Hardware Modeling [VU] (191.011) – WS25 –

Logic Synthesis, Place and Route

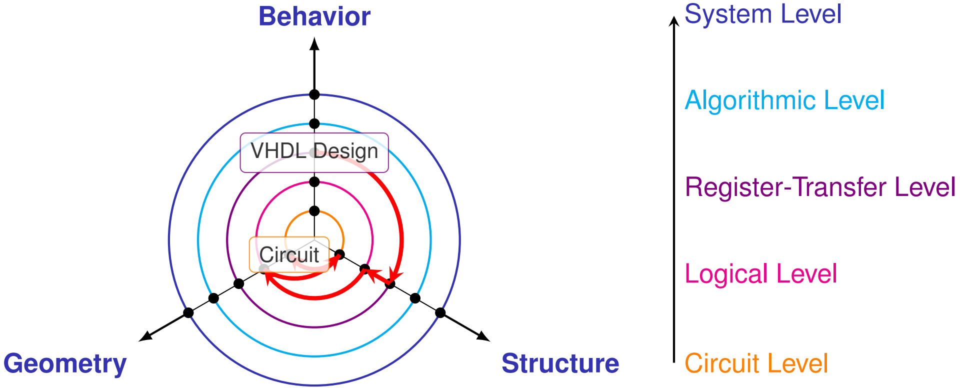
Florian Huemer & Sebastian Wiedemann & Dylan Baumann

WS 2025/26

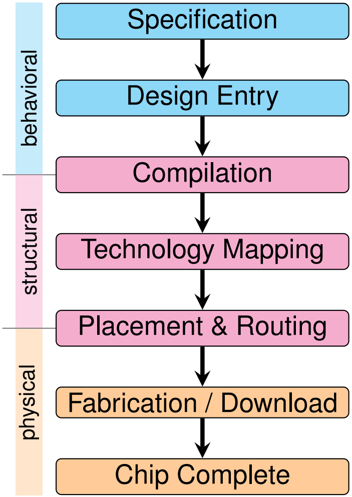
Modified: 2025-12-16, 16:07 (f8a58e9)

Recall: Y-Diagram

- Tools translate between different views
  - VHDL design to circuit implementation
- What do the tools do? How do translations work?



Hardware Design Flow



verbal / formal description of behavior

model in description language (e.g., VHDL)

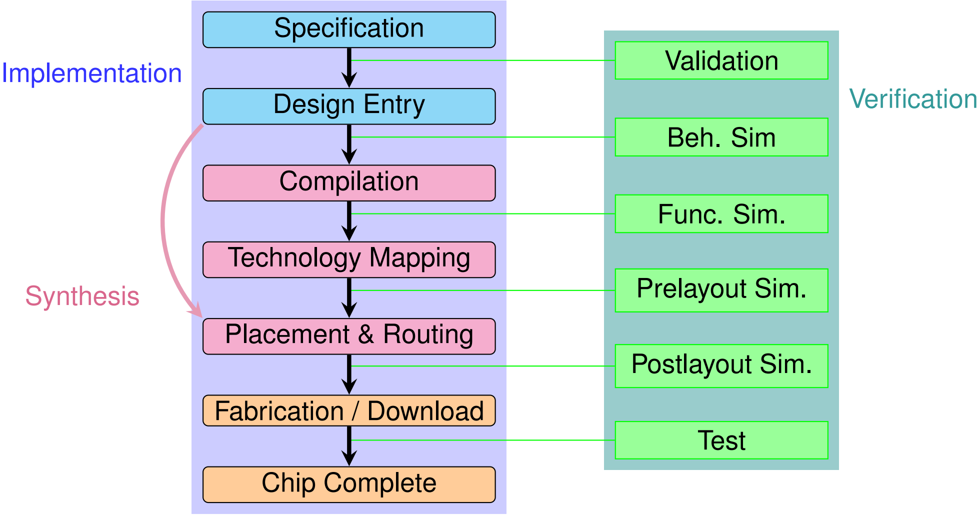
translation to network of generic logic elements

mapping to cells of target technology

placement and interconnection of cells

physical implementation

Hardware Design Flow (cont’d)



Design Entry

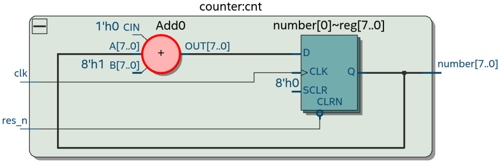
- Description of target circuit (behavioral RTL in VHDL)
  - Readable by tools, basis for simulation and documentation
- Example: Synchronous counter

```vhdl
entity counter is
  port(
    clk, res_n : in std_ulogic;
    number     : out unsigned(7 downto 0)
  );
end entity;

architecture beh of counter is begin
  sync : process (clk, res_n) begin
    if res_n = ’0’ then
      number <= (others => ’0’);
    elsif rising_edge(clk) then
      number <= number + 1;
    end if;
  end process;
end architecture;
```

Compilation

- Design entry converted to **technology-agnostic** *netlist*
- Netlist: circuit as graph ⇔ RTL schematic
- Generic components (MUX, adder, basic gates)
  - Tools determine interfaces and check connections
  - Might not exist in target technology



Technology Mapping

- Map generic elements to target technology
  - E.g.: ASIC, PAL, Gate-Array, **FPGA**
- Requires *target library* of available cells
  - Gates, sequential elements, etc.
- Target technology is now fixed

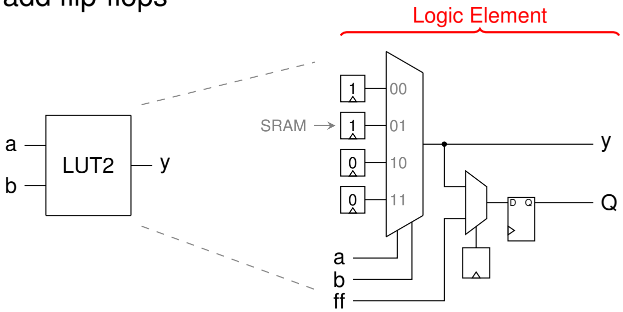
Introduction to FPGAs

- **F**ield **P**rogrammable **G**ate **A**rray ?!
- Generic circuit, configurable for target circuit
- Programmable logic cells and interconnect
  - + Large volumes ⇒ comparably cheap
  - + Already fabricated ⇒ simpler, faster design
  - - Already fabricated ⇒ less optimizations
  - - Generic design ⇒ inevitable overhead
- How can such circuits be built?

Look-Up Tables (LUTs)

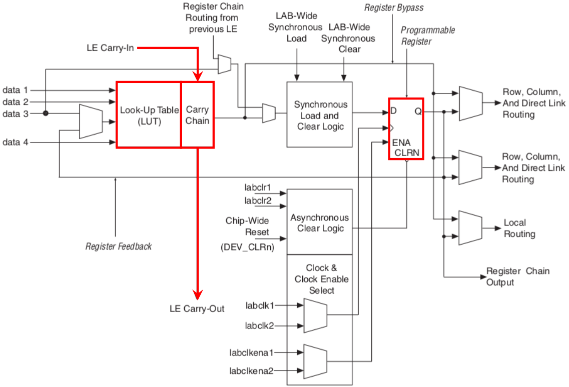
- How to implement arbitrary logic functions?
  - Map inputs to outputs ⇒ truth table
  - ⇒ Same in FPGAs: *Look-up Tables* (LUTs)
  - Store in SRAM cells, use inputs to select
- For sequential logic, add flip-flops
- *Logic Element* (LE)

a	b	y
0	0	1
0	1	1
1	0	0
1	1	0



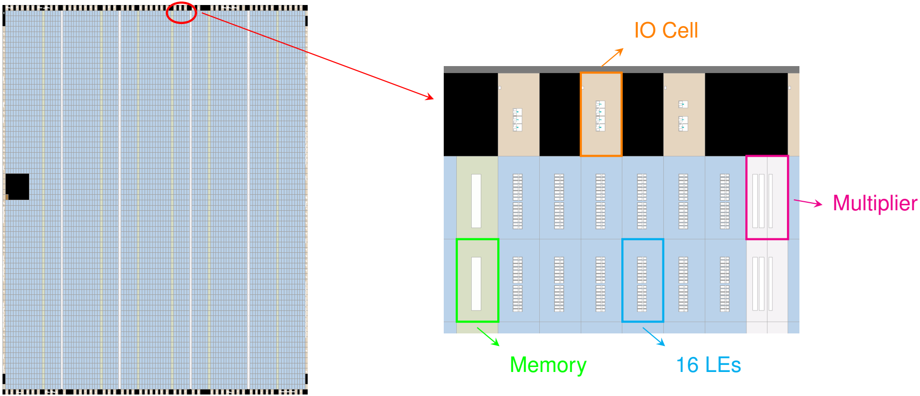
LEs in Real FPGAs

- Real LEs are more complex and powerful
- Bigger LUTs (3-6 inputs), dedicated carry chains
- Clear/set logic, sometimes multiple FFs



FPGA Structure

- FPGA: grid of LE clusters and other elements

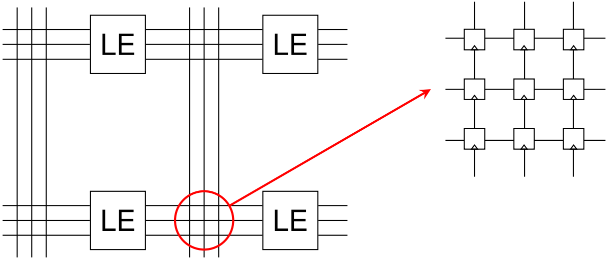


FPGA Programmable Interconnect

- How to connect LEs to form complex circuits?

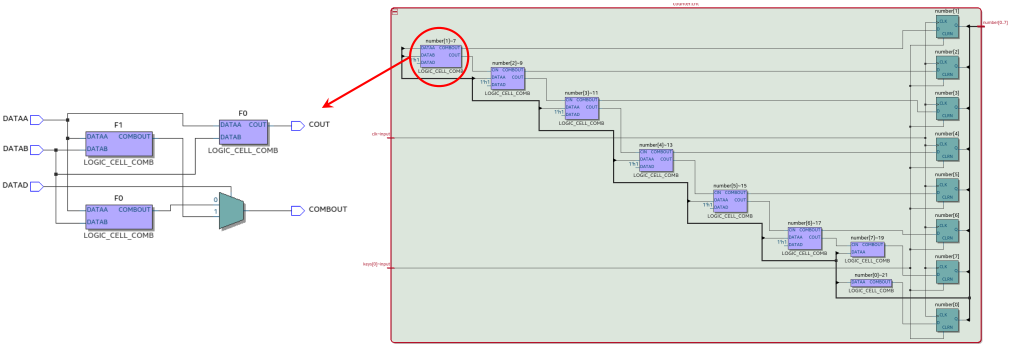
⇒ Programmable interconnect

  - Programmable via SRAM (connected through MUXes)



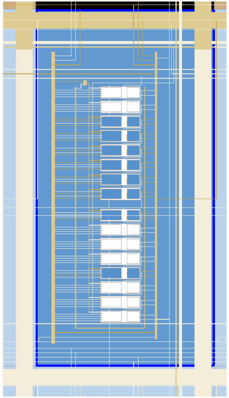
Technology Mapping to an FPGA

- Generic elements implemented using LEs
- Each LE implements part of the logic
  - Here: carry chain for adders is used



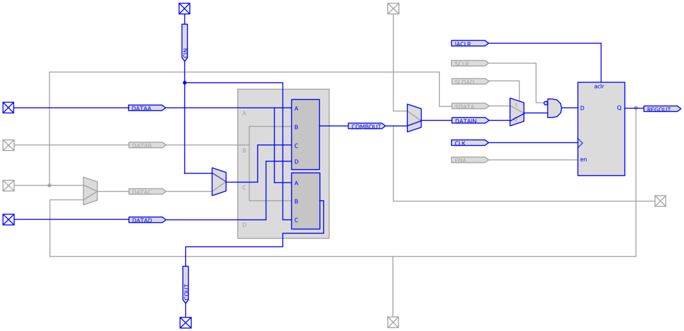
Placement & Routing

- Placement: Choose position of LEs
  - Constrained by physical availability
  - Minimize unknown interconnect
- Routing: Connect placed cells
  - Constrained by available interconnect
  - Goal: Meet timing constraints
- Paramount for timing, non-trivial
  - Multiple iterations
  - Driven by heuristics



FPGA Bitstream

- For FPGAs: result is a *bitstream*
  - Content of all SRAM cells
  - Configures LEs and interconnect



Lecture Complete!