Advanced digital design methodologies

Hipólito Guzmán Miranda
Profesor Titular
Universidad de Sevilla

Teaching context

B02: Advanced Programmable Logic Systems

- Tema 1: FPGA architecture
- Tema 2: Advanced digital design methodologies
- Tema 3: Advanced VHDL
- Tema 4: Verification capabilities for digital circuits

Required prior knowledge:

- FPGA architecture

Contexto

- 60+ years of Moore’s Law (1965)
- Design Gap
- Verification Gap

But before this…

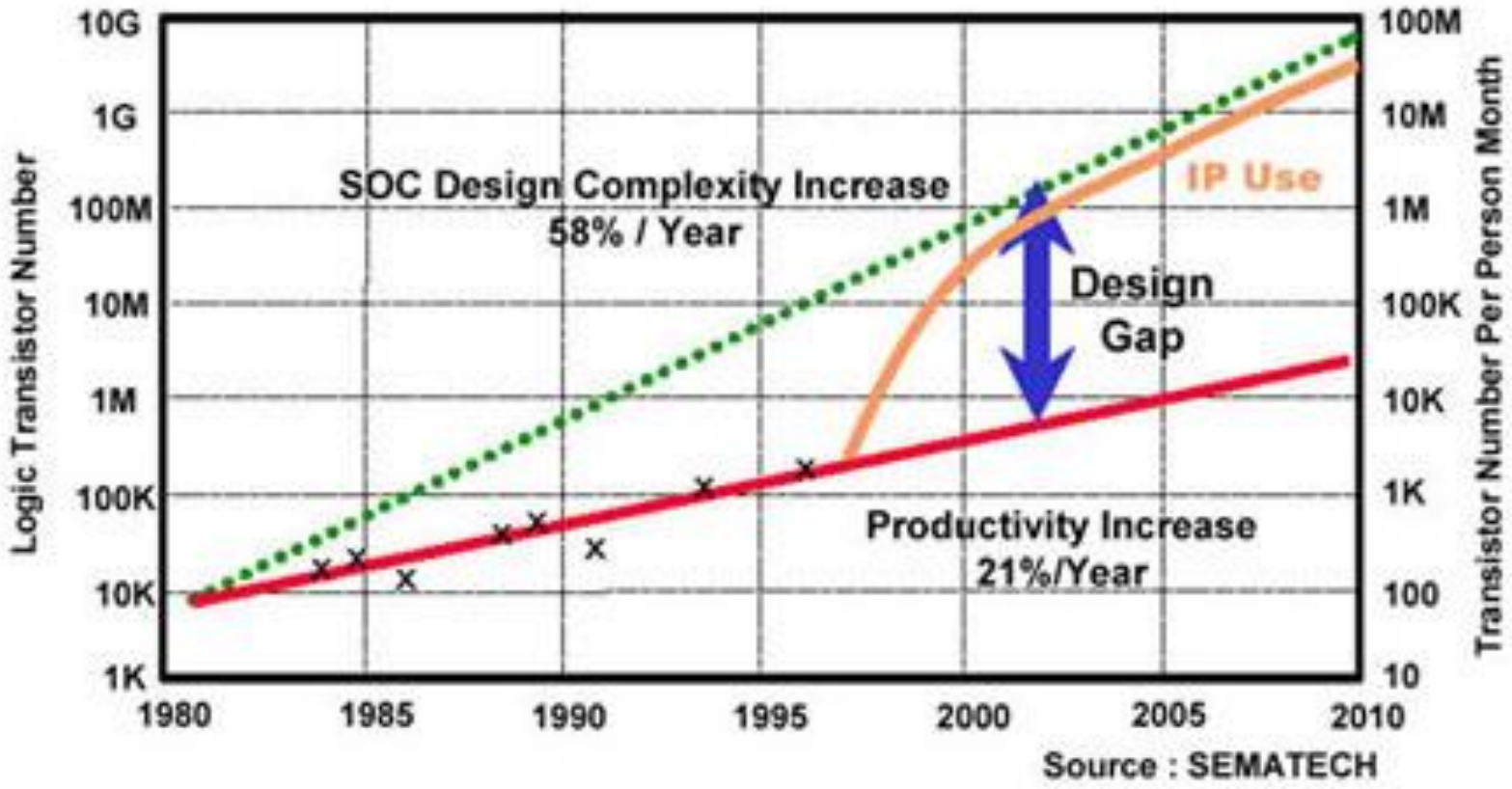
Do you know about what the inside of an FPGA is like?

Complex digital systems

Scaling of complexity:

- Design gap
- Verification gap

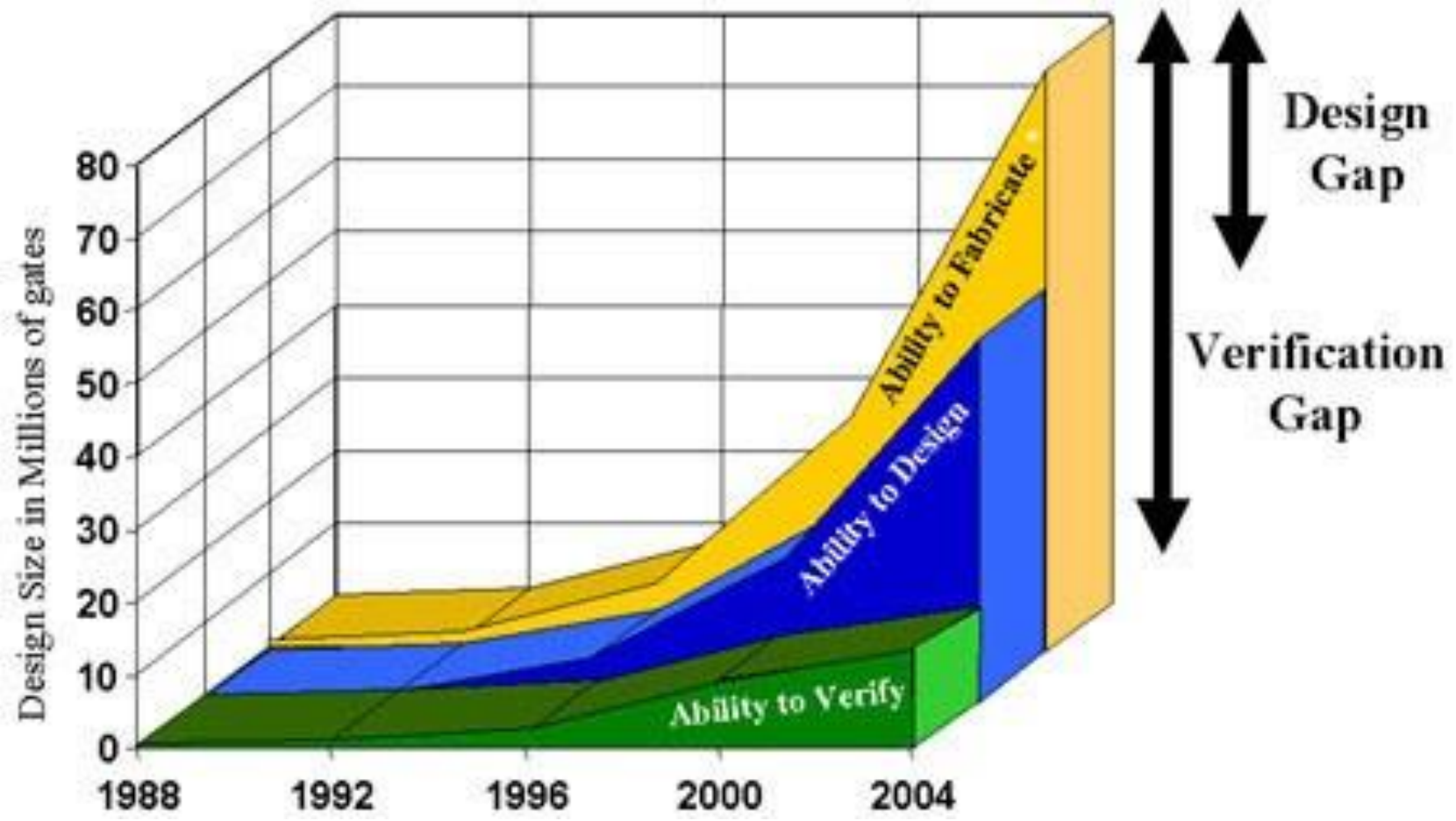
Design Gap


10G
1G
100M
10M
1M
100K
10K
1K
Logic Transistor Number
100M
10M
1M
100K
10K
1K
100
10
Transistor Number Per Person Month
SOC Design Complexity Increase
58% / Year
IP Use
Design
Gap
Productivity Increase
21%/Year
1980
1985
1990
1995
2000
2005
2010
Source : SEMATECH

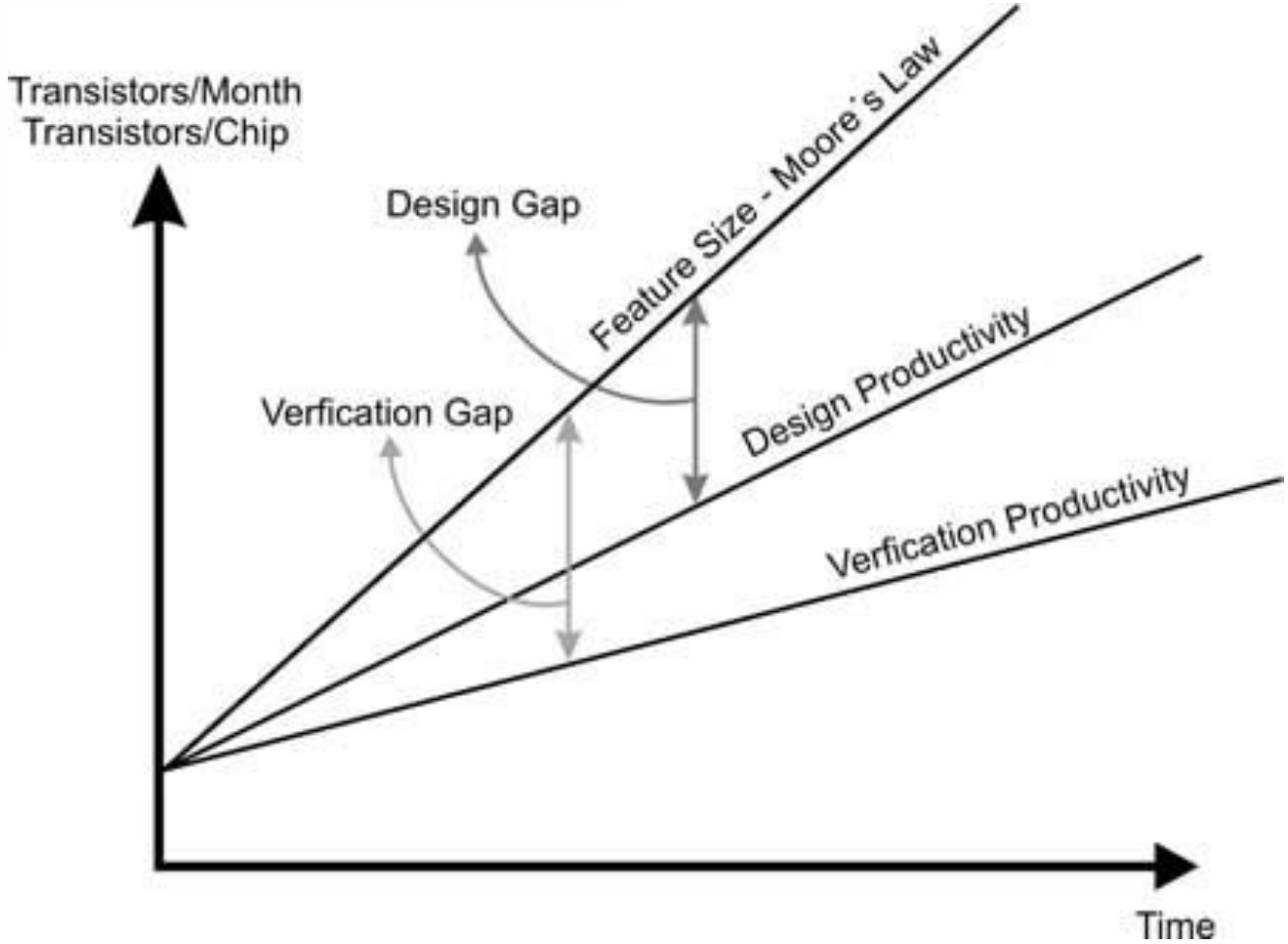
Design Gap

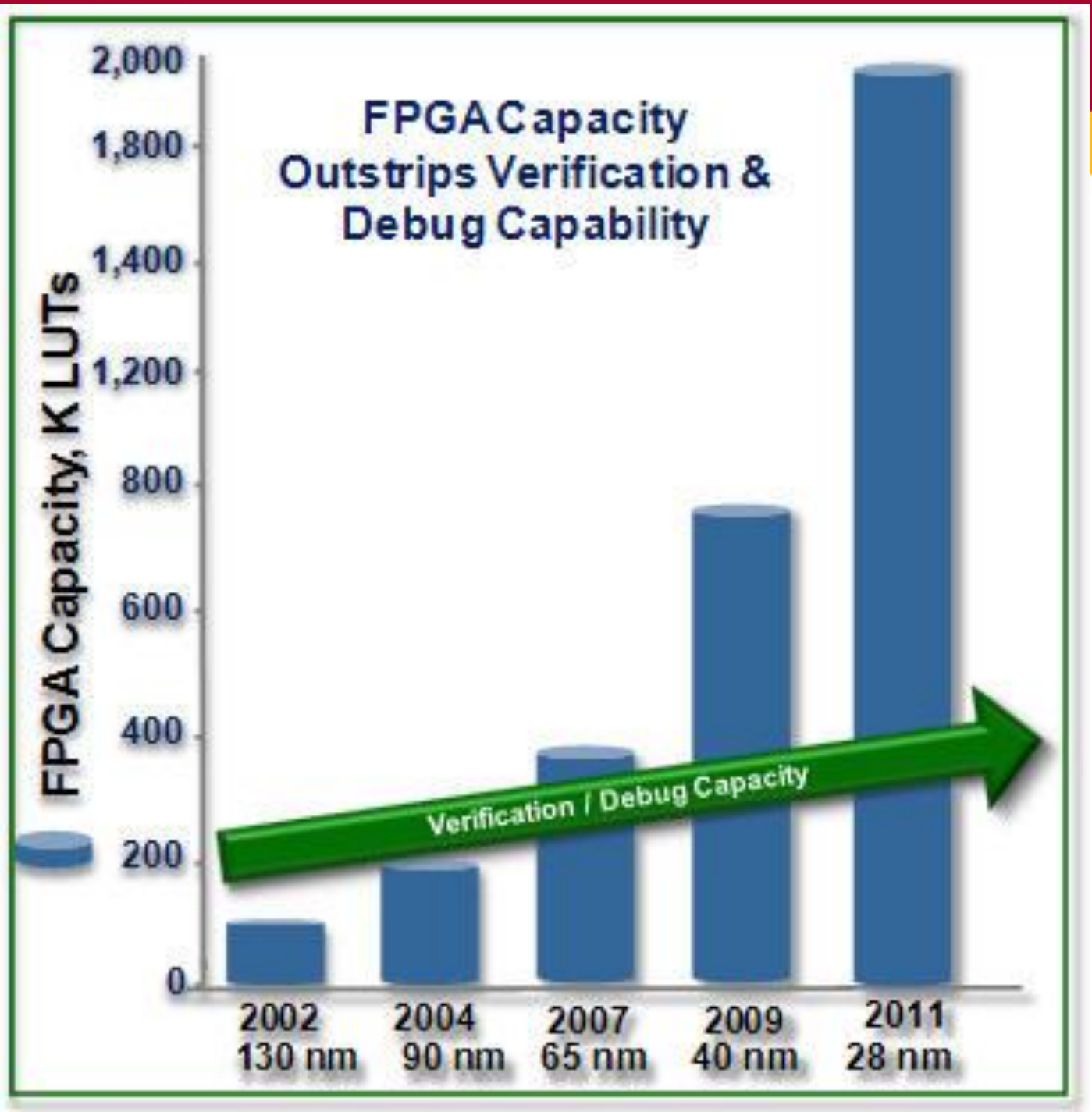
- Design capacity grows slower than fabrication capacity
- If a designer designs at 100 logic gates per day, and a chip can fit 10M puertas… we need 100K days to design the complete system : 500 engineers * 1 year!

Verification Gap



Verification Gap




FPGA Capacity
Outstrips Verification &
Debug Capability
FPGA Capacity, K LUTs
2,000
1,800
1,400
1,200
800
600
400
200
0
Verification / Debug Capacity
2002
130 nm
2004
90 nm
2007
65 nm
2009
40 nm
2011
28 nm

Design of complex digital systems

- Specifications
  - Describe clearly and unambiguously what the system must do
- Design
  - Implement the actual system
- Verification
  - Ensure the design complies with the specifications

Challenges in specification

- Specify unambiguously
- “Moving targets”: changes that the client asks during the project duration
- Increase of system complexity -> exponential increase of interactions between elements and failure modes
- The requirements document must be kept updated throughout the project duration

Challenges in design

- High performance requirements
  - High throughput
  - High bandwidth
  - High operation frequency
- Technical knowledge of the teams
- Specific protocols and FPGA primitives and IP cores
- Mixed Hardware/Software designs

Challenges in verification

- Functional verification: is the functionality correct?
  - For all possible use cases?
  - For all possible configurations?
- Coverage: did we test all our design?
  - All code?
  - All functionality?
- Can we do this in projects of exponentially increasing complexity?

Design Gap: solutions

- Use of embedded processors (soft processors / hard macros)
- Usage of IP cores
- Higher Level Synthesis (HLS)

Careful! More complex designs also require more **verification** efforts!

Verification Gap: solutions

- Verification metrics (when do I know I have finished verifying?)
- Verification methodologies (UVM, OSVVM, UVVM, pyUVM)
- Usage of Verification IP
- Usage of hardware accelerators

HAPS-54
Virtex-5 LX330
(May-2007)

Design methodologies

- FPGAs as Systems-on-Chip: Soft processors and hard macros
- Design with IP cores
- Higher Level Synthesis

Design methodologies

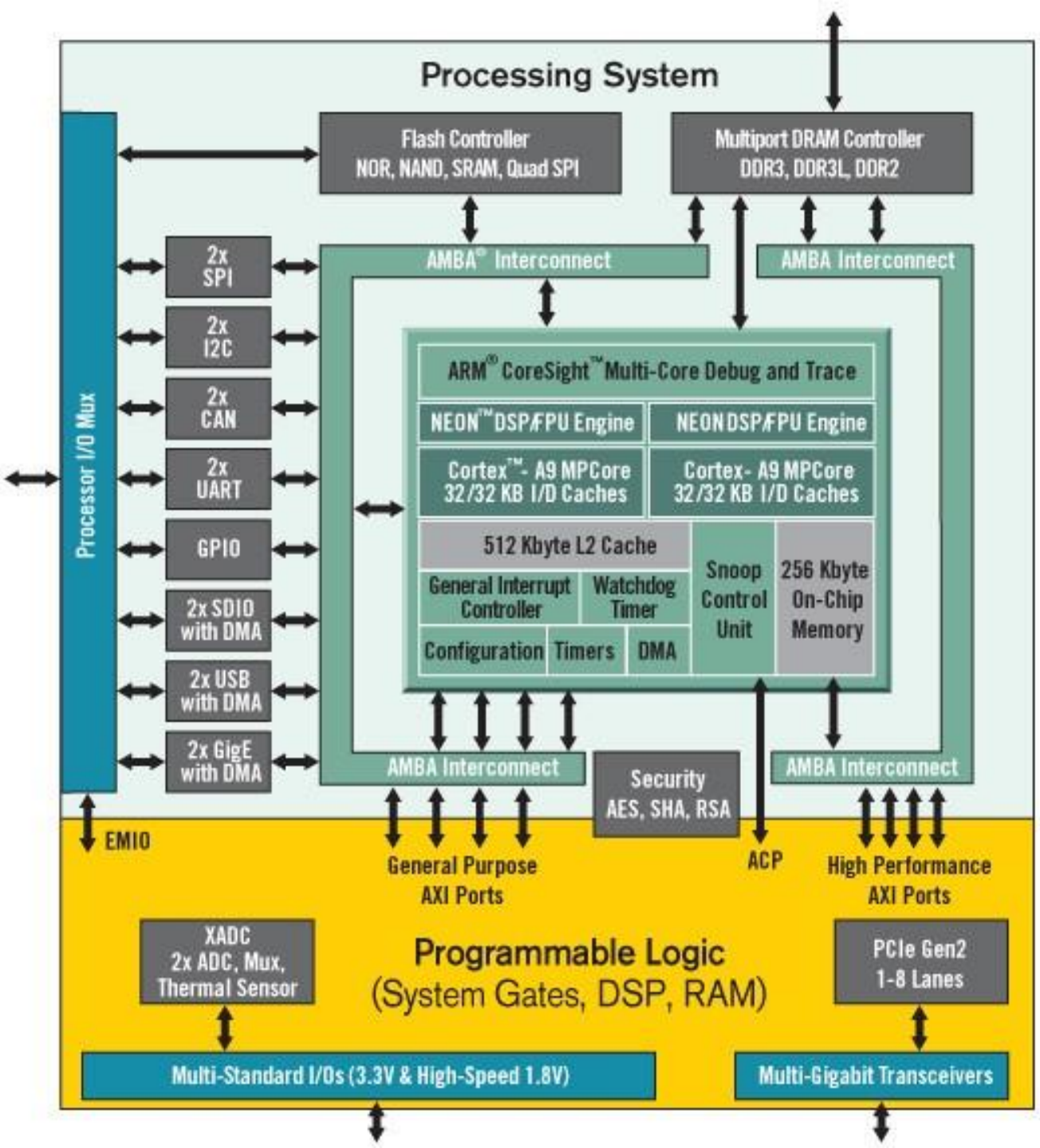
- FPGAs as Systems-on-Chip: Soft processors and hard macros
- Design with IP cores
- Higher Level Synthesis

FPGAs as Systems-on-Chip

- The evolution of microelectronics technologies (Moore's Law)
- Brings us to the following architecture of a modern FPGA:

Architecture of a modern FPGA

- Apart from IOBs, CLBs and routing resources:
- Embedded memories (Block RAMs)
- Digital Signal Processing (DSP) Blocks
- High-speed communications (Gigabit transceivers, PCIexpress, …)
- Microprocessors!



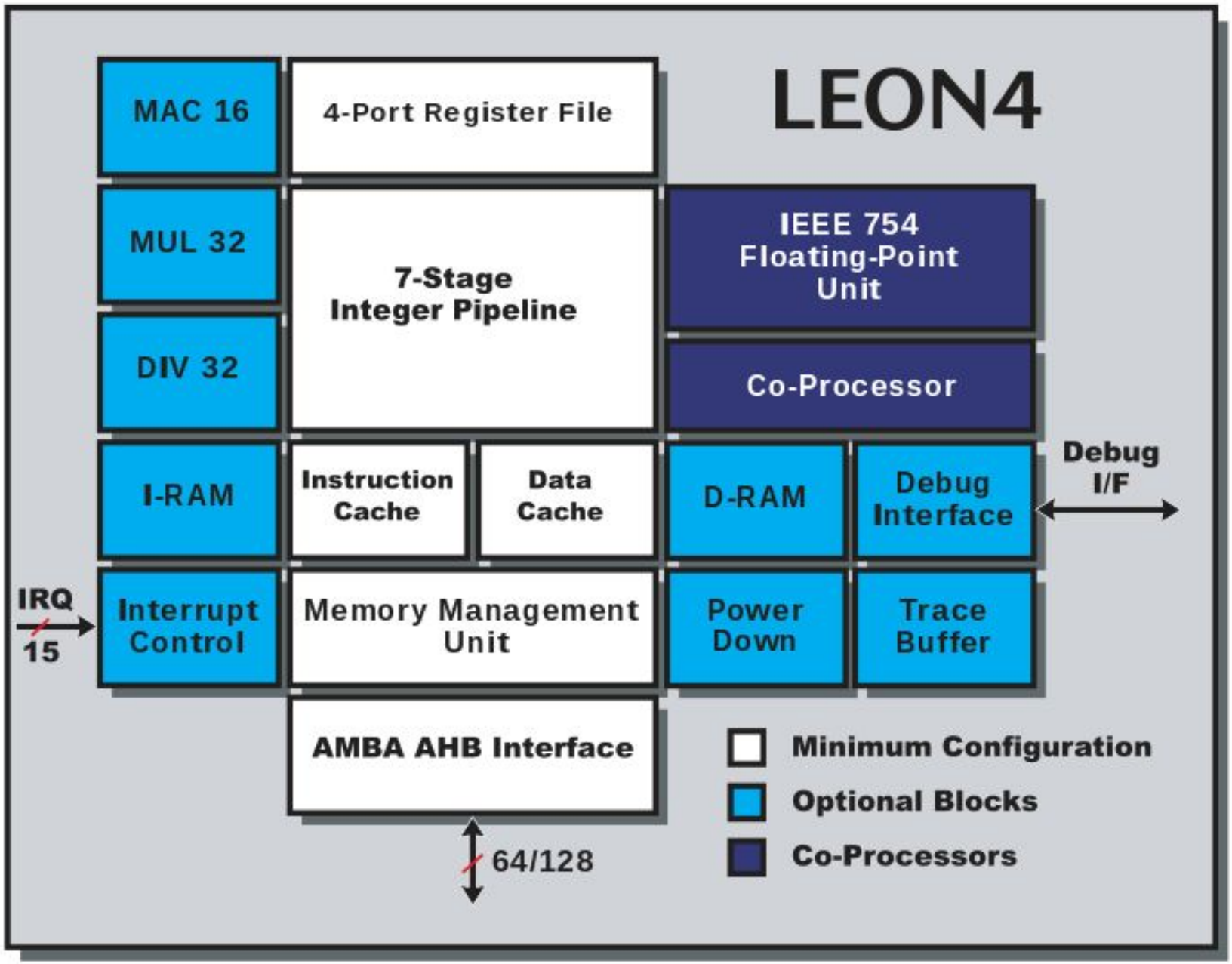
Zynq Family

Dual-core ARM integrated in Silicon

Soft processors

- We don’t need a next-generation FPGA to have an embedded microprocessor
- If we had a VHDL/Verilog description of an ALU, registers, progam counter, instruction decoder, … **= a microprocessor**
- Even if it's not implemented in silicon (like a BRAM), we could dedicate a portion of the FPGA to implementing a microprocessor.

Soft Processors



We don’t need to create them from scratch

Some soft processors:

- Microblaze (Xilinx)
- Nios II (Altera)
- Leon 4 (Aeroflex Gaisler)
- Plasma (Opencores)
- OpenSparc
- OpenRisc
- ...

Some problems:

- Configure peripherals and memory map
- Availability of a complete toolchain (compiler, linker, etc)
- Operating System or standalone program?
- Configure BRAMs with the executable
- Development of custom peripherals
- Availability of a simulation model
- Getting the microprocessor to boot!

Some advantages:

- Optimal design: HW and SW each handle what they do better
- Simplicity in communication of your HDL design with the outside world: USB, TCP/IP, …
- An update is not only changing the program: we can add new peripherals (for example, a timer)

Tips

- HW for parallel/concurrent task
- SW for sequential tasks
- Use HW for critical functionality: even if the SW hangs, the HW will continue to function
- Use SW for communications (TCP/IP): connect your VHDL to the outside world

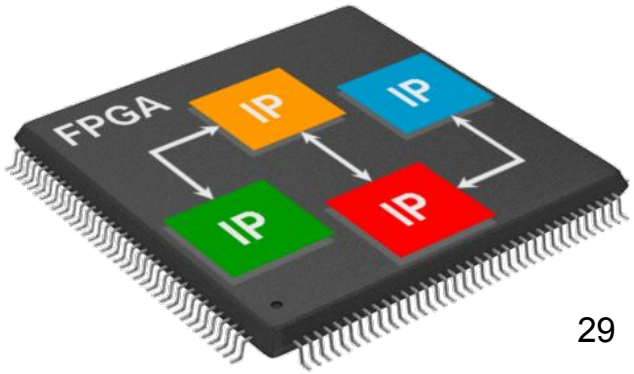
Design methodologies

Apart from mastering advanced VHDL concepts:

- FPGAs as System-on-Chip: Soft processors and hard macros
- Design with IP cores
- Higher Level Synthesis

Design with IP cores

- High effort required to develop a complex system
- Resue already-tested modules
- Reduction of design effort
- The main problem is the **integration** of the modules:
  - Interfaces
  - Quality of documentation
  - Configuration of the IP cores



IP (Intellectual Property) Cores must be:

- Reusables
- Configurables
- Simulables with industry-standard simulators
- With standards-based interfaces
- Verified to a high level of confidence
- Completely documented

Buses for embedded processors

Specific IP cores have specific interfaces. A specific softcore (soft processor) will support one or more buses:

- PLB, AXI4 (MicroBlaze)
- Wishbone ('standard' in Opencores and Free/Open-Source designs)
- AMBA (Leon)

Buses for embedded processors

Third-party IP cores that we integrated, and the ones we develop ourselves should interface with the chosen bus

Although if we don't need speed we could use a GPIO connection

This way: design of complex Systems-On-Chip using standards-based IP cores

Tips

- A good integrator speeds up the process as much as, or even more than, a good designer
- Manufactures and providers have their own ‘quirks’ which must be understood
- It is extremely advisable to simulate basic cases to become familiar with the blocks
- Read a lot and try things out little by little!

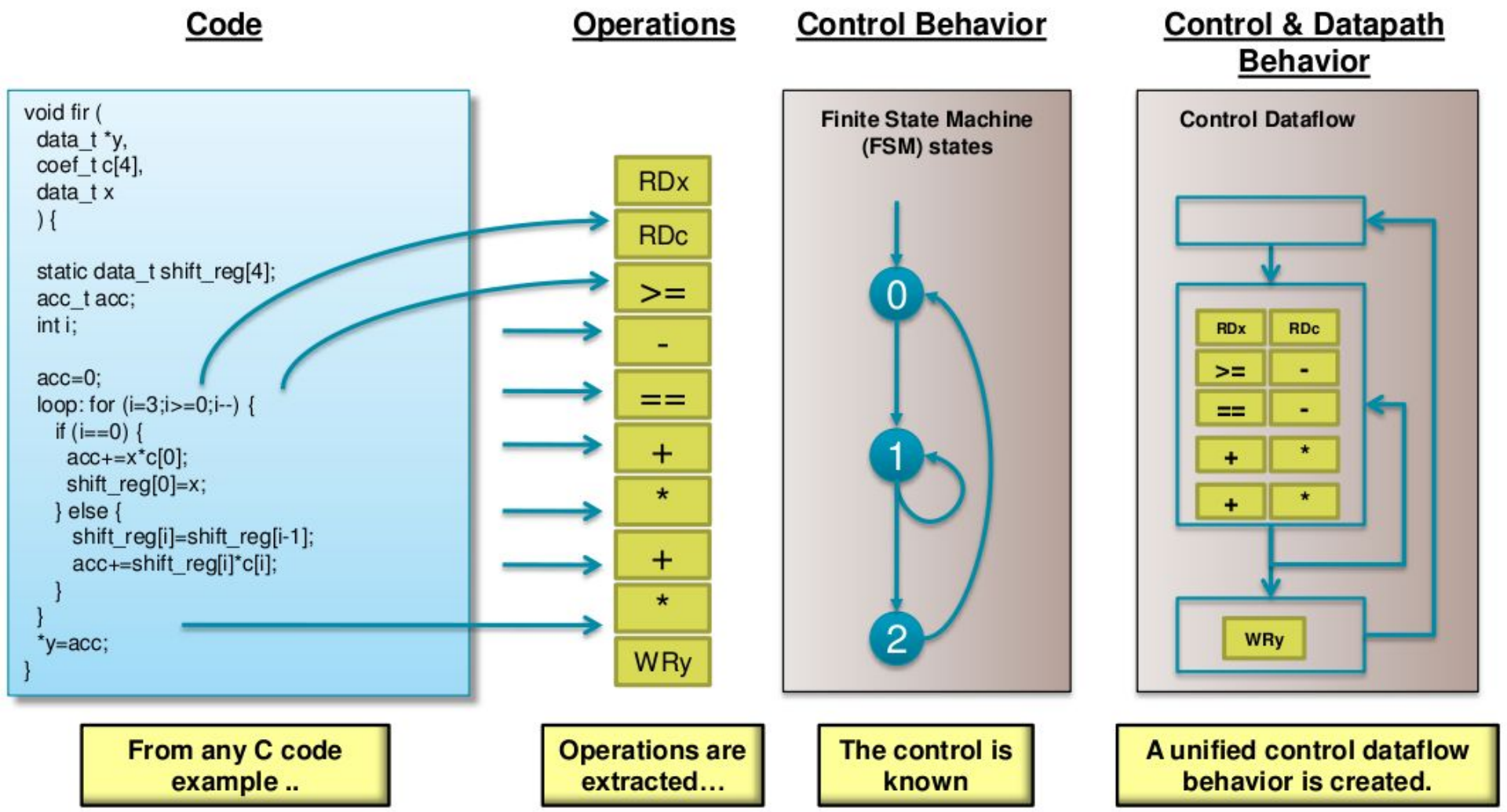
Design methodologies

- FPGAs as System-on-Chip: Soft processors and hard macros
- Design with IP cores
- Higher Level Synthesis

Higher Level Synthesis

- VHDL is already ‘high level’, ya es ‘alto nivel’, however, there are currently tools (in different stages of maturity) that translate from higher level code (C, SystemC, python, etc) to hardware

Higher Level Synthesis



Source: Xilinx

Higher Level Synthesis

```
void fir (
  data_t *y,
  coef_t c[4],
  data_t x
  ) {

  static data_t shift_reg[4];
  acc_t acc;
  int i;

  acc=0;
  loop: for (i=3;i>=0;i--) {
    if (i==0) {
      acc+=x*c[0];
      shift_reg[0]=x;
    } else {
      shift_reg[i]=shift_reg[i-1];
      acc+=shift_reg[i] * c[i];
    }
  }
  *y=acc;
}
```

Functions: represent the design's hierarchy

Top-level inputs: the arguments of the top-level function determine the ports of the generate hardware

Types: data types have influence in area and performance

Arrays: can influence the device's I/O and become bottlenecks

Operators: Operators in the C code are implemented in hardware and can be shared by different parts of the implementation

Higher Level Synthesis

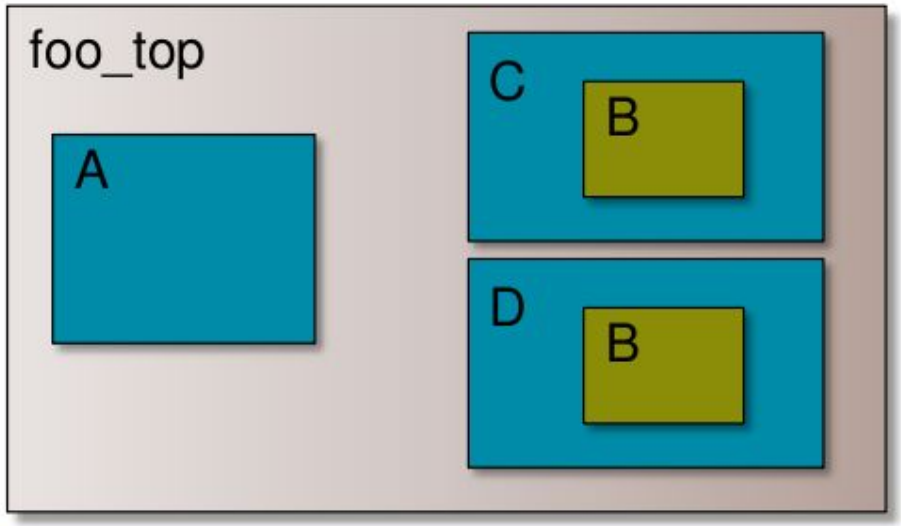
Source Code

```
void A() { ..body A..}
void B() { ..body B..}
void C() {
        B();
}
void D() {
        B();
}


void foo_top() {
        A(…);
        C(…);
        D(…)
}
```

my_code.c

RTL hierarchy



Each block can be shared like any other component provided it's not in use at the same time

Tips

- Just like when using a compiler, you have greater ease of design but less control over the final result
- The correct use of synthesis directives is fundamental to ensure an efficient implementation
- 2x area, ½ speed with respect to VHDL implementations

References

- Xilinx Large FPGA Methodology Guide (UG872)
- MicroBlaze Processor Reference Guide (UG081, v11.0 - EDK 12.1)
- EDK Concepts, Tools, and Techniques: A Hands-On Guide to Effective Embedded System Design (UG683 - EDK 12.1)

References (II)

- Xilinx Vivado Design Suite User Guide: High Level Synthesis
- Michael Keating, Pierre Bricaud, Reuse Methodology Manual for System-on-a-Chip Designs