

NetFPGA Summer Course



Presented by:

**Andrew W Moore, Noa Zilberman, Gianni Antichi
Stephen Ibanez, Marcin Wojcik, Jong Hun Han,
Salvator Galea, Murali Ramanujam, Jingyun Zhang,
Yuta Tokusashi**

**University of Cambridge
July 24 – July 28, 2017**

<http://NetFPGA.org>

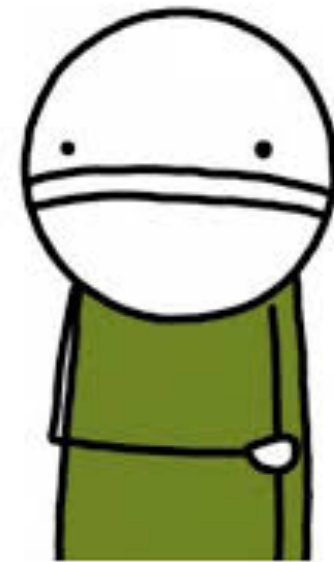
Finally, after hours of work you managed to finalise your HDL code and make it working in simulation!!!!



Once the bitfile is created you need to:

- Check if your design meet the timing.
- Debug your HW code if regression tests are not passed.

YAY.

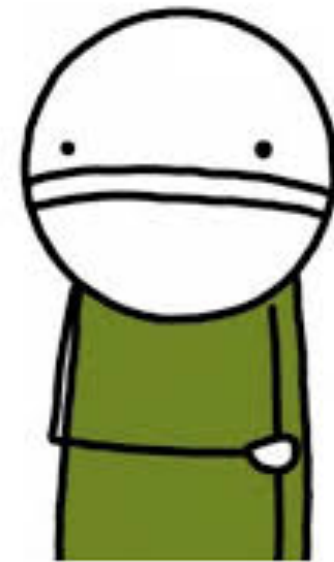


Natalie Dee Machine.com

Once the bitfile is created you need to:

- Check if your design meet the timing.
- Debug your HW code if regression tests are not passed.

YAY.



Natalie Dee Machine.com

Static Timing Analysis (STA)

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

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*

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

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 (STA)

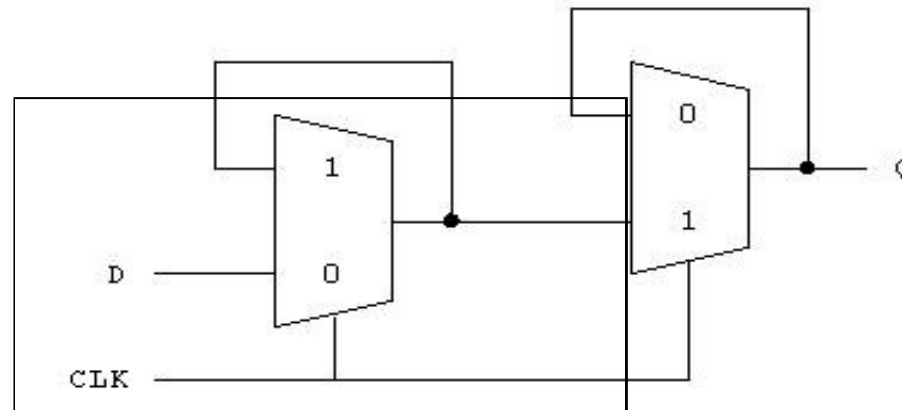
A design netlist is an interconnected set of ports, cells and nets

- 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



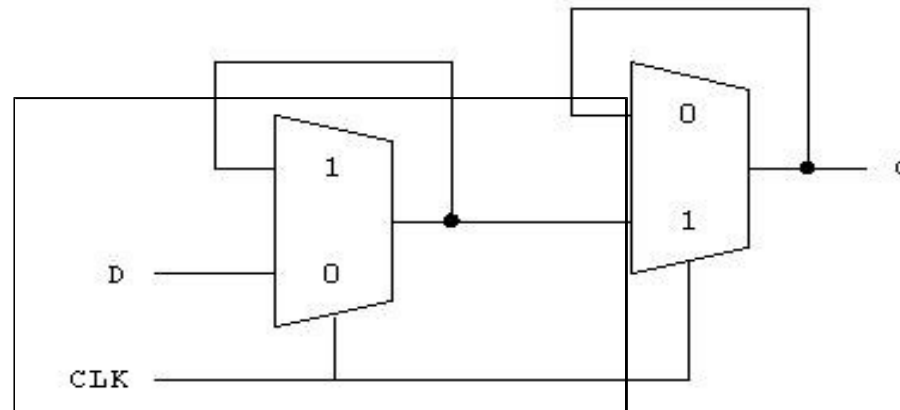
STA Goals

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



Component delays

Each component has delays to perform its function:

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

Component delays

Each component has delays to perform its function:

- LUT has propagation delay from its 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 → max)
- Temperature (min → max)

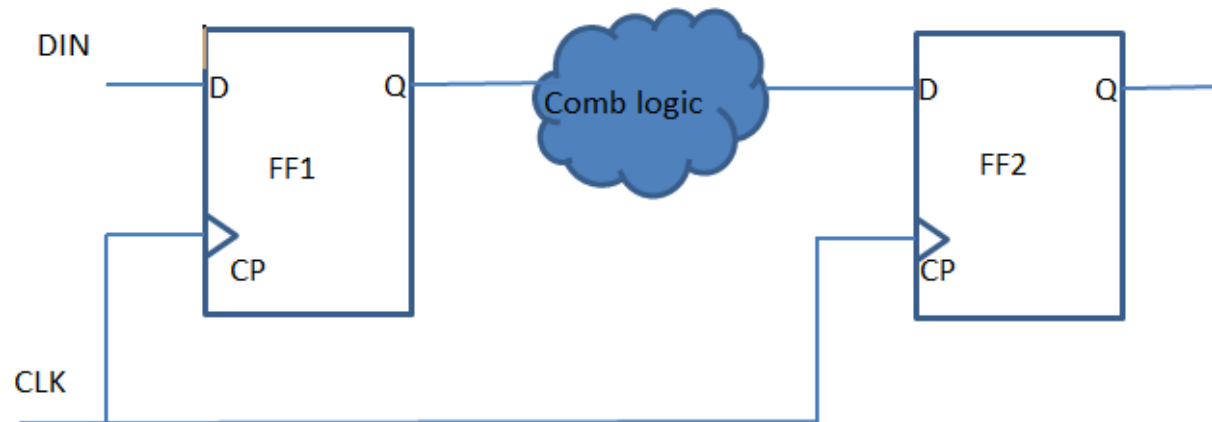
Static Timing Path

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

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

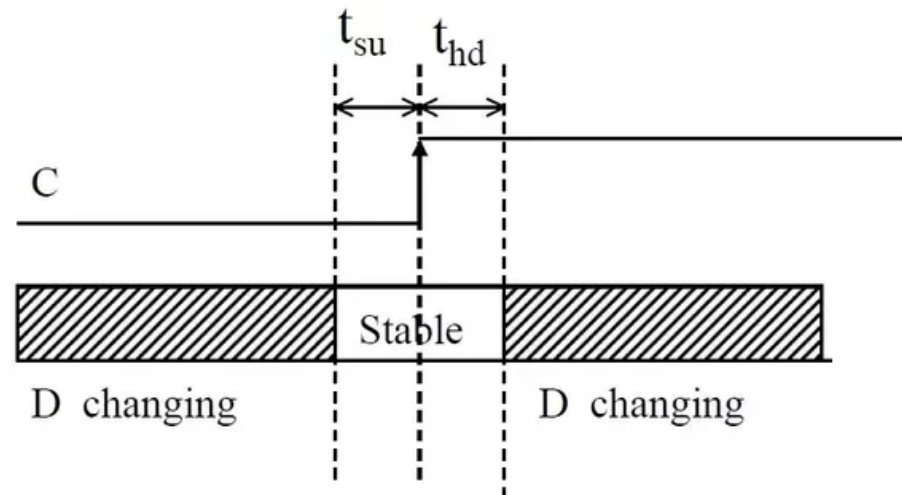
Data path delay – delay to the capturing FF

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



Setup check

Setup Timing Check checks that *data arrives in good time*
Setup, Hold Time



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

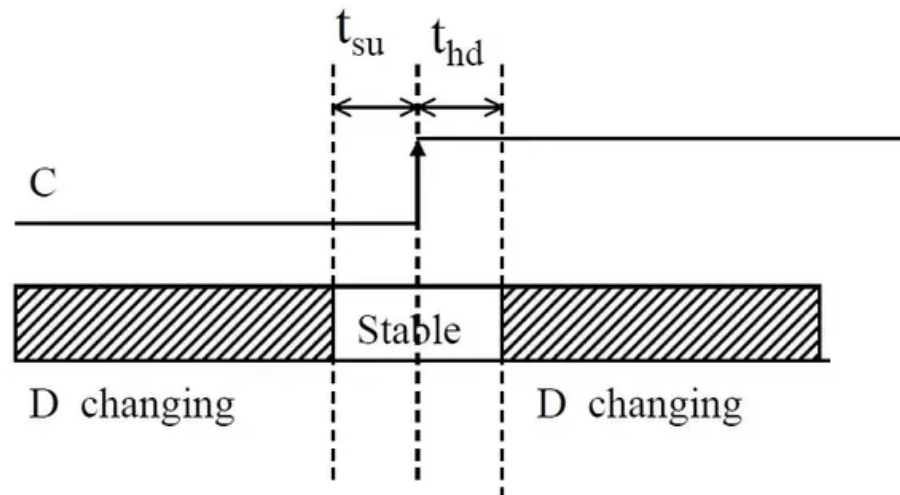
Simple case – same domain & only data path is considered:

$$T(D1_CLK) + T(FF1_{(CLK \rightarrow Q)}) + T(Comb) < T(CLK_{period}) - T(FF2_{(setup)}) - T(SU) + T(D2_CLK)$$

Hold check

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

Setup, Hold Time



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

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

$$T(D1_CLK) + T(FF1_{(clk \rightarrow Q)}) + T(Comb) > T(FF2_{(hold)}) + T(D2_CLK) + T(HU)$$

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



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**



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)**



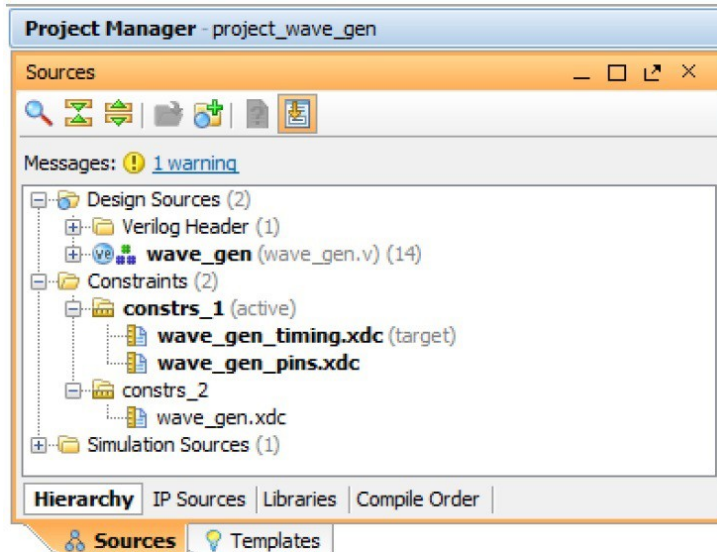
Constraints Methodology

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

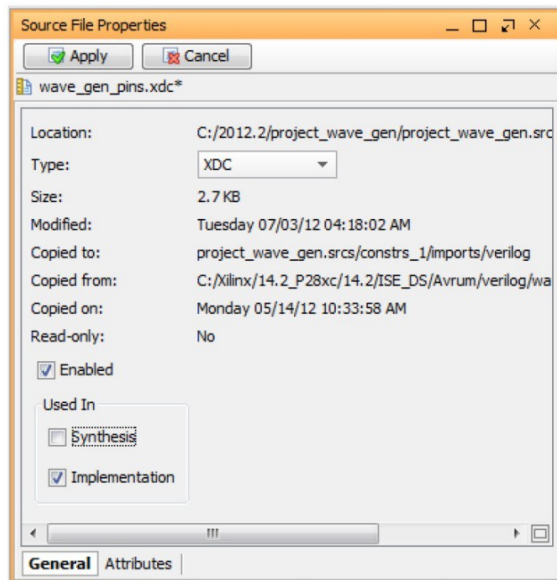


XDC constraints are a combination of:

- 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

Timing report

```
report_timing
INFO: [Timing 38-91] UpdateTimingParams: speed grade: -3 Delay Type: max.
INFO: [Timing 38-104] 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-ffg176l
Scripted 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_summe_dma/nf_riffa_dma_0/inst/riffa/riffa_inst/rRst_reg/C
              (rising edge-triggered cell FDPE clocked by userclk1 {rise@0.000ns fall@2.000ns period=4.000ns})
Destination: reference_nic_i/nf_summe_dma/nf_riffa_dma_0/inst/riffa_axis_attachment/riffa_to_axis_conv/tuser_reg[175]/R
              (rising edge-triggered cell FDPE 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)
GTHE2_CHANNEL_X1Y23  GTHE2_CHANNEL  0.000  0.000  r  reference_nic_i/nf_summe_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_summe_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_CLKINI_CLKOUT2)  0.069  1.044  r  reference_nic_i/nf_summe_dma/pcie3_7x_1/inst/gt_top_i/pipe_wrapper_i/pipe_clock_int.pipe_clock_i/mcm_i/CLKOUT2
net (fo=1, routed)  1.146  2.190  reference_nic_i/nf_summe_dma/pcie3_7x_1/inst/gt_top_i/pipe_wrapper_i/pipe_clock_int.pipe_clock_i/userclk1
```

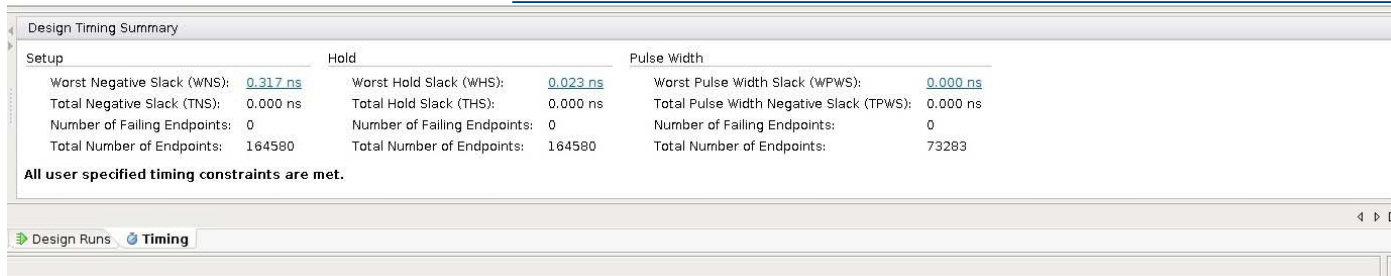
➤ Report Summary

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 command summary



Design Timing Summary

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 0.317 ns	Worst Hold Slack (WHS): 0.023 ns	Worst Pulse Width Slack (WPWS): 0.000 ns
Total Negative Slack (TNS): 0.000 ns	Total Hold Slack (THS): 0.000 ns	Total Pulse Width Negative Slack (TPWS): 0.000 ns
Number of Failing Endpoints: 0	Number of Failing Endpoints: 0	Number of Failing Endpoints: 0
Total Number of Endpoints: 164580	Total Number of Endpoints: 164580	Total Number of Endpoints: 73283

All user specified timing constraints are met.

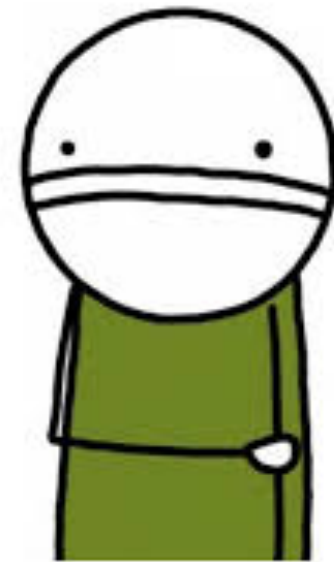
Design Runs: Timing

- **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**

Once the bitfile is created you need to:

- Check if your design meet the timing.
- **Debug your HW code if regression tests are not passed.**

YAY.



Natalie Dee Machine.com

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

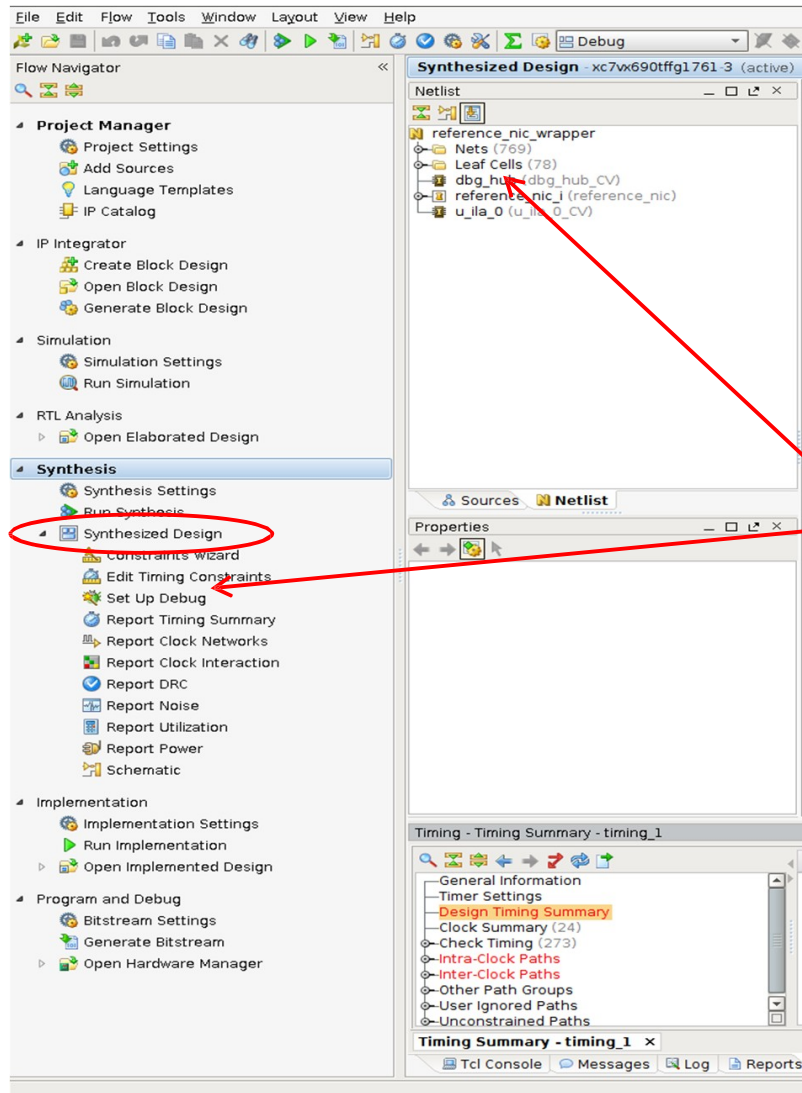
The first approach is more straight forward:

- 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.)

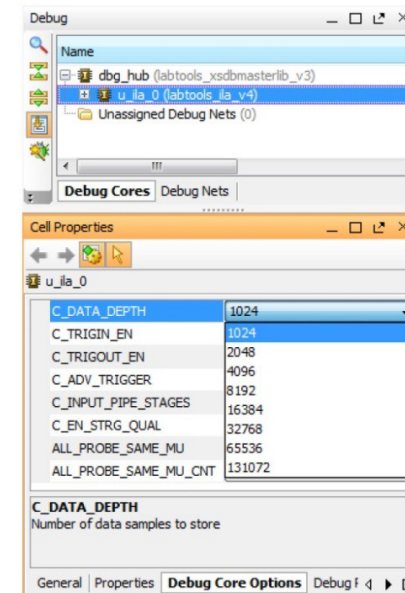
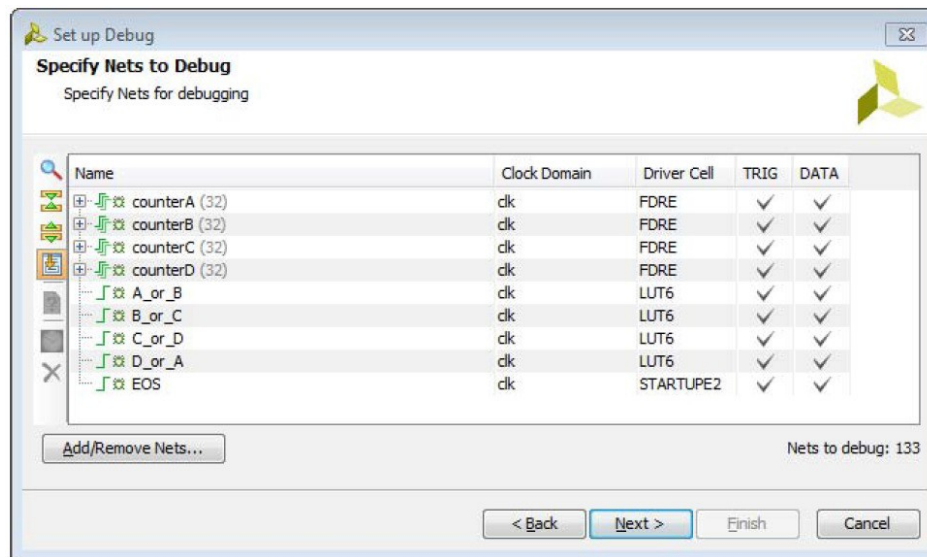


You can insert it from GUI as well:

- 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.)

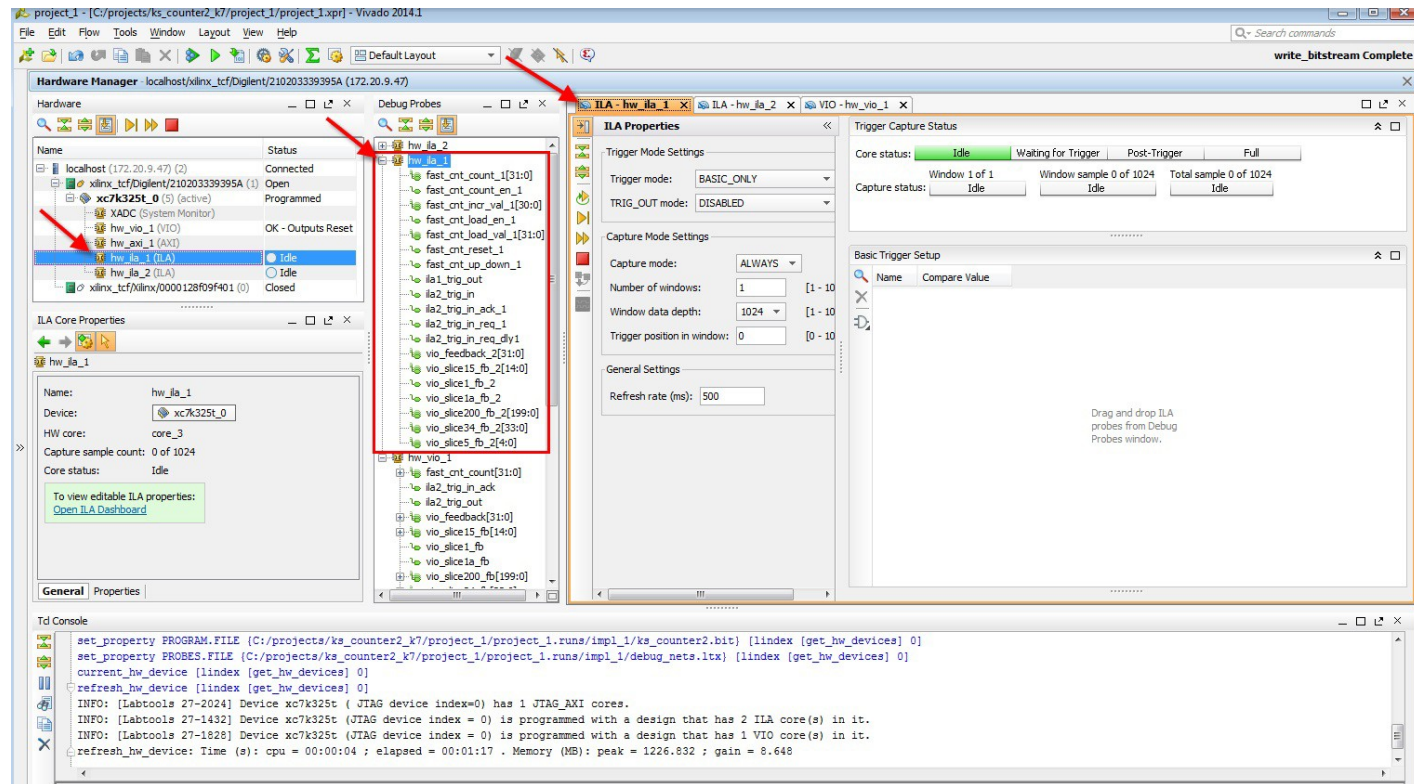
- XDC Commands can be also used to Insert Debug Cores

```
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

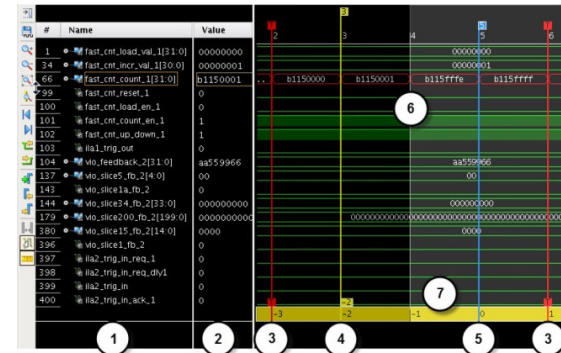
Debugging Logic Designs in Hardware

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



Hardware Manager: localhost/xilinx_tcf/Digilent/210203339395A (172.20.9.47)

ILA Properties:

- Trigger Mode Settings: Trigger mode: BASIC_ONLY, TRIG_OUT mode: DISABLED
- Capture Mode Settings: Capture mode: ALWAYS, Number of windows: 1, Window data depth: 1024, Trigger position in window: 0
- General Settings: Refresh rate (ms): 500

Basic Trigger Setup:

Name	Compare Value
fast_cnt_count[31:0]	== (R) XXXXX_XXXX
fast_cnt_count_en	== (B) X
fast_cnt_up_down	== (B) X

Td Console:

```
set_property PROGRAM_FILE [C:/projects/ks_counter2_k7/project_1/project_1.runs/impl_1/ks_counter2.bit] [lindex [get_hw_devices] 0]
set_property PROBES_FILE [C:/projects/ks_counter2_k7/project_1/project_1.runs/impl_1/debug_nets.tbx] [lindex [get_hw_devices] 0]
current_hw_device [lindex [get_hw_devices] 0]
refresh_hw_device [lindex [get_hw_devices] 0]
INFO: [Labtools 27-2024] Device xc7k325t ( JTAG device index=0) has 1 JTAG_AXI cores.
INFO: [Labtools 27-1432] Device xc7k325t (JTAG device index = 0) is programmed with a design that has 2 ILA core(s) in it.
INFO: [Labtools 27-1828] Device xc7k325t (JTAG device index = 0) is programmed with a design that has 1 VIO core(s) in it.
refresh_hw_device: Time (s): cpu = 00:00:04 ; elapsed = 00:01:17 . Memory (MB): peak = 1226.832 ; gain = 8.648
```

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)



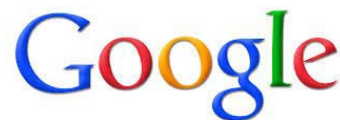
UNIVERSITY OF
CAMBRIDGE



EPSRC
Pioneering research
and skills



ALGO-LOGIC



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.