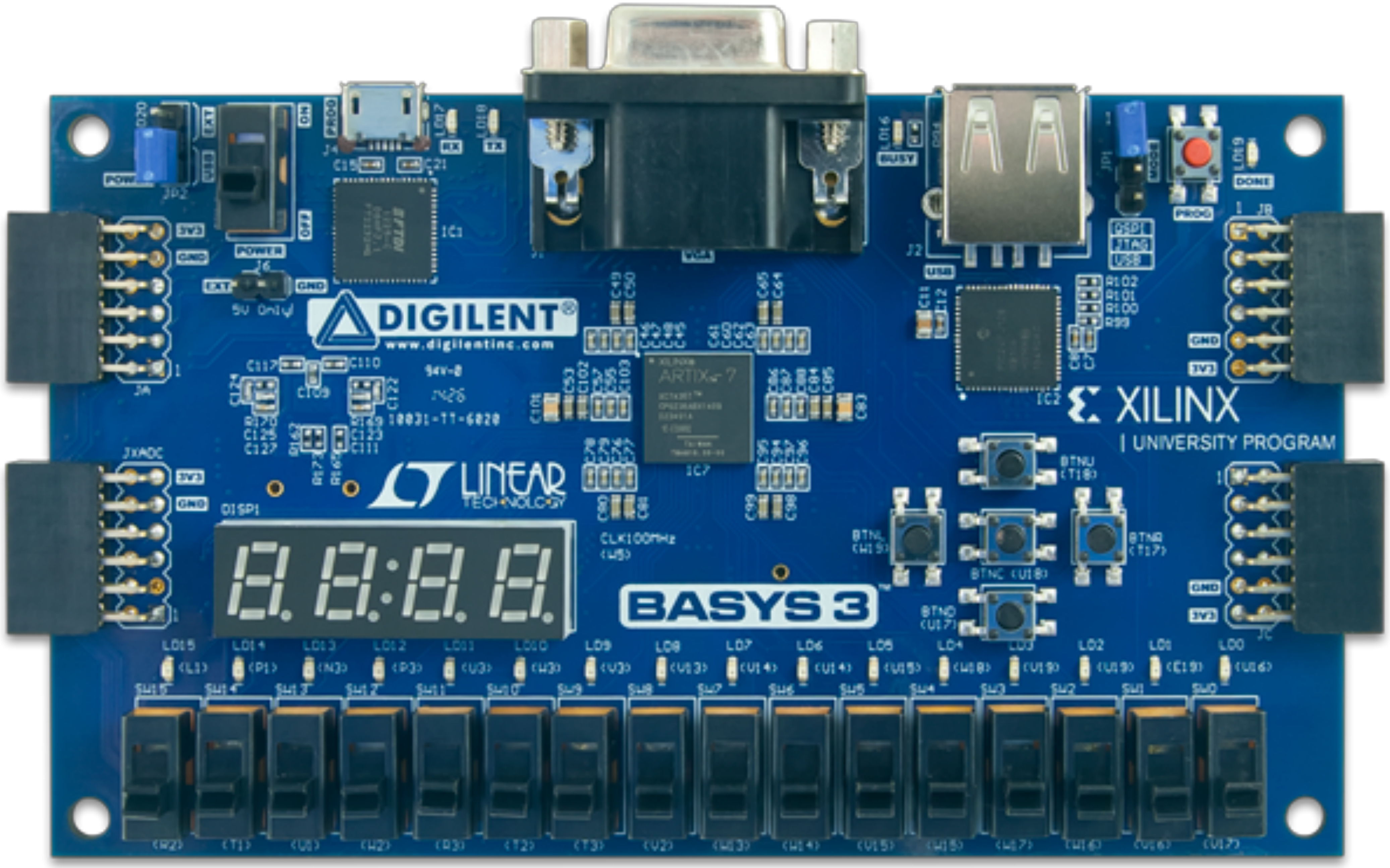
HDLs and SystemVerilog

Digital Computer Design

Logic Arrays

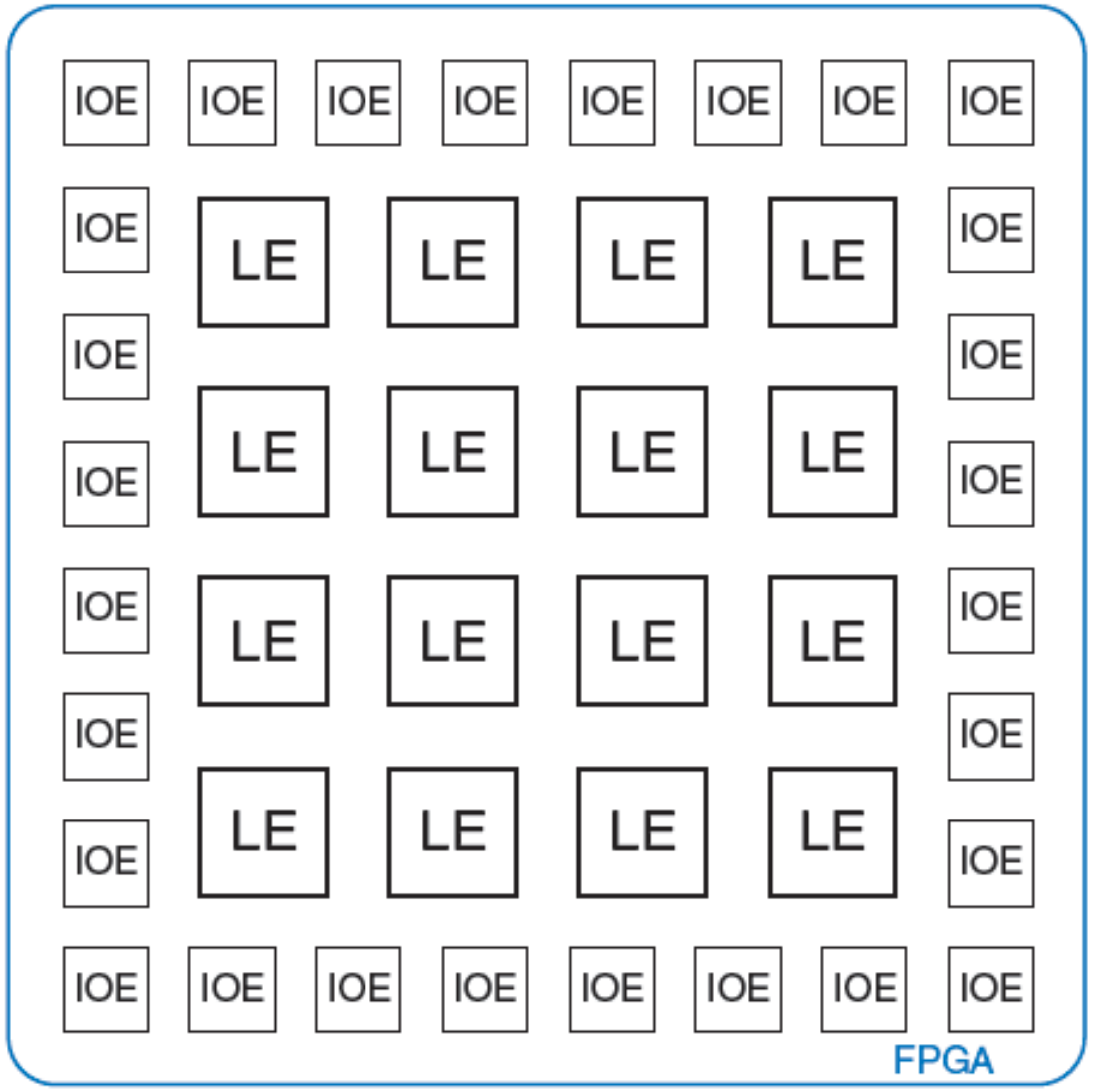
- Gates can be organized into regular arrays.
  - If the connections are made programmable, these logic arrays can be configured to perform any function without the user having to connect wires in specific ways.
- Software tools allow users to map logic designs onto these arrays.
- Two types of logic arrays:
  - programmable logic arrays (PLAs)
  - field programmable gate arrays (FPGAs)

A FPGA Board


DIGILENT
www.digilentinc.com
XILINX
UNIVERSITY PROGRAM
LINEAR TECHNOLOGY
BASYS 3
ARTIX-7
CLK100MHz
BTNU
BTNL
BTNC
BTNR
BTND

FPGA: Field Programmable Gate Array

- FPGA is a reconfigurable substrate
  - Reconfigurable functions
  - Reconfigurable interconnection of functions
  - Reconfigurable input/output (IO)
- FPGAs fill the gap between software and hardware
  - Achieves higher performance than software
  - Maintains more flexibility than hardware

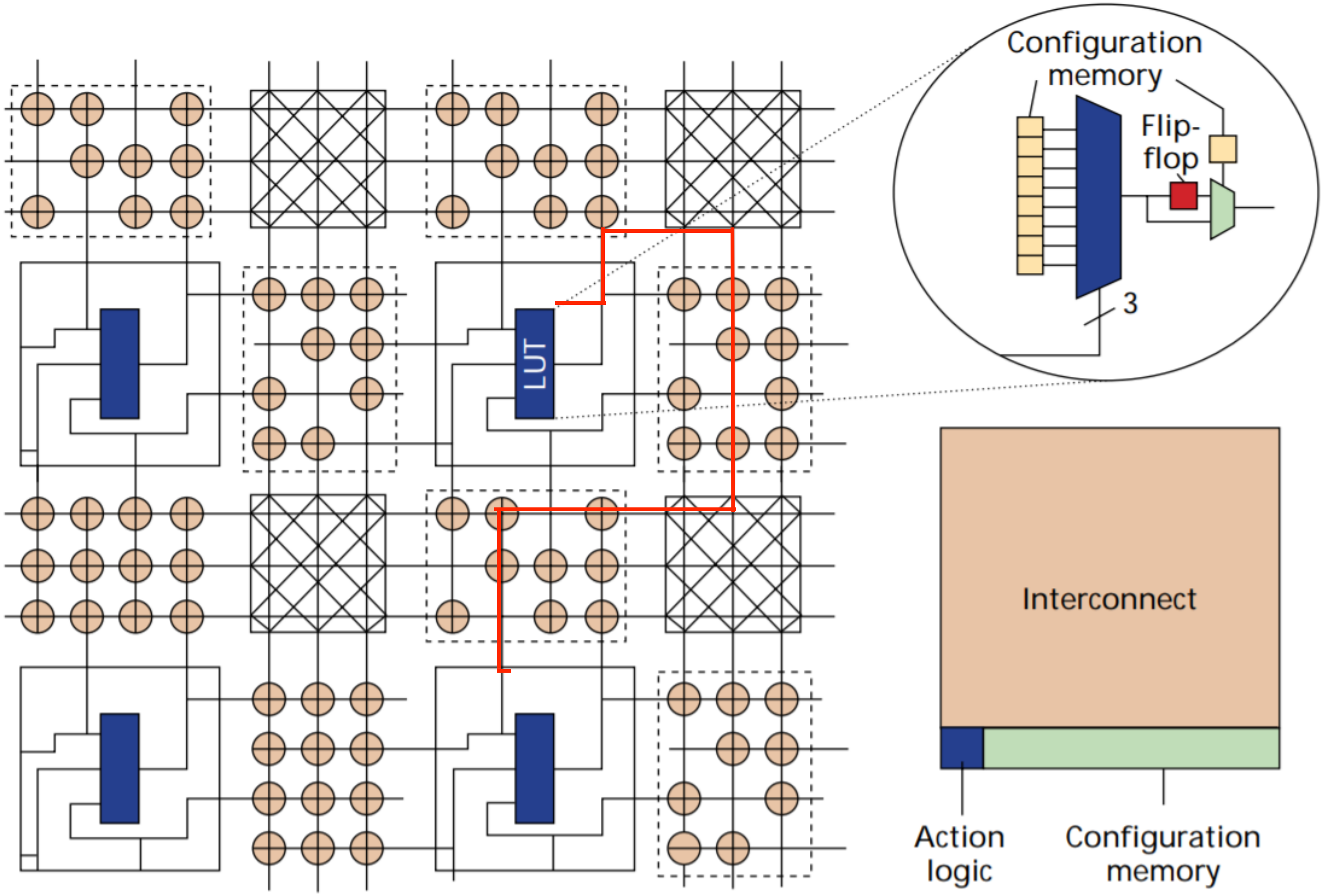


LE: Logic Element

- Composed of:
  - **LUTs** (lookup tables): perform combinational logic
  - **Flip-flops:** perform sequential logic
  - **Multiplexers:** connect LUTs and flip-flops

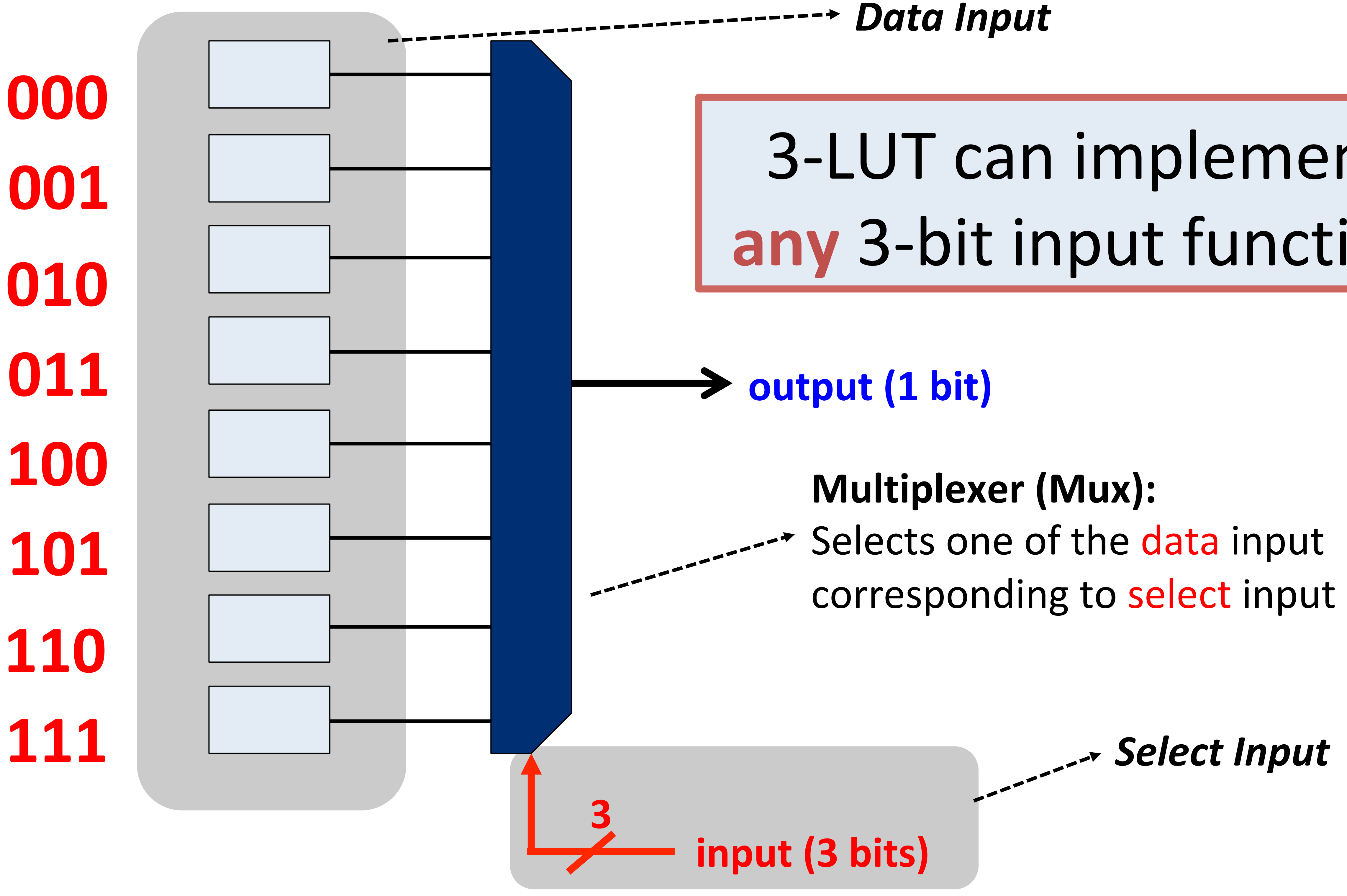
FPGA Architecture - Looking Inside an FPGA

- Two main building blocks:
  - Look-Up Tables (LUT) and Switches



How Do We Program LUTs?

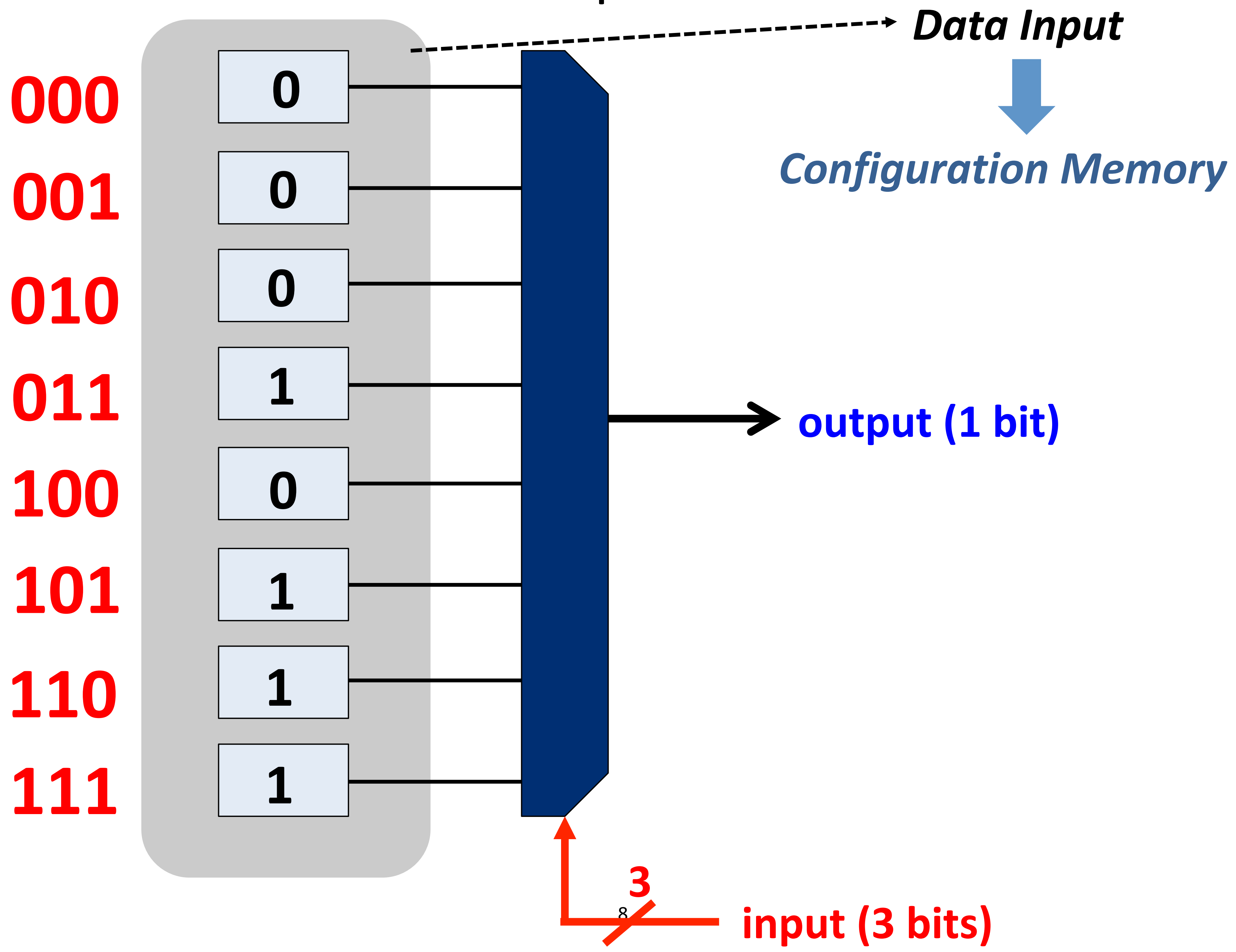
- 3-bit input LUT (3-LUT)



3-LUT can implement **any** 3-bit input function

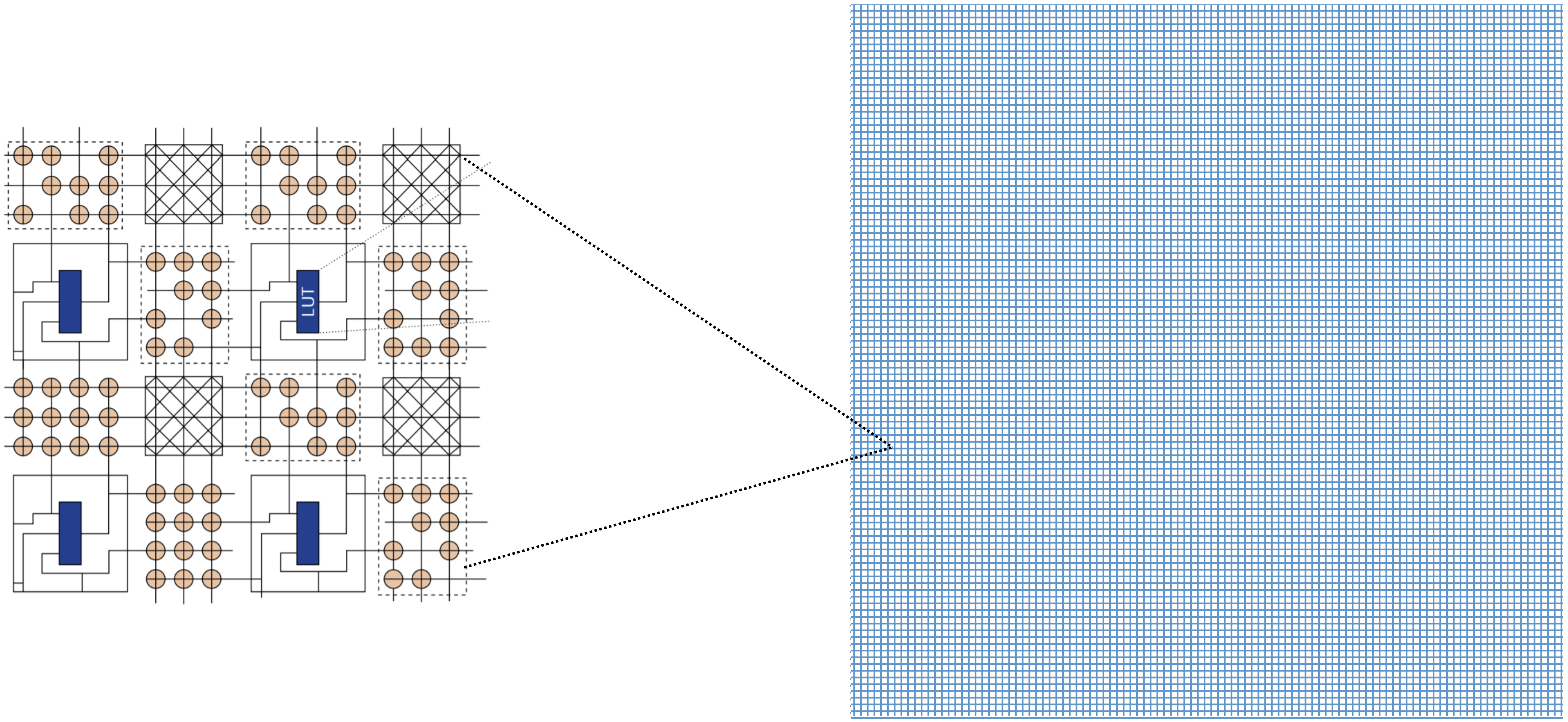
An Example of Programming a LUT

- Let's implement a function that outputs '1' when there are more than one '1' in select inputs



How to Implement Complex Functions?

- FPGAs are composed of a large number of LUTs and switches



Advantages & Disadvantages of FPGAs

- Advantages
  - Low development cost
  - Short time to market
  - Reconfigurable in the field
  - Reusability
  - An algorithm can be implemented directly in hardware
    - No ISA, high specialization
- Disadvantages
  - Not as fast and power efficient as *application specific hardware*
  - Reconfigurability adds significant area overhead

Computer-Aided Design (CAD) Tools

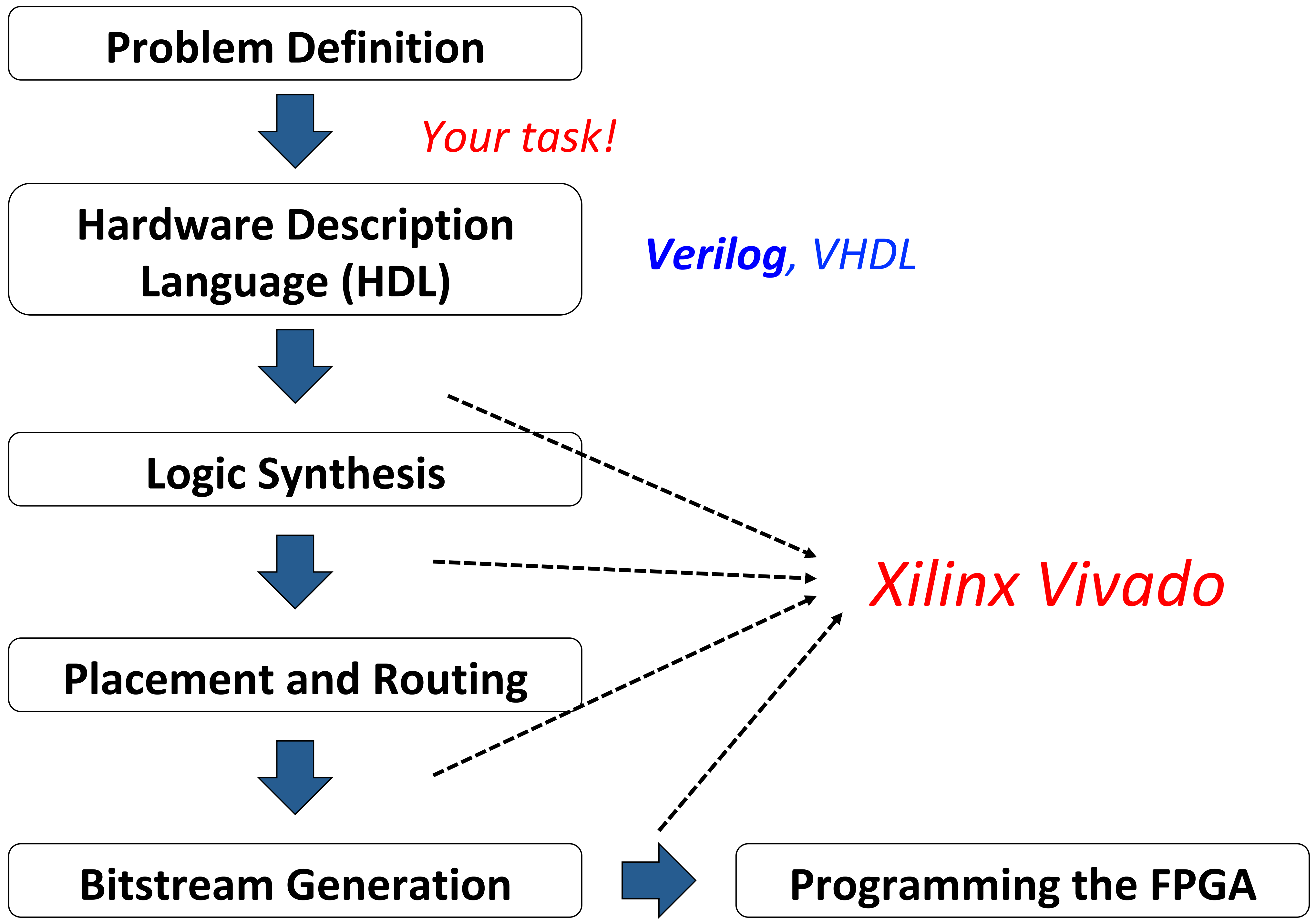
FPGAs have many resources (e.g., LUTs, switches)

They are hard to program manually

How can we

- represent a high-level functional description of our hardware circuit using the FPGA resources?
- select the resources to map our circuit to?
- optimally configure the interconnect between the selected resources?
- generate a final configuration file to properly configure an FPGA?

FPGA Design Flow



Hardware description language (HDL)

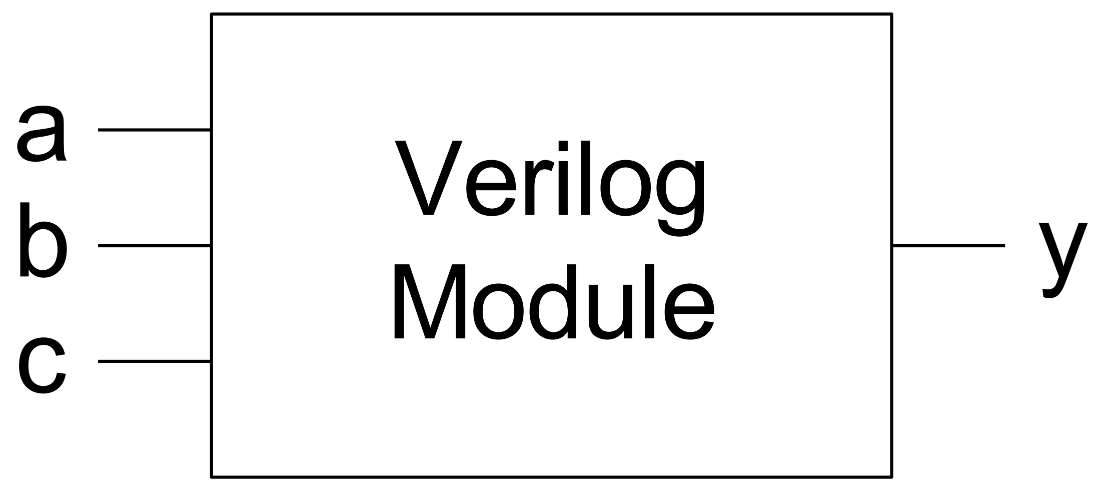
- Specifies logic function only
  - Computer-aided design (CAD) tool produces or ***synthesizes*** the optimized gates
  - Think of the **hardware** the HDL should produce!
- Most commercial designs built using HDLs
- Two leading HDLs:
  - **SystemVerilog**
    - developed in 1984 by Gateway Design Automation
    - IEEE standard (1364) in 1995
    - Extended in 2005 (IEEE STD 1800-2009)
  - **VHDL 2008**
    - Developed in 1981 by the Department of Defense
    - IEEE standard (1076) in 1987
    - Updated in 2008 (IEEE STD 1076-2008)

HDL to Gates

- **Simulation**
  - Inputs applied to circuit
  - Outputs checked for correctness
  - Millions of dollars saved by debugging in simulation instead of hardware
- **Synthesis**
  - Transforms HDL code into a *netlist* describing the hardware (i.e., a list of gates and the wires connecting them)

SystemVerilog Modules

- A block of hardware with inputs and outputs is called a module.
- **Two types of Modules:**
  - **Behavioral:** describe what a module does
  - **Structural:** describe how it is built from simpler modules



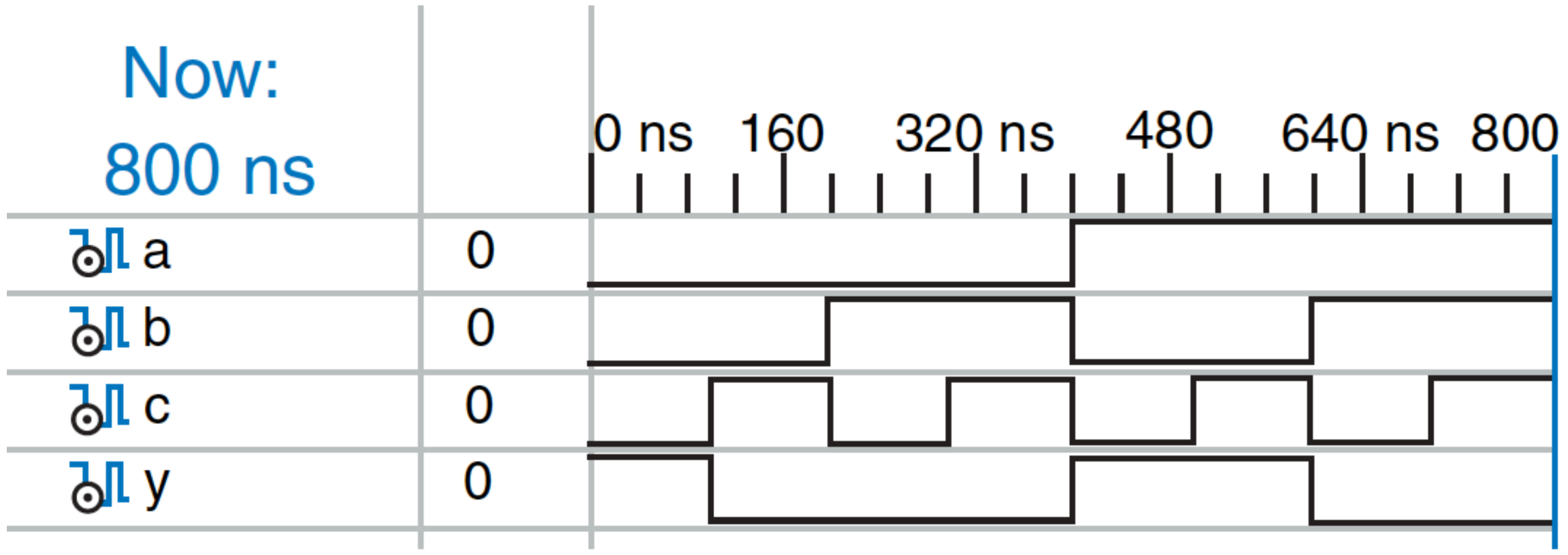
Behavioral SystemVerilog

- `module/endmodule`: required to begin/end module
- `example`: name of the module
- Operators:
  - `~`: NOT
  - `&`: AND
  - `|`: OR

```
module example(input  logic a, b, c,
                      output logic y);
  assign y = ~a & ~b & ~c | a & ~b & ~c | a
   & ~b &  c;
endmodule
```

HDL Simulation

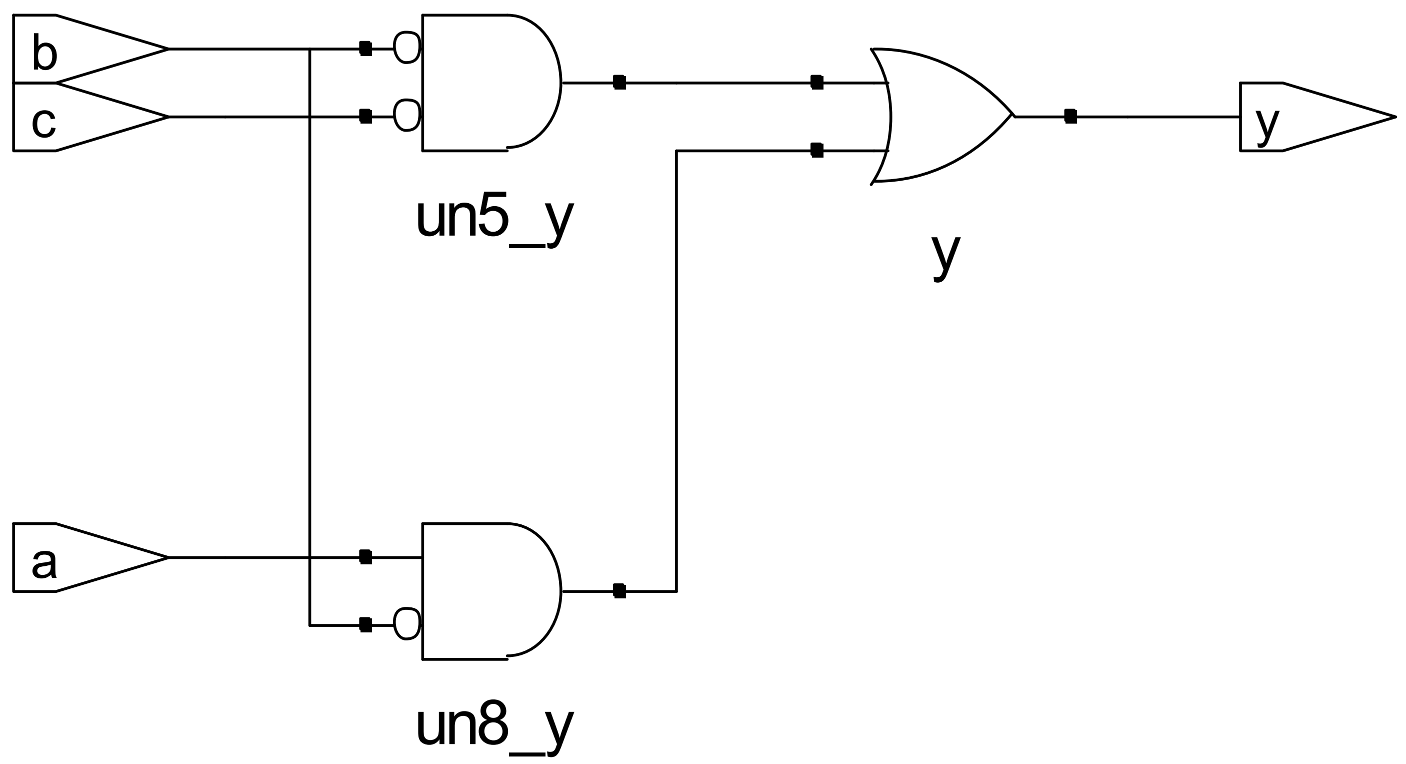
```
module example(input  logic a, b, c,
                      output logic y);
  assign y = ~a & ~b & ~c | a & ~b & ~c | a
   & ~b &  c;
endmodule
```



Inputs applied to circuit, Outputs checked for correctness

HDL Synthesis

```
module example(input  logic a, b, c,
                      output logic y);
  assign y = ~a & ~b & ~c | a & ~b & ~c | a
   & ~b &  c;
endmodule
```



Transforms HDL code into a ***netlist*** describing the hardware (i.e., a list of gates and the wires connecting them)

Further Reading

- You can read **Chapter 4** of your book.