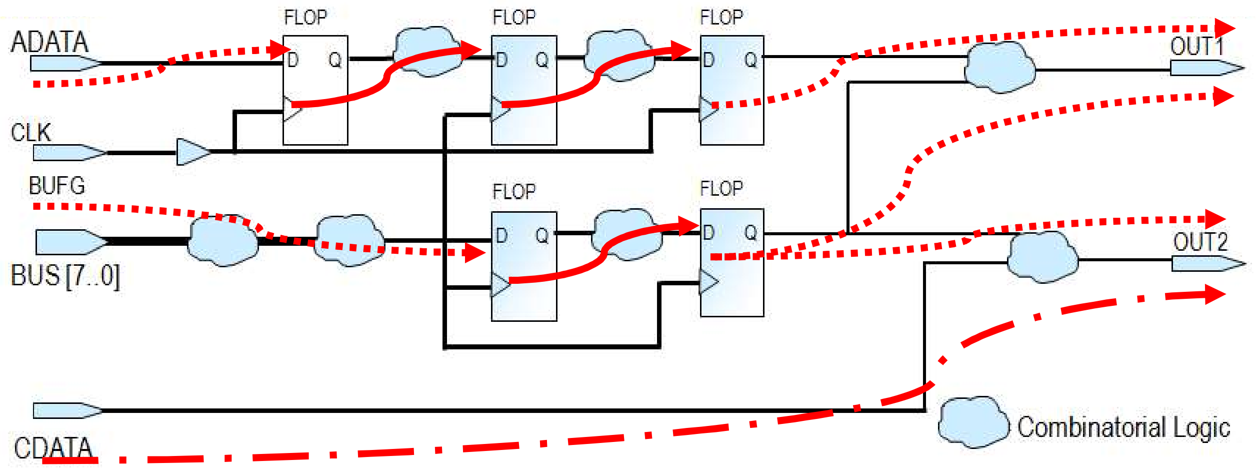
CORE | Vision
NEXT LEVEL | EMBEDDED DEVELOPMENT

Static Timing Analysis in a nutshell

Frank de Bont
Trainer / Consultant

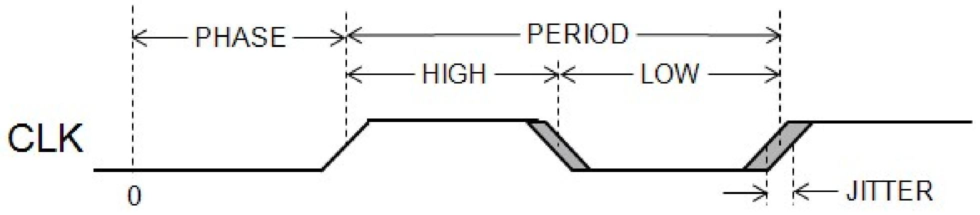
How does static timing analysis work ?

- Every device path in a design must be analyzed with respect to timing specifications / requirements
  - Catch timing-related errors faster and easier than gate-level simulation & board testing
- Designer must enter timing requirements & exceptions
  - Used to guide the FPGA tools during placement and routing
  - Used to compare against actual results (post-place and route)



Clocks

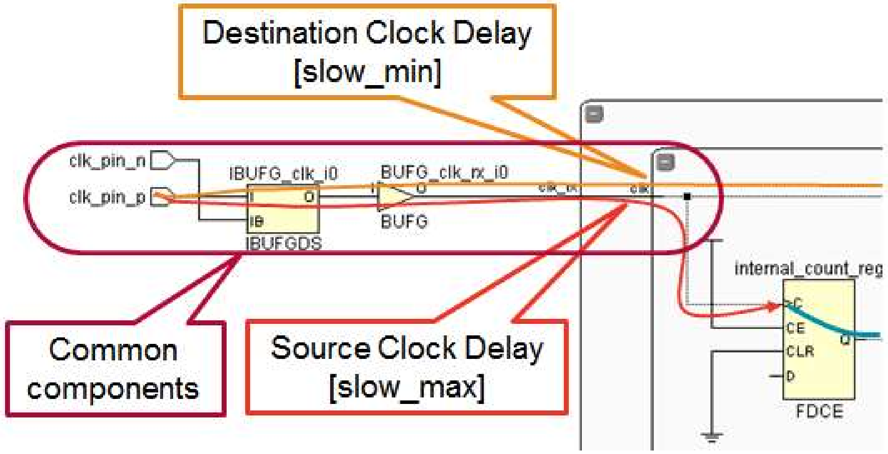
- Clocks are periodic signals
- Clocks have certain attributes
  - Period
  - Duty cycle
  - Jitter
  - Phase
    - create_clock –name CLK –period 10 –waveform {2.0 7.0} [get_ports CLK]



Clock Pessimism

- Often, the same components are on both the source clock path and the destination clock path
  - Source clock path is timed at [slow_max]
  - Destination clock path is timed at [slow_min]

- But a given cell cannot have two different delays at the same time
  - Results in pessimism due to the re-convergence of the clock path



Synchronous Input Interfaces

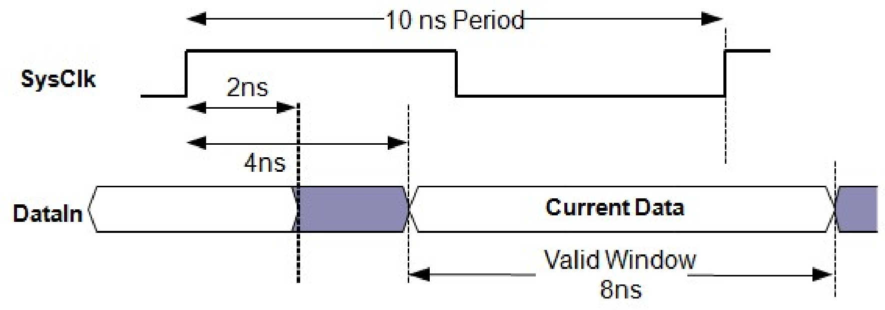
- Most interfaces to an FPGA use synchronous communication
  - The FPGA and the device driving the FPGA have some shared timing reference
    - This is usually a common clock or a related clock

- Complete static timing path through an input
  - Starts at a clocked element in the driving device
    - Referenced to a clock provided to the driving device
  - Ends at a clocked element in the FPGA
    - Referenced to the clock that propagates to the destination clocked element in the FPGA
  - Propagates through the elements between them
    - CLK -> Q of the external device
    - Board propagation time
    - Port of the FPGA
    - Combinatorial elements in the FPGA before the destination clocked element

Minimum and Maximum Delays

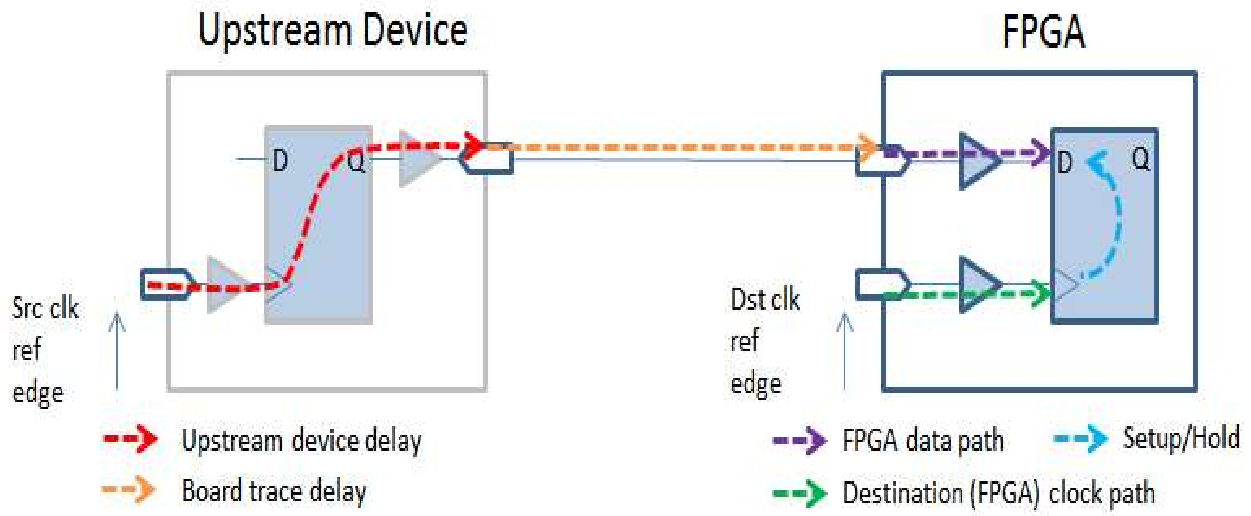
- By default, each input port can have one maximum delay and one minimum delay
  - The maximum delay is used for the setup check
  - The minimum delay is used for the hold check
  - Without the *–max* or *–min* option, the value supplied is used for both

```
create_clock –name SysClk –period 10
set_input_delay –clock SysClk 4 –max [get_ports DataIn]
set_input_delay –clock SysClk 2 –min [get_ports DataIn]
```



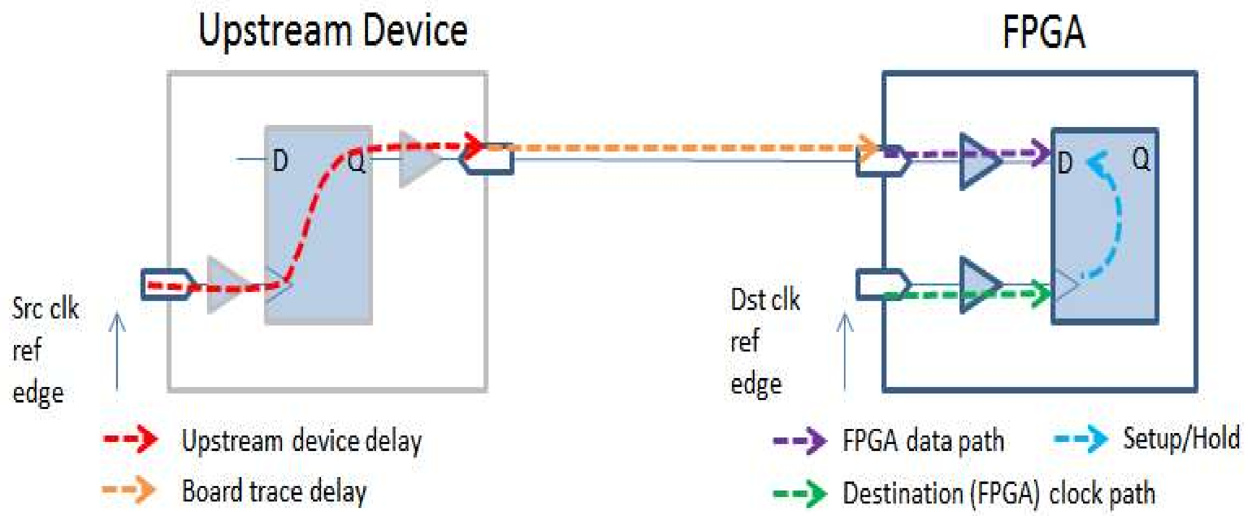
Input Timing Overview (1)

- Both setup and hold at the FPGA input register are analyzed for timing
- Source and destination clocks need to be defined
  - Create_clock (FPGA system clock and virtual upstream device clock)
  - System (common) clock or forwarded clock

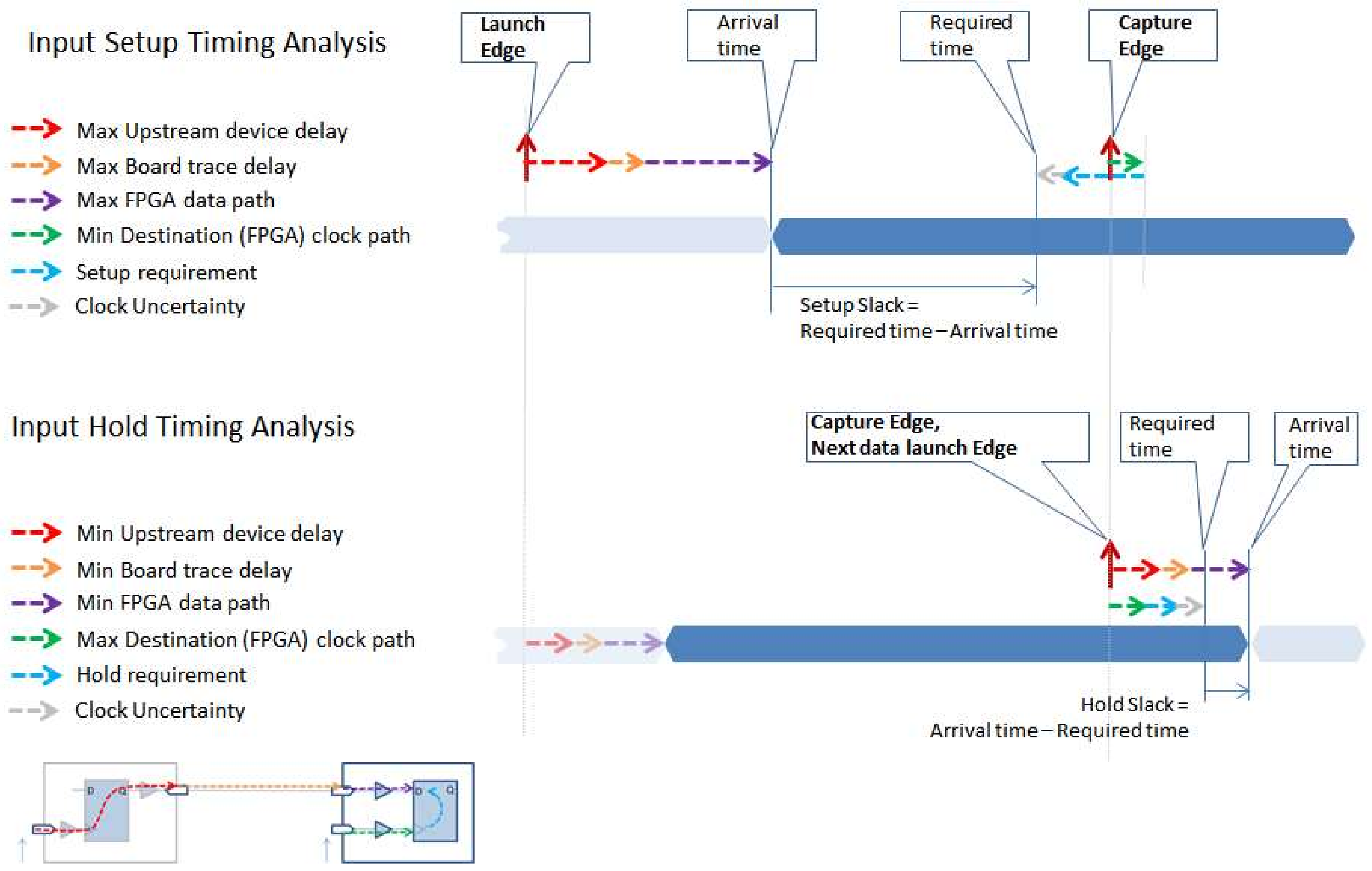


Input Timing Overview (2)

- ▶ Upstream device delay and board trace delay need to be specified
  - ▶ Maximum delay for setup analysis: set_input_delay *-max*
  - ▶ Minimum delay for hold analysis: set_input_delay *-min*
  - ▶ Referenced to the launch clock edge

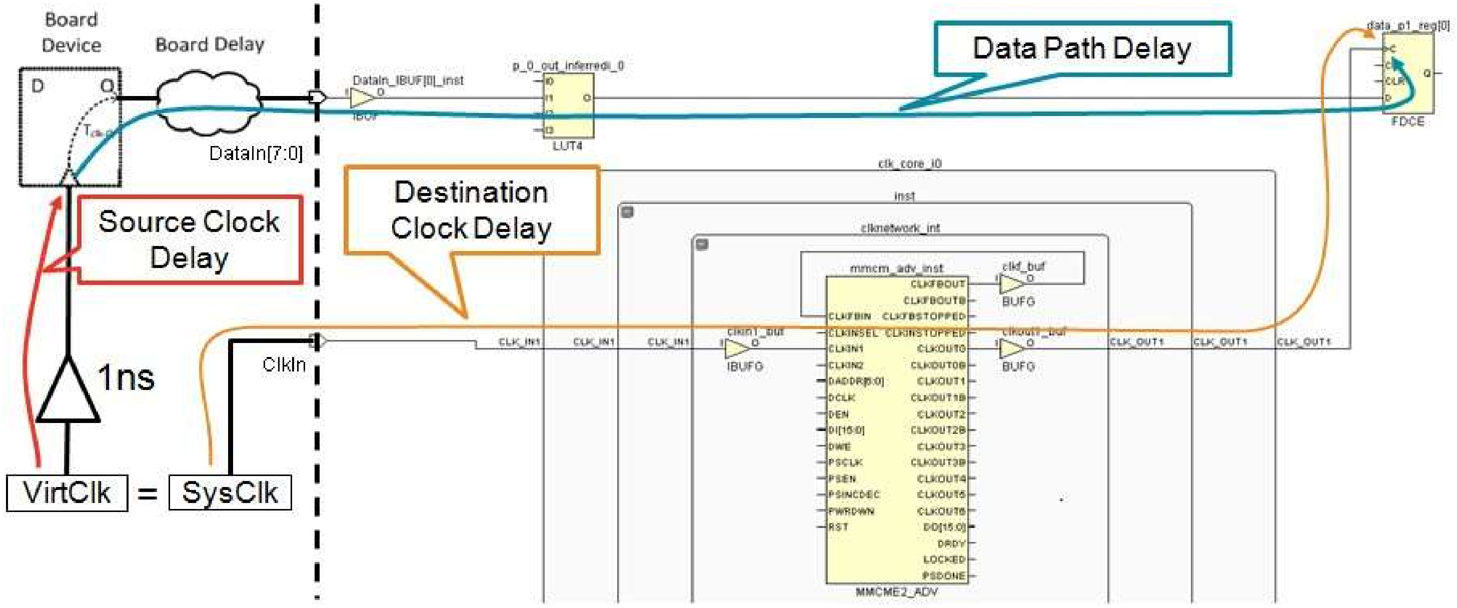


Input Timing Analysis


Input Setup Timing Analysis
Max Upstream device delay
Max Board trace delay
Max FPGA data path
Min Destination (FPGA) clock path
Setup requirement
Clock Uncertainty
Launch Edge
Arrival time
Required time
Capture Edge
Setup Slack = Required time – Arrival time
Input Hold Timing Analysis
Min Upstream device delay
Min Board trace delay
Min FPGA data path
Max Destination (FPGA) clock path
Hold requirement
Clock Uncertainty
Capture Edge, Next data launch Edge
Required time
Arrival time
Hold Slack = Arrival time – Required time
D
Q
D
Q

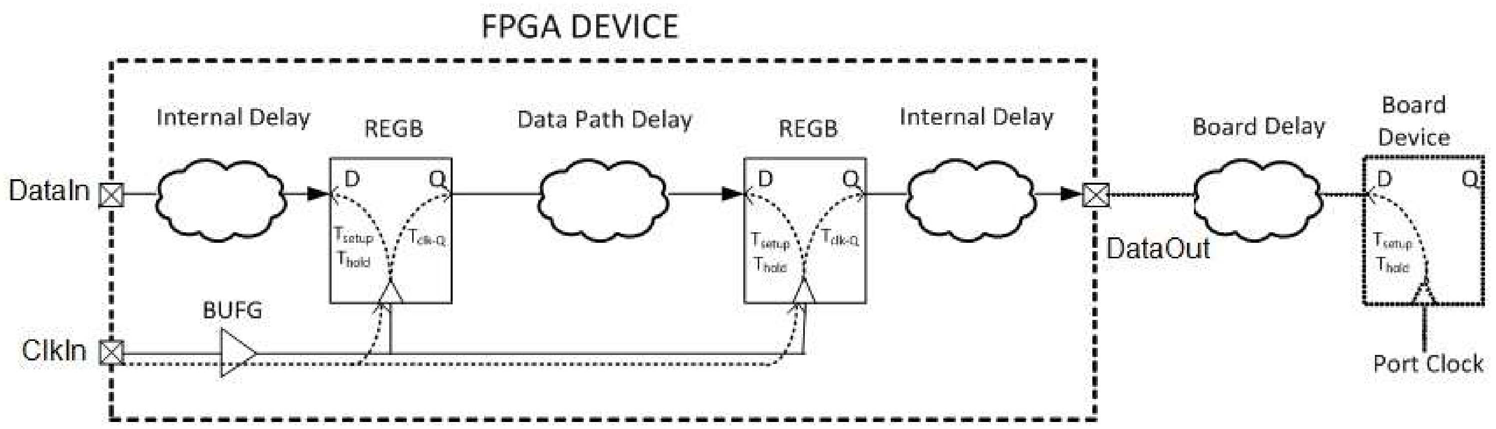
Input Static Timing Path with External Buffer

- ▶ create_clock –name SysClk –period 10 [get_ports Clkin]
- ▶ create_clock –name VirtClk –period 10
- ▶ set_clock_latency –source 1 [get_clocks VirtClk]
- ▶ set_input_delay –clock VirtClk 4 –*max* [get_ports DataIn]
- ▶ set_input_delay –clock VirtClk 2 –*min* [get_ports DataIn]



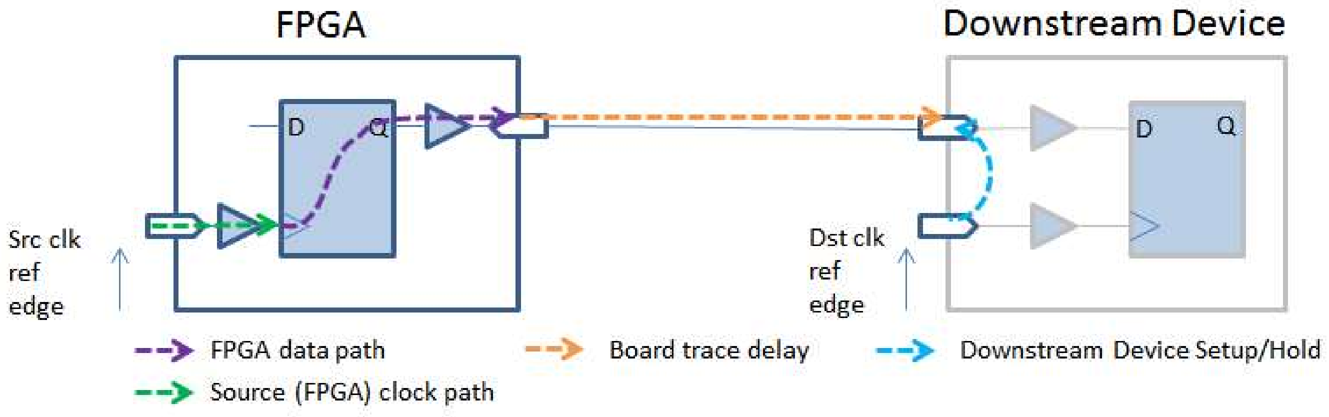
Completing the Static Timing Output Path

- To complete the static timing path, you need to describe the external elements for the static timing engine
  - What clock is used by the external device?
  - Delay between the output port of the FPGA and the external device's clock
  - Includes the required time of the external device and the board delay



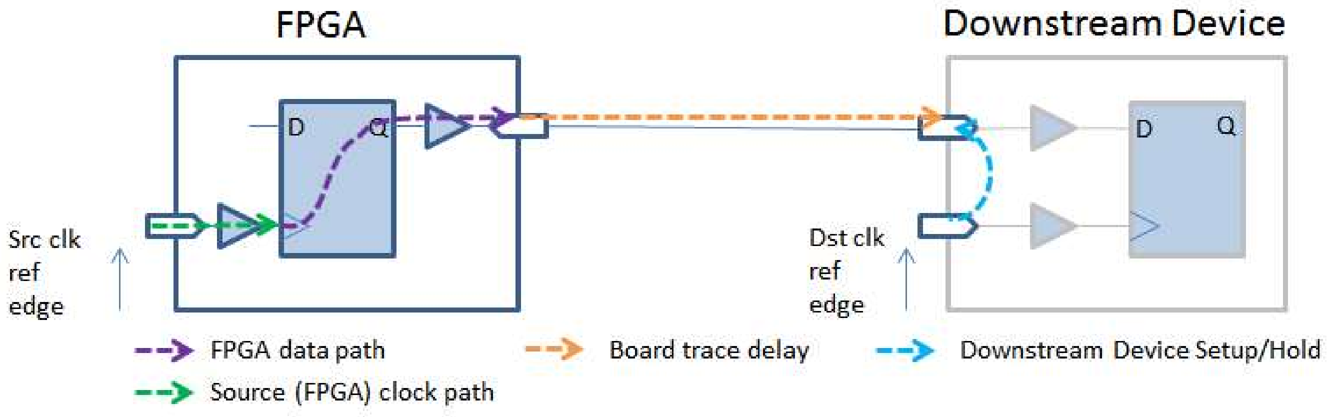
Output Timing Overview (1)

- Setup and hold at the downstream device input pins are analyzed for timing
- Source and destination clocks need to be defined, from which the tool derives the source and destination clock edges to consider
  - create_clock (actual FPGA clock and virtual downstream device clock)
  - system (common) clock or forwarded clock
    - create_generated_clock for forwarded output clocks

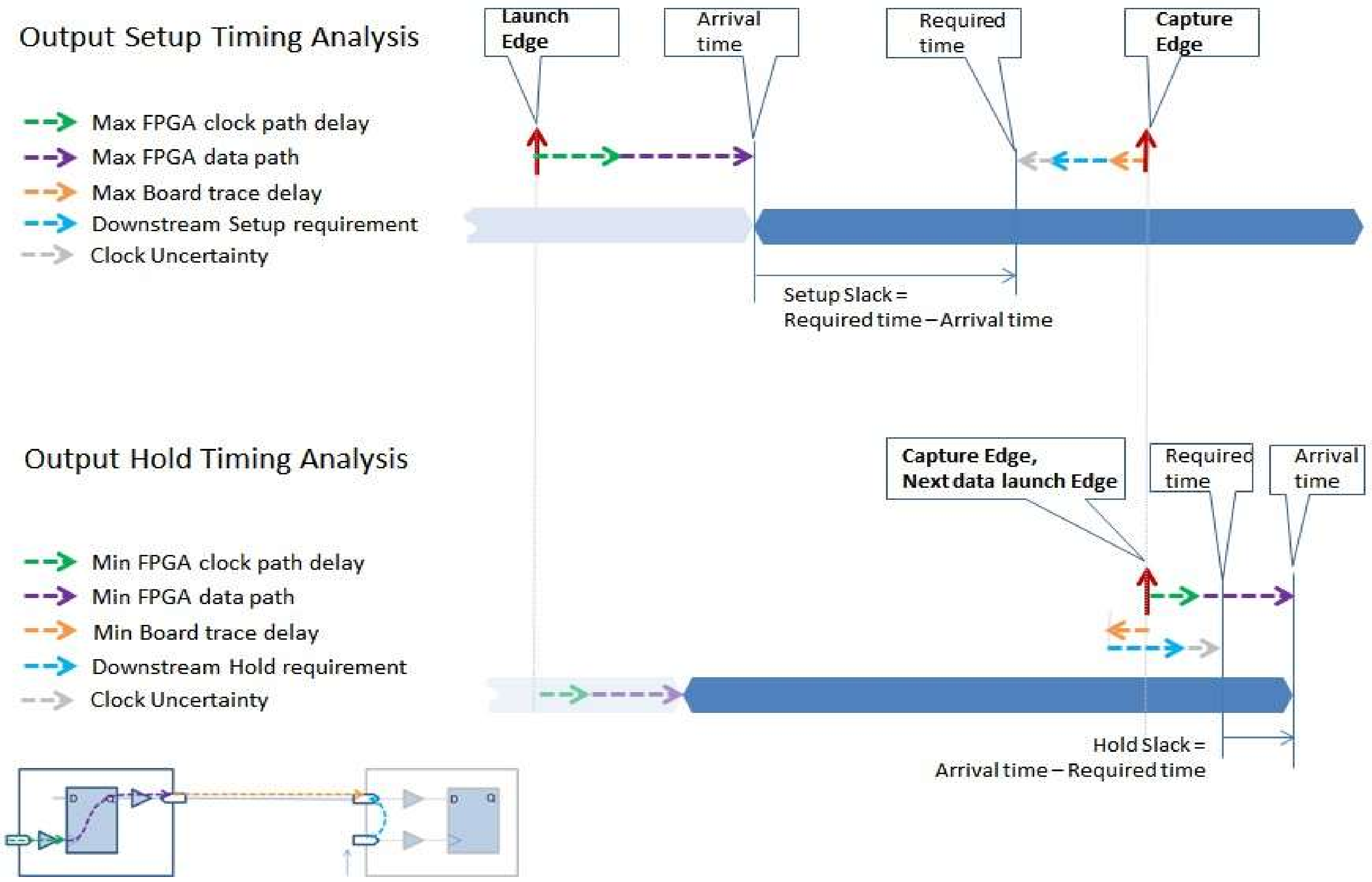


Output Timing Overview (2)

- Downstream device setup and hold requirements need to be specified
  - Downstream setup requirement:
    - set_output_delay –clock DstClk 4 –*max* [get_ports DataOut]
  - Downstream hold requirement:
    - set_output_delay –clock DstClk 2 –*min* [get_ports DataOut]
  - Board trace delays should be included
  - Referenced to the destination capture clock edge

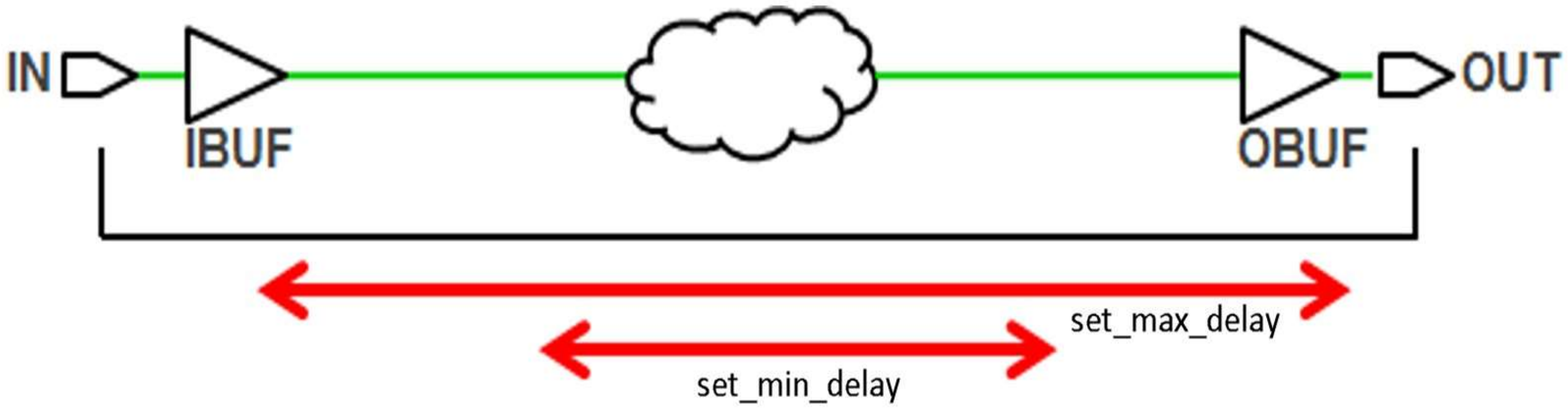


Output Timing Analysis


Output Setup Timing Analysis
Max FPGA clock path delay
Max FPGA data path
Max Board trace delay
Downstream Setup requirement
Clock Uncertainty
Launch Edge
Arrival time
Required time
Capture Edge
Setup Slack = Required time – Arrival time
Output Hold Timing Analysis
Min FPGA clock path delay
Min FPGA data path
Min Board trace delay
Downstream Hold requirement
Clock Uncertainty
Capture Edge, Next data launch Edge
Required time
Arrival time
Hold Slack = Arrival time – Required time
D
Q

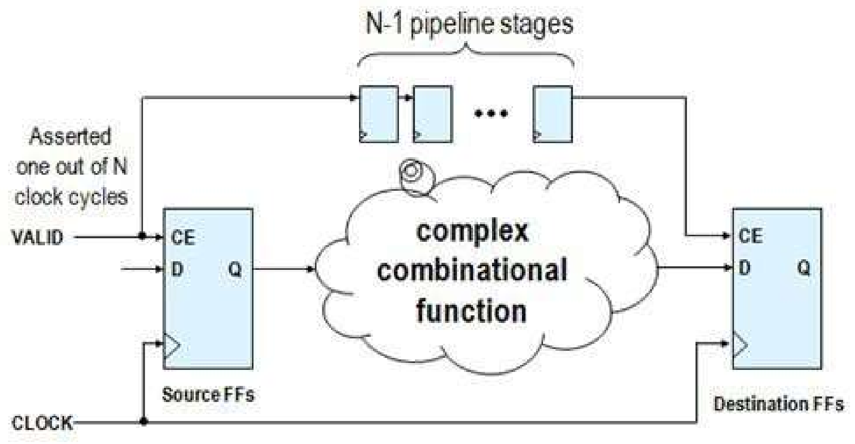
Combinational Delays

- Combinational delays are paths that enter and exit the FPGA without being captured by any sequential elements



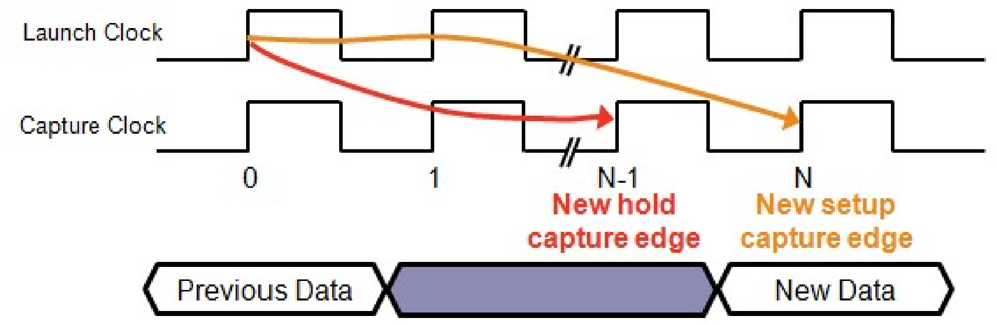
Multicycle Paths

- Multicycle paths occur when registers are not updated on consecutive clock cycles
  - Always at least N clock cycles between the time the source flip-flops update and the destination flip-flops capture the result
  - Typically, the registers are controlled by a clock enable
- Example shows one form of an N-cycle, multicycle path


N-1 pipeline stages
Asserted one out of N clock cycles
VALID
CE
D
Q
complex combinational function
Source FFs
CE
D
Q
Destination FFs
CLOCK

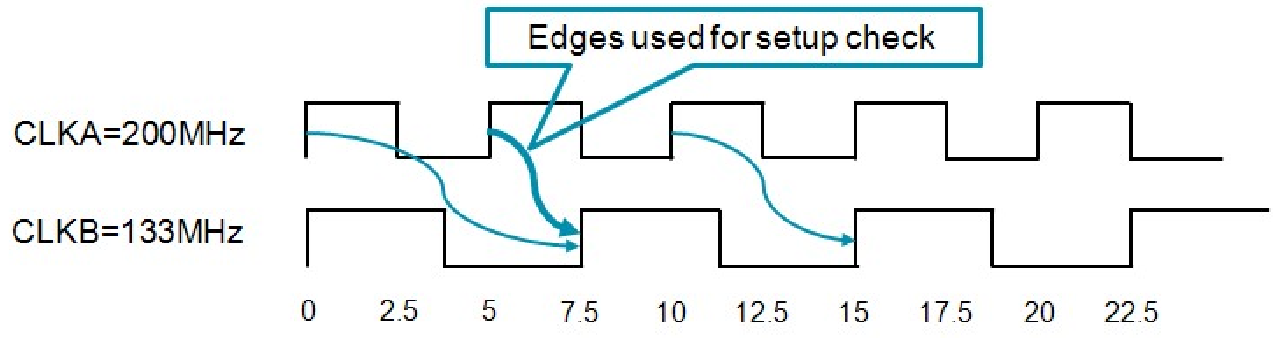
Modifying the Setup and Hold Capture Edge

- The setup capture edge can be modified by specifying a multiplier value N command:
  - set_multicycle_path –from $from_list to $to_list –setup <N>
  - The new setup capture edge will be <N> edges after the setup launch edge
- The hold capture edge usually needs to be modified with the –hold command:
  - set_ multicycle_path –from $from_list –to $to_list –hold <N-1>
  - Returns the hold capture edge to the correct edge



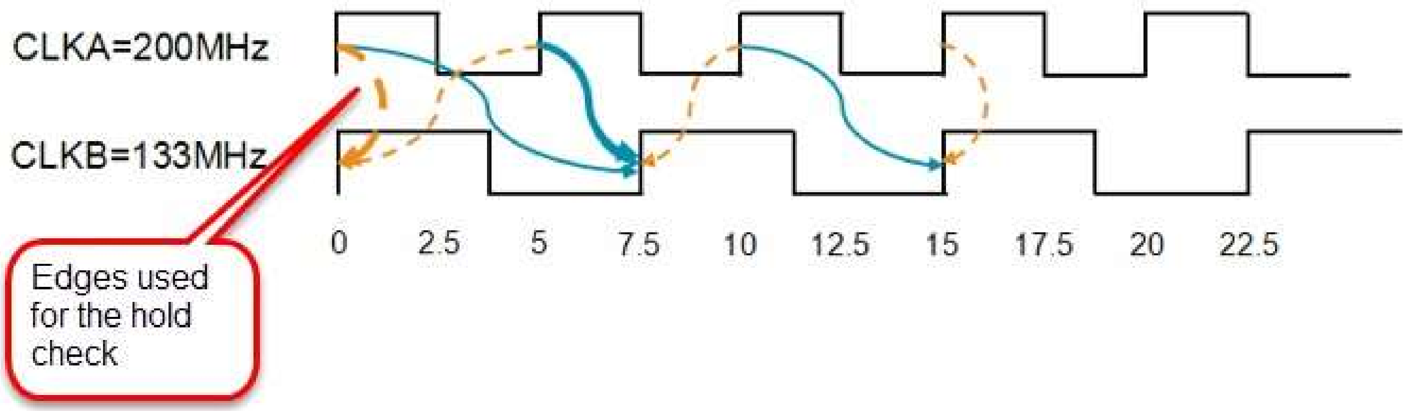
Setup Checks on Paths with Different Clocks

- When the static timing path starts and ends on different clocks, the setup check requirement is more complex
- All possible combinations of the source clock and destination clock are examined
  - From each edge on the source clock to the first edge of the destination clock that is later in time
  - The pair with the tightest requirement is used as the launch and capture edge
- Once the launch and capture edge is determined, the setup check is performed



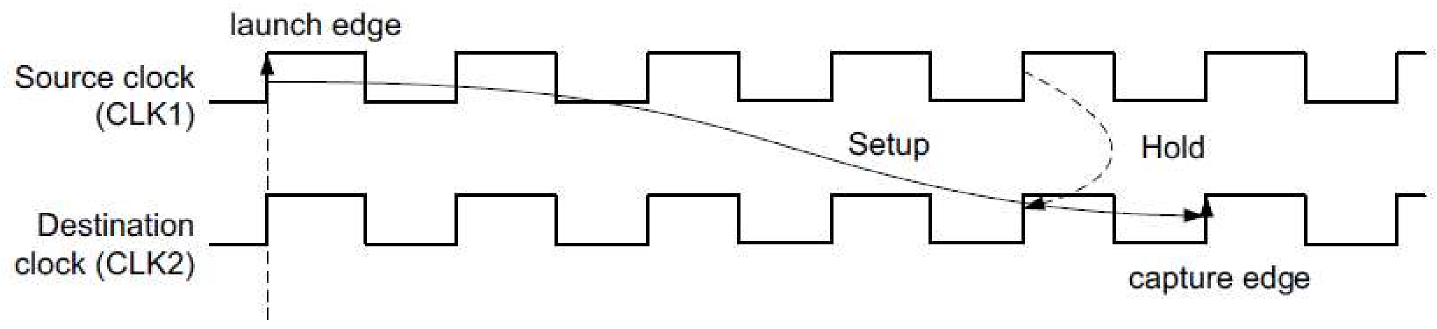
Hold Checks on Paths with Different Clocks

- There are two possible hold checks for each possible setup check
  - From the setup launch edge to the edge before the setup capture edge
  - From the edge after the setup launch edge to the setup capture edge
- The launch and capture edges with the tightest requirement are used for the hold check



Example of Multicycle Path (1)

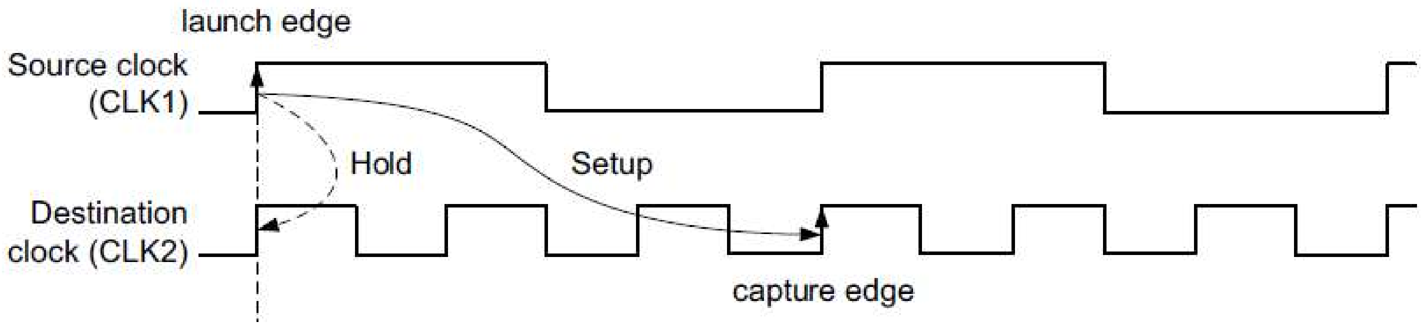
- Same clock domain or between synchronous clock domains with same period and no phase-shift
  - set_multicycle_path N –setup -from CLK1 –to CLK2
  - set_multicycle_path N-1 –hold -from CLK1 –to CLK2



Example of Multicycle Path (2)

- Between SLOW-FAST synchronous clock domains

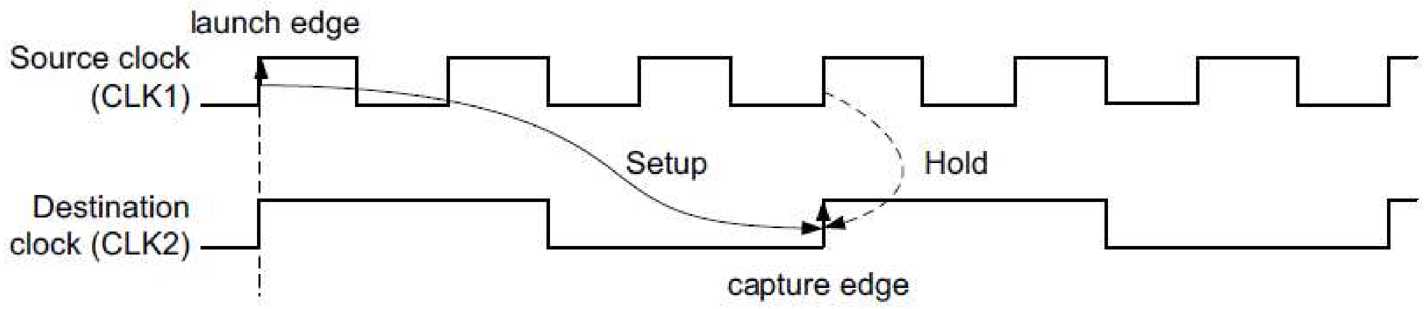
  - set_multicycle_path N –setup -from CLK1 –to CLK2
  - set_multicycle_path N-1 –hold –*end* -from CLK1 –to CLK2



Example of Multicycle Path (3)

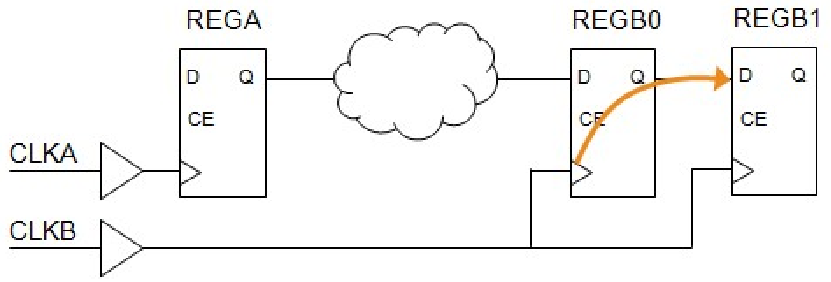
- Between FAST-SLOW synchronous clock domains

  - set_multicycle_path N –setup –*start* -from CLK1 –to CLK2
  - set_multicycle_path N-1 –hold -from CLK1 –to CLK2



Constraining Metastability Flip-Flops

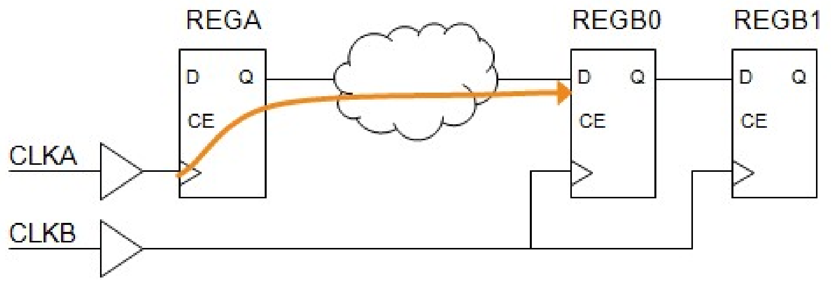
- When a flip-flop (REGB0) samples an asynchronous input, the flip-flop can go metastable
  - The metastability will probabilistically resolve after some time
    - Back to back flip-flops allow one clock period for the metastability to resolve before the second flip-flop samples it (REGB1)
- The Vivado timing engine sees the REGB0 → REGB1 path as a normal static timing path, subject to a normal setup check
  - Allows one clock period of CLKB for the propagation, including routing
- To leave time for metastability resolution, the requirement on this path should be changed



- set_max_delay –from [get_cells REGB0] –to [get_cells REGB1] 2

Paths Between Different Clock Domains

- When crossing asynchronously between different clock domains a clock crossing circuit is required
  - Many approaches exist
    - Use Gray code
    - Use an enable to determine a stable point
    - Use a FIFO
  - All of these approaches have implicit assumptions
  - Generally, the skew between different bits of the bus to be crossed cannot exceed one clock period
    - It is not possible to constrain skew, but it is possible to constrain the maximum delay on all bits of the bus
- In these cases, the clock propagation analysis is irrelevant and should be overridden with the –datapath_only option



- set_max_delay –from [get_cells REGA] –to [get_cells REGB0] 5 –datapath_only

Core|Vision

- Core|Vision has more then 200 man years of design experience in hard- and software development. Our competence areas are:

- Real-time Embedded Systems Solutions
- FPGA Design Consultancy
- FPGA Training
- IIoT Embedded Sensor Solutions

LLOYD'S REGISTER • LRQA
ISO 9001

DOULOS

IBM Registered Business Partner™

XILINX® | AUTHORIZED TRAINING PROVIDER

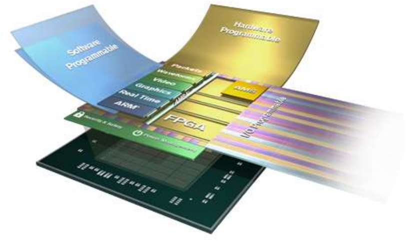
AnySens Powered by CORE | Vision

Visit our booth #31

Training Program

Essentials of FPGA Design	*1 day*
Designing for Performance	*2 days*
Advanced FPGA Implementation	*2 days*
Design Techniques for Lower Cost	*1 day*
Designing with Spartan-6 and Virtex-6 Family	*3 days*
Essential Design with the PlanAhead Analysis Tool	*1 day*
Advanced Design with the PlanAhead Analysis Tool	*2 days*
Xilinx Partial Reconfiguration Tools and Techniques	*2 days*
Designing with the 7 Series Families	*2 days*

XILINX®
AUTHORIZED TRAINING PROVIDER



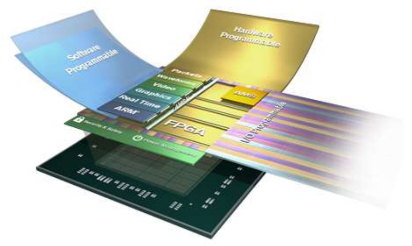
CORE | Vision
NEXT LEVEL EMBEDDED DEVELOPMENT

8 OKT
VAN DER VALK HOTEL EINDHOVEN
D&E event 2019

Training Program

▶ Designing FPGAs Using the Vivado Design Suite 1	*2 days*
▶ Designing FPGAs Using the Vivado Design Suite 2	*2 days*
▶ Designing FPGAs Using the Vivado Design Suite 3	*2 days*
▶ Designing FPGAs Using the Vivado Design Suite 4	*2 days*
▶ Designing with the UltraScale and UltraScale^{+} Architecture	*2 days*
▶ Vivado Design Suite for ISE Software Project Navigator User	*1 day*
▶ Vivado Design Suite Advanced XDC and Static Timing Analysis for ISE Software User	*2 days*

XILINX®
AUTHORIZED TRAINING PROVIDER



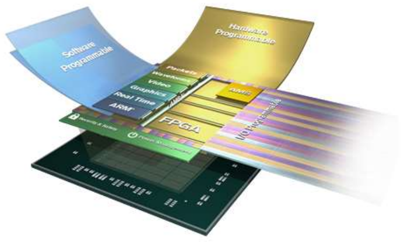
CORE | Vision
NEXT LEVEL EMBEDDED DEVELOPMENT

8 OKT
VAN DER VALK HOTEL EINDHOVEN
D&E event 2019

Training Program

▶ Designing with Multi Gigabit Transceivers	*2 days*
▶ C-Based Design: High Level Synthesis with Vivado HLx	*2 days*
▶ DSP Design Using System Generator	*2 days*
▶ Essential DSP Implementation Techniques for Xilinx FPGAs	*2 days*

XILINX®
AUTHORIZED TRAINING PROVIDER



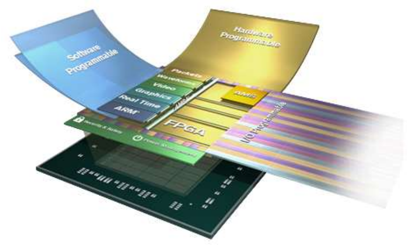
CORE | Vision
NEXT LEVEL EMBEDDED DEVELOPMENT

8 OKT
VAN DER VALK HOTEL EINDHOVEN
D&E event 2019

Training Program

▶ Embedded Systems Design	*2 days*
▶ Embedded Systems Software Design	*2 days*
▶ Advanced Features and Techniques of SDK	*2 days*
▶ Advanced Features and Techniques of EDK	*2 days*
▶ Zynq All Programmable SoC Systems Architecture	*2 days*
▶ Zynq UltraScale$^+$ MPSoC for the System Architect	*2 days*
▶ Introduction to the SDSoC Development Environment	*1 day*
▶ Advanced SDSoC Development Environment & Methodology	*2 days*

XILINX®
AUTHORIZED TRAINING PROVIDER



CORE | Vision
NEXT LEVEL EMBEDDED DEVELOPMENT

8 OKT
VAN DER VALK HOTEL EINDHOVEN
D&E event 2019

Training Program

▶ VHDL for Designers	*3 days*
▶ Advanced VHDL	*2 days*
▶ Comprehensive VHDL	*5 days*
▶ Expert VHDL Verification	*3 days*
▶ Expert VHDL Design	*2 days*
▶ Expert VHDL	*5 days*
▶ Essential Digital Design Techniques	*2 days*
▶ Developing with Embedded Linux	*2 days*
▶ Essential Phython	*2 days*

DOULOS Delivering KnowHow

CORE NEXT LEVEL | Vision EMBEDDED DEVELOPMENT

8 OKT VAN DER VALK HOTEL EINDHOVEN D&E event 2019

Training Program

- Solving Clock Domain Crossover Conflicts *2 days*

VSYNC circuits

CORE | Vision
NEXT LEVEL EMBEDDED DEVELOPMENT

8 OKT
VAN DER VALK HOTEL
EINDHOVEN
D&E event 2019