Joint ICTP-IAEA School on Detector Signal Processing and Machine Learning for Scientific Instrumentation and Reconfigurable Computing

The Abdus Salam International Centre for Theoretical Physics

IAEA International Atomic Energy Agency

unesco United Nations Educational, Scientific and Cultural Organization

Advanced Topics in SoC-FPGA

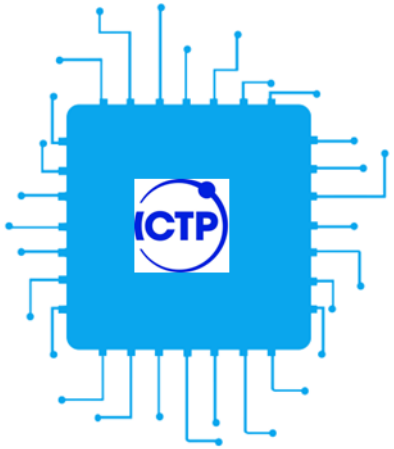
Cristian Sisterna

Senior Associate, ICTP-MLAB

Professor at Universidad Nacional San Juan- Argentina

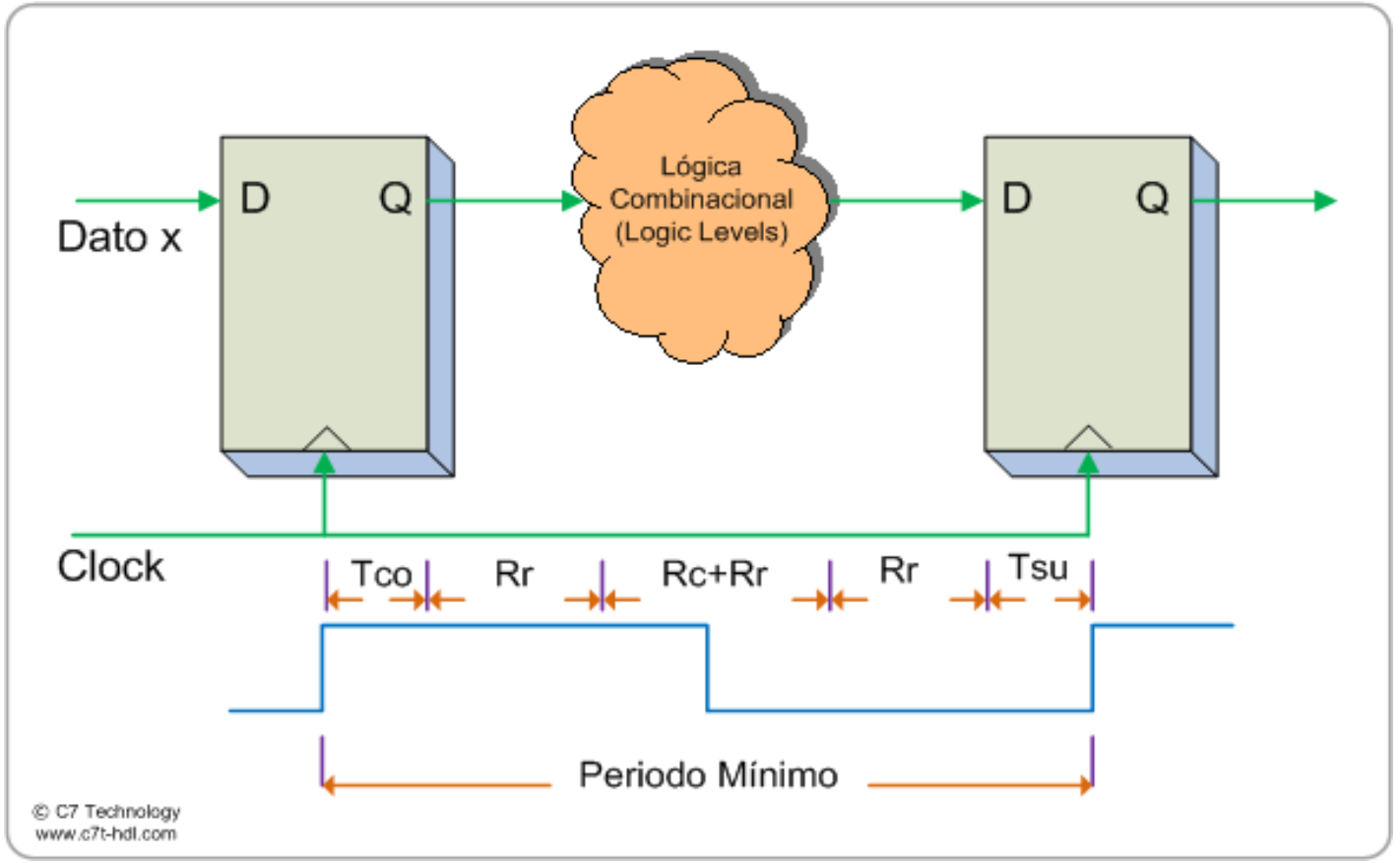
Agenda

- FPGA Timing
- Solutions for Timing Issues
- Systems with Different Clock Domains
- Handshaking Between Components
- FPGA Routing



FPGA Timing

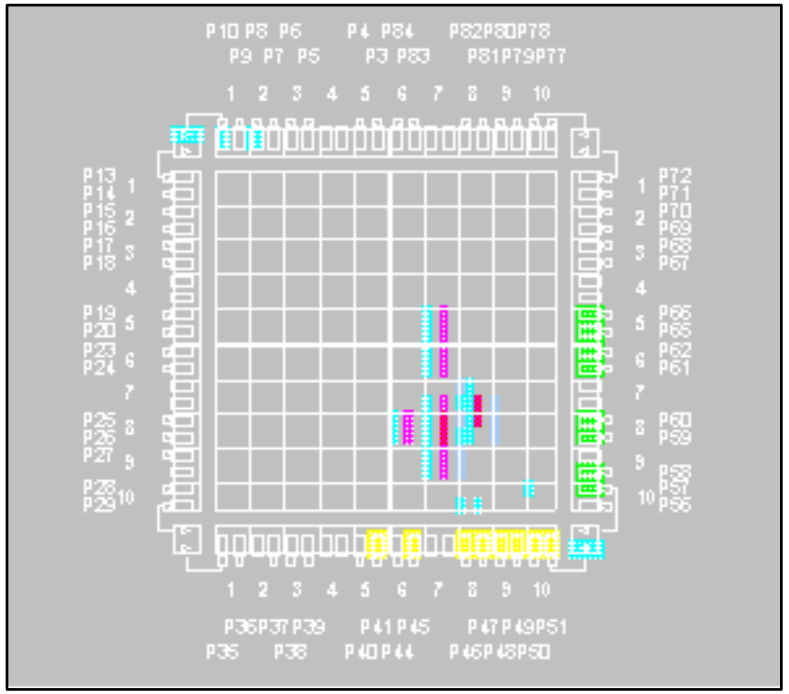
Mínimum Period



$$Tmin \cong Tco_{max} + RetRuteoTotal_{max} + (RetCombTotal + RetRuteo)_{max} + Tsu_{max} + 10\%Margen$$

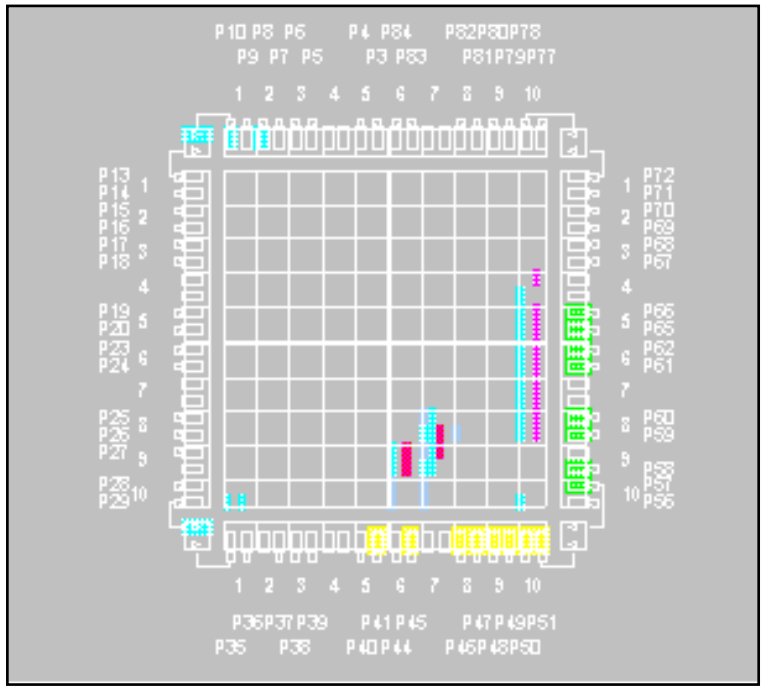
Constraints and P&R

Without global timing constraints



Logic is placed randomly

With global timing constraints



Logic is placed to result in a faster design

Clock Constraint

In Vivado, a clock period constraint defines the frequency of a clock signal to guide the timing analysis of an FPGA design, typically using the **create_clock** command.

This constraint is critical for ensuring the design meets performance requirements by specifying the clock's period (in nanoseconds, e.g., 4.0 ns).

- ✓ **Syntax**: create_clock -period <value> -name <clock_name> [get_ports <port_name>]
- ✓ **Example**: **create_clock -period 10.0 -name sys_clk [get_ports sys_clk_pin]**
  - ✓ **period**: Sets the clock's period in nanoseconds. For a 100 MHz clock, the period is 10 ns
  - ✓ **name**: Assigns a user-friendly name to the clock object, making it easier to reference.
  - ✓ **[get_ports <port_name>]**: Links the constraint to the physical input pin of your FPGA design that receives the clock signal.

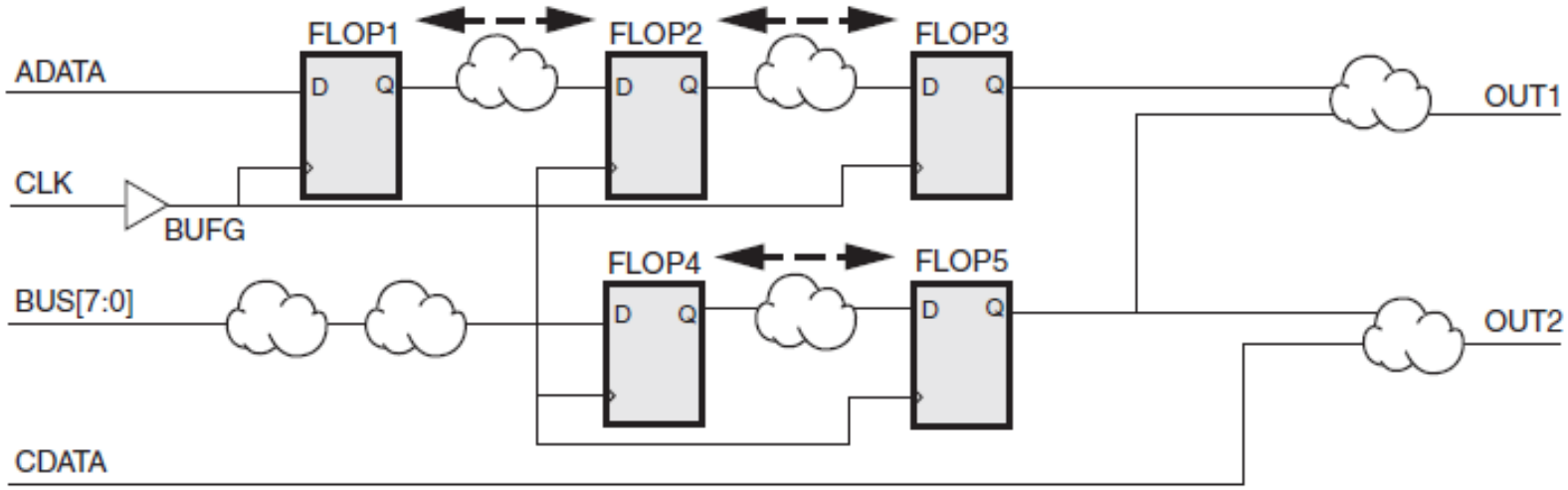
Clock Constraint

Why it's important

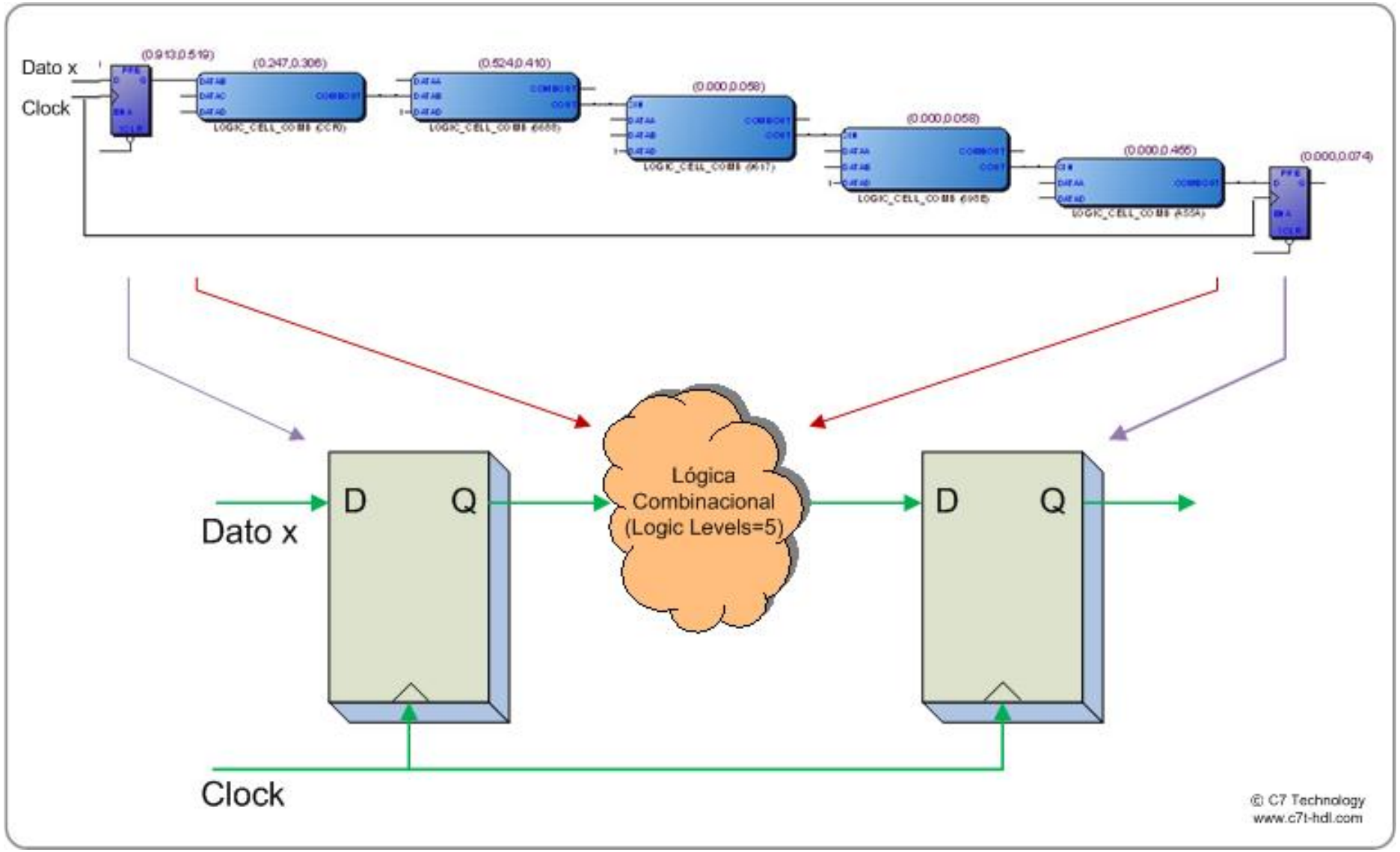
- ✓ **Timing analysis**: The constraint tells the Vivado timing engine the maximum frequency the design should run at.
- ✓ **Path identification**: The engine uses this to analyze all timing paths between registers and to calculate slack.
- ✓ **Positive vs. negative slack**:
  - ✓ **Positive slack**: The design meets or exceeds the timing requirements.
  - ✓ **Negative slack**: A timing violation has occurred, and the design cannot run at the specified frequency.
    - ✓ This often indicates a need for design changes like pipelining or simplifying logic in critical paths.

Clock Constraints Coverage

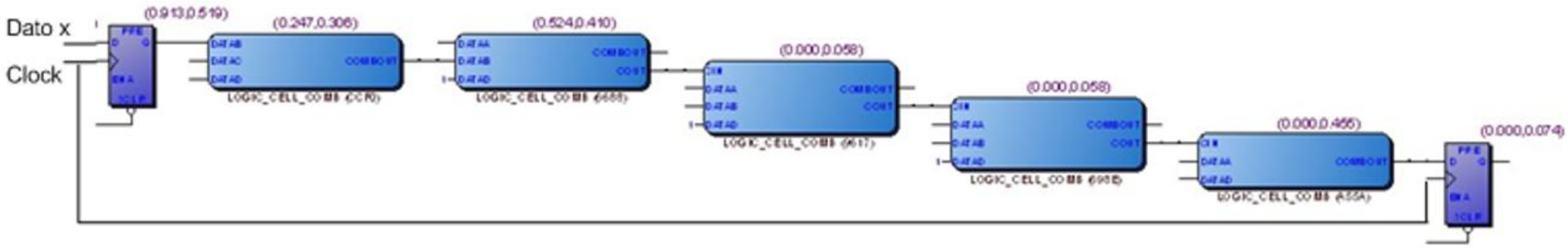
- The PERIOD constraint covers paths between synchronous elements clocked by the reference net
- The PERIOD constraint **does *NOT* analyze** delay paths:
  - From input pads to output pads (purely combinatorial)
  - From input pads to synchronous elements
  - From synchronous elements to output pads
  - Between unrelated clocks



Clock Path Example



Clock Path Report



2.331					data path
0.199		uTco	1	FF X14 Y1 N1	in2 reg[0]
0.000	RR	CELL	1	FF X14 Y1 N1	in2 reg[0]|q
0.247	RR	IC	1	LCCOMB X14 Y1 N10	s1[0]~0|datab
0.306	RR	CELL	1	LCCOMB X14 Y1 N10	s1[0]~0|combout
0.524	RR	IC	2	LCCOMB X14 Y1 N16	out reg[0]~4|datab
0.410	RR	CELL	1	LCCOMB X14 Y1 N16	out reg[0]~4|cout
0.000	RR	IC	2	LCCOMB X14 Y1 N18	out reg[1]~6|cin
0.058	RF	CELL	1	LCCOMB X14 Y1 N18	out reg[1]~6|cout
0.000	FF	IC	2	LCCOMB X14 Y1 N20	out reg[2]~8|cin
0.058	FR	CELL	1	LCCOMB X14 Y1 N20	out reg[2]~8|cout
0.000	RR	IC	1	LCCOMB X14 Y1 N22	out reg[3]~10|cin
0.455	RR	CELL	1	LCCOMB X14 Y1 N22	out reg[3]~10|combout
0.000	RR	IC	1	FF X14 Y1 N23	out reg[3]|d
0.074	RR	CELL	1	FF X14 Y1 N23	out reg[3]

Timing Analysis Basics

Launch vs. latch edges

Setup & hold times

Data & clock arrival time

Data required time

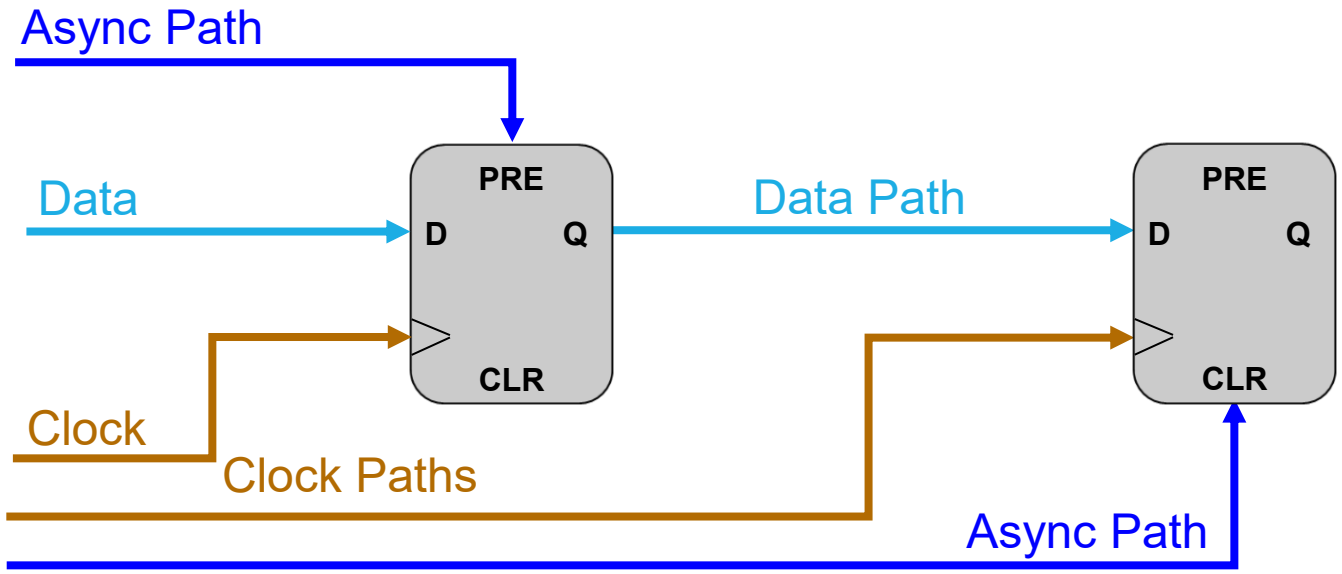
Setup & hold slack analysis

I/O analysis

Recovery & removal

Timing models

Path & Analysis Types



Three types of Paths:

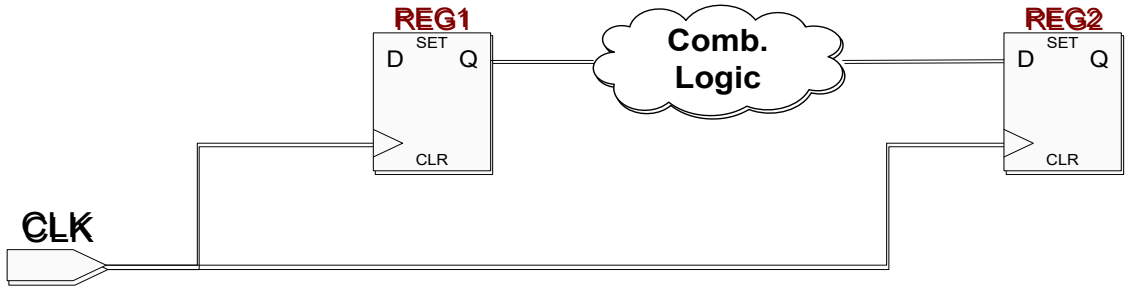
1. Clock Paths
2. Data Path
3. Asynchronous Paths*

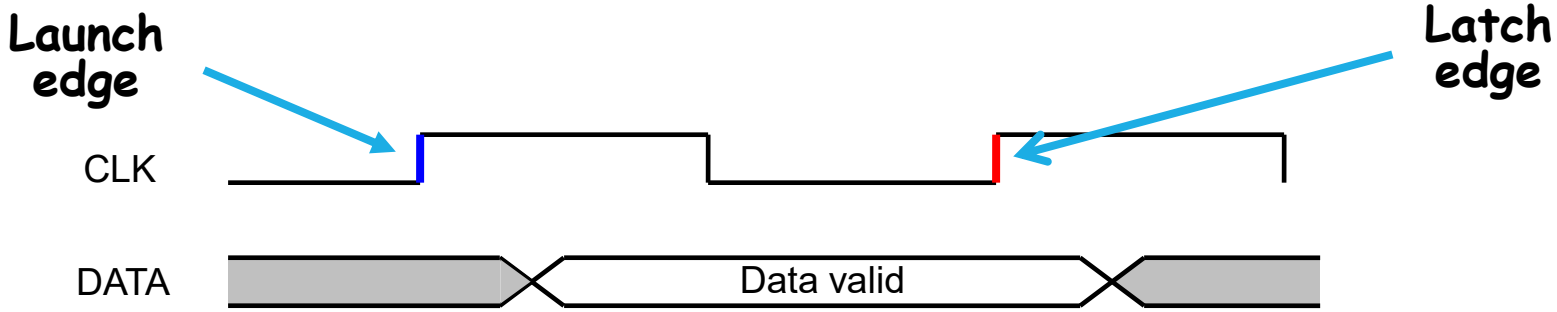
Two types of Analysis:

1. Synchronous – clock & data paths
2. Asynchronous* – clock & async paths

Asynchronous refers to signals feeding the asynchronous control ports of the registers

Launch & Latch Edges

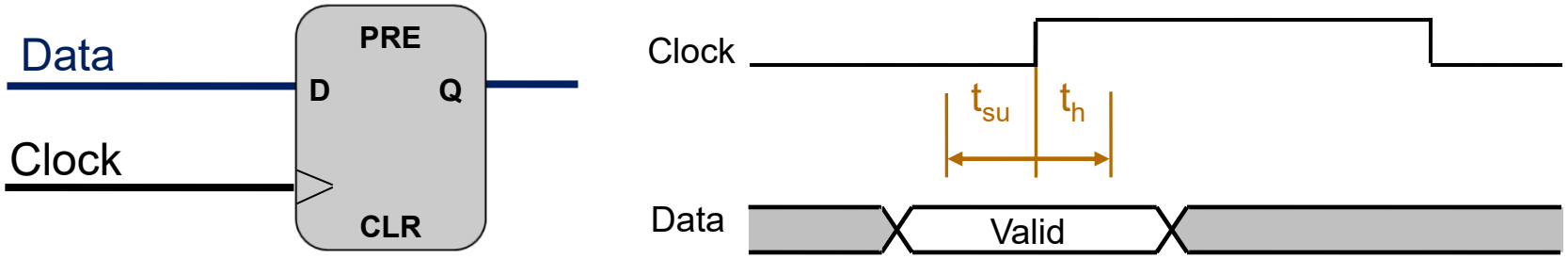




Launch Edge: the edge which "launches" the data from source register

Latch Edge: the edge which "latches" the data at destination register (with respect to the launch edge, typically 1 clock cycle)

Setup & Hold



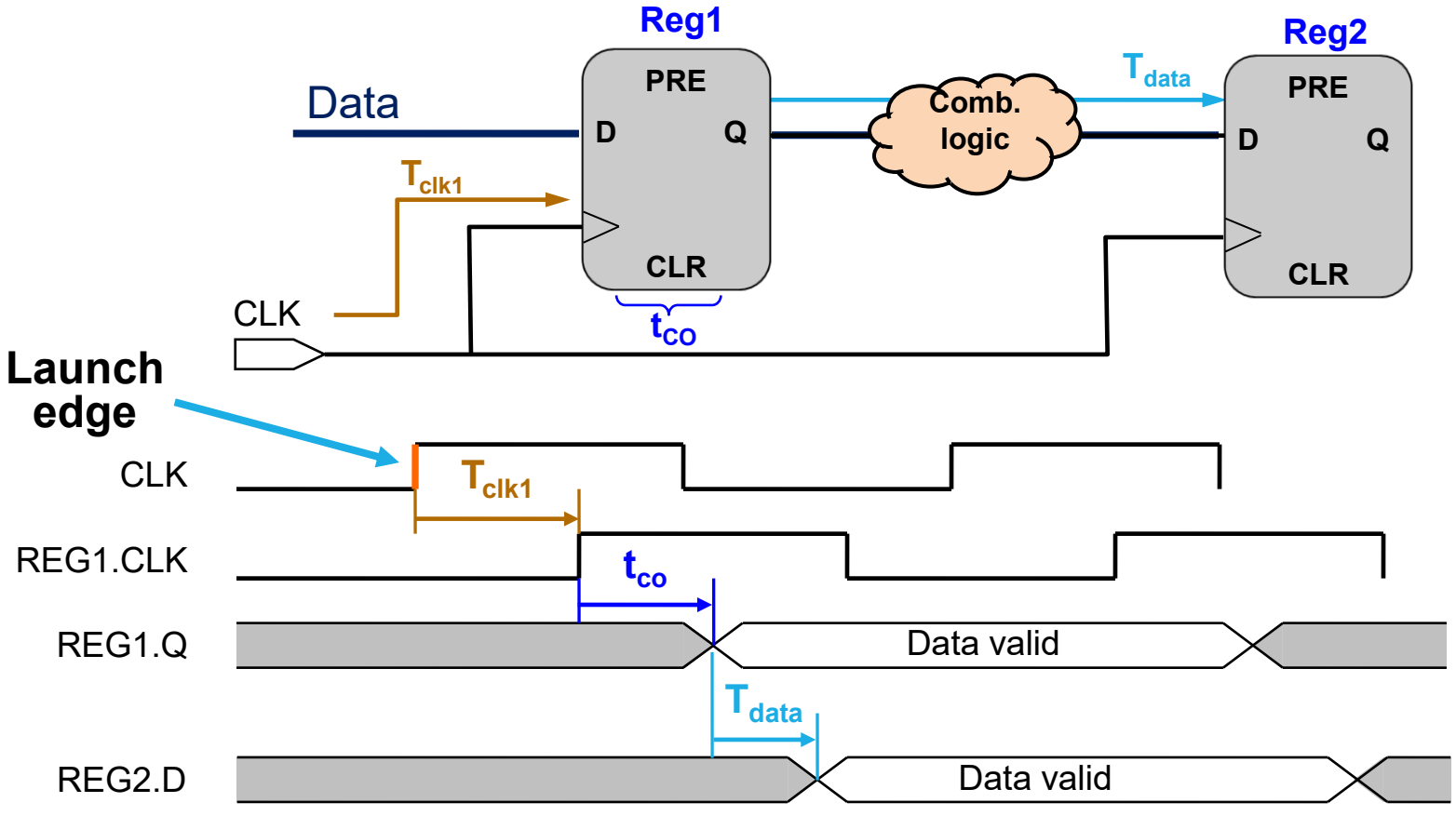
t_{su}: Setup Time: The minimum time data signal must be stable **BEFORE clock edge**

t_h: Hold Time: The minimum time data signal must be stable **AFTER clock edge**

*Together, the setup time and hold time form a **Data Required Window**, the time around a clock edge in which data must be stable*

Data Arrival Time

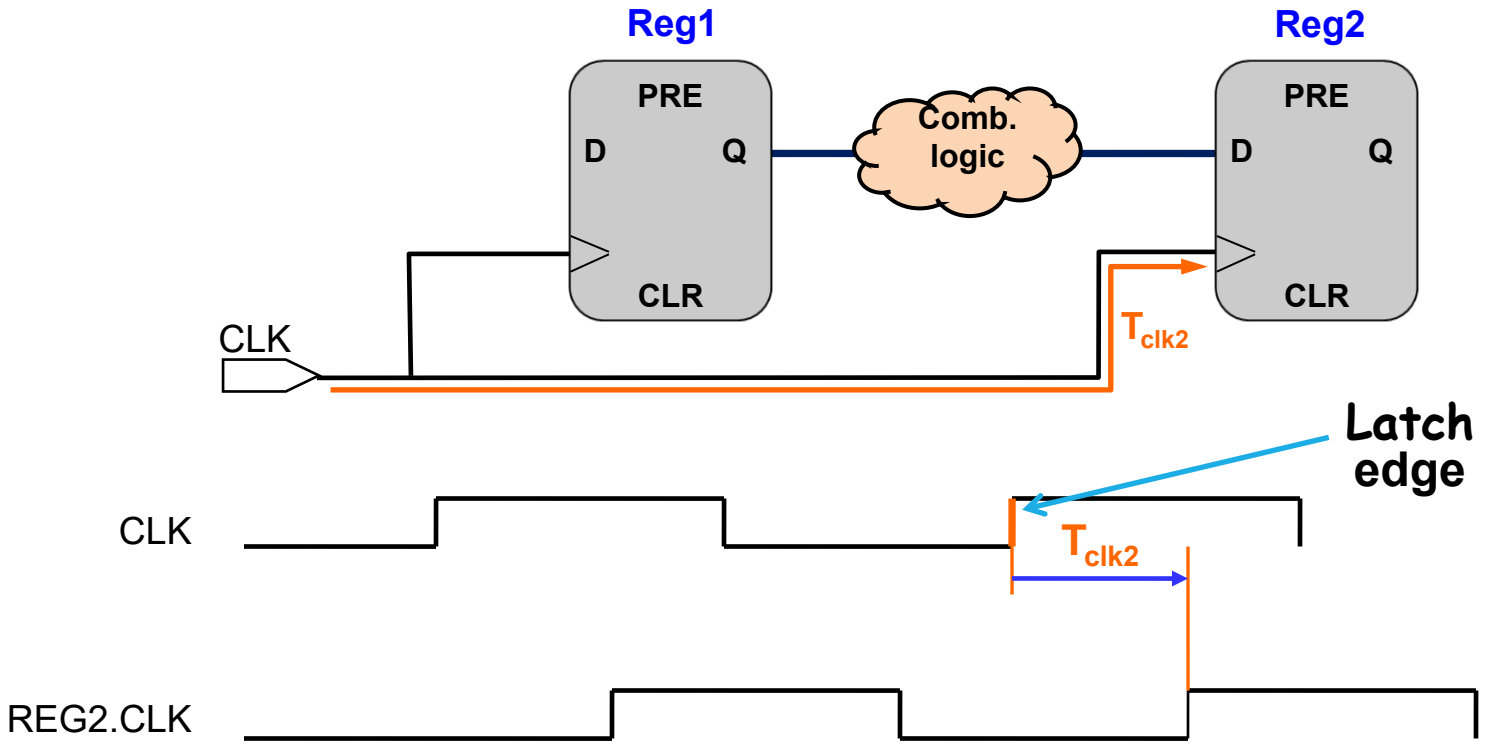
The time for data to arrive at destination register's D input



Data Arrival Time = launch edge + T_{clk1} + t_{co} +T_{data}

Clock Arrival Time

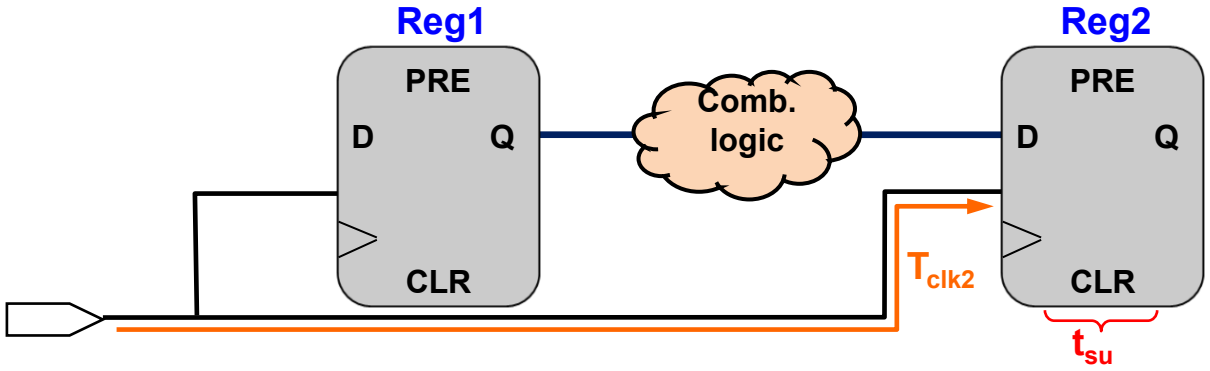
The time for clock to arrive at destination register's clock input

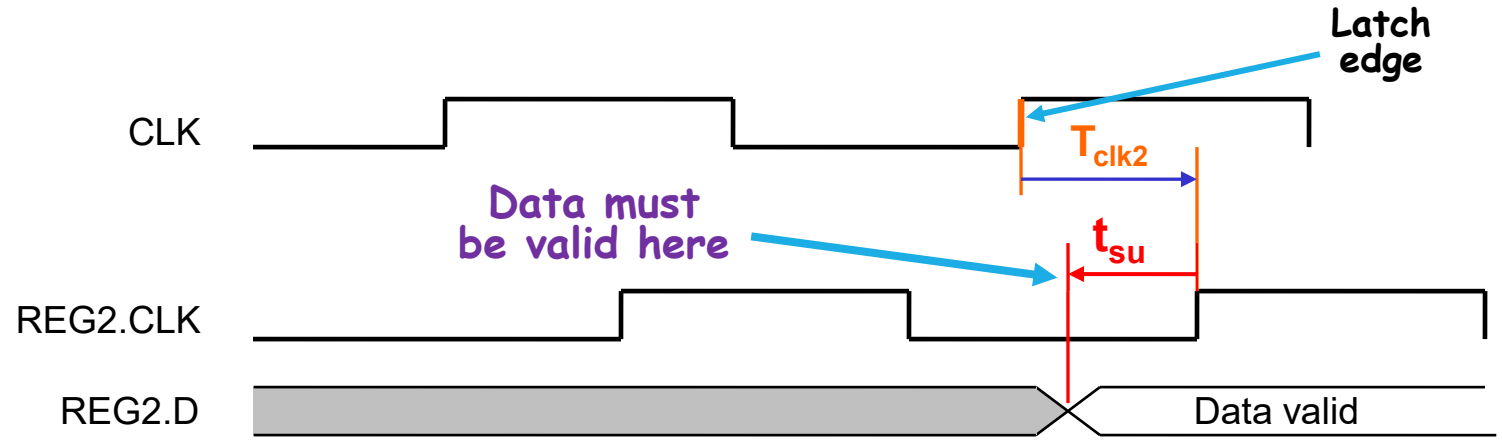


Clock Arrival Time = latch edge + T_{clk2}

Data Required Time (Setup)

The minimum time required before the latch edge for the data to get latched into the destination register

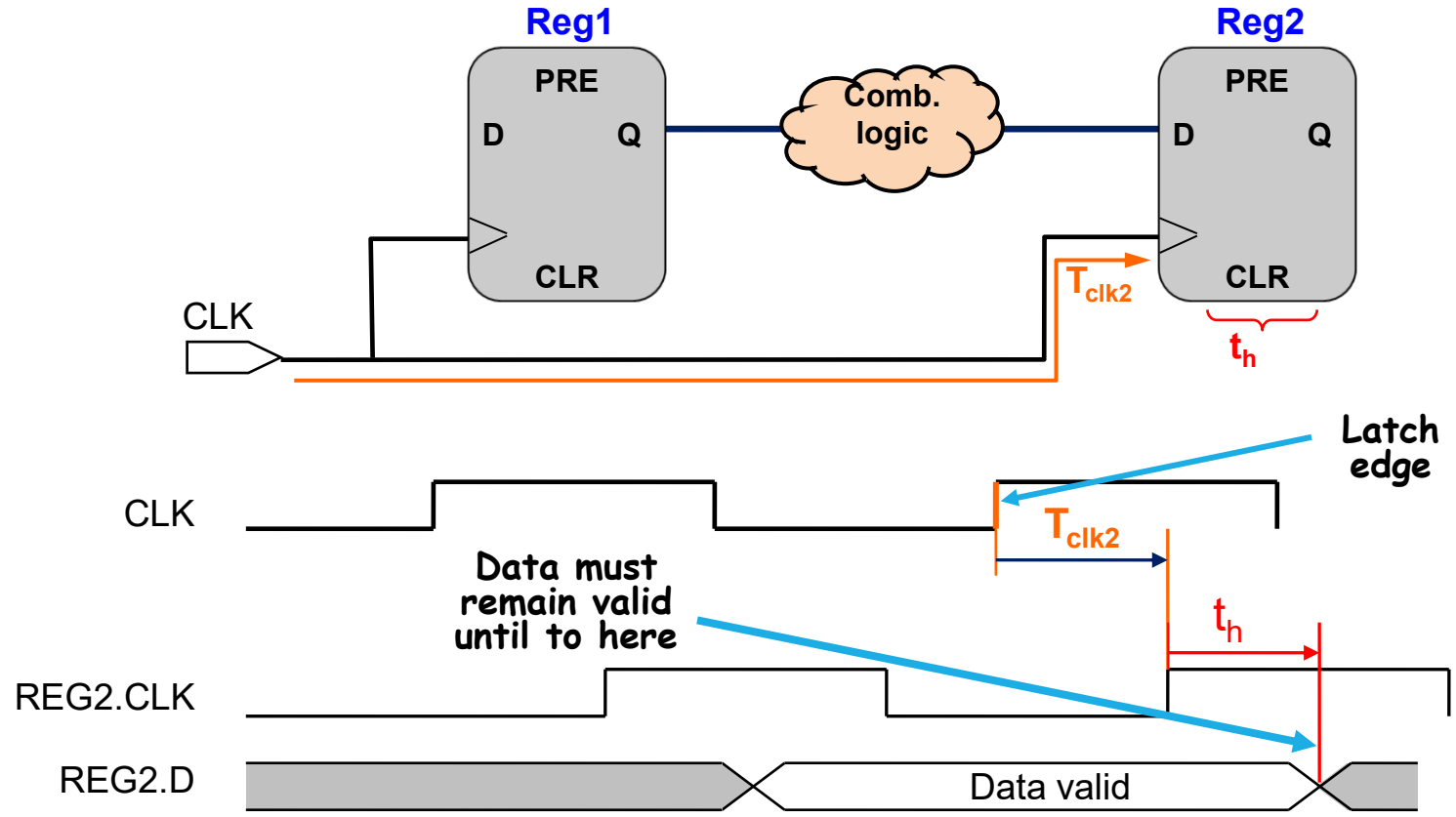




Data Required Time (Setup) = Clock Arrival Time - t_{su} - Setup Uncertainty

Data Required Time (Hold)

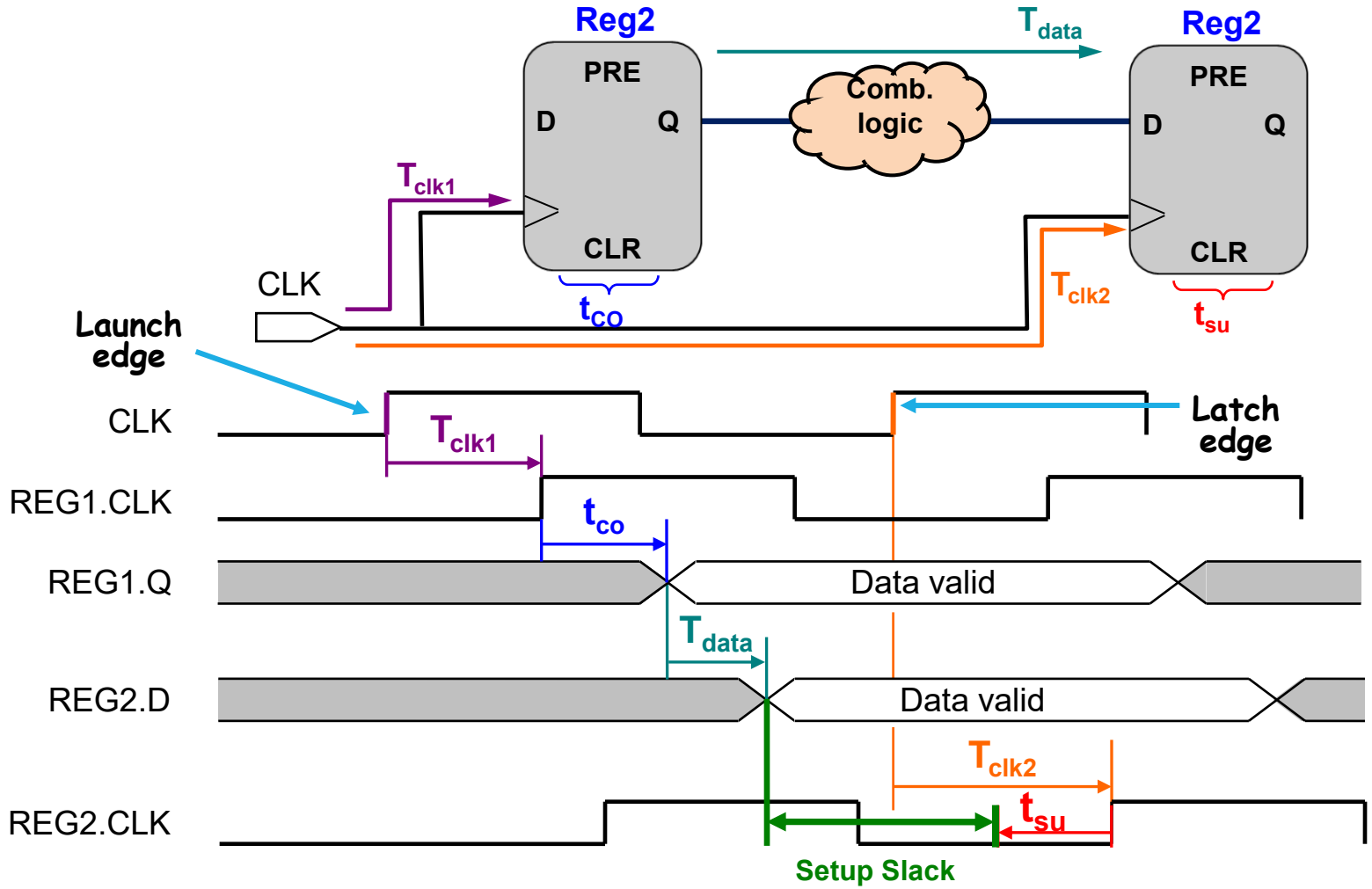
The minimum time required after the latch edge for the data to remain valid for successful latching into the destination register



Data Required Time (Hold) = Clock Arrival Time + t_h + Hold Uncertainty

Setup Slack

The margin by which the setup timing requirement is met. It ensures launched data arrives in time to meet the latching requirement



Setup Slack (cont'd)

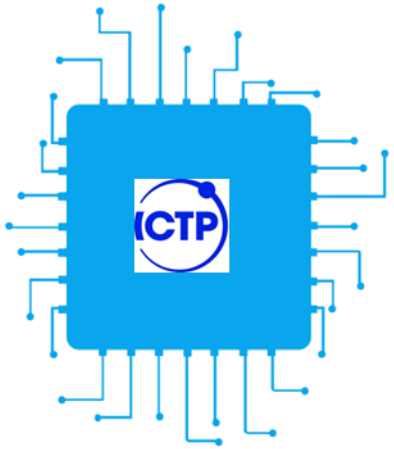
Setup Slack = ***Minimum*** *Data Required Time (Setup)* – ***Maximum*** *Data Arrival Time*

Positive slack

- *Timing requirement met*

Negative slack

- *Timing requirement not met*



Solutions for Timing Issues

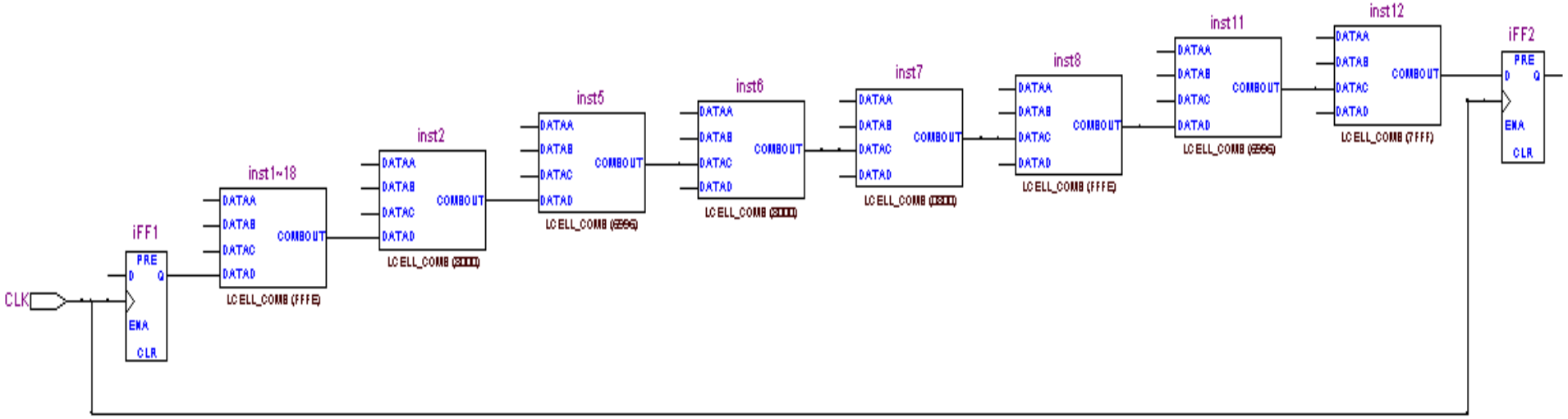
Timing Issues Problems and Solutions

Analysis of the most common timing failures and their possible solutions.

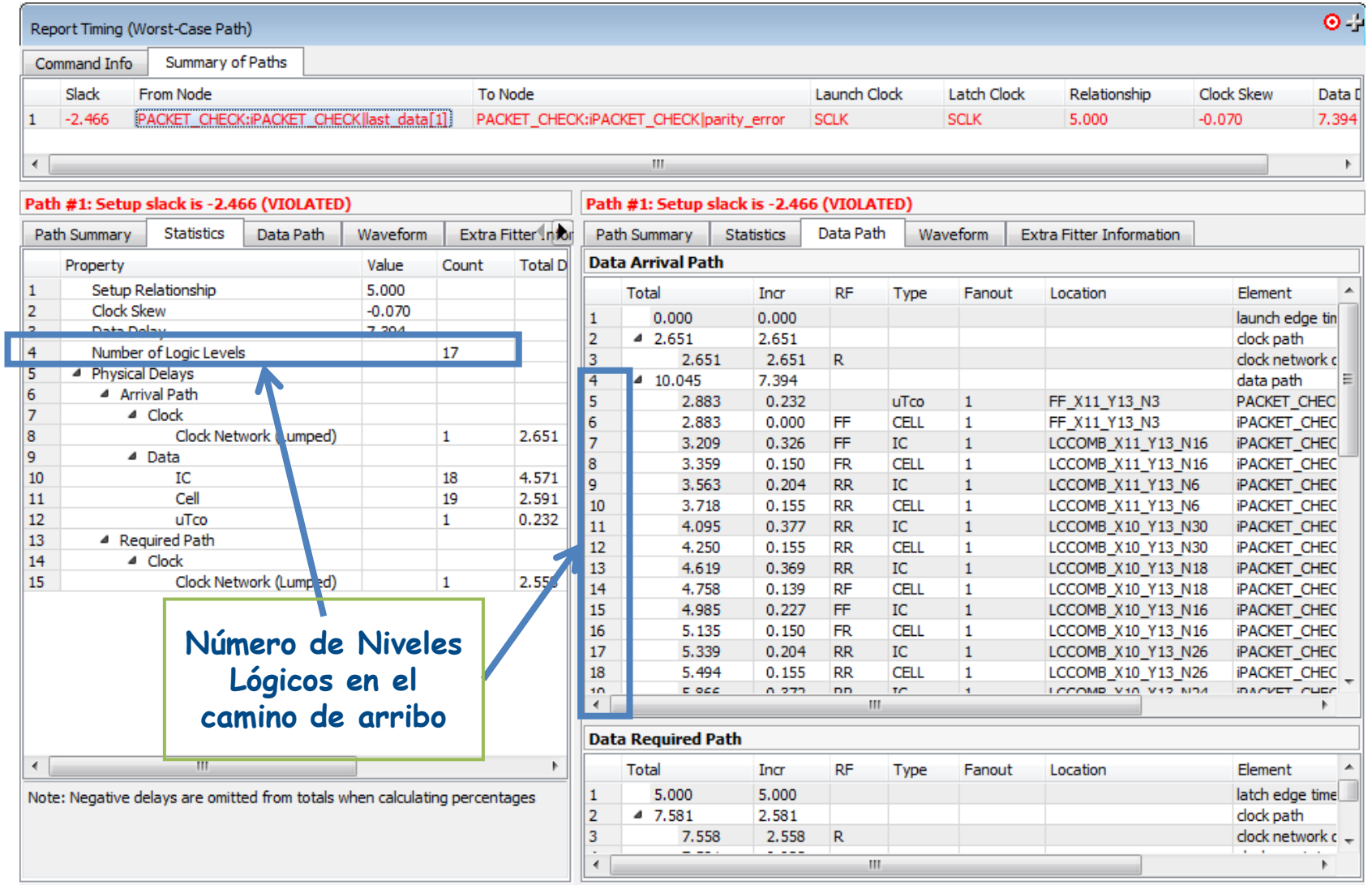
- ✓ *Too many logic levels*
- ✓ *High signal fan-out*
- ✓ *Conflicting timing requirements*
- ✓ *Overly demanding timing requirements*

Case 1 – Too many logic levels

Eight logic levels

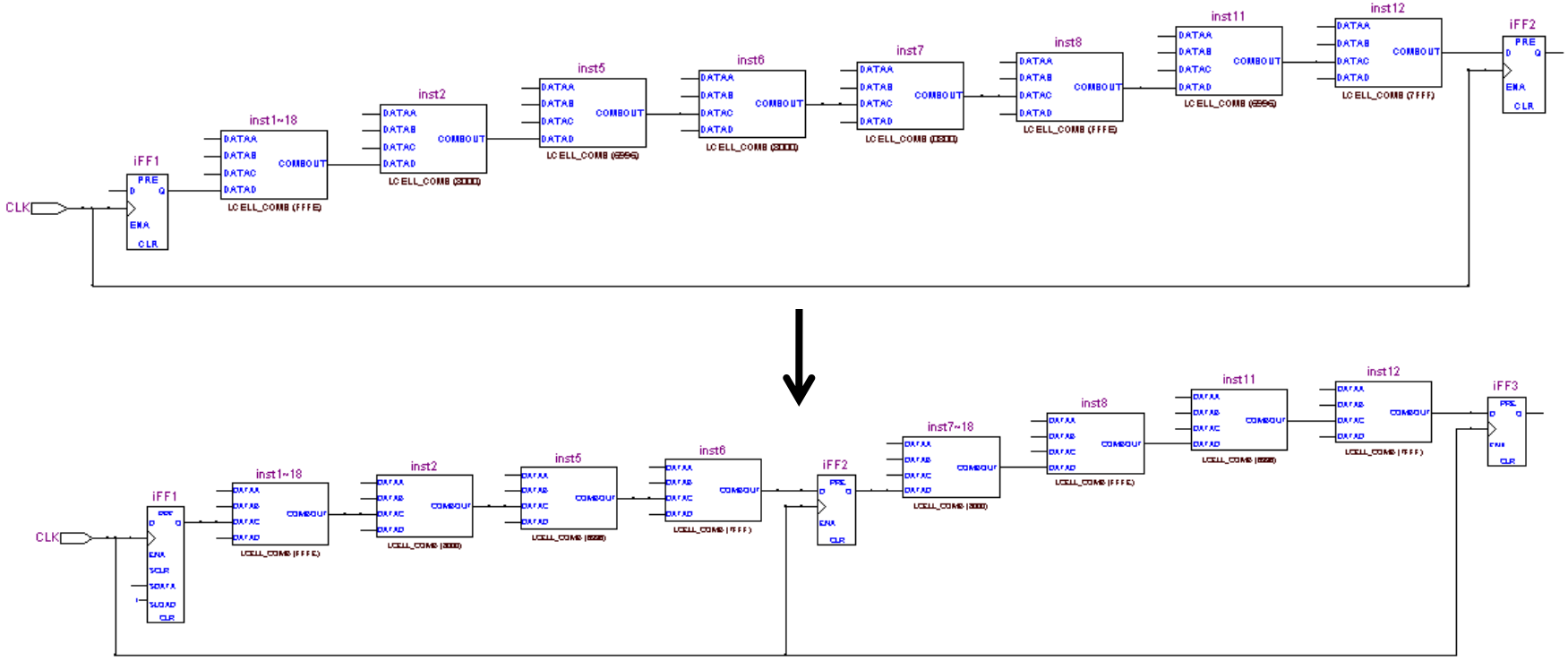


Case 1 – Too many logic levels - Example



Case 1 – Solution to too many logic levels

- Add pipeline registers to reduce T_{data}



Case 1 - Advice

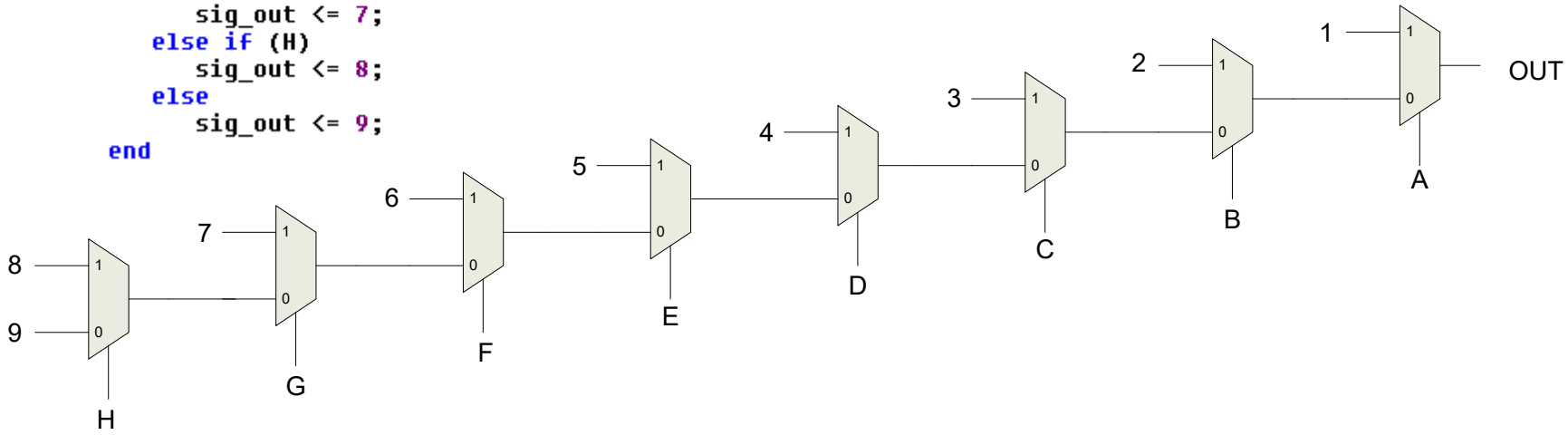
- Do not use if-elsif-endif nested, used case instead

VHDL

```
-- Too many embedded IF statements
process(A, B, C, D, E, F, G, H)
begin
   if A = '1' then
      sig_out <= 1;
   elsif B = '1' then
      sig_out <= 2;
   elsif C = '1' then
      sig_out <= 3;
   elsif D = '1' then
      sig_out <= 4;
   elsif E = '1' then
      sig_out <= 5;
   elsif F = '1' then
      sig_out <= 6;
   elsif G = '1' then
      sig_out <= 7;
   elsif H = '1' then
      sig_out <= 8;
   else
      sig_out <= 9;
   end if;
end process;
```

Verilog

```
// Too many embedded IF statements
always @(*)
begin
   if (A)
      sig_out <= 1;
   else if (B)
      sig_out <= 2;
   else if (C)
      sig_out <= 3;
   else if (D)
      sig_out <= 4;
   else if (E)
      sig_out <= 5;
   else if (F)
      sig_out <= 6;
   else if (G)
      sig_out <= 7;
   else if (H)
      sig_out <= 8;
   else
      sig_out <= 9;
end
```

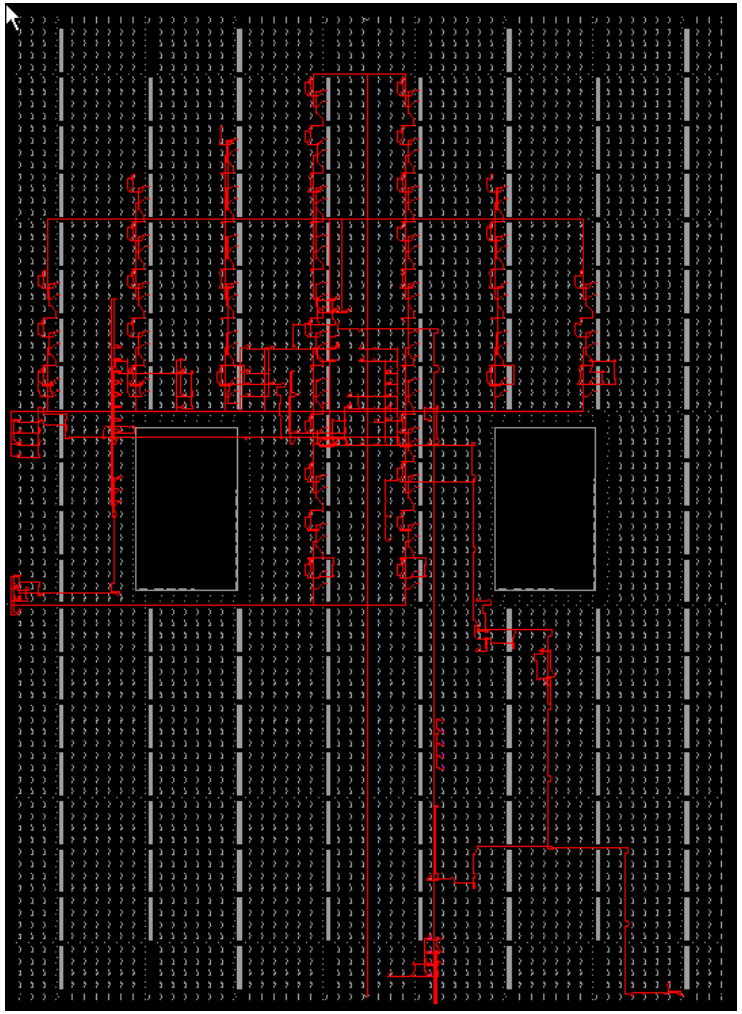


Case 2: High Fan-Out Signals

```
The following nets have been assigned to a chip global resource:
    Fanout  Type           Name
    ----------------------------
    4378    INT_NET        Net    : SysClk
                           Driver: Instance_Clk_M1Rst_Gen/Pll_25Mhz_0/Core
                           Source: ESSENTIAL
    2277    INT_NET        Net    : SysRstN_c_c
                           Driver: Instance_SG_FEDK/CortexM1Top_0/CortexM1Top_II/U_CLKSRC
                           Source: NETLIST
    1861    SET/RESET_NET  Net    : Instance_SG_FEDK/CMACIO1II_RNI964
                           Driver:
Instance_SG_FEDK/CORE10100_AHBAPB_0/CMACOOO10I/CMACII11I/CMACIO1II_RNI964/U_CLKSRC
                           Source: NETLIST
    776     INT_NET        Net    : Instance_SG_FEDK/CortexM1Top_0/DBGRESETn
                           Driver:
Instance_SG_FEDK/CortexM1Top_0/genblk1085.genblk1087.CortexM1Top_O1/U_CLKSRC
                           Source: NETLIST
    322     CLK_NET        Net    : Instance_SG_FEDK/CLKINT_1_Y
                           Driver: Instance_SG_FEDK/CLKINT_1/U_CLKSRC
                           Source: NETLIST
    273     CLK_NET        Net    : Instance_SG_FEDK/CLKINT_0_Y
                           Driver: Instance_SG_FEDK/CLKINT_0/U_CLKSRC
                           Source: NETLIST
```

Clock Net	Resource	Locked	Fanout	Net Skew(ns)	Max Delay(ns)
fpga_clk_250	BUFGMUX3S	No	126	0.280	1.410
baud_gen/rs232_clk_i	BUFGMUX7S	No	72	0.307	1.410
sys_clk_100_BUFGP	BUFGMUXOP	No	34	0.210	1.322

Case 2: High Fan-Out Signals



Case 2: High Fan-Out Signals

Report Timing (Worst-Case Path)

Command Info | Summary of Paths

	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
1	-1.716	sig	my_regv[1095]	clk	clk	5.000	4.083	9.667

Path #1: Setup slack is -1.716 (VIOLATED)

Path Summary | Statistics | Data Path | Waveform | Extra Fitter Information

Data Arrival Path

	Total	Incr	RF	Type	Fanout	Location	Element
1	0.000	0.000					launch edge time
2	0.000	0.000					clock path
3	0.000	0.000	R				clock network delay
4	1.000	1.000	R	iExt	1	PIN_T34	sig
5	10.667	9.667					data path
6	1.000	0.000	RR	IC	1	IOIBUF_X0_Y55_N32	sig~input|i
7	1.782	0.782	RR	CELL	4001	IOIBUF_X0_Y55_N32	sig~input|o
8	9.945	8.163	RR	IC	1	MLABCELL_X92_Y47_N38	my_regv[1095]~feeder|datab
9	10.539	0.594	RR	CELL	1	MLABCELL_X92_Y47_N38	my_regv[1095]~feeder|combout
10	10.539	0.000	RR	IC	1	FF_X92_Y47_N39	my_regv[1095]|d
11	10.667	0.128	RR	CELL	1	FF_X92_Y47_N39	my_regv[1095]

Fanout de 4001 retardo de ruteo de 8.163 ns

Data Required Path

	Total	Incr	RF	Type	Fanout	Location	Element
1	5.000	5.000					latch edge time
2	9.083	4.083					clock path
4	8.951	-0.132		uTsu	1	FF_X92_Y47_N39	my_regv[1095]

Case 2: High Fan-Out Signals – Main Problem

In FPGAs, a signal with **high fan-out** refers to a net (like a clock, reset, or data signal) that drives a large number of downstream logic elements, such as flip-flops or gates.

This can lead to issues like:

- ✓ excessive capacitive loading,
- ✓ increased propagation delays,
- ✓ signal skew,
- ✓ power consumption spikes,
- ✓ timing violations (e.g., setup/hold time failures), which degrade performance or prevent the design from meeting clock frequency targets.

Case 2: High Fan-Out Signals – Solution

- ✓ Register/Logic duplication
- ✓ Pipelining
- ✓ Utilize Global or Low-Skew Resources
- ✓ Buffering and Routing Optimization

Case 3: Floorplaning

A graphical tool used to display and edit the design within the FPGA. Poor floorplanning can decrease design performance, just as good floorplanning can improve it.

Uses

- ✓ Increase productivity and performance
- ✓ View the design placed on the FPGA
- ✓ Make placement modifications
- ✓ Create placement areas and groups

Case 3: Floorplaning

Before using floorplaning, understand the following:

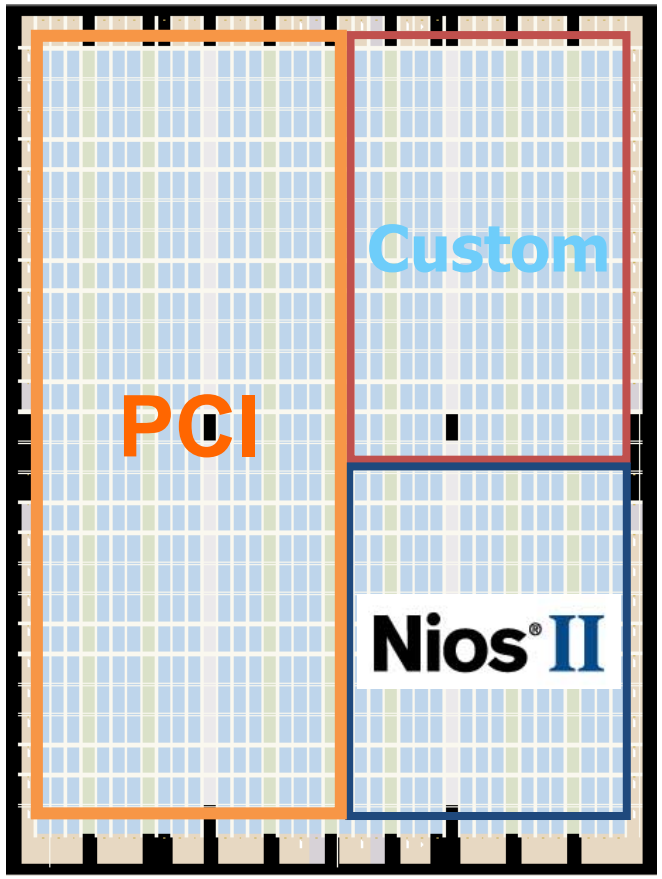
- ✓ The whole design
- ✓ The FPGA architecture
- ✓ The software you use

Before using floorplanner, consider:

- ✓ Time constraints
- ✓ Increasing P&R effort
- ✓ Using pipelines on critical paths
- ✓ Using re-entrant or multi-pass P&R routing

Case 3: Floorplanning – Physical Partition

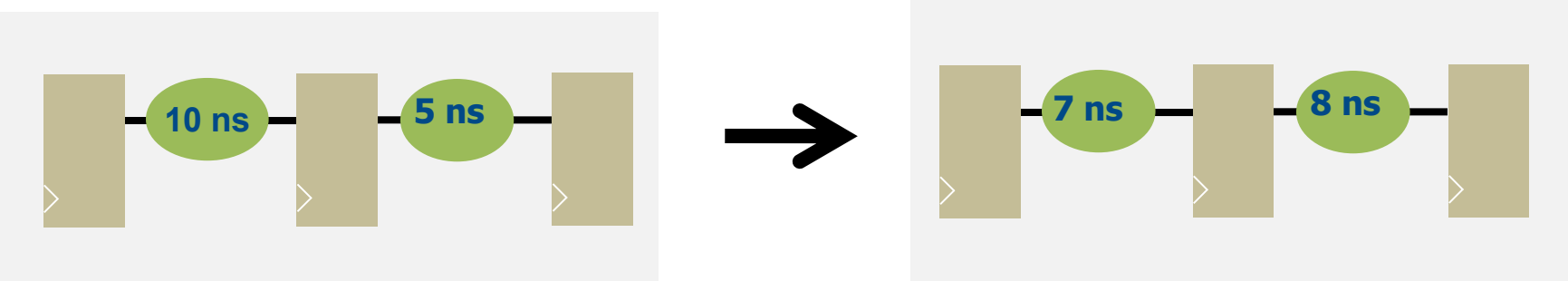
Assignment of logical function to specific area in the FPGA



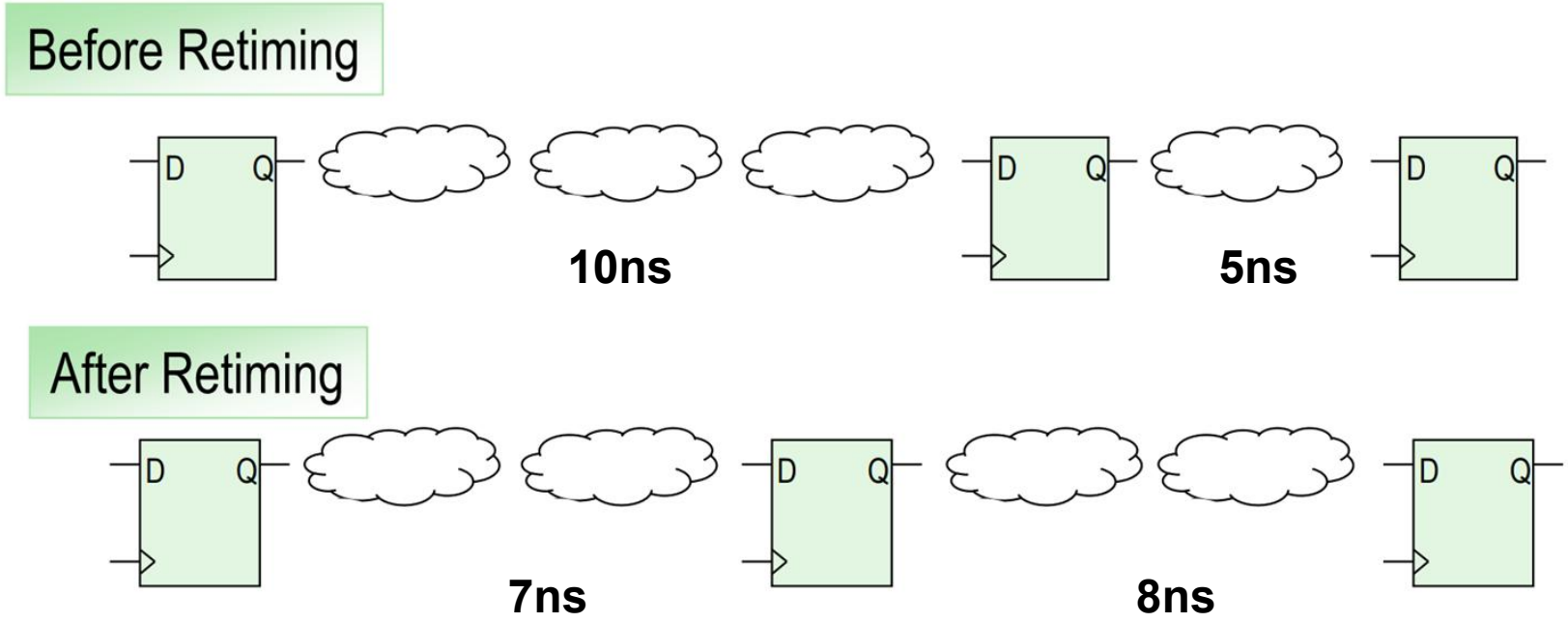
Case 4 : Retiming

Retiming is a technique to improve throughput and latency by increasing **F**, with the possibility of an area change

- Moves registers through combinational logic to balance timing.
- Swaps critical and non-critical paths.
- Does not change the logic's functionality.



Case 4 : Retiming



Case 5 : Resource Sharing

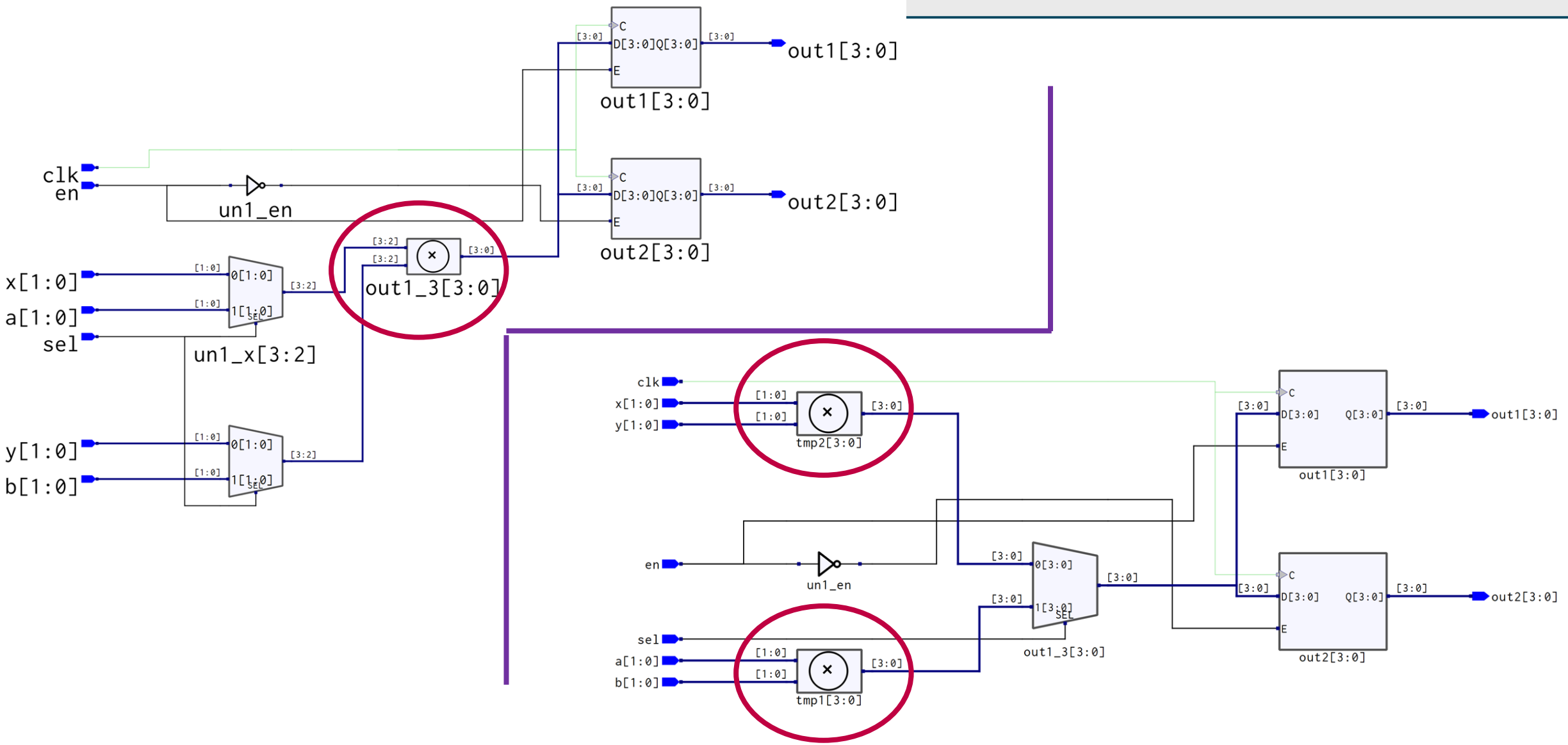
Resource sharing is a technique to trade frequency and latency for area.

Resource sharing is when a single hardware resource (typically area-expensive) is used to implement several operations in the design.

What kind of operations can share hardware?

- Addition, Subtraction, Comparisons.
- Multiplication, Division

Case 4 : Resource Sharing



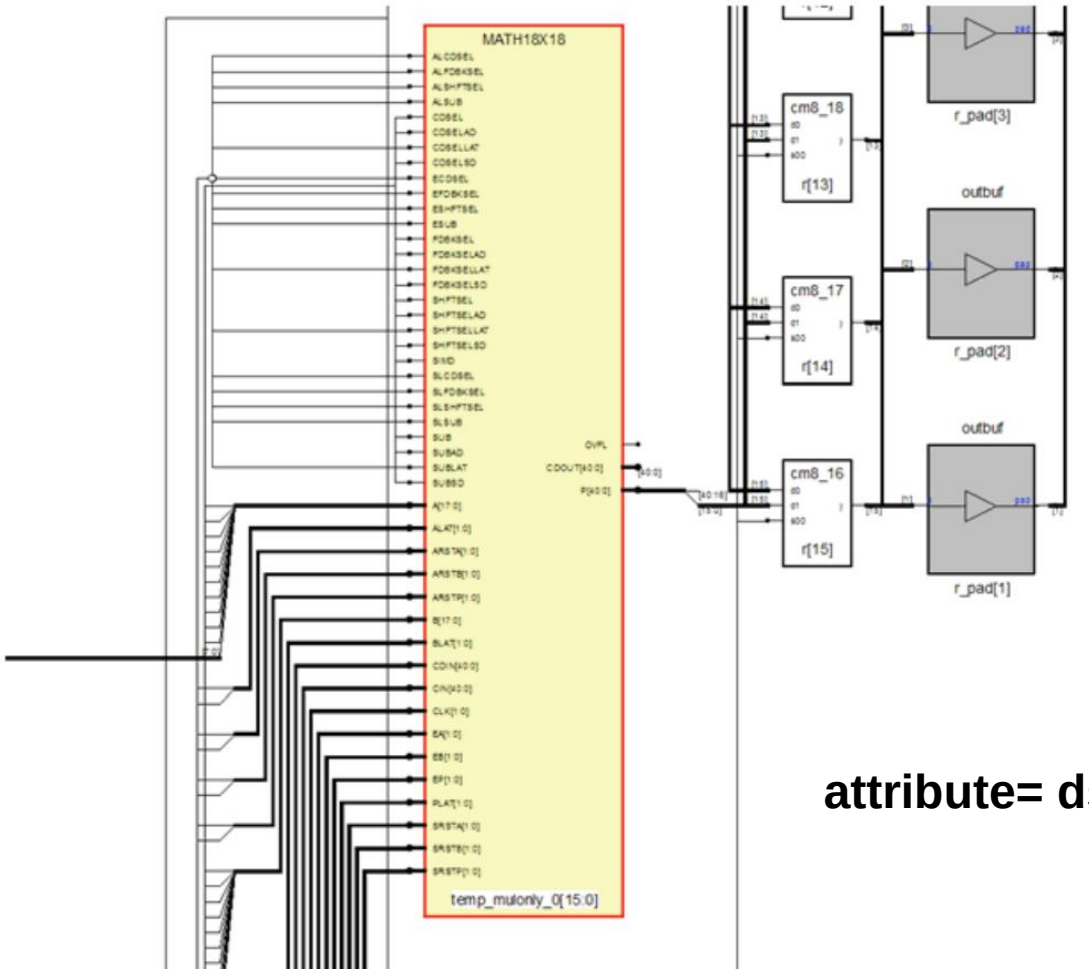
Case 6: Synthesis Attributes

Synthesis attributes can be used to make the synthesis result more efficient in area or in performance.

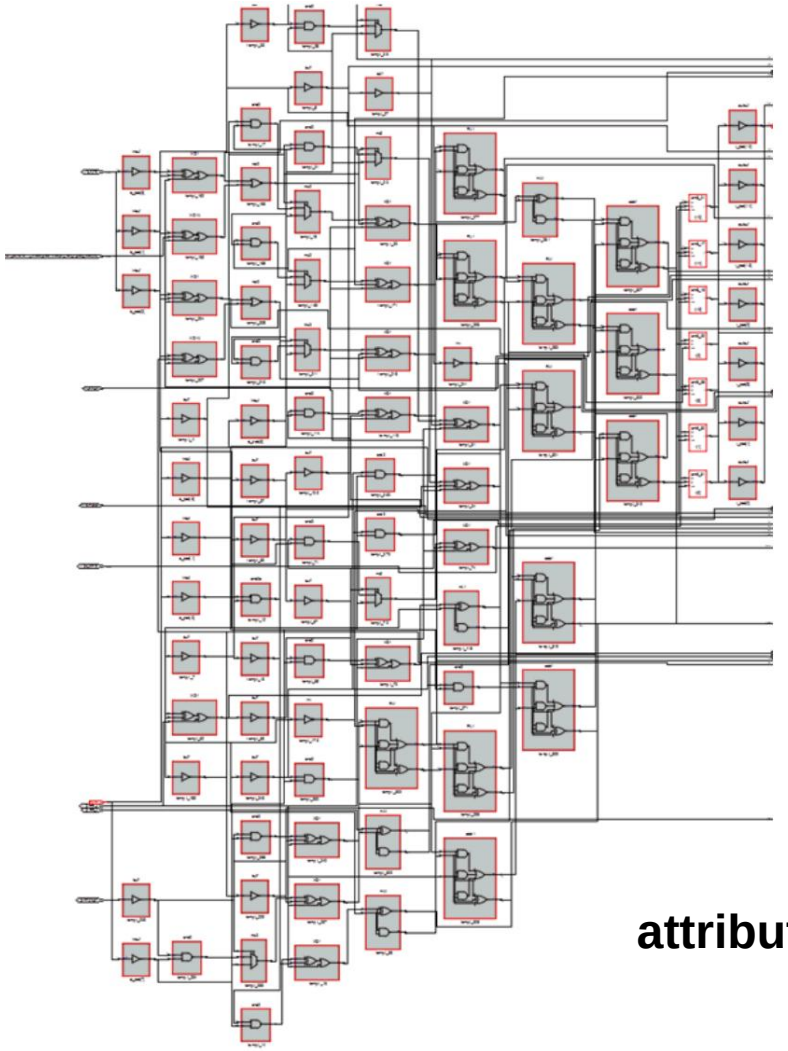
Here, there is an example using synthesis attributes to reduce area, therefore, increasing performance.

```
library ieee;
use ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

entity mult is
port (clk : in std_logic;
    a : in std_logic_vector(7 downto 0);
    b : in std_logic_vector(7 downto 0);
    c : out std_logic_vector(15 downto 0))
end mults;

architecture rtl of mult is
 signal mult_i : std_logic_vector(15 downto 0);
 attribute syn_multstyle : string;
 attribute syn_multstyle of mult_i : signal is "logic";

 begin
    mult_i <= std_logic_vector(unsigned(a)*unsigned(b));
 process(clk)
 begin
    if (clk'event and clk = '1') then
        c <= mult_i;
    end if;
 end process;
end rtl;
```

Case 6: Synthesis Attributes

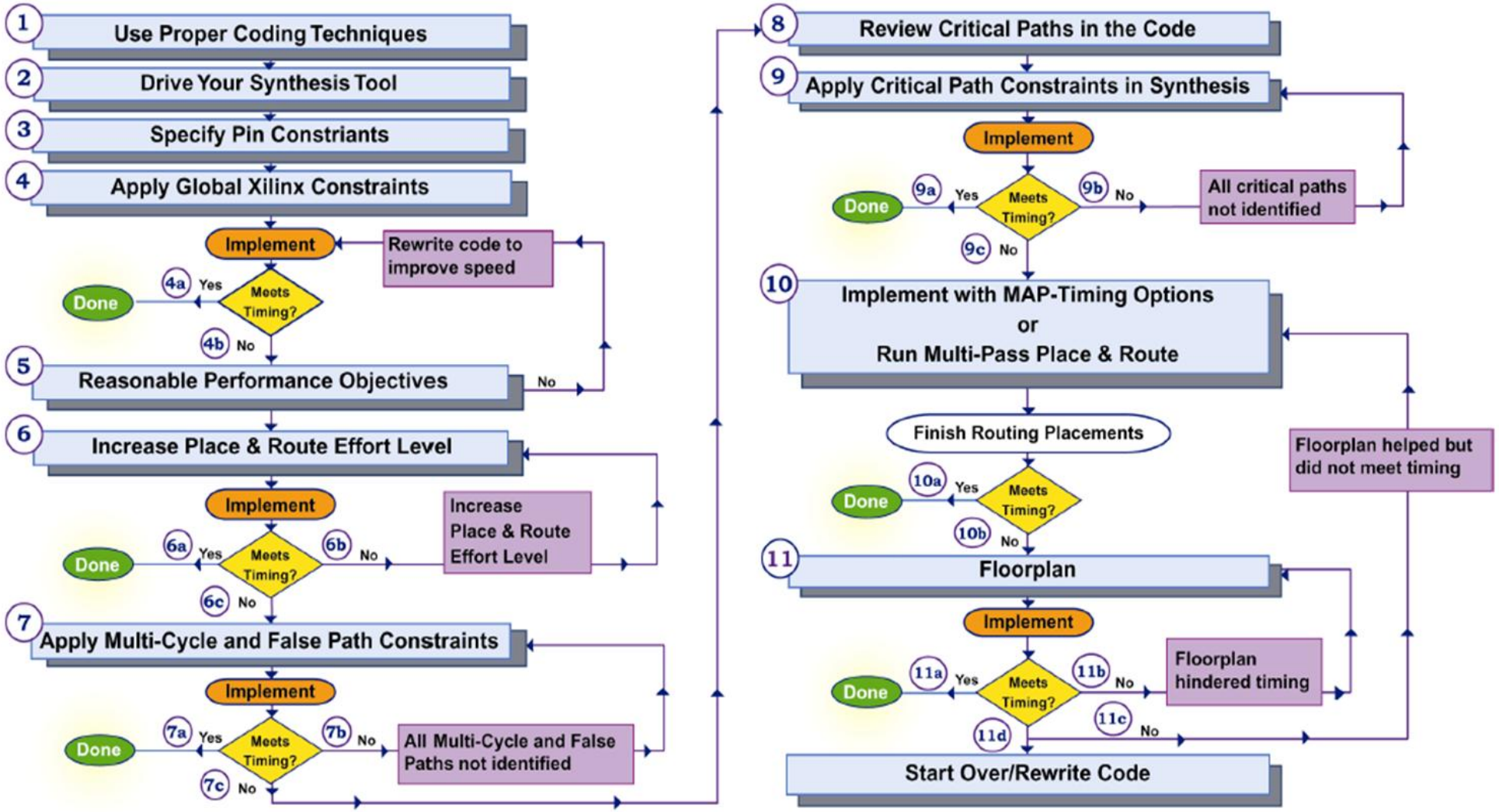


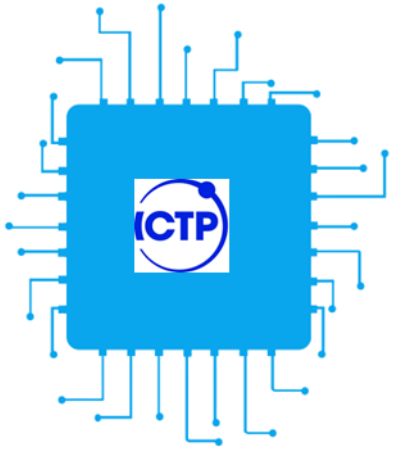
attribute= dsp



attribute= logic

Timing Clousure Flow

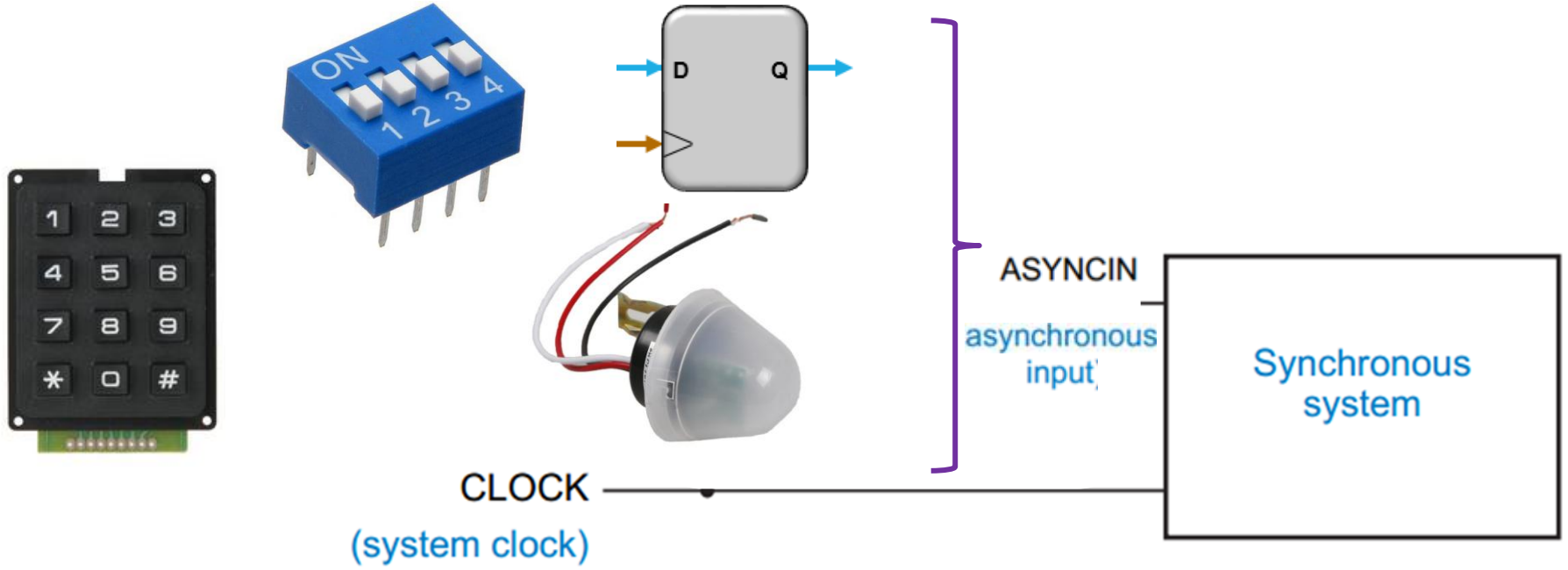




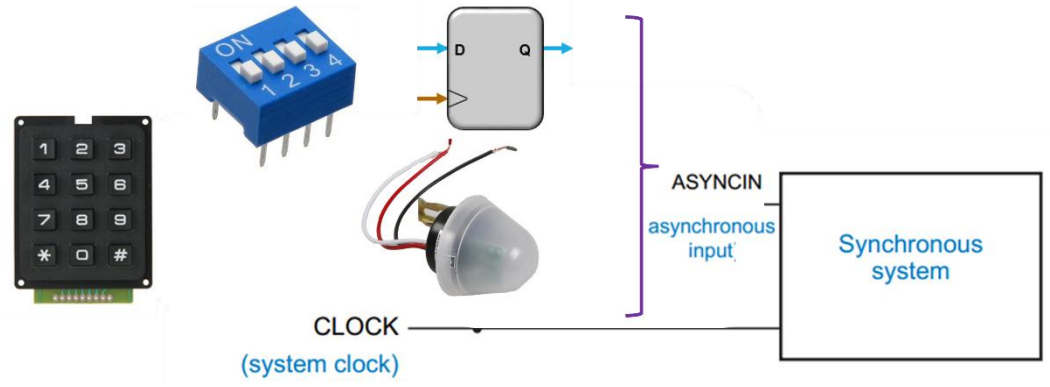
Systems with Different Clock Domains

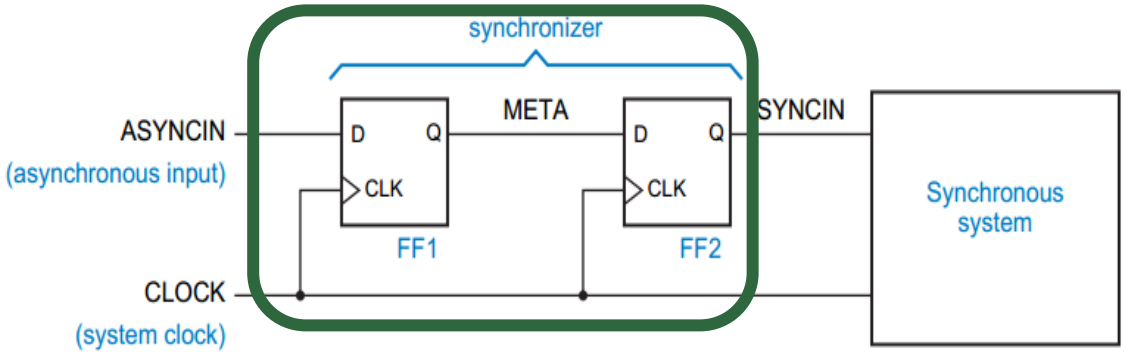
Asynchronous Inputs

- Digital systems of all types inevitably have to deal with asynchronous input signals that are not synchronized with the system clock.
- Asynchronous inputs can come from various sources, e.g.: keyboard, key, push button, sensor, output of another IC (with a different clock signal), etc.



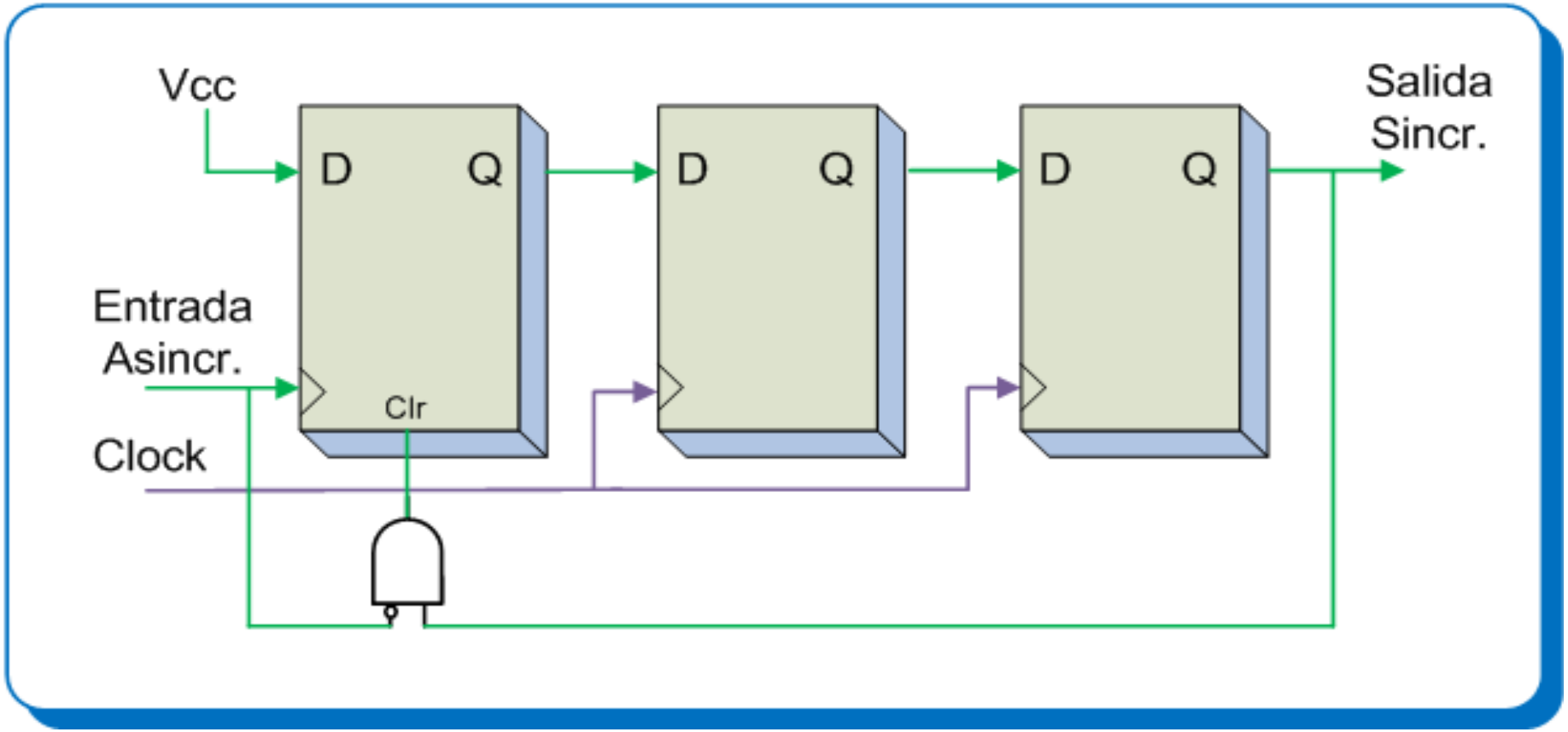
Synchronizer in VHDL





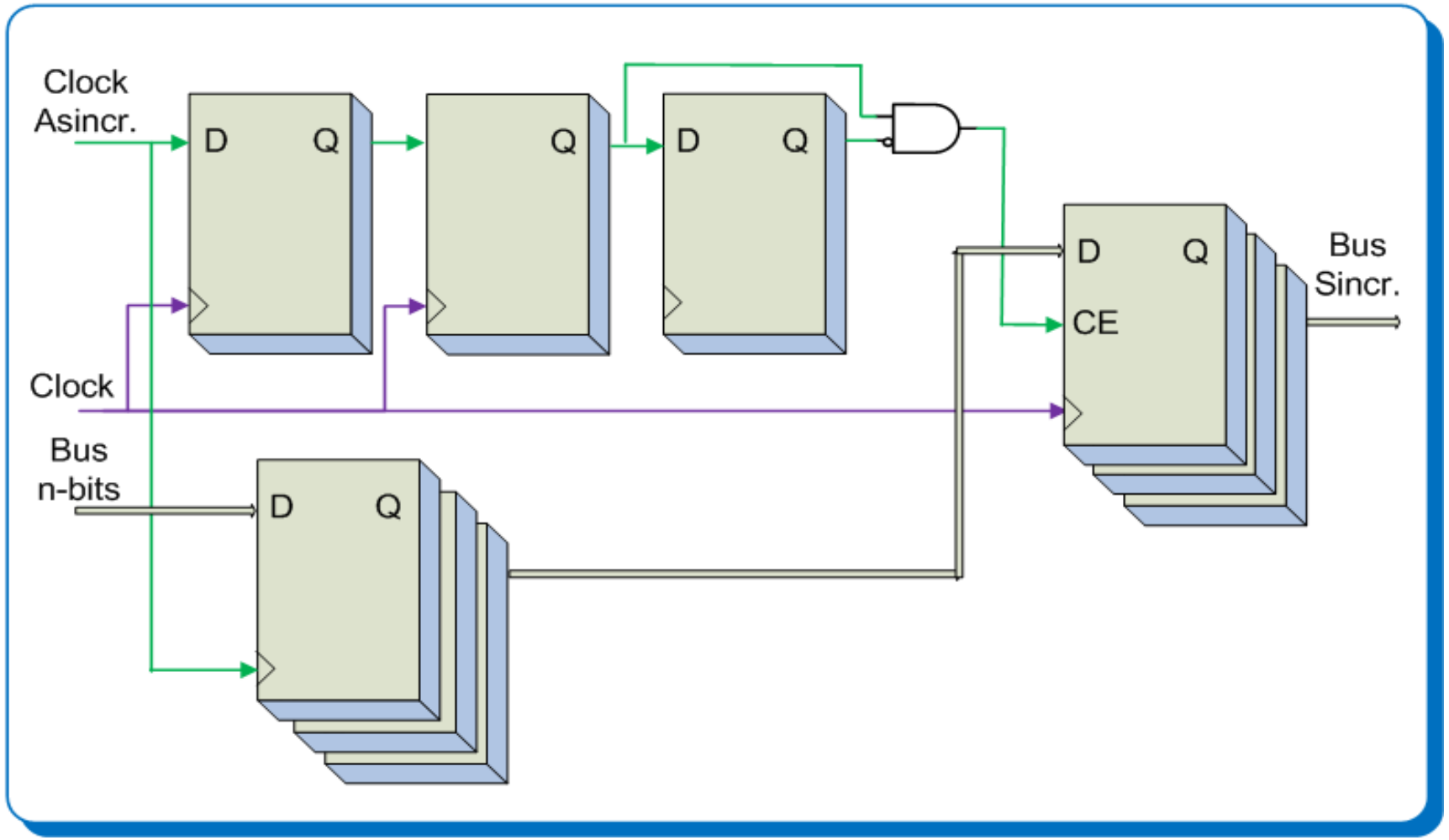

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity synchronizer is
   port(
        clk       : in  std_logic;
        asyncin   : in  std_logic;
        syncin    : out std_logic);
end synchronizer;

architecture behave of synchronizer is

  signal meta: std_logic;
 begin
 sync_proc: process(clk)
 begin
     if (rising_edge(clk)) then
         meta   <= asyncin;
         syncin <= meta;
     end if;
  end process;
 end behave;
```

Synchronizer for Narrow Input Signal



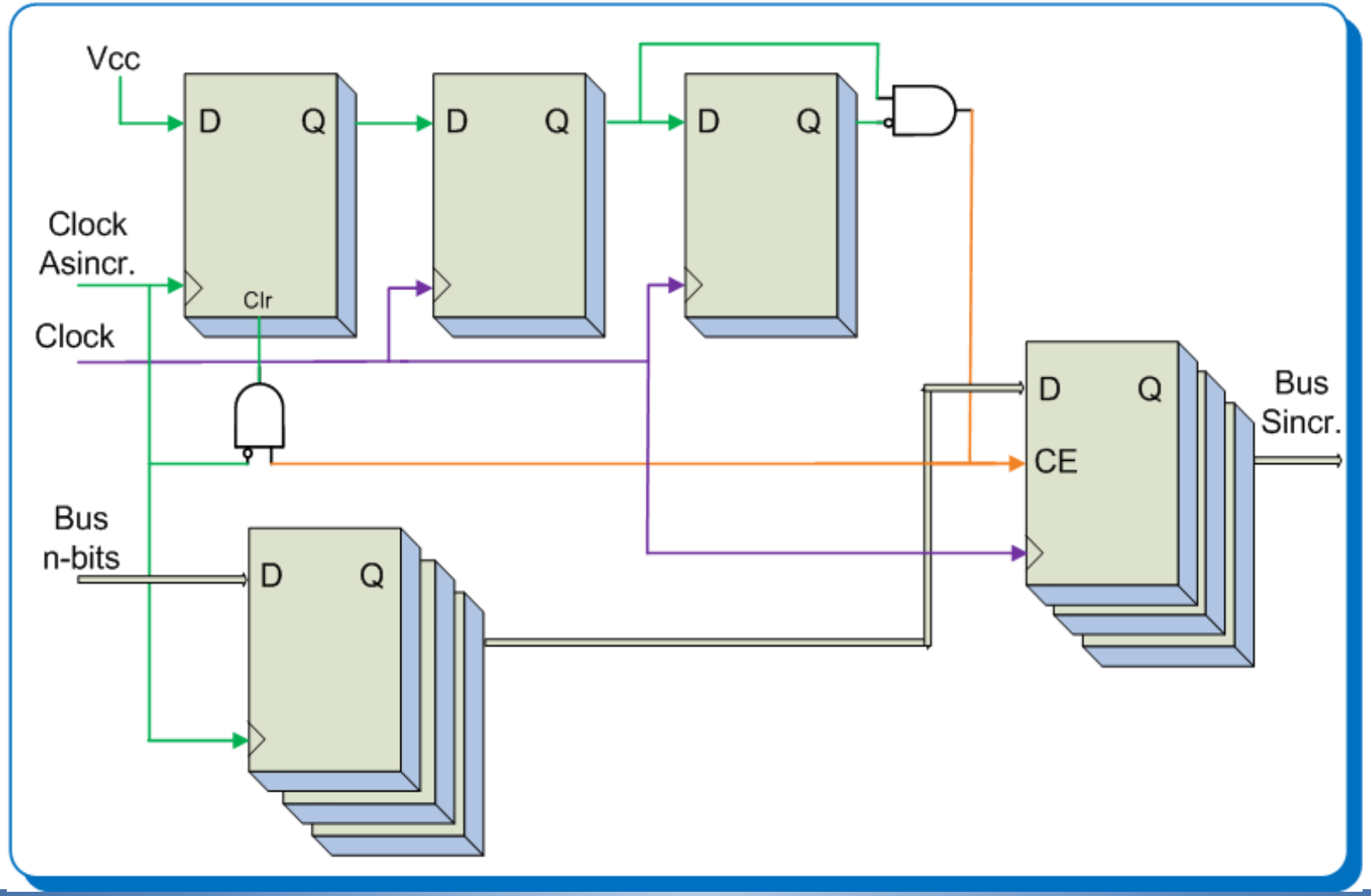
Bus Synchronizer

- From a low frequency to a high frequency

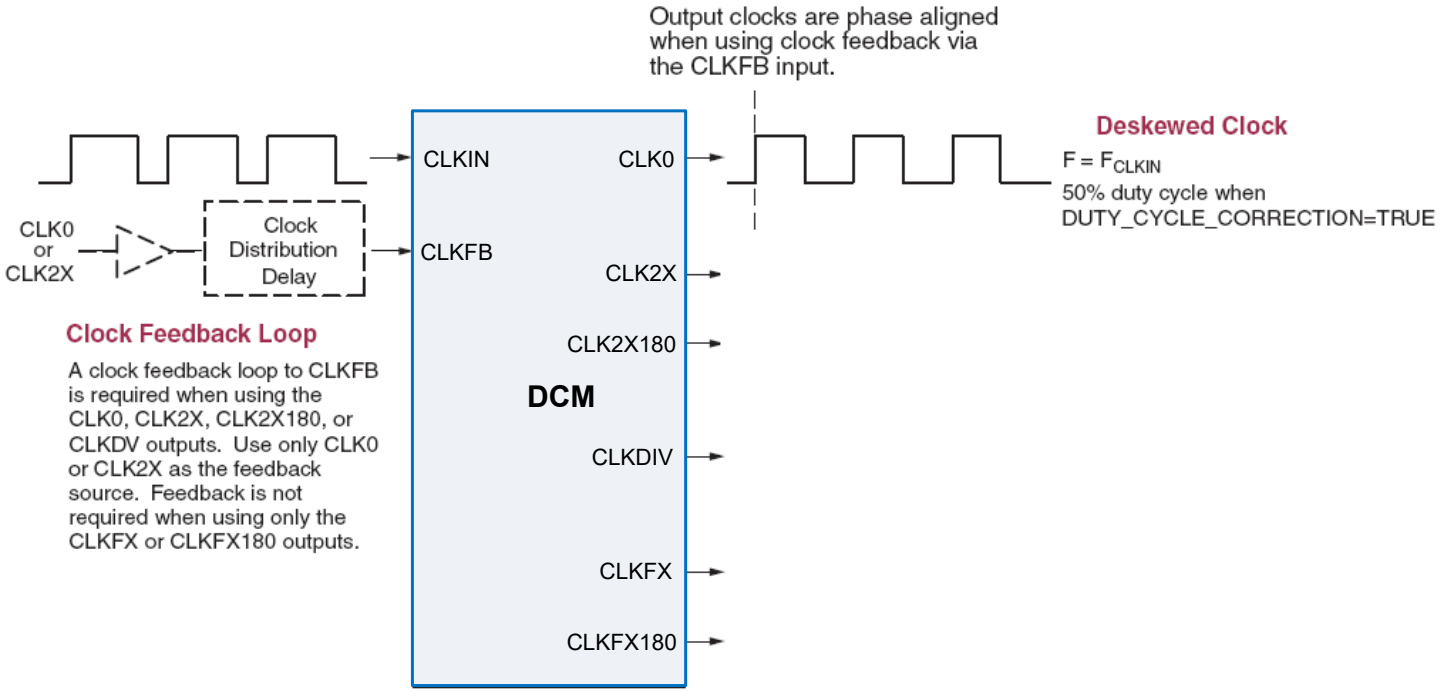


Bus Synchronizer

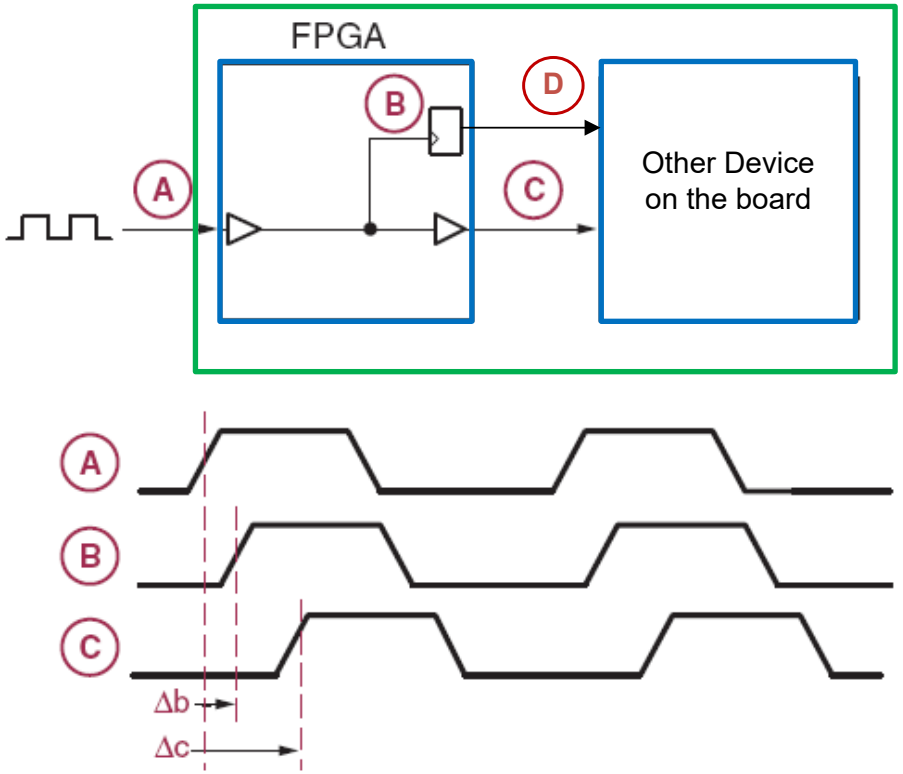
- From a high frequency to a low frequency

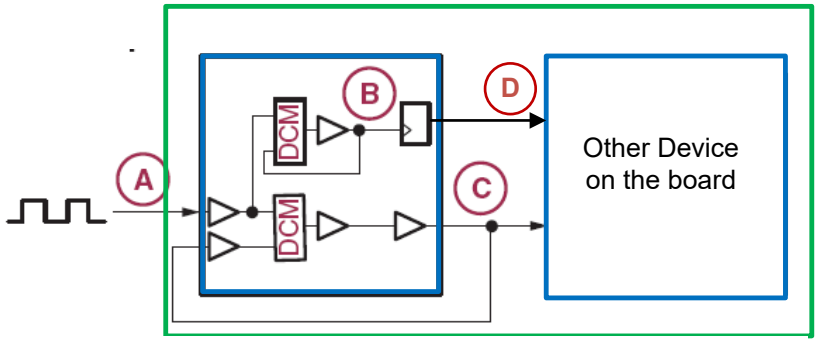


Digital Clock Management (DCM)

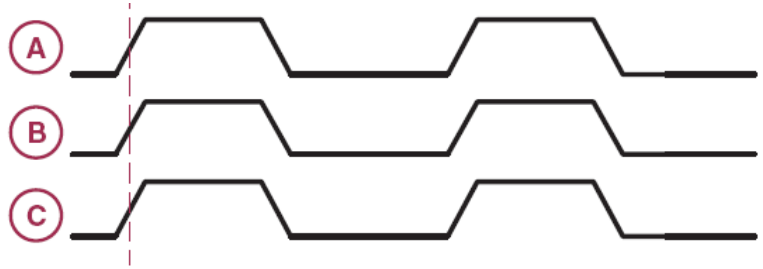


Clock Skew (delay) Elimination

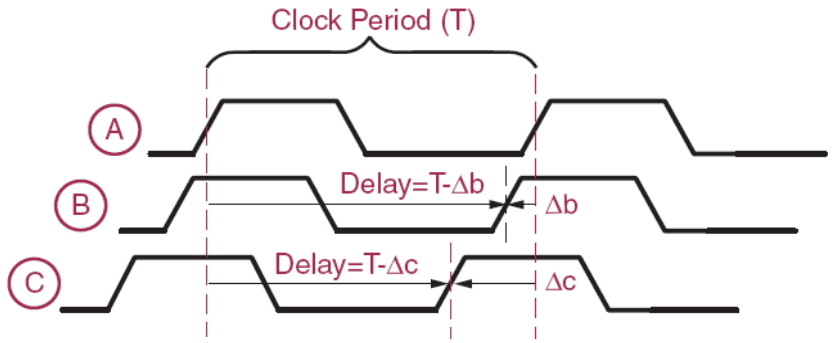




A) Esquema de configuración de los DCMs

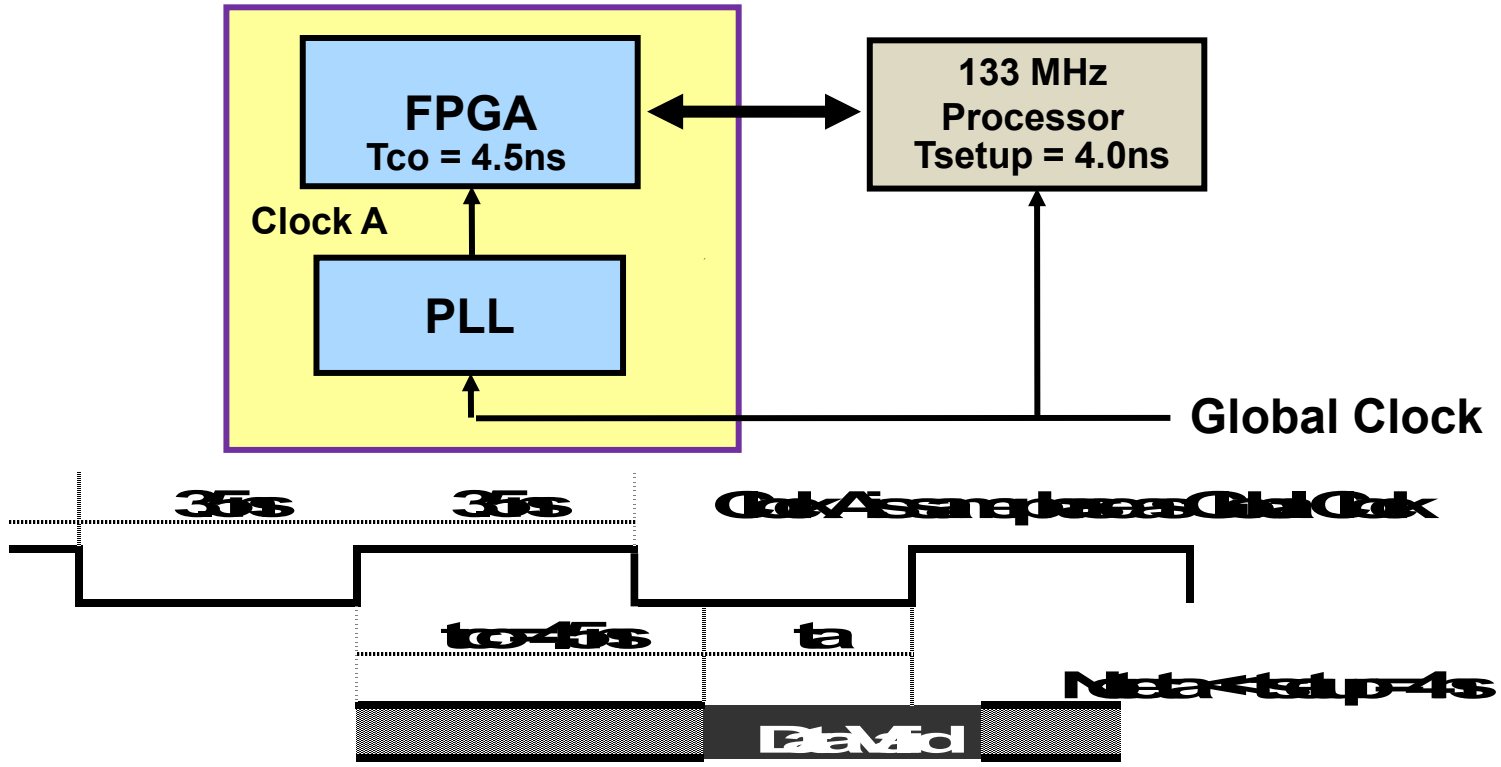


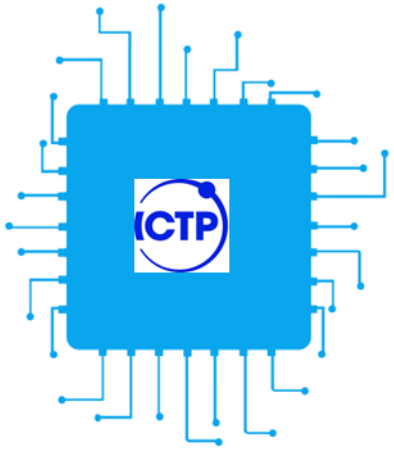
B) Alineamiento ideal del reloj



C) Retardando el reloj, parece un adelantamiento

Clocking Phase Advance





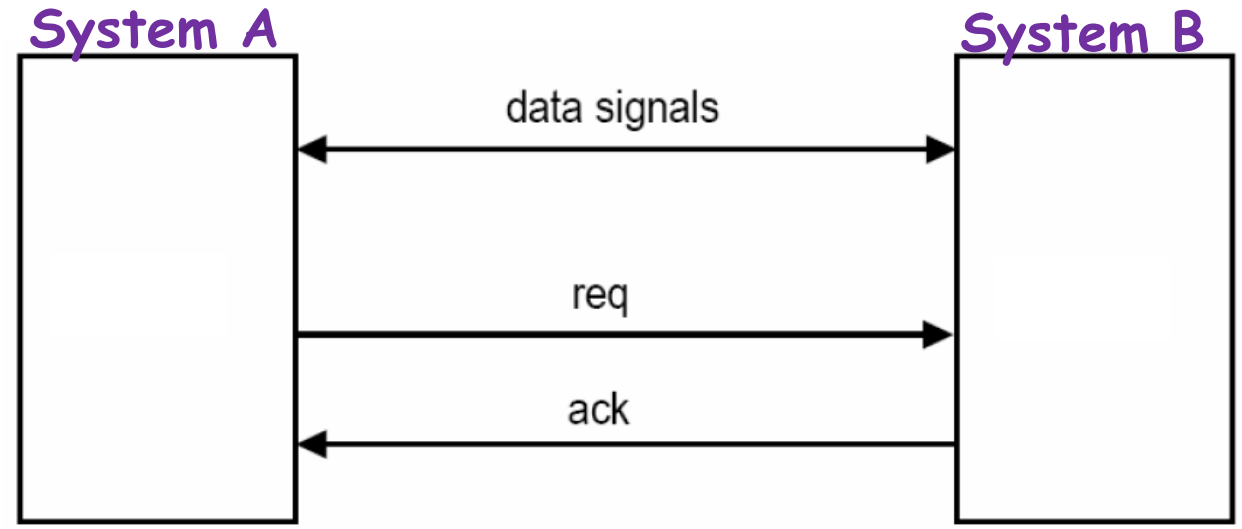
Handshaking Between Components

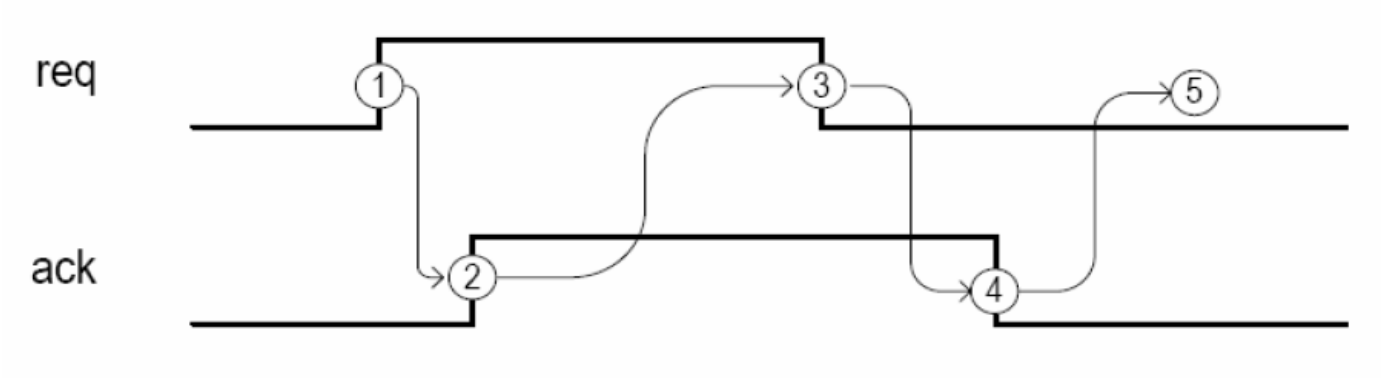
Tx/Rx Protocol Between Systems

- Problemas to face:
  - Asynchronous systems
  - No signal, clock, etc., relation between the systems

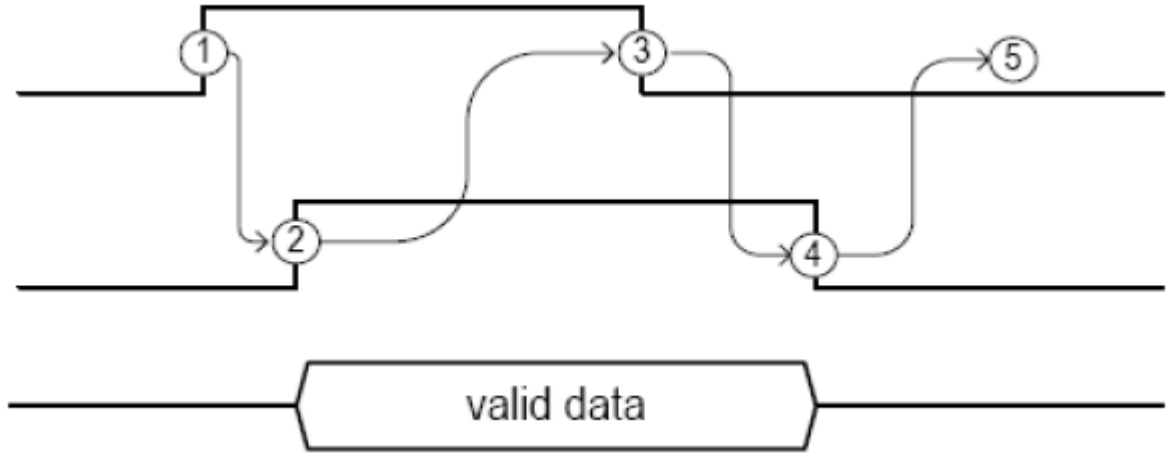
- There are several schemes, one of the most used is the '***For Phases***"

Four Phases Protocol





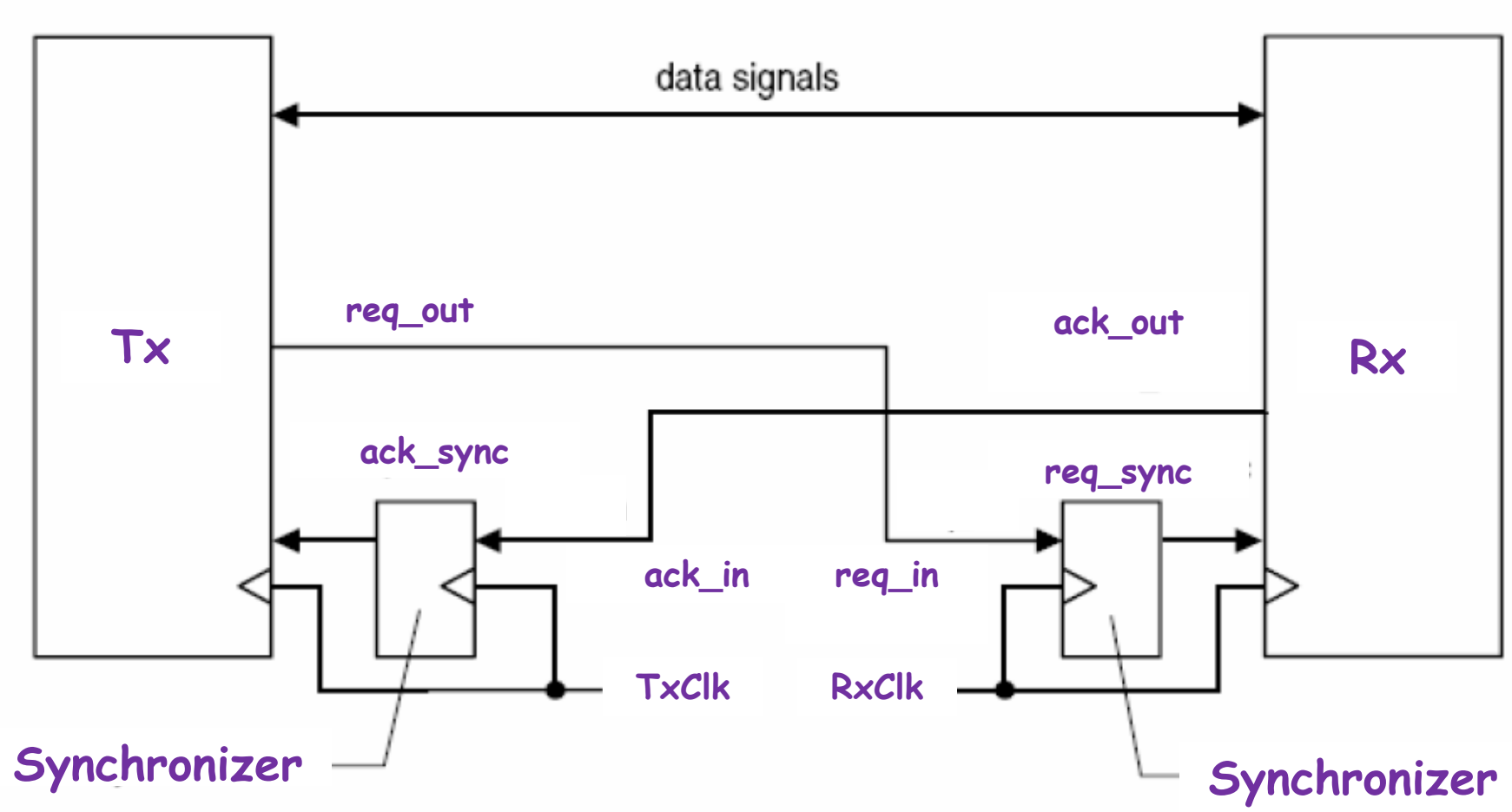
Four Phases Protocol

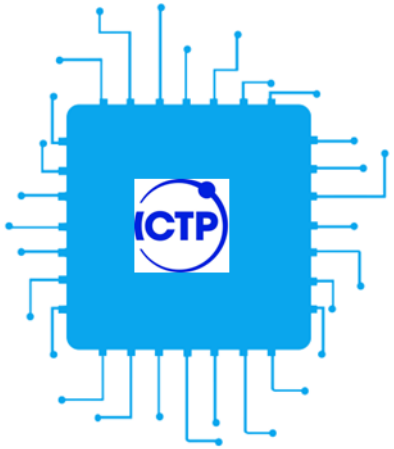


- *Phase 1*: Tx activate the request
- *Phase 2*: Rx activate the acknowledge of the request
- *Phase 3*: Tx de-activate the request
- *Phase 4*: Rx deactivate the acknolwdege

Four Phases Protocol

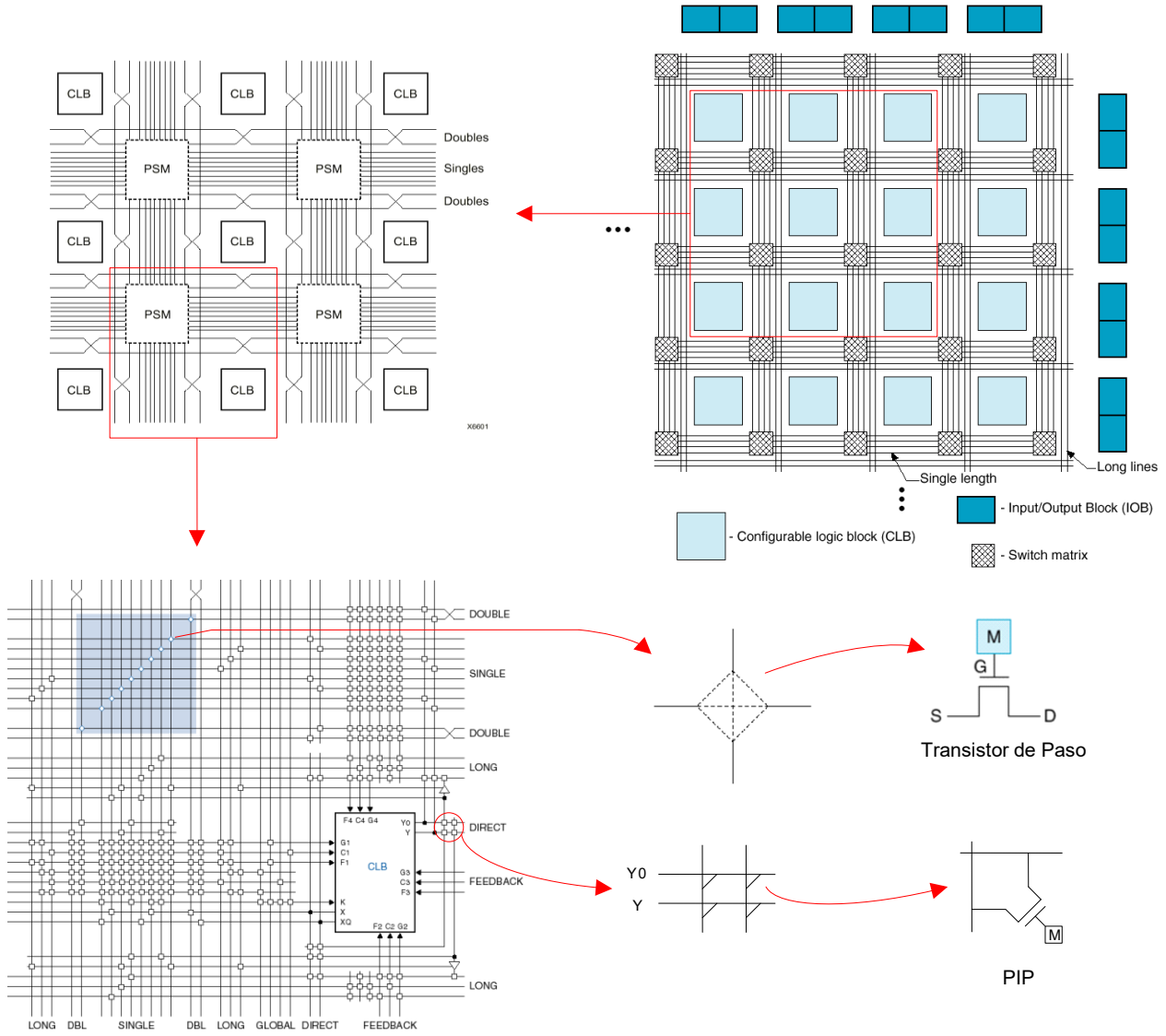
Important: If Tx and Rx are in different clock domains, use synchronizers



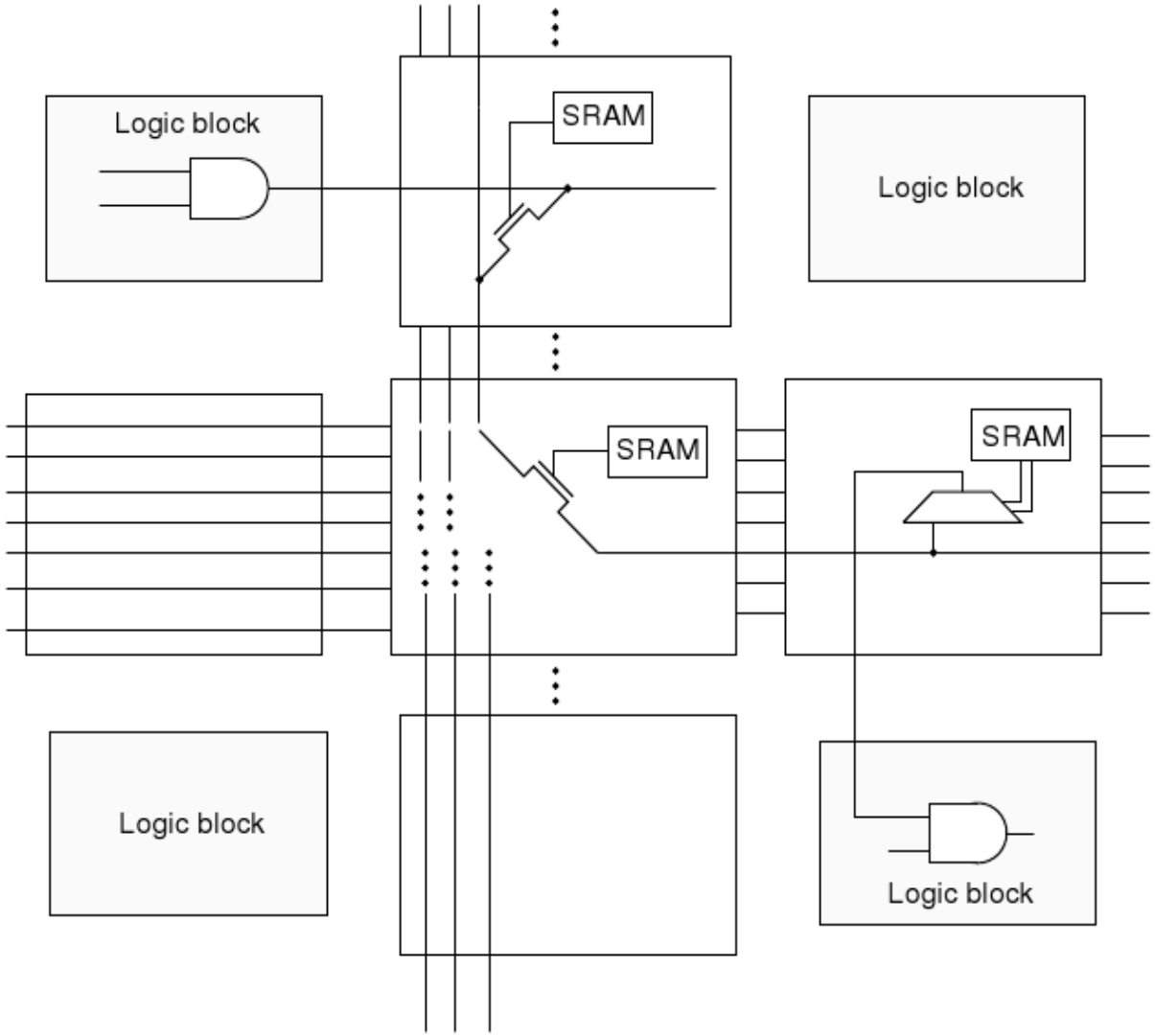


FPGA Routing

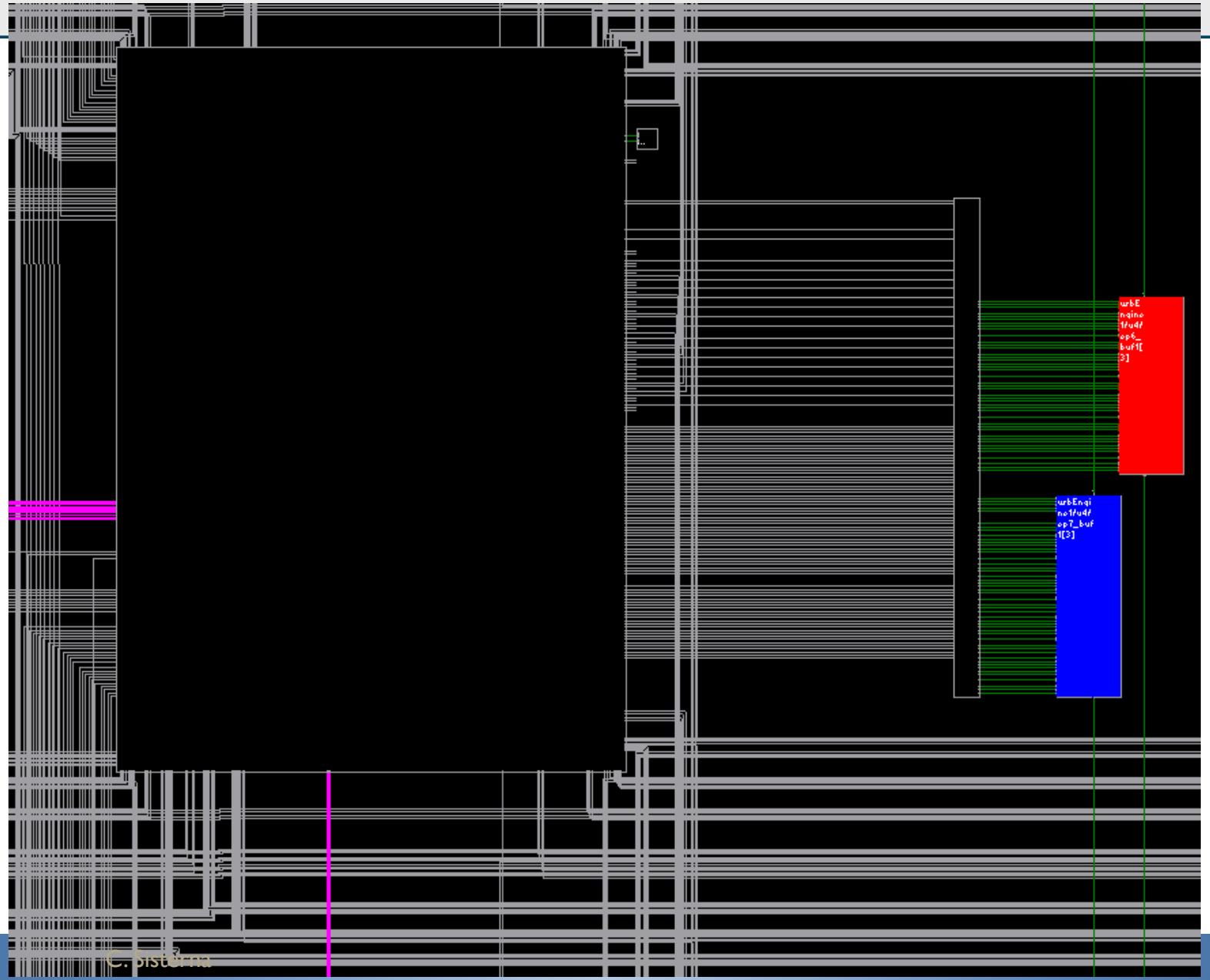
FPGA: Interconnect - Routing



FPGA: Interconnect - Routing

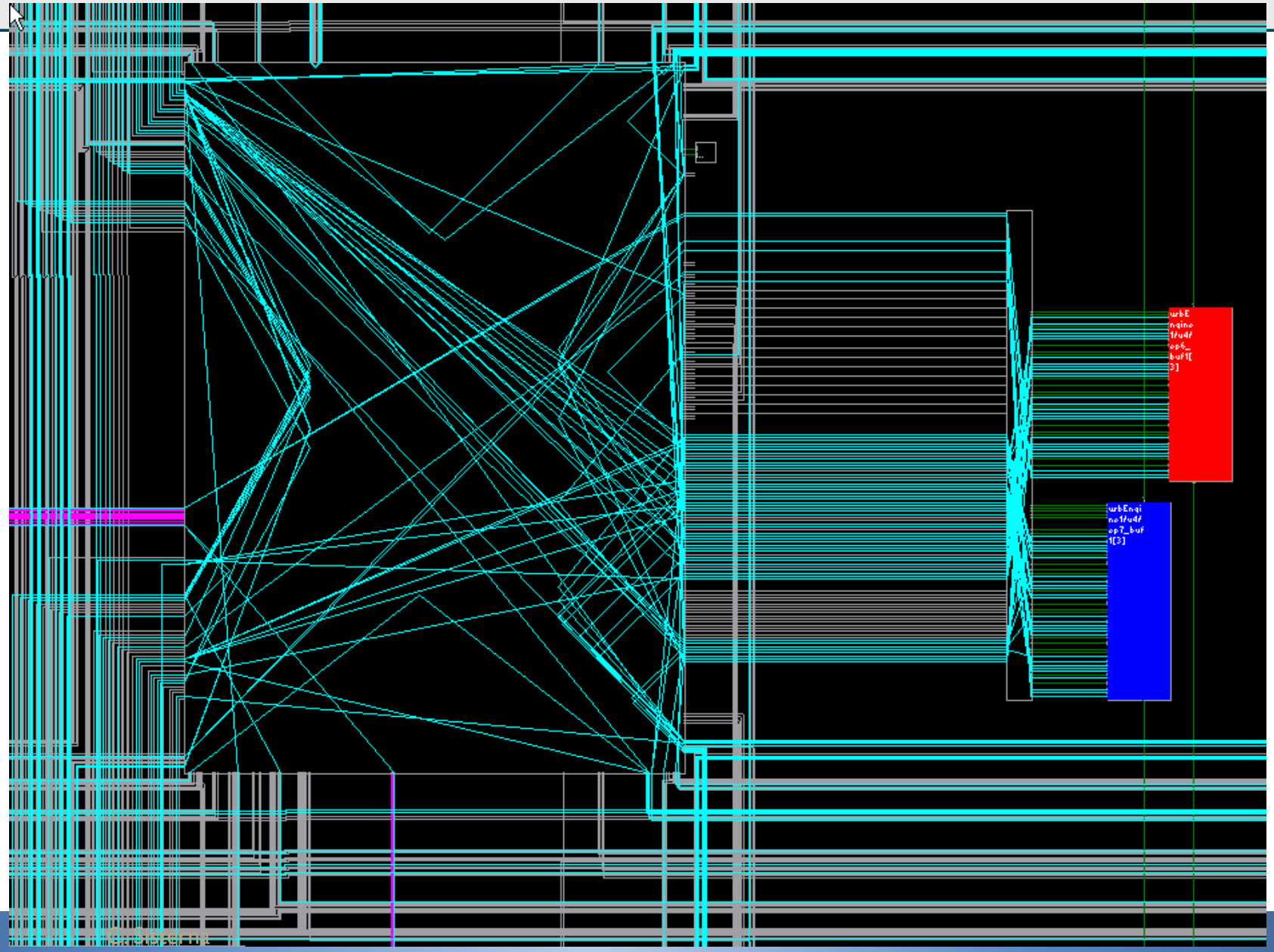


FPGA: Interconnect - Routing



C. Sisterna

FPGA: Interconnect - Routing



- **Thanks for your participation !**

	**cristian@unsj.edu.ar**
	Cristian @Linkedin
	Diseño de Sistemas Digitales Avanzados con VHDL-FPGA
	C7 Technology