

Advanced FPGA design

ISOTDAQ 2020 @ University of Valencia, Valencia (Spain)

22/01/2020

Manoel Barros Marin



BE-BI-BP

Advanced FPGA design

ISOTDAQ 2020 @ University of Valencia, Valencia (Spain)

22/01/2020

Outline:

- ... from the previous lesson
- Key concepts about FPGA design
- FPGA gateway design work flow
- Summary

Manoel Barros Marin



BE-BI-BP

Advanced FPGA design

ISOTDAQ 2020 @ University of Valencia, Valencia (Spain)

22/01/2020

Outline:

- **... from the previous lesson**
- Key concepts about FPGA design
- FPGA gateware design work flow
- Summary

Manoel Barros Marin



BE-BI-BP

... from the previous lesson

What is an Field Programmable Gate Array (FPGA)?

... from the previous lesson

What is an Field Programmable Gate Array (FPGA)?

FPGA - Wikipedia

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

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".

... from the previous lesson

What is an Field Programmable Gate Array (FPGA)?

FPGA - Wikipedia

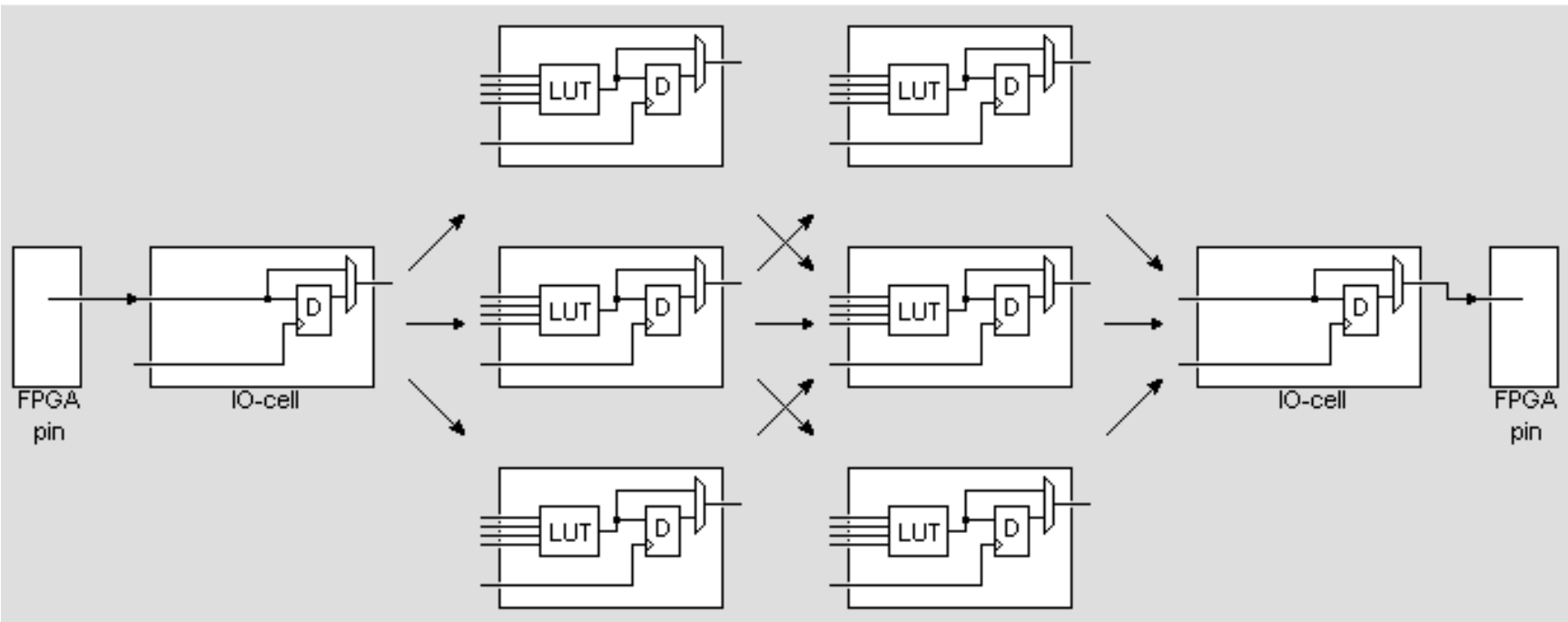
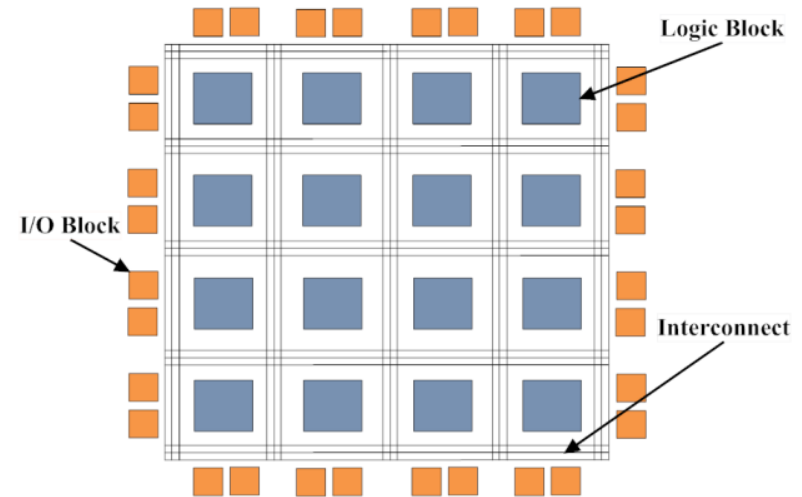
https://en.wikipedia.org/wiki/Field-programmable_gate_array

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".



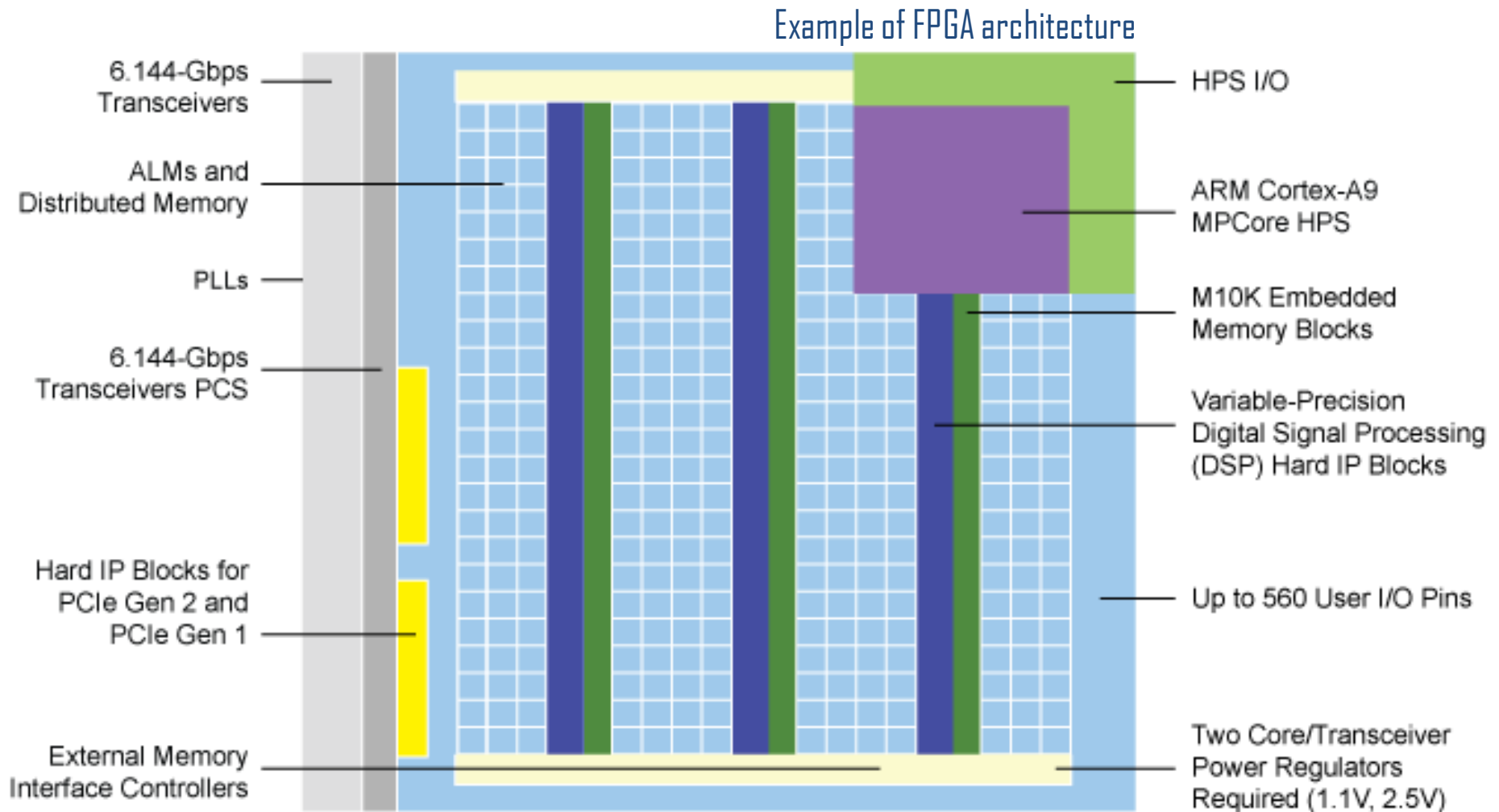
... from the previous lesson

- **FPGA fabric (matrix like structure) made of:**
 - I/O-cells to communicate with outside world
 - Logic cells
 - Look-Up-Table (LUT) to implement combinatorial logic
 - Flip-Flops (D) to implement sequential logic
 - Interconnect network between logic resources
 - Clock tree to distribute the clock signals



... from the previous lesson

- But it also features Hard Blocks:



Advanced FPGA design

ISOTDAQ 2020 @ University of Valencia, Valencia (Spain)

22/01/2020

Outline:

- ...from the previous lesson
- **Key concepts about FPGA design**
- FPGA gateware design work flow
- Summary

Manoel Barros Marin



BE-BI-BP

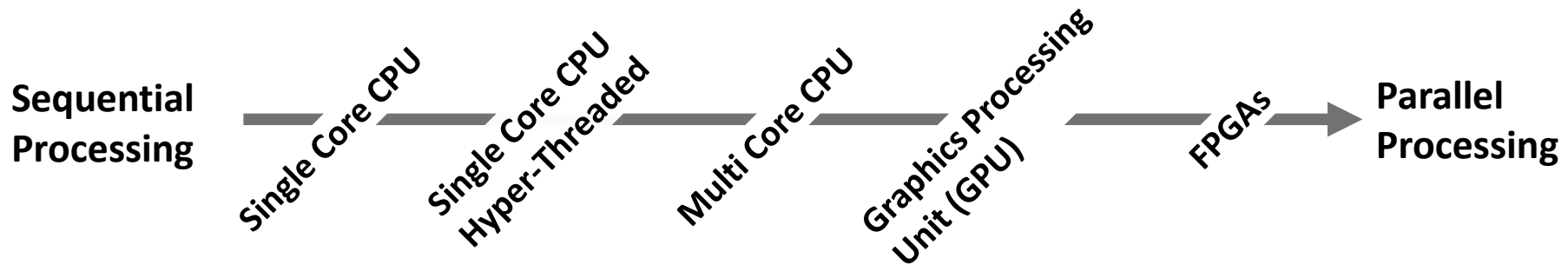
Key concepts about FPGA design

FPGA gateware design is NOT programming



Key concepts about FPGA design

FPGA gateware design is NOT programming



- **Programming**

- Code is written and translated into instructions
- Instructions are executed sequentially by the CPU(s)
- Parallelism is achieved by running instructions on multiple threads/cores
- Processing structures and instructions sets are fixed by the architecture of the system

VS.

- **FPGA gateware design**

- No fixed architecture, the system is built according to the task
- Building is done by describing/defining system elements and their relations
- Intrinsicly parallel, sequential behaviour is achieved by registers and Finite-State-Machines (FSMs)
- Description done by schematics or a hardware description language (HDL)

Key concepts about FPGA design

HDL are used for describing HARDWARE



Key concepts about FPGA design

HDL are used for describing HARDWARE



- Example of a WAIT statement (Programming Language VS. HDL)

Key concepts about FPGA design

HDL are 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`

Key concepts about FPGA design

HDL are 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
```



Key concepts about FPGA design

HDL are 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/or FPGA implementation)



```
simple_delay_counter : process (delay_rst, delay_clk, delay_ena)
begin -- process
  if delay_rst = '1' then
    s_count    <= delay_ld_value;
    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;
```


Key concepts about FPGA design

HDL are 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/or FPGA implementation)



```
simple_delay_counter : process (delay_rst, delay_clk, delay_ena)
begin -- process
  if delay_rst = '1' then
    s_count    <= delay_ld_value;
    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

Key concepts about FPGA design

HDL are 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)

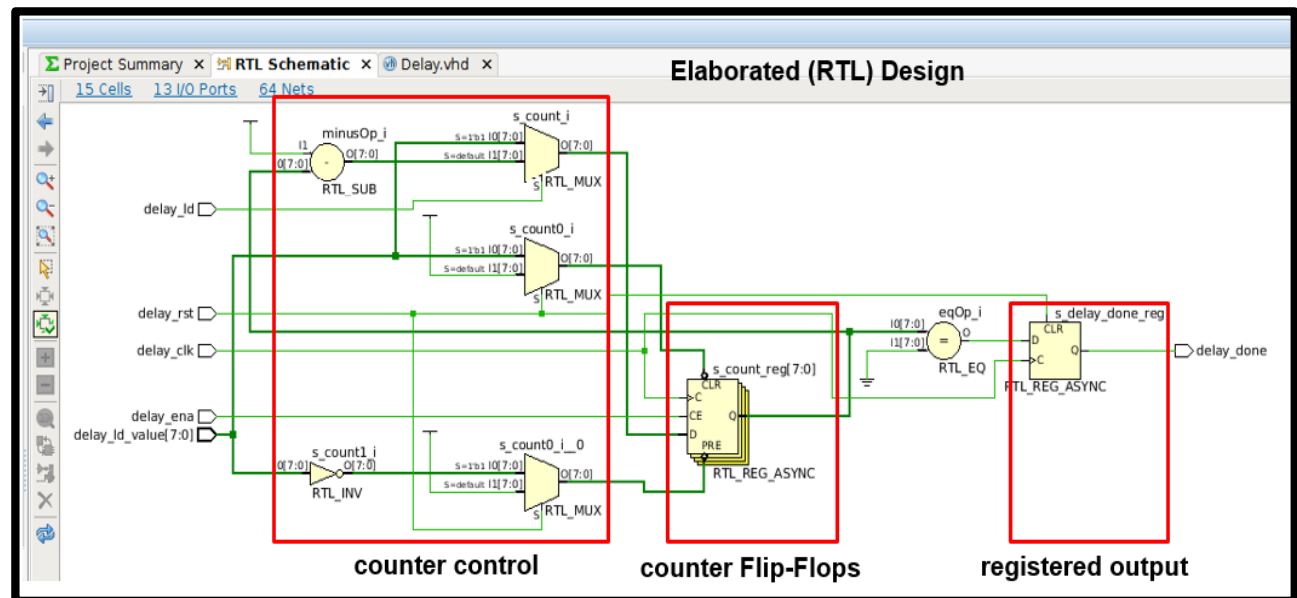
```
wait for 5 sec; -- handy for TB clocks
```

- Synthesizable (for simulation and/or FPGA implementation)



```
simple_delay_counter : process (delay_rst, delay_clk, delay_ena)
begin -- process
  if delay_rst = '1' then
    s_count <= delay_ld_value;
    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;
```

HDL to RTL



Key concepts about FPGA design

HDL are 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)

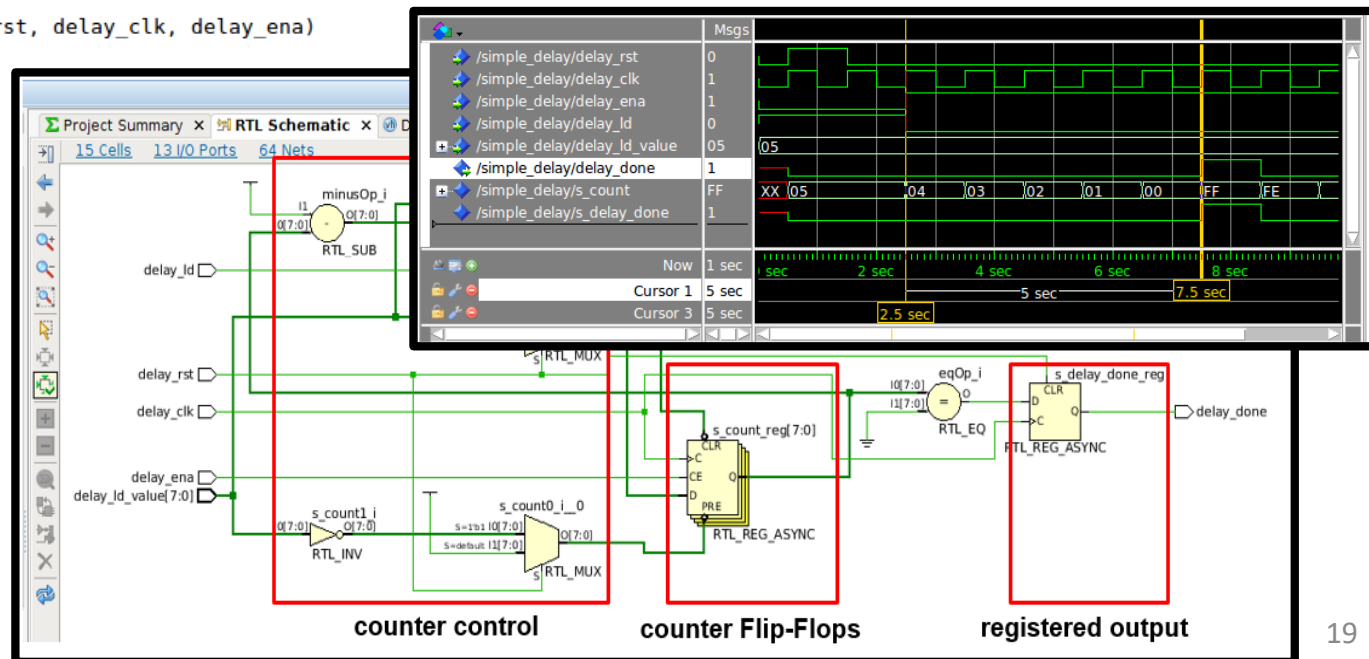
```
wait for 5 sec; -- handy for TB clocks
```

- Synthesizable (for simulation and/or FPGA implementation)



```
simple_delay_counter : process (delay_rst, delay_clk, delay_ena)
begin -- process
  if delay_rst = '1' then
    s_count <= delay_ld_value;
    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;
```

HDL to RTL



Key concepts about FPGA design

HDL are 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)

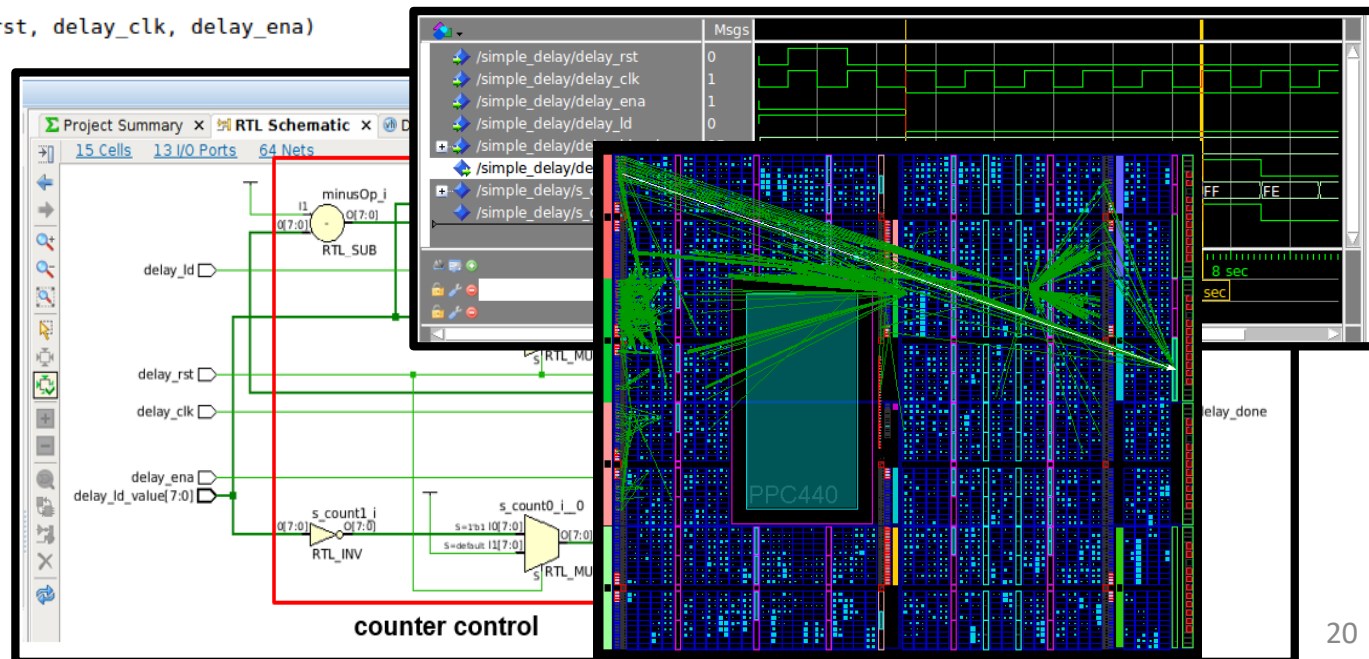
```
wait for 5 sec; -- handy for TB clocks
```

- Synthesizable (for simulation and/or FPGA implementation)



```
simple_delay_counter : process (delay_rst, delay_clk, delay_ena)
begin -- process
  if delay_rst = '1' then
    s_count <= delay_ld_value;
    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 if;
end process;
```

HDL to RTL



Key concepts about FPGA design

Timing in FPGA gateway design is critical

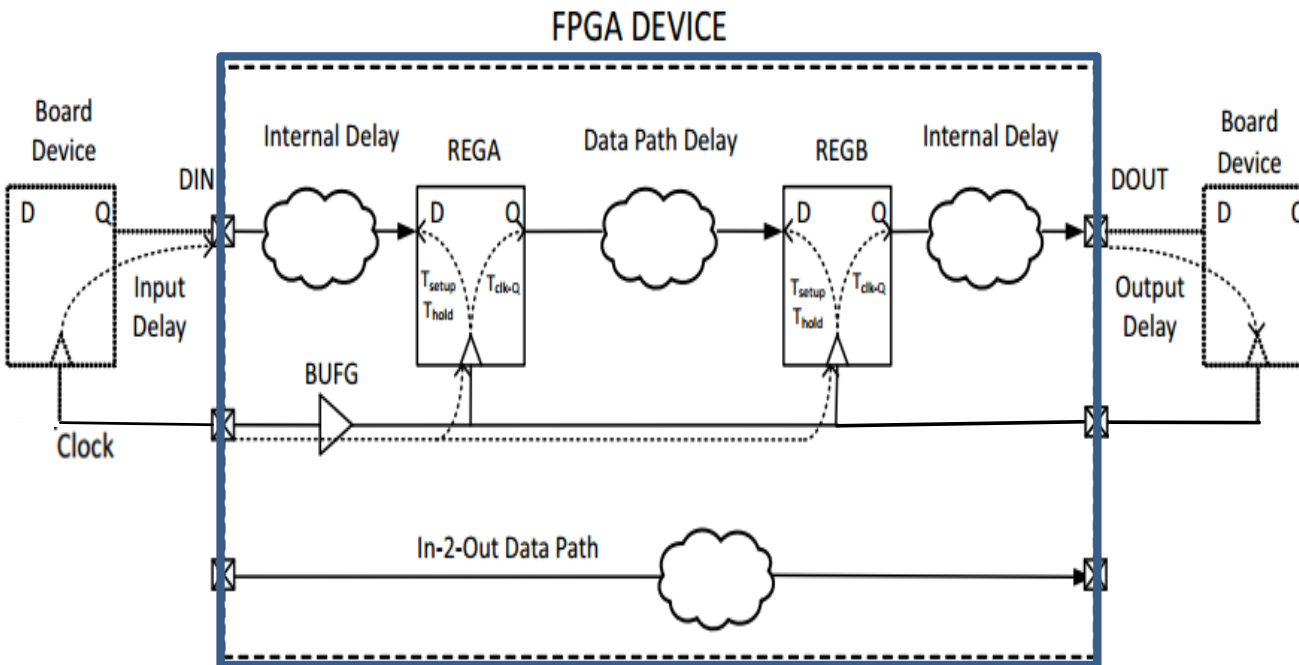


Key concepts about FPGA design

Timing in FPGA gateway design is critical



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

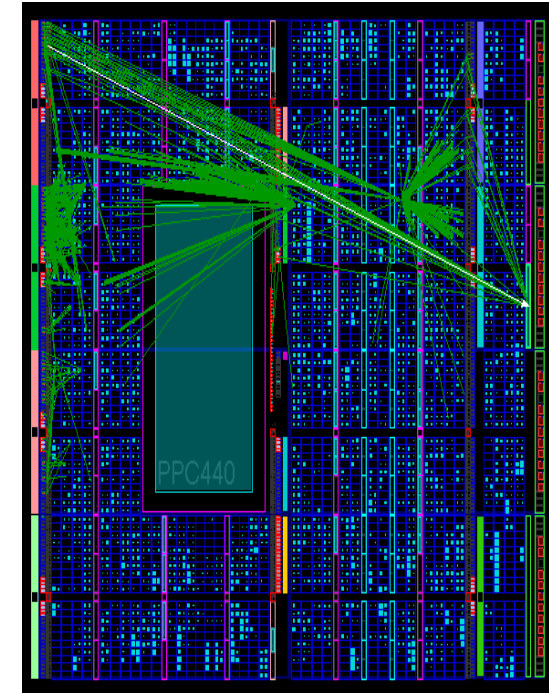
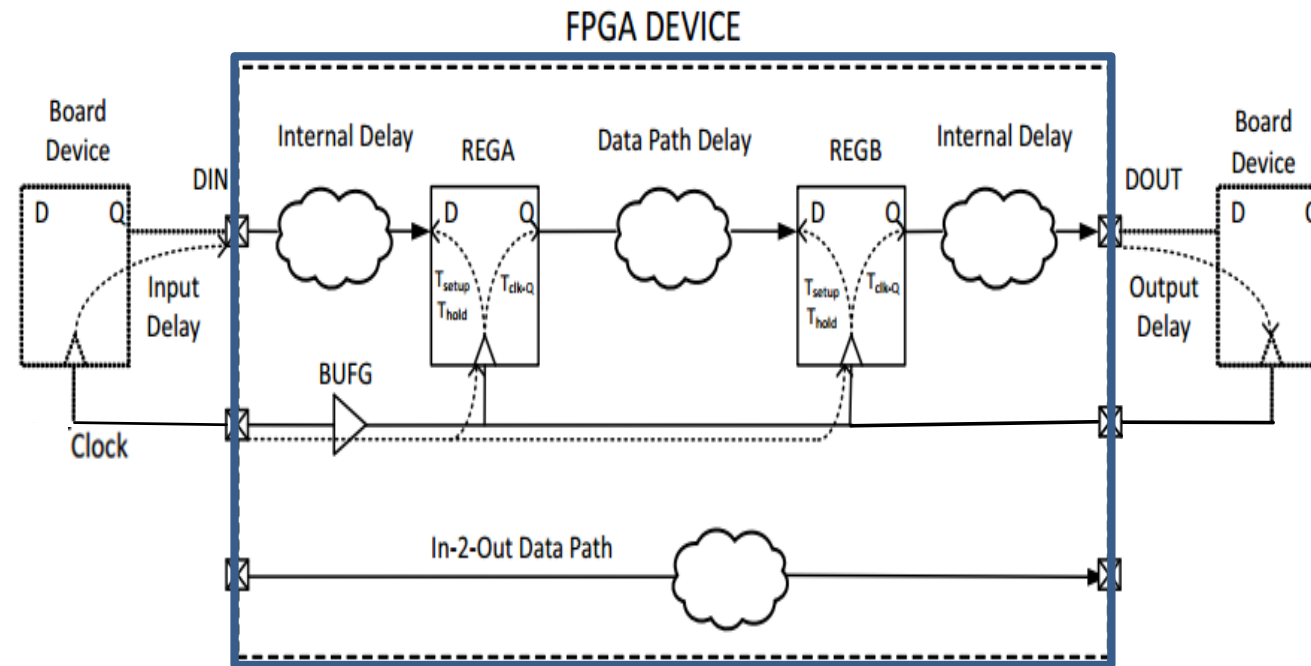


Key concepts about FPGA design

Timing in FPGA gateway design is critical



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



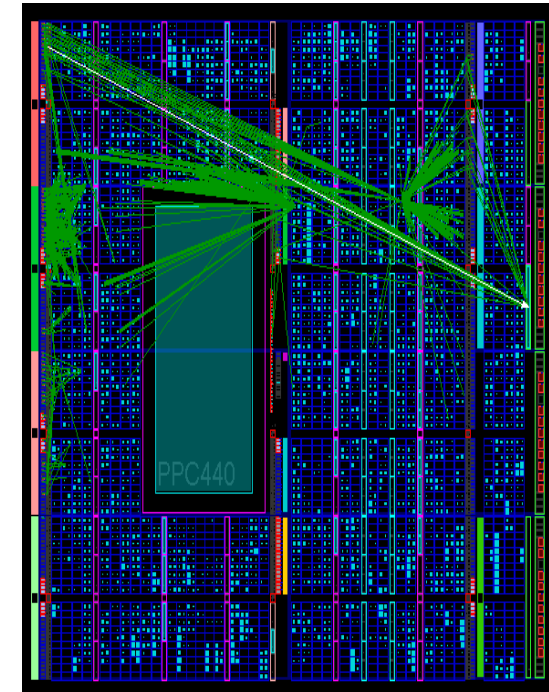
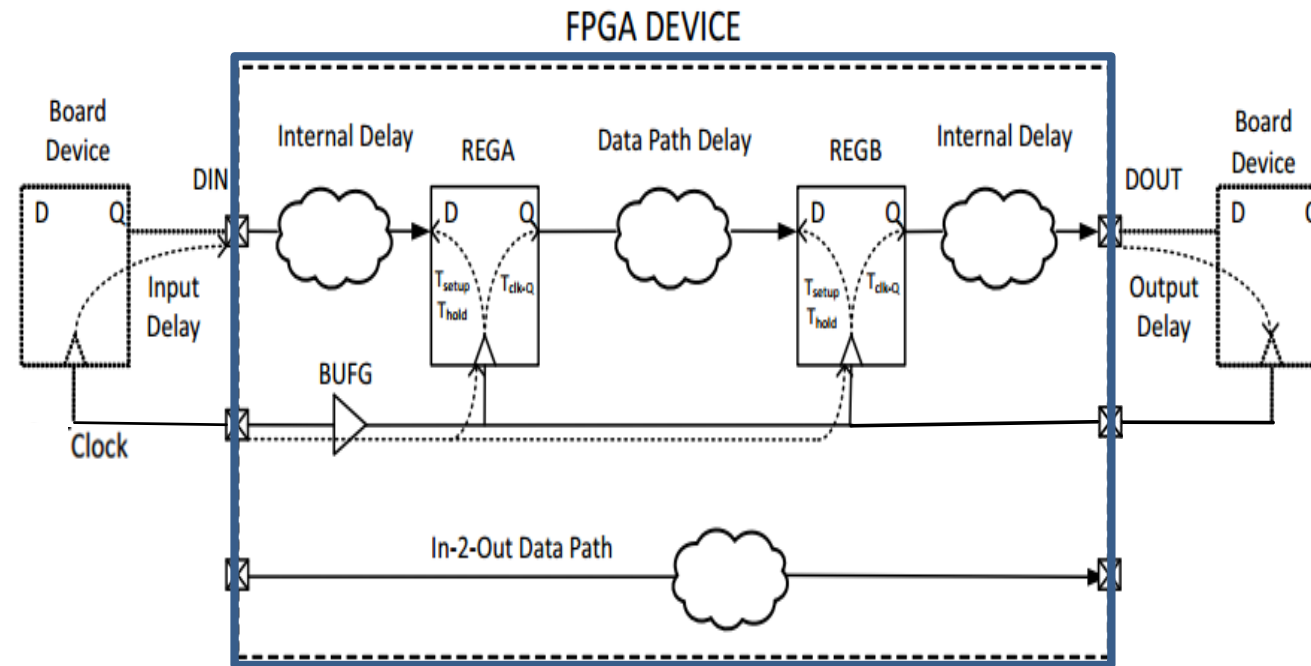
Synthesized RTL (Netlist) is implemented into FPGA

Key concepts about FPGA design

Timing in FPGA gateway design is critical



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



Synthesized RTL (Netlist) is implemented into FPGA

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

The consequences may be catastrophic!!!

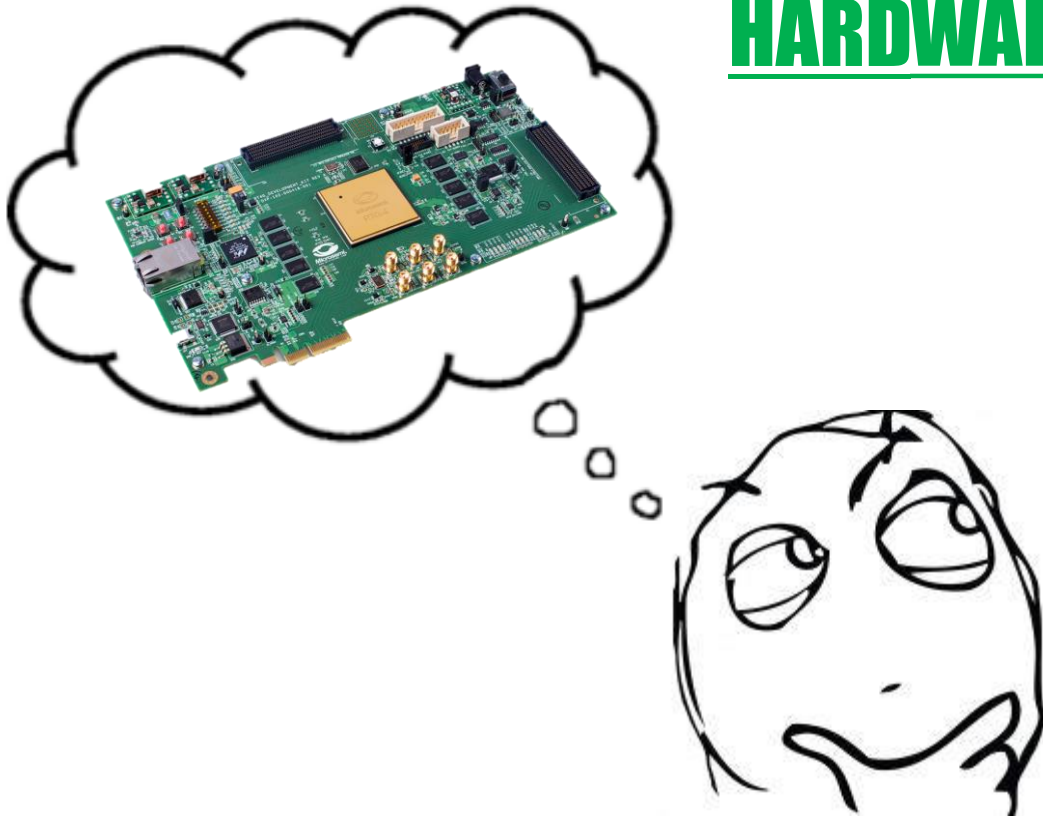
Key concepts about FPGA design

When designing FPGA gateware you have to think HARD...



Key concepts about FPGA design

When designing FPGA gateware you have to think **HARDWARE**



Advanced FPGA design

ISOTDAQ 2020 @ University of Valencia, Valencia (Spain)

22/01/2020

Outline:

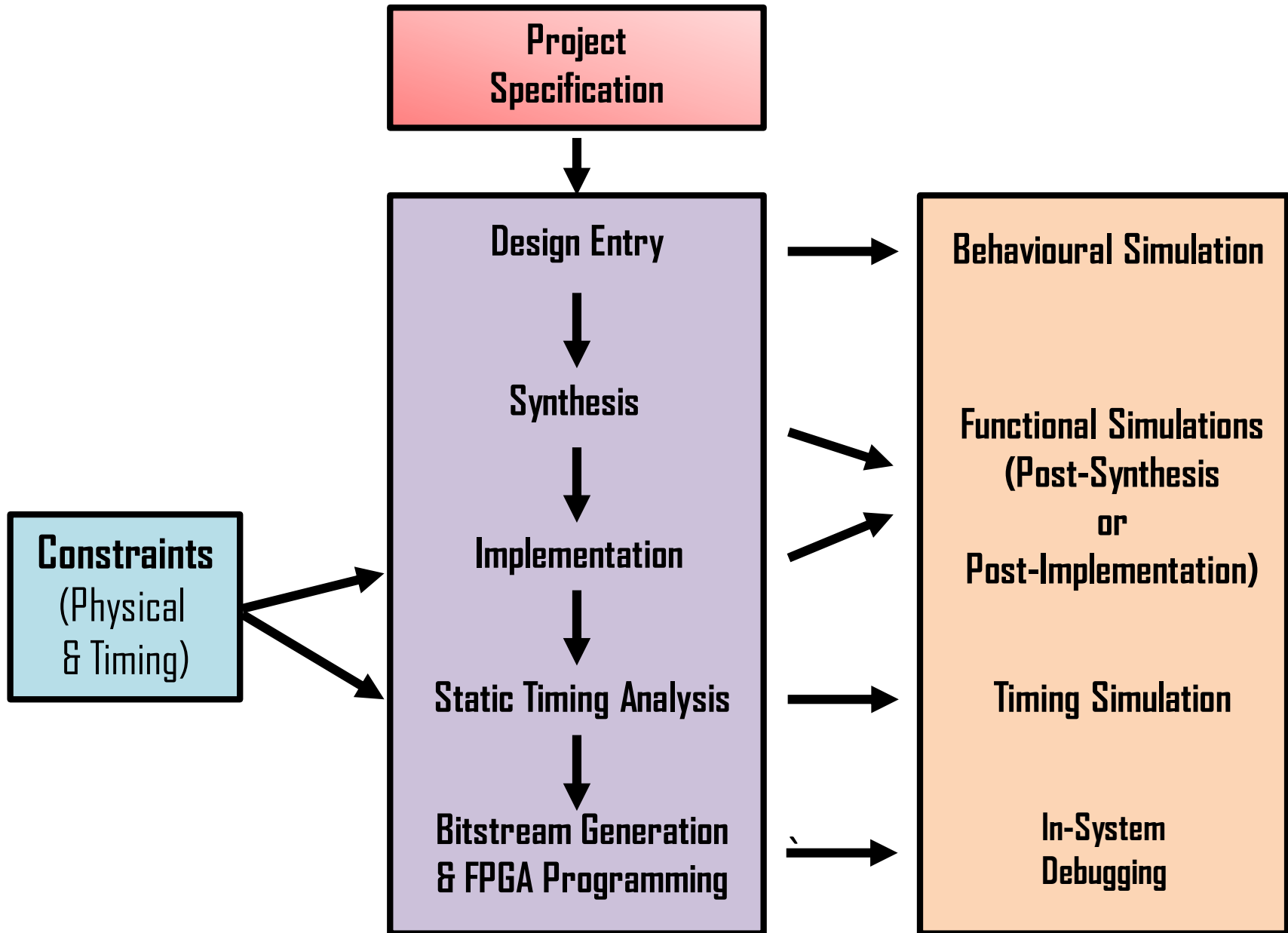
- ...from the previous lesson
- Key concepts about FPGA design
- **FPGA gateway design work flow**
- Summary

Manoel Barros Marin



BE-BI-BP

FPGA gateway design work flow



FPGA gateway design work flow

Project Specification

This is the most critical step...

The rest of the design process is based on it!!!



FPGA gateway design work flow

Project Specification

This is the most critical step...

The rest of the design process is based on it!!!



FPGA gateway design work flow

Project Specification

This is the most critical step...

- Gather requirements from the users
- The rest of the design process is based on it!!!

FPGA gateway design work flow

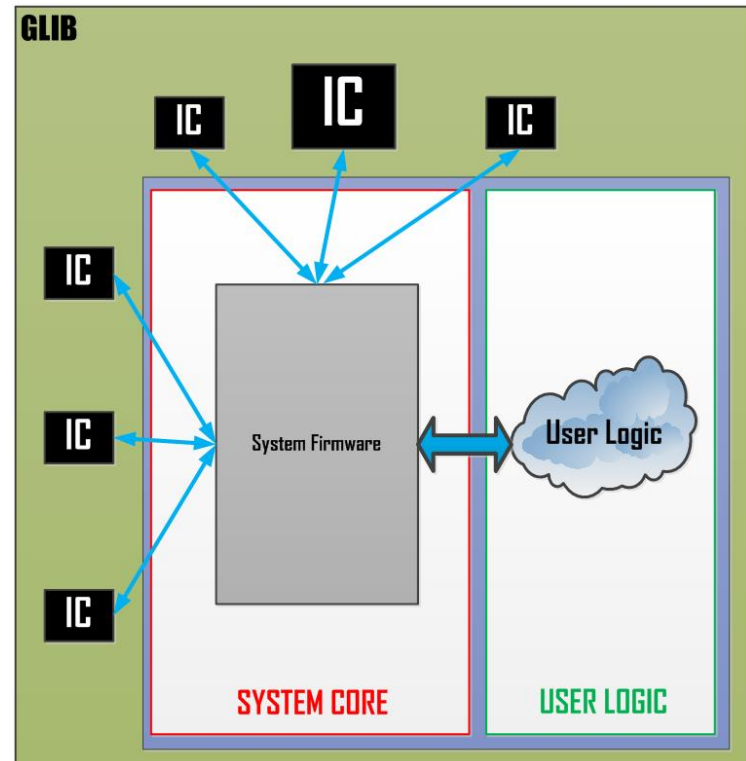
Project Specification

This is the most critical step...

- Gather requirements from the users
- Specify:
 - Target application (General purpose or Specific)

The rest of the design process is based on it!!!

Example of General Purpose Gateway



FPGA gateway design work flow

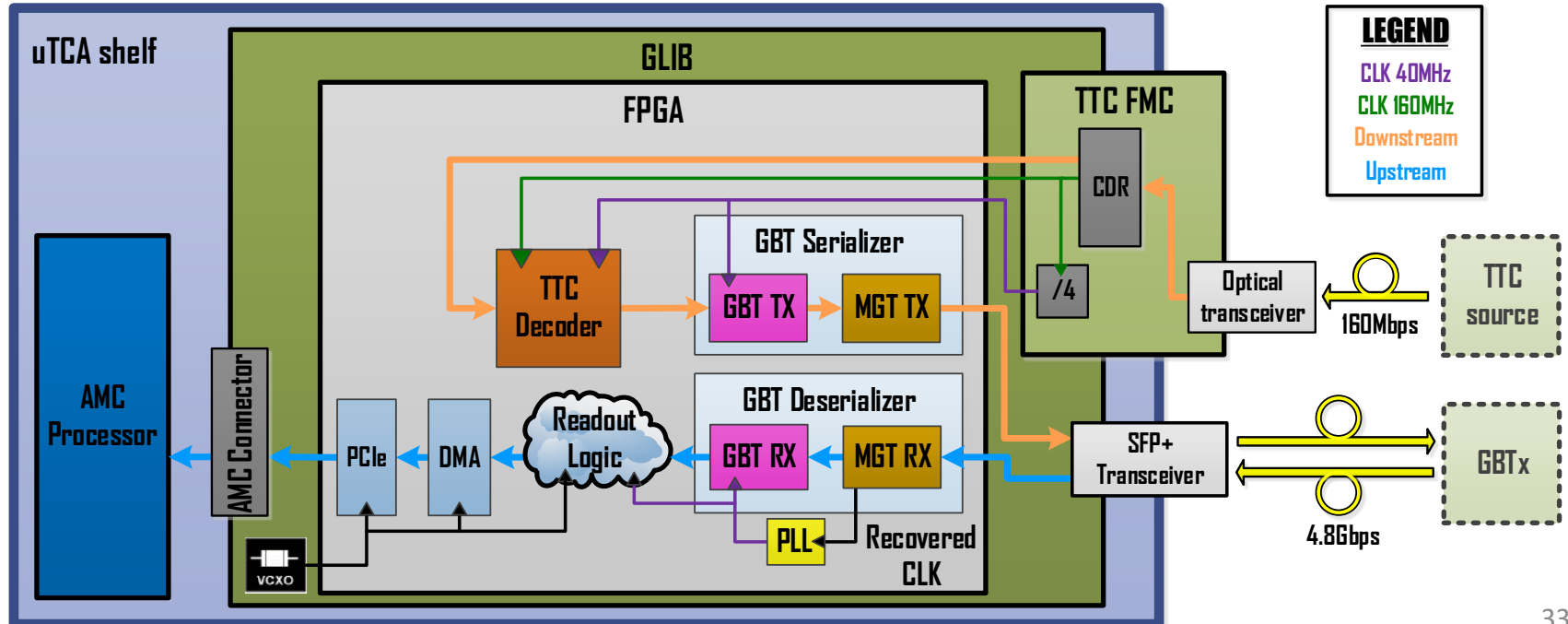
Project Specification

This is the most critical step...

- Gather requirements from the users
- Specify:
 - Target application (General purpose or Specific)

The rest of the design process is based on it!!!

Example of Application Specific Gateway

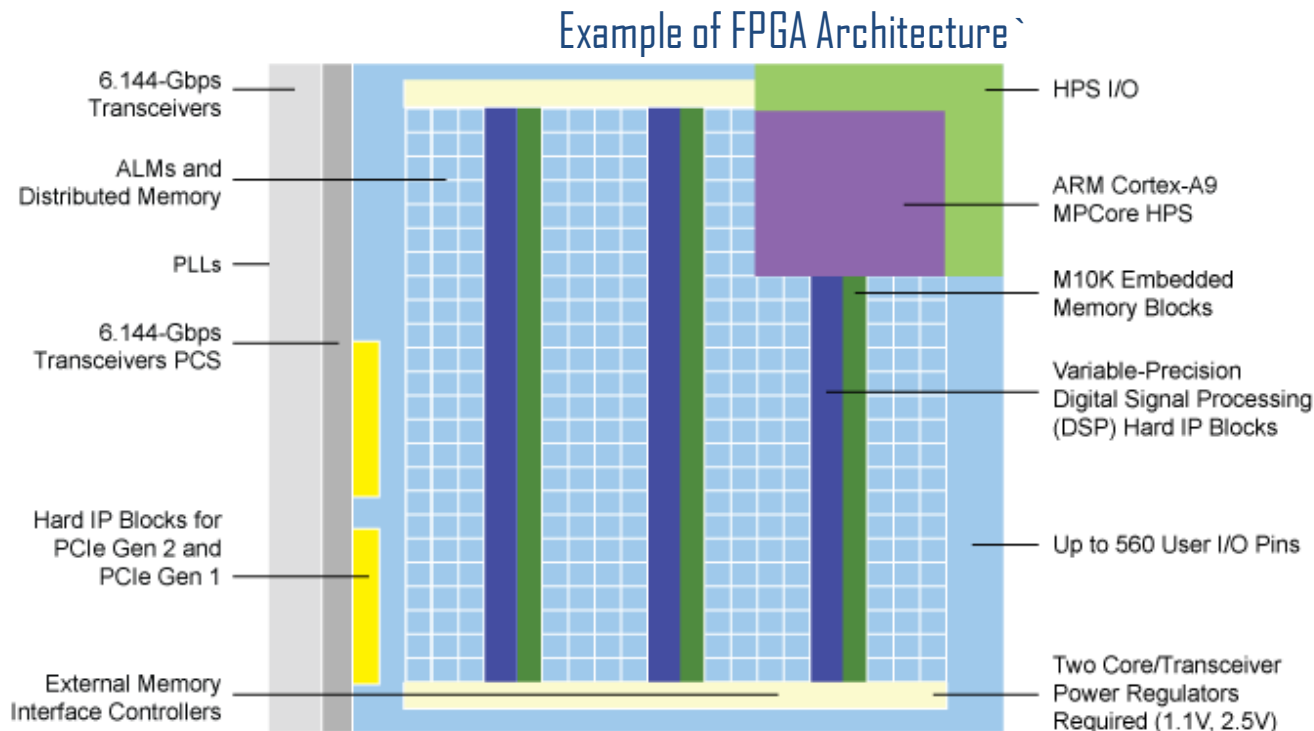


FPGA gateway design work flow

Project Specification

This is the most critical step...

- Gather requirements from the users
 - Specify:
 - Target application (General purpose or Specific)
 - Main features (e.g. System bus, SoC, Multi-gigabit transceivers, etc.)
- The rest of the design process is based on it!!!



FPGA gateway design work flow

Project Specification

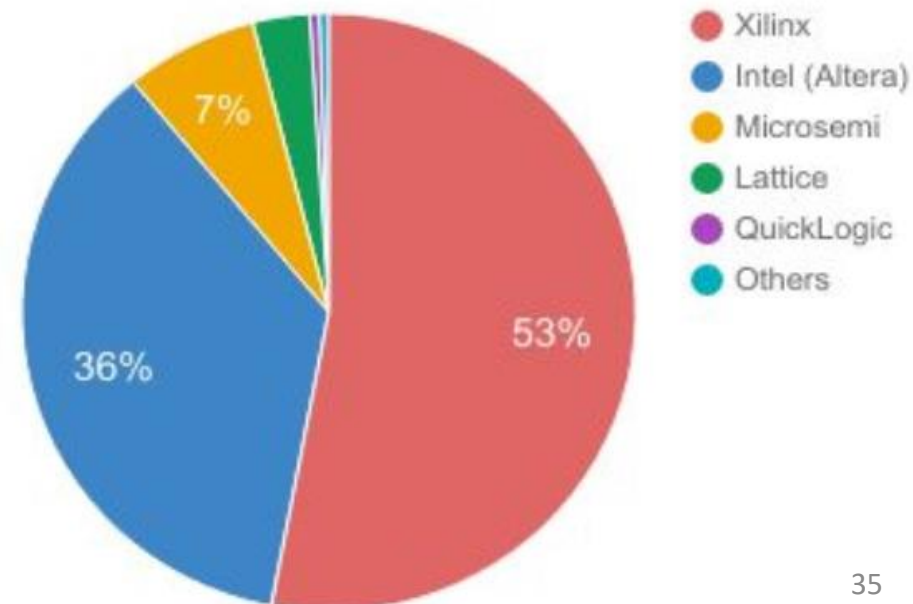
This is the most critical step...

- Gather requirements from the users
- Specify:
 - Target application (General purpose or Specific)
 - Main features (e.g. System bus, SoC, Multi-gigabit transceivers, etc.)
 - FPGA vendor (e.g. Xilinx, IntelFPGA (Altera), Microsemi (Actel), Lattice, etc.)

The rest of the design process is based on it!!!



FPGA market share in 2016



Small FPGA vendors may target specific markets (e.g. Microsemi offers high reliable FPGAs, etc..)

FPGA gateway design work flow

Project Specification

This is the most critical step...

- Gather requirements from the users
- Specify:
 - Target application (General purpose or Specific)
 - Main features (e.g. System bus, SoC, Multi-gigabit transceivers, etc.)
 - FPGA vendor (e.g. Xilinx, Intel (Altera), Microsemi (Actel), Lattice, etc.)
 - Electronic board (Custom or COTS (*))

The rest of the design process is based on it!!!

Example of COTS board (Xilinx Devkit)



Example of Custom Board



FPGA gateway design work flow

Project Specification

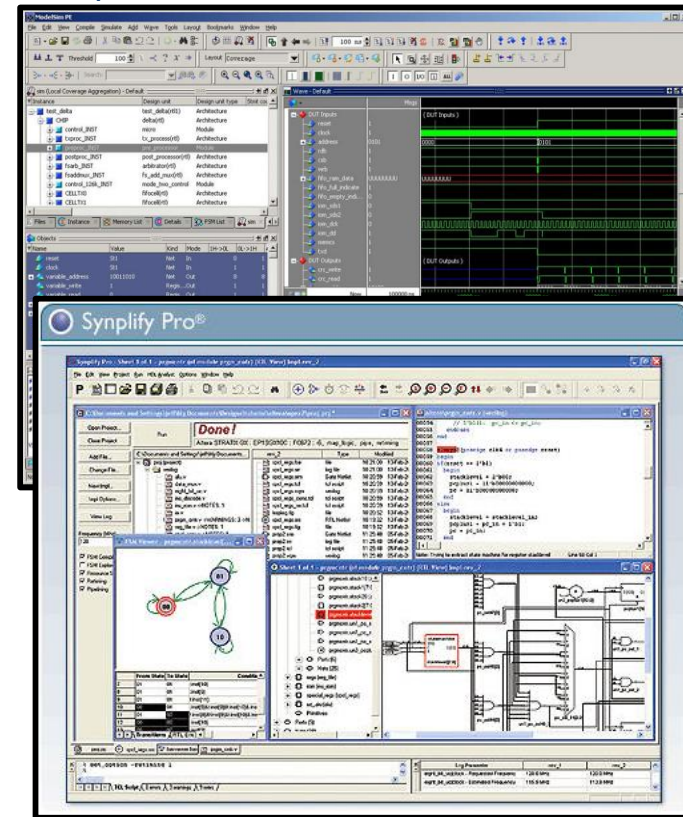
This is the most critical step...

- Gather requirements from the users
- Specify:

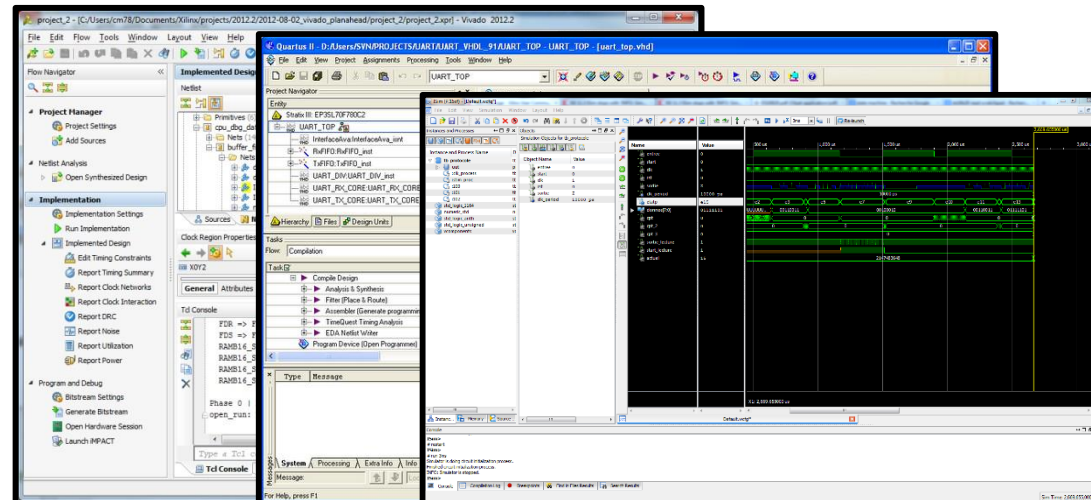
The rest of the design process is based on it!!!

- Target application (General purpose or Specific)
- Main features (e.g. System bus, SoC, Multi-gigabit transceivers, etc.)
- FPGA vendor (e.g. Xilinx, Intel (Altera), Microsemi (Actel), Lattice, etc.)
- Electronic board (Custom or COTS (*))
- Development tools (FPGA vendor or Commercial)

Example of Commercial Tools



Example of FPGA Vendor Tools



(* Commercial Off-The-Shelf (COTS) ³⁷

FPGA gateway design work flow

Project Specification

This is the most critical step...



The rest of the design process is based on it!!!

- **Gather requirements from the users**
- **Specify:**
 - Target application (General purpose or Specific)
 - Main features (e.g. System bus, SoC, Multi-gigabit transceivers, etc.)
 - FPGA vendor (e.g. Xilinx, Intel (Altera), Microsemi (Actel), Lattice, etc.)
 - Electronic board (Custom or COTS (*))
 - Development tools (FPGA vendor or Commercial)
 - Optimization (Speed, Area, Power or default)

FPGA gateway design work flow

Project Specification

This is the most critical step...



- Gather requirements from the users
- Specify:

The rest of the design process is based on it!!!

- Target application (General purpose or Specific)
- Main features (e.g. System bus, SoC, Multi-gigabit transceivers, etc.)
- FPGA vendor (e.g. Xilinx, Intel (Altera), Microsemi (Actel), Lattice, etc.)
- Electronic board (Custom or COTS (*))
- Development tools (FPGA vendor or Commercial)
- Optimization (Speed, Area, Power or default)



FPGA gateway design work flow

Project Specification

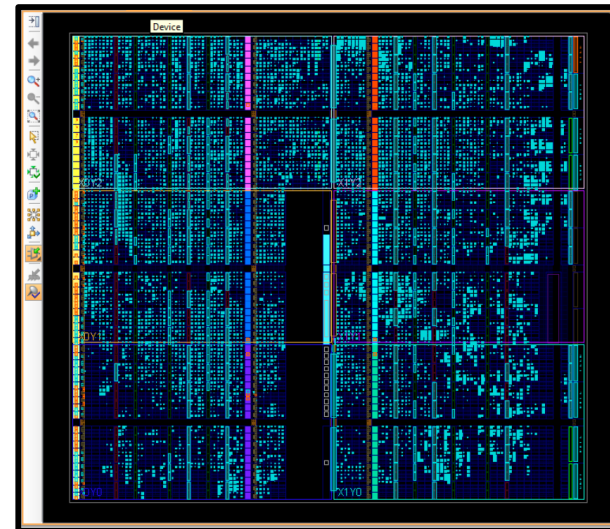
This is the most critical step...



- Gather requirements from the users
- Specify:

The rest of the design process is based on it!!!

- Target application (General purpose or Specific)
- Main features (e.g. System bus, SoC, Multi-gigabit transceivers, etc.)
- FPGA vendor (e.g. Xilinx, Intel (Altera), Microsemi (Actel), Lattice, etc.)
- Electronic board (Custom or COTS (*))
- Development tools (FPGA vendor or Commercial)
- Optimization (Speed, Area, Power or default)



FPGA gateway design work flow

Project Specification

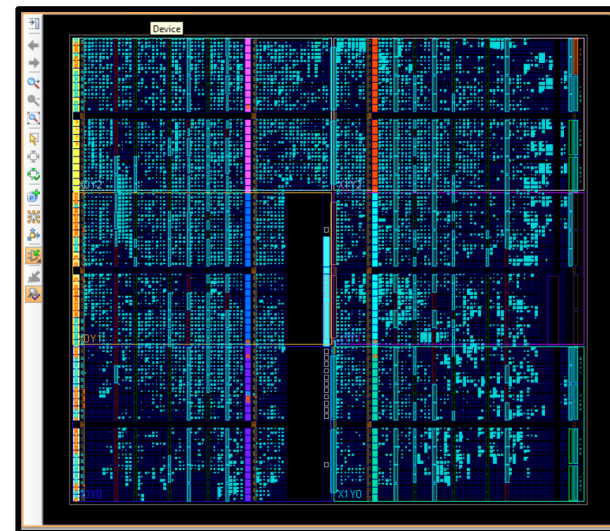
This is the most critical step...



- Gather requirements from the users
- Specify:

The rest of the design process is based on it!!!

- Target application (General purpose or Specific)
- Main features (e.g. System bus, SoC, Multi-gigabit transceivers, etc.)
- FPGA vendor (e.g. Xilinx, Intel (Altera), Microsemi (Actel), Lattice, etc.)
- Electronic board (Custom or COTS (*))
- Development tools (FPGA vendor or Commercial)
- Optimization (Speed, Area, Power or default)



FPGA gateway design work flow

Project Specification

This is the most critical step...



- Gather requirements from the users
- Specify:
 - Target application (General purpose or Specific)
 - Main features (e.g. System bus, SoC, Multi-gigabit transceivers, etc.)
 - FPGA vendor (e.g. Xilinx, Intel (Altera), Microsemi (Actel), Lattice, etc.)
 - Electronic board (Custom or COTS (*))
 - Development tools (FPGA vendor or Commercial)
 - Optimization (Speed, Area, Power or default)
 - Design language (HDL, Schematics or HLS)

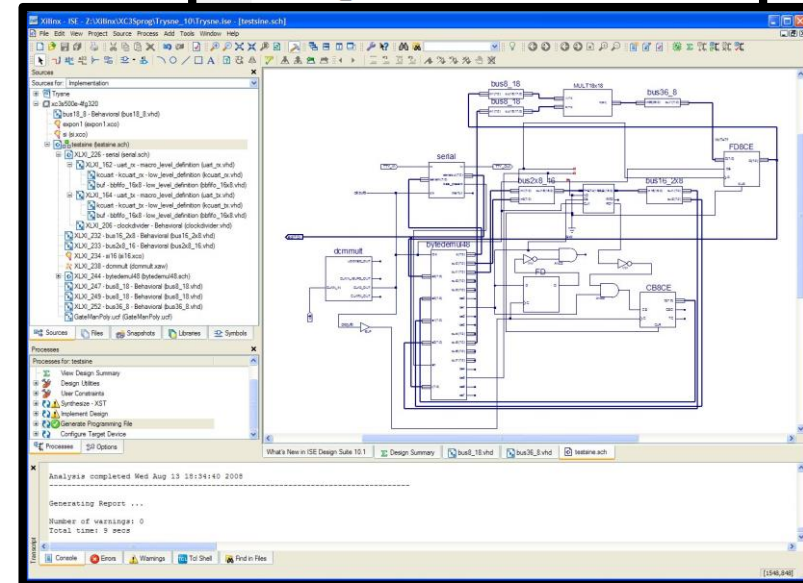
The rest of the design process is based on it!!!

Examples of Design Languages

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

**HDL are the most popular for RTL design
but...**

**Schematics or HLS
may be better in some cases
(e.g. SoC bus interconnect, etc..)**



FPGA gateway design work flow

Project Specification

This is the most critical step...



- Gather requirements from the users
- Specify:

The rest of the design process is based on it!!!

- Target application (General purpose or Specific)
- Main features (e.g. System bus, SoC, Multi-gigabit transceivers, etc.)
- FPGA vendor (e.g. Xilinx, Intel (Altera), Microsemi (Actel), Lattice, etc.)
- Electronic board (Custom or COTS (*))
- Development tools (FPGA vendor or Commercial)
- Optimization (Speed, Area, Power or default)
- Design language (HDL, Schematics or HLS)
- Coding convention

Example of Coding 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

**Your code
should be
readable**

FPGA gateway design work flow

Project Specification

This is the most critical step...

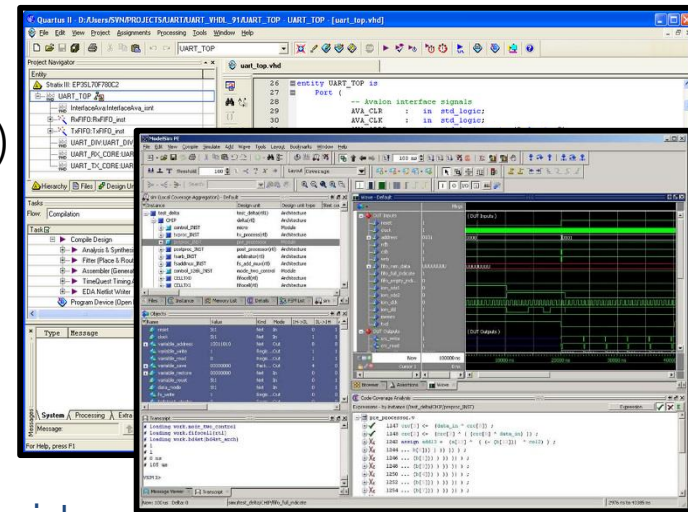


- Gather requirements from the users
- Specify:

The rest of the design process is based on it!!!

Example of GUIs

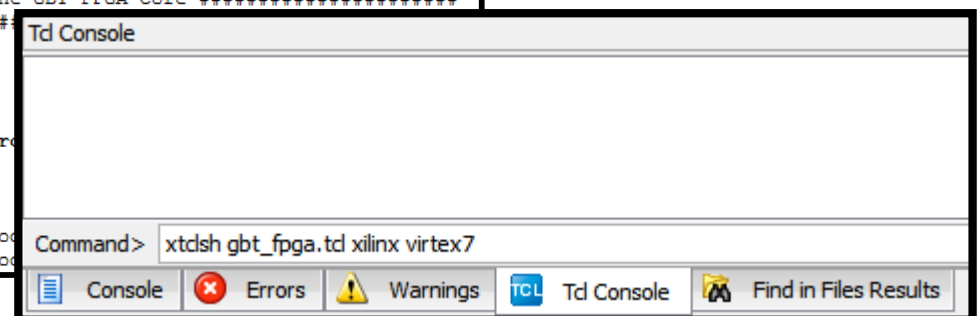
- Target application (General purpose or Specific)
- Main features (e.g. System bus, SoC, Multi-gigabit transceivers, etc.)
- FPGA vendor (e.g. Xilinx, Intel (Altera), Microsemi (Actel), Lattice, etc.)
- Electronic board (Custom or COTS (*))
- Development tools (FPGA vendor or Commercial)
- Optimization (Speed, Area, Power or default)
- Design language (HDL, Schematics or HLS)
- Coding convention
- Software interface (GUI, Scripts or both)



Example of TCL script

```
45 #####
46 ##### Commands for Adding the Source Files of the GBT-FPGA Core #####
47 #####
48
49 ## Comment: Adding Common files:
50
51 puts "->"
52 puts "-> Adding common files of the GBT-FPGA Core to the ISE project"
53 puts "->"
54
55 xfile add $SOURCE_PATH/gbt_bank/core_sources/gbt_rx/gbt_rx.vhd
56 xfile add $SOURCE_PATH/gbt_bank/core_sources/gbt_rx/gbt_rx_decoder.vhd
57 xfile add $SOURCE_PATH/gbt_bank/core_sources/gbt_rx/gbt_rx_decoder.vhd
```

Xilinx ISE TCL console



FPGA gateway design work flow

Project Specification

This is the most critical step...



- **Gather requirements from the users**
- **Specify:**

The rest of the design process is based on it!!!

- Target application (General purpose or Specific)
- Main features (e.g. System bus, SoC, Multi-gigabit transceivers, etc.)
- FPGA vendor (e.g. Xilinx, Intel (Altera), Microsemi (Actel), Lattice, etc.)
- Electronic board (Custom or COTS (*))
- Development tools (FPGA vendor or Commercial)
- Optimization (Speed, Area, Power or default)
- Design language (HDL, Schematics or HLS)
- Coding convention
- Software interface (GUI, Scripts or both)
- Use of files repository (SVN, GIT, etc.. or none)



git

FPGA gateway design work flow

Project Specification

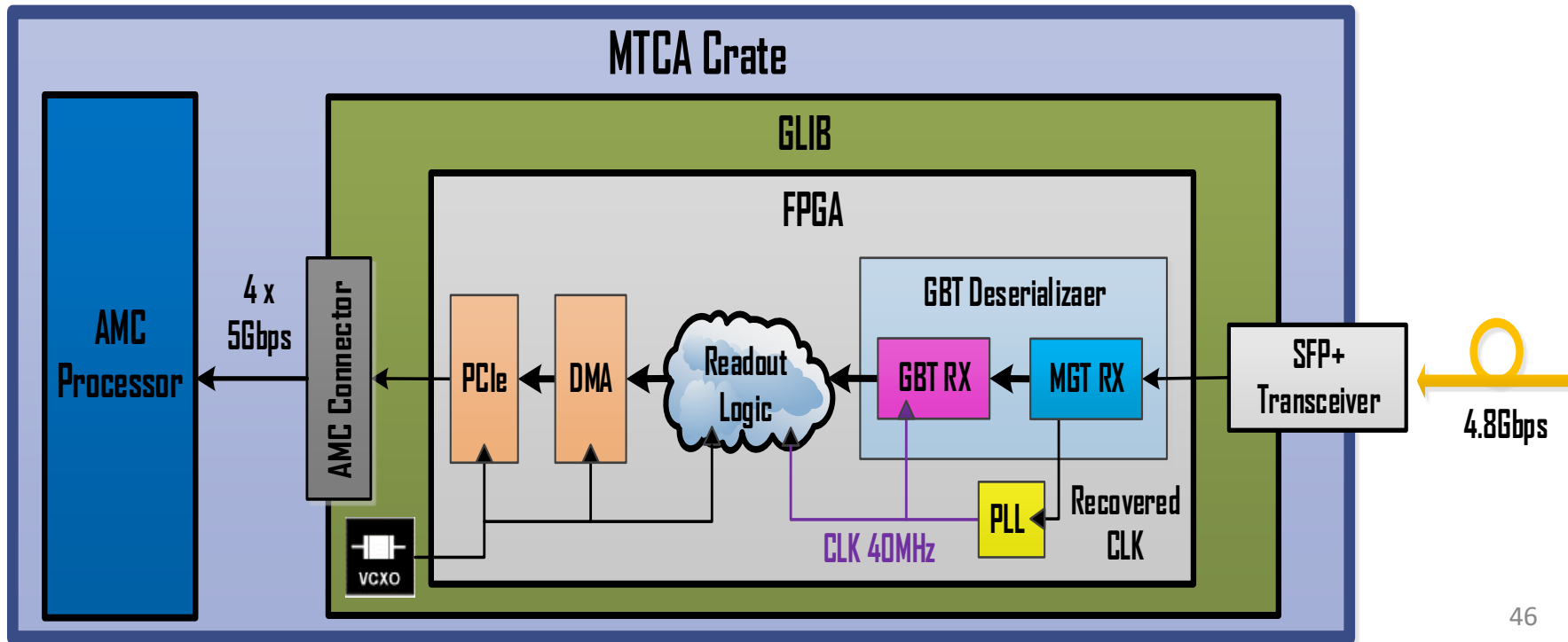
This is the most critical step...



- Block diagram of the system
 - Include the FPGA logic...
 - ... but also the on-board devices and related devices
 - May combine different abstraction levels

The rest of the design process is based on it!!!

Example of system block diagram



FPGA gateway design work flow

Project Specification

This is the most critical step...



- Pin planning

The rest of the design process is based on it!!!

Pin assignments are one type of Location Constraints

Critical for Custom Boards!!!



Pin Planner - /home/adp1/CPRE281/project/Project - Project

File Edit View Processing Tools Window Help

Search altera.com

Groups

Named: *

Node Name	Direction	Location
<<new group>>		

Top View - Wire Bond
Cyclone II - EP2C35F672C6

Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved
Bit0	Input	PIN_N25	5	B5_N1	3.3-V ...fault)	
Bit1	Input	PIN_N26	5	B5_N1	3.3-V ...fault)	
Bit2	Input	PIN_P25	6	B6_N0	3.3-V ...fault)	
Bit3	Input	PIN_AE14	7	B7_N1	3.3-V ...fault)	
Clear	Input	PIN_V2	1	B1_N0	3.3-V ...fault)	
Clock	Input	PIN_N2	2	B2_N1	3.3-V ...fault)	
Control	Input	PIN_C13	3	B3_N0	3.3-V ...fault)	
Gn	Input	PIN_B13	4	B4_N1	3.3-V ...fault)	
inputa	Output	PIN_I3	2	B2_N1	3.3-V ...fault)	

Named: * Edit: X ✓ Filter: Pins: all

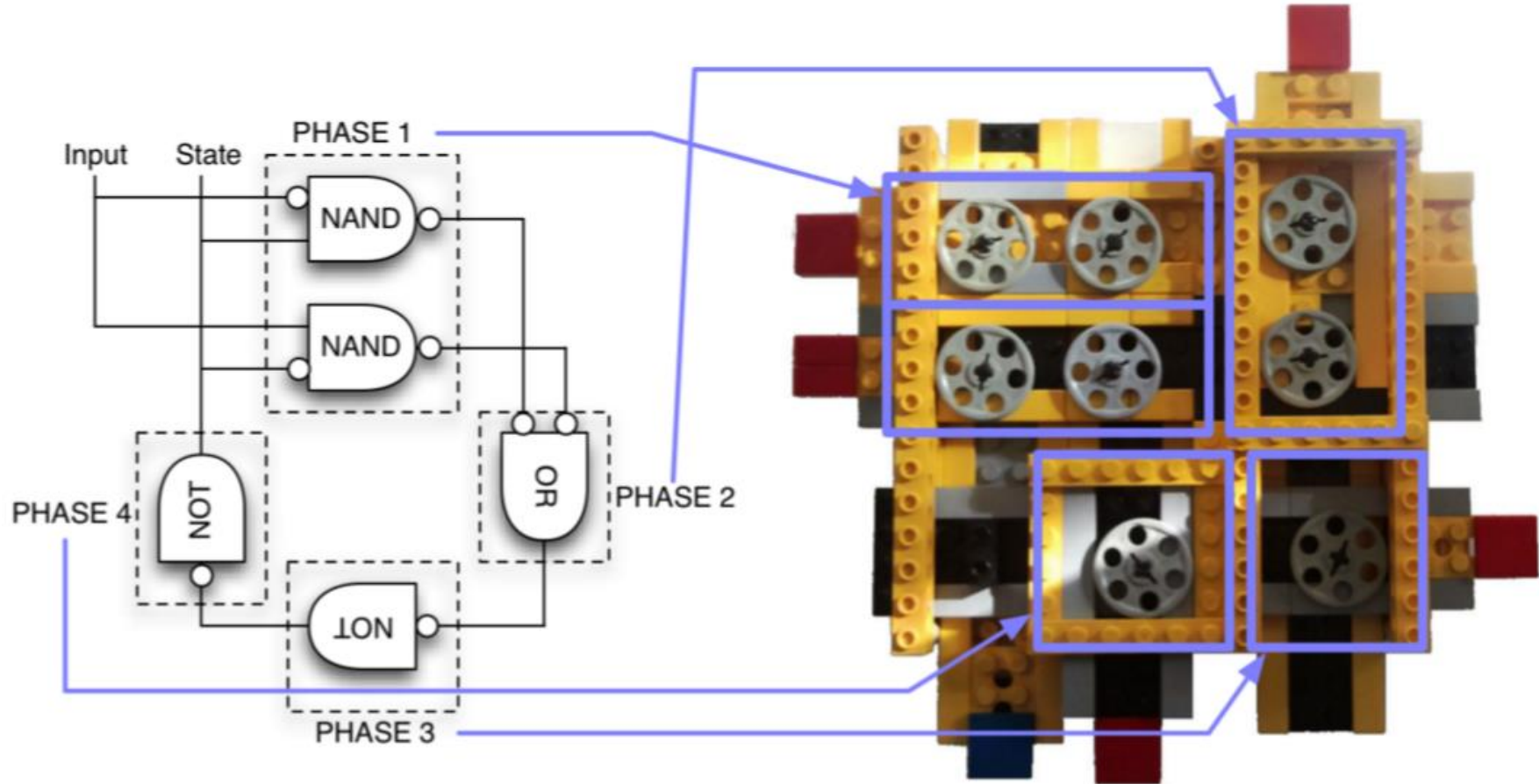
All Pins

0% 00:00:00

Example of Pin Planner GUI

FPGA gateway design work flow

Design Entry

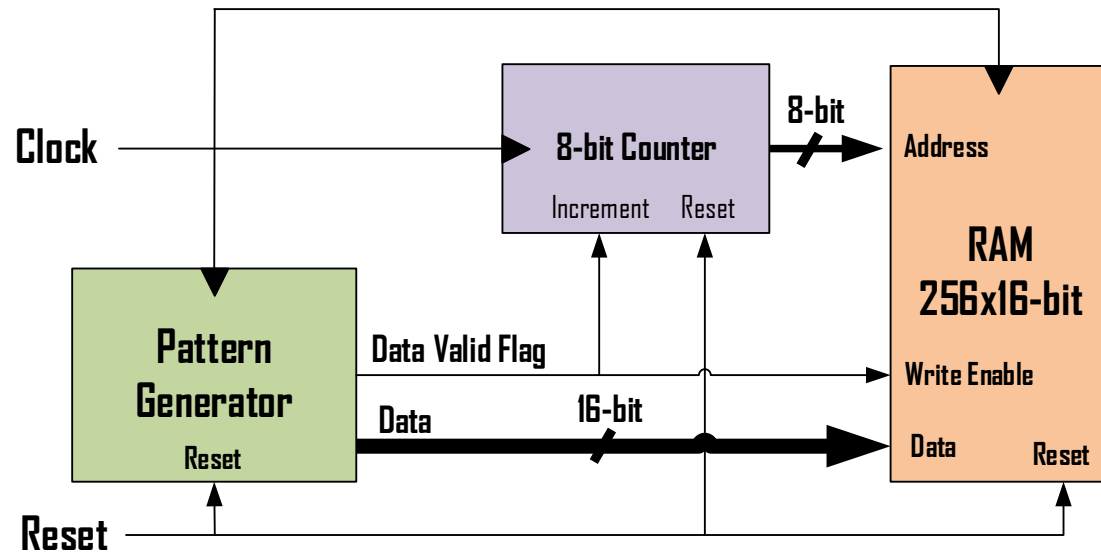


FPGA gateway design work flow

Design Entry: Modularity & Reusability

- Your system should be **Modular**
 - Design at RTL level (think hard...ware)
 - Well defined clocks and resets schemes
 - Separated Data & Control paths
 - Multiple instantiations

Good example of Modular System



- Your code should be **Reusable**
 - Add primitives (and modules) to the system by inference when possible
 - Use parameters in your code (e.g. generics in VHDL, parameters in Verilog, etc.)
 - Centralise parameters in external files (e.g. packages in VHDL, headers in Verilog, etc.)
 - Use configurable modules interfaces when possible (e.g. parametrised vectors, records in VHDL, etc.)
 - Use standard features (e.g. I2C, Wishbone, etc.)
 - Use standard IP Cores (e.g. from www.OpenCores.org, etc.)
 - Avoid vendor specific IP Cores when possible
 - Talk with your colleagues and see what other FPGA designers are doing

FPGA gateway design work flow

Design Entry: Coding for Synthesis

Synthesizable code is intended for FPGA implementation

- Use non-synthesizable HLD statements only in simulation test benches

A fundamental guiding principle when coding for synthesis is to minimize, if not eliminate, all structures and directives that could potentially create a mismatch between simulation and synthesis.

From book "Advanced FPGA Design" by Steve Kilts (Copyright © 2007 John Wiley & Sons, Inc.)

- The RTL synthesis tool is expecting a synchronous design...

FPGA gateway design work flow

Design Entry: Coding for Synthesis

Synthesizable code is intended for FPGA implementation

- Use non-synthesizable HLD statements only in simulation test benches

A fundamental guiding principle when coding for synthesis is to minimize, if not eliminate, all structures and directives that could potentially create a mismatch between simulation and synthesis.

From book "Advanced FPGA Design" by Steve Kilts (Copyright © 2007 John Wiley & Sons, Inc.)

- The RTL synthesis tool is expecting a synchronous design...

But what is a synchronous design???



FPGA gateway design work flow

Design Entry: Coding for Synthesis

Synthesizable code is intended for FPGA implementation

- Use non-synthesizable HLD statements only in simulation test benches

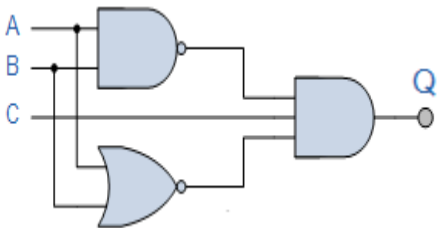
A fundamental guiding principle when coding for synthesis is to minimize, if not eliminate, all structures and directives that could potentially create a mismatch between simulation and synthesis.

From book "Advanced FPGA Design" by Steve Kilitz (Copyright © 2007 John Wiley & Sons, Inc.)

- The RTL synthesis tool is expecting a synchronous design...

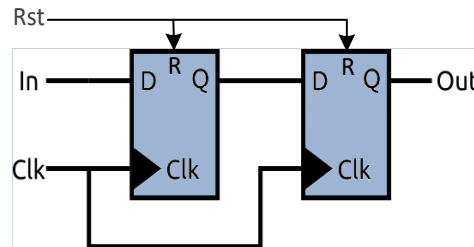
Synchronous design is the one composed by combinatorial logic (e.g. logic gates, multiplexors, etc..) and sequential logic (registers that are triggered on the edge of a single clock),

Combinatorial Logic



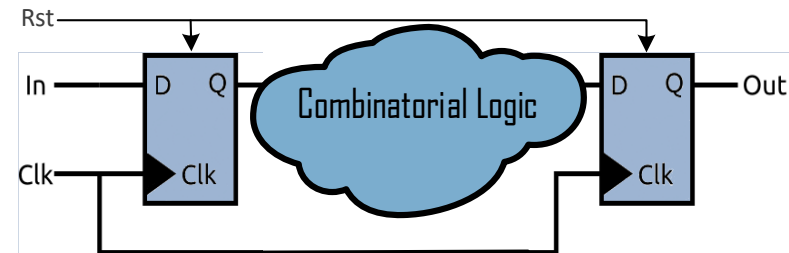
+

Sequential Logic



=

Synchronous design

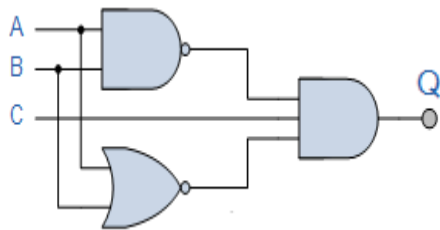


FPGA gateway design work flow

Design Entry: Coding for Synthesis

- **Combinatorial logic coding rules**
 - Sensitivity list must include ALL input signals
Not respecting this may lead to non responsive outputs under changes of input signals
 - ALL output signals must be assigned under ALL possible input conditions
Not respecting this may lead to undesired latches (asynchronous storage element)
 - No feedback from output to input signals
Not respecting this may lead to unknown output states (metastability) & undesired latches

Good combinatorial coding for synthesis

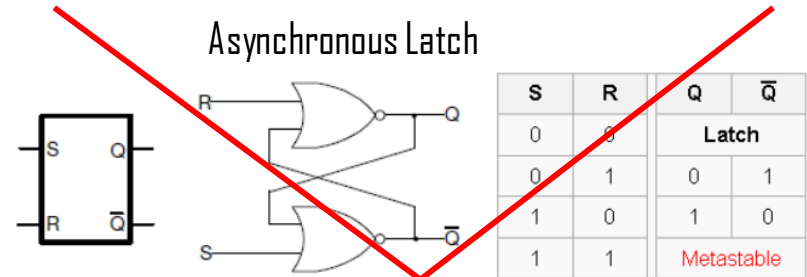


Typical Truth Table

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)
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;
```

Bad combinatorial coding for synthesis



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

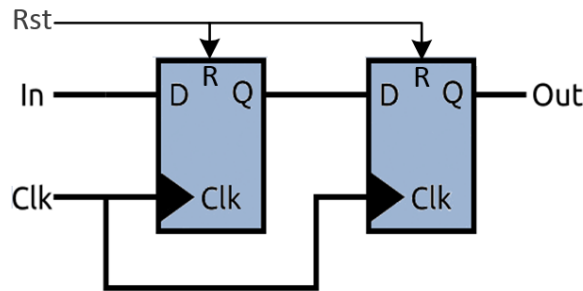
FPGA gateway design work flow

Design Entry: Coding for Synthesis

- **Sequential logic coding rules**
 - Only clock signal (and asynchronous set/reset signals when used) in sensitivity list
Not respecting this may produce undesired combinatorial logic
 - All registers of the sequence must be triggered by the same clock edge (either Rising or Falling)
Not respecting this may lead to metastability at the output of the registers
 - Include all registers of the sequence in the same reset branch
Not respecting this may lead to undesired register values after reset

Good sequential coding for synthesis

```
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;
```



Bad sequential coding for synthesis

```
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;
```

FPGA gateway design work flow

Design Entry: Coding for Synthesis

- **Synchronous design coding rules:**

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

Not respecting this may lead to incorrect analysis from the FPGA design tool

- Register ALL output signals (input signals also recommended)

Not respecting this may lead to uncontrolled length of combinatorial paths

- Properly design of reset scheme (mentioned later)

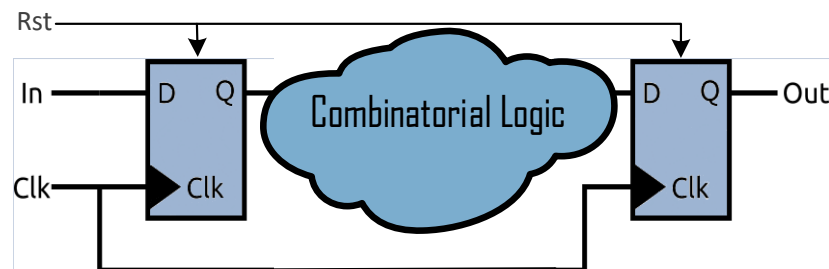
Not respecting this may lead to undesired register values after reset

- Properly design of clocking scheme (mentioned later)

Not respecting this may lead to metastability at the output of the registers & Misuse of resources

- Properly handle Clock Domain Crossings (CDC) (mentioned later)

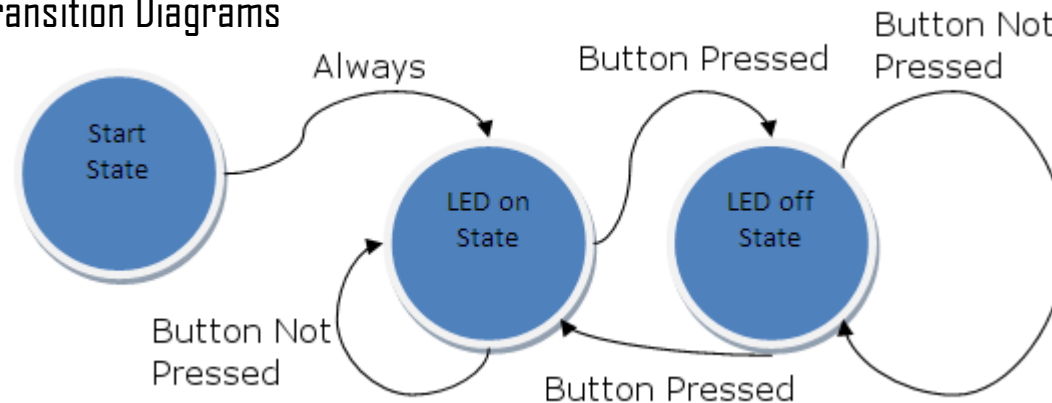
Not respecting this may lead to metastability at the output of the registers



FPGA gateway design work flow

Design Entry: Coding for Synthesis

- **Finite State Machines (FSMs):**
 - Digital logic circuit with a finite number of internal states
 - Widely used for system control
 - Two variants of FSM
 - Moore: Outputs depends only on the current state of the FSM
 - Mealy: Outputs depends only on the current state of the FSM as well as the current values of the inputs
 - Modelled by State Transition Diagrams



- Many different FSM coding styles (**But not all of the are good!!**)
- FSM coding considerations:
 - Synchronize inputs & outputs
 - Outputs may be assigned during states or state transitions
 - Be careful with unreachable/illegal states
 - You can add counters to FSMs

FPGA gateway design work flow

Design Entry: Reset Scheme

A bad reset scheme may get you crazy!!!

- Used to initialize the output of the registers 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!!!**)

FPGA gateway design work flow

Design Entry: Reset Scheme

A bad reset scheme may get you crazy!!!

- Used to initialize the output of the registers 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!!!**)

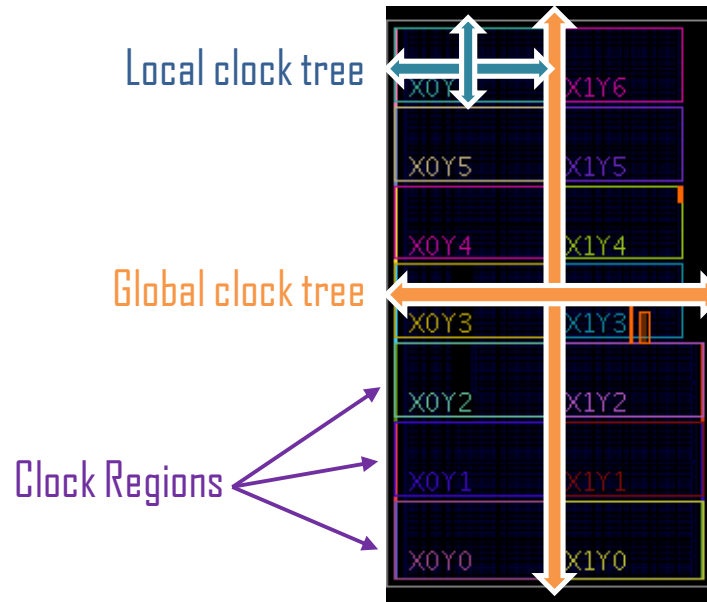
My advise is...
You should use
SYNCHRONOUS RESET
by default

FPGA gateway design work flow

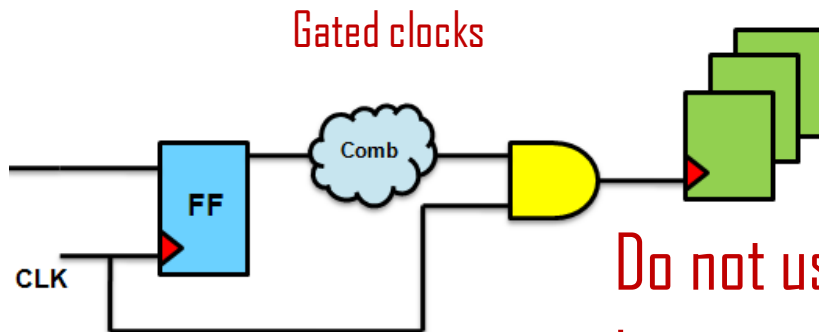
Design Entry: Clocks Scheme

Clocking resources are very precious!!!

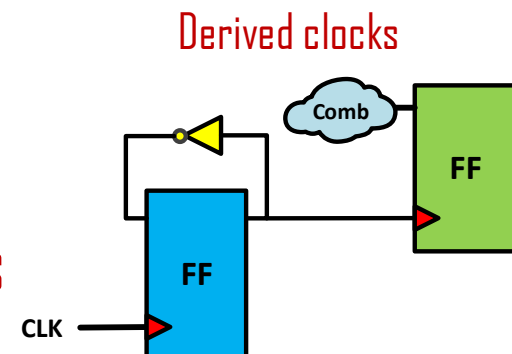
- Clock regions
- Clock trees (Global & Local)
- Other FPGA clocking resources
 - Clock capable pins
 - Clock buffers
 - Clock Multiplexors
 - PLLs & DCM



- Bad practices when designing your clocking scheme



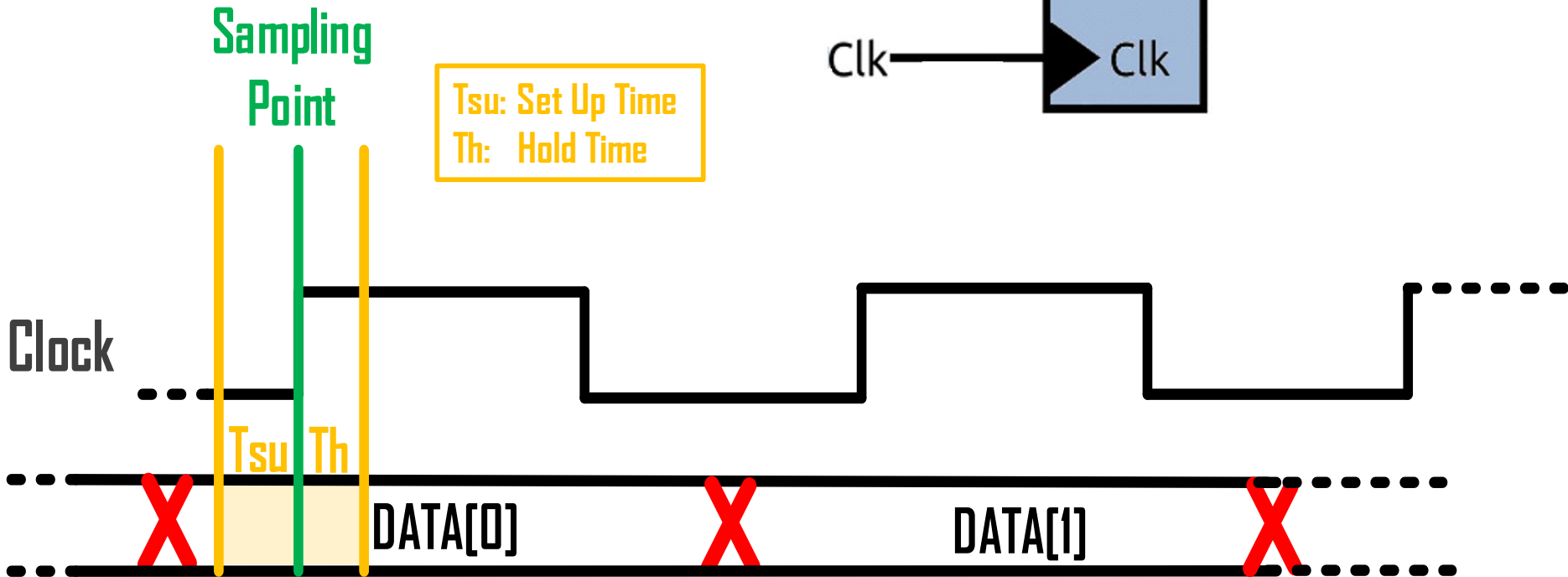
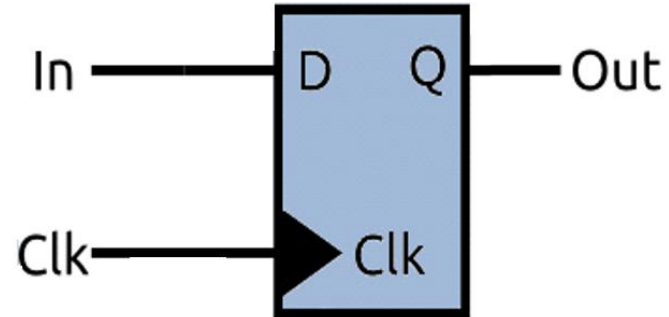
**Do not use these clocks
in your system!!!**



FPGA gateway design work flow

Design Entry: Timing

- Sampling



**No Stable Data
(Metastable Area)**

FPGA gateway design work flow

Design Entry: Timing

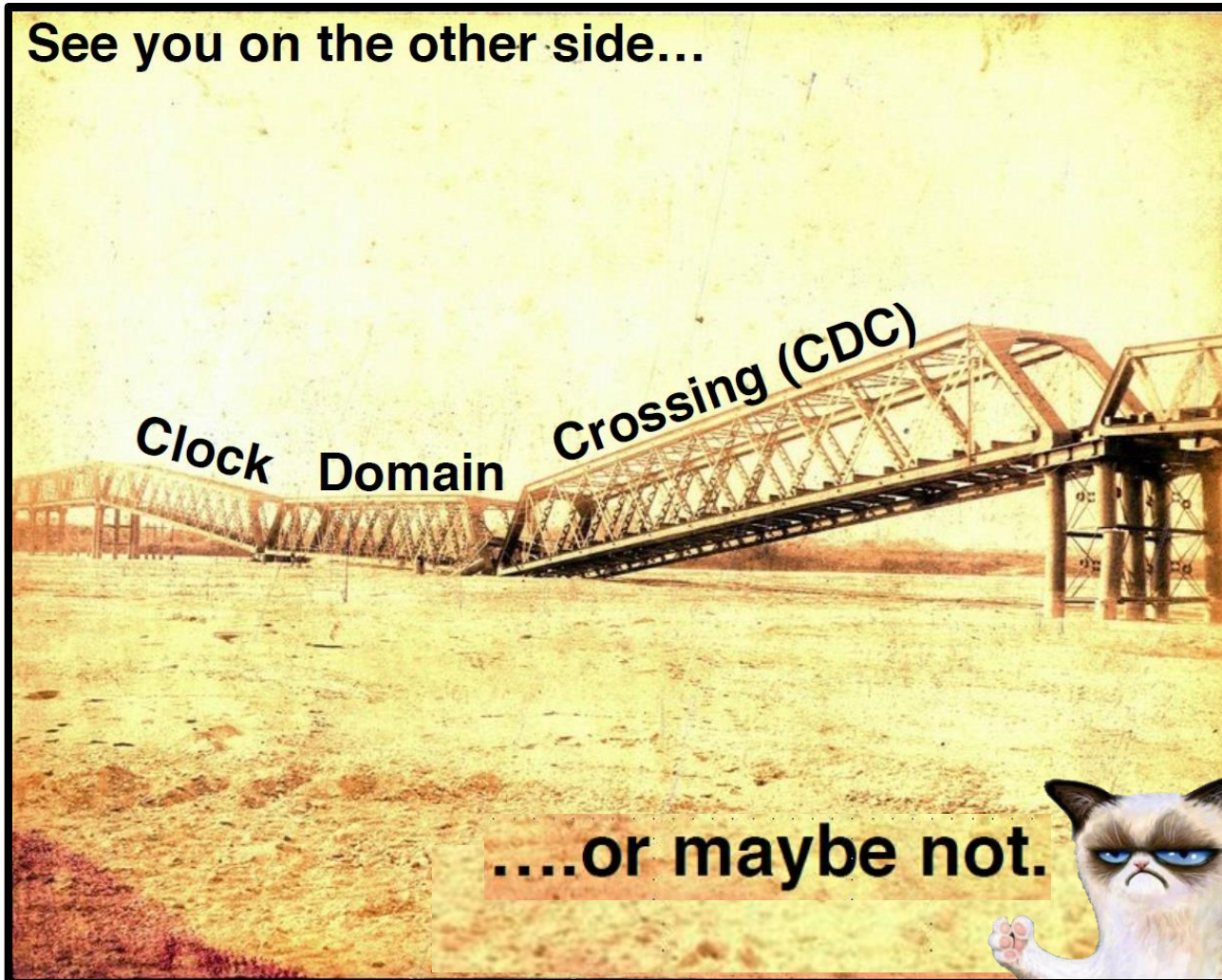
- Clock Domain Crossing (CDC)



FPGA gateway design work flow

Design Entry: Timing

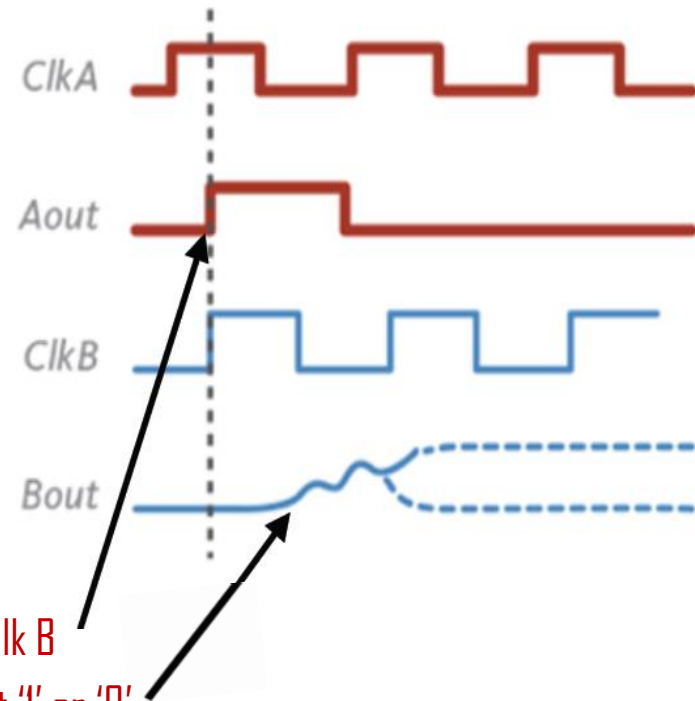
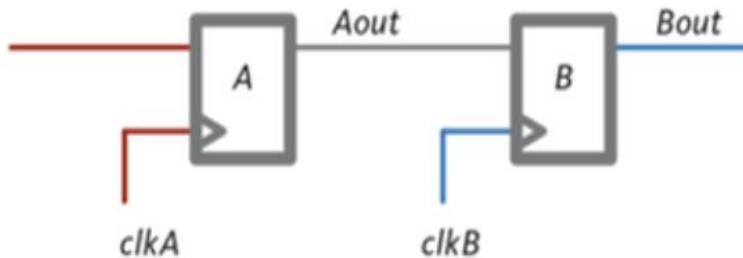
- Clock Domain Crossing (CDC)



FPGA gateway design work flow

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!!!**



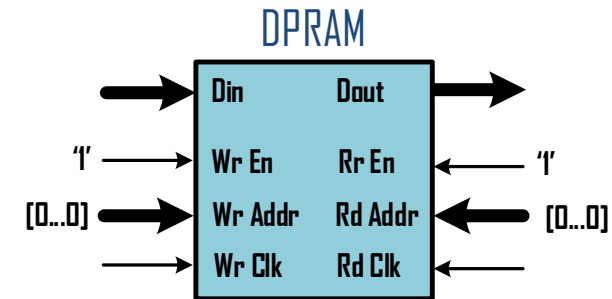
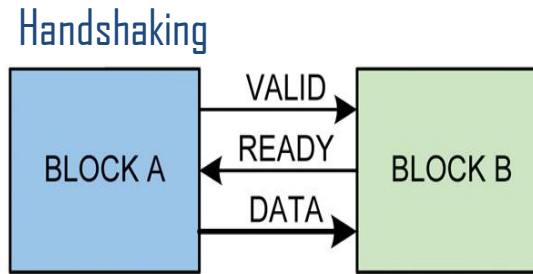
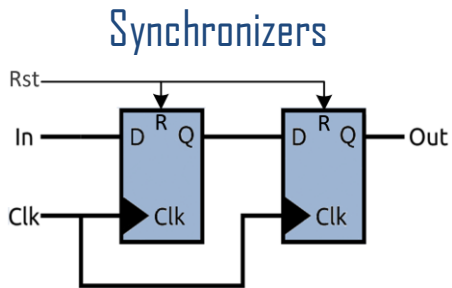
Signal violates the setup-time of FlipFlop B clocked by Clk B
Bout becomes metastable and then settles at either at '1' or '0'

Avoid creating unnecessary clock domains

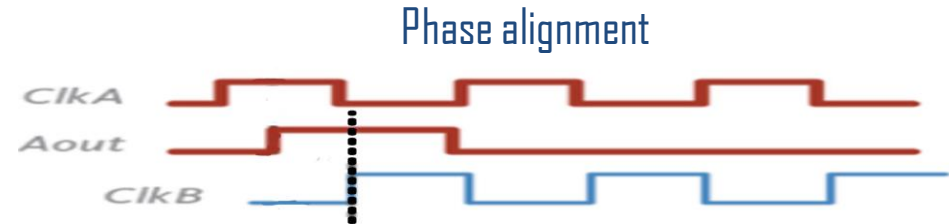
FPGA gateway design work flow

Design Entry: Timing

- Clock Domain Crossing: The workaround...



Be aware of FIFO overflow/underflow!!!



FPGA gateway design work flow

Design Entry: Primitives & IP Cores

- **Primitives:** Basic components of the FPGA
 - Vendor (and device) specific
 - Examples: Buffers (I/O & Clock), Registers, BRAMs, DSP blocks, Logic Gates (programed LUTs)
- **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)
 - Examples: : PLLs, Multi-gigabit Transceivers, Ethernet MAC, Microprocessors, etc..
- **Soft IP Cores:** Complex (or simple) modules ready to be implemented
 - They may be vendor specific or agnostic:
 - Vendor Specific: Encrypted Code or Requires Hard IP Core
 - Vendor Agnostic: Commercial or Open Source (www.OpenCores.org)
 - In many cases they may be set through GUI (Wizards)
 - Examples: : All kind of modules
- **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

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
```

FPGA gateway design work flow

Design Entry: Primitives & IP Cores

Add Primitives by Inference

- **Primitives:** Basic components of the FPGA
 - Vendor (and device) specific
 - Examples: Buffers (I/O & Clock), Registers, BRAMs, DSP blocks, Logic Gates (programed LUTs)
- **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)
 - Examples: : PLLs, Multi-gigabit Transceivers, Ethernet MAC, Microprocessors, etc..
- **Soft IP Cores:** Complex (or simple) modules ready to be implemented
 - They may be vendor specific or agnostic:
 - Vendor Specific: Encrypted Code or Requires Hard IP Core
 - Vendor Agnostic: Commercial or Open Source (www.OpenCores.org)
 - In many cases they may be set through GUI (Wizards)
 - Examples: : All kind of modules
- **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

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
```

FPGA gateway design work flow

Design Entry: Primitives & IP Cores

- **Primitives:** Basic components of the FPGA
 - Vendor (and device) specific
 - Examples: Buffers (I/O & Clock), Registers, BRAMs, DSP blocks, Logic Gates (programed LUTs)
- **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)
 - Examples: : PLLs, Multi-gigabit Transceivers, Ethernet MAC, Microprocessors, etc..
- **Soft IP Cores:** Complex (or simple) modules ready to be implemented
 - They may be vendor specific or agnostic:
 - Vendor Specific: Encrypted Code or Requires Hard IP Core
 - Vendor Agnostic: Commercial or Open Source (www.OpenCores.org)
 - In many cases they may be set through GUI (Wizards)
 - Examples: : All kind of modules
- **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
```

FPGA gateway design work flow

Synthesis

- **What does it do?**
 - Translates the schematic or HDL code into elementary logic functions
 - Defines the connection of these elementary functions
 - Uses Boolean Algebra and Karnaugh maps to optimize logic functions

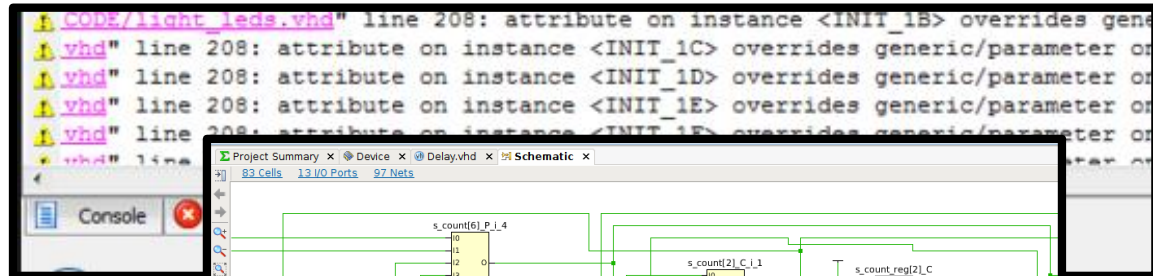
- **The FPGA design tool optimizes the design during synthesis**

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

- **Always check the synthesis report**

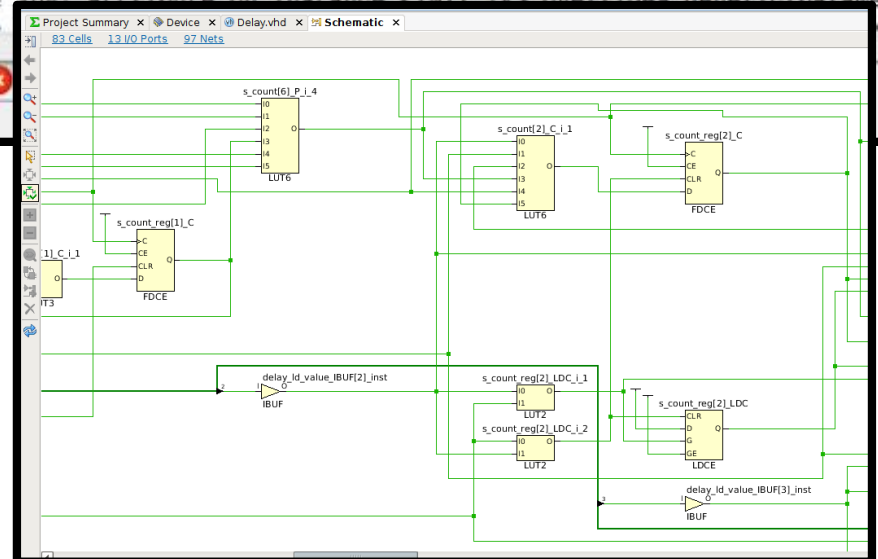
- Warnings & Errors
- Estimated resource utilization
- Optimizations
- And more...

Example of Synthesis Report



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

Example of RTL Schematic

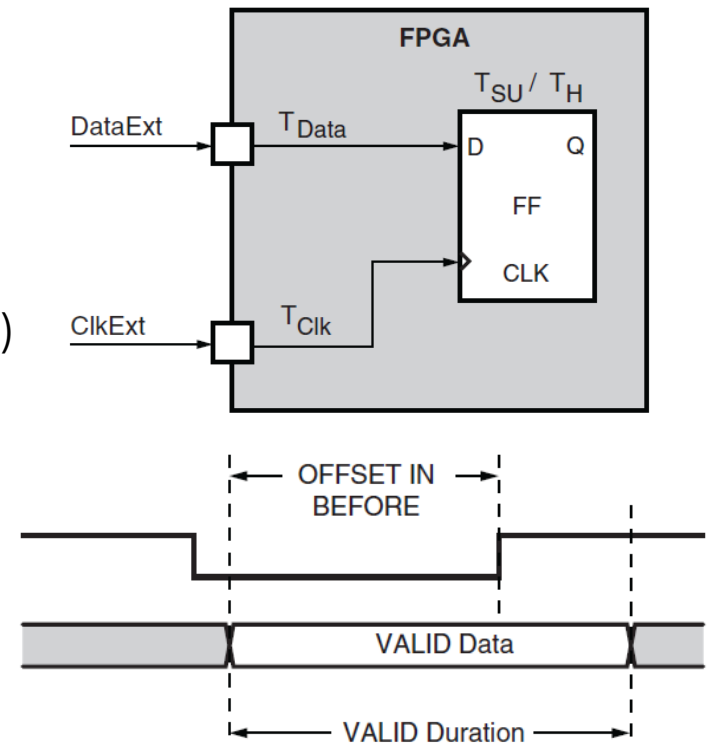


FPGA gateway design work flow

Constraints: Timing

- For a reliable system, the timing requirements for all paths must be provided to the FPGA design tool.
- Provided through constraint files (e.g. Xilinx .XDC, etc..) or GUI (that creates/writes constraint files).
- The most common types of path categories include:
 - 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 (in many cases with this is enough)
 - 2) Add path specific exceptions as needed
- Over constraining will difficult the routing

Example of timing constraint (Xilinx .ucf)

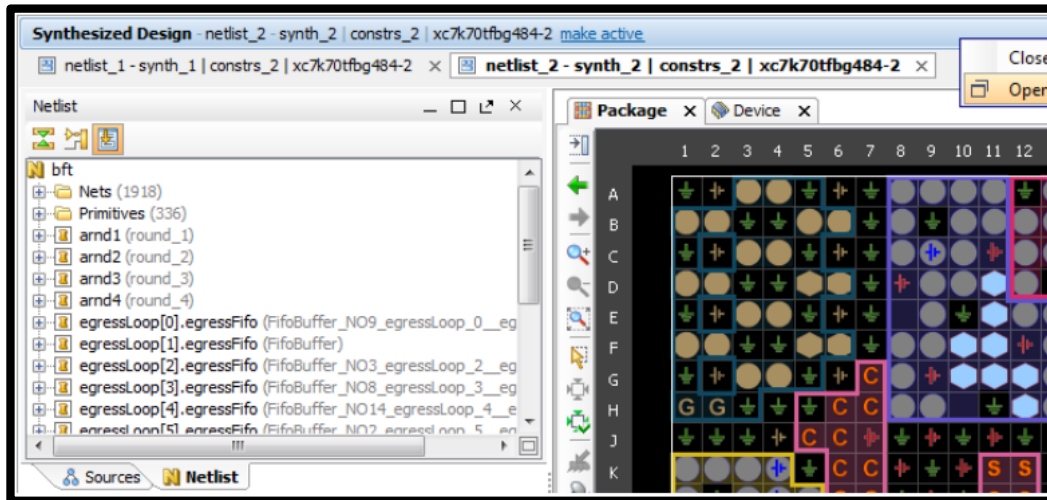


```
TIMEGRP DATA_IN OFFSET = IN 1 VALID 3 BEFORE CLK RISING;
```

FPGA gateway design work flow

Constraints: Physical

- Pin planning

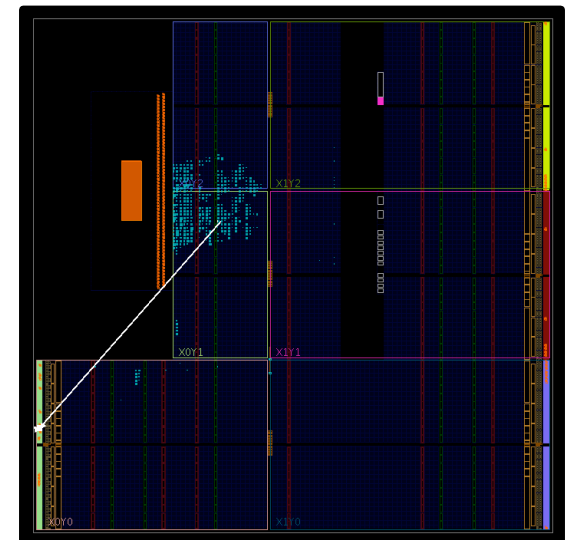


As previously mentioned...
You should do Pin Planning
during Specification Stage

- Floorplanning

- Try to place logic close to their related I/O pins
- Try to avoid routing across the chip
- Place the Hard IP cores, the related logic will follow
- You can separate the logic by areas (e.g. Xilinx Pblocks)

Floorplanning may improve routing times and allow
faster system speeds... but too much will complicate routing!!!



FPGA gateway design work flow

Implementation

- **The FPGA design tool:**

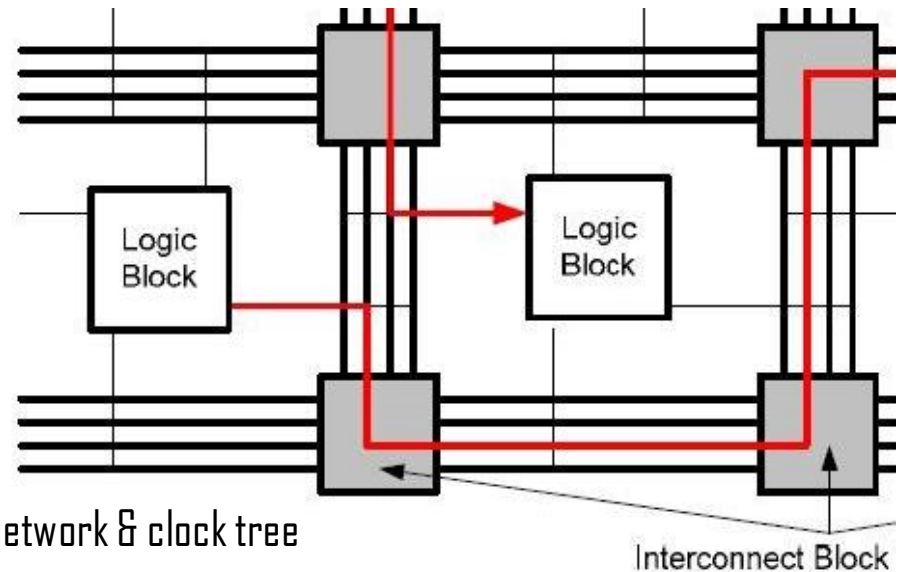
- 1) Translates the Timing and Physical constraints in order to guide the implementation

- 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 default)**

- **Physical Placement & Timing change after re-implementing (use constraints to minimize these changes)**

- **You should always check the different reports generated during implementation**

FPGA gateway design work flow

Static Timing Analysis

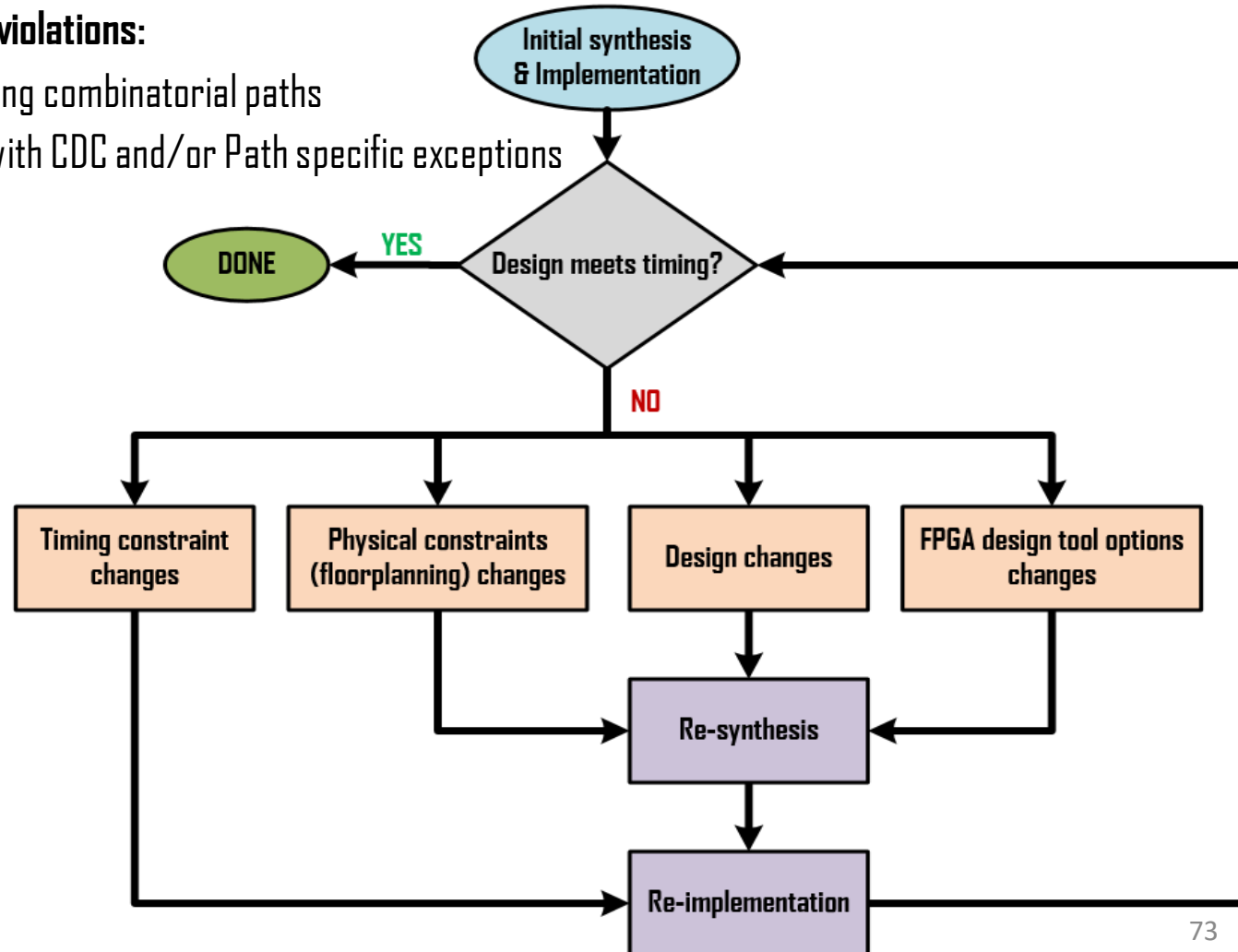
- The FPGA design tool analyses the signals propagation delays and clock relationships after P&R
- A timing report is generated, including the paths that did not meet the timing requirements
- Rule of thumb for timing violations:
 - Setup violations: Too long combinatorial paths
 - Hold violations: Issue with CDC and/or Path specific exceptions
- The timing closure flow:



FPGA gateway design work flow

Static Timing Analysis

- The FPGA design tool analyses the signals propagation delays and clock relationships after P&R
- 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:



FPGA gateway design work flow

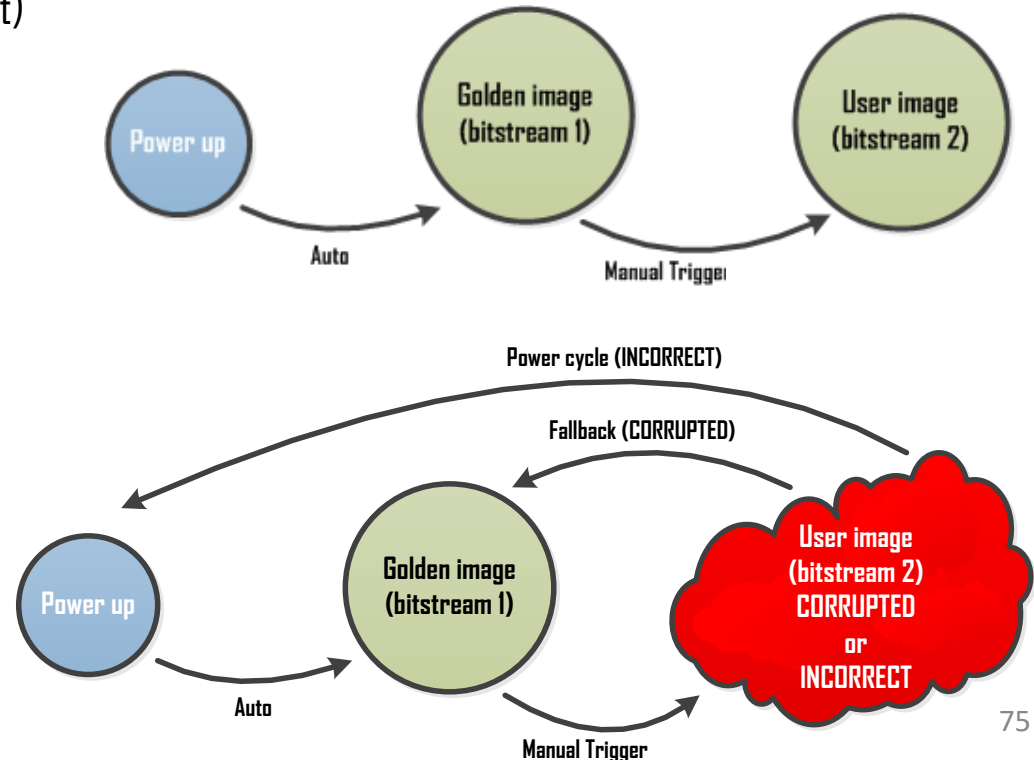
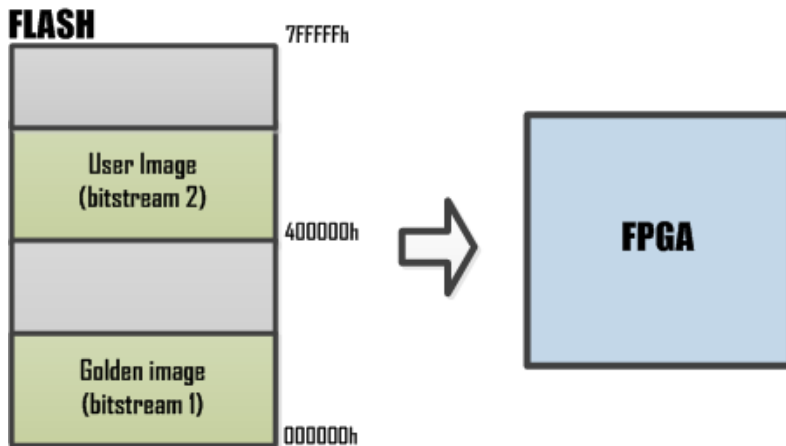
Bitstream Generation & FPGA Programming

- **Bitstream:**
 - Binary file containing the FPGA configuration data
 - Each FPGA vendor has its own bitstream file extension (e.g. .bit (Xilinx), .sof (Altera))
- **FPGA programming:**
 - Bitstream is loaded into the FPGA through JTAG
 - 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

FPGA gateway design work flow

Bitstream Generation & FPGA Programming

- **Bitstream:**
 - Binary file containing the FPGA configuration data
 - Each FPGA vendor has its own bitstream file extension (e.g. .bit (Xilinx), .sof (Altera))
- **FPGA programming:**
 - Bitstream is loaded into the FPGA through JTAG
 - 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

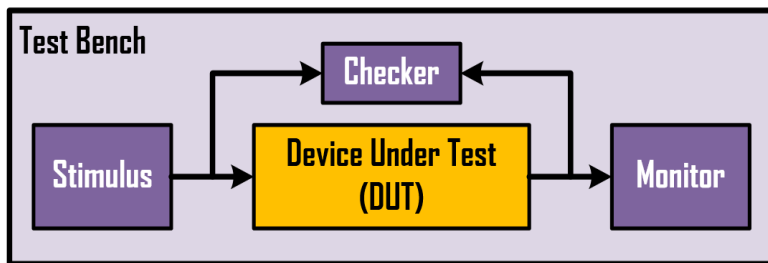


Multiboot/Safe FPGA configuration diagrams

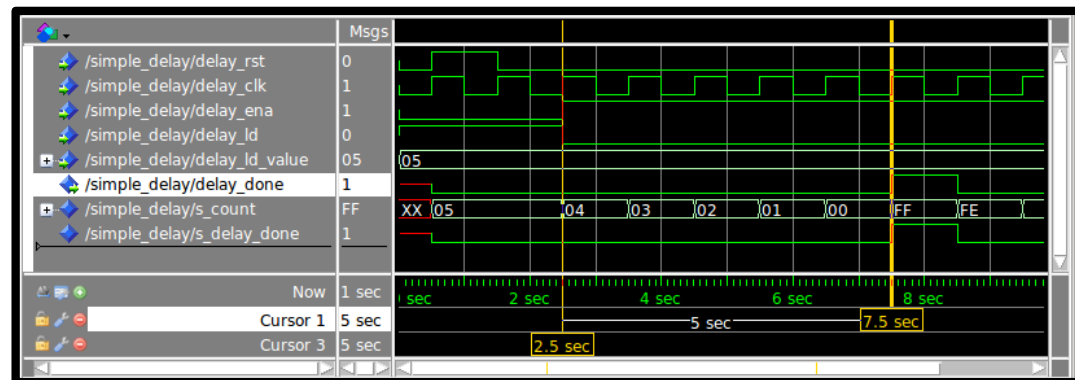
FPGA gateway design work flow

Simulation

- Event-based simulation to recreate the parallel nature of digital designs
- Verification of HDL modules and/or full systems
- HDL simulators:
 - Most popular: Modelsim/Questa
 - Other simulators: Vivado Simulator (Xilinx), Icarus Verilog (Open-source), etc.
- Different levels of simulation
 - Behavioural: simulates only the behaviour of the design **Fast**
 - Functional: uses realistic functional models for the target technology **Slow**
 - Timing : **most accurate**. Uses Implemented design after timing analysis **Very Slow**



Example of simulator wave window



FPGA gateway design work flow

In-System Analysers & 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 control of the FPGA signals
- Minimize interfering with the 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

Example of In-System Analyser (Altera SignalTap II)



Example of Virtual I/Os (Xilinx VIO)

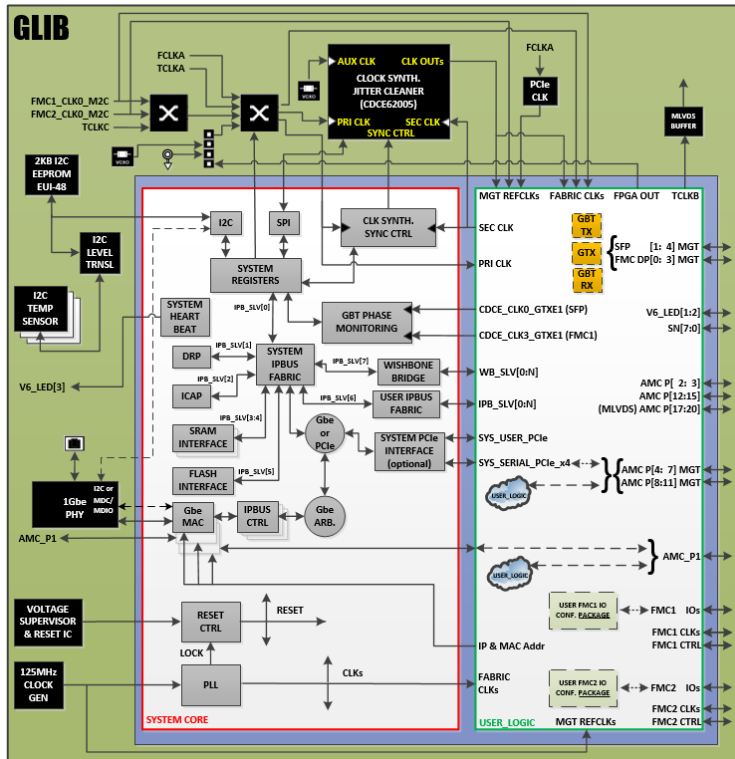
Bus/Signal	Value
LATENCY-OPTIMIZED GBT LINK (LOW WHEN STANDARD GBT)	<input type="radio"/>
TX FLL LOCKED	<input checked="" type="radio"/>
MGT READY	<input checked="" type="radio"/>
RX_WORDCLK ALIGNED (ALIGNED TO TX_WORDCLK) (LOW WHEN STANDARD GBT)	<input checked="" type="radio"/>
RX_FRAMECLK ALIGNED (ALIGNED TO TX_FRAMECLK) (LOW WHEN STANDARD GBT)	<input checked="" type="radio"/>
RX GBT READY	<input checked="" type="radio"/>
RX BITS/SLIP NUMBER	00
FPGA_CLKOUT ('0' -> TX_FRAMECLK '1' -> TX_WORDCLK)	0
LOOPBACK ('0' -> NORMAL '2' -> PMA LOOPBACK) (XILINX UG366 PAGE 124)	0
GENERAL RESET	<input type="checkbox"/>
ENCODING SELECTOR ('0' -> GBT FRAME '1' -> WIDE-BUS)	0
PATTERN SELECT ('1' -> COUNTER '2' -> STATIC others -> NO DATA ERROR DETECTION)	2
RESET RX GBT READY LOST FLAG	<input type="checkbox"/>
RX GBT READY LOST FLAG	<input type="checkbox"/>
RESET DATA ERROR SEEN FLAG	<input type="checkbox"/>
COMMON DATA ERROR SEEN FLAG	<input type="checkbox"/>
WIDE-BUS EXTRA DATA ERROR SEEN FLAG	<input type="checkbox"/>
ENCSB10B EXTRA DATA ERROR SEEN FLAG	<input type="checkbox"/>
TX HEADER SELECTOR ('0' -> IDLE '1' -> DATA)	<input type="checkbox"/>
RX HEADER IS DATA FLAG ('0' -> IDLE '1' -> DATA)	<input type="checkbox"/>

FPGA gateway design work flow

Debugging Techniques

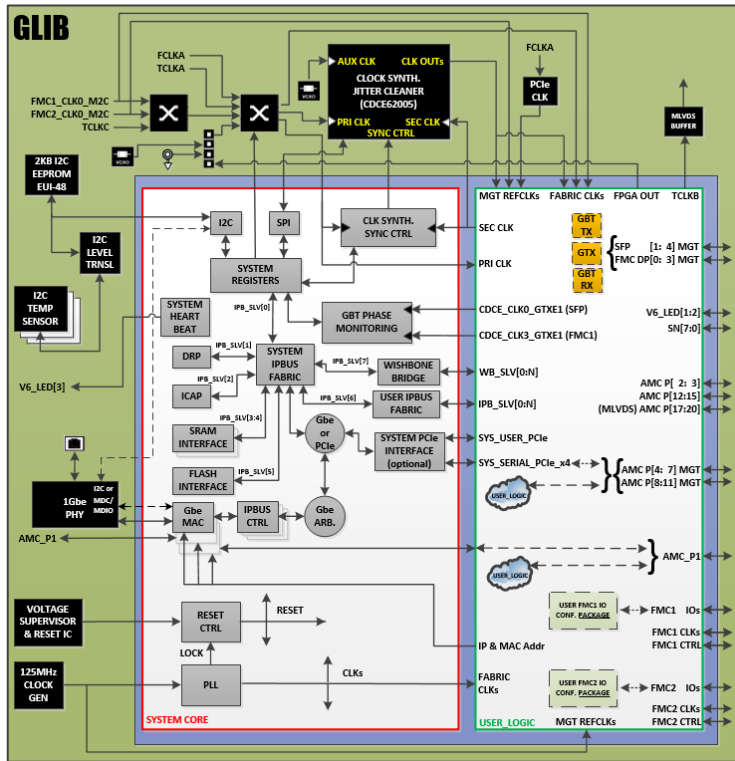
FPGA gateway design work flow

Debugging Techniques



FPGA gateway design work flow

Debugging Techniques

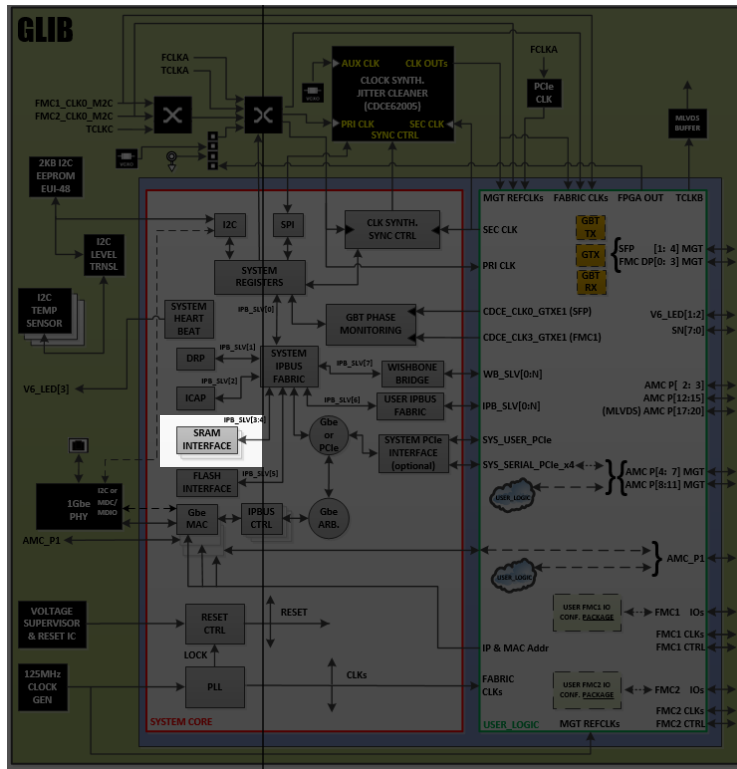


OMG!!!

FPGA gateway design work flow

Debugging Techniques

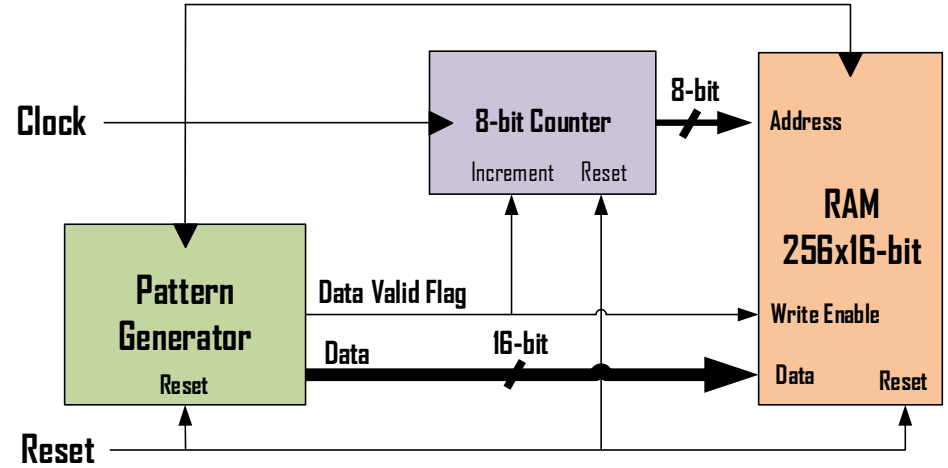
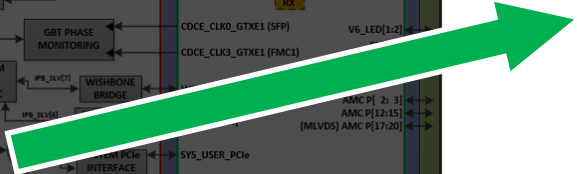
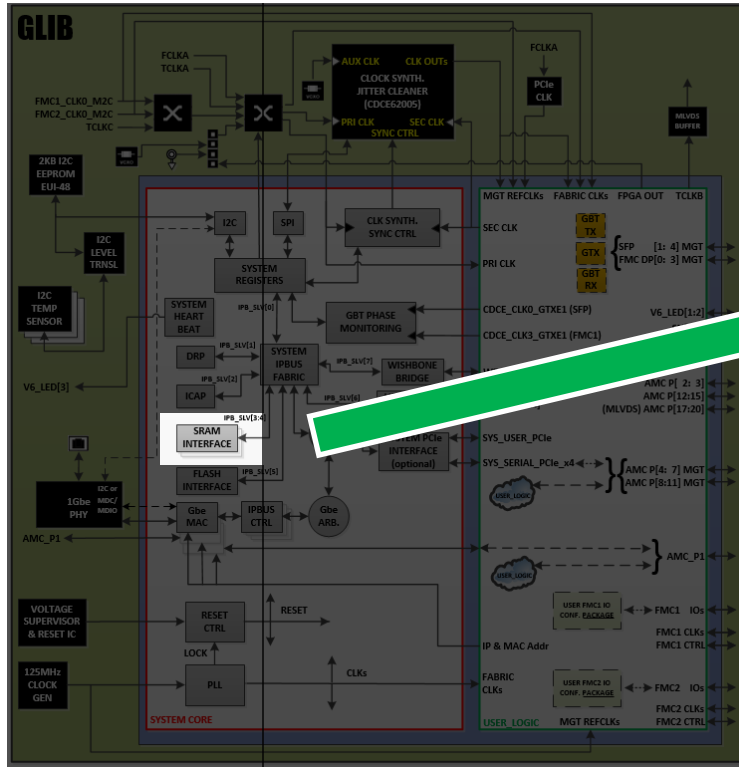
Divide & Conquer



FPGA gateway design work flow

Debugging Techniques

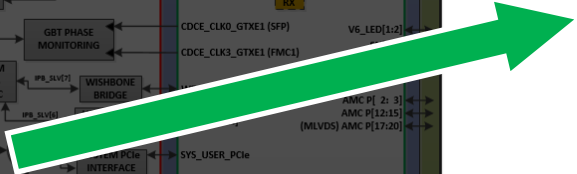
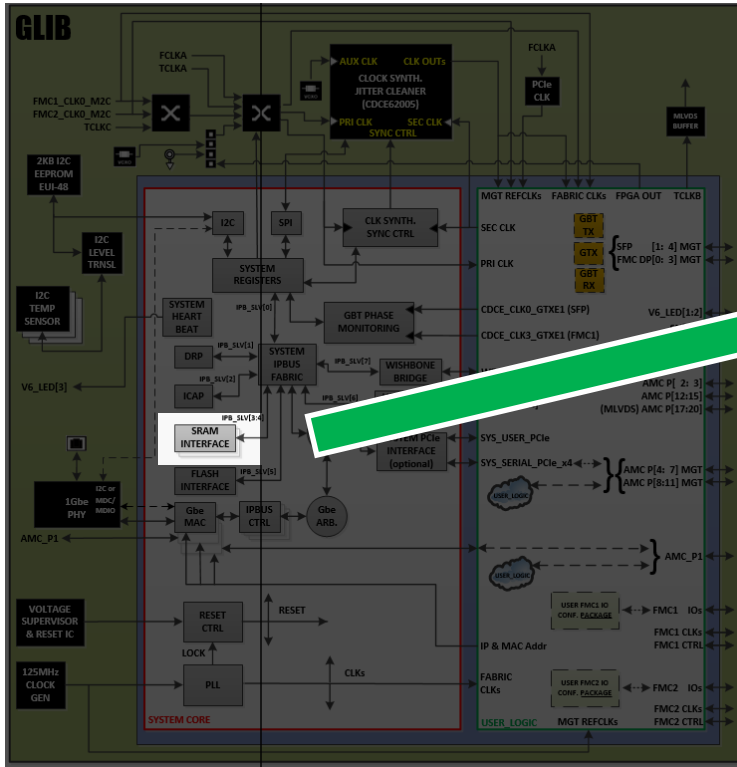
Divide & Conquer



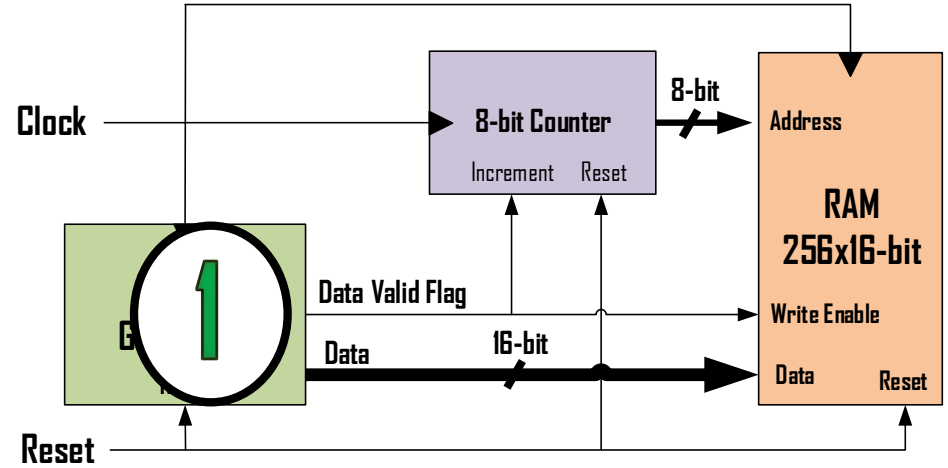
FPGA gateway design work flow

Debugging Techniques

Divide & Conquer



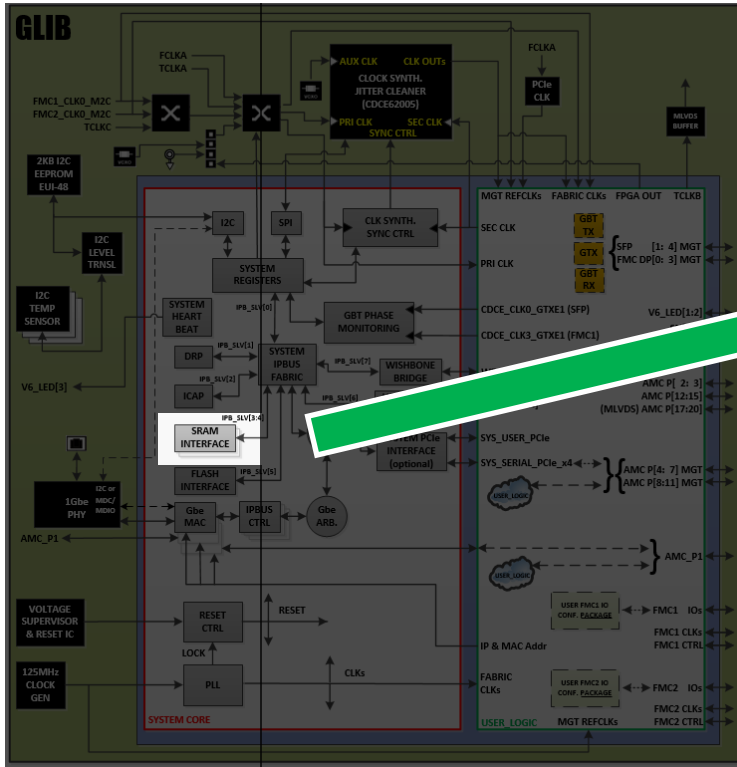
Follow the chain



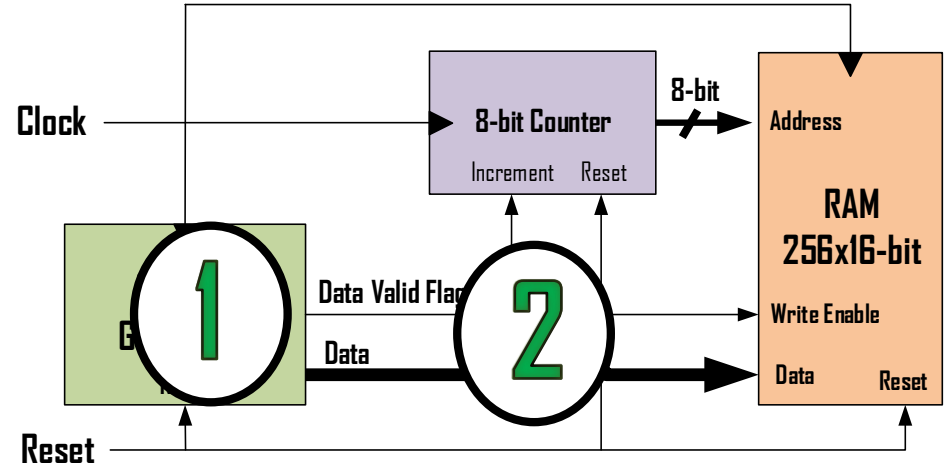
FPGA gateway design work flow

Debugging Techniques

Divide & Conquer



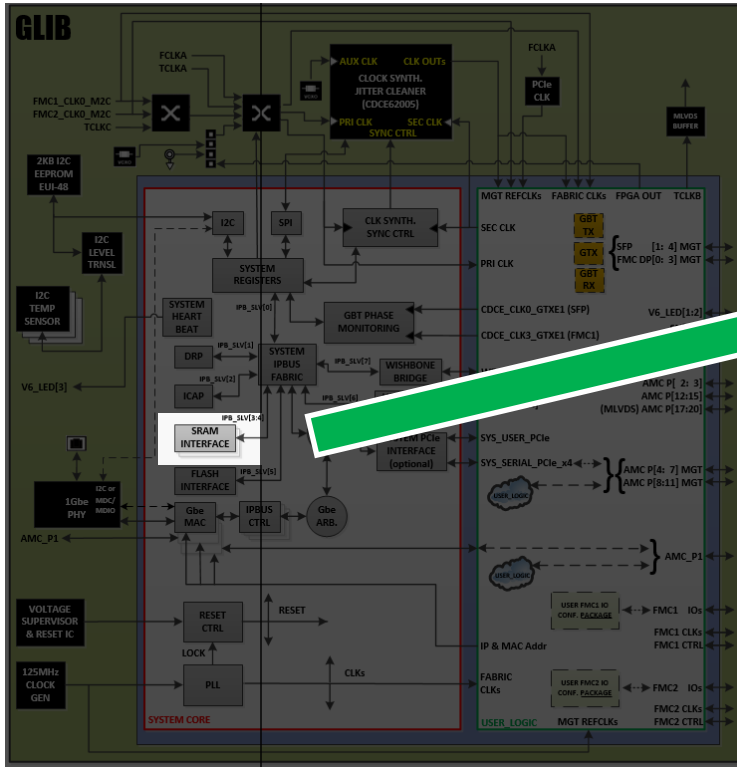
Follow the chain



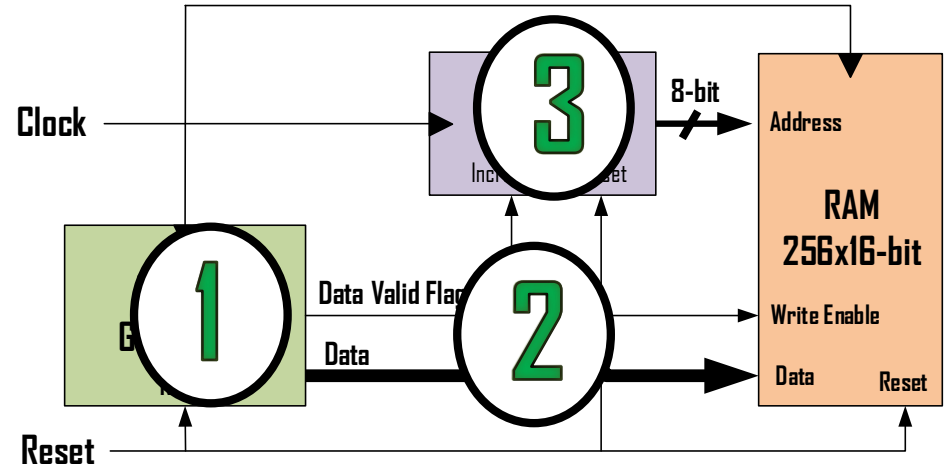
FPGA gateway design work flow

Debugging Techniques

Divide & Conquer



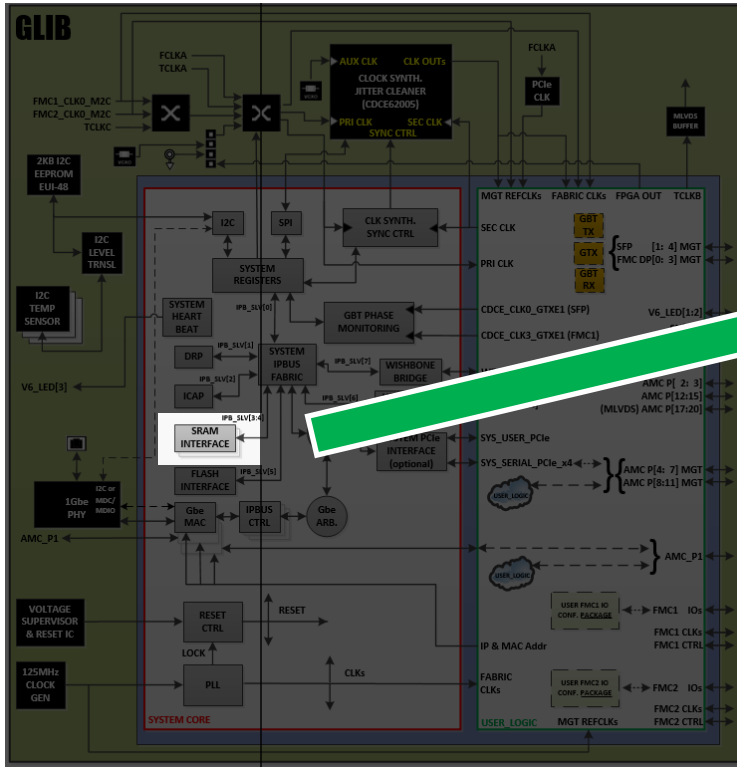
Follow the chain



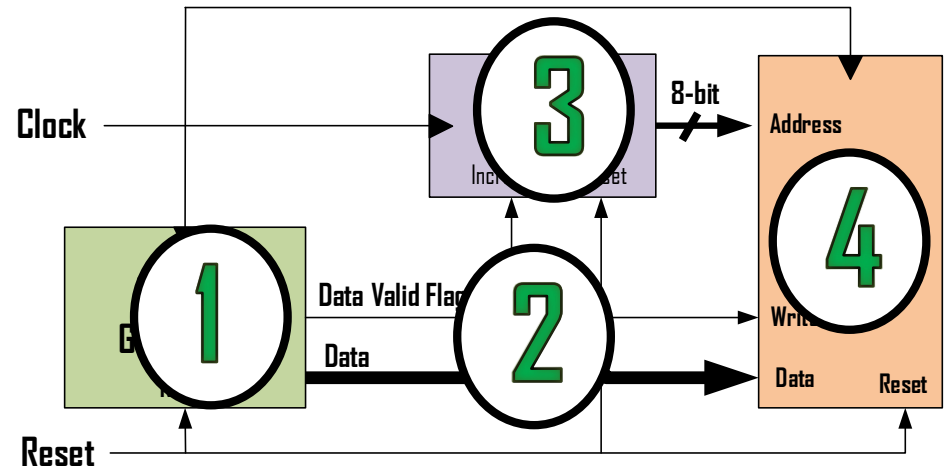
FPGA gateway design work flow

Debugging Techniques

Divide & Conquer



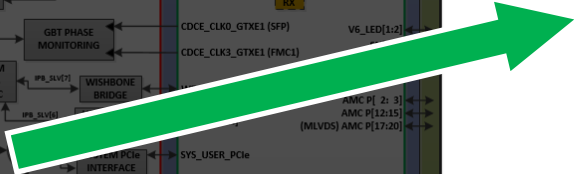
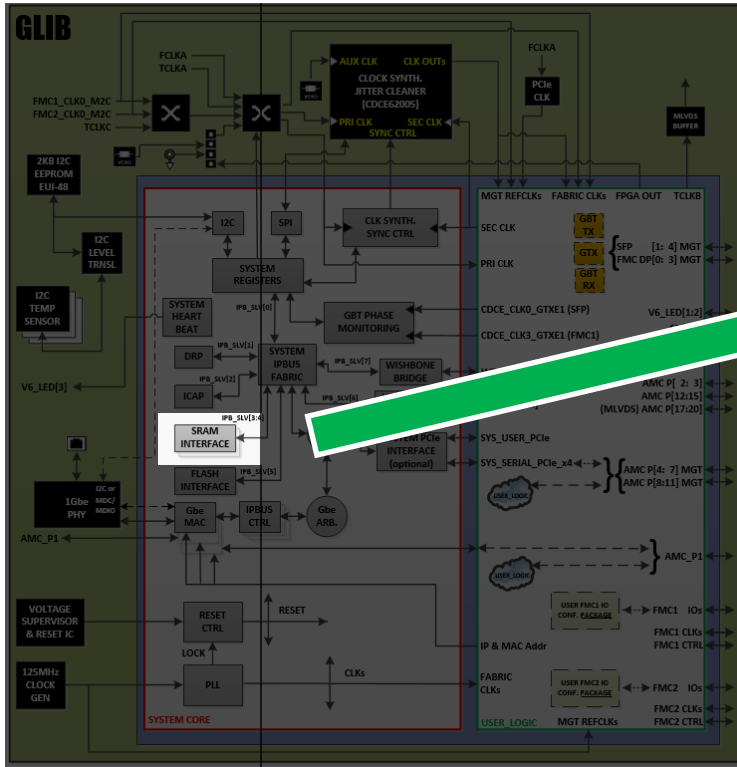
Follow the chain



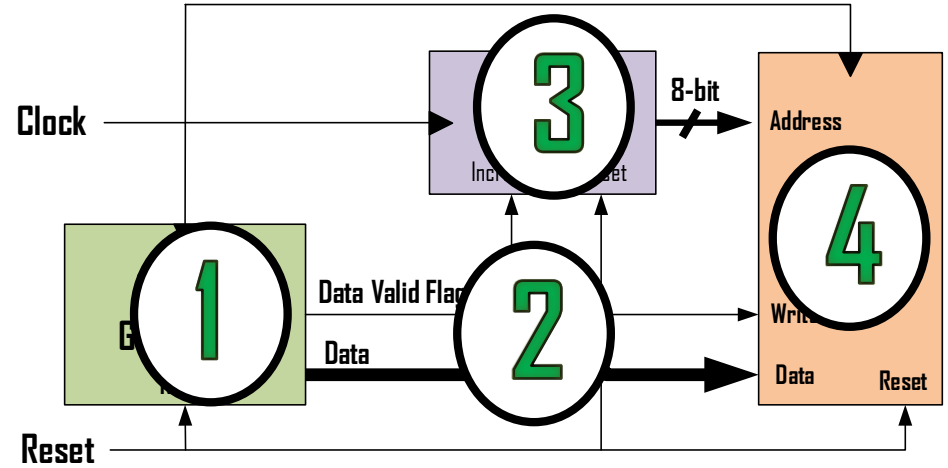
FPGA gateway design work flow

Debugging Techniques

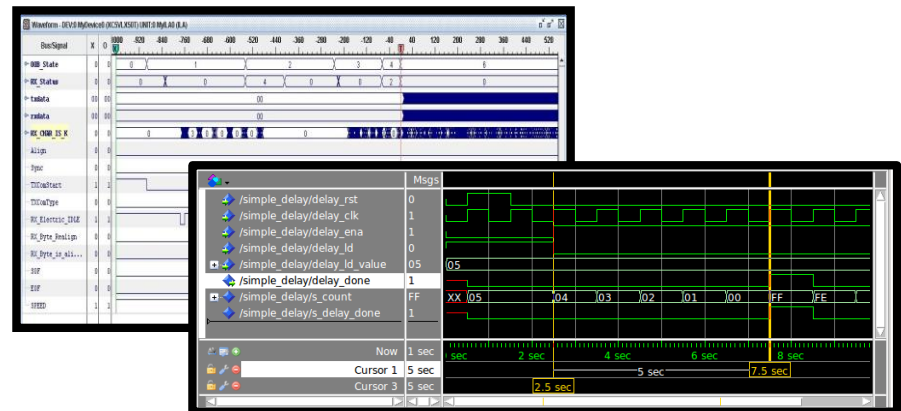
Divide & Conquer



Follow the chain



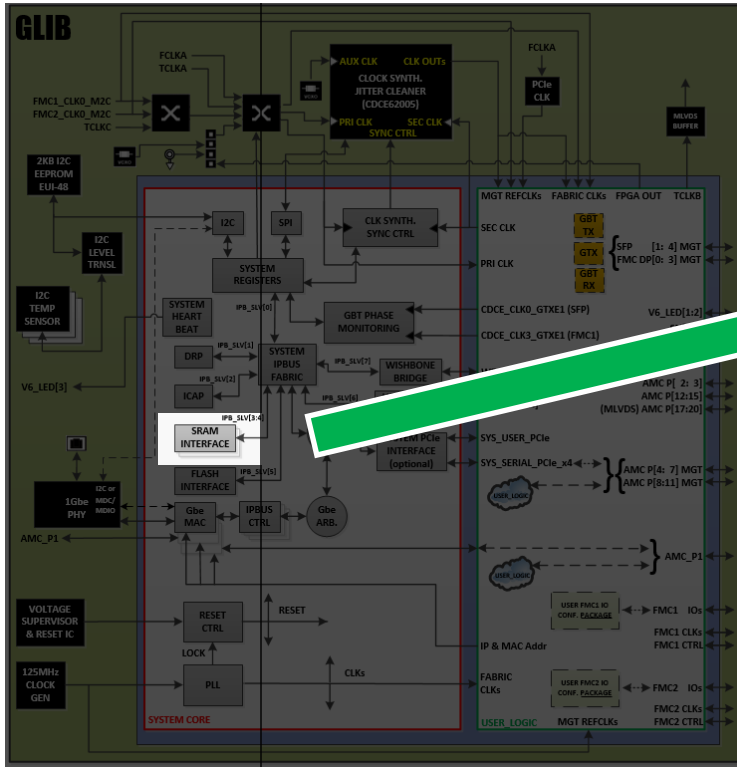
Open the box



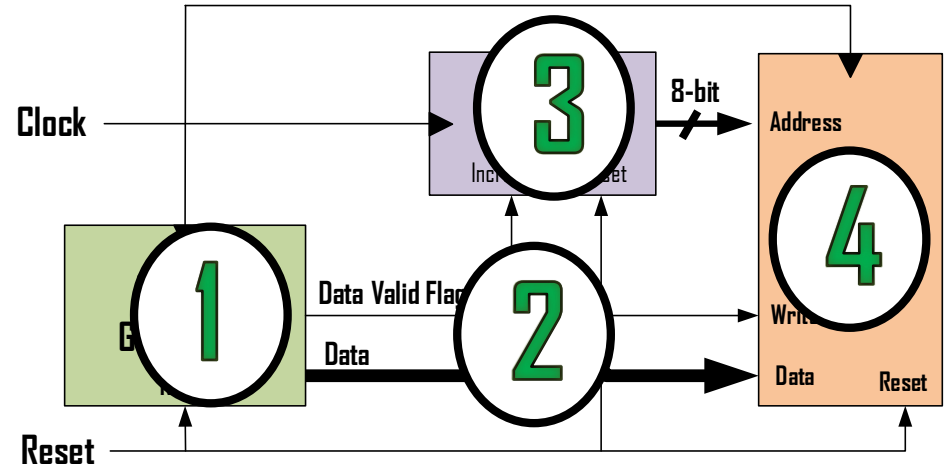
FPGA gateway design work flow

Debugging Techniques

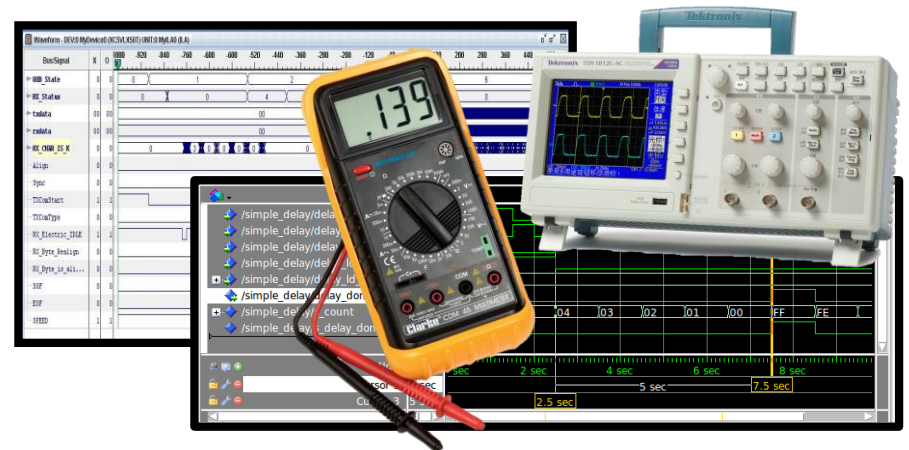
Divide & Conquer



Follow the chain



Open the box



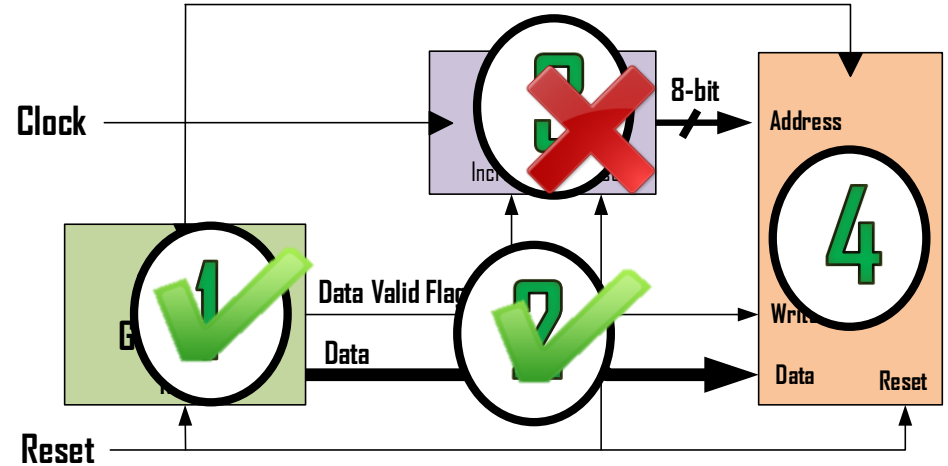
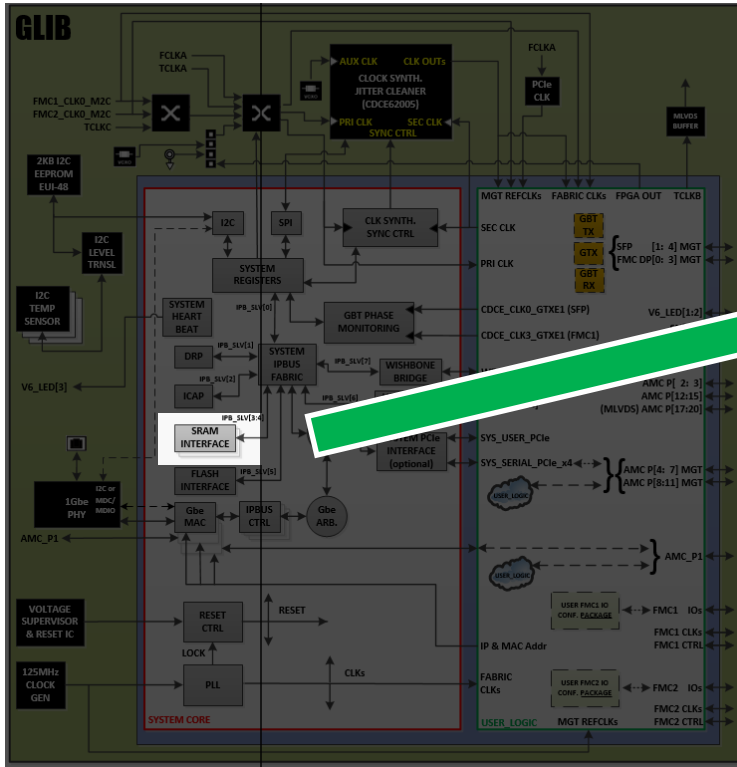
We are debugging **HARDWARE!!!**



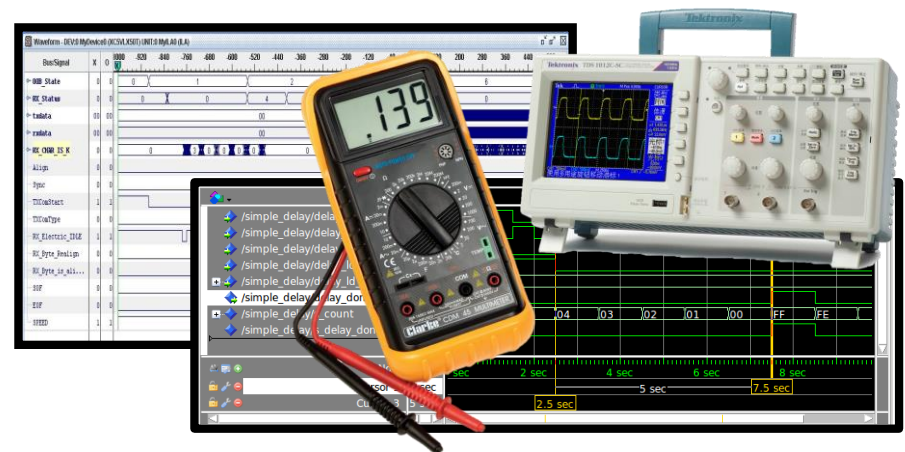
FPGA gateway design work flow

Debugging Techniques

Divide & Conquer



Open the box



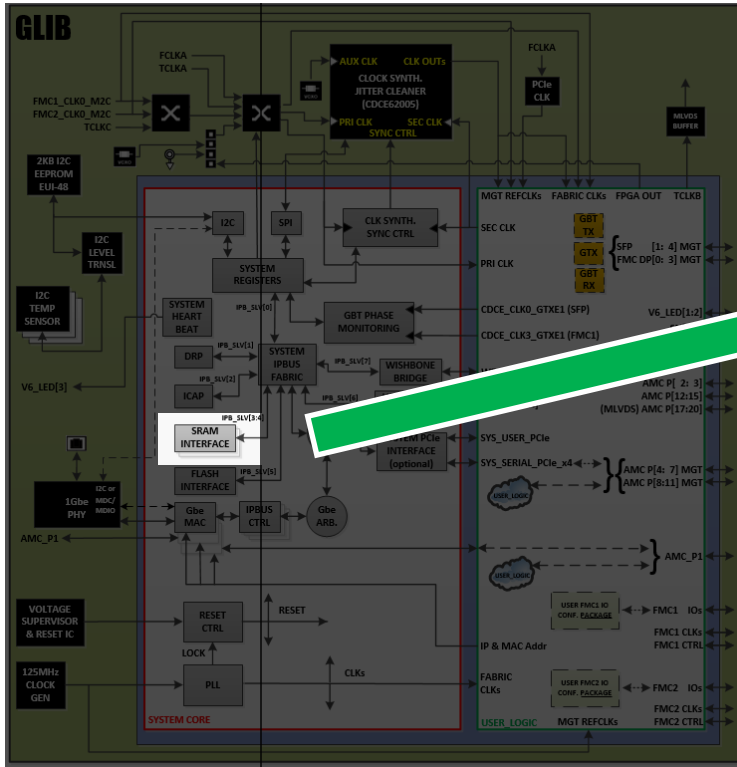
We are debugging **HARDWARE!!!**



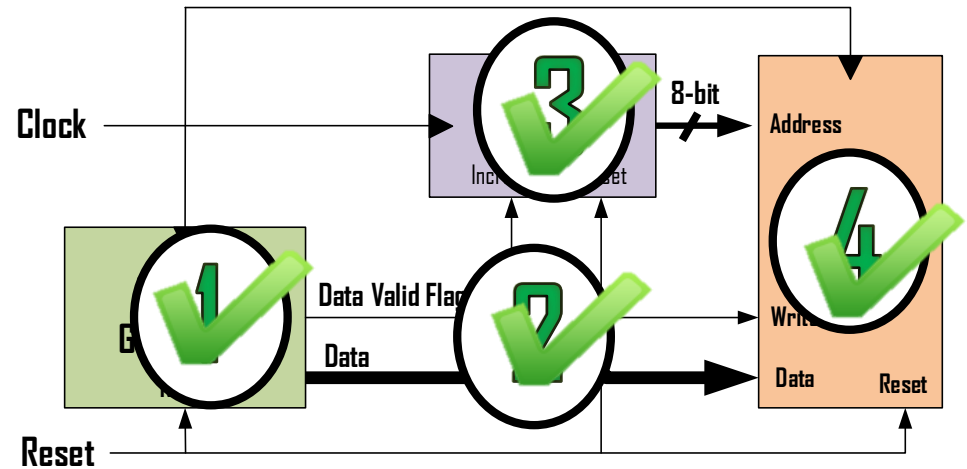
FPGA gateway design work flow

Debugging Techniques

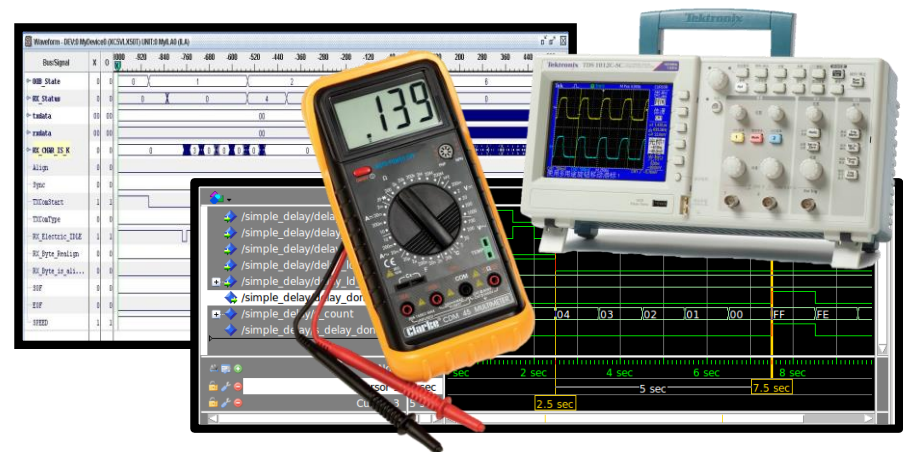
Divide & Conquer



Follow the chain



Open the box



We are debugging **HARDWARE!!!**



FPGA gateway design work flow

After debugging...

FPGA gateway design work flow

After debugging...



FPGA gateway design work flow

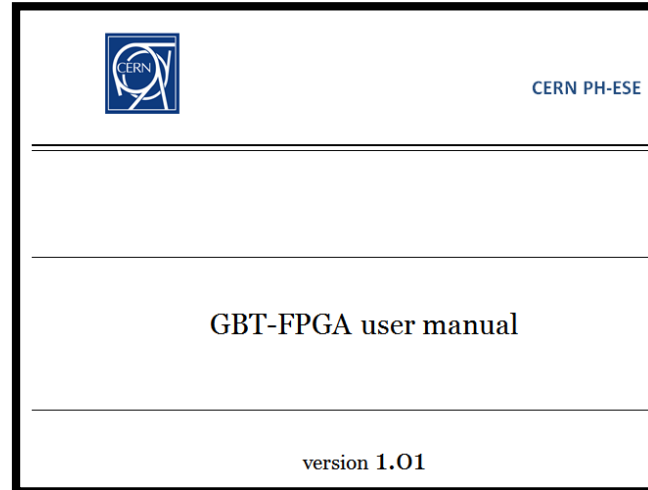
After debugging...



FPGA gateway design work flow

After debugging...

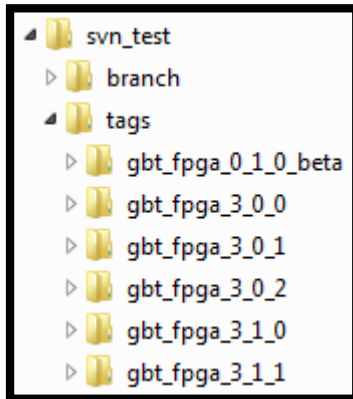
- Documentation



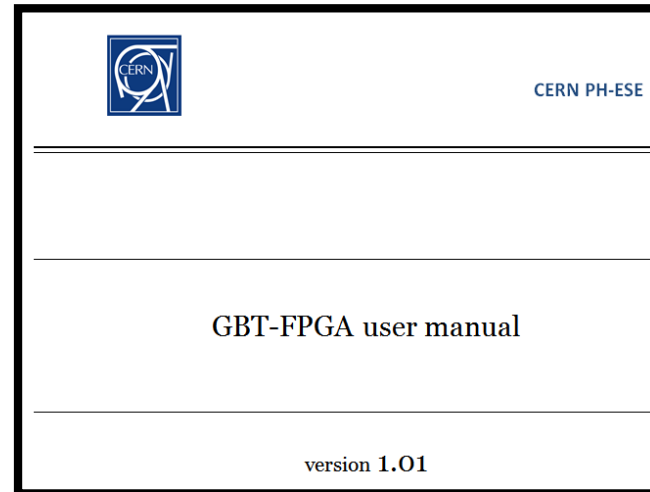
FPGA gateway design work flow

After debugging...

- **Documentation**



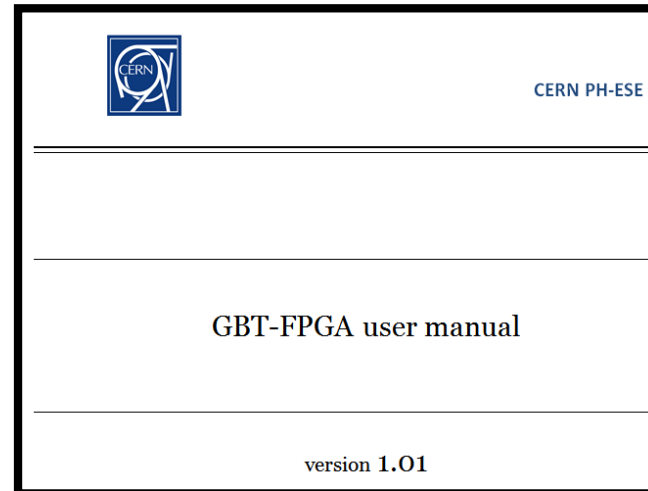
- **Maintenance**



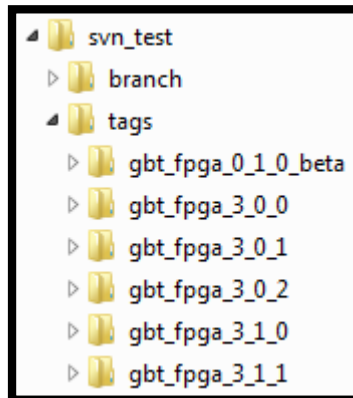
FPGA gateway design work flow

After debugging...

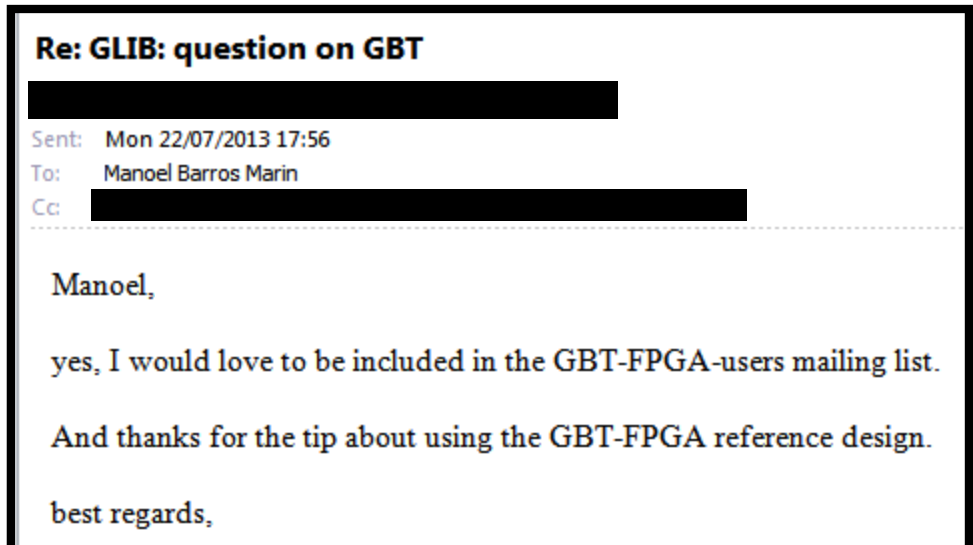
- Documentation



- Maintenance



- ... and maybe User Support



Advanced FPGA design

ISOTDAQ 2020 @ University of Valencia, Valencia (Spain)

22/01/2020

Outline:

- ...from the previous lesson
- Key concepts about FPGA design
- FPGA gateware design work flow
- **Summary**

Manoel Barros Marin



BE-BI-BP

Summary

- **FPGA - Wikipedia**

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".



Summary

- **FPGA - Wikipedia**

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".

- **Key concepts about FPGA design**

- FPGA gateway design is NOT programming
- HDL are used for describing HARDWARE
- Timing in FPGA gateway design is critical



Summary

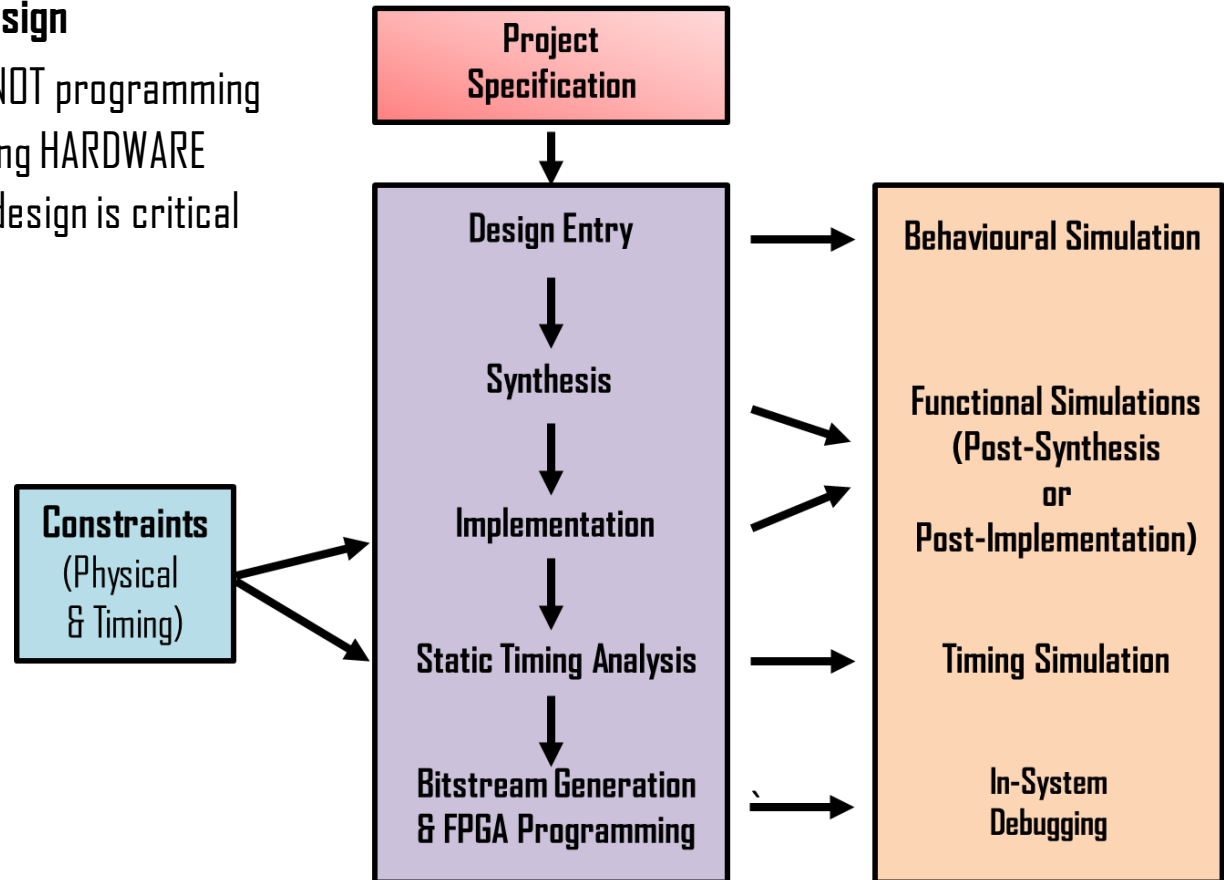
- **FPGA - Wikipedia**

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".

- **Key concepts about FPGA design**

- FPGA gateway design is NOT programming
- HDL are used for describing HARDWARE
- Timing in FPGA gateway design is critical

- **FPGA gateway design flow**



Summary

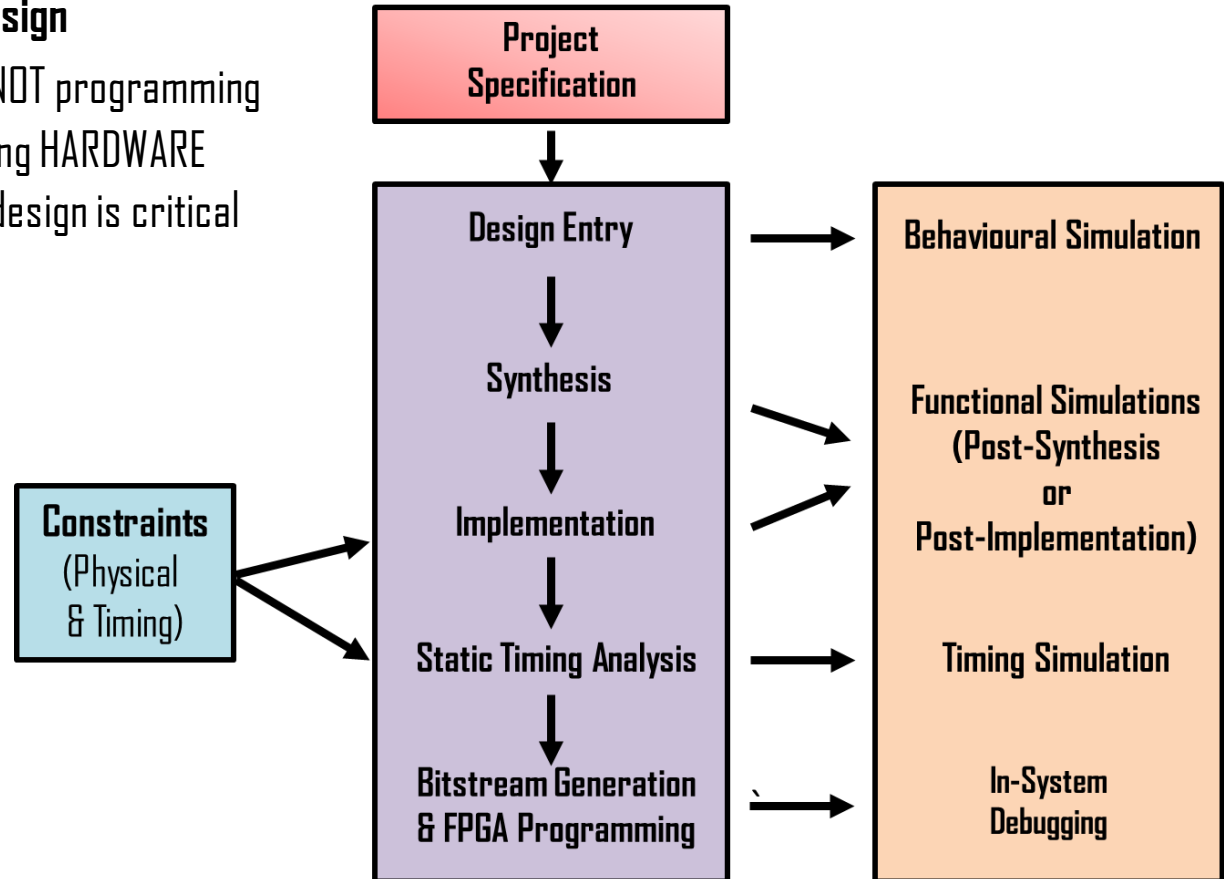
- **FPGA - Wikipedia**

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".

- **Key concepts about FPGA design**

- FPGA gateway design is NOT programming
- HDL are used for describing HARDWARE
- Timing in FPGA gateway design is critical

- **FPGA gateway design flow**



- **A running system is not the end of the road... (Documentation, Maintenance, User Support)**

Summary

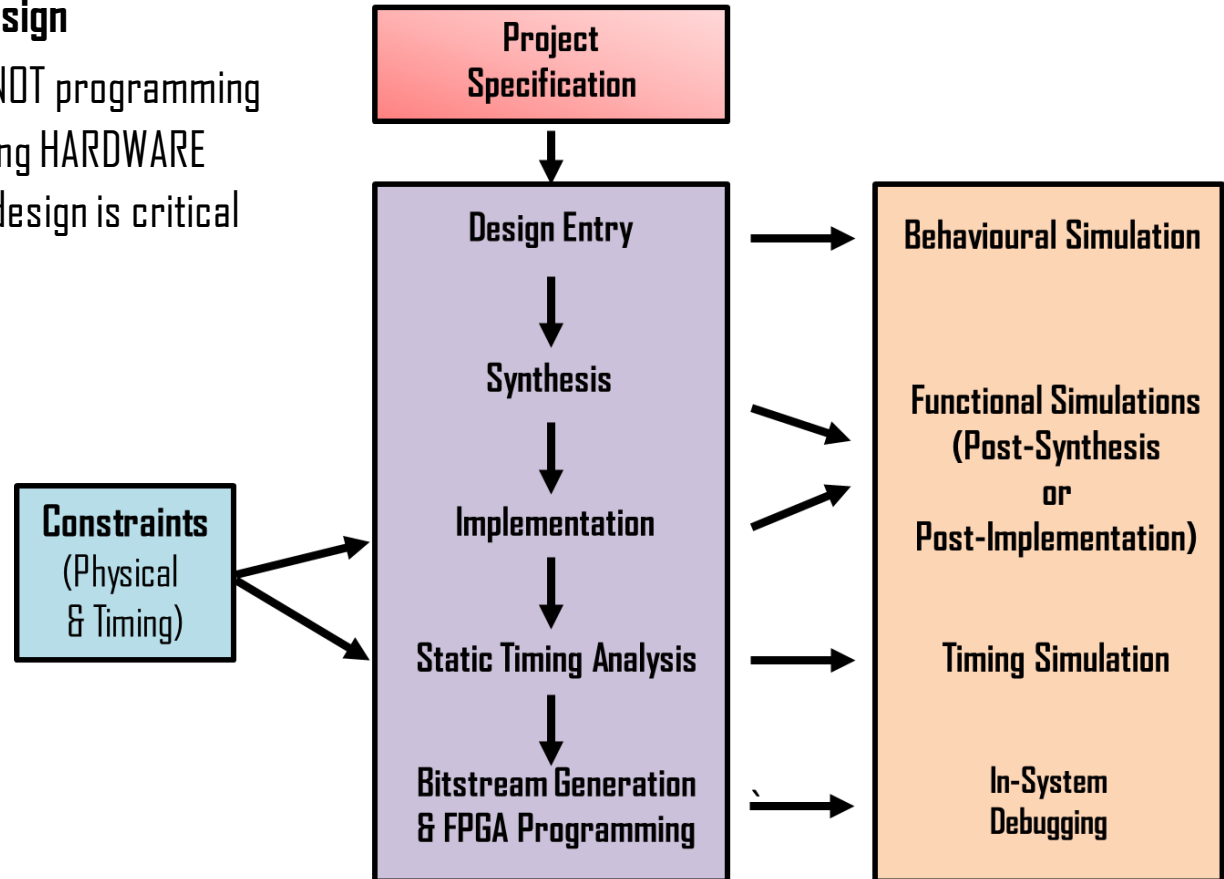
- **FPGA - Wikipedia**

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".

- **Key concepts about FPGA design**

- FPGA gateway design is NOT programming
- HDL are used for describing HARDWARE
- Timing in FPGA gateway design is critical

- **FPGA gateway design flow**



- **A running system is not the end of the road... (Documentation, Maintenance. User Support)**

← **But it works 😊**

Summary

Where do I find more info about this??

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

Acknowledges

- **Markus Joos (CERN), Alberto Valero Biot (IFIC-CSIC) & other organisers of ISOTDAQ-20**
- **Andrea Borga (OpenCores), Torsten Alt (FIAS) for their contribution to this lecture**
- **Rhodri Jones, Manfred Wendt, Andrea Boccardi & other colleagues from CERN BE-BI-BP**

**Any
Question?**

