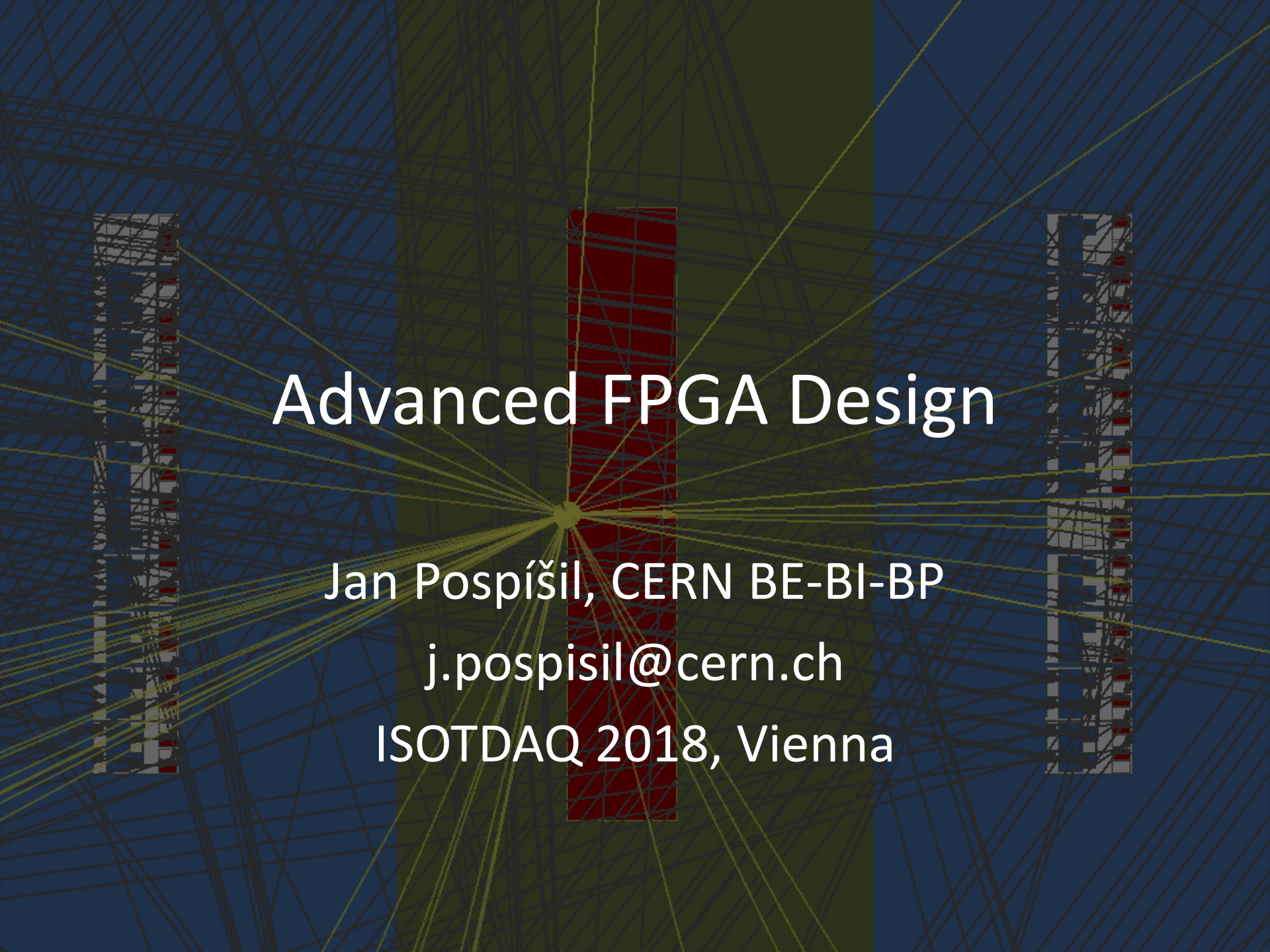


Advanced FPGA Design

Jan Pospíšil, CERN BE-BI-BP

j.pospisil@cern.ch

ISOTDAQ 2018, Vienna

Acknowledgement

- **Manoel Barros Marin** (CERN)
  - lecturer of ISOTDAQ-17
- **Markus Joos** (CERN)
  - & other organisers of ISOTDAQ-18
- **Andrea Borga** (NikheF), **Torsten Alt** (FIAS)
  - for their contribution to this lecture
- All colleagues from **CERN BE-BI-BP**

Outline

… from the previous lesson

Key concepts about FPGA design

FPGA gateware design work flow

Summary

Outline

… from the previous lesson

Key concepts about FPGA design

FPGA gateware design work flow

Summary

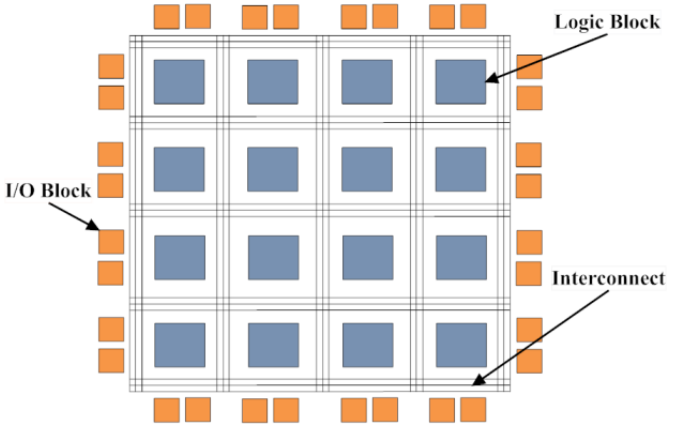
What is an FPGA?

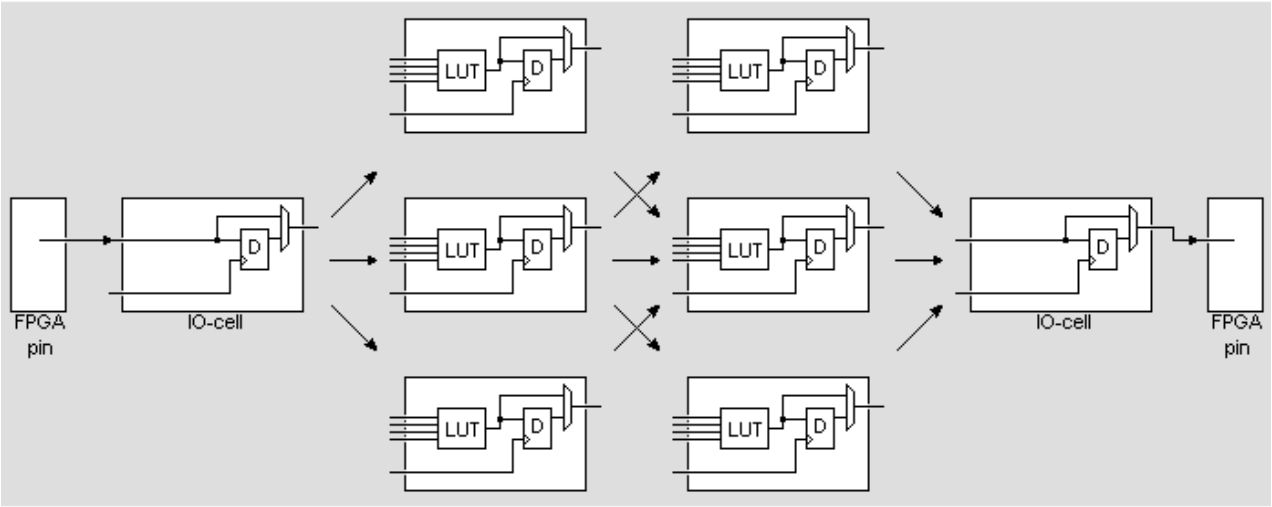
A **Field-Programmable Gate Array** (FPGA) is an integrated circuit designed to be configured by a customer or a designer after manufacturing – hence "field-programmable".

https://en.wikipedia.org/wiki/Field-programmable_gate_array

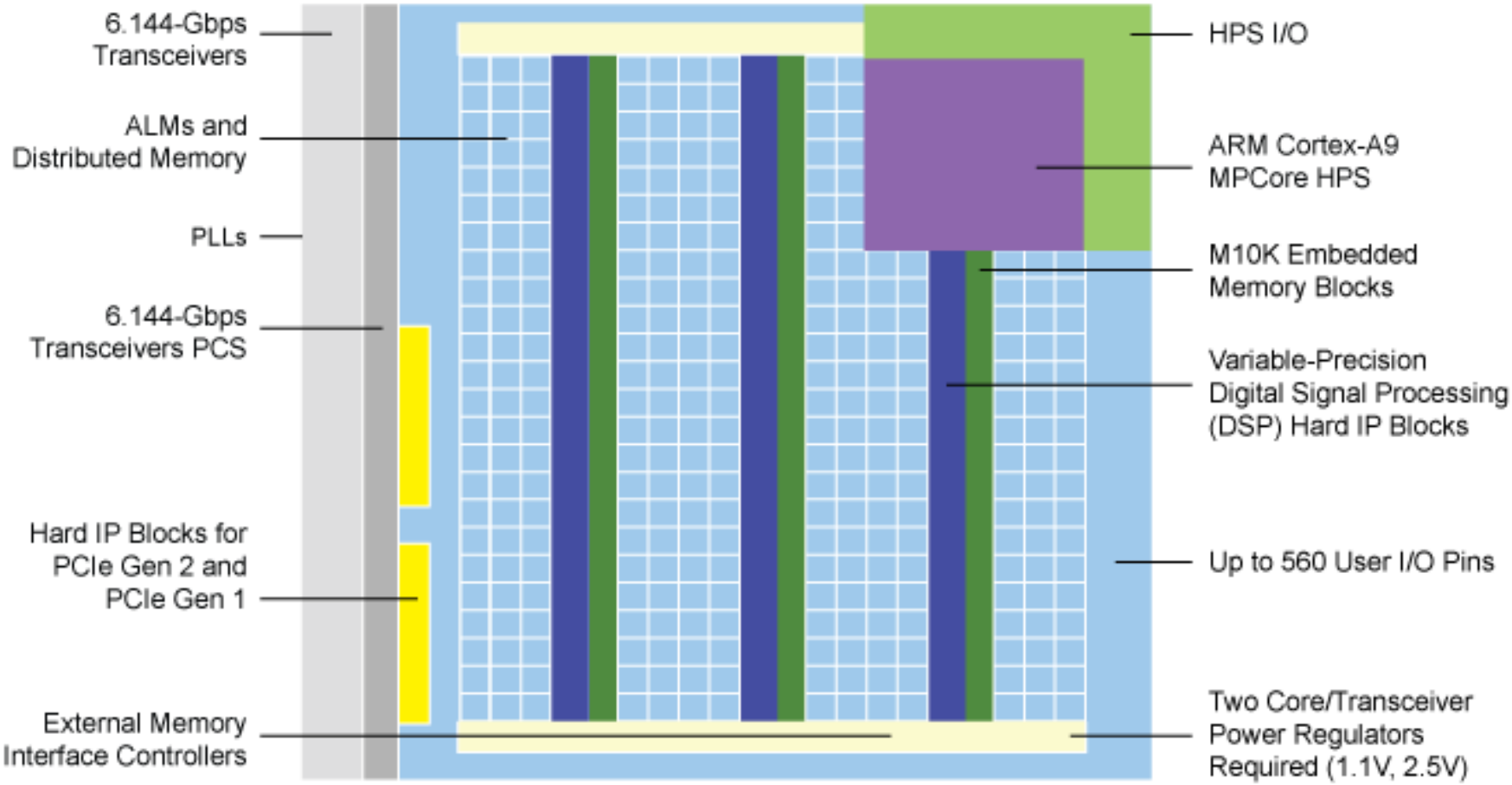
FPGA Fabric

- Matrix-like structure
- I/O cells
- Logic cells (LUT, D flip-flops…)
- Configurable interconnect

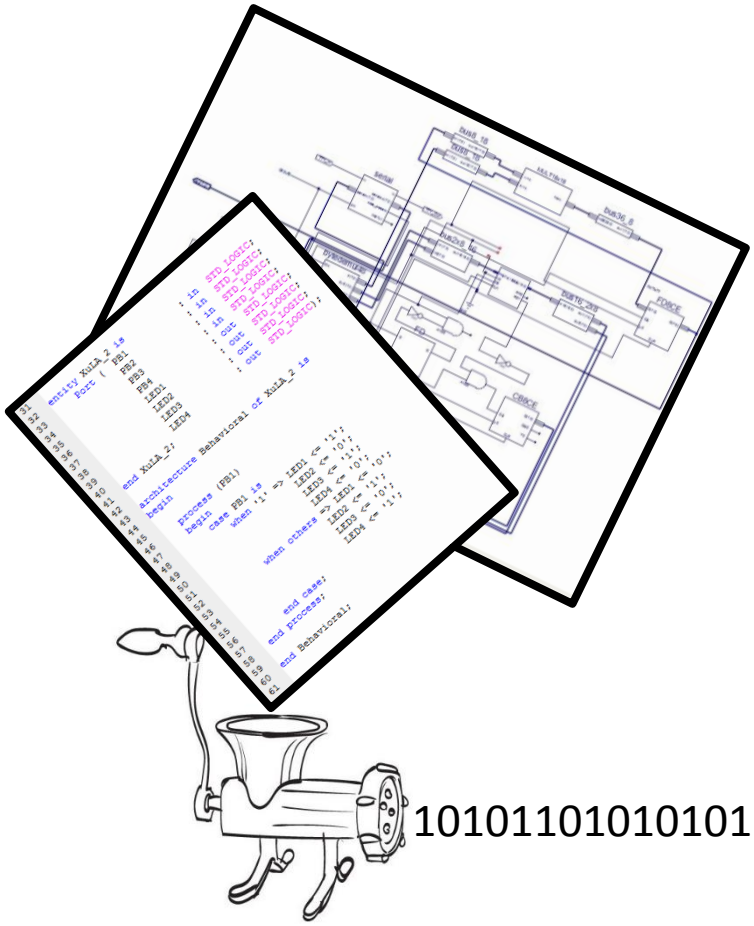




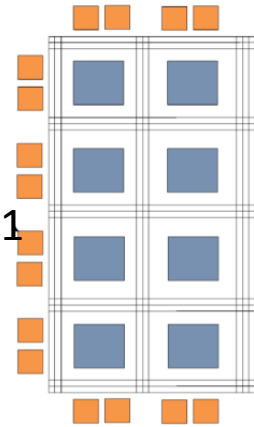
Hard Blocks


6.144-Gbps Transceivers
ALMs and Distributed Memory
PLLs
6.144-Gbps Transceivers PCS
Hard IP Blocks for PCIe Gen 2 and PCIe Gen 1
External Memory Interface Controllers
HPS I/O
ARM Cortex-A9 MPCore HPS
M10K Embedded Memory Blocks
Variable-Precision Digital Signal Processing (DSP) Hard IP Blocks
Up to 560 User I/O Pins
Two Core/Transceiver Power Regulators Required (1.1V, 2.5V)

FPGA Design Flow



10101101010101100101011010101010101011010101101001 1



Outline

… from the previous lesson

Key concepts about FPGA design

FPGA gateware design work flow

Summary

FPGA Gateware Design is not a Programming

- **Software programming**
  - Code translated into program (set of instructions)
  - Program executed sequentially (on CPU)
  - Parallelism achieved by running program on multiple cores
  - Processing structures and instruction sets are fixed
- **FPGA gateware design**
  - Processing structures defined by HW designer
  - Individual elements and their connections are described
    - by schematics or Hardware Description Language (HDL)
  - Intrinsically parallel, sequentialbehavior achieved by registers and Finite-State-Machines (FSMs)

HDL is Used for Describing Hardware

- **Example of a WAIT statement (programming language vs. HDL)**
  - In programming language (e.g. C, Unix, #include <unistd.h>)

```
sleep(5); // sleep 5 seconds
```

  - In HDL (e.g. VHDL):
    - Not synthesizable (only for simulation test benches)

```
wait for 5 sec;  -- handy for TB clocks
```

    - Synthesizable (for simulation and FPGA implementation)

```
simple_delay_counter : process (delay_rst, delay_clk)
begin  -- process
  if delay_rst = '1' then
    s_count    <= 0 ;
    s_delay_done <= '0';
  elsif rising_edge(delay_clk) then
    if delay_ena = '1' then
      if delay_ld = '1' then
        s_count <= delay_ld_value;
      else
        s_count <= s_count - 1;
      end if;
    end if;
    if s_count = 0 then
      s_delay_done <= '1';
    else
      s_delay_done <= '0';
    end if;
  end if;
 end process;
```

Register Transfer Level (RTL)

http://en.wikipedia.org/wiki/Register-transfer_level

A design abstraction which models a synchronous digital circuit in terms of the flow of digital signals (data) between registers and logical operations performed on those signals

HDL is Used for Describing Hardware

- **Example of a WAIT statement (programming language vs. HDL)**
  - In programming language (e.g. C, Unix, #include <unistd.h>)

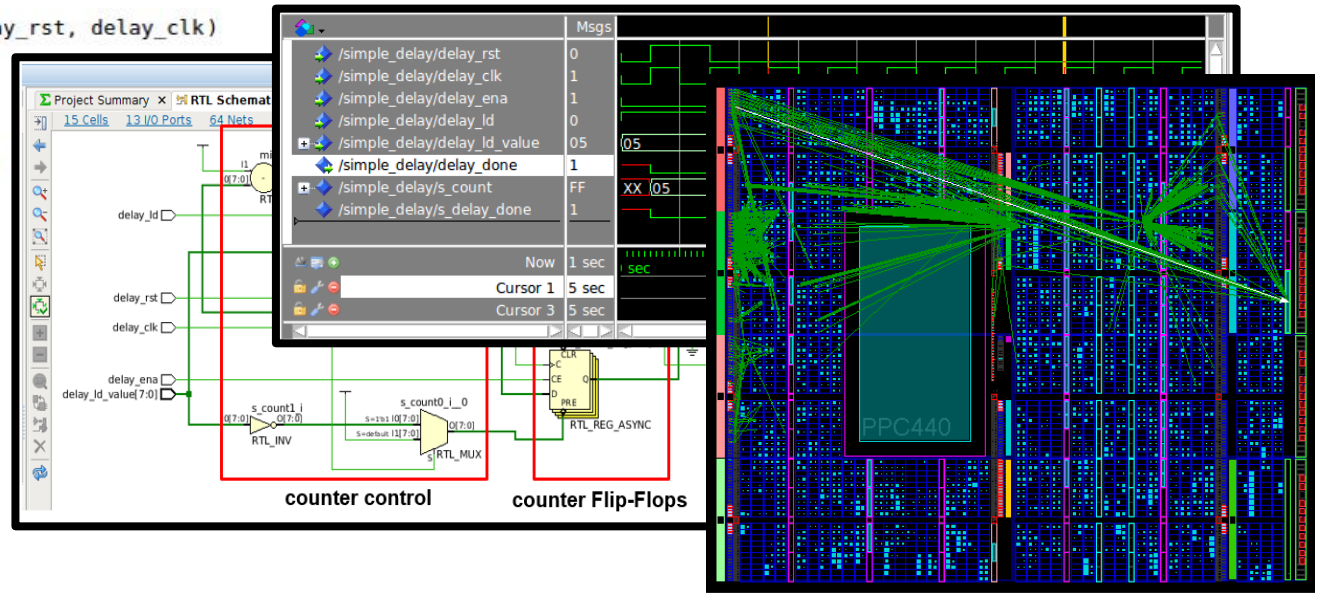
```
sleep(5); // sleep 5 seconds
```

  - In HDL (e.g. VHDL):
    - Not synthesizable (only for simulation test benches)

```
wait for 5 sec;  -- handy for TB clocks
```

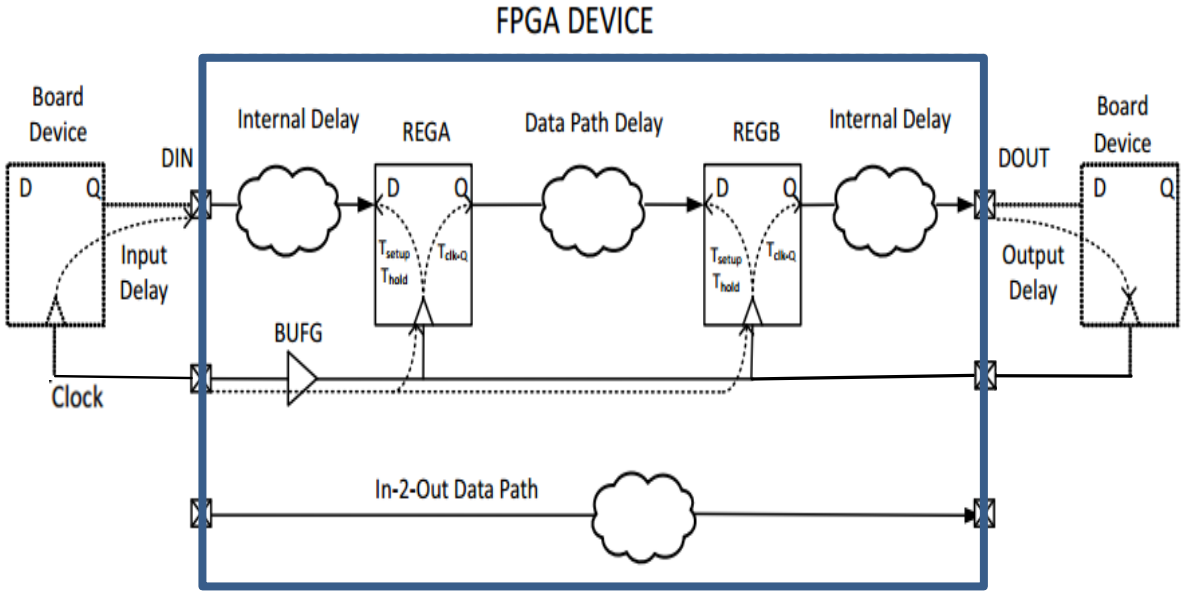
    - Synthesizable (for simulation and FPGA implementation)

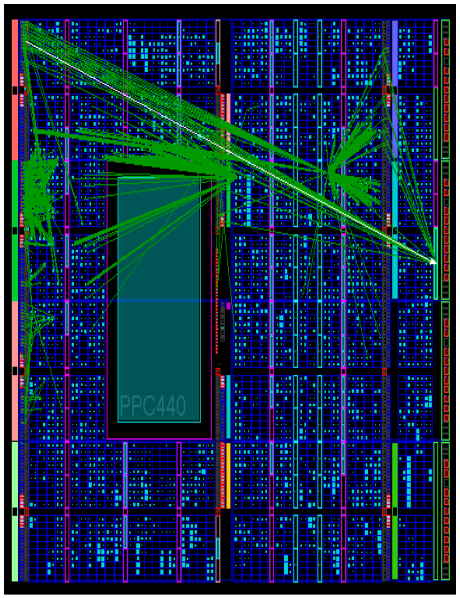
```
simple_delay_counter : process (delay_rst, delay_clk)
begin  -- process
  if delay_rst = '1' then
    s_count    <= 0 ;
    s_delay_done <= '0';
  elsif rising_edge(delay_clk) then
    if delay_ena = '1' then
      if delay_ld = '1' then
        s_count <= delay_ld_value;
      else
        s_count <= s_count - 1;
      end if;
    end if;
    if s_count = 0 then
      s_delay_done <= '1';
    else
      s_delay_done <= '0';
    end if;
  end if;
end process;
```



Timing in FPGA Gateware is Critical

- **Data propagates in the form of electrical signals through the FPGA**



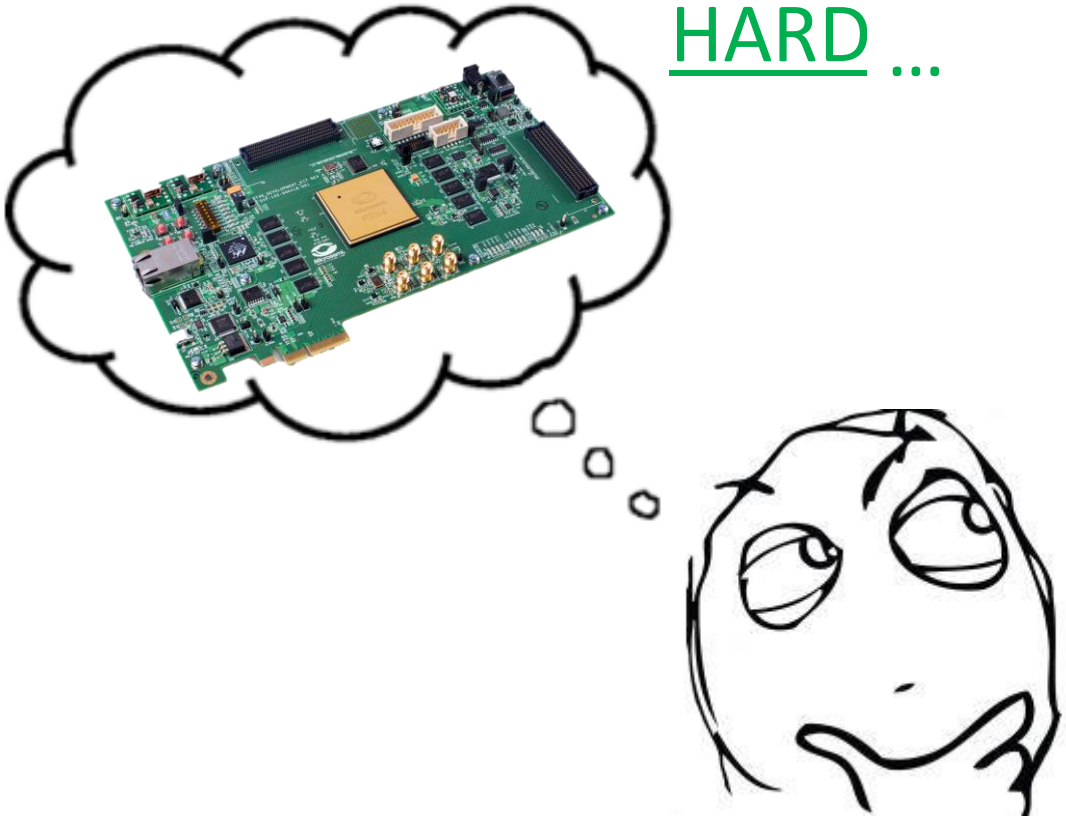


Synthesized RTL (Netlist) implemented into FPGA

- **If these signals do not arrive to their destination on time…**

The consequences may be catastrophic!!!

When designing FPGA gateware you have to think HARD …



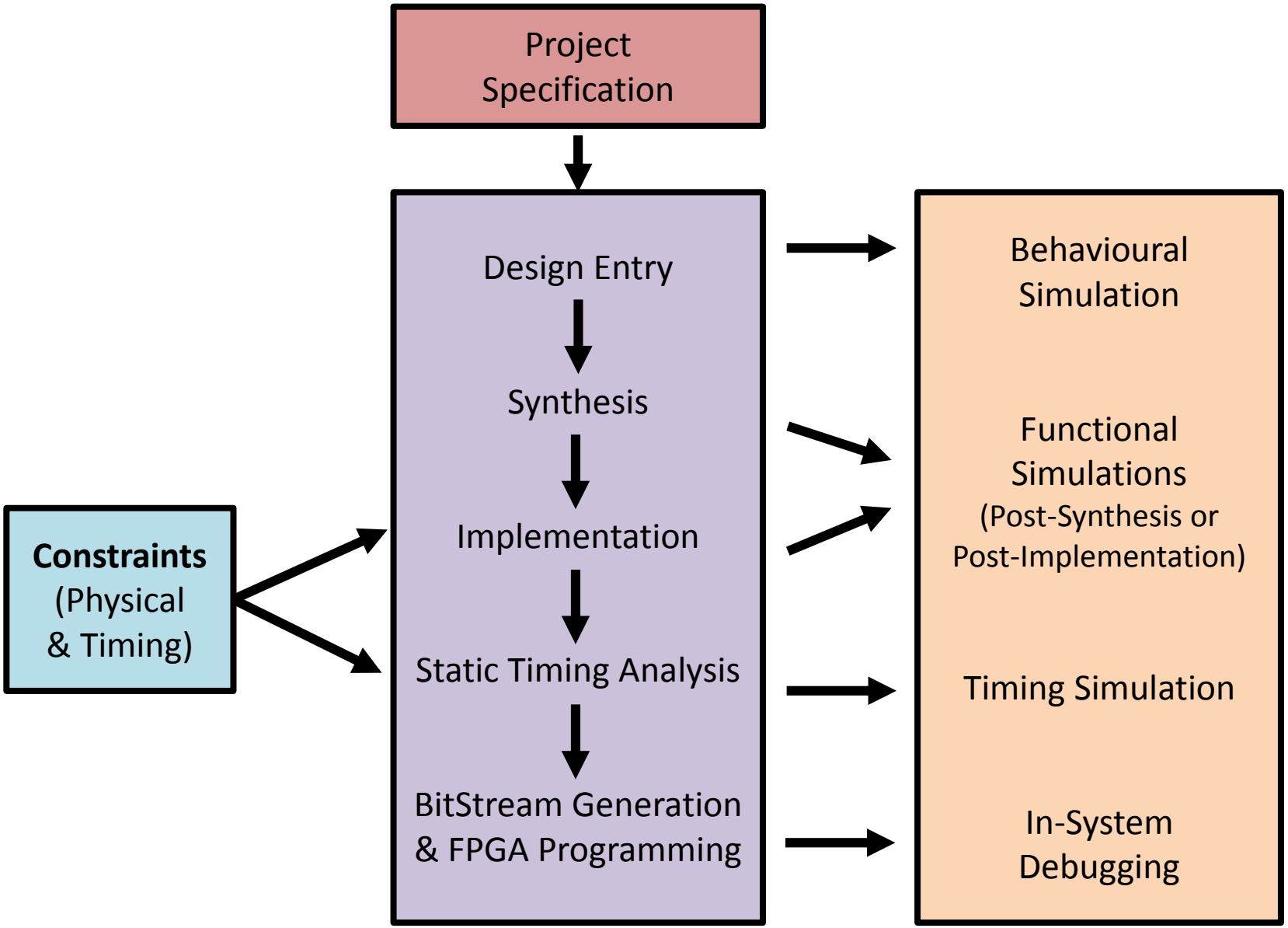
Outline

… from the previous lesson

Key concepts about FPGA design

FPGA gateware design work flow

Summary


Project Specification
Design Entry
Synthesis
Implementation
Static Timing Analysis
BitStream Generation & FPGA Programming
Constraints (Physical & Timing)
Behavioural Simulation
Functional Simulations (Post-Synthesis or Post-Implementation)
Timing Simulation
In-System Debugging

Project Specification

This is the most critical step…

The rest of the design process is based on it!

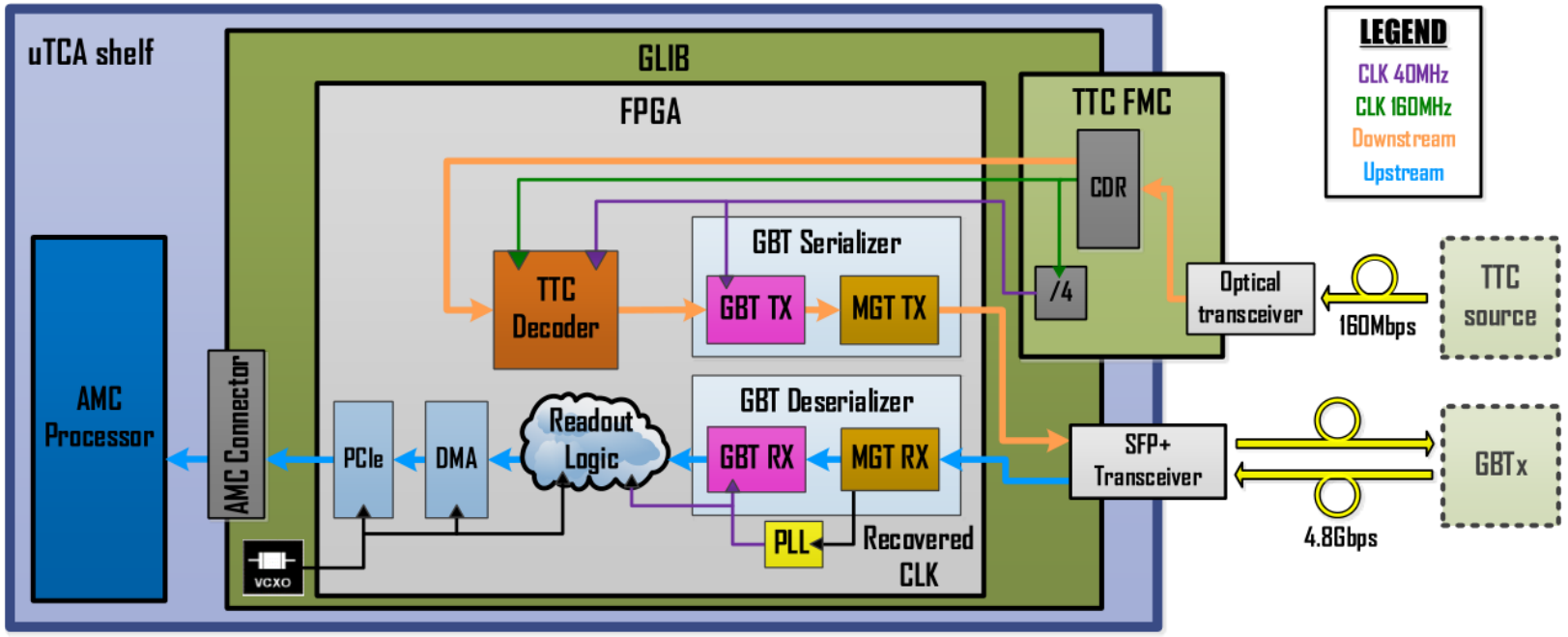
Project Specification

1. Gather requirements
2. Specify:

Project Specification

1. Gather requirements
2. Specify:
   - **Target application (specific or general purpose)**

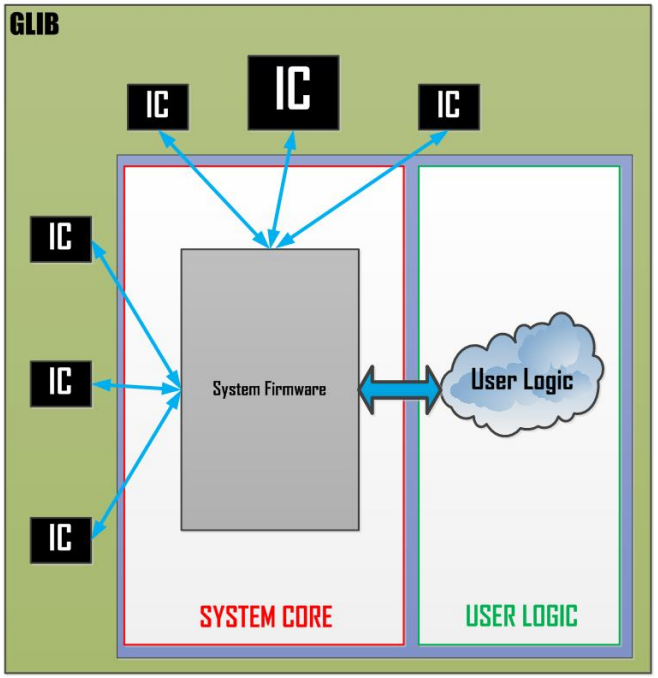
Application Specific Gateware



Project Specification

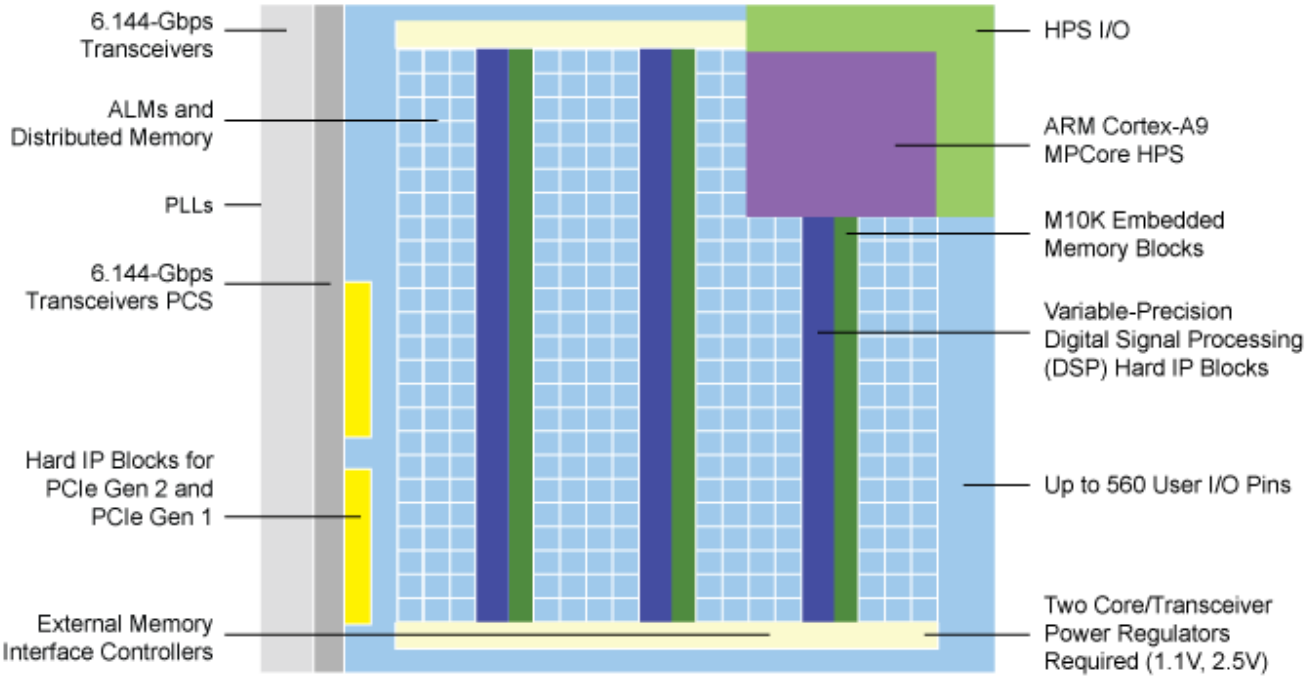
1. Gather requirements
2. Specify:
   - **Target application (specific or general purpose)**

General Purpose Gateware



Project Specification

1. Gather requirements
2. Specify:
   - Target application (specific or general purpose)
   - **Features needed (SoC, multi-gigabit transceivers…)**



Project Specification

1. Gather requirements
2. Specify:
   - Target application (specific or general purpose)
   - Features needed (SoC, multi-gigabit transceivers…)
   - **FPGA vendor (Xilinx, Intel (Altera), Microsemi (Actel), Lattice…)**

XILINX ALL PROGRAMMABLE™

ALTERA now part of Intel

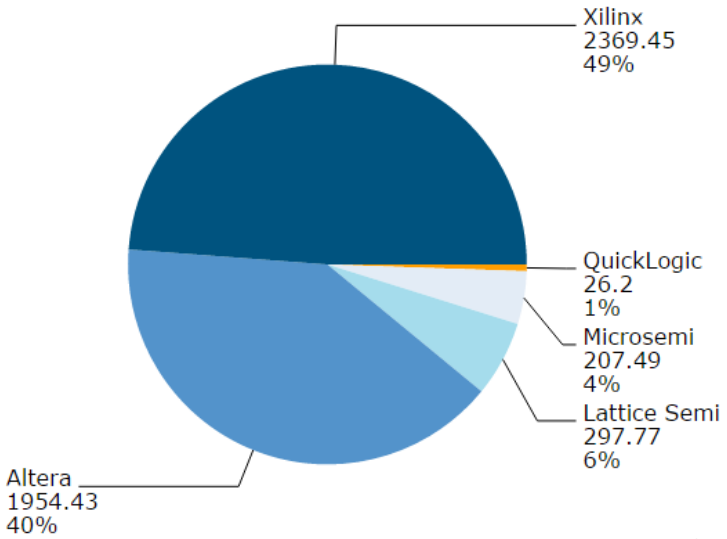
ATMEL

Microsemi

LATTICE SEMICONDUCTOR

QuickLogic

Small FPGA vendors may target specific markets (Microsemi offers high reliable FPGAs…)



FPGA market-share by 2010 (M$)

Project Specification

1. Gather requirements
2. Specify:
   - Target application (specific or general purpose)
   - Features needed (SoC, multi-gigabit transceivers…)
   - FPGA vendor (Xilinx, Intel (Altera), Microsemi (Actel),
   - **Electronic board (custom or COTS*)**

Custom Board

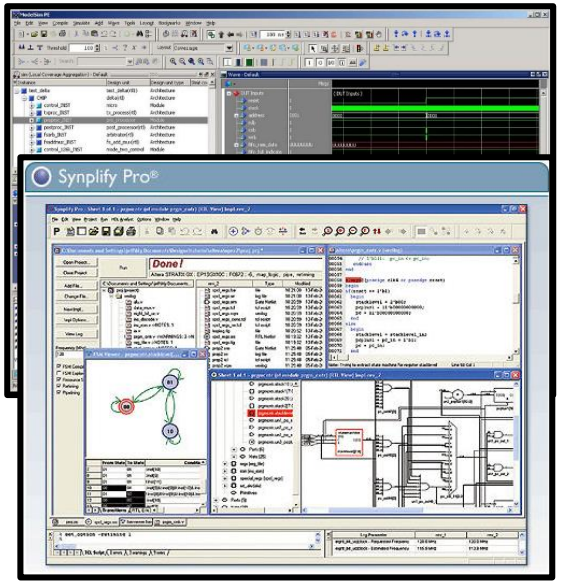
COTS board (Xilinx Devkit)

(*) Commercial Off-The-Shelf (COTS)

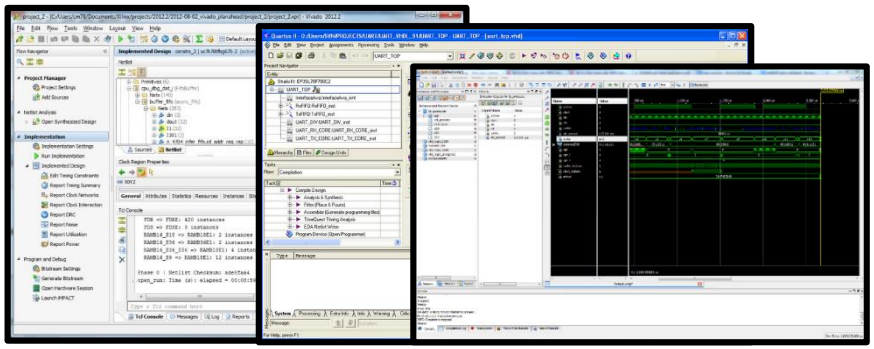
Project Specification

1. Gather requirements
2. Specify:
   - Target application (specific or general purpose)
   - Features needed (SoC, multi-gigabit transceivers…)
   - FPGA vendor (Xilinx, Intel (Altera), Microsemi (Actel), Lattice…)
   - Electronic board (custom or COTS)
   - **Development tools (FPGA vendor or commercial)**

Commercial Tools



FPGA Vendor Tools



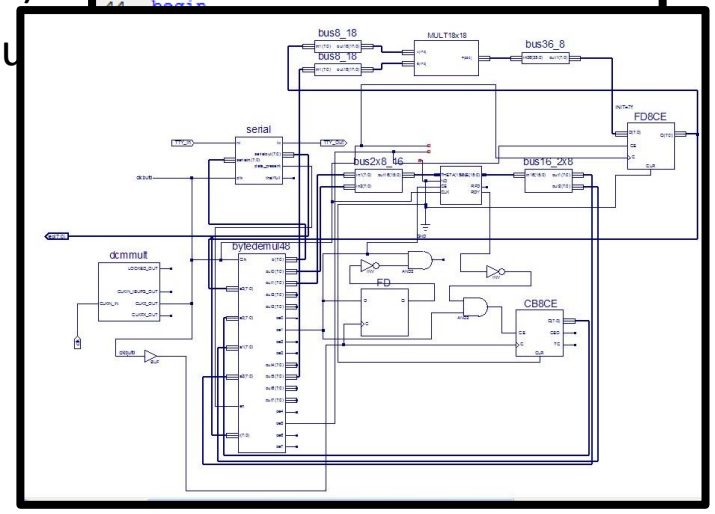
Project Specification

1. Gather requirements
2. Specify:
   - Target application (specific or general purpose)
   - Features needed (SoC, multi-gigabit transceivers…)
   - FPGA vendor (Xilinx, Intel (Altera), Microsemi (Actel),
   - Electronic board (custom or COTS)
   - Development tools (FPGA vendor or commercial)
   - **Optimization (speed, area, power or none/default)**

Project Specification

1. Gather requirements
2. Specify:
   - Target application (specific or general purpose)
   - Features needed (SoC, multi-gigabit transceivers…)
   - FPGA vendor (Xilinx, Intel (Altera), Microsemi (Actel)
   - Electronic board (custom or COTS)
   - Development tools (FPGA vendor or commercial)
   - Optimization (speed, area, power or none/defau
   - **Design entry (schematics or HDL – which?)**

```
31
32  entity XuLA_2 is
33      Port (  PB1             : in  STD_LOGIC;
34              PB2             : in  STD_LOGIC;
35              PB3             : in  STD_LOGIC;
36              PB4             : in  STD_LOGIC;
37              LED1            : out  STD_LOGIC;
38              LED2            : out  STD_LOGIC;
39              LED3            : out  STD_LOGIC;
40              LED4            : out  STD_LOGIC);
41  end XuLA_2;
42
43  architecture Behavioral of XuLA_2 is
```



HDL most popular for RTL design but…
Schematics may be better in some cases
(SoC bus interconnect…)

Project Specification

1. Gather requirements
2. Specify:
   - Target application (specific or general purpose)
   - Features needed (SoC, multi-gigabit transceivers…)
   - FPGA vendor (Xilinx, Intel (Altera), Microsemi (Actel), Lattice…)
   - Electronic board (custom or COTS)
   - Development tools (FPGA vendor or commercial)
   - Optimization (speed, area, power or none/default)
   - Design entry (schematics or HDL – which?)
   - **Coding convention**

How to write
- FSM
- Synchronizers
- …

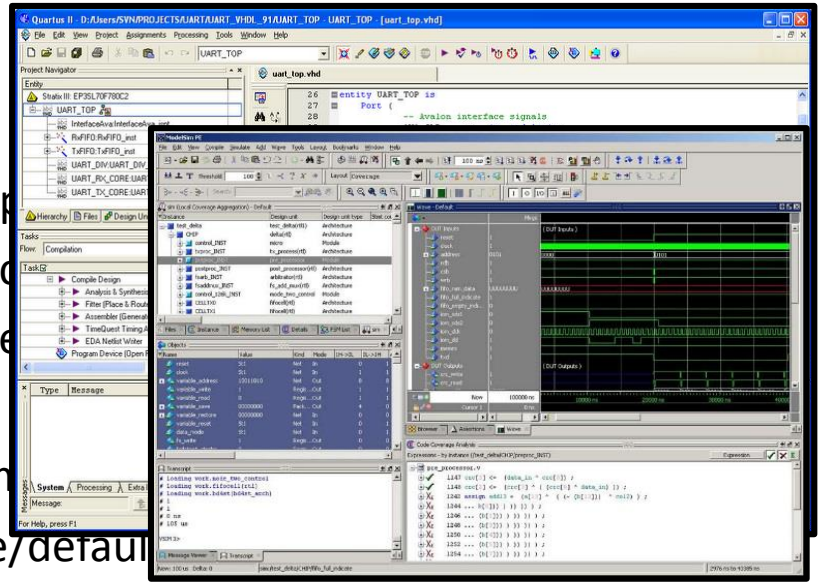
Your code should be readable

Variable Naming Convention

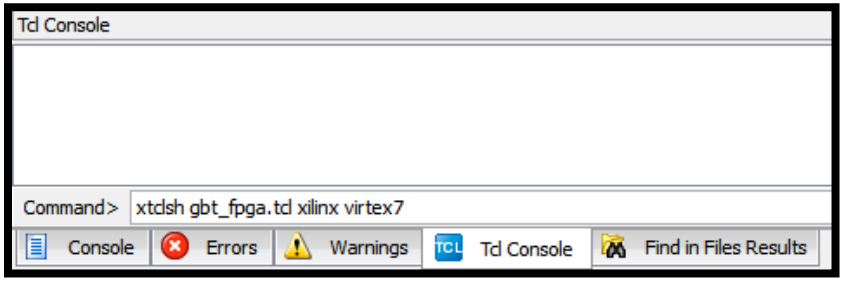
description	extension	example
variable	prefix v	v_Buffer
alias	prefix a	a_Bit5
constant	prefix c	c_Lenght
type definition	prefix t	t_MyType
generics	prefix g	g_Width

Project Specification

1. Gather requirements
2. Specify:
   - Target application (specific or general purp
   - Features needed (SoC, multi-gigabit transc
   - FPGA vendor (Xilinx, Intel (Altera), Microse
   - Electronic board (custom or COTS)
   - Development tools (FPGA vendor or comm
   - Optimization (speed, area, power or none/defaul
   - Design entry (schematics or HDL – which?)
   - Coding convention
   - **Tool interface (GUI, scripts, both)**



Xilinx ISE Tcl console



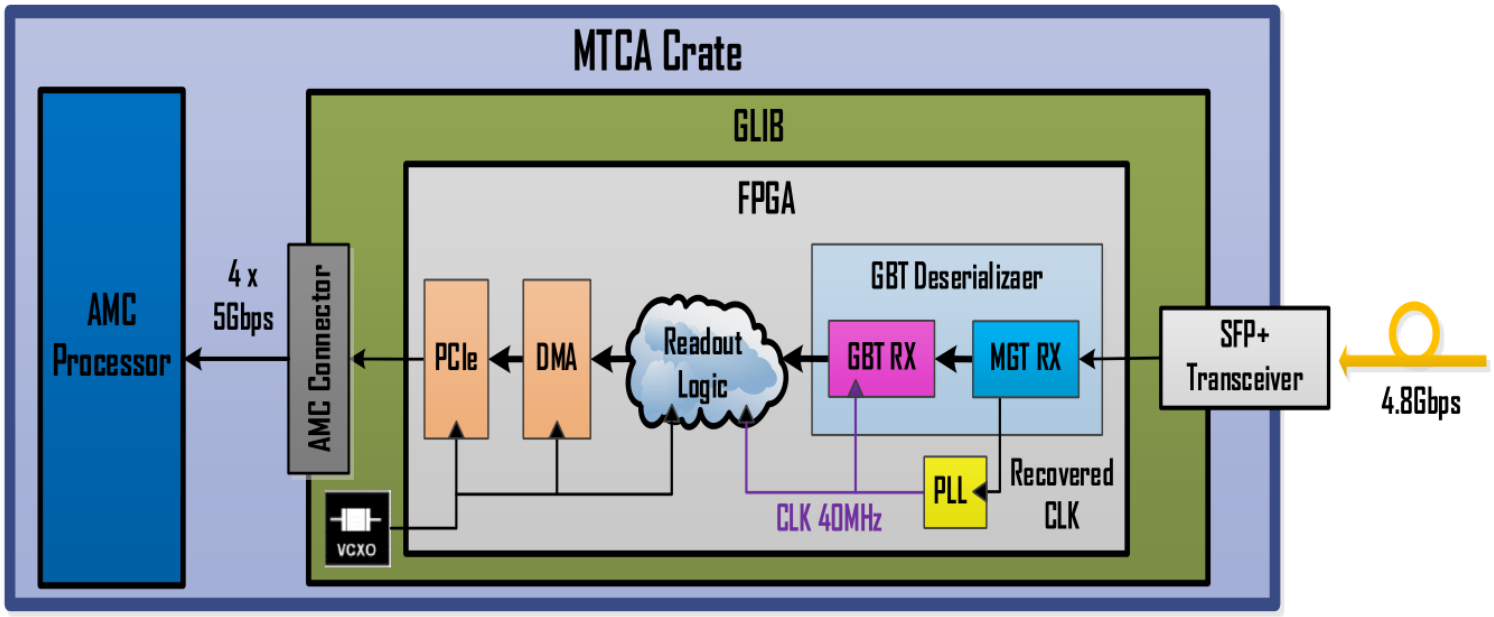

```
Info: Quartus Prime Fitter was successful. 0 errors, 70 warnings
    Info: Peak virtual memory: 5679 megabytes
    Info: Processing ended: Wed Jan 31 13:50:58 2018
    Info: Elapsed time: 00:14:45
    Info: Total CPU time (on all processors): 00:45:15
[fosfor@kravicka Quartus]$
```

Project Specification

1. Gather requirements
2. Specify:
   - Target application (specific or general purpose)
   - Features needed (SoC, multi-gigabit transceivers…)
   - FPGA vendor (Xilinx, Intel (Altera), Microsemi (Actel), Lattice…)
   - Electronic board (custom or COTS)
   - Development tools (FPGA vendor or commercial)
   - Optimization (speed, area, power or none/default)
   - Design entry (schematics or HDL – which?)
   - Coding convention
   - Tool interface (GUI, scripts, both)
   - **Code/project management (SVN, GIT…)**

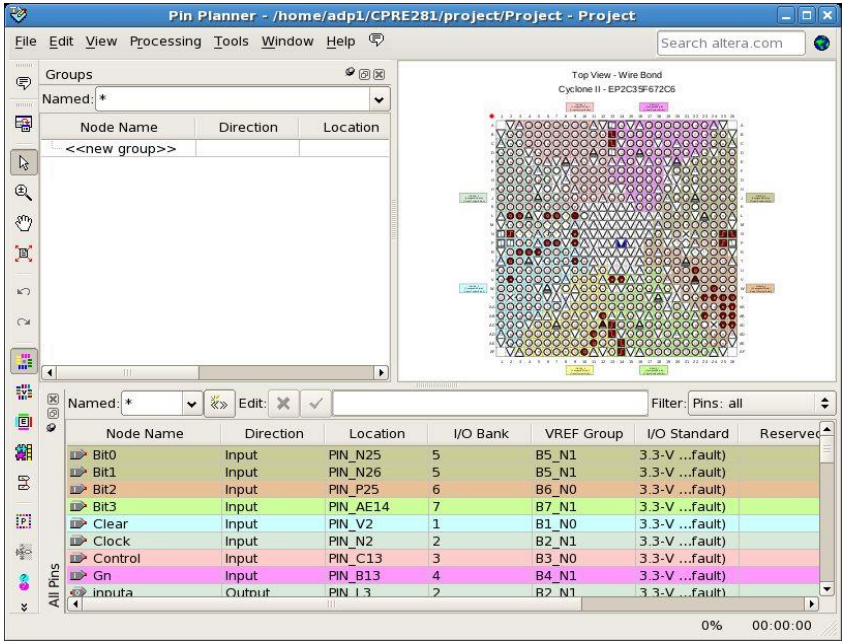
Project Specification

3. Block diagram of the system
   - FPGA logic
   - But also related/connected devices



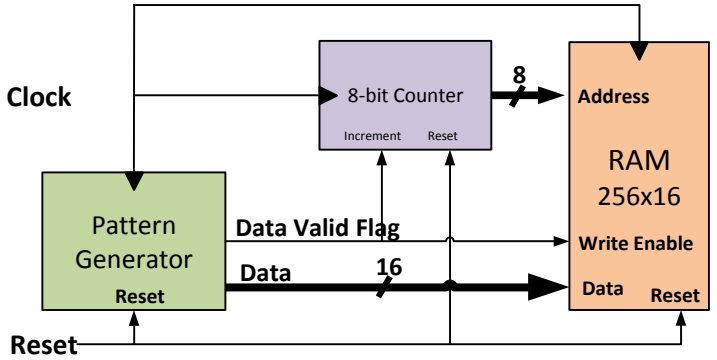
Project Specification

4. Pin planning
   - Critical for board development
   - One type of location constraints



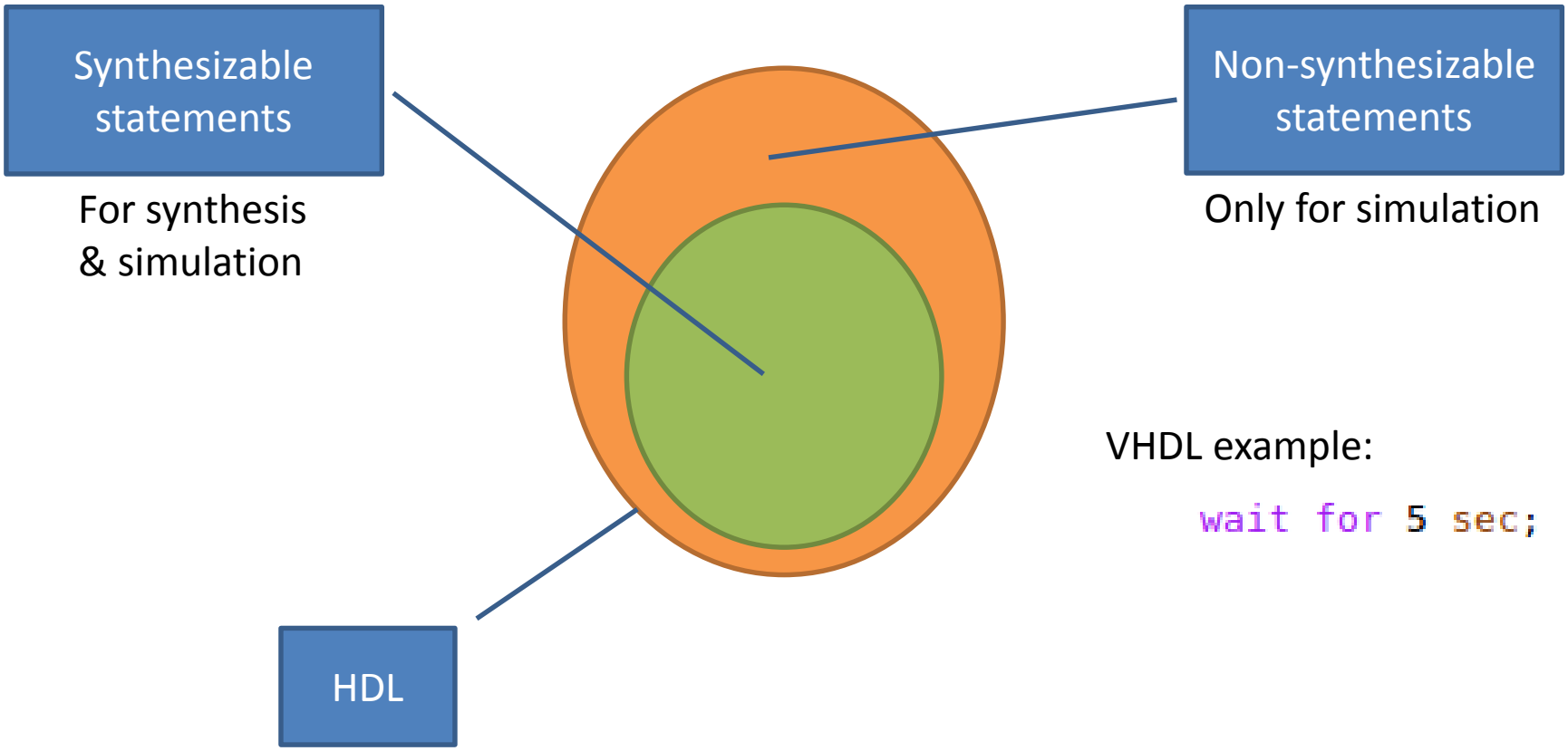
Design Entry

- System should be **Modular**
  - Modules and instantiations
  - Separated Data & Control paths
  - Well defined clock and reset schemes
  - Design at RTL level (think hard…ware)



- Code should be **Reusable**
  - Add primitives (and modules) by inference when possible
  - Parameterize code (VHDL generics, SystemVerilog parameters…)
  - Centralize parameters (VHDL packages, SystemVerilog packages…)
  - Use configurable modules interfaces (VHDL records, SystemVerilog interfaces)
  - Use standard features (I2C, SPI, Wishbone…)
  - Use existing IP cores (e.g. from www.OpenCores.org)
  - Avoid vendor specific IP Cores when possible
  - Talk with your colleagues and see what other FPGA designers are doing

Design Entry: Coding for Synthesis

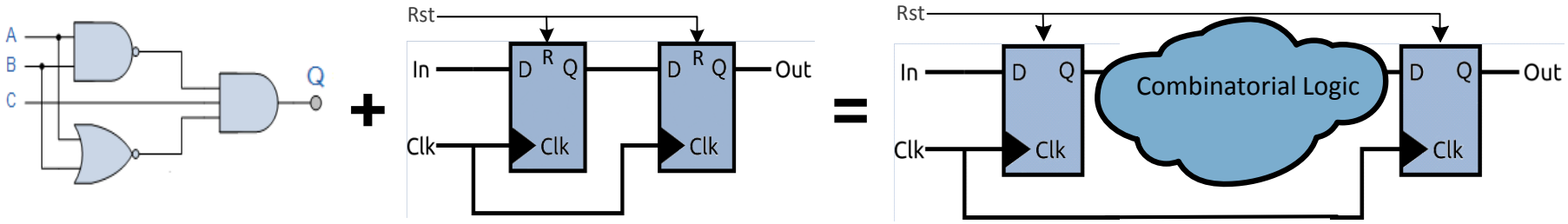


VHDL example:

```
wait for 5 sec;
```

Design Entry: Synchronous Design

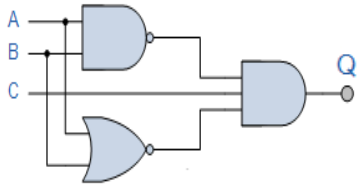
- All memory elements are synchronized on clock
- Simplifies the designing process
- Synchronous design separates:
  - Combinatorial logic (logic function)
  - Sequential logic (memory elements)



- FPGAs architecture and tools are designed to be use with synchronous designs – use it!

Design Entry: Synchronous Design

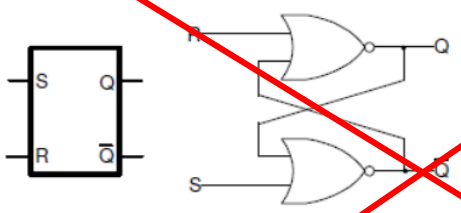
- Combinatorial logic rules
  - Sensitivity list must include ALL input signals
    Otherwise outputs can be non-responsive under changes of inputs
  - ALL output signals must be assigned under ALL possible input conditions
    Otherwise undesired latches can be created (asynchronous storage element)
  - No feedback from output to input signals
    Otherwise unknown output states (metastability) & undesired latches



C	B	A	Q
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

```
process (Input_A, Input_B, Input_C, Output_nand, Output_nor)
begin
    Output_nand <= Input_A nand Input_B;
    Output_nor  <= Input_A nor  Input_B;
    --
    Output_Q    <= Output_nand and Input_C and Output_nor;
end process;
```

Asynchronous Latch



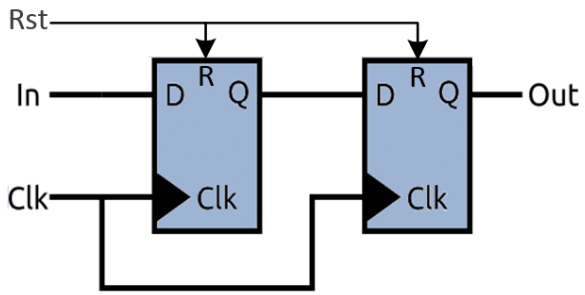
S	R	Q	$\overline{Q}$
0	0	Latch	
0	1	0	1
1	0	1	0
1	1	Metastable	

```
process(Input_R)
begin
    Output_Q   <= Input_R nor Output_Q_n;
    Output_Q_n <= Input_S nor Output_Q;
end process;
```

Design Entry: Synchronous Design

- Sequential logic rules
  - Only clock signal (and asynchronous set/reset when used) in sensitivity list
    Otherwise undesired combinatorial logic can be produced
  - All registers of the sequence must be triggered by the same clock
    Otherwise metastability can happen
  - Include all registers of the sequence in the same reset branch
    Otherwise undesired register values can appear after reset

```
process(Clk,Rst)
begin
    if (Rst = '1') then
        Output_Out <= '0';
        Output_Q   <= '0';
    elsif rising_edge(Clk) then
        Output_Out <= Output_Q;
        Output_Q   <= Input_In;
    end if;
end process;
```




```
process(Clk,Rst,Input_In)
begin
    if (Rst = '1') then
        Output_Out <= '0';
    elsif rising_edge(Clk) then
        Output_Out <= Output_Q;
        Output_Q   <= Input_In;
    end if;
end process;
```

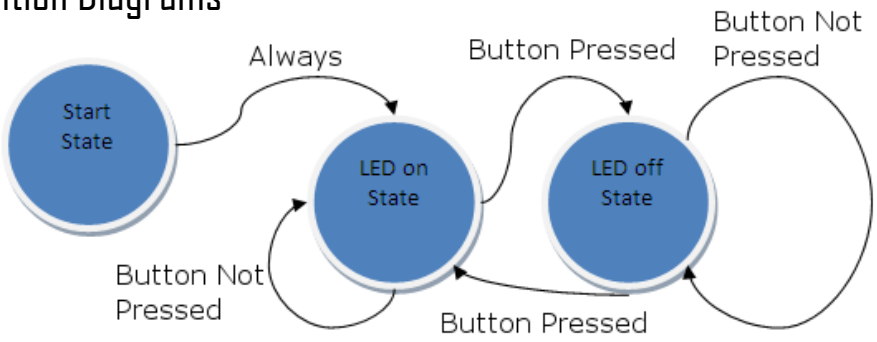
Design Entry: Synchronous Design

- Synchronous design rules
  - FULLY synchronous design
    - No combinatorial feedback
    - No asynchronous latches

    Otherwise design tools can do incorrect analysis
  - Time-constrain ALL input/output signals (registering may help)
    Otherwise uncontrolled length of paths can happen
  - Properly design the reset scheme (mentioned later)
    Otherwise undesired register values can appear after reset
  - Properly design the clocking scheme (mentioned later)
    Otherwise metastability & resources misuse can happen
  - Properly handle Clock Domain Crossings (CDC) (mentioned later)
    Otherwise metastability can happen

Design Entry: Finite State Machines

- **Finite State Machines (FSMs):**
  - Digital logic circuit with a finite number of internal states
  - Used modelling sequential behavior
  - Two variants of FSM
    - Moore: outputs depends only on the current state of the FSM
    - Mealy: outputs depends on the current state of the FSM and current values of inputs
  - Modelled by State Transition Diagrams



  - Many different FSM coding styles (But not all of them are good!!)
  - FSM coding considerations:
    - Outputs may be assigned during states or state transitions
    - Be careful with unreachable/illegal states

Design Entry: Reset Scheme

- **Used to initialize registers outputs to a know state**
- **It has a direct impact on:**
  - Performance
  - Logic utilization
  - Reliability
- **Different approaches:**
  - Asynchronous
    Pros: No free running clock required, easier timing closure
    Cons: skew, glitches, simulation mismatch, difficult to debug, extra constraints, etc.
  - Synchronous
    Pros: No Skew, No Glitches, No simulation mismatch, Easier to debug, No extra constraints, etc..
    Cons: Free-running clock required, More difficult timing closure
  - No Reset Scheme
    Pros; Easier Routing, Less resources, Easiest timing closure
    Cons: Only reset at power up (in some devices not even that…) **<- In fact, reset is not always needed**
  - Hybrid: Usually in big designs (**Avoid when possible!!!**)

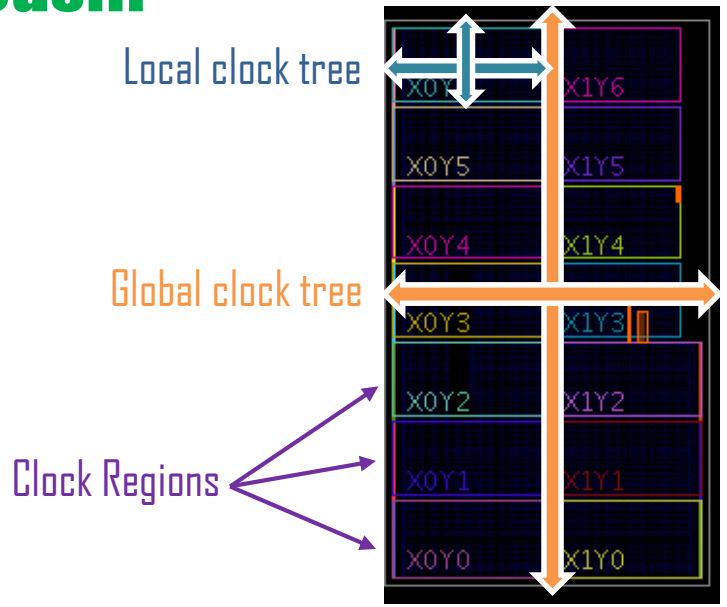
A bad reset scheme may get you crazy!!!

You should use SYNCHRONOUS RESET by default

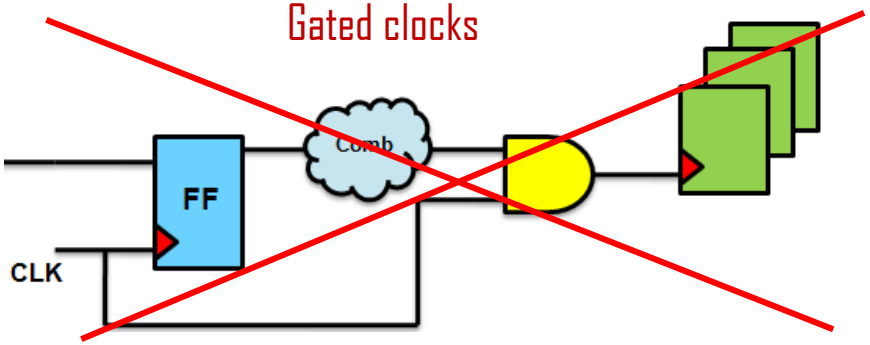
Design Entry: Clock Scheme

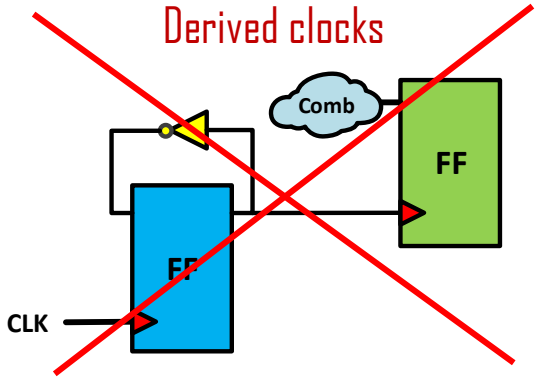
Clocking resources are very precious!!!

- **Clock regions**
- **Clock trees (Global & Local)**
- **Other FPGA clocking resources**
  - Clock capable pins
  - Clock buffers
  - Clock Multiplexors
  - PLLs & DCM

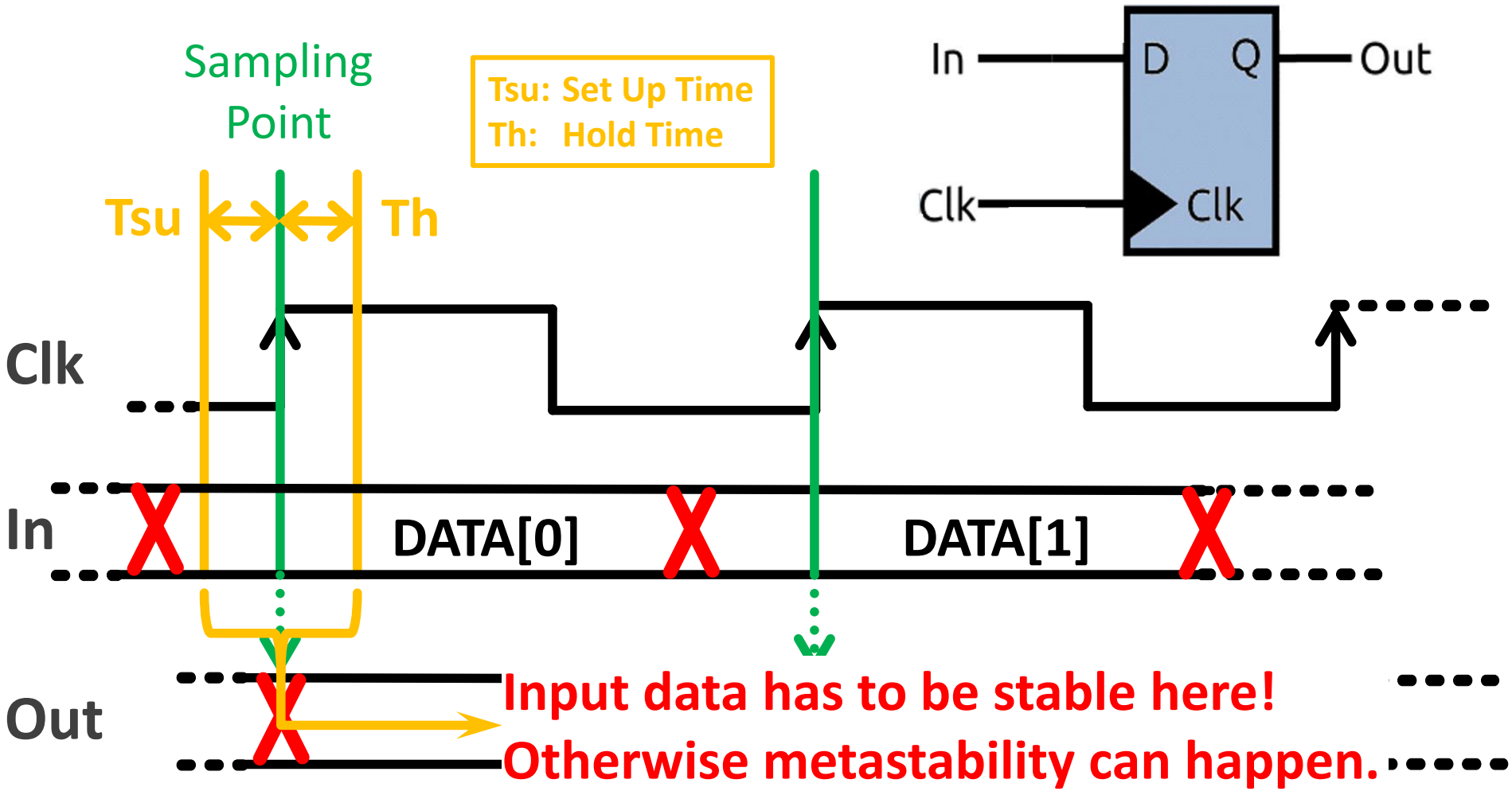


- **Bad practices when designing your clocking scheme**



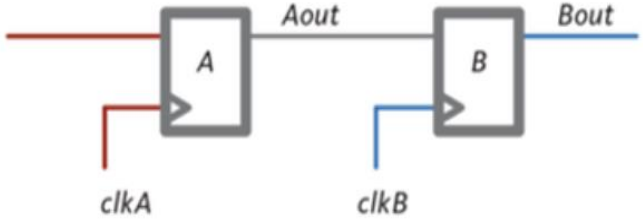


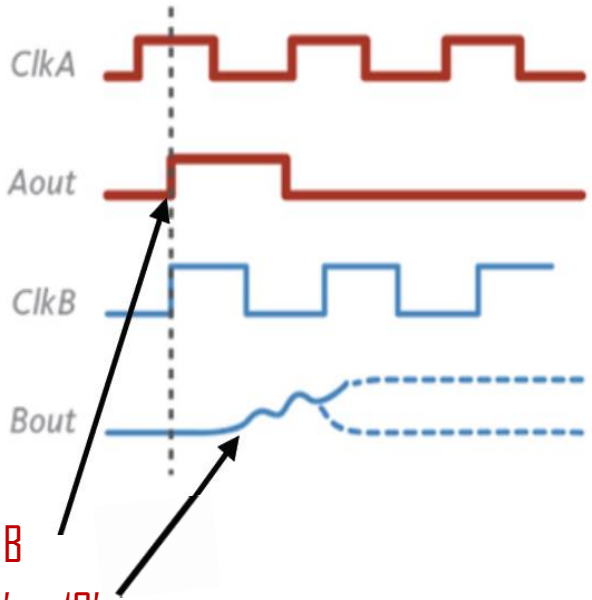
Design Entry: Timing



Design Entry: Timing

- **Clock Domain Crossing (CDC): The problem**
  - Clock Domain Crossing (CDC) : passing a signal from one clock domain to another (A to B)
  - If clocks are unrelated to each other (asynchronous) timing analysis is not possible
  - Setup and Hold times of FlipFlop B are likely to be violated -> Metastability!!!



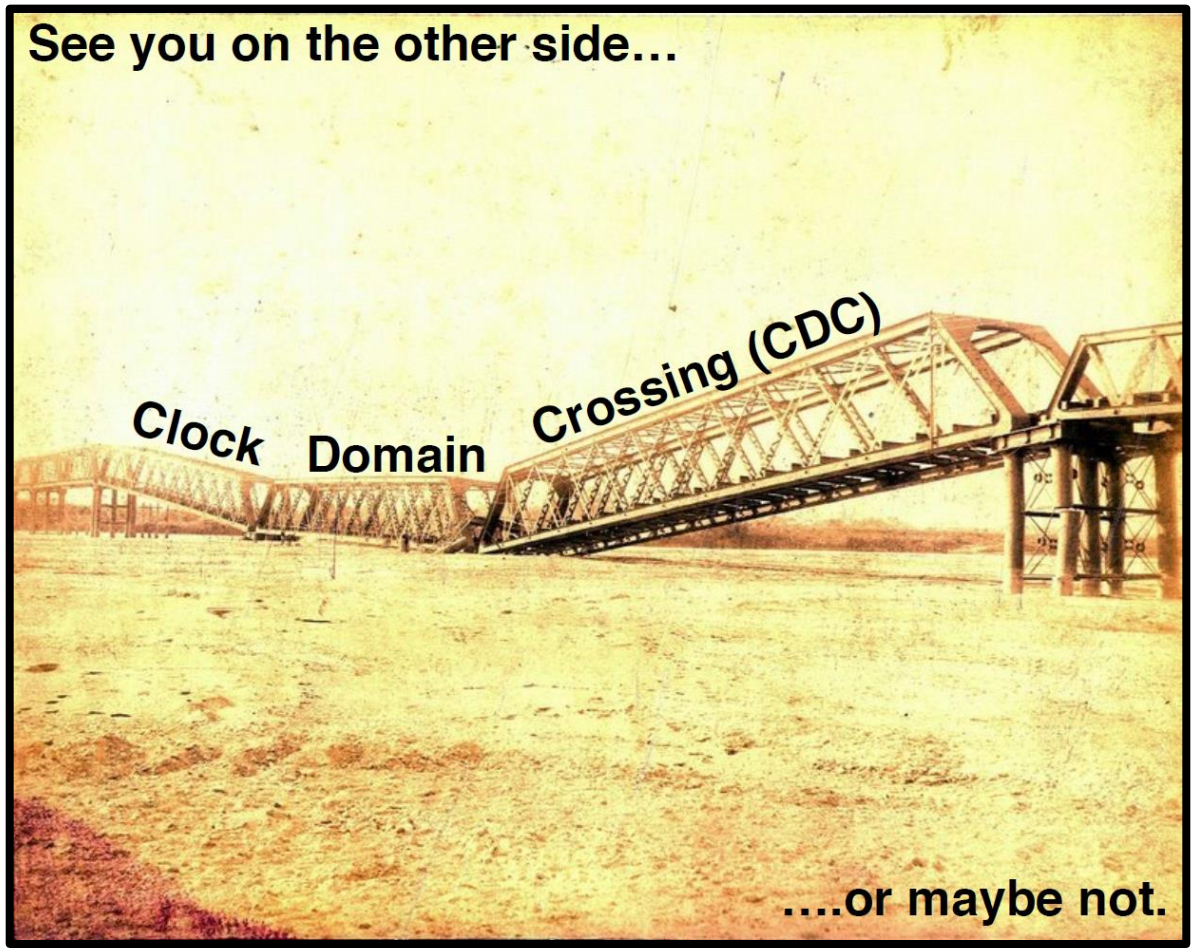


Signal violates the setup-time of FlipFlop B clocked by Clk B

Bout becomes metastable and then settles at either at '1' or '0'

Design Entry: Timing

- **Clock Domain Crossing (CDC)**

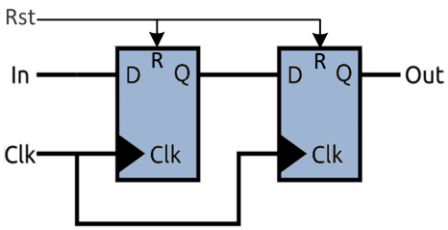


Design Entry: Timing

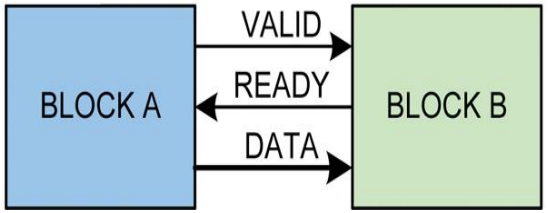
- **Clock Domain Crossing (CDC):** The solution

Avoid creating unnecessary clock domains

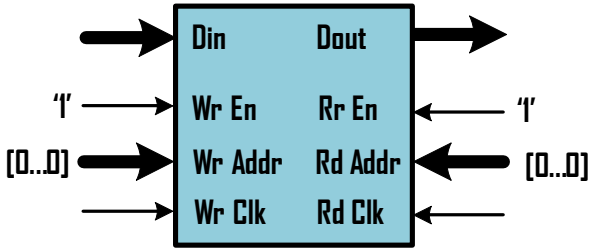
Synchronizers



Handshaking

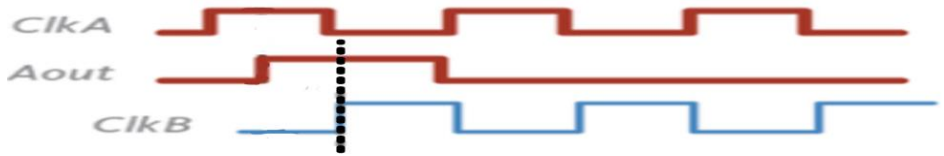


DPRAM



Be aware of FIFO overflow/underflow!!!

Clock phase alignment



Design Entry: Primitives & IP Cores

- **Primitives:** Basic components of the FPGA
  - Vendor (and device) specific
  - E.g. buffers (I/O & clock), registers, BRAMs, DSP blocks…
- **Hard IP Cores:** Complex hardware blocks embedded into the FPGA
  - Vendor (and device) specific
  - Fixed I/O location
  - In many cases they may be set through GUI (wizards)
  - E.g. PLLs, multi-gigabit transceivers, Ethernet MAC, CPU…
- **Soft IP Cores:** Complex (or simple) modules ready to be implemented
  - They may be
    - vendor specific (encryption code, memory controller…) or
    - vendor agnostic (commercial or open source (www.OpenCores.org))
  - In many cases they may be set through GUI (wizards)
- **Two ways of adding Primitives & IP Cores to your system:**
  - Instantiation: The module is EXPLICITLY added to the system
  - Inference: The module is IMPLICITLY added to the system

Add primitives by inference

Add IP cores by instantiation (and use the wizard if possible)

Instantiated FlipFlop (for Microsemi ProAsic3)

```
DFN1C1 FlipFlop (
     .D    (Input_D),
     .CLK  (Clk),
     .CLR  (Rst),
     .Q    (Output_Q));
```

Inferred FlipFlop (Verilog)

```
always @(posedge Clk or posedge Rst)
begin
    if (Rst)
        Output_Q <= 0;
    else
        Output_Q <= Input_D;
    end
```

Synthesis

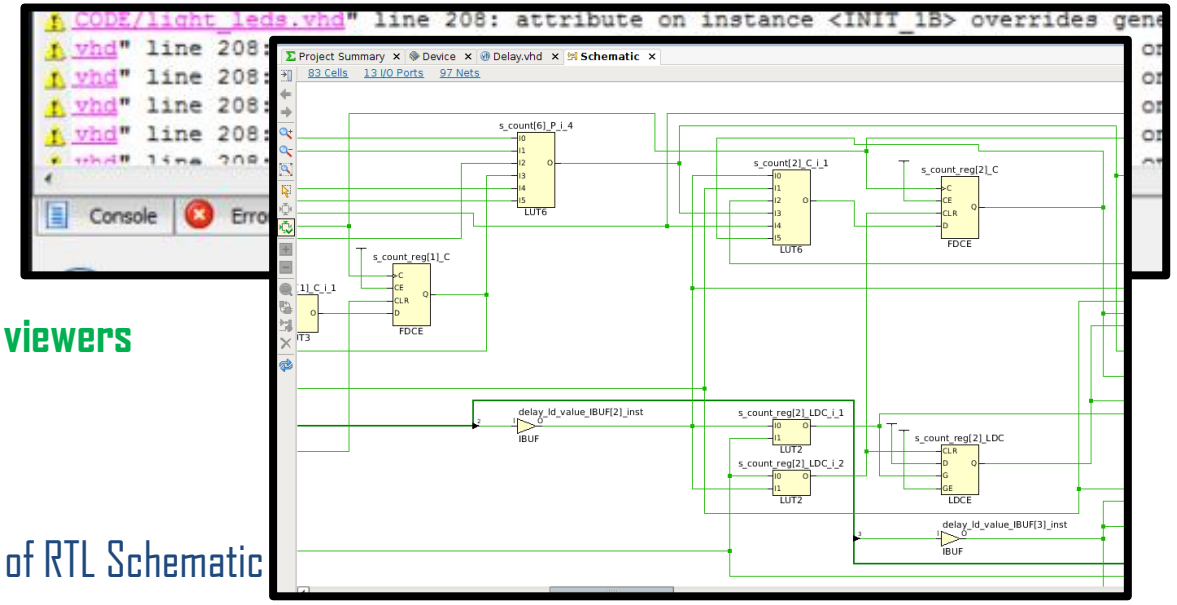
- **What does it do?**
  - Translates the schematic or HDL code into elementary logic functions
  - Defines the connection of these elementary functions

- **The FPGA design tool optimizes the design during synthesis**

  It may do undesired changes to the system (e.g. remove modules, change signal names, etc.)!!!

- **Always check the synthesis report**
  - Warnings & Errors
  - Estimated resource utilization
  - Optimizations
  - And more…

Example of Synthesis Report



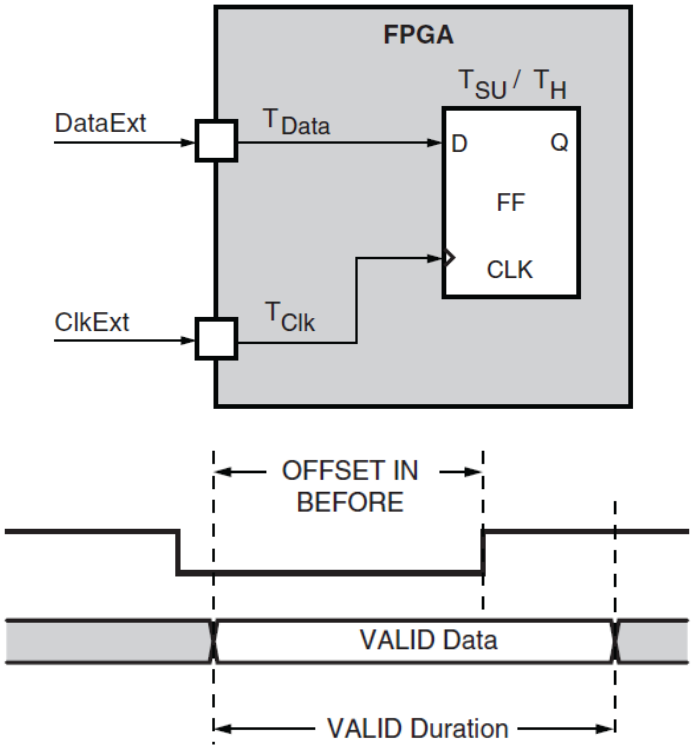
- **And also check the RTL/Technology viewers**

Example of RTL Schematic

Constraints: Timing

- **Timing requirements for all paths must be provided to the FPGA design tool**
- **Defined in a constraint files (Xilinx .XDC, Altera .SDC) – can be generated in GUI**
- **Most common types of constraints**
  - Input paths
  - Output paths
  - Register-to-register paths (combinatorial paths)
  - Path specific exceptions (e.g. false path, multi-cycle paths, etc.)
- **To efficiently specify these constraints:**
  1) Begin with global constraints
  2) Add path specific exceptions as needed (only for special cases)
- **Over-constrained system is difficult to route**

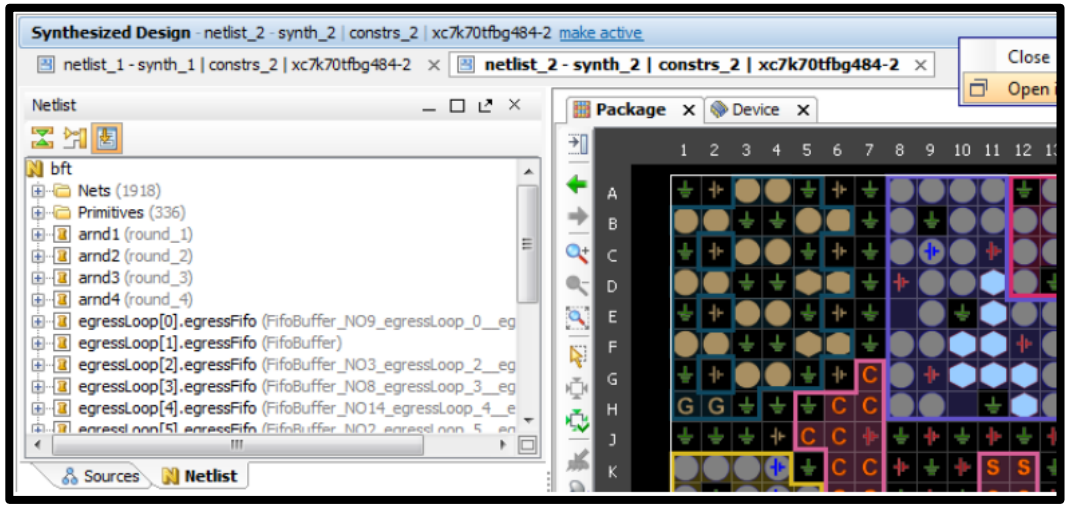
Timing constraint (Xilinx .ucf)




```
TIMEGRP DATA_IN OFFSET = IN 1 VALID 3 BEFORE CLK RISING;
```

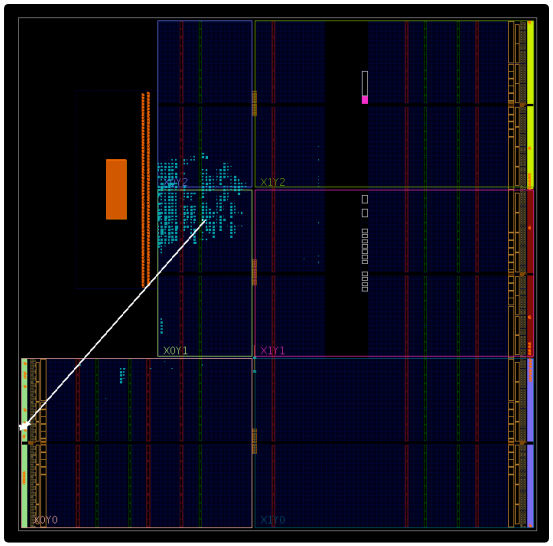
Constraints: Physical

- **Pin planning**



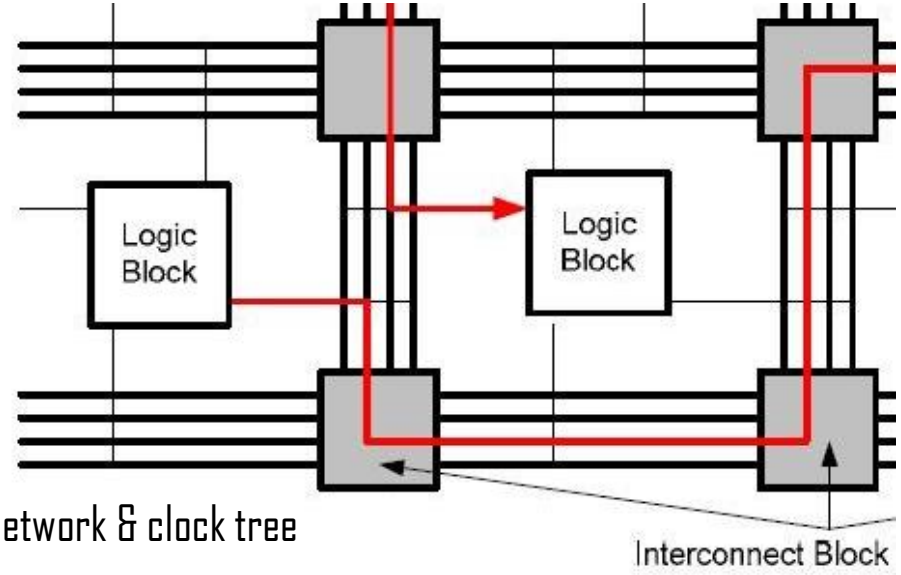
As previously mentioned...
You should prepare Pin Planning during Specification Stage

- **Floorplanning**
  - Only when really needed
  - To place logic close to their related I/O pins
  - To avoid routing across the chip
  - Can improve timing (faster system speed)
  - Over-constrained system is difficult to route



Implementation

- **The FPGA design tool:**
  1) Translates timing and physical constraints in order to guide the implementation
  2) Maps the synthesized netlist:
     - Logic elements to FPGA logic cells
     - Hard IP cores to FPGA hard blocks
     - Verifies that the design can fit the target device
  3) Places and Routes (P&R) the mapped netlist:
     - Physical placement of the FPGA logic cells
     - Physical placement of the FPGA hard blocks
     - Routing of the signals through the interconnect network & clock tree

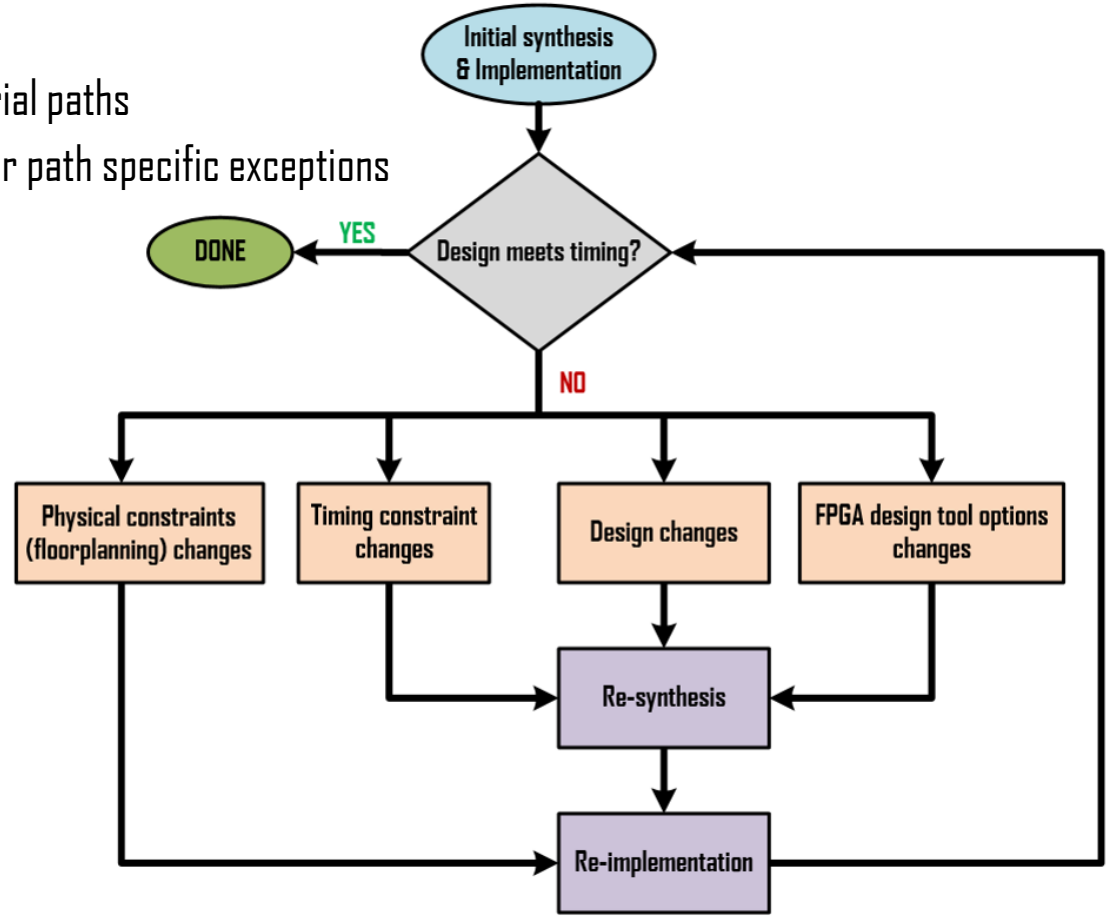


- **The FPGA design tool may be set for different optimizations (Speed, Area, Power or none)**
- **Physical placement and timing change after re-implementing (use constraints to minimize these changes)**
- **You should always check the different reports generated during implementation**

Static Timing Analysis

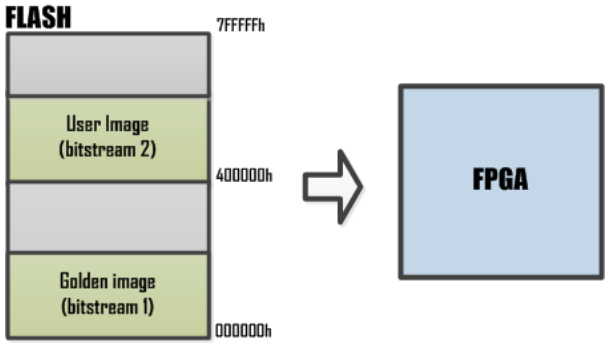
- **The FPGA design tool analyses the signals propagation delays and clock relationships after implementation**
- **A timing report is generated, including the paths that did not meet the timing requirements**
- **Rule of thumb for timing violations:**

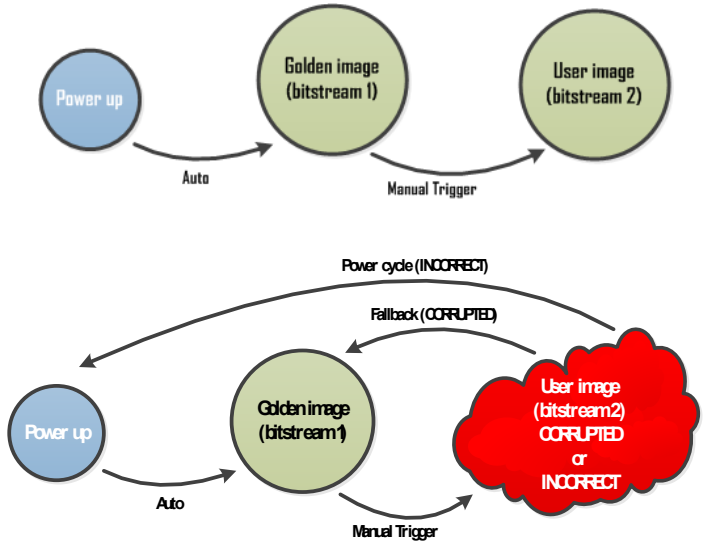
  - Setup violations: Too long combinatorial paths
  - Hold violations: Issue with CDC and/or path specific exceptions
- **The timing closure flow:**



Bitstream Generation & FPGA Programming

- **Bitstream:**
  - Binary file containing the FPGA configuration data
  - Each FPGA vendor has its own bitstream file format (e.g. .bit (Xilinx), .sof (Altera))
- **FPGA programming:**
  - Bitstream is loaded into the FPGA through JTAG or SPI
  - Configuration data may be stored in on-board FLASH and loaded by the FPGA at power up
  - Remote programming (e.g. through Ethernet)
  - Multiboot/Safe FPGA configuration

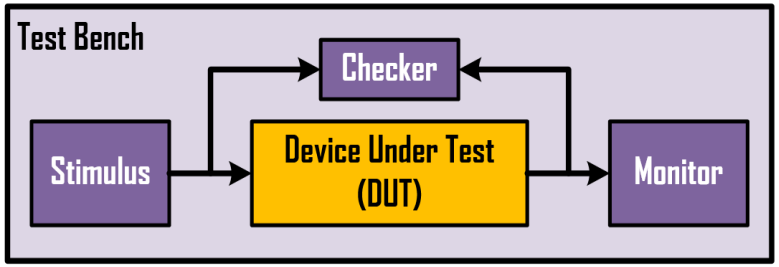




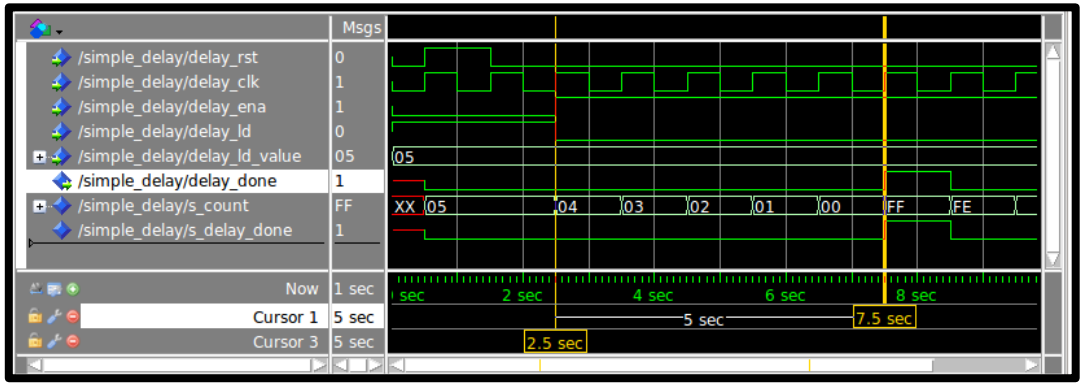
Multiboot/Safe FPGA configuration diagrams

Simulation

- **Event-based simulation to recreate the parallel nature of digital designs**
- **Verification of individual modules or full systems**
- **HDL simulators:**
  - Most popular: Modelsim
  - Other simulators: Vivado Simulator (Xilinx), Icarus Verilog (Open-source)…
- **Different levels of simulation**
  - Behavioral: simulates only the behavior of the design — **Fast**
  - Functional: uses realistic functional models for the target technology — **Slow**
  - Timing: most accurate, uses Implemented design — **Very Slow**



Simulator wave window



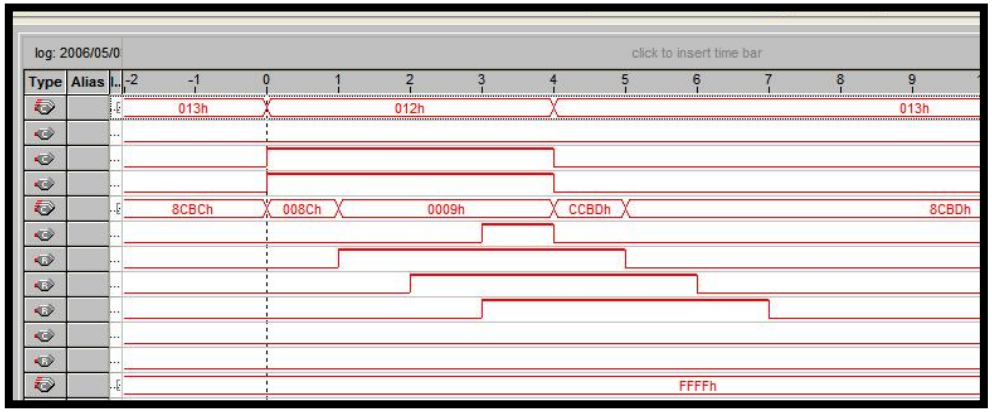
In-System Analyzers & Virtual I/Os

- Your design is up… and also running?
- Most FPGA vendors provide in-system analyzers & virtual I/Os
- Can be embedded into the design and controlled by JTAG
- Allow monitoring but also controlling the FPGA signals
- Minimize interfering with your system by:

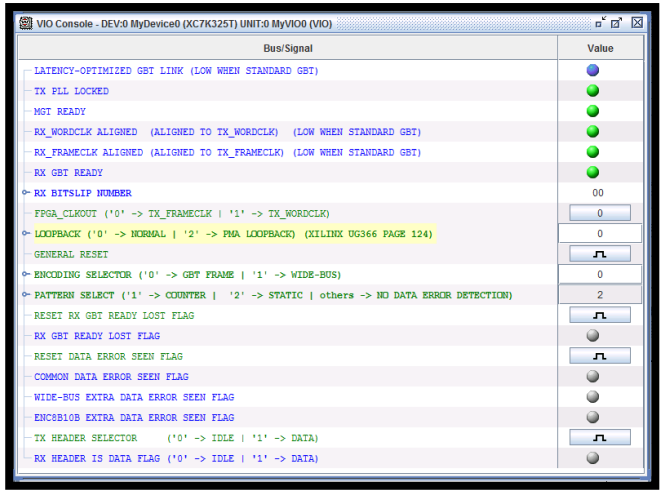
Placing extra registers between the monitored signals and the In-System Analyser

- It is useful to spy inside the FPGA… but the issue may come from the rest of the board!!!
- Remember… it is HARDWARE

In-System Analyser (Altera SignalTap II)

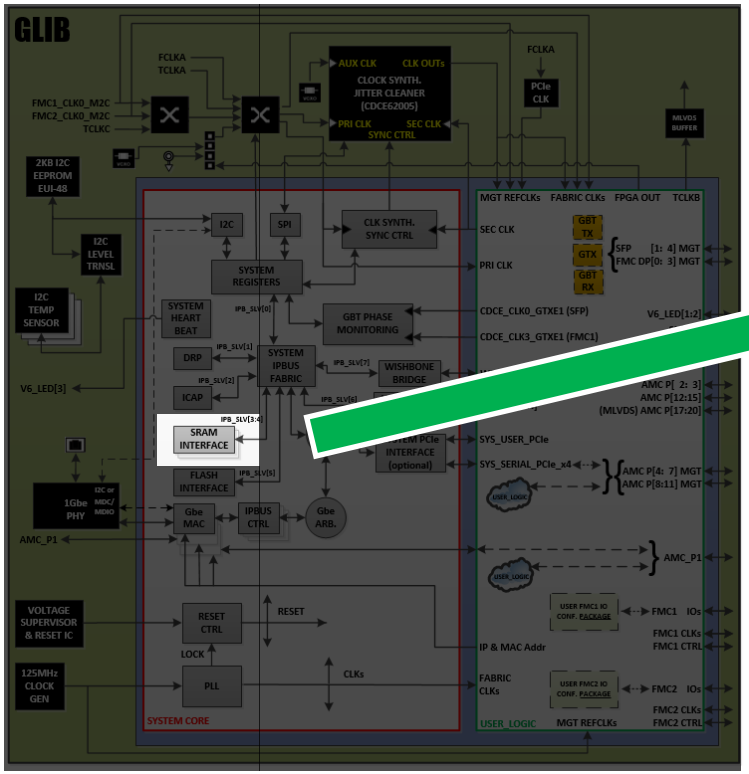


Virtual I/Os (Xilinx VIO)

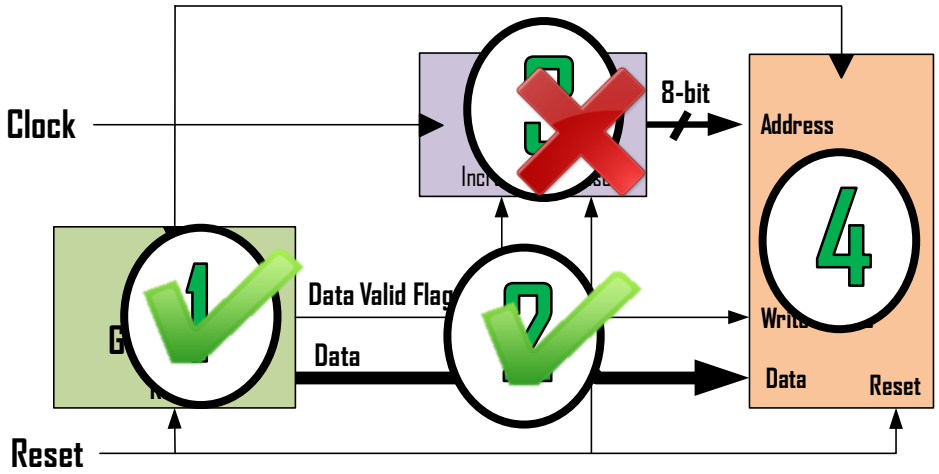


Debugging Techniques

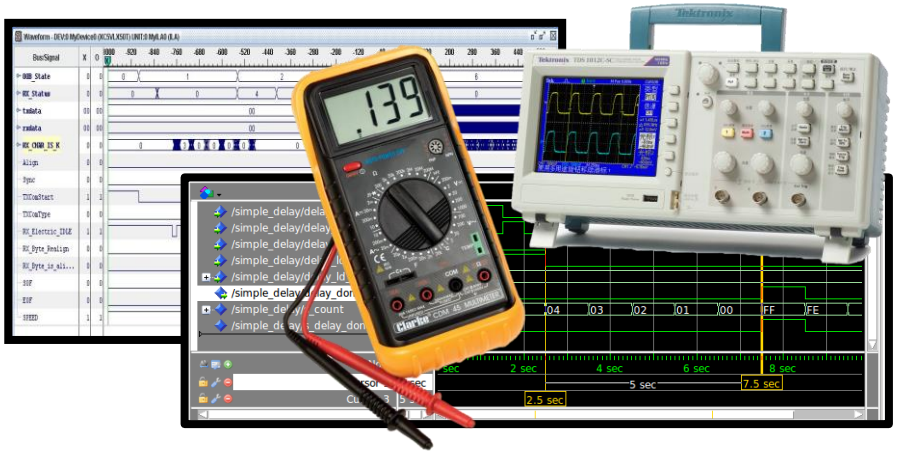
Divide & Conquer



Follow the chain



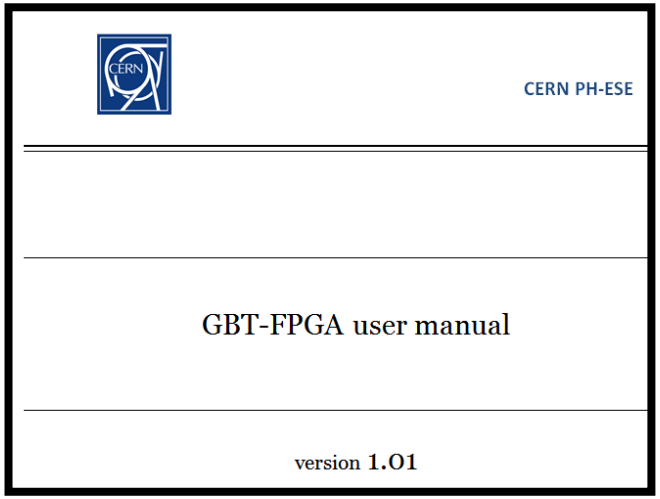
"Open the box"



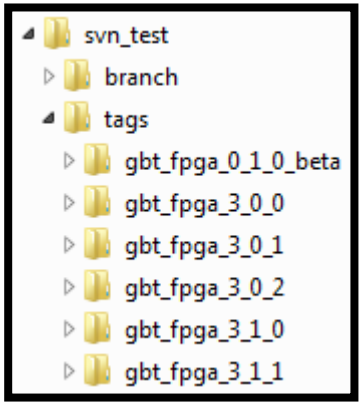
We are debugging HARDWARE

After debugging…

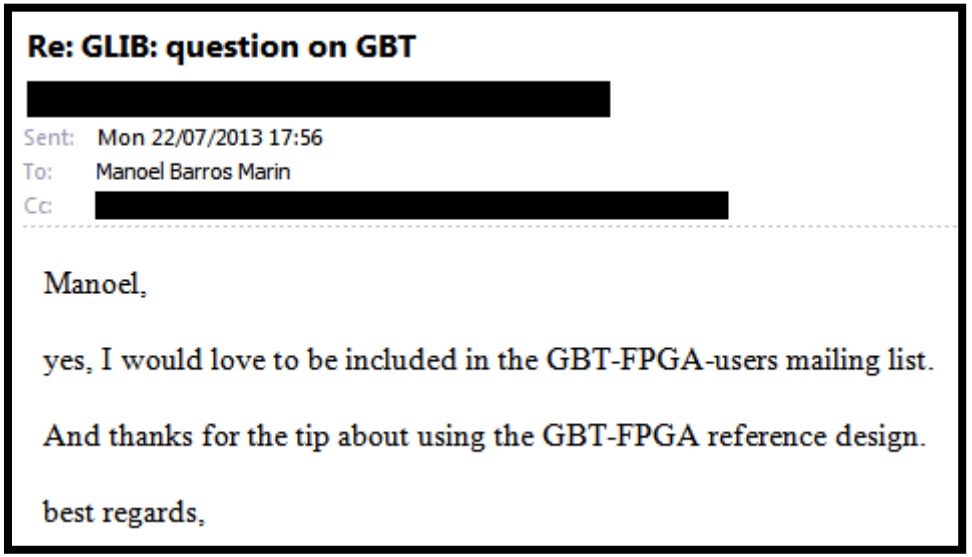
- **Documentation**



- **Maintenance**



- **… and maybe User Support**



Outline

… from the previous lesson

Key concepts about FPGA design

FPGA gateware design work flow

Summary

What to Remember

- FPGA gateware design is not a programming
- HDL are used for describing hardware
- Timing is critical in FPGA gateware design

What to Remember – GW Flow

- Plan, plan and plan again
- Modular and reusable system
- Coding for synthesis
- Resets and clocks schemes
- Clock Domain Crossing
- Constraint the design properly
- Read all the reports
- Be methodic when debugging & use all tools available
- A running system is not the end of the road… (documentation, maintenance, user support)

Additional Resources

- There are nice papers & books but…
- FPGA vendors provide very good documentation about all topics mentioned in this lecture

Any questions?

Jan Pospíšil
j.pospisil@cern.ch