FPGA Architecture, Technologies, and Tools

Neeraj Goel

IIT Delhi

Plan

- FPGA architecture
  - Basics of FPGA
- FPGA technologies
  - Architectures of different commercial FPGAs
- FPGA tools
  - FPGA implementation flow and software involved
- HDL coding for FPGA
  - Some coding examples and techniques

What is FPGA

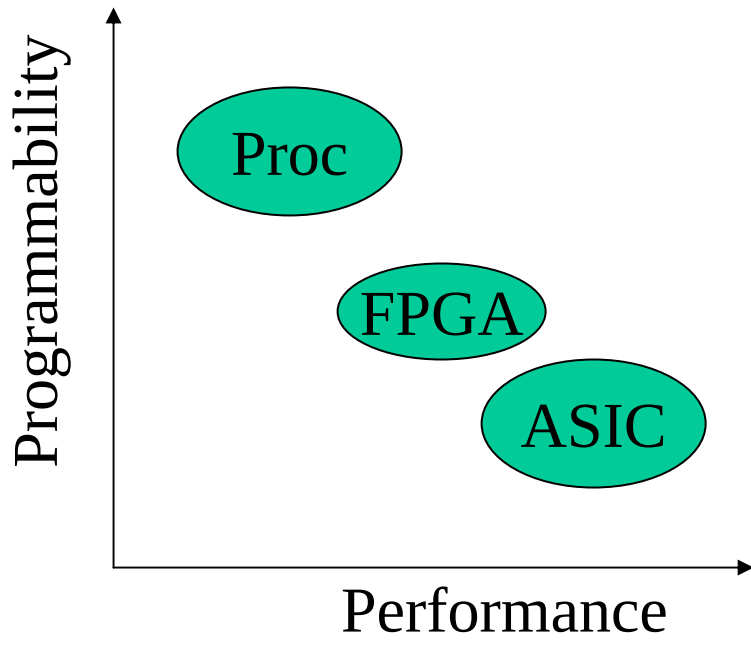
- FPGA – Field Programmable Gate Array
  - A programmable hardware
- Relation between VHDL and FPGA
  - VHDL models hardware and FPGA implements the hardware modeled by VHDL
- Relation between ASIC and FPGA
  - Same in functionality
  - FPGA are reprogrammable

FPGA

- "Field Programmable Gate Array"
- A plane and regular structure in which logic and interconnect both are programmable
- Programmability of logic – any combinational or sequential logic can be implemented
- Programmability of interconnect – any logic component can be connected to anyone else

ASIC verses FPGA

- FPGA
  - Low cost solution
  - Larger area, power and speed
  - Less design and testing time
- ASIC
  - Low cost for large volume
  - Area and power efficient
  - High frequencies can be achieved
  - Huge testing cost in term of time and money



Applications of FPGAs

- Conventional applications
  - For design prototyping
  - For emulation
- New applications
  - As hardware acceralator
  - In place of ASIC
    - Less time to market
  - Complete System on Chip (SoC) solution

Programming technology

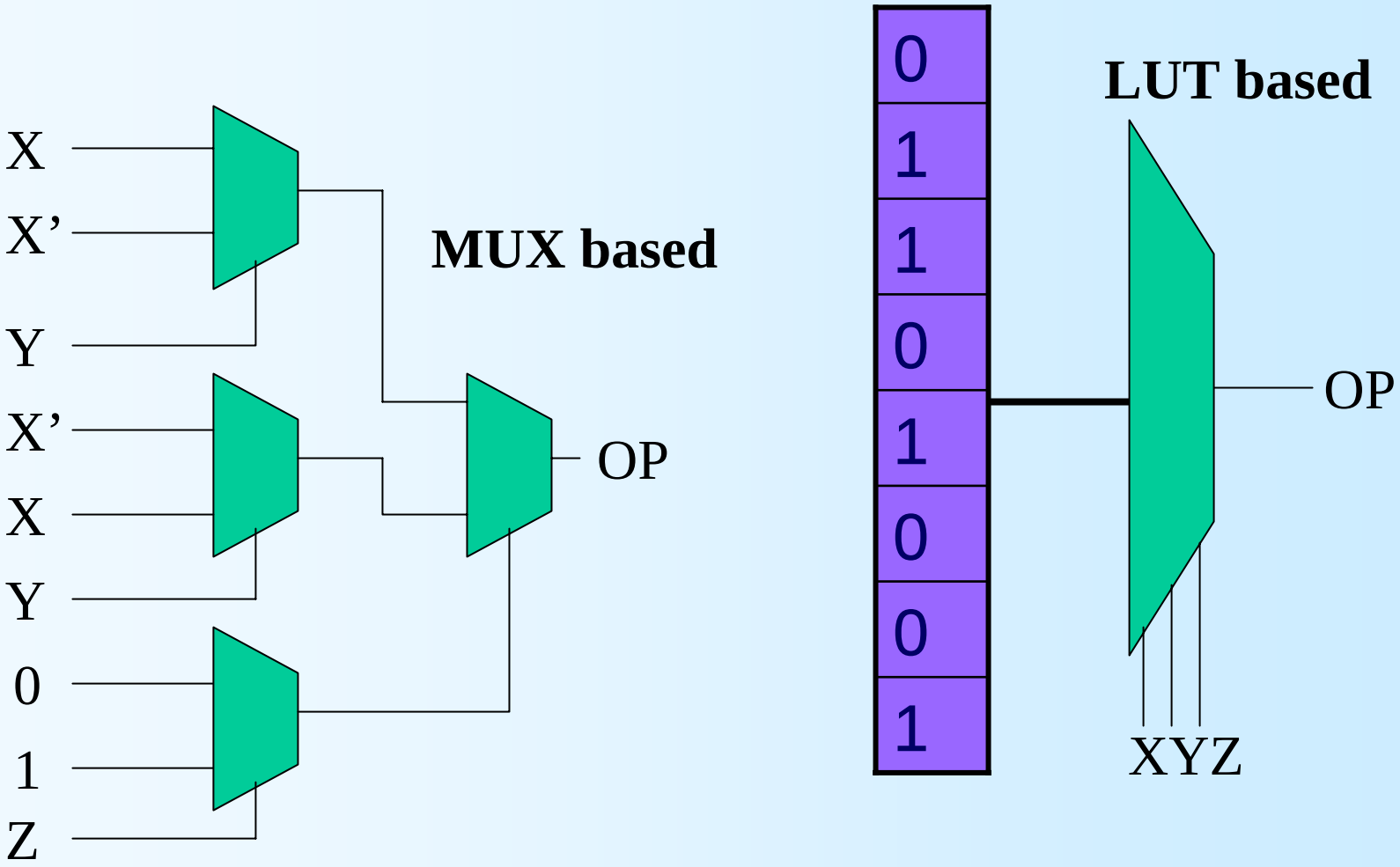
- Anti-fuse based
  - All the contacts or open initially
  - Programming converts selected locations as conducting
  - One time programmable (OTP)
- SRAM based
- E^2ROM or Flash based
- Tradeoffs
  - Anti-fuse is less area, less power consuming
  - E^2RAM takes more time for programming
  - SRAM is technology leaders

Programmable Logic

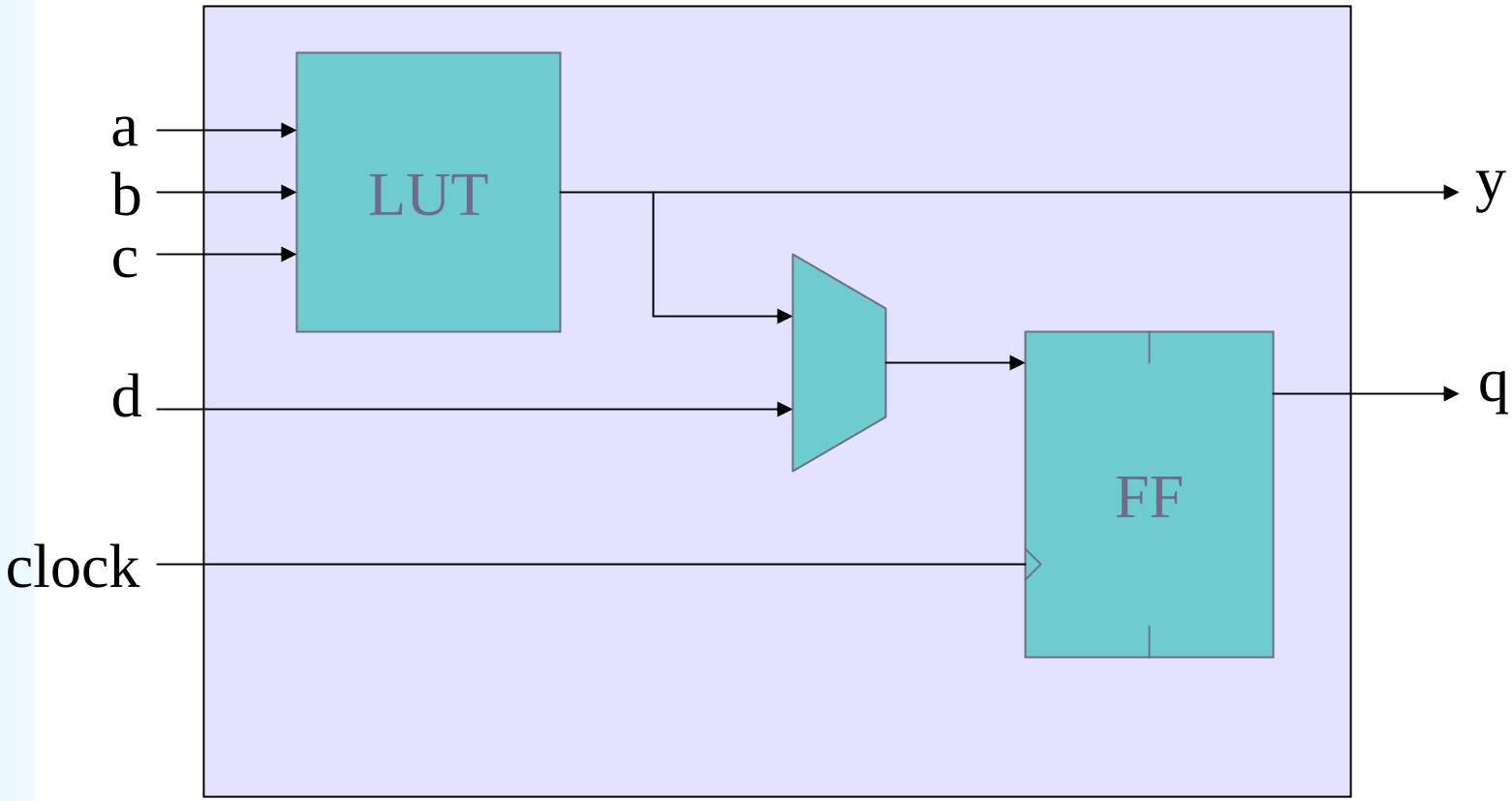
- Fine grain “fabric”
  - A universal gate like NAND or AND-OR-NOT
- Middle grain
  - Multiplexer based
  - ROM/RAM based
- Coarse grain
  - FFT or a processor as a basic unit
- Tradeoffs
  - Fine grain FPGA involves more interconnection overhead
  - Coarse grain are application specific

Programmable Logic

- Op = X xor Y xor Z



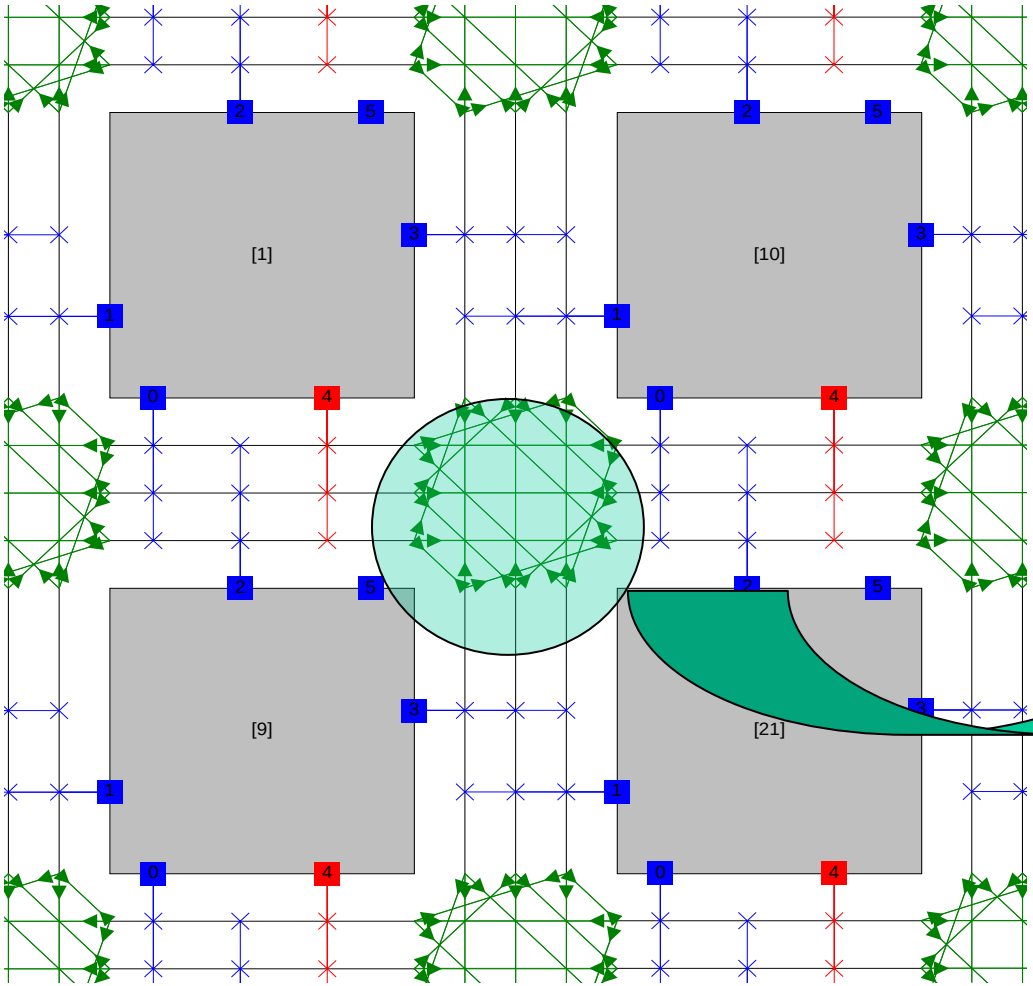
A simple programmable logic block

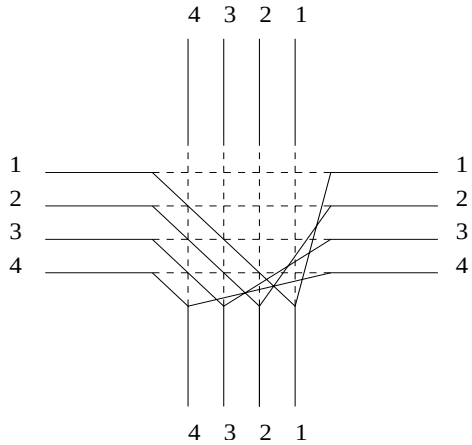


Programmable interconnects

- Connection box
  - Connects input/output of logic block to interconnect channels
- Switch box
  - Connects horizontal channels to vertical channels
- Transmission gate (or a pass transistor) is used for each connection

Interconnections

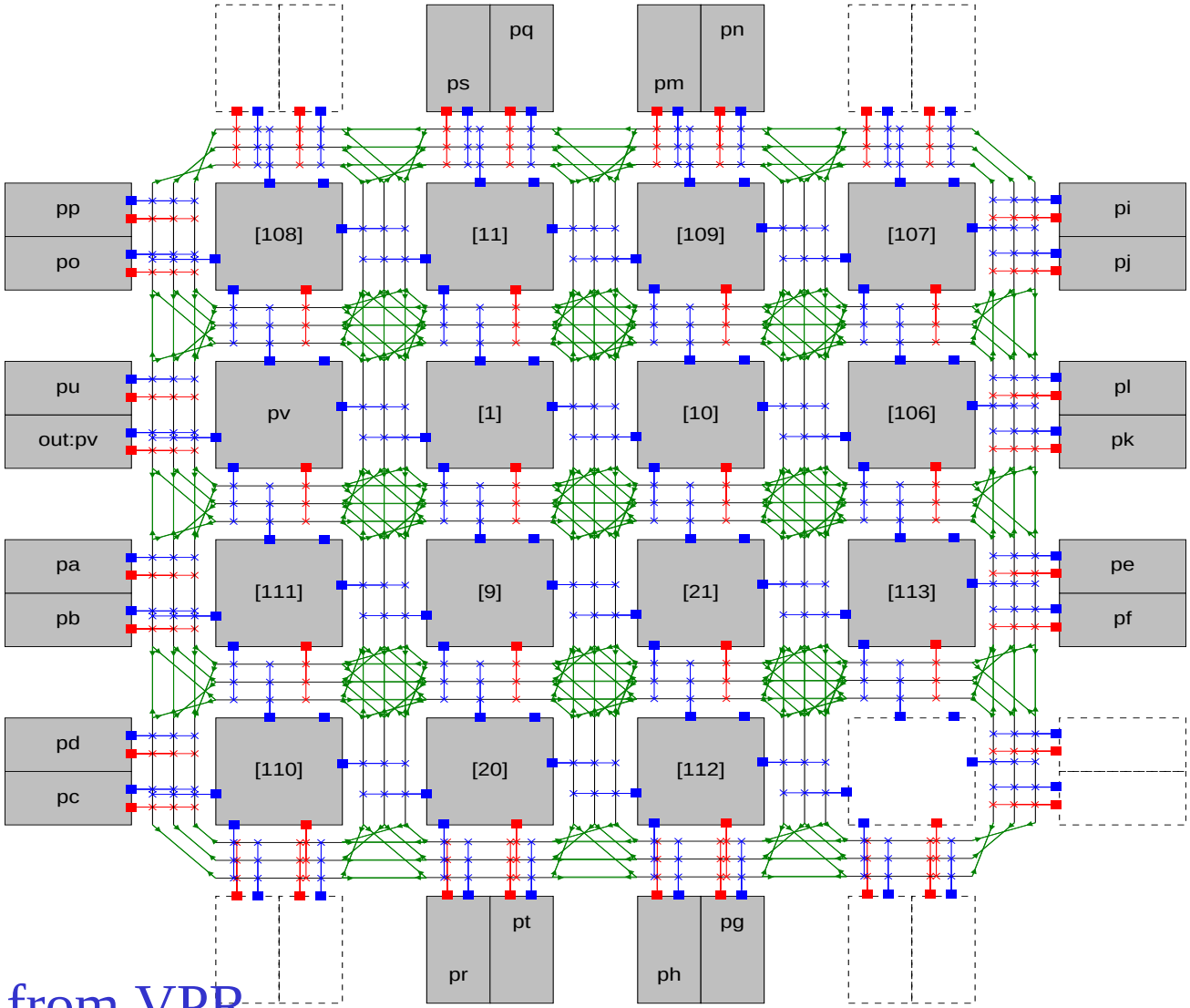




A switch box

A snapshot from VPR

Top view of a simple FPGA Architecture



A snapshot from VPR

Review and questions

- Is FPGA an ASIC?
- Can we implement an processor in FPGA?
- Are PLAs same as FPGA?
- The companies which produce FPGA?
- Why FPGAs are important to our VLSI?
- Do we need to study FPGA internals?

Questions?

Plan

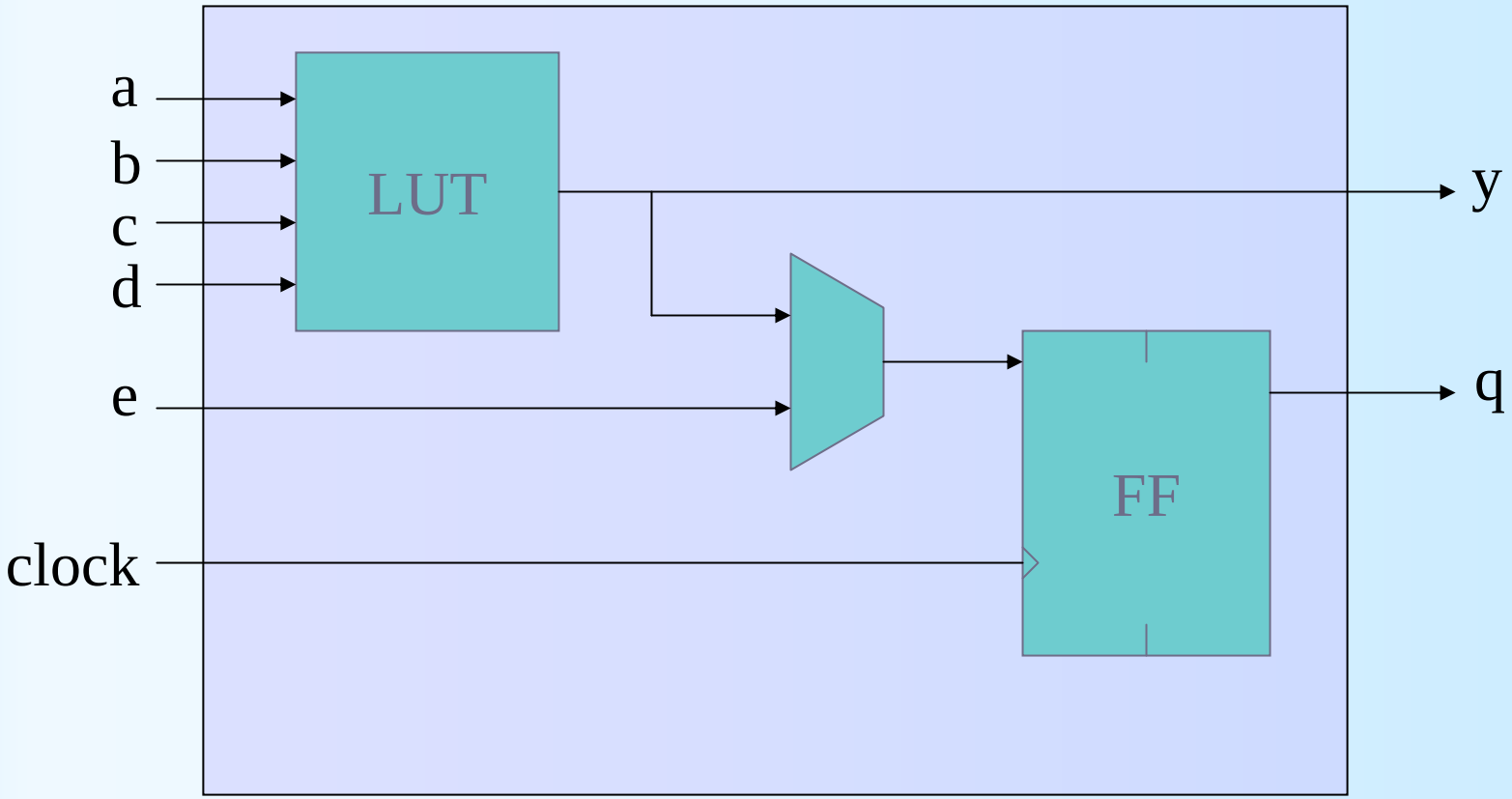
- FPGA architecture
  - Basics of FPGA
- FPGA technologies
  - Architectures of different commercial FPGAs
- FPGA tools
  - FPGA implementation flow and software involved
- HDL coding for FPGA
  - Some coding examples and techniques

Advanced FPGA Architectures

- Companies
  - Xilinx
  - Altera
  - Actel
  - Amtel
  - Quicklogic

Xilinx FPGA Architecture

- Basic blocks are a logical cell

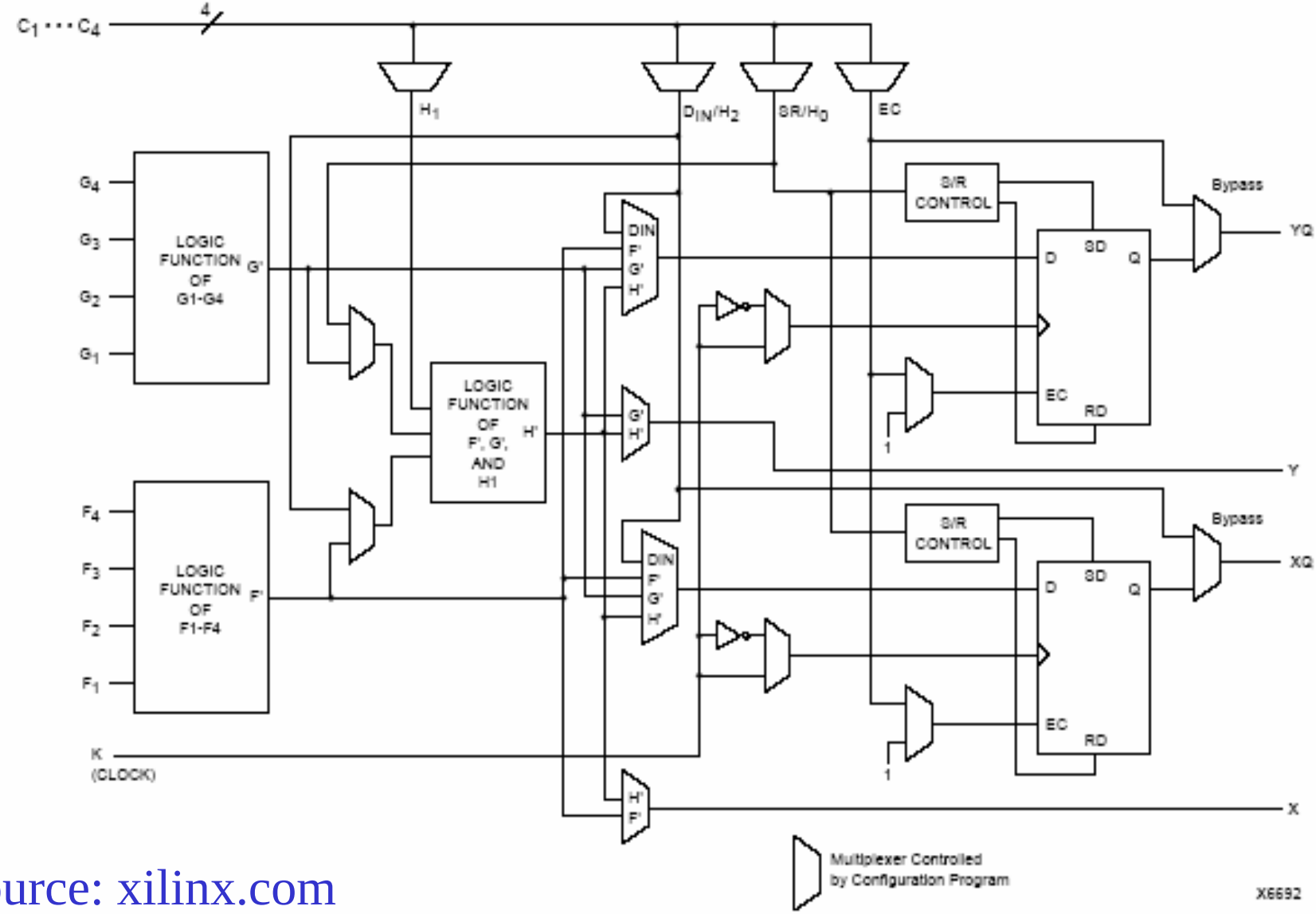


- A 4 input LUT can also act as 16x1 RAM or Shift register

Xilinx FPGA Architecture

- Basic blocks are a logical cell
- A slice comprise of two logic cells
- A configurable logic block (CLB) may have upto 4 slices
  - CLB of XC4000 series have 1 slice
  - CLB of virtex series have 2 or 4 slices
- A hierarchical structure help in reducing interconnections
  - Interconnections are costly resource in FPGA

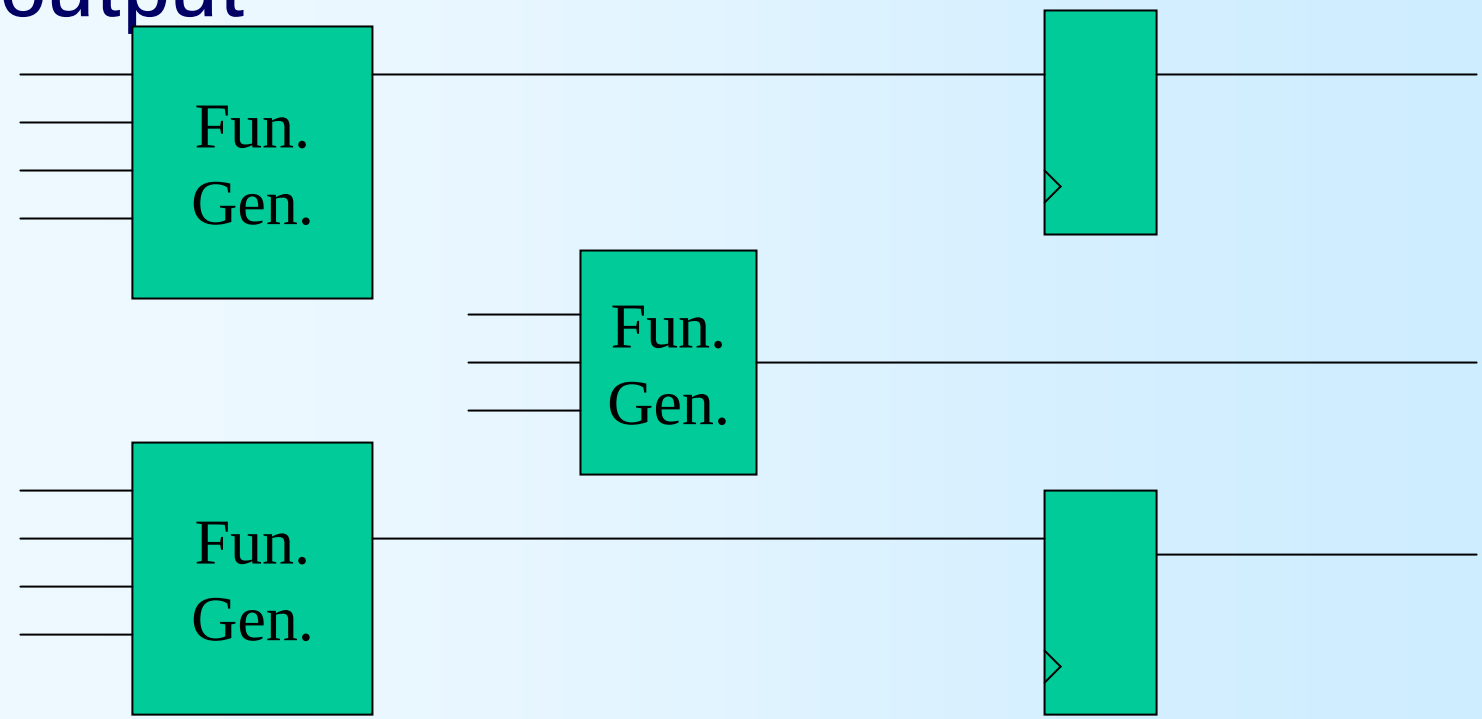
Xilinx FPGA Architecture: a CLB in XC4000



Source: xilinx.com

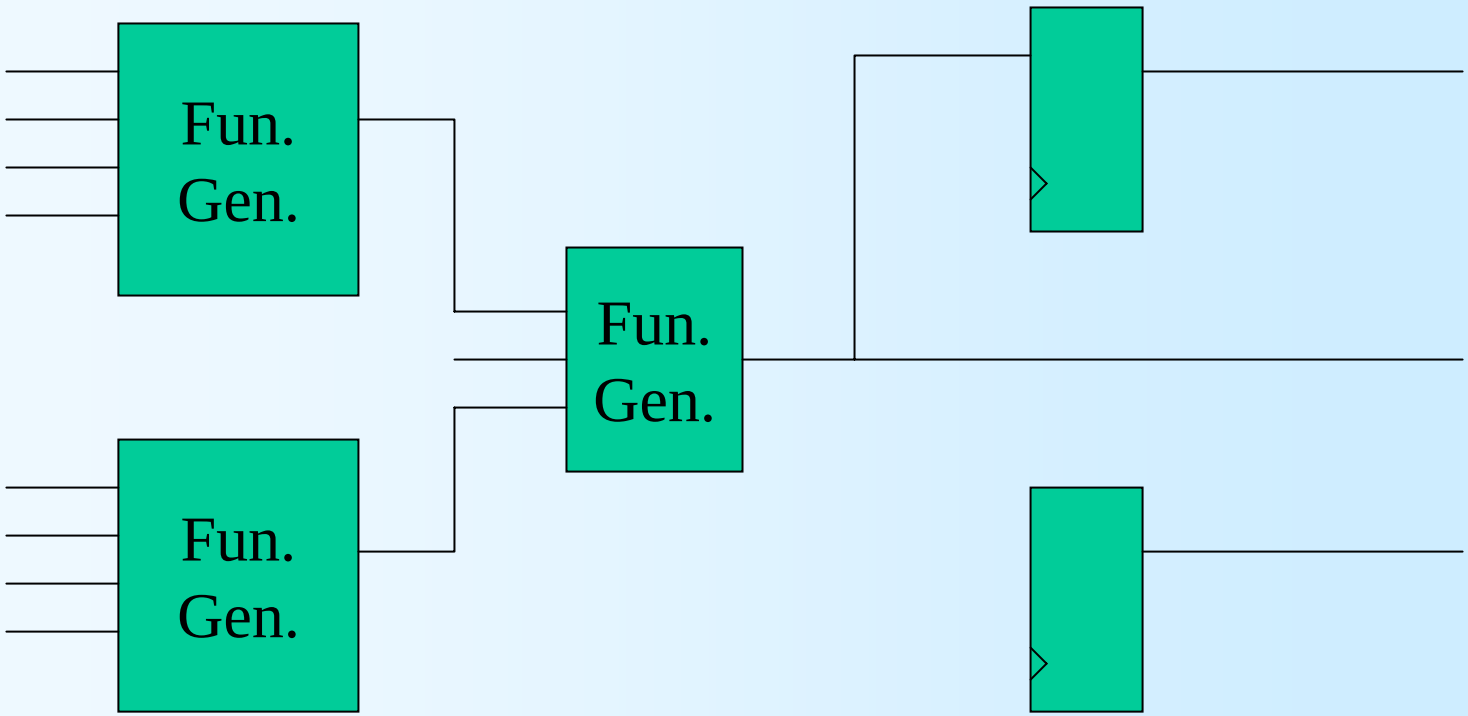
Xilinx FPGA Architecture: a CLB in XC4000

- Two 4-input and one 3-input function generator
- Two latched outputs and two unlatched output



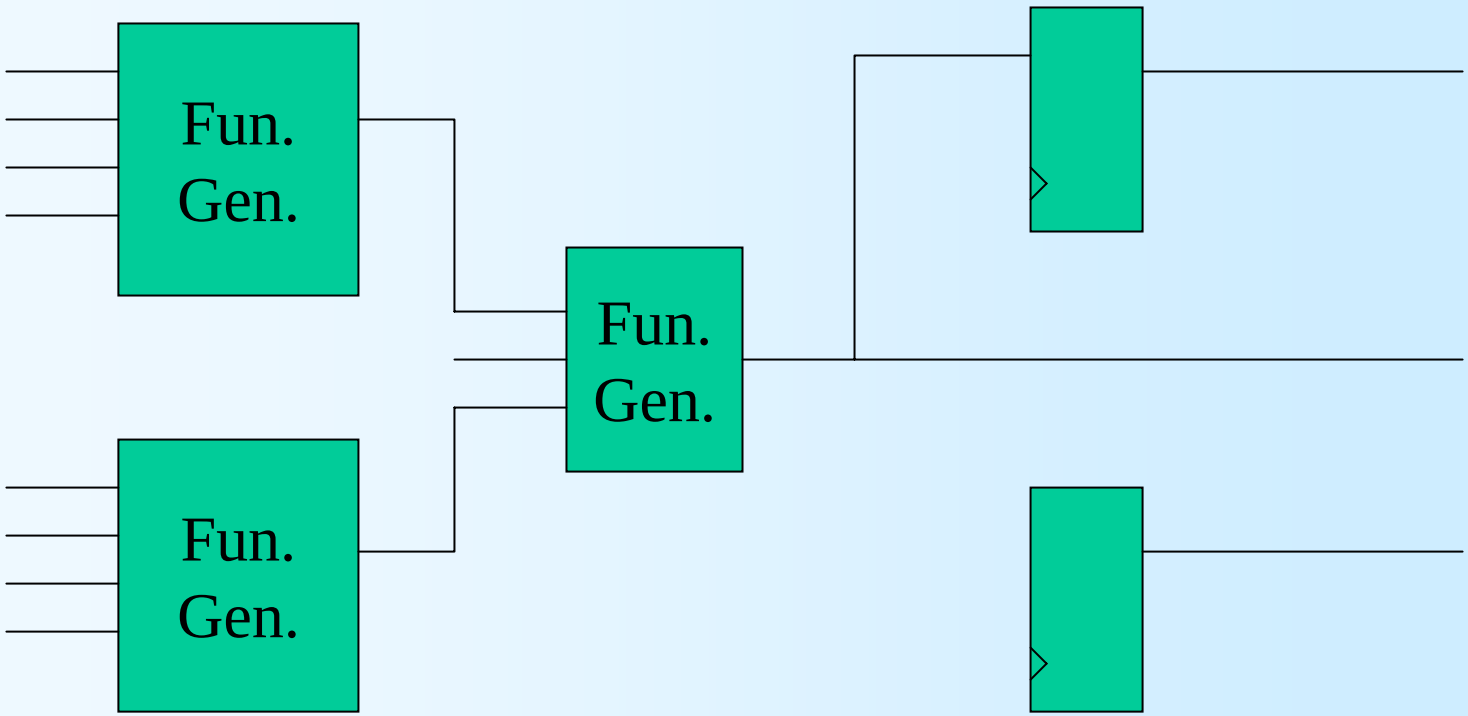
Xilinx FPGA Architecture: a CLB in XC4000

- One 9-input function generator
- Latched or unlatched output



Xilinx FPGA Architecture: a CLB in XC4000

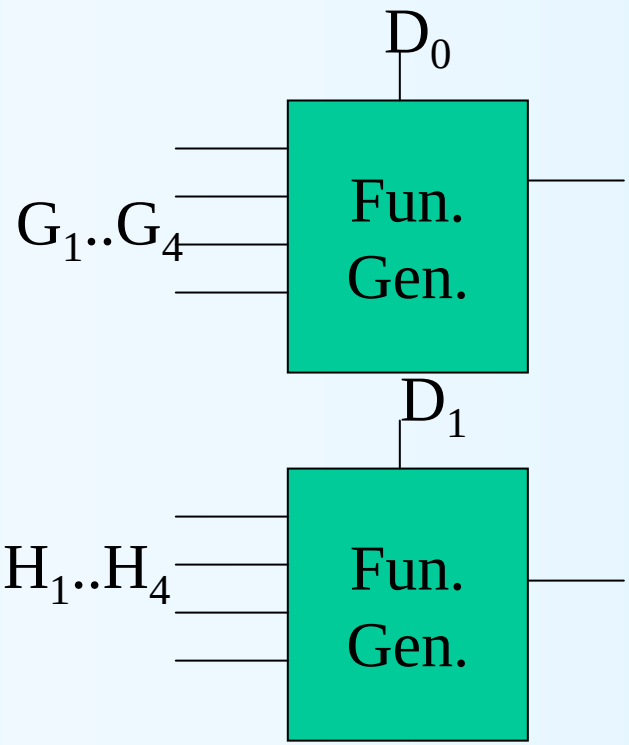
- One 9-input function generator
- Latched or unlatched output



Xilinx FPGA Architecture: a CLB in XC4000

function generator as RAM

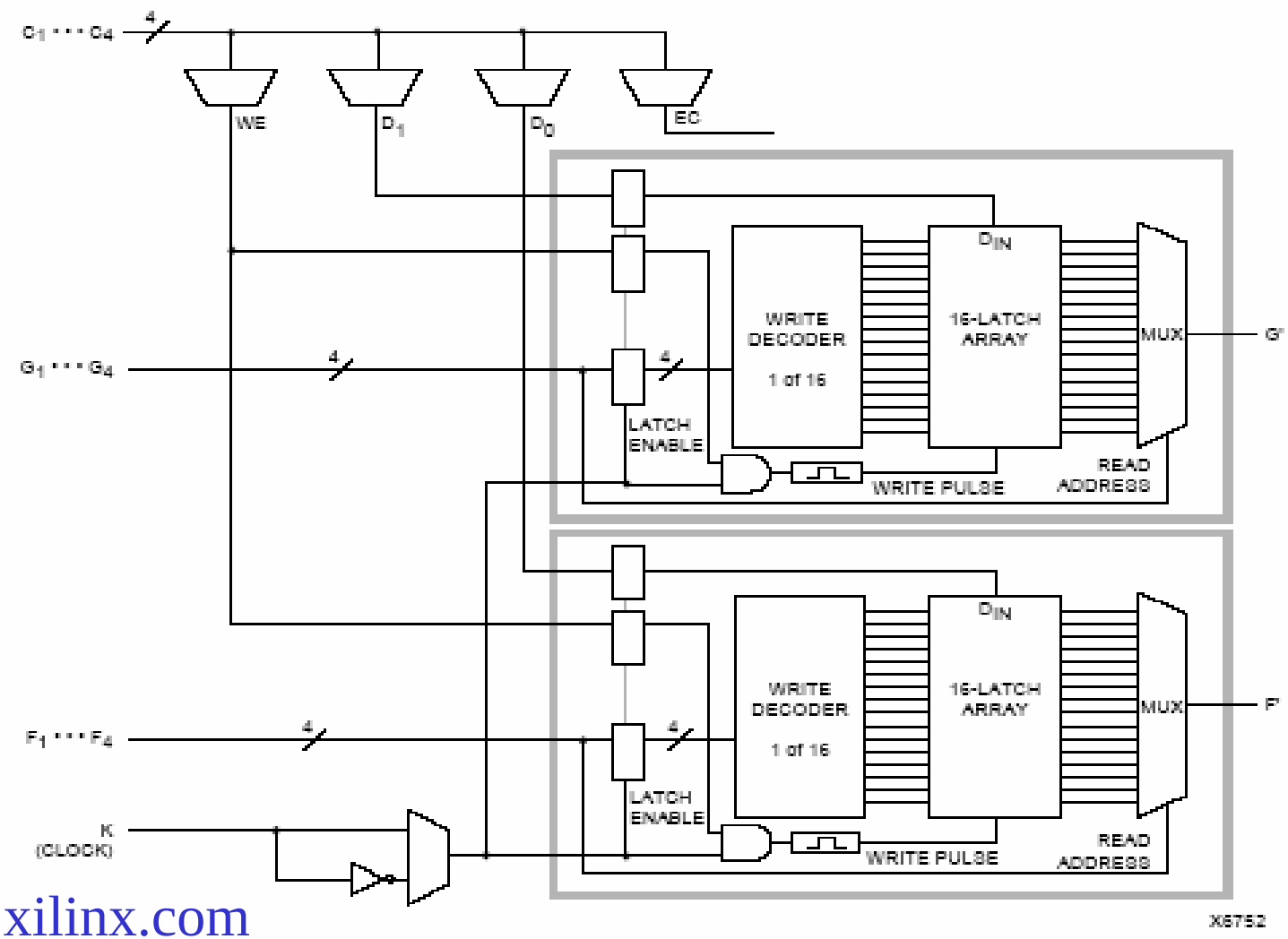
- Level triggered, edge triggered, single port, dual port
- 16x2, 32x1, 16x1 bit array



16x2 single port bit array

Xilinx FPGA Architecture: a CLB in XC4000

- function generator as 16x2 edge triggered single port RAM



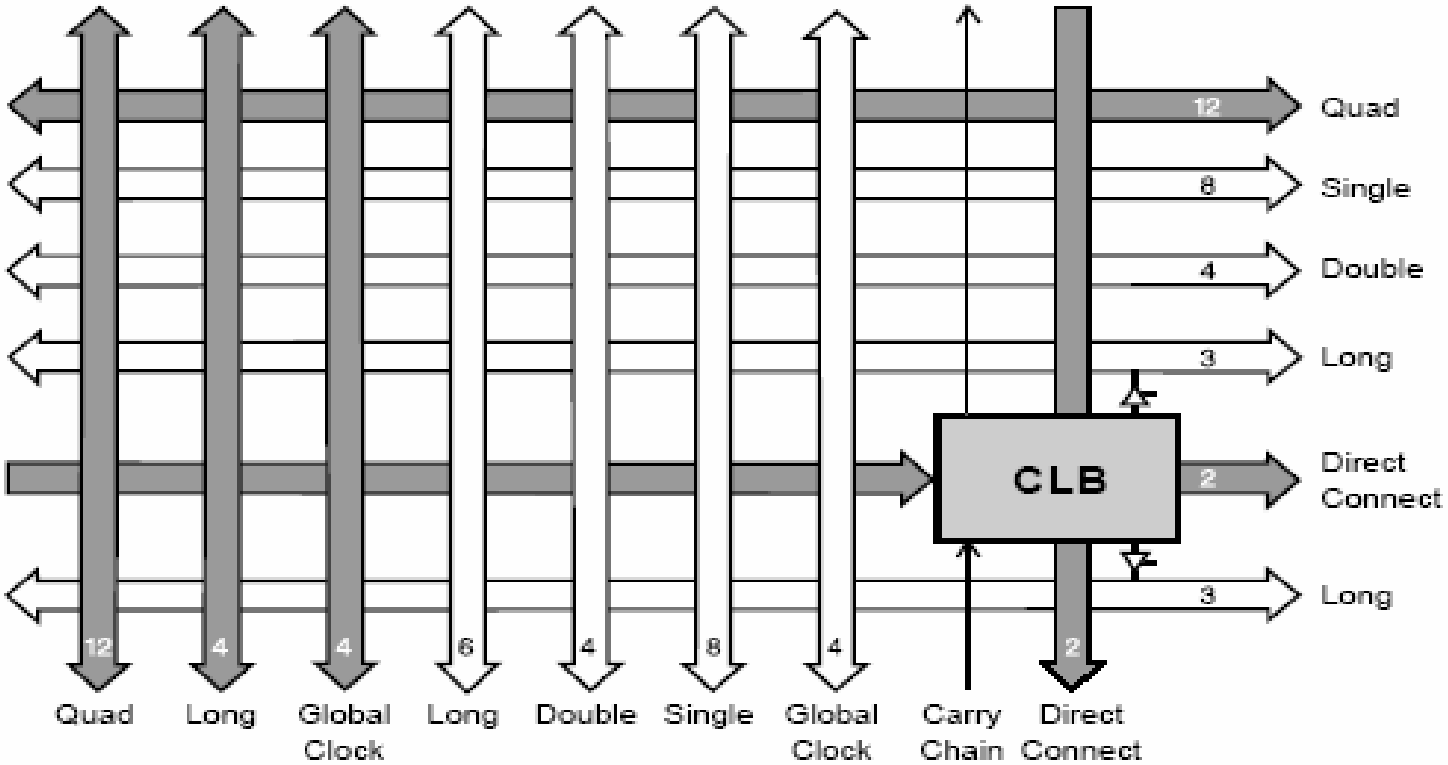
Source: xilinx.com

Xilinx FPGA Architecture: a CLB in XC4000

- Fast carry chains
  - Dedicated logic in F and G function generators for fast carry generation
  - Dedicated routing resources for carry chains

Xilinx FPGA Architecture: Interconnections

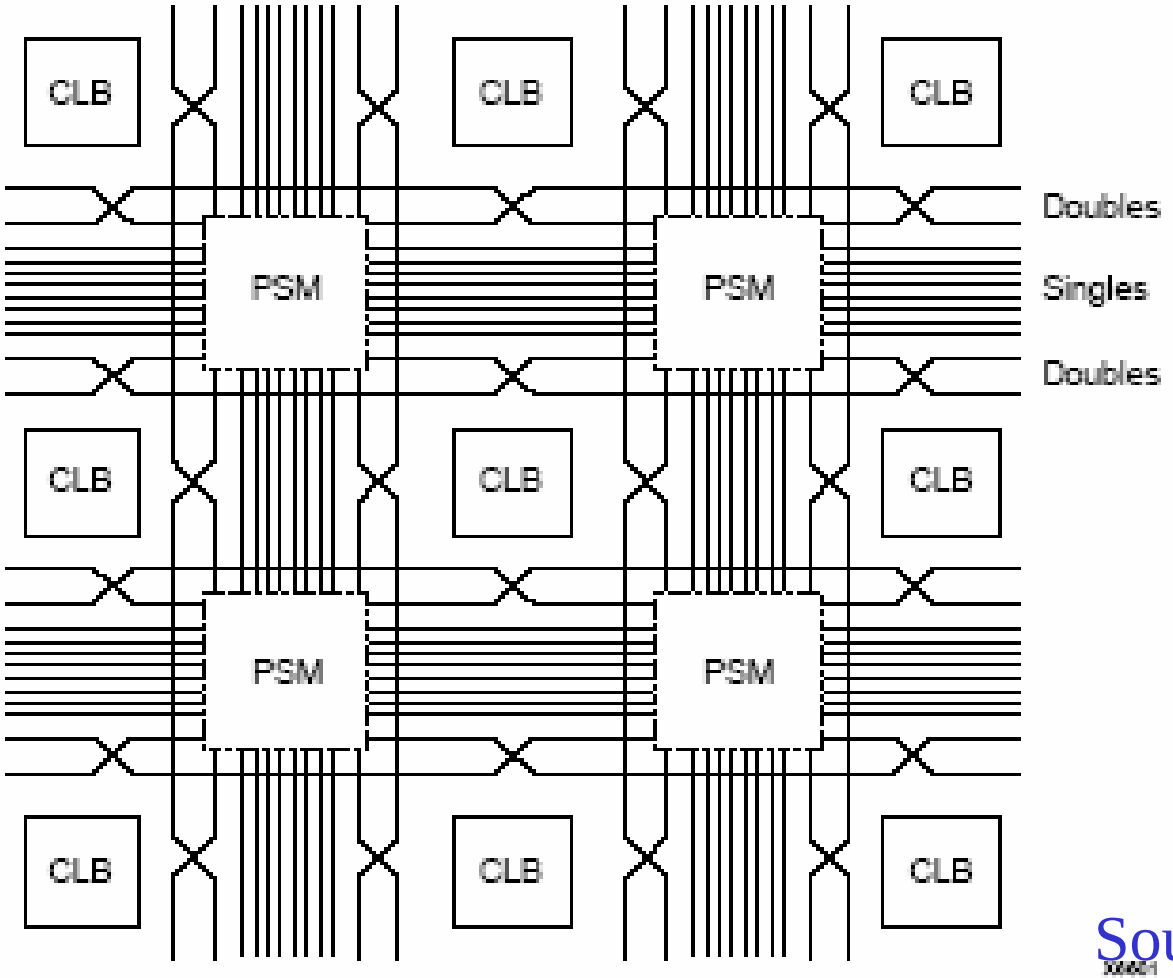
- Five type of interconnection based on length
  - Single length lines, double length lines, Quad, Octal and long lines



Source: xilinx.com

Xilinx FPGA Architecture: Interconnections

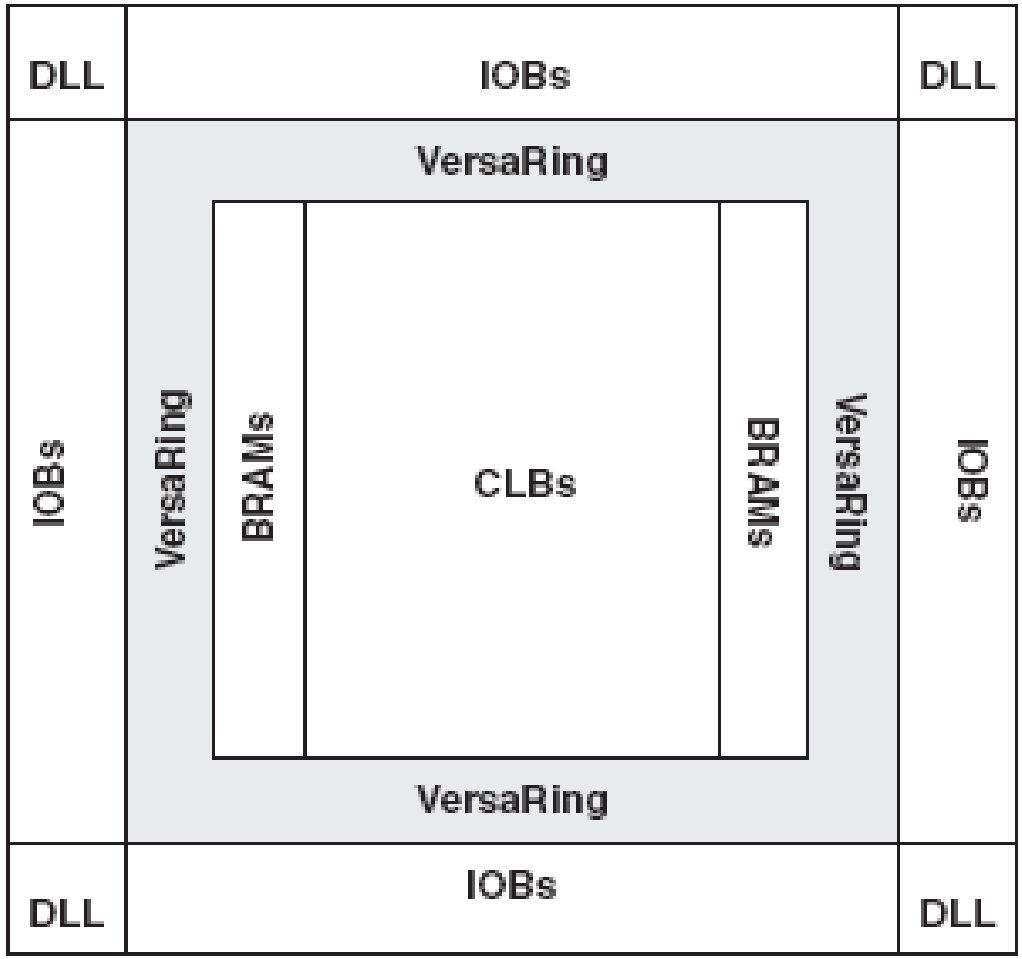
- Single and double lines with programmable switch box



Source: xilinx.com

Xilinx FPGA Architecture: Virtex array

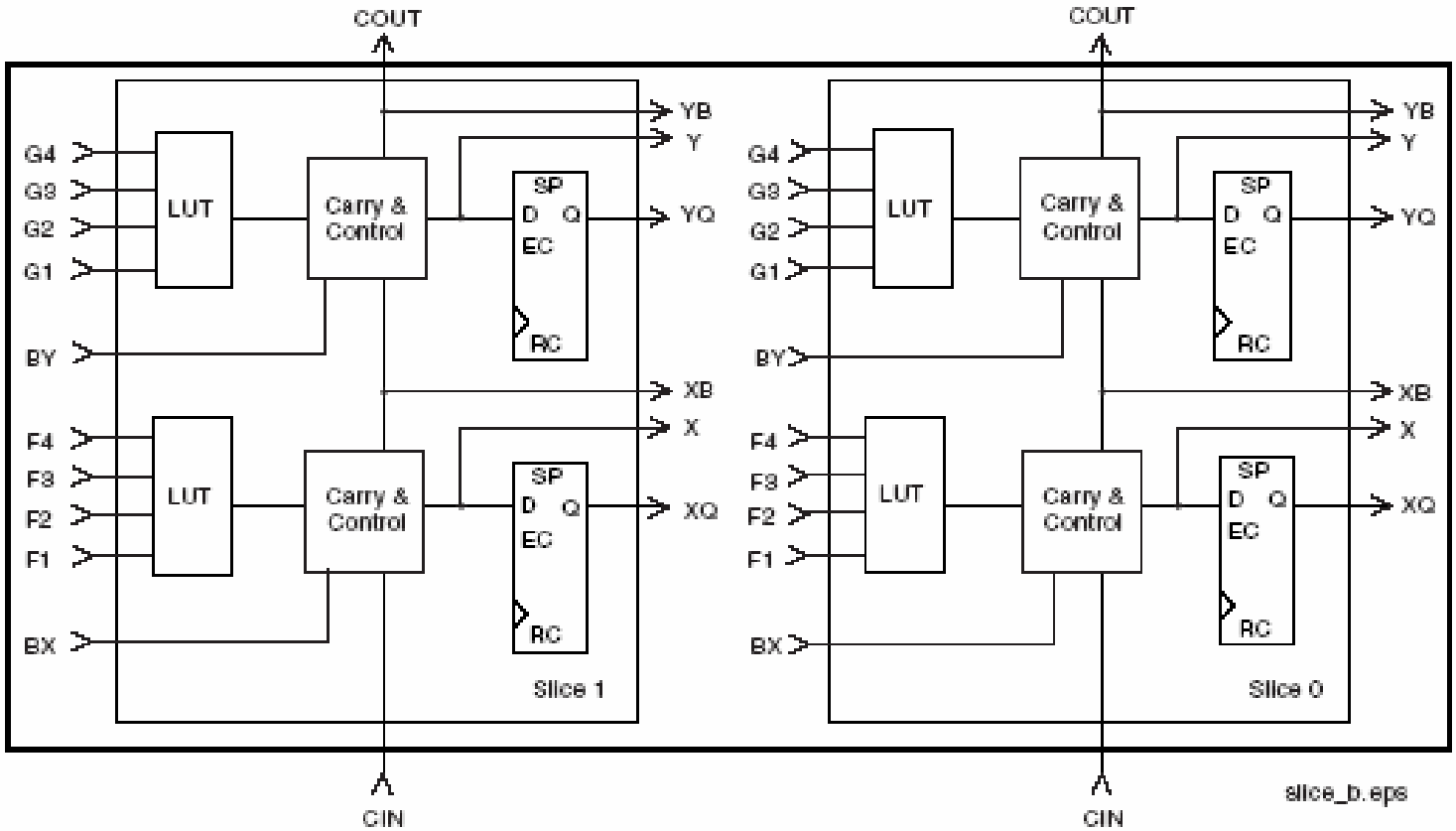
- Architecture overview



Source: xilinx.com

Xilinx FPGA Architecture: Virtex array

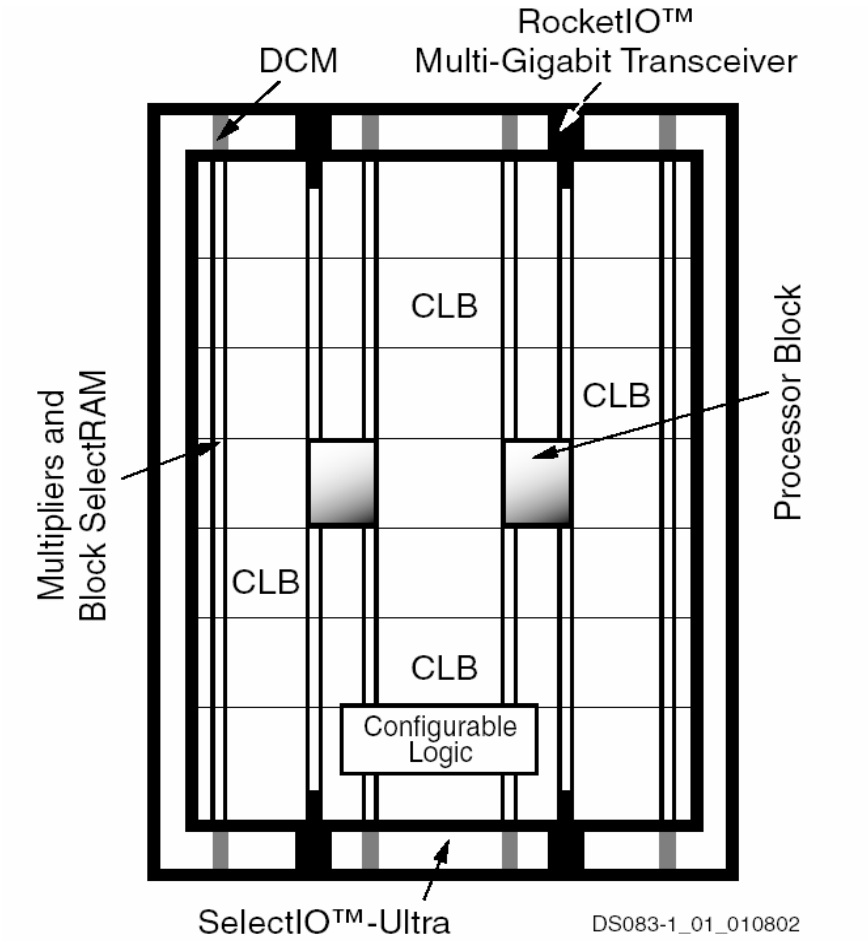
- One CLB – 2 slice



Source: xilinx.com

Xilinx FPGA Architecture: Platform Computing

- Latest FPGA features
  - 4 slices in a CLB
  - Block RAM
  - Embedded multiplier and DSP block
  - Embedded processors
    - PowerPC, a hard core
    - Microblaze a soft core
  - Other interface cores
  - Gbps rocket IO
  - Partial reconfigurability



Altera FPGA families

- Similar to Xilinx FPGAs
  - Basic block is LE (logic element)
  - Basic unit is LAB (Logic array block) equivalent to CLB
- Platform computing
  - MegaRAM®
  - DSP block having embedded multiplier
  - Nios® embedded processor

Review and questions

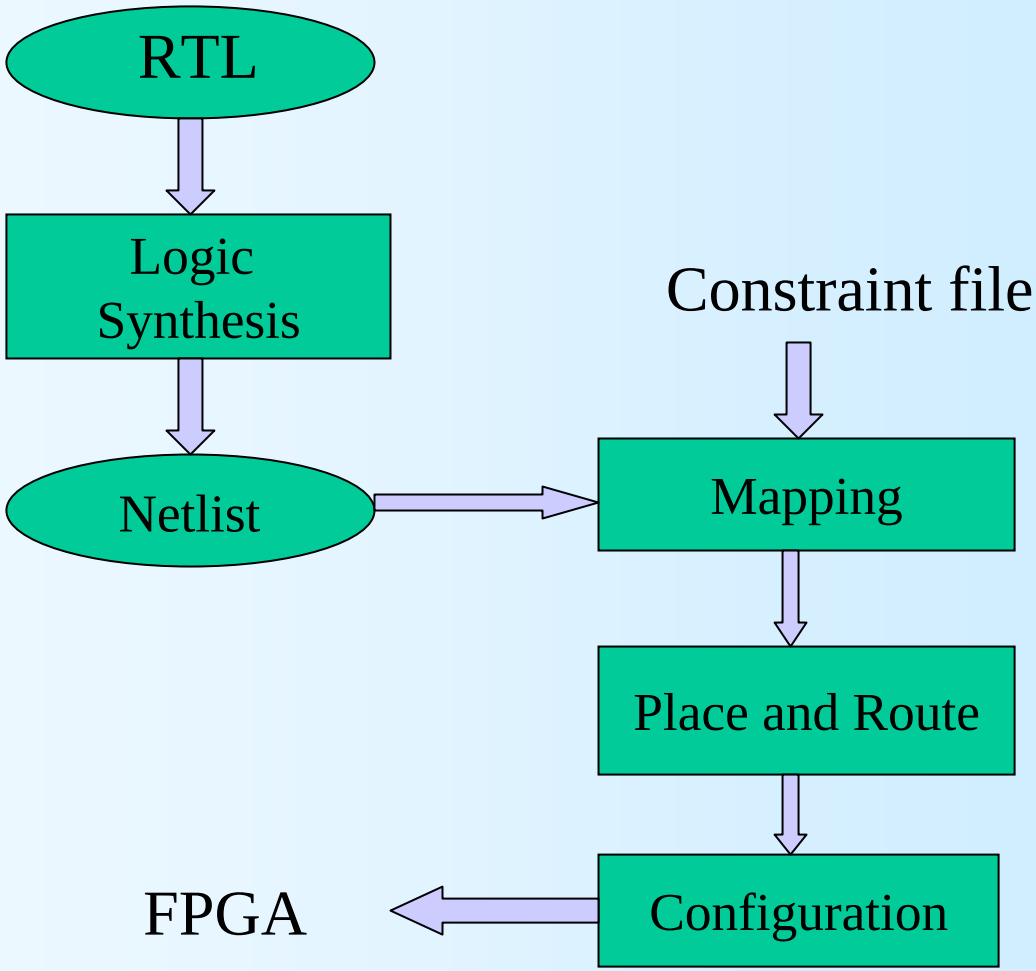
- Effect of new technologies
  - Good for DSP computing
    - Embedded multipliers and BRAMs
  - A new player in embedded computing
  - A good solution for network applications
- Are FPGA internals helpful for a designer?

Questions?

Plan

- FPGA architecture
  - Basics of FPGA
- FPGA technologies
  - Architectures of different commercial FPGAs
- FPGA tools
  - FPGA implementation flow and software involved
- HDL coding for FPGA
  - Some coding examples and techniques

FPGA implementation flow



HDL Synthesis

- Input: HDL – VHDL or Verilog
- Output: Netlist
- Process
  - Analysis of the HDL
  - Behavior synthesis steps include scheduling and binding
    - Datapath and FSM are implemented
  - Logic synthesis is logic minimization
  - Output is in terms of basic gates and flip-flops
  - Also estimates area and delay

HDL Syhthesis

- EDA Tools
  - Synplify
  - Xilinx – XST
  - Mentor – FPGA express
  - Synopsys – DC compiler

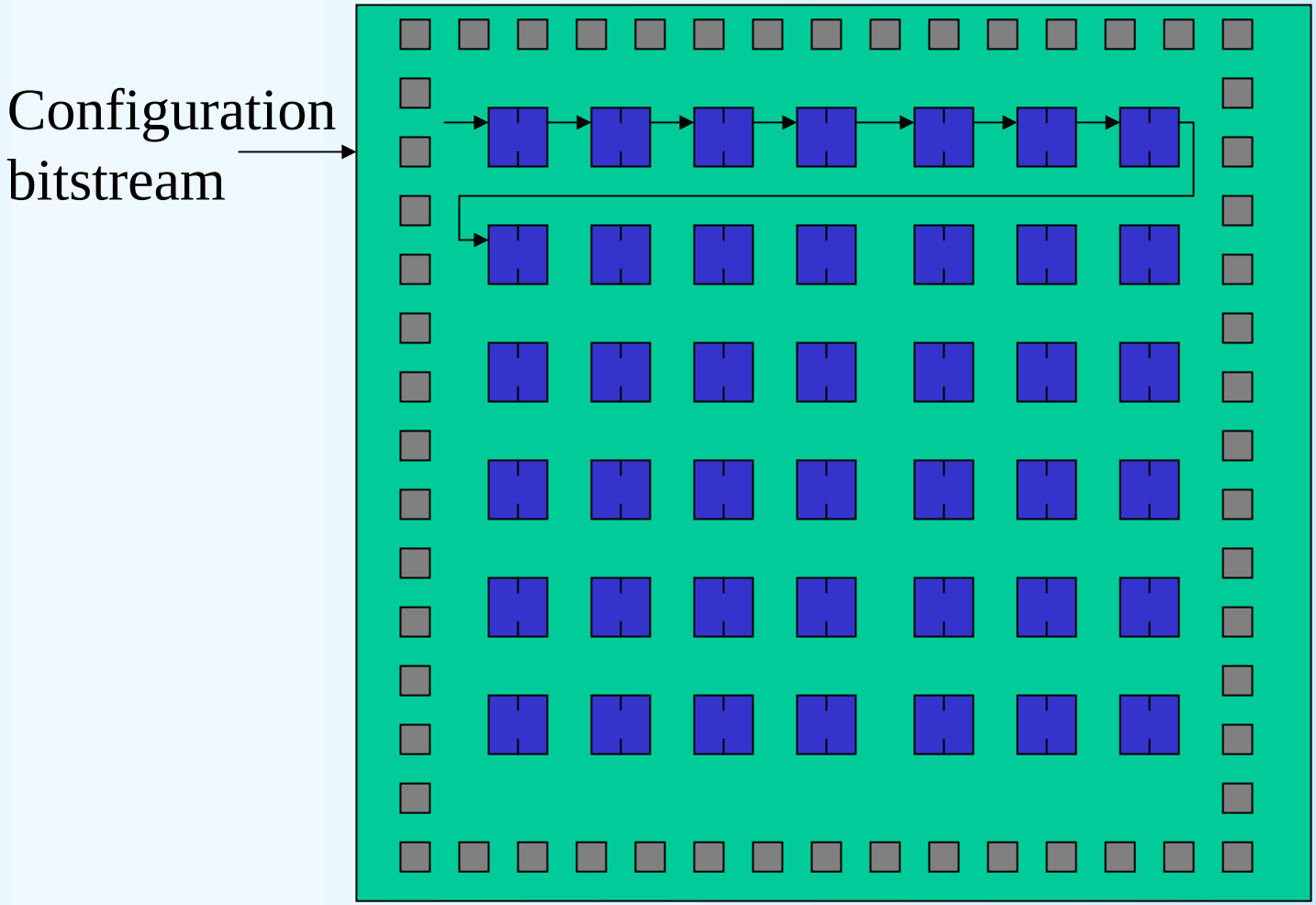
Mapping

- Input: Netlist and ucf
- Output: FPGA specific logic and gates
- Process (
  - For LUT based FPGA
    - For k input LUT, find the sub-graph with k input and one output
- Tools: Vendor specific

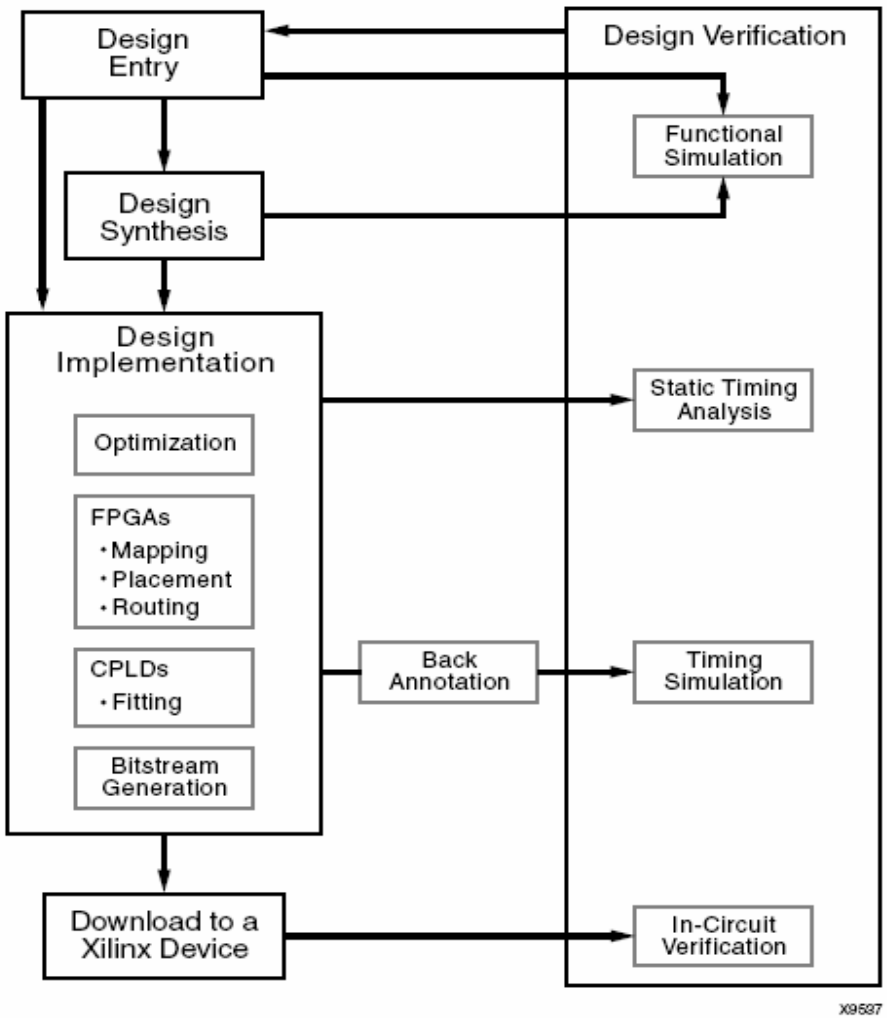
Place and Route

- Place
  - Place the LUTs physically close which are connected most
    - Reduce the overall net length
- Route
  - Use of routing resources to minimize the delay
    - Router have the delay model of interconnects
- Both place and route are NP complete problem
  - Heuristics are used
  - Mostly the process of placement and routing is iterative in nature
- Configuration file generation
  - Based on place and route data configuration file is generated

FPGA configuration

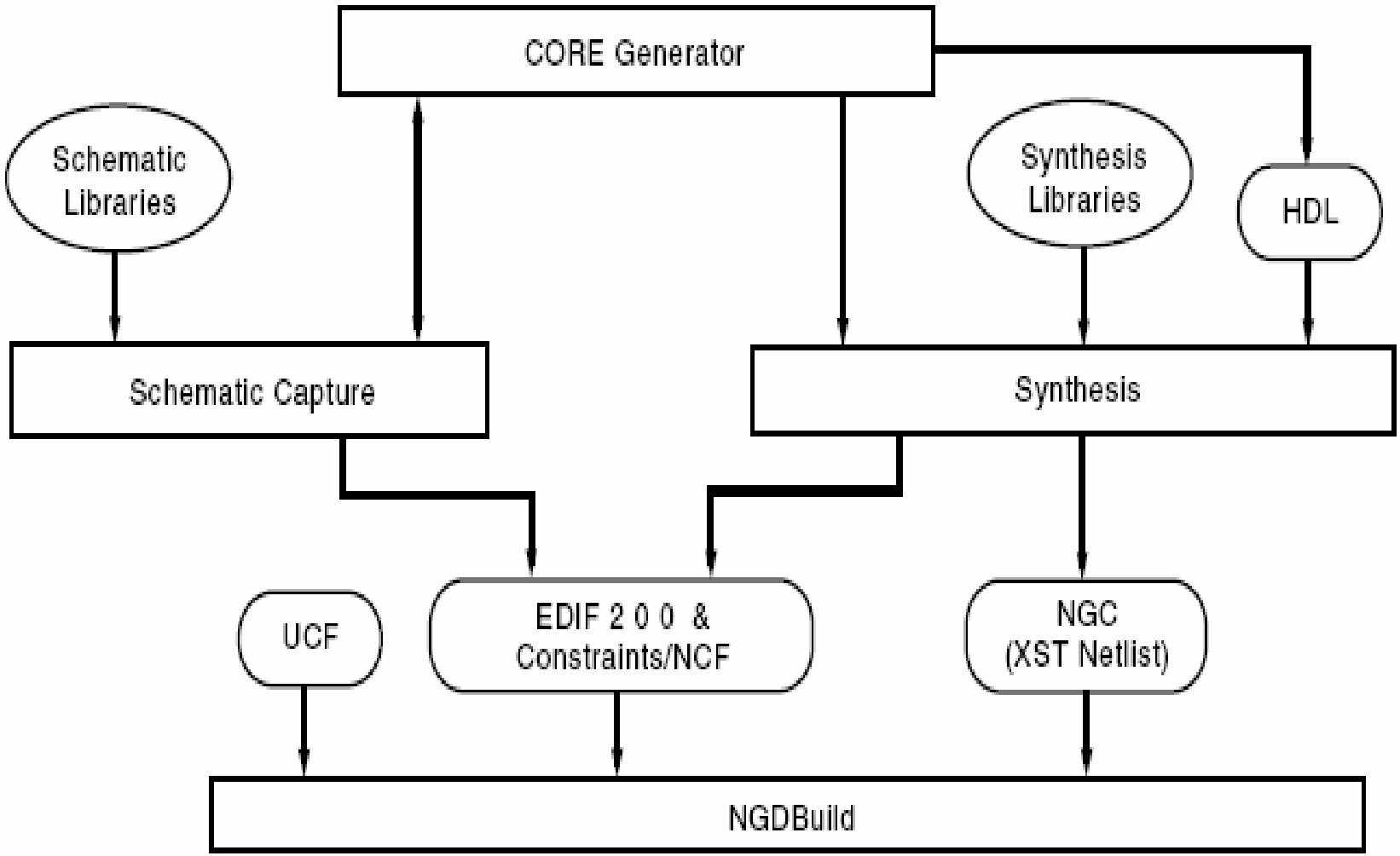


Xilinx tools flow



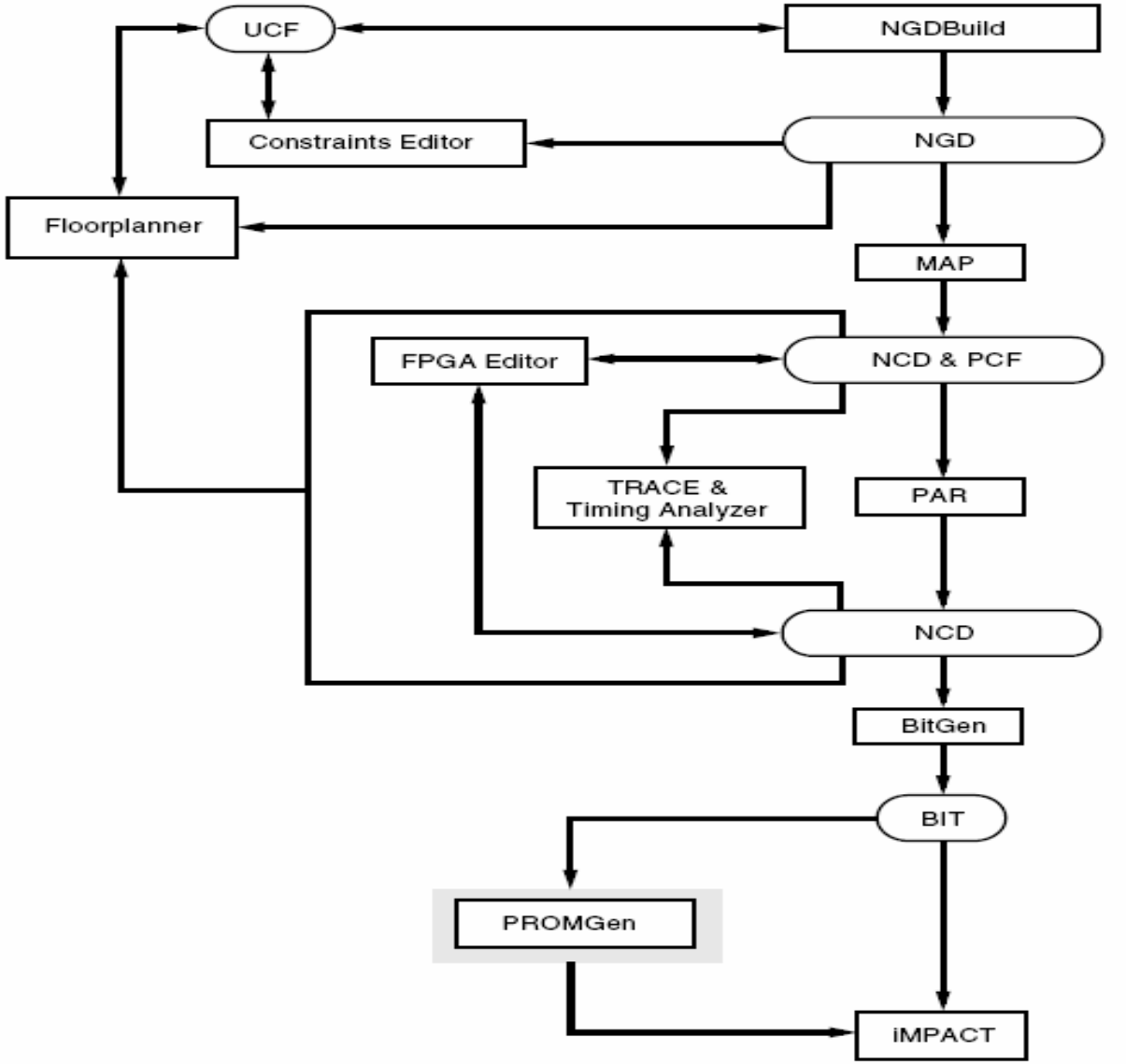
Source: dev manual, Xilinx.com

Design entry and synthesis



Source: dev manual, Xilinx.com

Design implementation process



Source: dev manual, Xilinx.com

Design entry and synthesis

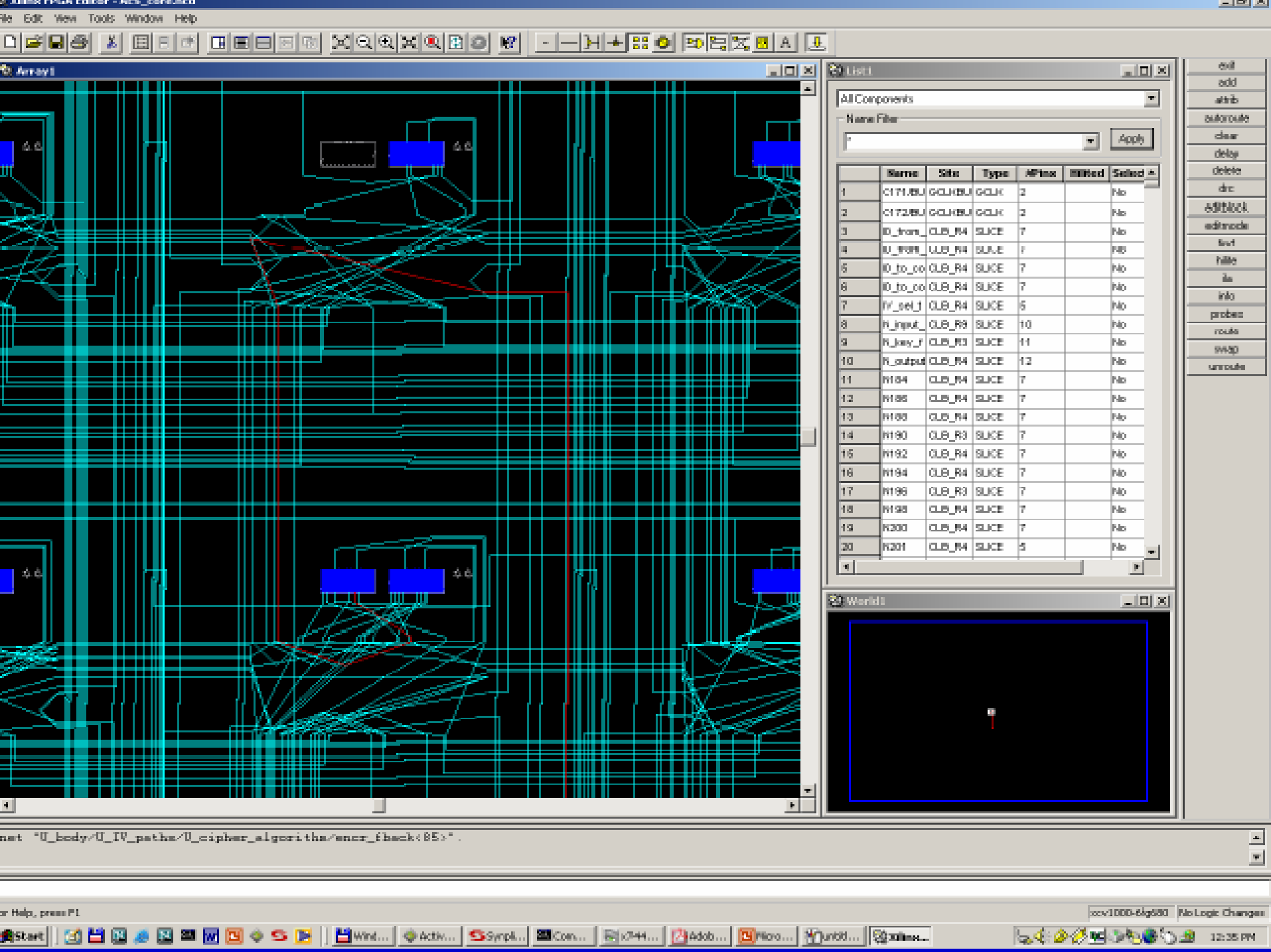
- Input
  - Schematic
    - Basic cells
    - Core generator
  - HDL
- Synthesis process
  - Can have various different module
    - Each module is synthesized as different native generic object (ngo) file
    - All ngo files are combined to form native generic database (ngd) file
  - Constraints can be given as input to ngdbuild process

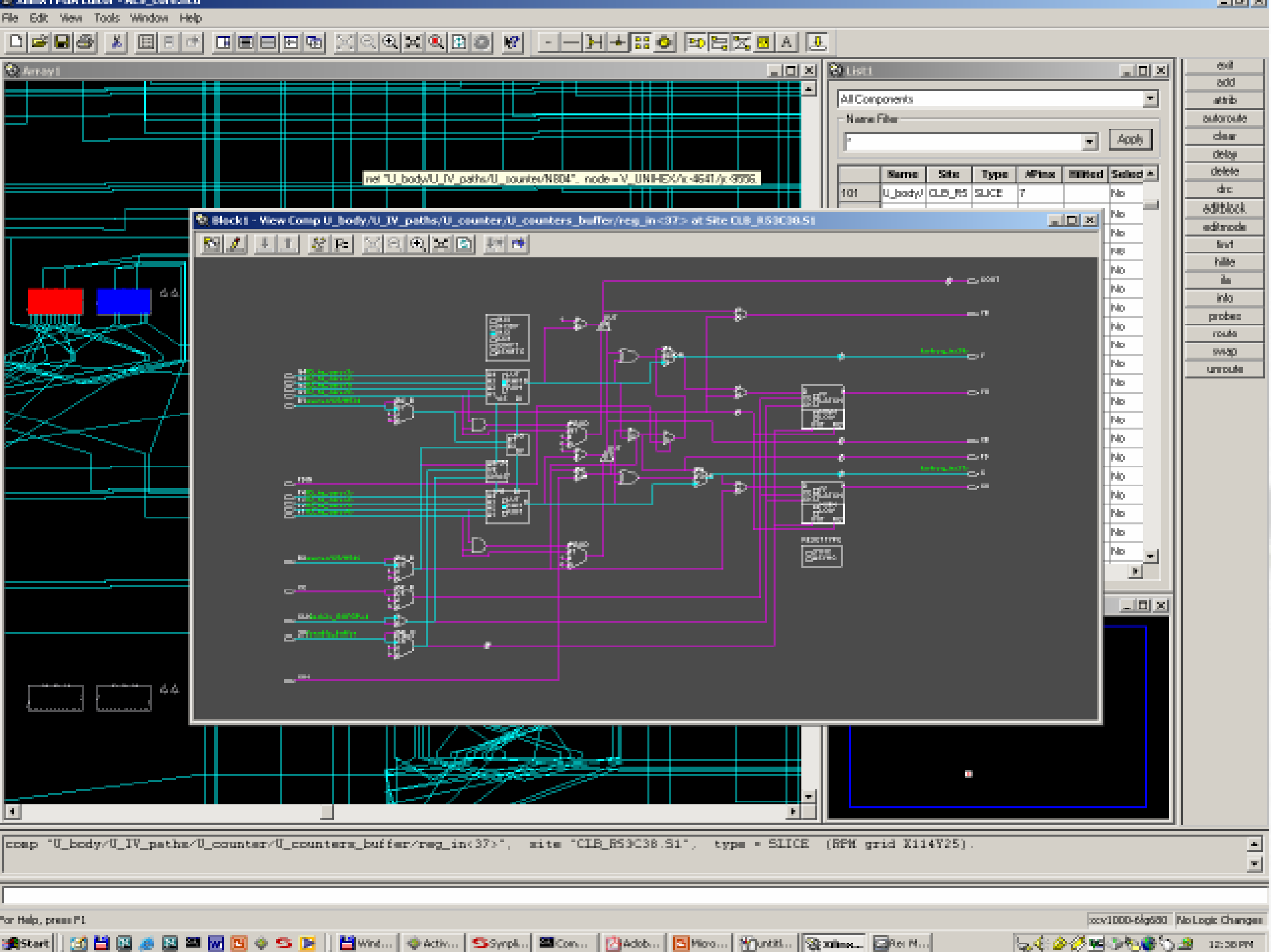
Floorplanner

- Supports hand-placement of FPGA components
- Creates FNF or UCF file
- Some components like DLLs need to be placed manually

FPGA Editor

- Very powerful surgical tool
- Can change any configuration detail of FPGA
  - Placement of components
  - Configuration of CLB Slices
  - Routing of particular nets
  - Logic inside the LUTs


File Edit View Tools Window Help
Array1
List1
All Components
Name Filter
Apply
Name
Site
Type
#Pins
Routed
Select
exit
add
attrib
autoroute
clear
delay
delete
drc
Editblock
editnode
find
hilite
ilo
info
probes
route
swap
unroute
World1
Start

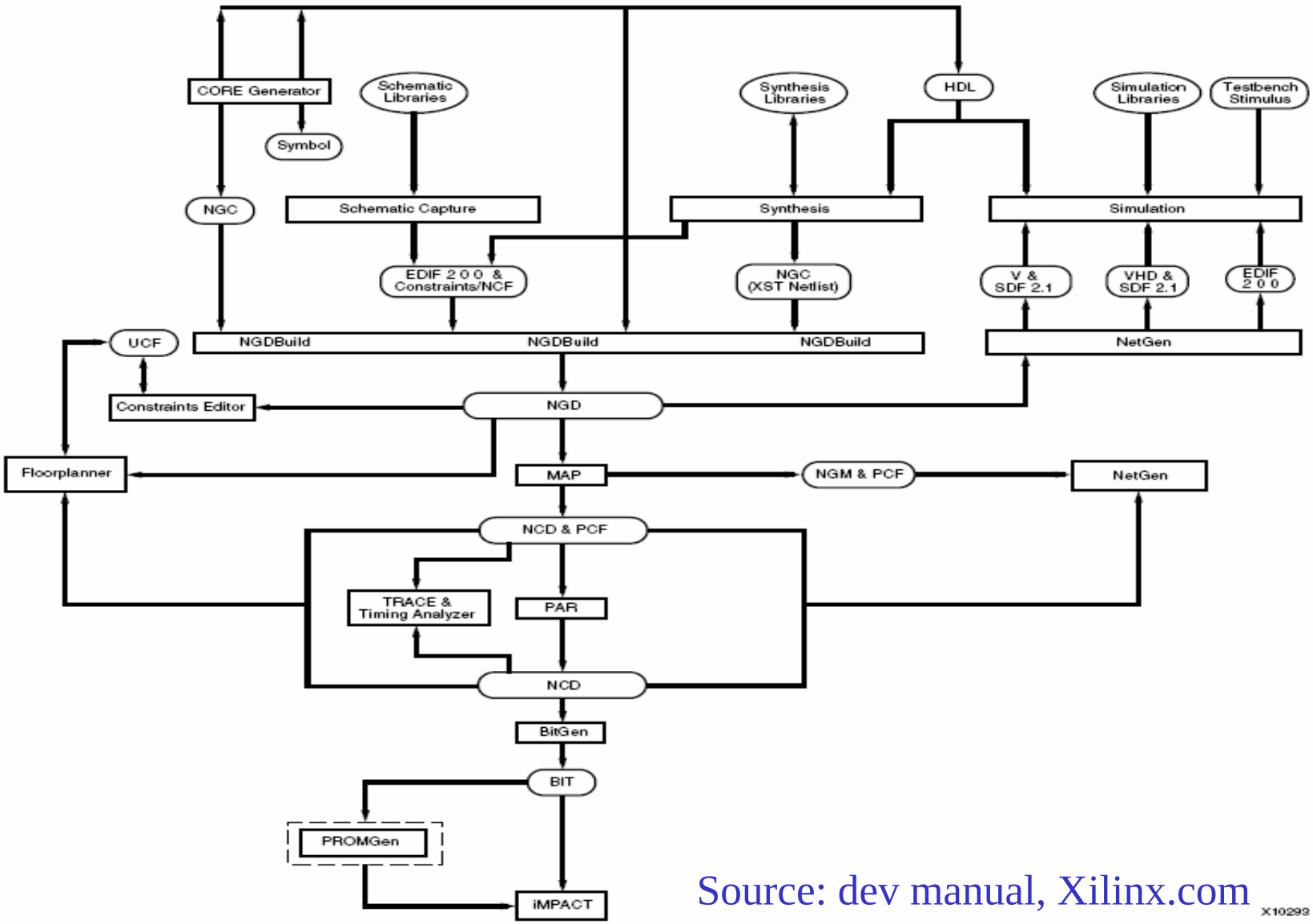


File Edit View Tools Window Help
List1
All Components
Name Filter
Apply
Name Site Type #Pins Hilited Select
101 U_body/ CLB_R5 SLICE 7 No
Block1 - View Comp U_body/U_IV_paths/U_counter/U_counters_buffer/reg_in<37> at Site CLB_R53C38.S1
exit
add
attrib
autoroute
clear
delay
delete
drc
editblock
editmode
find
hilite
ila
info
probes
route
swap
unroute
comp "U_body/U_IV_paths/U_counter/U_counters_buffer/reg_in<37>", site "CLB_R53C38.S1", type = SLICE (RPM grid X114Y25).
For Help, press F1
No Logic Changes
Start
12:38 PM

Timing Analyzer

- Performs static analysis of the circuit performance
- Reports critical paths with all sources of delays
- Determines maximum clock frequency

Xilinx tool flow revisited



Source: dev manual, Xilinx.com

Questions?

Plan

- FPGA architecture
  - Basics of FPGA
- FPGA technologies
  - Architectures of different commercial FPGAs
- FPGA tools
  - FPGA implementation flow and software involved
- HDL coding for FPGA
  - Some coding examples and techniques

Writing HDL code for FPGA

- While writing HDL code, one should be know
  - Resources available in FPGA
  - Mapping of code to resource
- If multiplication is performed
  - Embedded multipliers should be used
    - Various reports during synthesis and implementation convey the resource usage information
- For array variables
  - Block ram should be used

Writing HDL code for FPGA

- If a synthesis tool will infer a BRAM or Multiplier depends on
  - Internals of synthesis tool
  - Quality of HDL code
- Best practice for good results
  - Read the documentation of synthesis tool
    - They will give example; how to write code
  - Read the synthesis report carefully

XST: How to write DFF code

Flip-Flop With Positive Edge Clock VHDL Coding Example

```
--
-- Flip-Flop with Positive-Edge Clock
--
library ieee;
use ieee.std_logic_1164.all;

entity registers_1 is
    port(C, D : in std_logic;
         Q    : out std_logic);
end registers_1;

architecture archi of registers_1 is
begin

    process (C)
    begin
        if (C'event and C='1') then
            Q <= D;
        end if;
    end process;

end archi;
```

Source: XST user guide, Xilinx.com

XST: How to write DFF code

- Note
  - Positive edge triggering
- Synthesis report must say
  - Inferred a D type flip-flop

XST: How to write counter code

4-Bit Unsigned Up Counter With Asynchronous Reset VHDL Coding Example

```
--
-- 4-bit unsigned up counter with an asynchronous reset.
--

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity counters_1 is
    port(C, CLR : in std_logic;
         Q : out std_logic_vector(3 downto 0));
end counters_1;

architecture archi of counters_1 is
    signal tmp: std_logic_vector(3 downto 0);
begin
    process (C, CLR)
    begin
        if (CLR='1') then
            tmp <= "0000";
        elsif (C'event and C='1') then
            tmp <= tmp + 1;
        end if;
    end process;

    Q <= tmp;

end archi;
```

Source: XST user guide, Xilinx.com

XST: How to write adder code

Signed 8-Bit Adder VHDL Coding Example

```
--
-- Signed 8-bit Adder
--

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

entity adders_5 is
    port(A,B : in std_logic_vector(7 downto 0);
         SUM : out std_logic_vector(7 downto 0));
end adders_5;

architecture archi of adders_5 is
begin

    SUM <= A + B;

end archi;
```

Source: XST user guide, Xilinx.com

XST: How to write multiplier code

Unsigned 8x4-Bit Multiplier VHDL Coding Example

```
--
-- Unsigned 8x4-bit Multiplier
--

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity multipliers_1 is
    port(A : in std_logic_vector(7 downto 0);
         B : in std_logic_vector(3 downto 0);
         RES : out std_logic_vector(11 downto 0));
end multipliers_1;

architecture beh of multipliers_1 is
begin
    RES <= A * B;
end beh;
```

Source: XST user guide, Xilinx.com

Summary

- Present day FPGAs are quite powerful
- Need to understand their strengths and internal characteristics to fully exploit their potential
- Designer must understand what will be designed
  - Apart from functional correctness, insight in structure is necessary for optimization
  - If the implemented output is not desired
    - Something wrong
    - EDA tools is not provided enough information!
- Good to have understanding of tool flow for advanced manipulations

Questions?

Thank you!