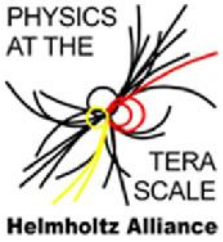
FPGA Design Flow

- from HDL to physical implementation -

Victor Andrei

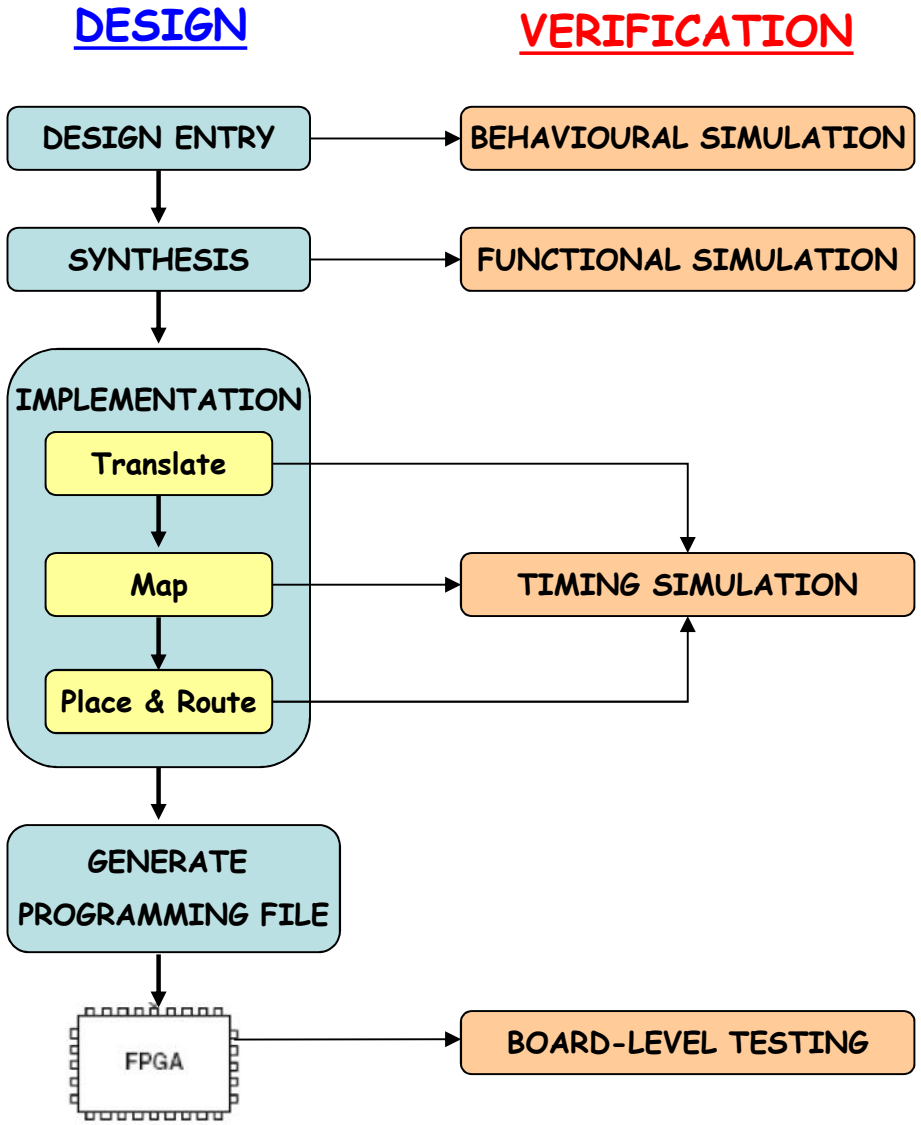
Kirchhoff-Institut für Physik (KIP)
Ruprecht-Karls-Universität Heidelberg

6th Detector Workshop of the Helmholtz Alliance
"Physics at the Terascale", Mainz, 26/02/2013



Overview

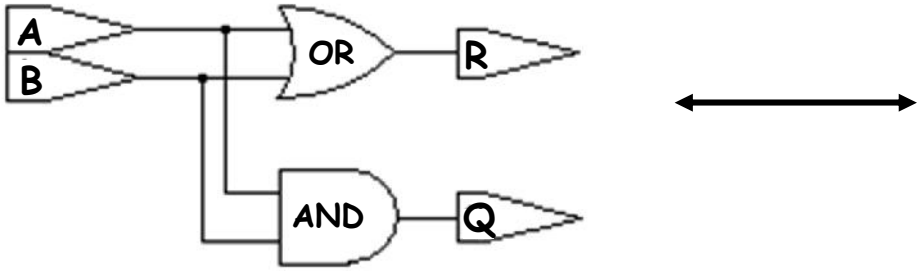
- Five main FPGA development phases:
  - Design Entry
  - Synthesis
  - Implementation
  - Bitstream Generation
  - Simulation
- Complexity of HDL design
  - does not change the flow
  - varies only the execution time
- Implementation of processes is manufacture-dependent → concept is basically the same
- 'Xilinx approach' in this course



Design Entry

- Can be subdivided into two phases:
  - define functionality and structure of the design (*input*)
    - requirements specification, basic architecture, timing diagrams, etc.
    - hierarchy → break design into small, manageable pieces
    - FPGA selection (application, package, temperature range, size, price)
  - create the design (*output*)
    - schematic based → easily readable, but not very convenient for large projects
    - **HDL based** → more convenient and faster, but lower in performance and density
    - mixed types also possible (e.g. schematic with instantiated HDL modules)
    - tools available to convert HDL to schematic and vice versa

Schematic design



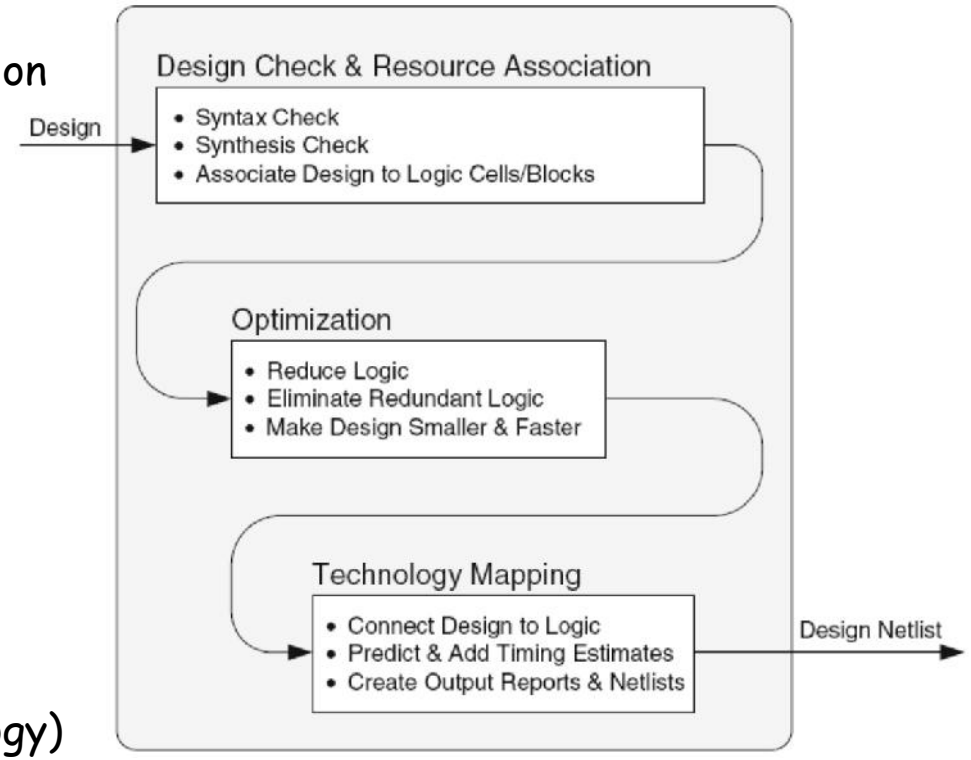
VHDL design

```
entity gates is
    port( A,B: in std_logic;
          Q,R: out std_logic);
end gates;

architecture implement of gates is
begin
  Q <= A and B;
  R <= A or B;
end implement;
```

Synthesis (1/2)

- Converts the input HDL source files into a netlist
  - describes a list of logical FPGA elements and their connectivity
  - multiple netlists generated for complex designs
- Three-step process:
  - syntax check & element association
  - optimisation
  - technology mapping
- Output files:
  - design netlist → Implementation
    - Xilinx format: Native Generic Circuit (NGC)
  - functional netlist → Simulation
  - reports
  - schematic views (RTL & Technology)



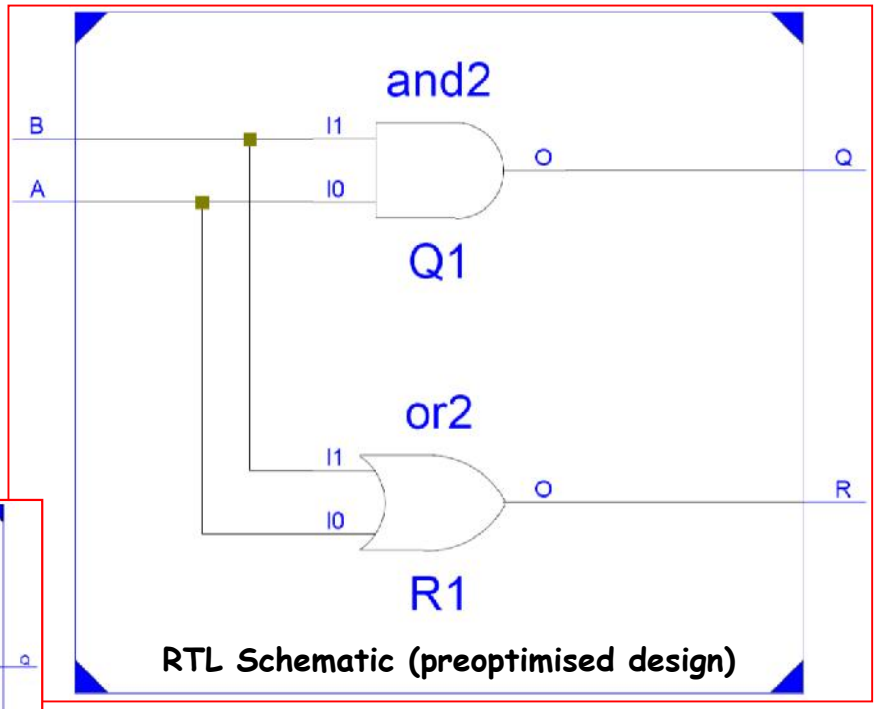
from "FPGAs 101" (G.R.Smith)

Synthesis (2/2)

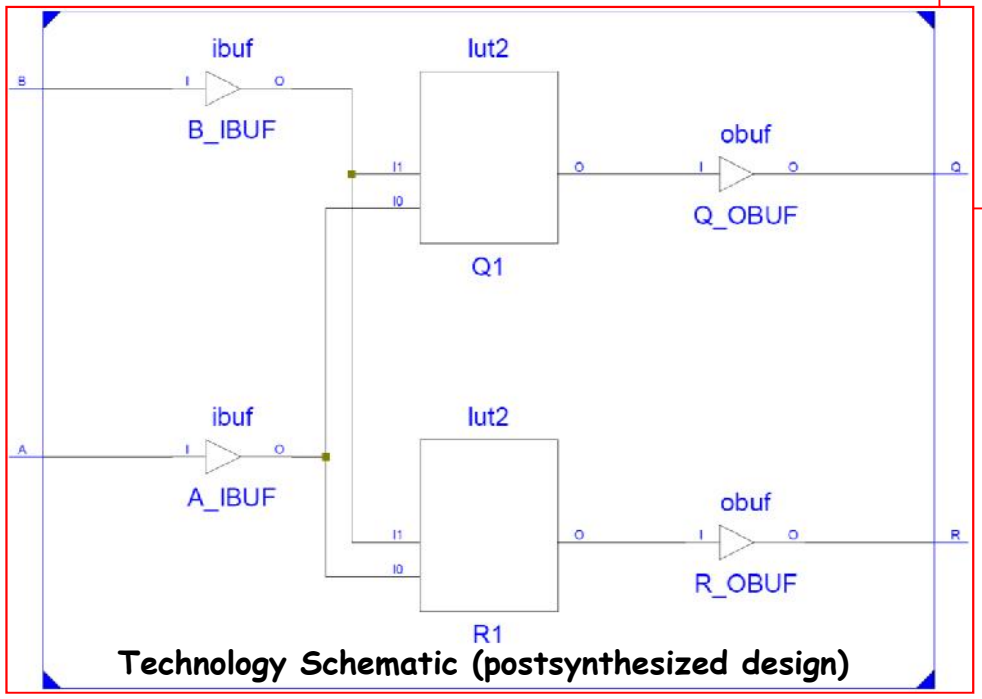
VHDL design

```
entity gates is
    port( A,B: in std_logic;
          Q,R: out std_logic);
end gates;

architecture implement of gates is
begin
  Q <= A and B;
  R  <= A or B;
end implement;
```



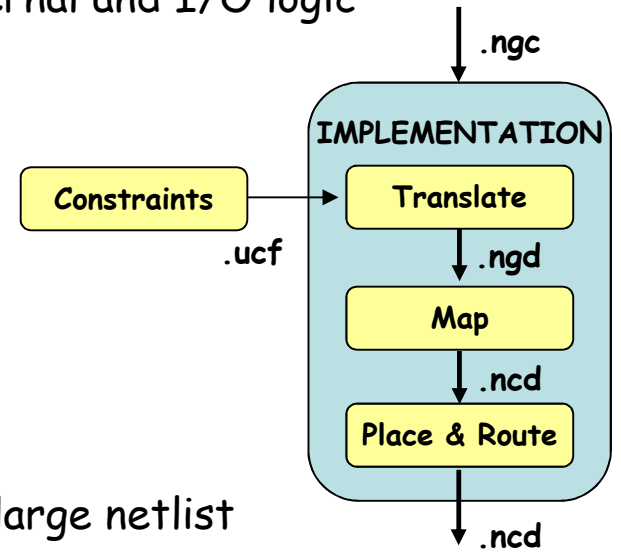
RTL Schematic (preoptimised design)



Technology Schematic (postsynthesized design)

Implementation (1/4)

- Determines the physical design layout
  - maps synthesised netlists (.ngc) to the target FPGA's structure
  - interconnects design resources to FPGA's internal and I/O logic
- Sequence of three sub-processes:
  - Translate
  - Map
  - Place & Route (PAR)

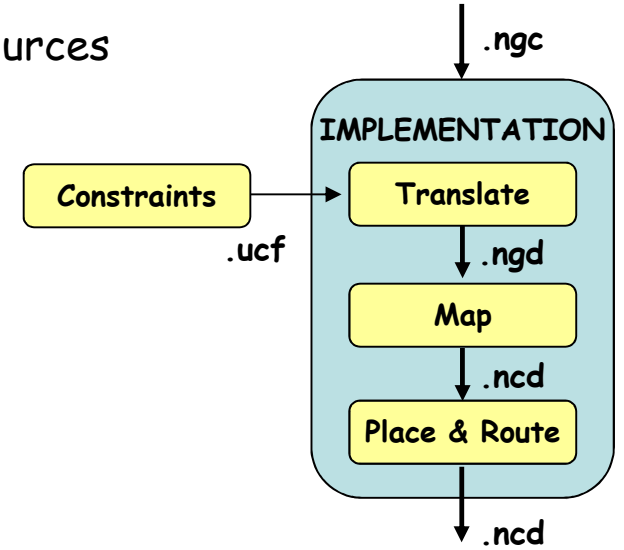


- Translate
  - combines all netlists and constraints into one large netlist
    - Xilinx format: Native Generic Database (NGD)
  - user-defined constraints:
    - pin assignment & time requirements (e.g. input clock period, maximum delay, etc.)
    - information provided via a User Constraints File (UCF)

```
NET "A"        LOC = B3  | IOSTANDARD = LVCMOS33 | PULLDOWN;
NET "Q"        LOC = L6  | IOSTANDARD = LVCMOS18;
```

Implementation (2/4)

- Map
  - compares the resources specified in the input netlist (.ngd) against the available resources of the target FPGA
    - insufficient or incorrectly specified resources generate errors
  - divides netlist circuit into sub-blocks to fit into the FPGA logic blocks
  - output:
    - Native Circuit Description (NCD, .ncd) file



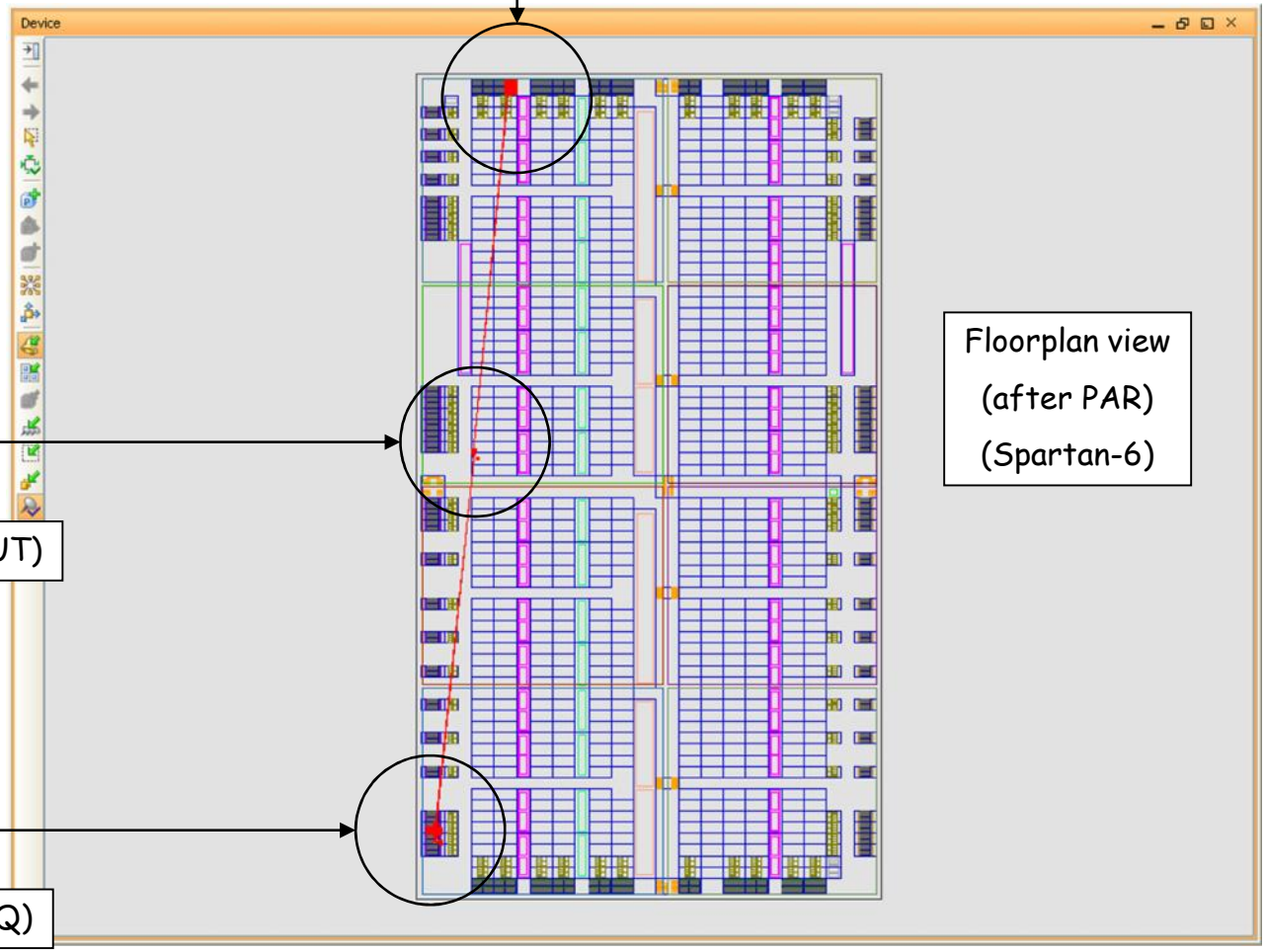
- Place & Route (PAR)
  - iterative process, very time intensive
  - *places* physically the NCD sub-blocks into FPGA logic blocks
  - *routes* signals between logic blocks such that timing constraints are met
  - output:
    - a completely routed NCD file

Implementation (3/4)

VHDL design

```
entity gates is
    port( A,B: in std_logic;
          Q,R: out std_logic);
end gates;

architecture implement of
        gates is
begin
  Q <= A and B;
  R  <= A or B;
end implement;
```

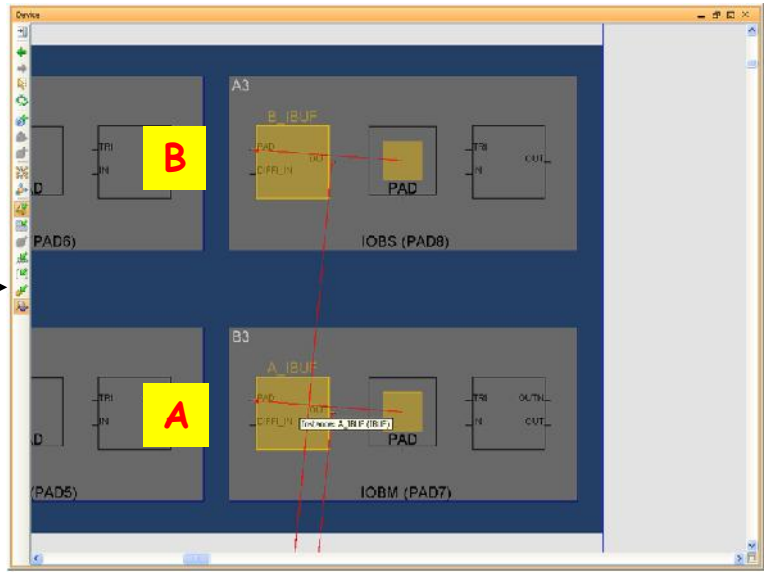


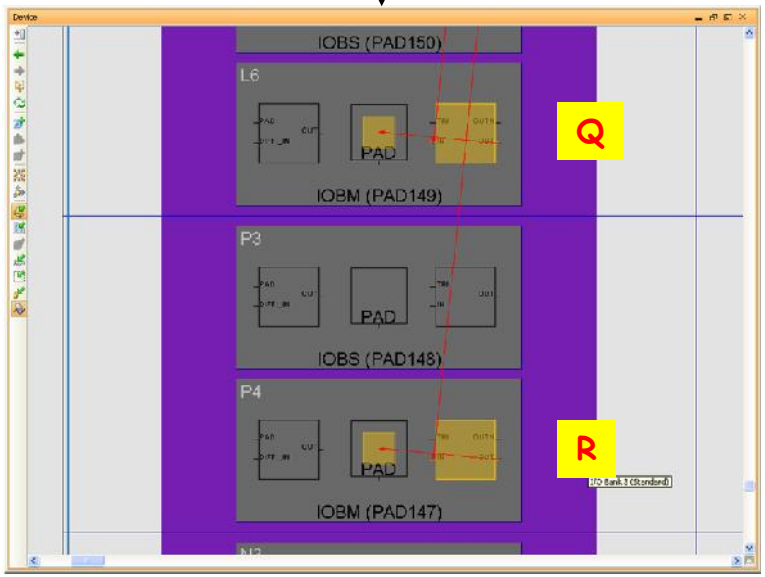
Implementation (4/4)

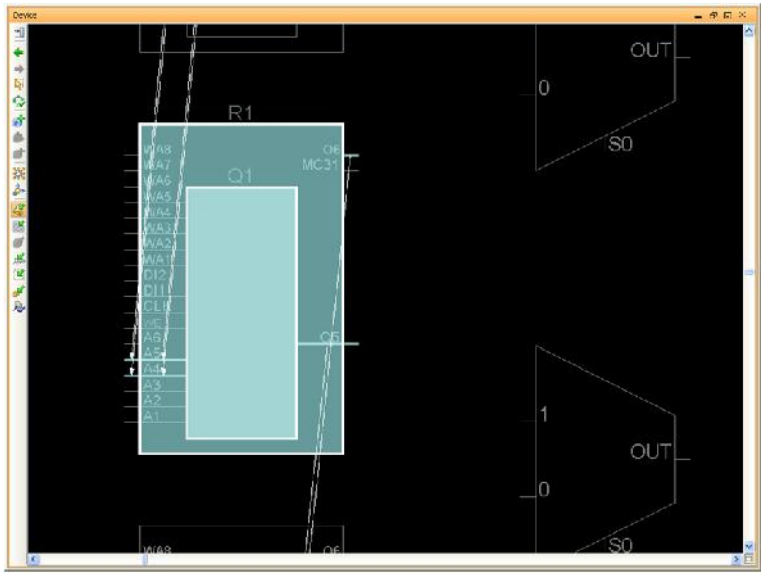
Input pins & logic (A,B)

Combinational Logic (LUT)

Output pins & logic (R,Q)





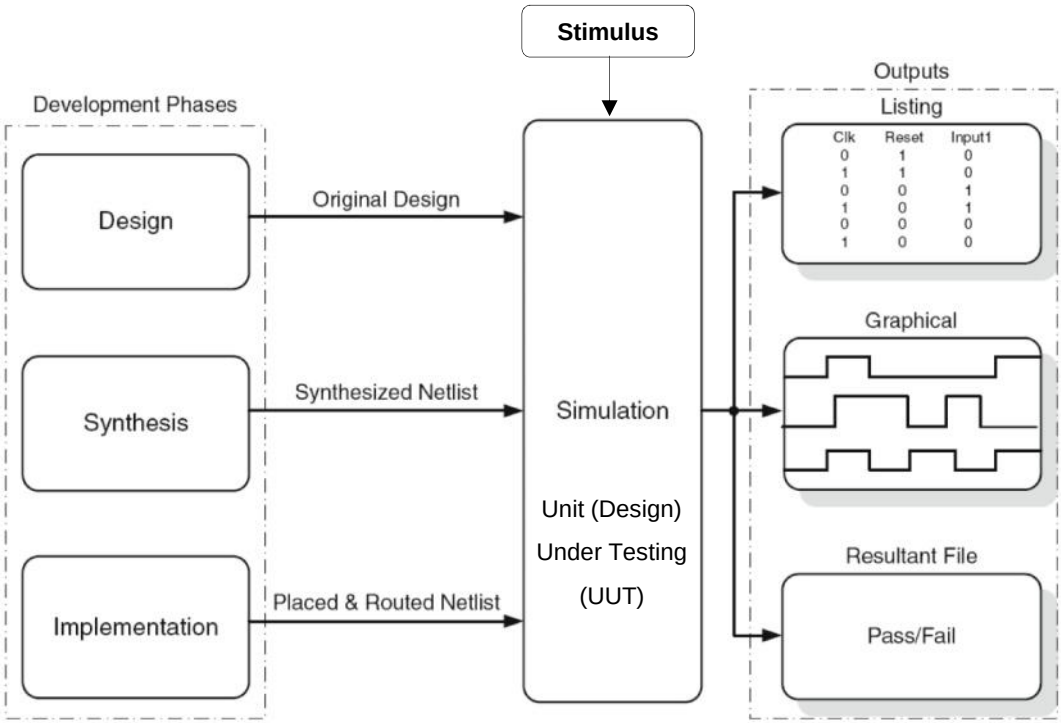


Generate Programming File

- Converts the final NCD file into a format the FPGA understands
  - bitstream (.bit) is the usual choice
    - can be directly loaded into FPGA (e.g. via JTAG interface)
    - or stored in non-volatile devices (PROMs, Flash) → downloaded to FPGA automatically (e.g. at power-up) or upon request
  - other similar choice: IEEE 1532 configuration file format (.isc)

Simulation (1/3)

- Verifies that design performs at the required functionality
- Not mandatory, but should not be completely ignored
- Approach:
  - apply stimulus that mimics real input data to the design
  - observe the output
- Can be performed at different stages:



adapted from "FPGAs 101" (G.R.Smith)

  - Design Entry: *Behavioural* simulation → most frequently run, detects logic errors
  - Post-Synthesis: *Functional* (*Netlist*) simulation
  - Post-Translate, -Map, -PAR: *Timing* simulation → most accurate after PAR

Simulation (2/3)

Testbench

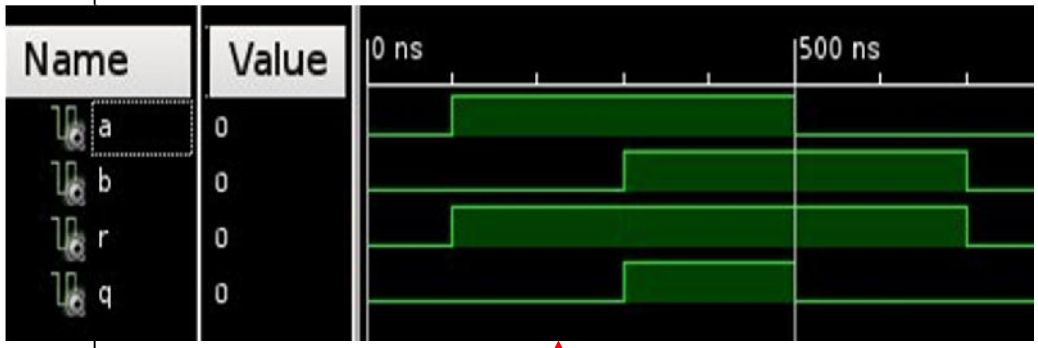
```
ARCHITECTURE behaviour OF tb_gates IS
 -- Component Declaration for the Unit Under Test (UUT)
 COMPONENT gates
   PORT( A, B : IN  std_logic;
         Q, R : OUT  std_logic; );
END COMPONENT;
BEGIN
-- Instantiate the Unit Under Test (UUT)
  uut: gates PORT MAP (A => A, B => B, Q => Q, R => R);
-- Stimulus process
  stim_proc: process
  begin
     A <= '0'; B <= '0';
    -- insert stimulus here
    wait for 100 ns; A <= '1';
    wait for 200 ns; B <= '1';
    wait for 200 ns; A <= '0';
    wait for 200 ns; B <= '0';
    wait;
  end  process;
END;
```

VHDL design (UUT)

```
entity gates is
     port( A,B: in std_logic;
           Q,R: out std_logic);
  end gates;

  architecture implement of gates is
  begin
    Q <= A and B;
    R  <= A or B;
  end implement;
```

Behavioural Simulation – Output Waveform



Simulation (3/3)

Testbench

```
ARCHITECTURE behaviour OF tb_gates IS
 -- Component Declaration for the Unit Under Test (UUT)
 COMPONENT gates
   PORT( A, B : IN  std_logic;
         Q, R : OUT  std_logic; );
END COMPONENT;
BEGIN
-- Instantiate the Unit Under Test (UUT)
  uut: gates PORT MAP (A => A, B => B, Q => Q, R => R);
-- Stimulus process
  stim_proc: process
  begin
     A <= '0'; B <= '0';
    -- insert stimulus here
    wait for 100 ns; A <= '1';
    wait for 200 ns; B <= '1';
    wait for 200 ns; A <= '0';
    wait for 200 ns; B <= '0';
    wait;
  end  process;
END;
```

VHDL design (UUT)

```
entity gates is
     port( A,B: in std_logic;
           Q,R: out std_logic);
  end gates;

  architecture implement of gates is
  begin
    Q <= A and B;
    R  <= A or B;
  end implement;
```

Timing Simulation – Output Waveform