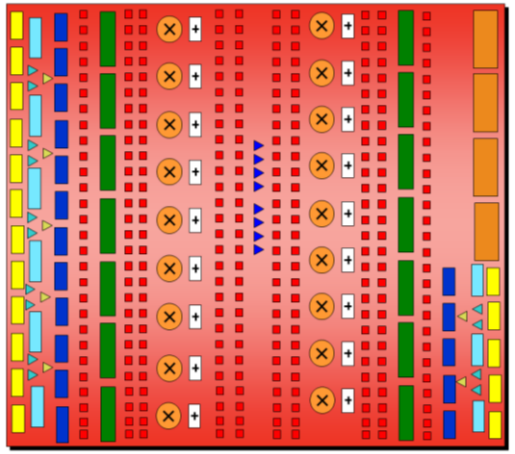
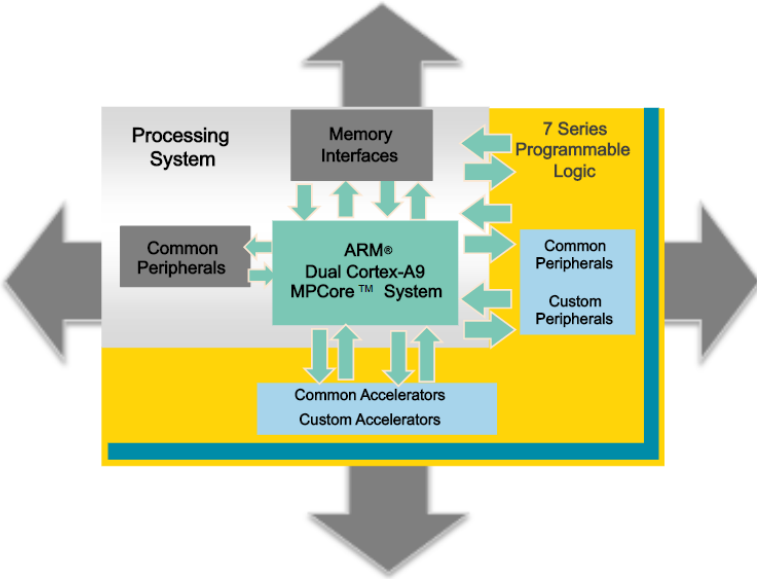
DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

A2A
Algorithms to Architecture

ECE 270: Embedded Logic Design





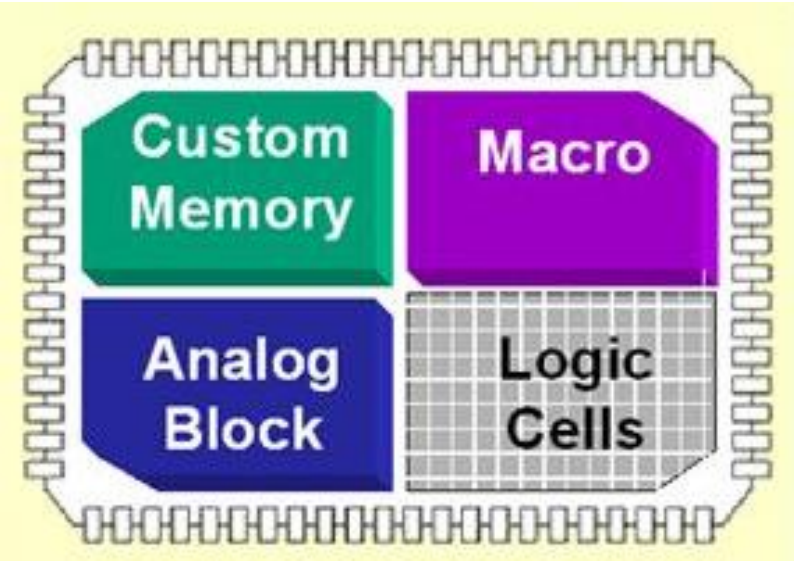
Dr. Sumit J Darak
Algorithms to Architectures Lab
Associate Professor, ECE, IIIT Delhi
http://faculty.iiitd.ac.in/~sumit/

ASIC vs. FPGA vs. Microcontroller*

https://www.youtube.com/watch?v=vxSvQ-IcmHM

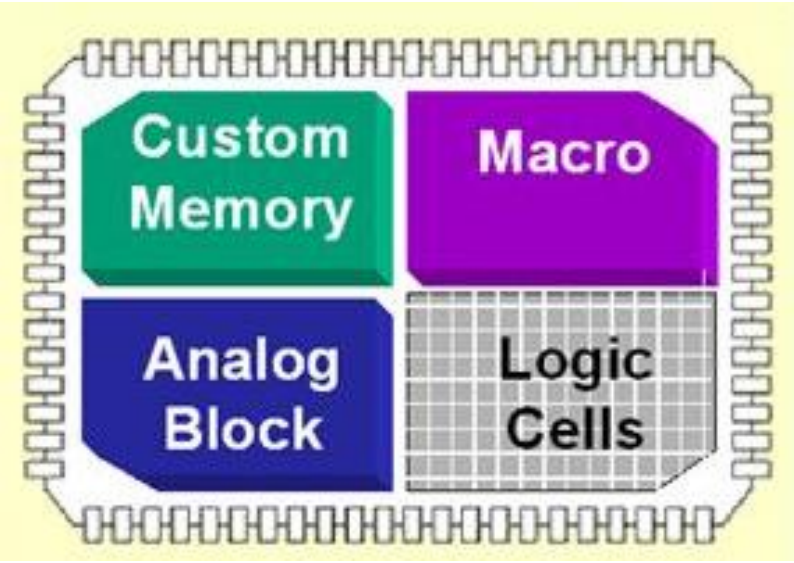
ASIC

- **ASIC**: Application Specific Integration Circuits



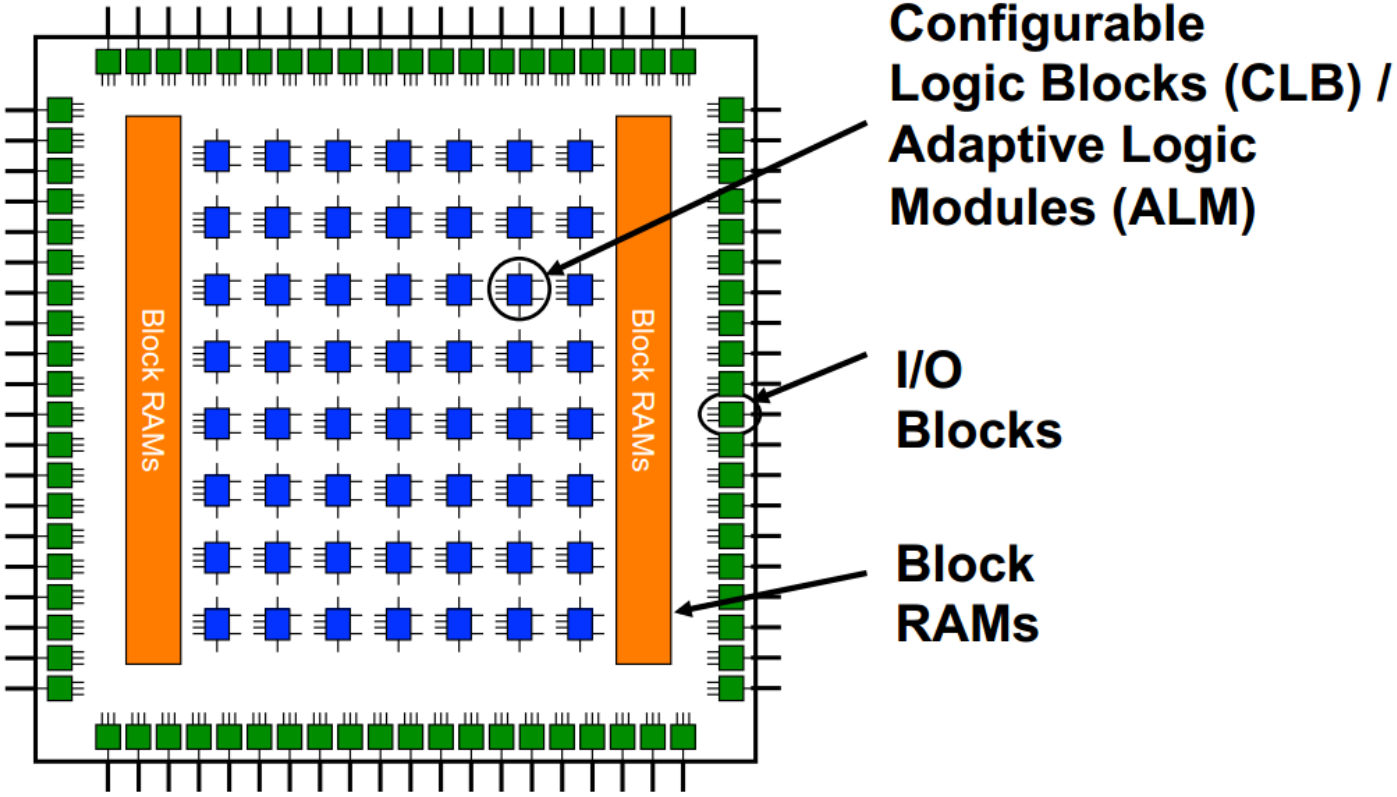
ASIC

- **ASIC**: Application Specific Integration Circuits
- High non-recurring engineering cost (one-time cost to research, design, develop and test a new product) and longest design cycle
- High cost for engineering change orders (Testing is critical! And hence, preferred when design is finalized)
- Lowest price for high volume production
- Fastest clock performance (high performance)
- Unlimited size and Low power
- Design and test tools are expensive
- Expensive IPs

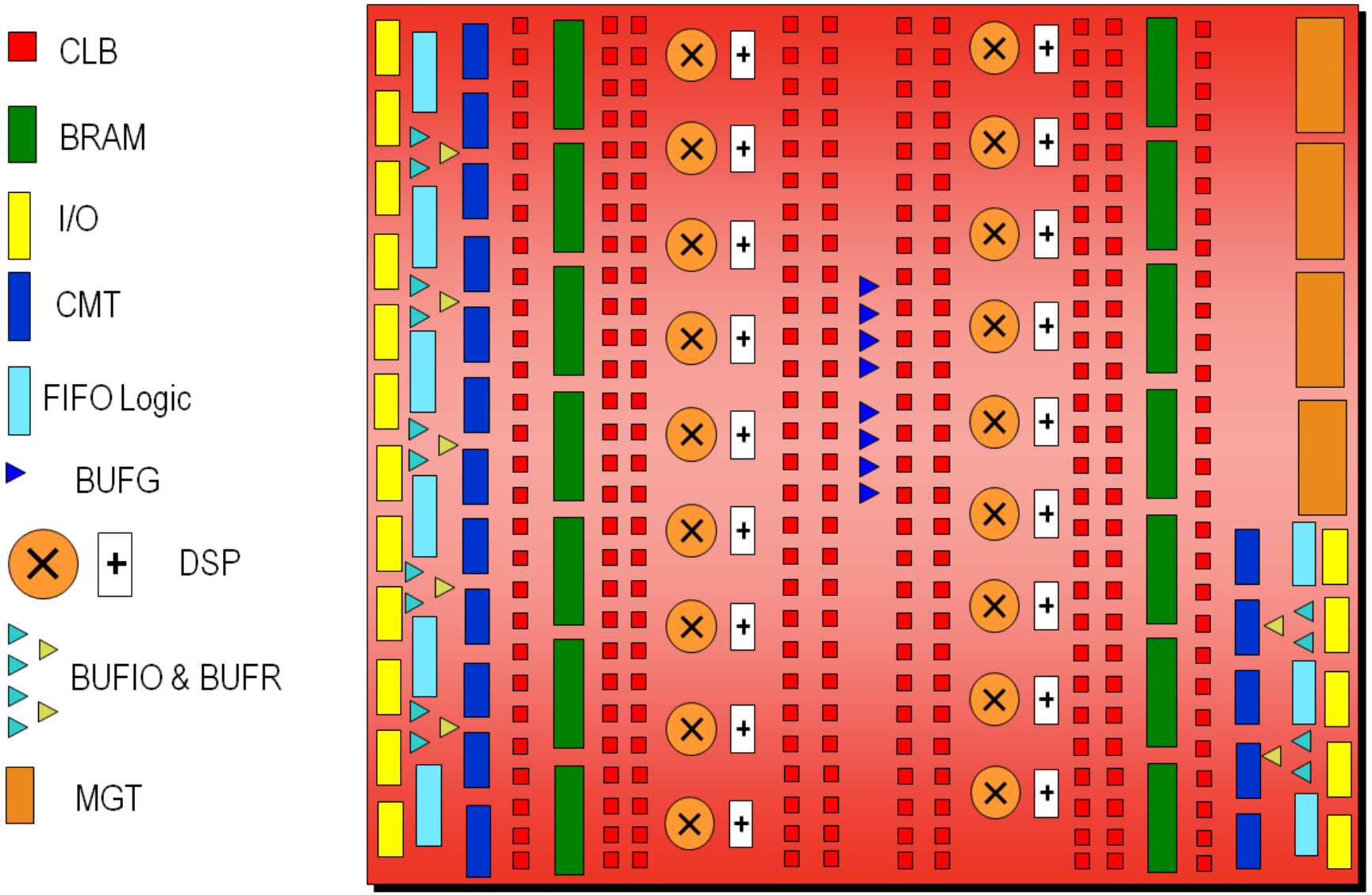


FPGA: Field Programmable Gate Array

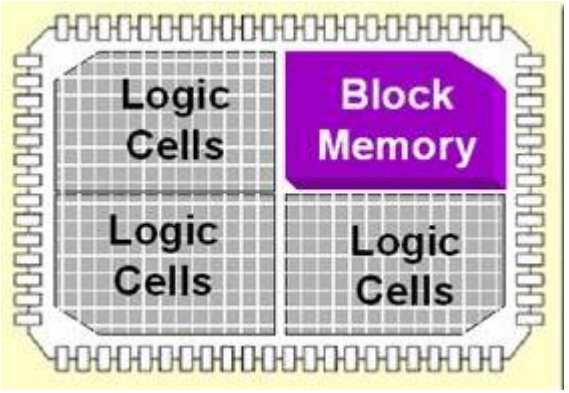
- **Array** of generic logic gates
- **Gates** where logic function can be programmed
- **Programmable** interconnection between gates
- **Field**: System can be reprogrammed in the field (After fabrication)



FPGA Architecture



FPGA



- Lowest cost for low to medium volume
- No NRE cost and Fastest time to market
- Field reconfigurable and partial reconfigurable
- Slower performance than ASIC (still 550 MHz)
- Limited size and steep learning curve
- Digital only*
- Industry often use FPGAs to prototype their chips before creating them (FPGA before ASIC).

FPGA vs ASIC

FPGA	ASIC
Reconfigurable circuitry after manufacturing	Fixed circuitry for product's lifespan
Suitable for digital designs only	Analog/mixed-signal circuitry can be fully implemented
Can be purchased as off-the-shelf products	Can only be designed as custom, private-label devices
Low-performance efficiencies, higher power consumption	Low power consumption, high-performance efficiencies
No non-recurring engineering (NRE) costs	NRE costs are part of the design process
Difficult to attain high-frequency rates	Operate at higher frequency rates
Faster time-to-market, high per unit costs	Long time-to-market, lower per unit costs
Are typically larger than ASICs	Can be much smaller than FPGA devices
Prototyping and validating with FPGAs is easier	Prototypes must be accurately validated to avoid design iterations
Lower barrier to entry for competitors	Higher barrier to entry for competitors

- Demand for specialized systems and short device life -> FPGAs

Microcontrollers

- Similar to simple computer placed in a single chip with all necessary components like memory, timers etc. embedded inside and performs a specific task.
- **Example:** Arduino, Pic
- Sequential execution, easy to use, control over **software**
- Consumes less power than FPGAs and mostly suitable for edge operations.
- Supports fixed as well as floating point operations
- **Microprocessors:** Completely different than microcontrollers.

Microprocessors

- ICs that come with a computer or CPU inside and are equipped with processing power. Examples: Pentium 3, 4, i5 etc.
- No peripherals such as RAM, ROM on the chip.
- Microprocessors form the heart of a computing system (general complex high-speed tasks) while microcontrollers drive embedded systems (specific tasks).
- Bulky due to the external peripherals

- **Expensive than micro-controllers**

Microprocessor vs FPGA vs ASIC

	Microprocessor	FPGA	ASIC
Example	ARM Cortex-A9	Virtex Ultrascale 440	Bitfury 16nm
Flexibility during development	Medium	High	Very high
Flexibility after development[1]	High	High	Low
Parallelism	Low	High	High
Performance[2]	Low	Medium	High
Power consumption	High	Medium	Low
Development cost	Low	Medium	High
Production setup cost[3]	None	None	High
Unit cost[4]	Medium	High	Low
Time-to-market	Low	Medium	High

[1]E.g. to fix bugs, add new functionality when already in production

[2]For a sufficiently parallel application

[3]Cost of producing the first chip

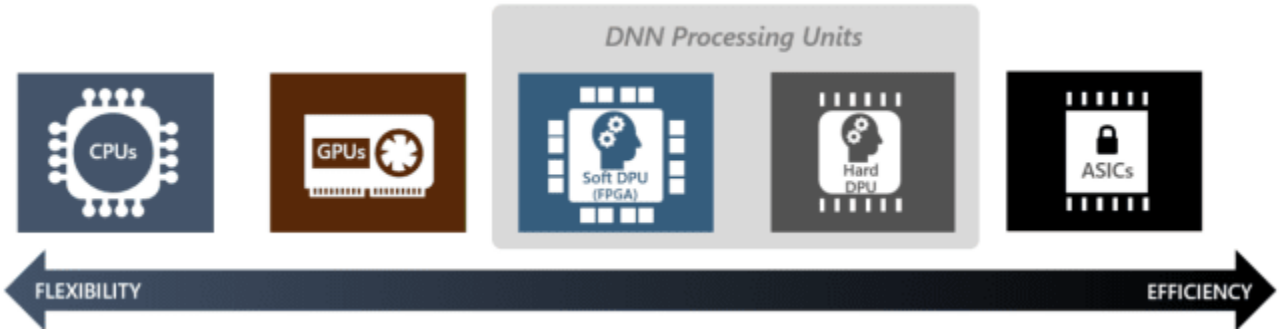
Microprocessor vs FPGA vs ASIC vs GPU

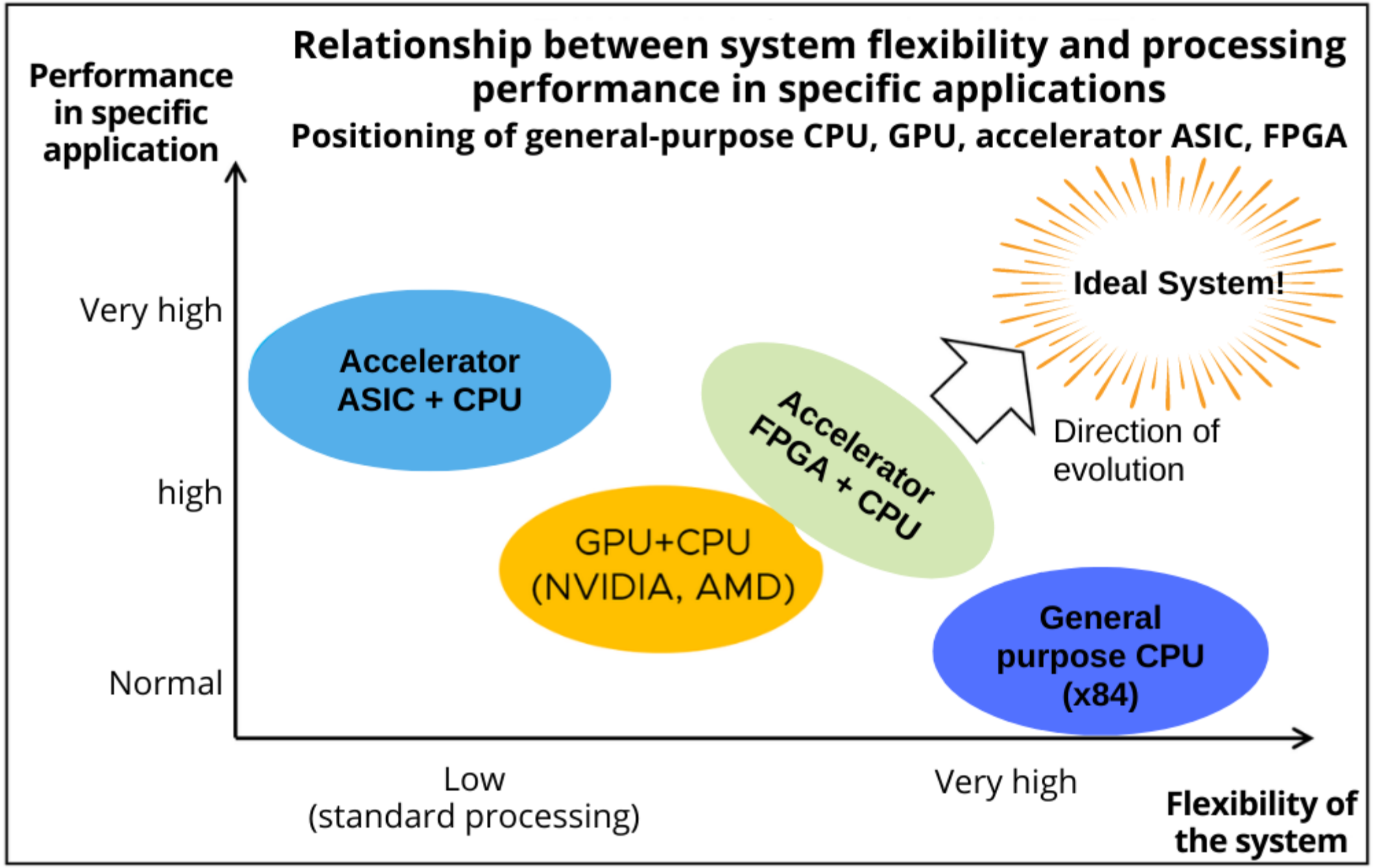
	Microprocessor	FPGA	ASIC	GPU
Example	ARM Cortex-A9	Virtex Ultrascale 440	Bitfury 16nm	Nvidia Titan X
Flexibility during development	Medium	High	Very high	Low
Flexibility after development[1]	High	High	Low	High
Parallelism	Low	High	High	Medium
Performance[2]	Low	Medium	High	Medium
Power consumption	High	Medium	Low	High
Development cost	Low	Medium	High	Low
Production setup cost[3]	None	None	High	None
Unit cost[4]	Medium	High	Low	High
Time-to-market	Low	Medium	High	Medium

[1]E.g. to fix bugs, add new functionality when already in production

[2]For a sufficiently parallel application

[3]Cost of producing the first chip




Relationship between system flexibility and processing performance in specific applications
Positioning of general-purpose CPU, GPU, accelerator ASIC, FPGA
Performance in specific application
Very high
high
Normal
Accelerator ASIC + CPU
Accelerator FPGA + CPU
GPU+CPU (NVIDIA, AMD)
General purpose CPU (x84)
Ideal System!
Direction of evolution
Low (standard processing)
Very high
Flexibility of the system

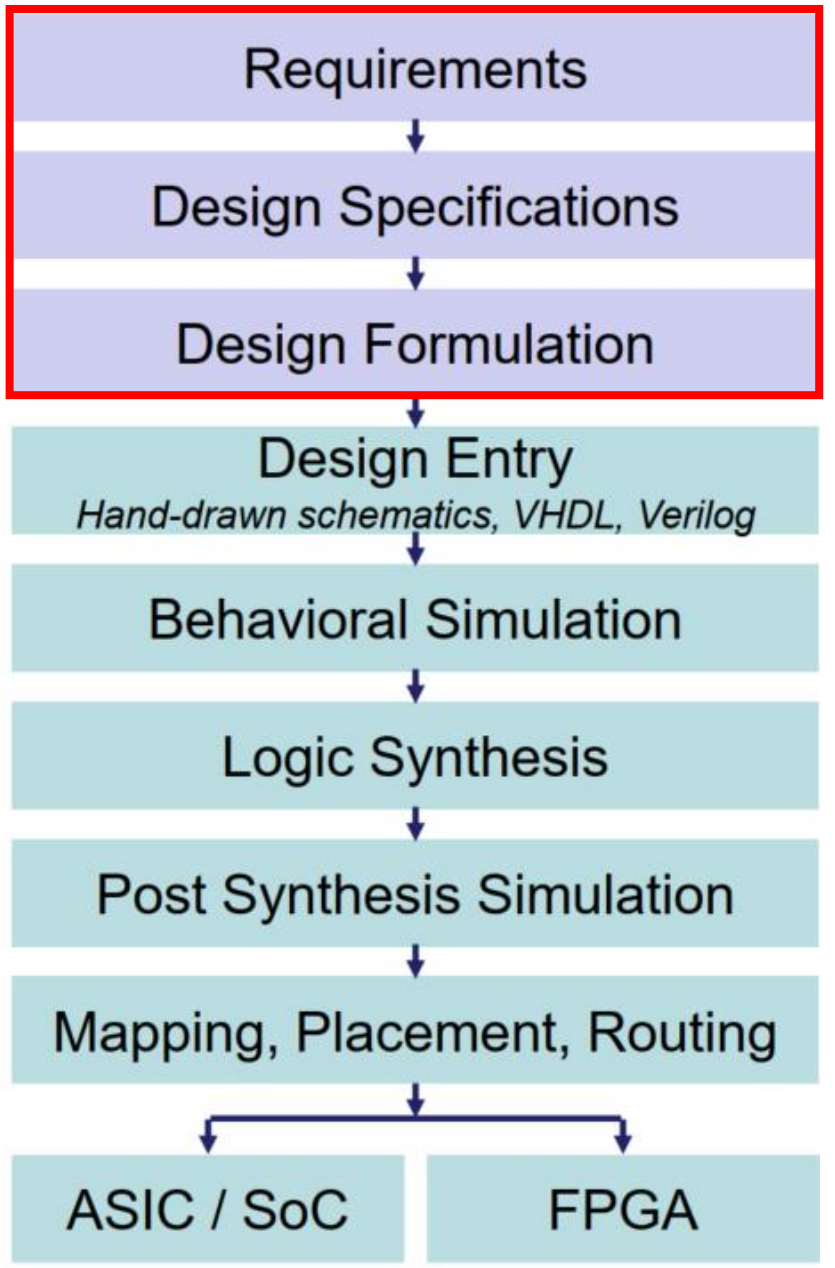
Summary

- **FPGA/ASIC:** Parallel execution, HDL (Verilog/VHDL), control over **hardware**
- You can make microcontroller inside FPGA/ASIC but not the other way round
- Time vs space limited
- FPGAs can perform any task while microcontrollers are limited by instruction sets while ASICs are application specific. FPGA are field-reconfigurable.
- Power consumption is high in FPGA
- FPGAs/ASICs can not be avoided in applications with stringent computational and memory requirements or applications with high level of determinism
- New world SoC: ARM + FPGA + GPU

Generic FPGA Design Flow

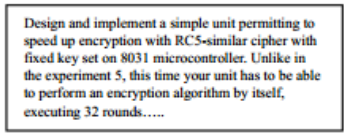
FPGA Design Flow

- All the designs start with design requirements and design specifications
- Next step is to formulate the design conceptually either at block diagram level or at an algorithmic level



FPGA Design Flow

- Design Entry:
- Olden days: Hand-drawn schematic
- Now, computer-aided design (CAD) tools: Mostly using HDLs



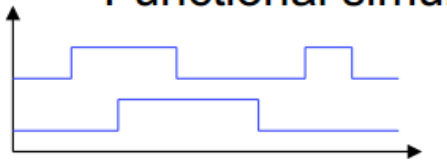
Specification / Pseudocode

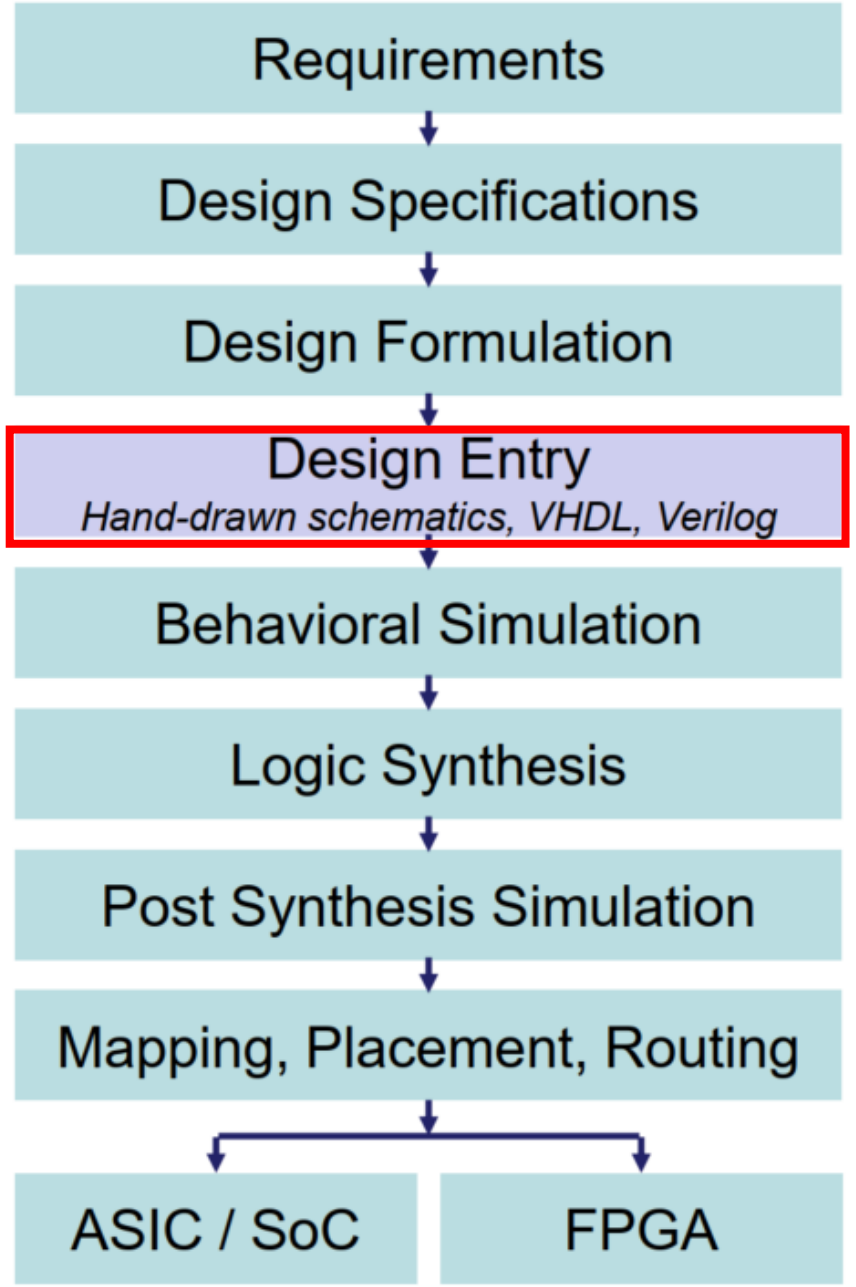
On-paper hardware design
(Block diagram & ASM chart)

HDL description (Your Source Files)

```
Library IEEE;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity RC5_core is
        port(
                clock, reset, encr_decr: in std_logic;
                data_input: in std_logic_vector(31 downto 0);
                data_output: out std_logic_vector(31 downto 0);
                out_full: in std_logic;
                key_input: in std_logic_vector(31 downto 0);
                key_read: out std_logic;
        );
end AES_core;
```

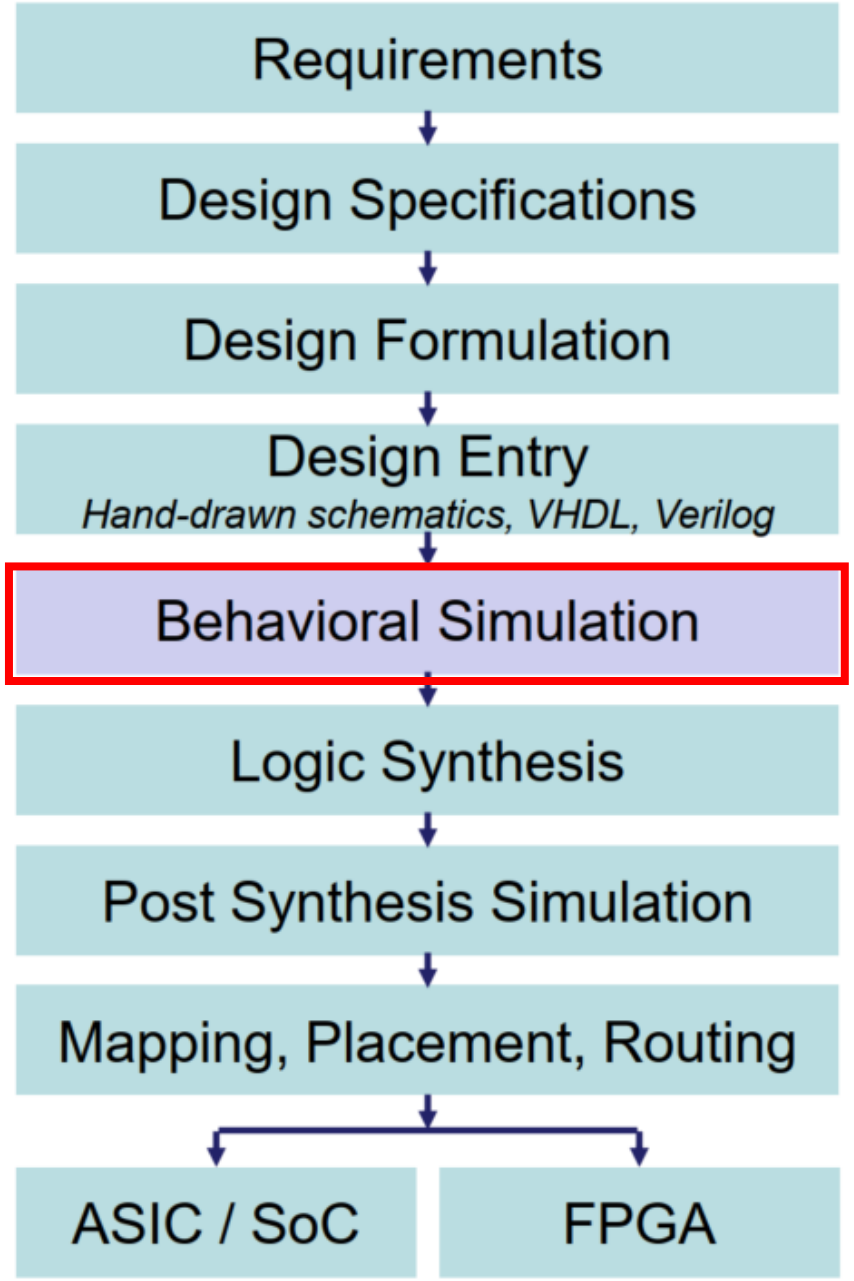
Functional simulation



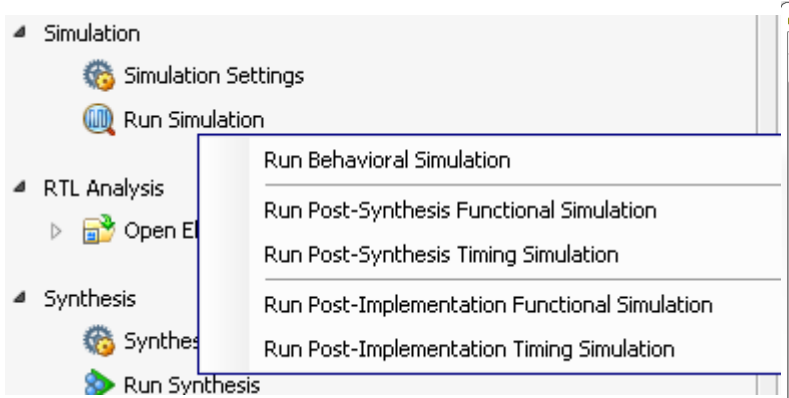


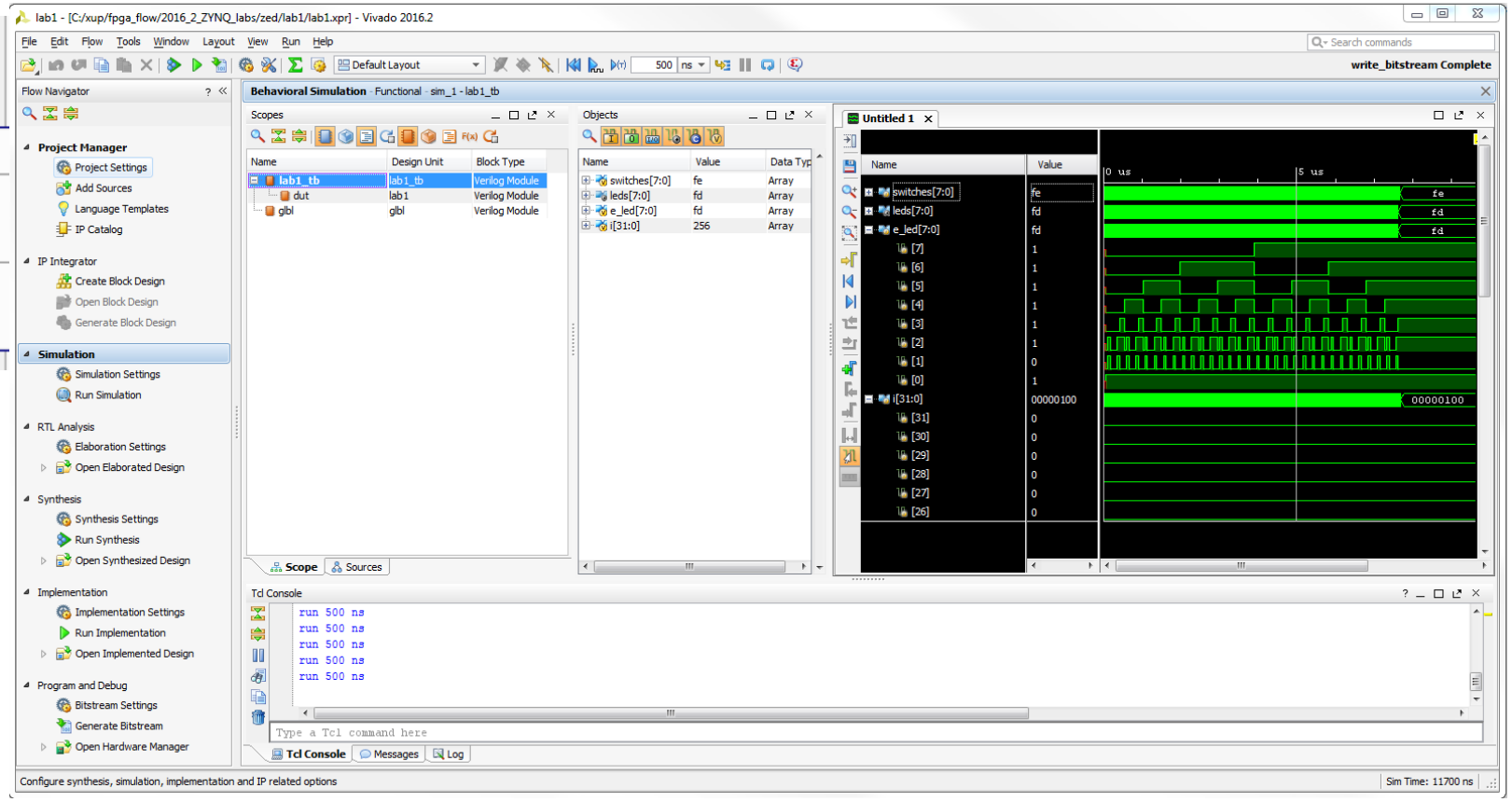
FPGA Design Flow

- Behavioral simulation to ensure that the design is functionally correct
- Can be done using: Test benches, test bench waveforms



Functional (HDL/RTL) Simulation





HDL/RTL Simulation

- Self-checking test bench: Contains output data and self-checking code that can be used to compare the data from later simulation runs
- When running a simulation with this kind of test bench, simulation outputs will be monitored. If a difference is detected between the predicted and the actual outputs, an error is reported.
- Automate the task of verifying simulation results i.e. no need to manually check waveform results

HDL/RTL Simulation

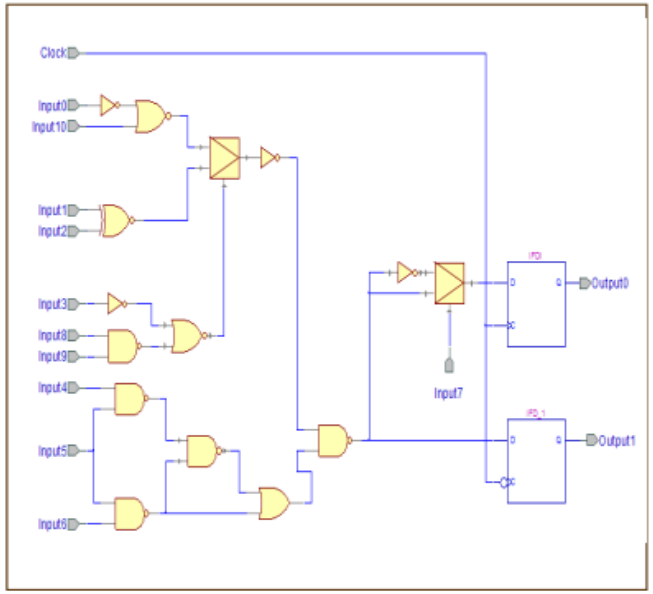
```
module testbench2();
  reg  a, b, c;
  wire y;

  // instantiate device under test
  sillyfunction dut(.a(a), .b(b), .c(c), .y(y));

  // apply inputs one at a time
  initial begin
     a = 0; b = 0; c = 0; #10; // apply input, wait
     if (y !== 1) $display("000 failed."); // check
     c = 1; #10;               // apply input, wait
     if (y !== 0) $display("001 failed."); // check
     b = 1; c = 0; #10;        // etc.. etc..
     if (y !== 0) $display("010 failed."); // check
  end
endmodule
```

FPGA Design Flow

VHDL description

```
architecture MLU_DATAFLOW of MLU is

signal A1:STD_LOGIC;
signal B1:STD_LOGIC;
signal Y1:STD_LOGIC;
signal MUX_0, MUX_1, MUX_2, MUX_3: STD_LOGIC;

begin
        A1<=A when (NEG_A='0') else
                not A;
        B1<=B when (NEG_B='0') else
                not B;
        Y<=Y1 when (NEG_Y='0') else
                not Y1;

        MUX_0<=A1 and B1;
        MUX_1<=A1 or B1;
        MUX_2<=A1 xor B1;
        MUX_3<=A1 xnor B1;

        with (L1 & L0) select
                Y1<=MUX_0 when "00",
                        MUX_1 when "01",
                        MUX_2 when "10",
                        MUX_3 when others;

end MLU_DATAFLOW;
```

Circuit netlist



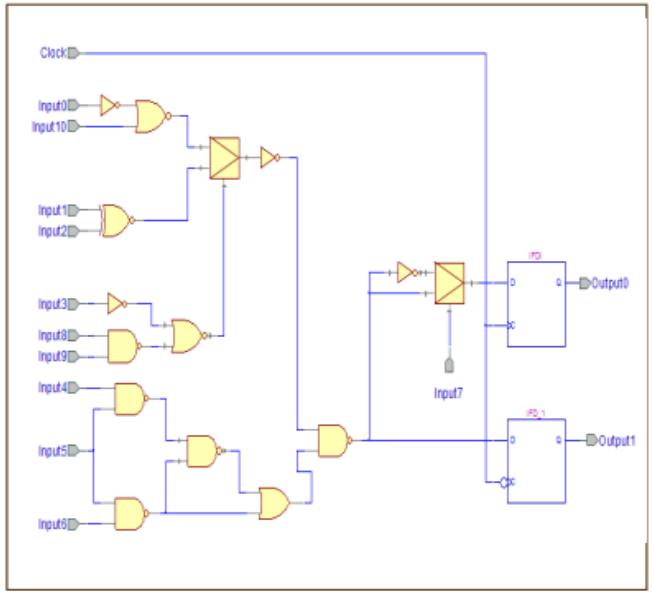
- Requirements
- Design Specifications
- Design Formulation
- Design Entry
  Hand-drawn schematics, VHDL, Verilog
- Behavioral Simulation
- **Logic Synthesis**
- Post Synthesis Simulation
- Mapping, Placement, Routing
- ASIC / SoC
- FPGA

FPGA Design Flow

VHDL description

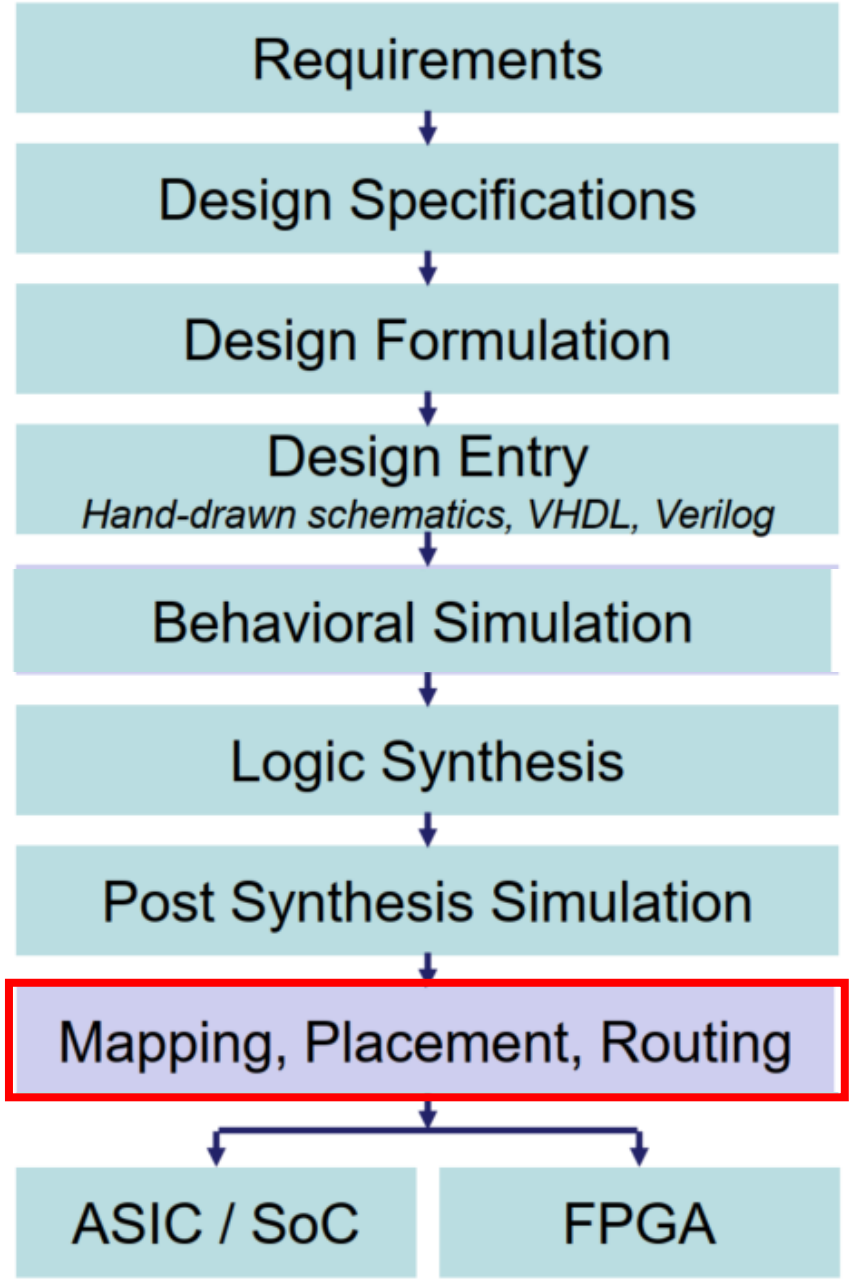
```
architecture MLU_DATAFLOW of MLU is

signal A1:STD_LOGIC;
signal B1:STD_LOGIC;
signal Y1:STD_LOGIC;
signal MUX_0, MUX_1, MUX_2, MUX_3: STD_LOGIC;

begin
        A1<=A when (NEG_A='0') else
                not A;
        B1<=B when (NEG_B='0') else
                not B;
        Y<=Y1 when (NEG_Y='0') else
                not Y1;

        MUX_0<=A1 and B1;
        MUX_1<=A1 or B1;
        MUX_2<=A1 xor B1;
        MUX_3<=A1 xnor B1;

        with (L1 & L0) select
                Y1<=MUX_0 when "00",
                        MUX_1 when "01",
                        MUX_2 when "10",
                        MUX_3 when others;

end MLU_DATAFLOW;
```

Circuit netlist



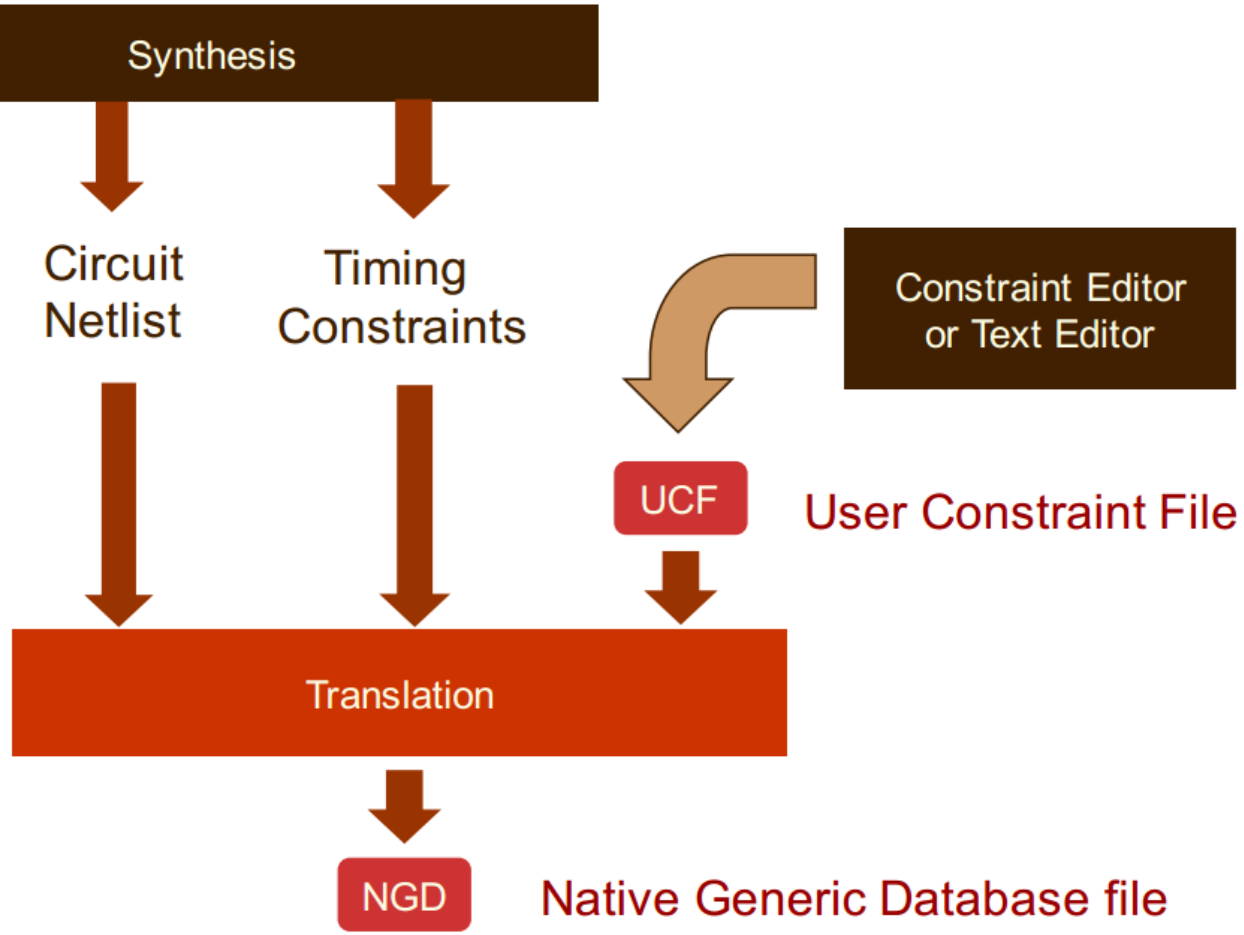
- Requirements
- Design Specifications
- Design Formulation
- Design Entry
  Hand-drawn schematics, VHDL, Verilog
- Behavioral Simulation
- Logic Synthesis
- **Post Synthesis Simulation**
- Mapping, Placement, Routing
- ASIC / SoC
- FPGA

FPGA Design Flow



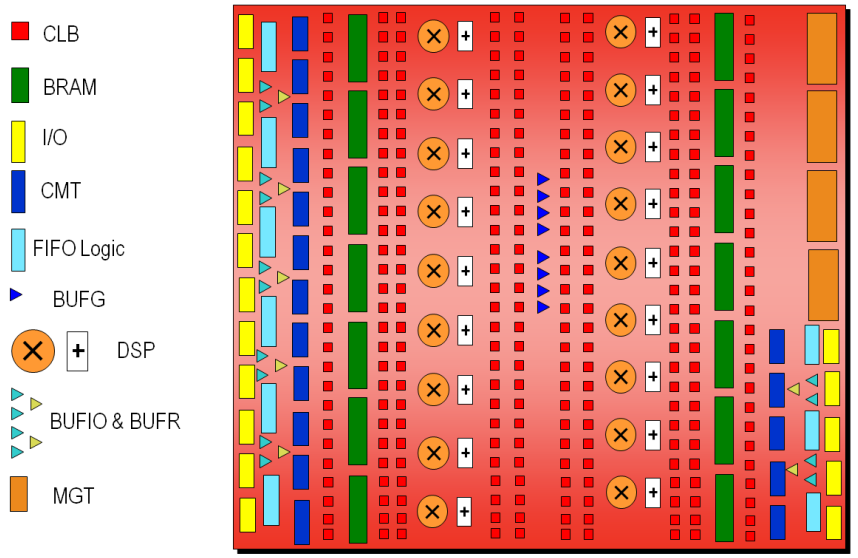
Translation

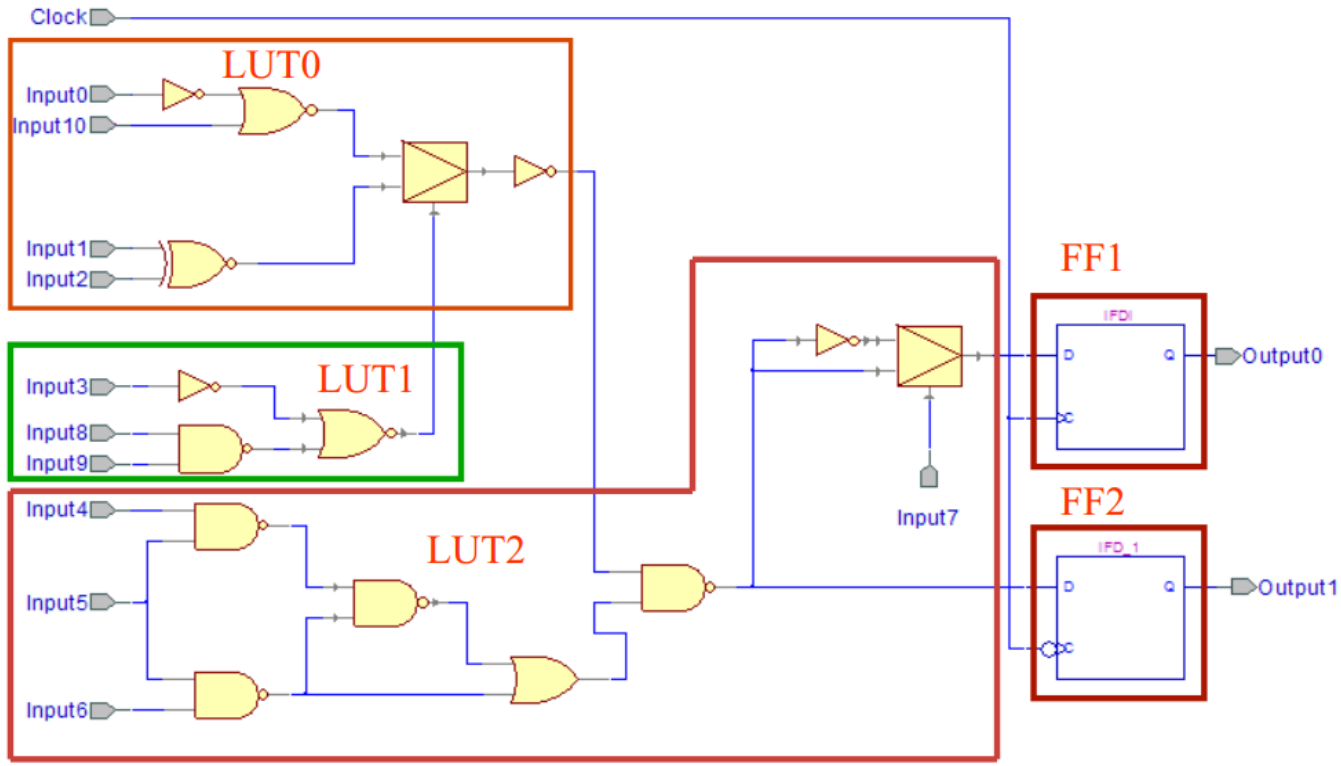
- Translate process merges all of the input netlists and design constraints and outputs a Xilinx native generic database (NGD) file



Mapping

- The Map process maps the logic defined by an NGD file into FPGA elements, such as CLBs and IOBs.
- The output design is a native circuit description (NCD) file that physically represents the design mapped to the components in the FPGA.



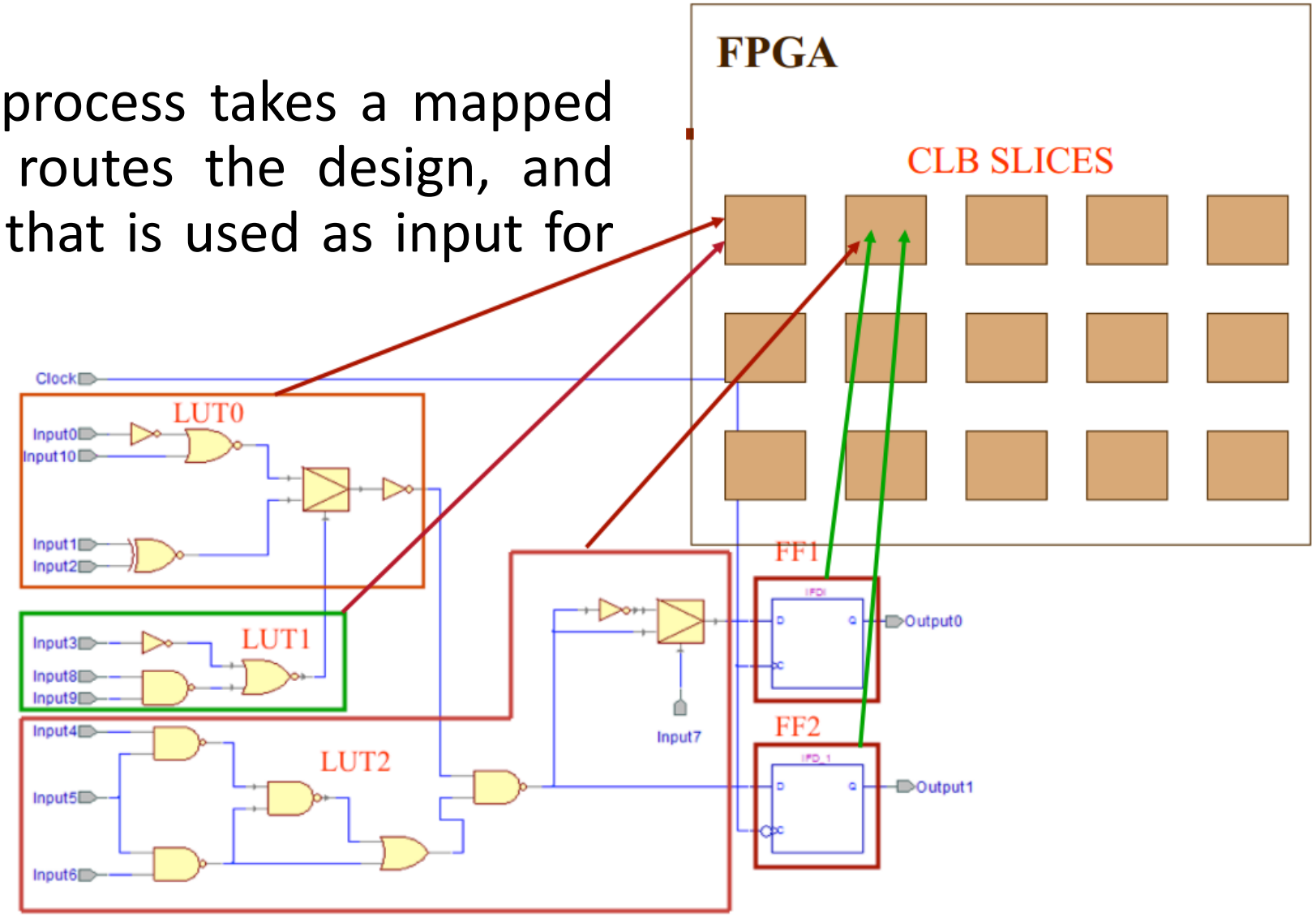


Post-map Static Timing

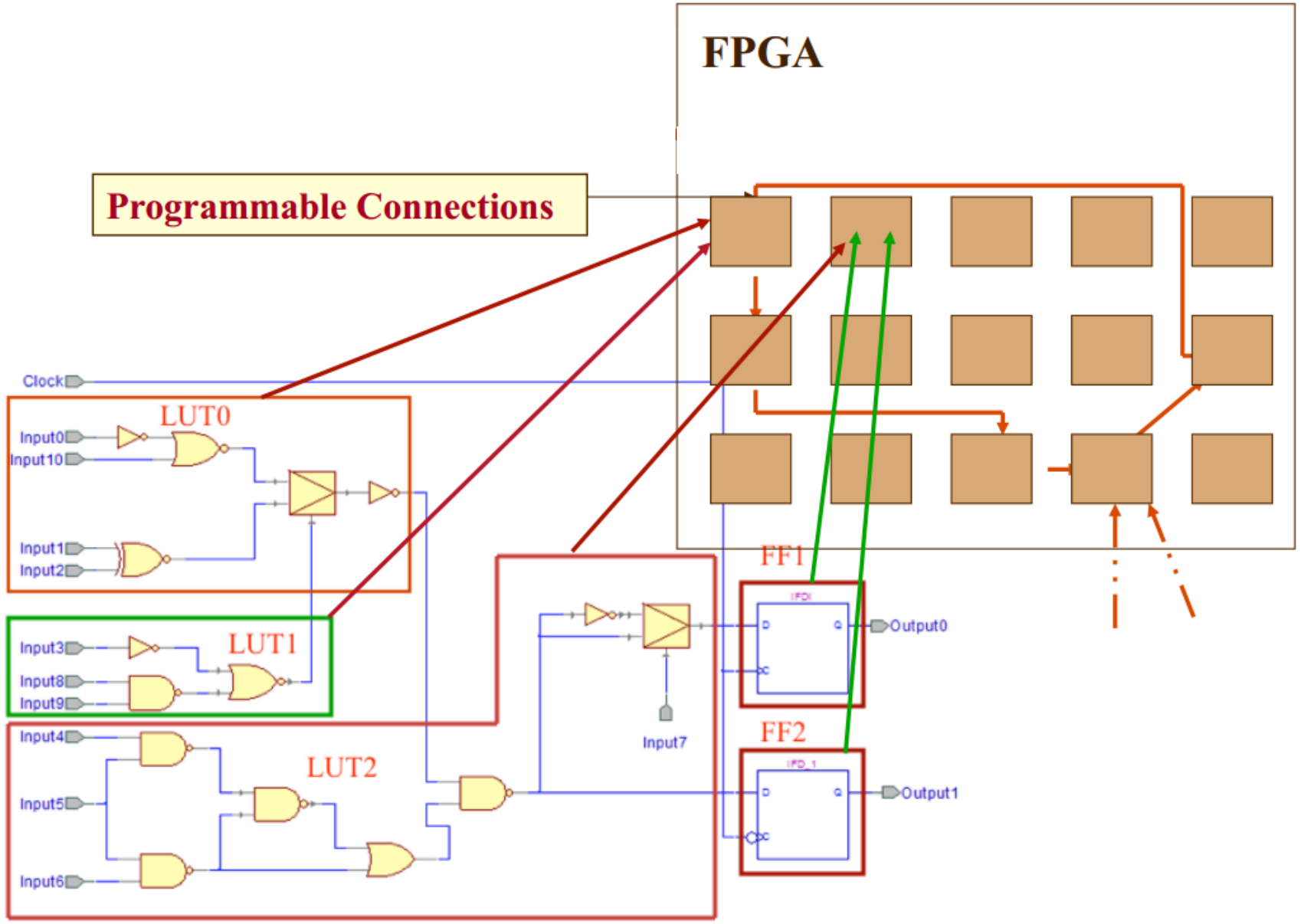
- You can generate a post-map static timing report for your design.
- It lists only the signal path delays in your design, derived from the design logic.
- Useful in evaluating timing performance of the logic paths, particularly if your design does not meeting timing requirements
- Route delays are not accounted (You can eliminate potential problems before investing time in examining routing delays)
- To eliminate problems, you may choose to redesign the logic paths to use fewer levels of logic, tag the paths for specialized routing resources, move to a faster device, or allocate more time for the path.

Placing

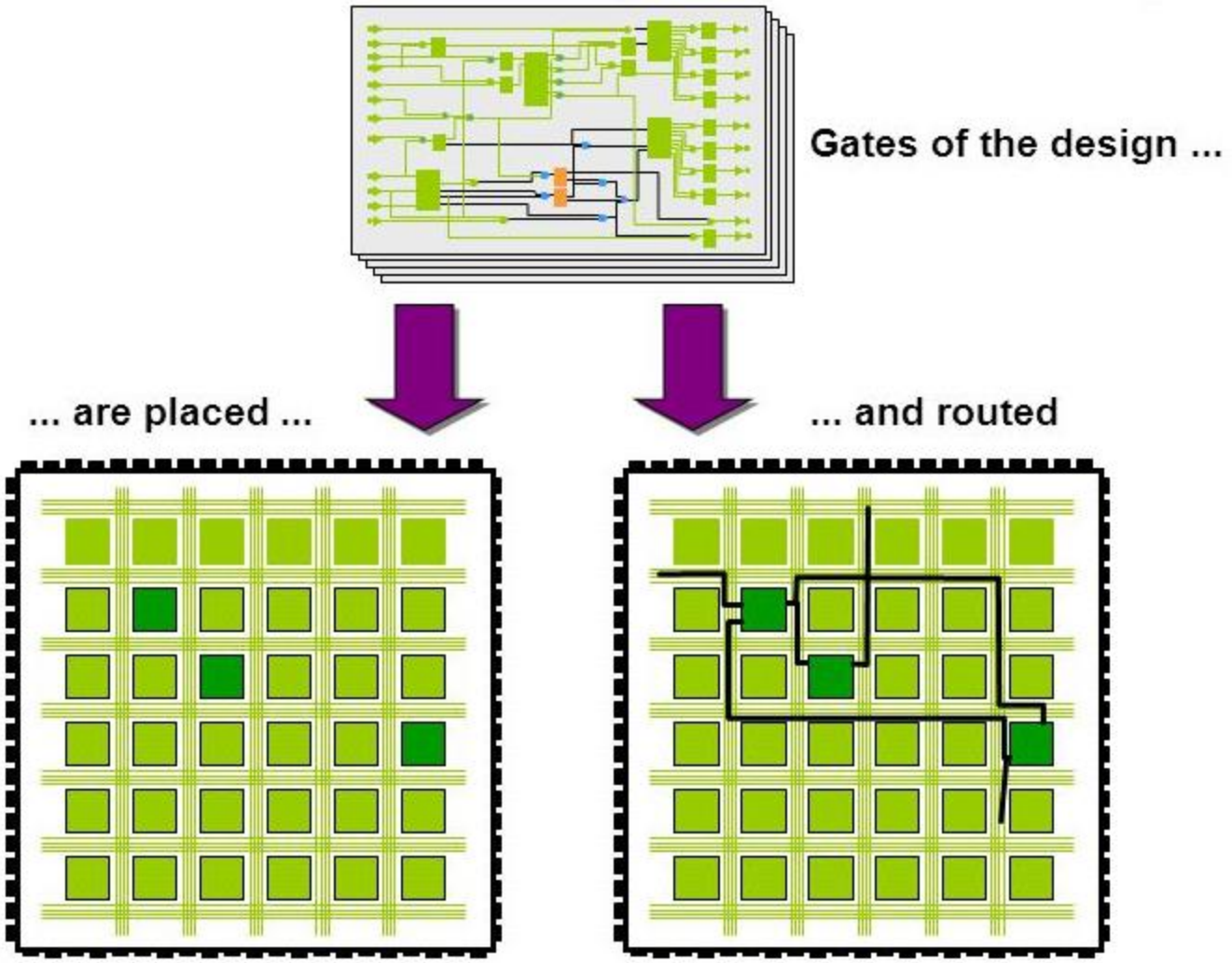
- The Place and Route process takes a mapped NCD file, places and routes the design, and produces an NCD file that is used as input for bitstream generation.



Routing


FPGA
Programmable Connections
Clock
LUT0
Input0
Input10
Input1
Input2
LUT1
Input3
Input8
Input9
LUT2
Input4
Input5
Input6
Input7
FF1
Output0
FF2
Output1

Routing

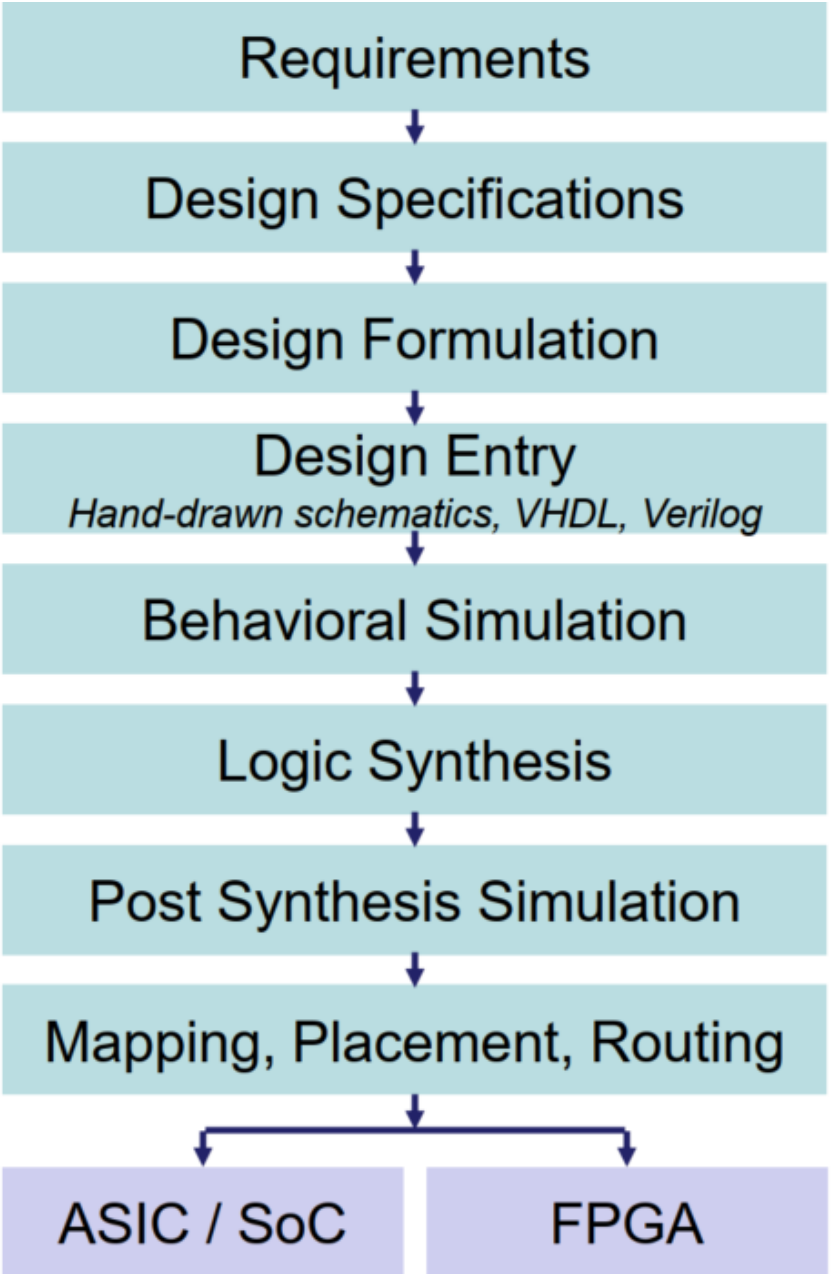


Post-place and Route Static Timing

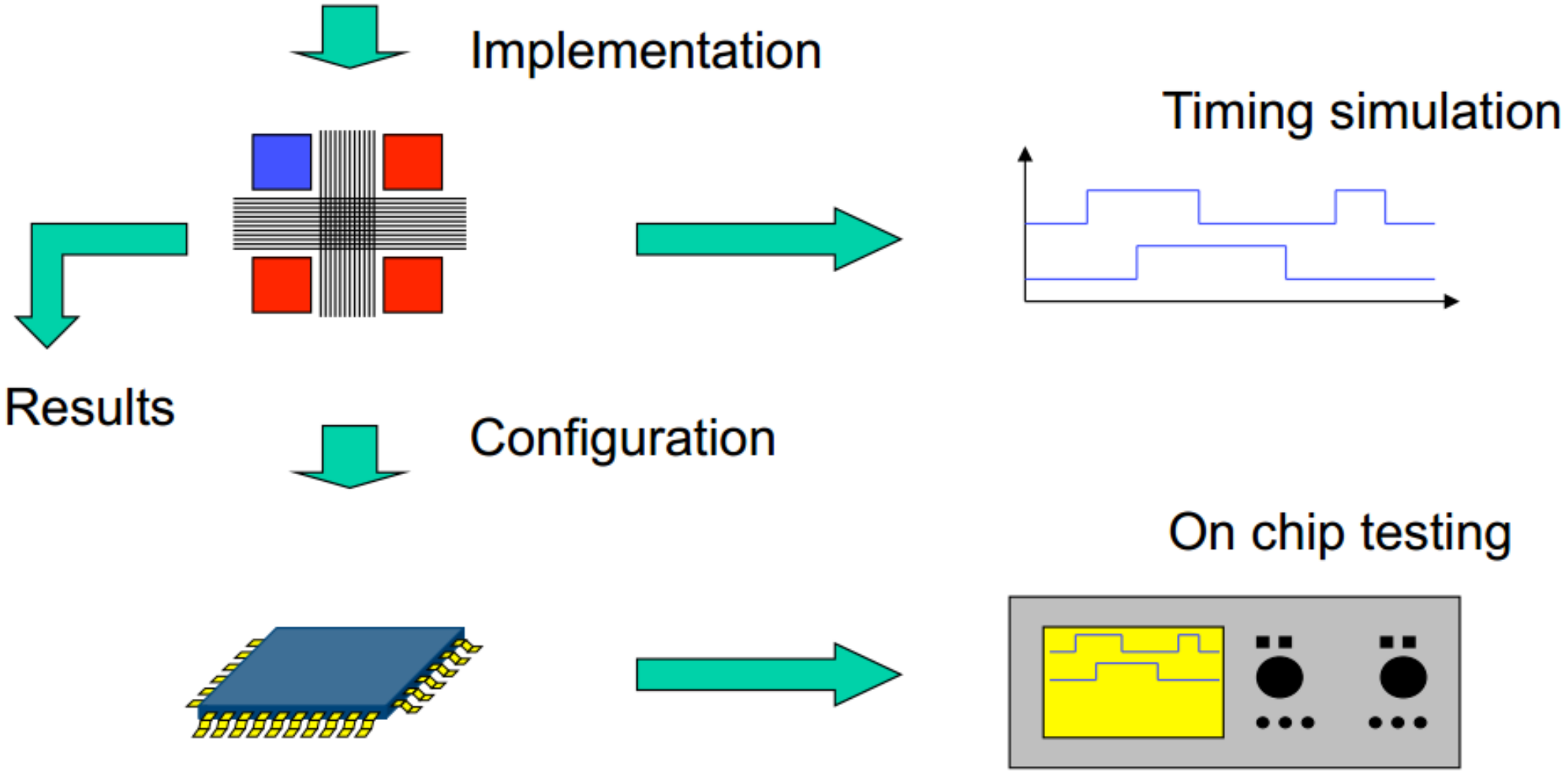
- Incorporates timing delay (signal and routing delay) information to provide a comprehensive timing summary of your design
- If you identify problems in the timing report, you can try fixing the problems by increasing the placer effort level, using re-entrant routing, or using multi-pass place and route.
- You can also redesign the logic paths to use fewer levels of logic, move to a faster device, or allocate more time for the paths.
- If a placed and routed design has met all of your timing constraints, then you can then create configuration data.

FPGA Design Flow

- FPGA programming simply involves writing a sequence of 0 and 1 into the programmable cells of FPGAs
- Once a design is implemented, you must create a file that the FPGA can understand. This file is called a bitstream: a BIT file (.bit extension)
- The BIT file can be downloaded directly to the FPGA or can be converted into a PROM file which stores the programming information.



FPGA Design Process



Digital Circuits Revisited!

Combinational vs Sequential Circuits