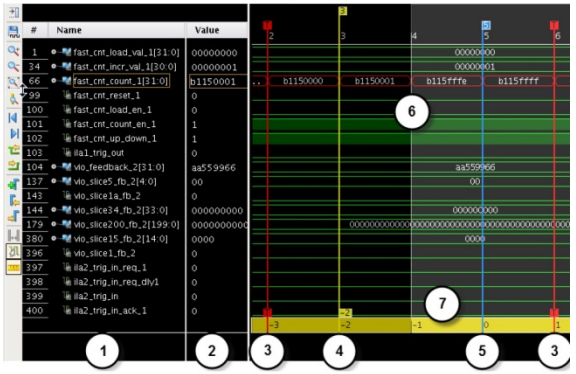
Debugging Logic Designs in Hardware

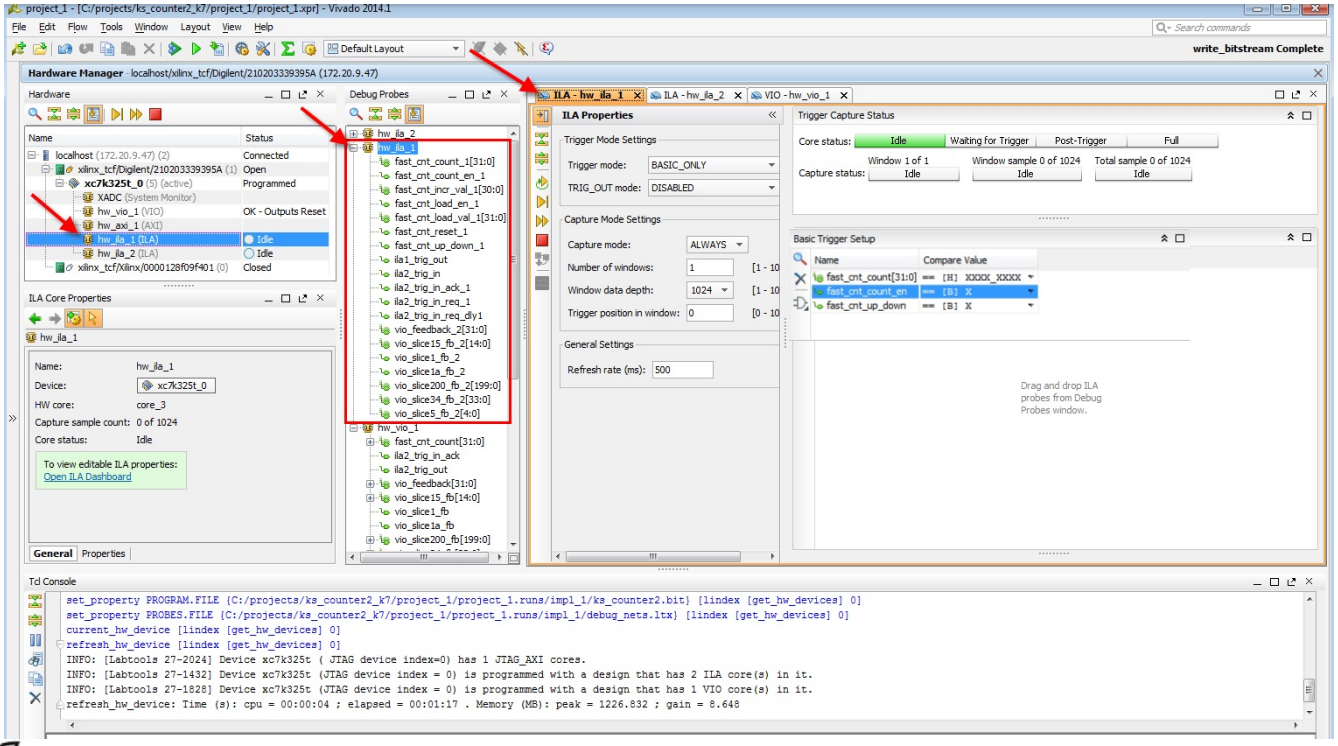
1. Connect to the hardware target and program the FPGA with the .bit file
2. Set up the ILA debug core trigger and capture controls.
3. Arm the ILA debug core trigger.
4. View the captured data from the ILA debug core in the **Waveform** window



Taking measurements

- ***Add Probes to Waveform***
- ***Add Probes to Basic Trigger Setup***
- ***Add Probes to Basic Capture Setup***
- ***Specify capture conditions***
- ***Arm the core and analyse received data***





Section IX: Conclusion

Acknowledgments (I)

NetFPGA Team at University of Cambridge (Past and Present):

Andrew Moore, David Miller, Muhammad Shahbaz, Martin Zadnik
Matthew Grosvenor, Yury Audzevich, Neelakandan Manihatty-Bojan,
Georgina Kalogeridou, Jong Hun Han, Noa Zilberman, Gianni Antichi,
Charalampos Rotsos, Marco Forconesi, Jinyun Zhang, Bjoern Zeeb

NetFPGA Team at Stanford University (Past and Present):

Nick McKeown, Glen Gibb, Jad Naous, David Erickson,
G. Adam Covington, John W. Lockwood, Jianying Luo, Brandon Heller, Paul Hartke, Neda Beheshti, Sara Bolouki, James Zeng,
Jonathan Ellithorpe, Sachidanandan Sambandan, Eric Lo

All Community members (including but not limited to):

Paul Rodman, Kumar Sanghvi, Wojciech A. Koszek,
Yahsar Ganjali, Martin Labrecque, Jeff Shafer, Eric Keller ,
Tatsuya Yabe, Bilal Anwer, Yashar Ganjali, Martin Labrecque,
Lisa Donatini, Sergio Lopez-Buedo

Kees Vissers, Michaela Blott, Shep Siegel, Cathal McCabe

Acknowledgements (II)

Disclaimer: *Any opinions, findings, conclusions, or recommendations expressed in these materials do not necessarily reflect the views of the National Science Foundation or of any other sponsors supporting this project.*

This effort is also sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL), under contract FA8750-11-C-0249. This material is approved for public release, distribution unlimited. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.