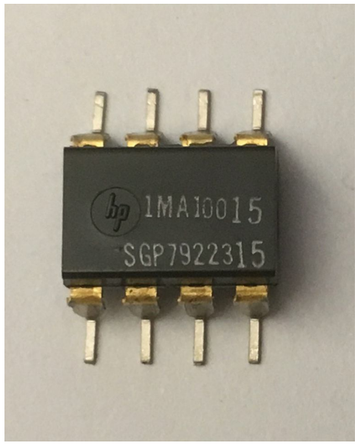
EECS151 : Introduction to Digital Design and ICs

Lecture 3 – Verilog I

Bora Nikolić and Sophia Shao

The Berkeley Remix Podcast, Season 4, Episode 2, “Berkeley Lightning: A Public University’s Role in the Rise of Silicon Valley”



IC chip from Hewlett Packard 34C Calculator, 1979-83. Some of the calculator’s revolutionary features were designed by UC Berkeley computer scientist William M. Kahan

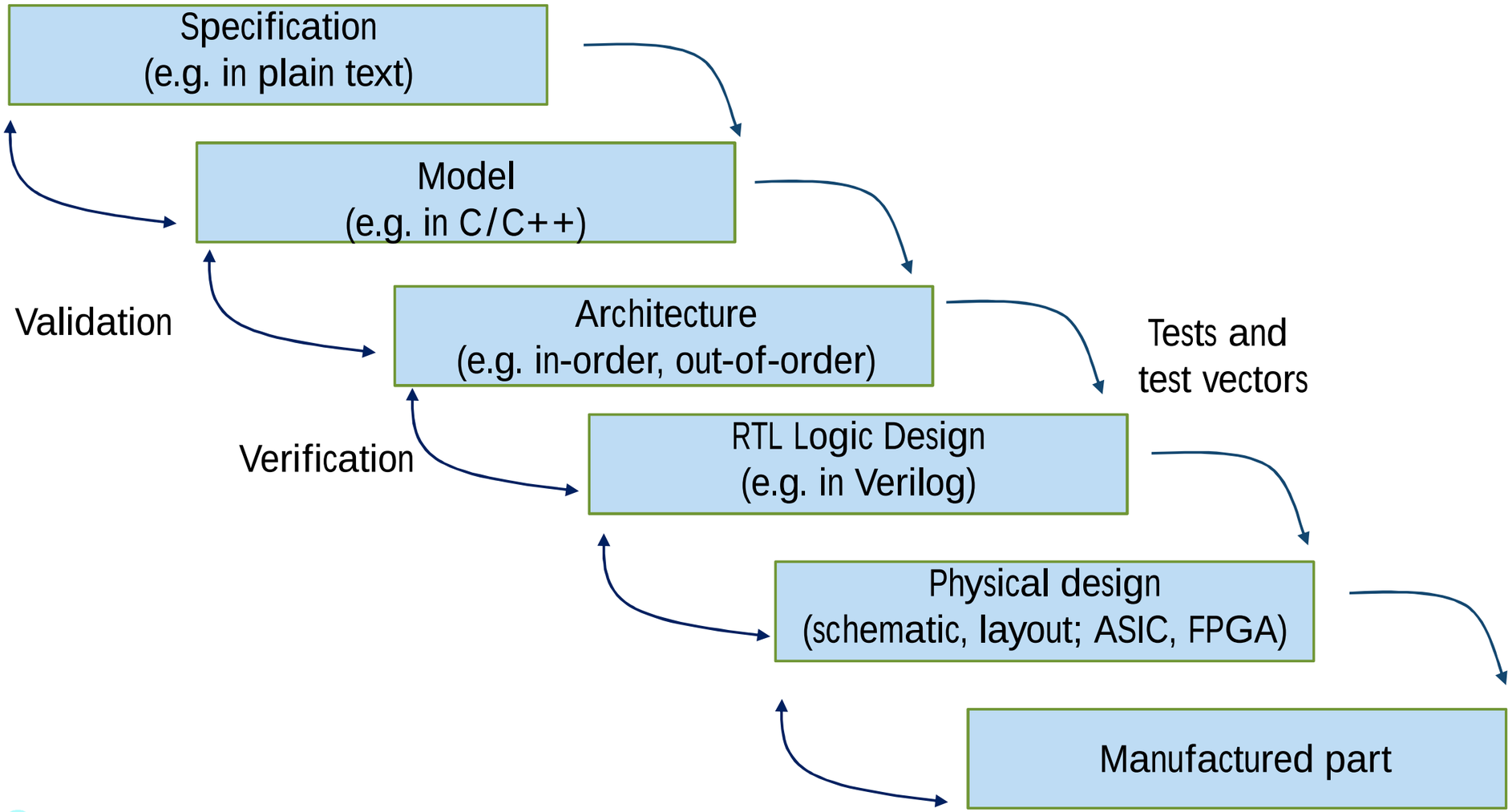
Review

- Performance, power and cost set the design targets
- A product design iterates through specification, modeling, architectural design, RTL design, physical design and fabrication phases
- In combinatorial logic, outputs only depend on current inputs
- In sequential logic blocks, outputs depend on current inputs and the state
- We have seen several abstractions:
  - Digital logic abstraction of 0s and 1s
  - Logic gate abstraction
  - Static timing
  - RTL abstraction

Options for Designing Digital Systems

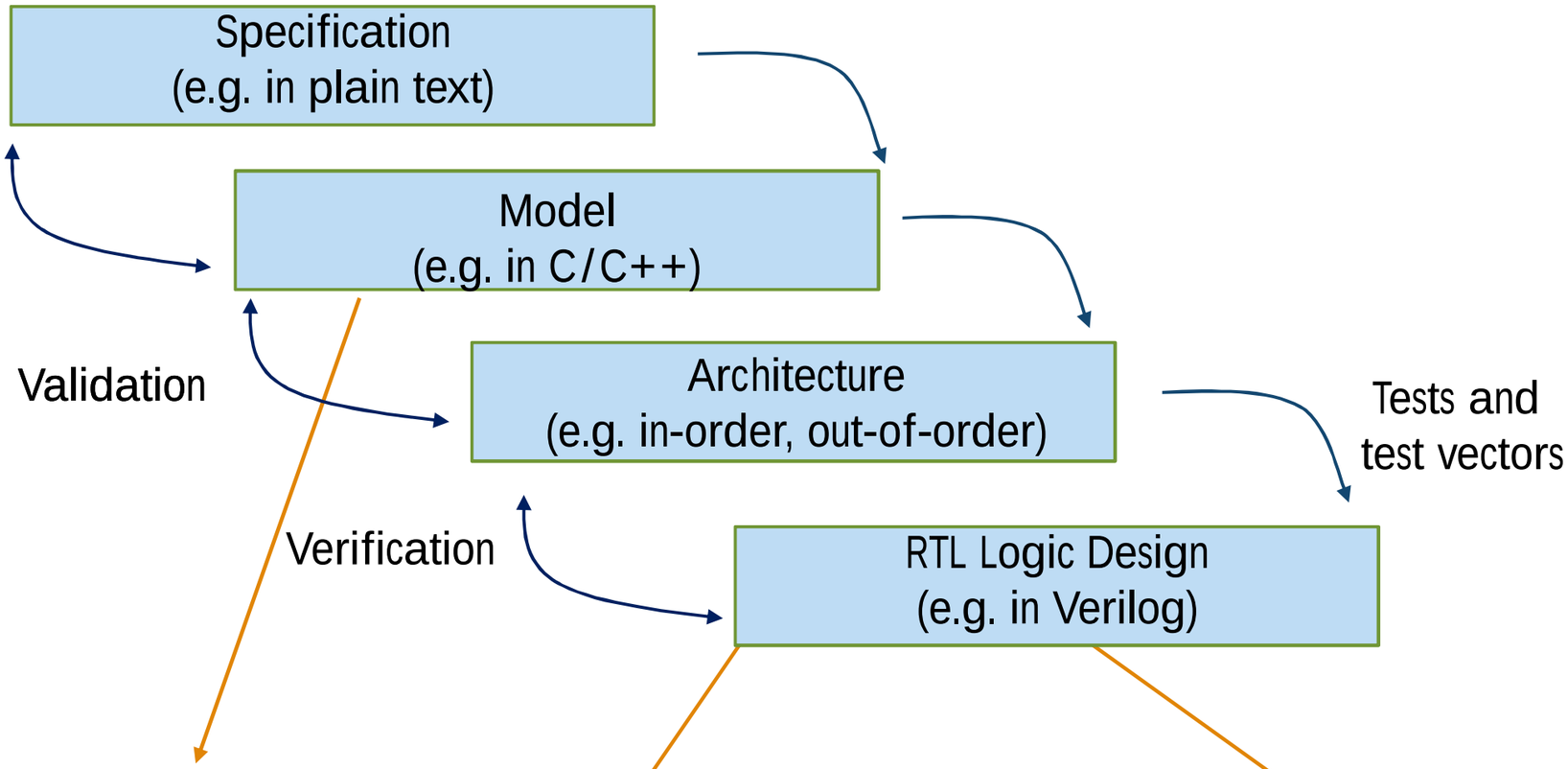
Review: Design Process

- Design through layers of abstractions



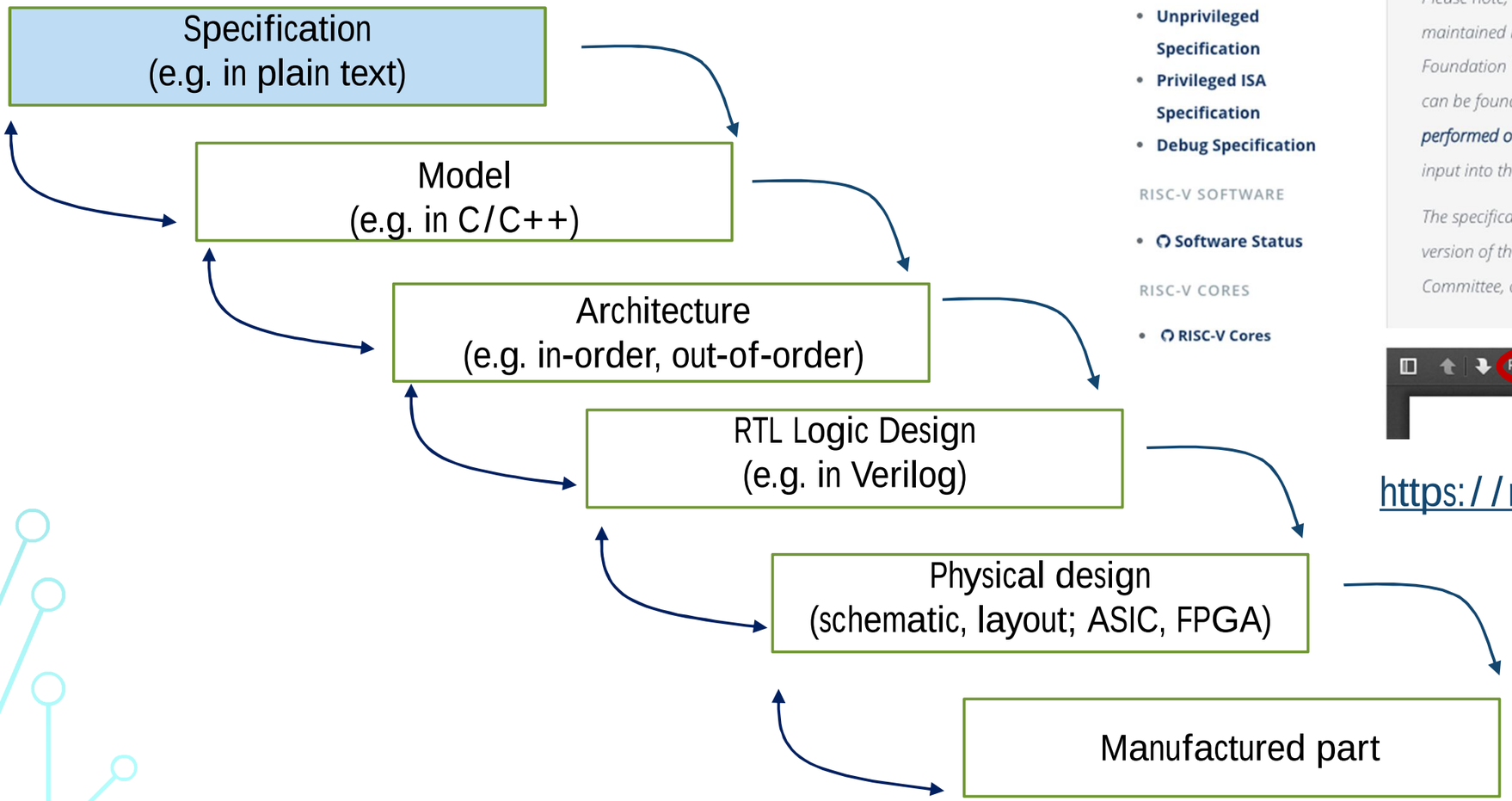
Implementing Physical Systems

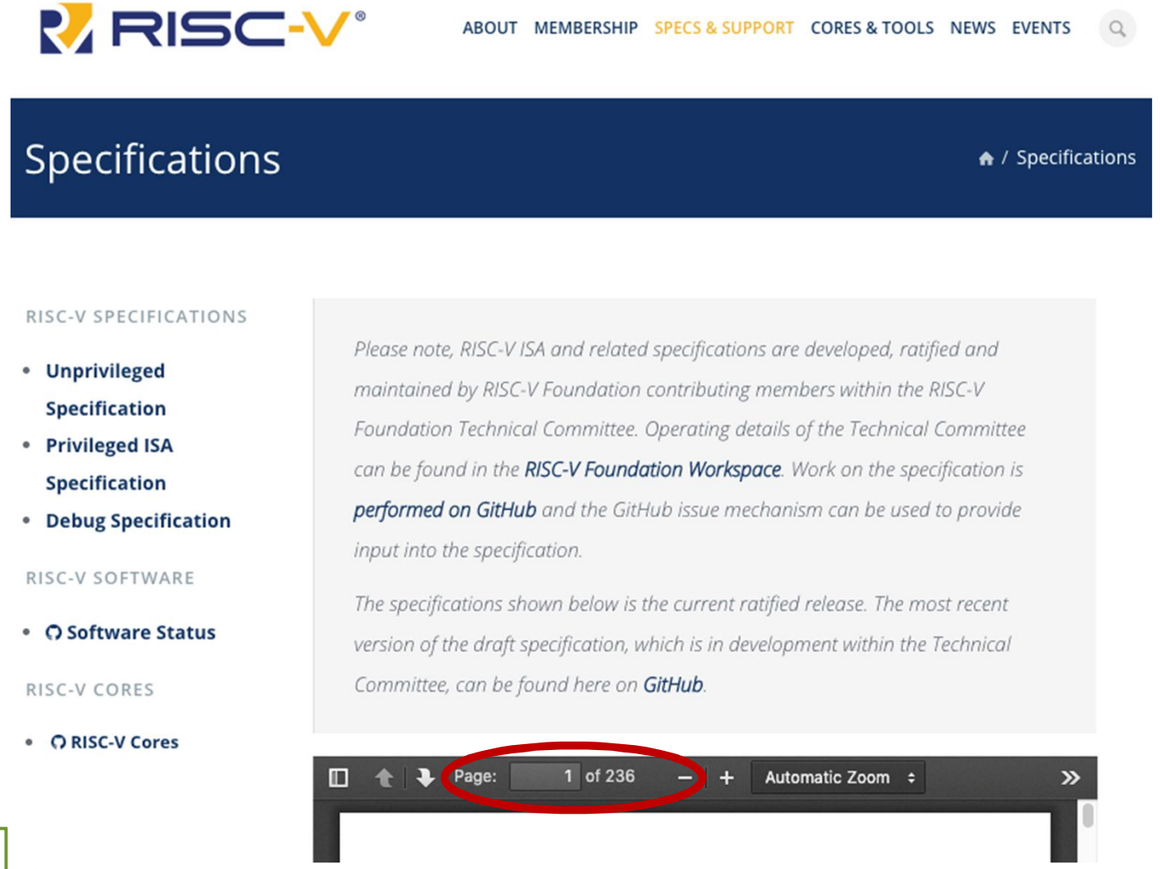
- Design through layers of abstractions



Example: RISC-V Design Process

- Design through layers of abstractions

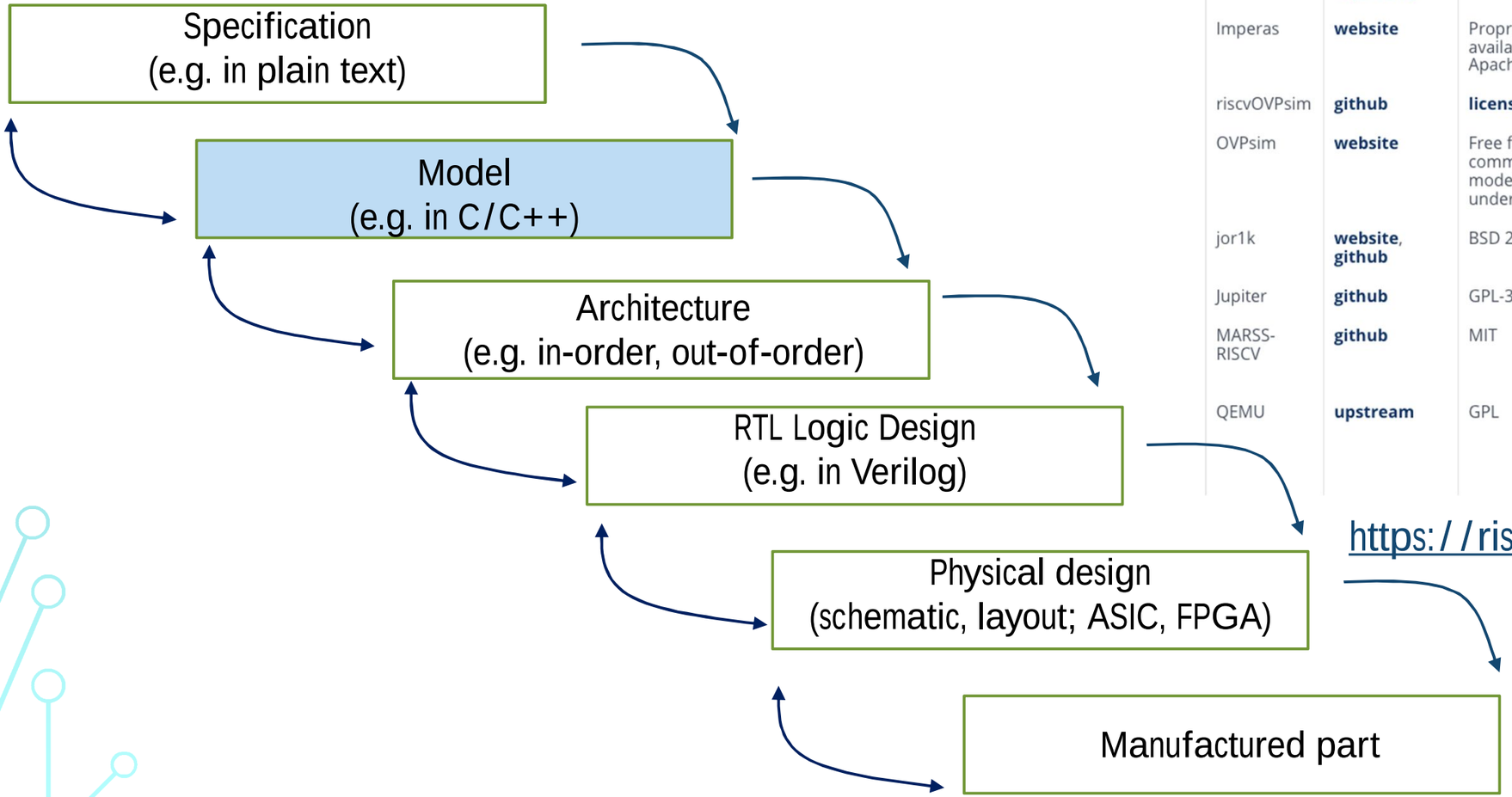




https://riscv.org/specifications/

Example: RISC-V Design Process

- Design through layers of abstractions



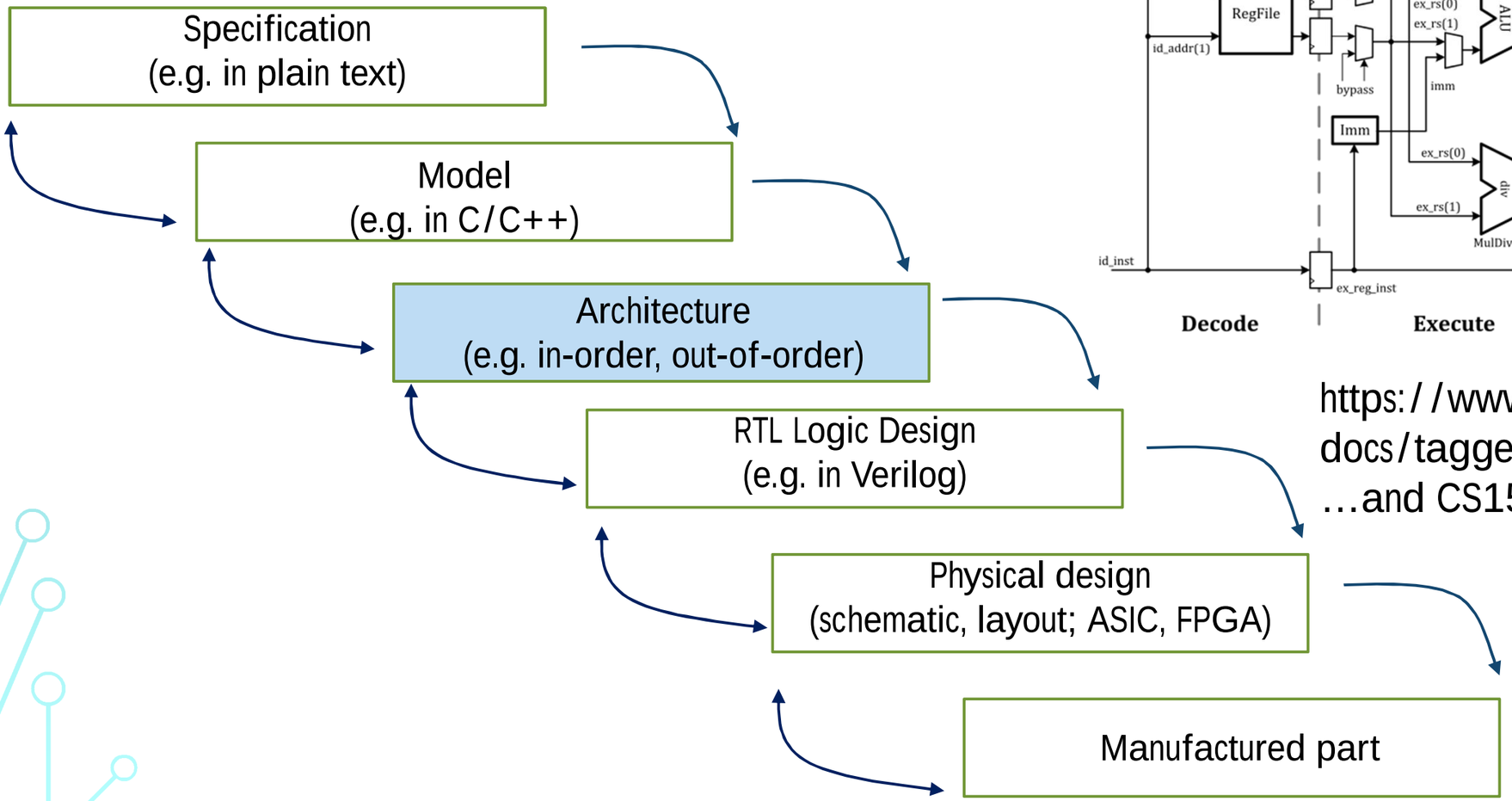
Simulators

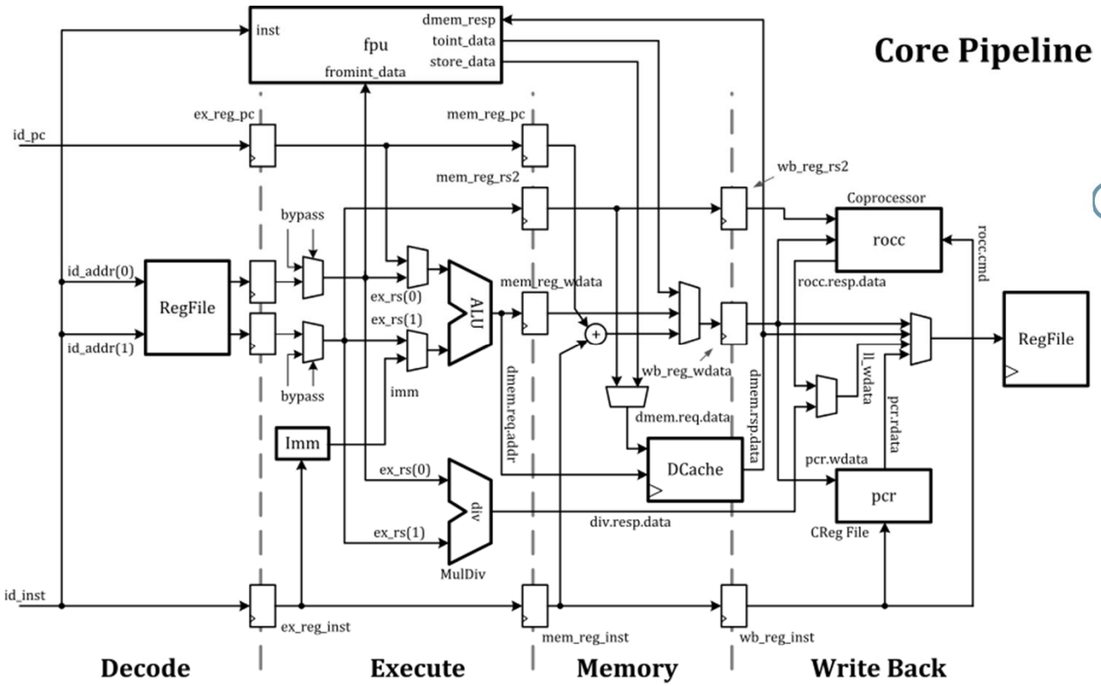
Name	Links	License	Maintainers
DBT-RISE-RISCV	**github**	BSD-3-Clause	**MINRES Technologies**
FireSim	**website**, **mailing list**, **github**, **ISCA 2018 Paper**	BSD	Sagar Karandikar, Howard Mao, Donggyu Kim, David Biancolin, Alon Amid, **Berkeley Architecture Research**
gem5	**SW-dev thread**, **repository**	BSD-style	Alec Roelke (University of Virginia)
Imperas	**website**	Proprietary, models available under Apache 2.0	**Imperas**
riscvOVPsim	**github**	**license**	**Imperas**
OVPsim	**website**	Free for non commercial use, models available under Apache 2.0	**Imperas**
jor1k	**website**, **github**	BSD 2-Clause	Sebastian Macke
Jupiter	**github**	GPL-3.0	Andrés Castellanos
MARSS-RISCV	**github**	MIT	Gaurav N Kothari, Parikshit P Sarnaik, Gokturk Yuksek (State University of New York at Binghamton)
QEMU	**upstream**	GPL	Sagar Karandikar (University of California, Berkeley), Bastian Koppelmann (University of Paderborn), Alex Suykov, Stefan O'Rear and Michael Clark (SiFive)

https://riscv.org/software-status/#simulators

Example: RISC-V Design Process

- Design through layers of abstractions

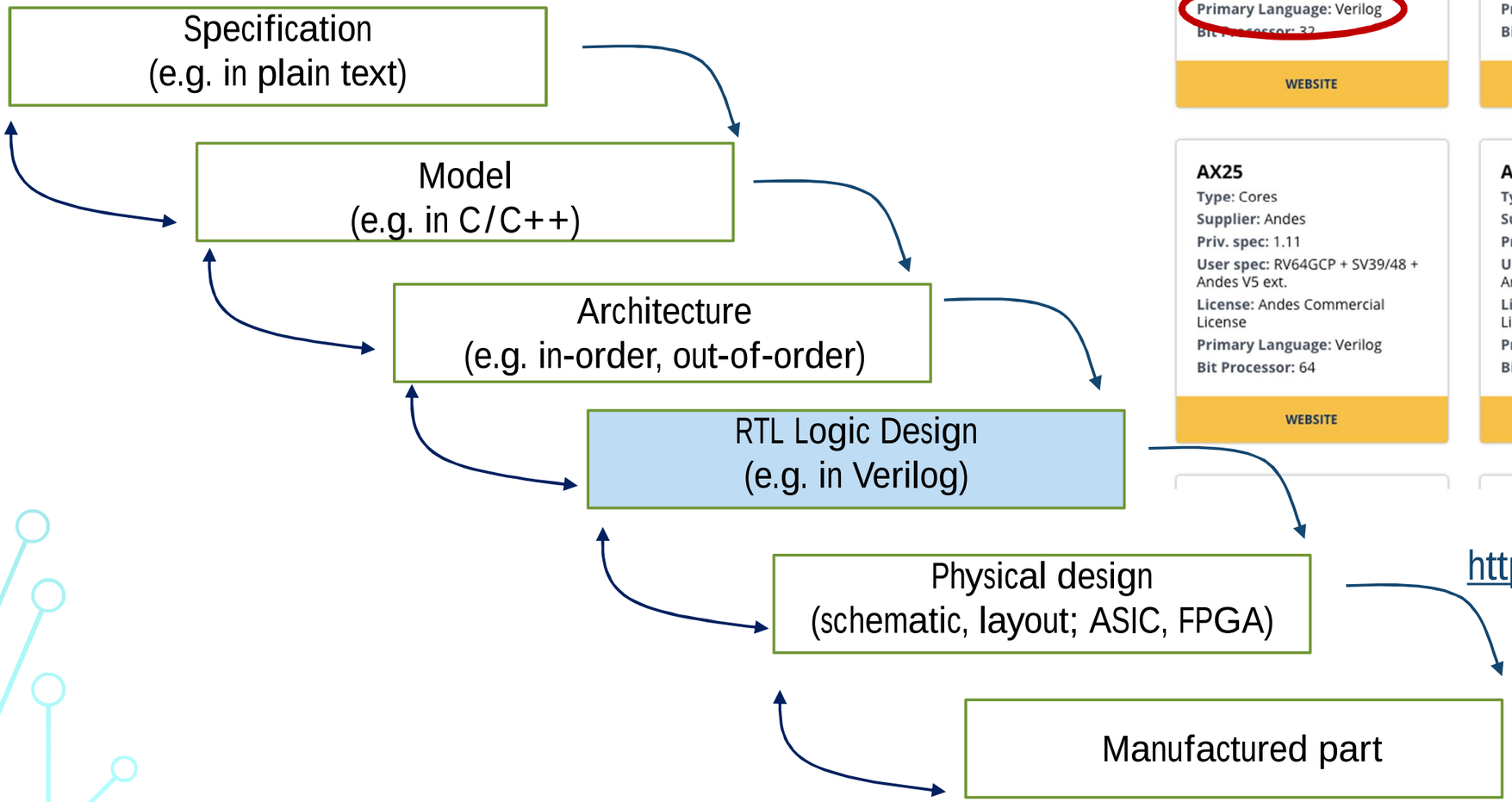


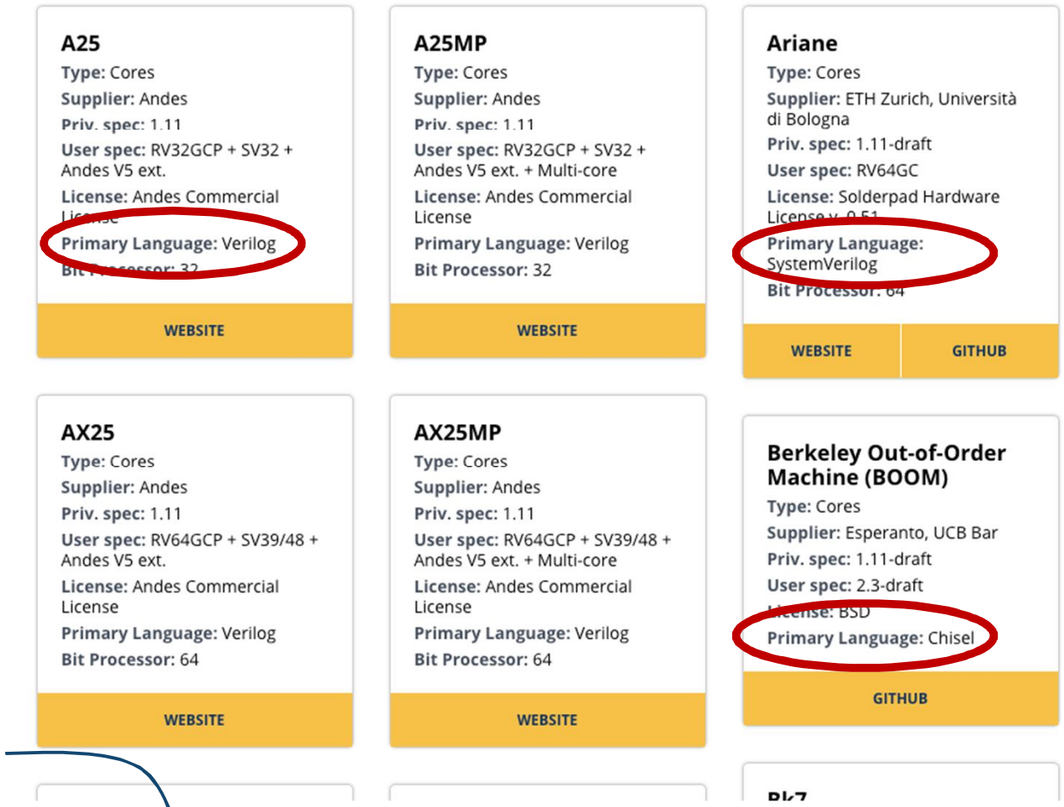


https://www.lowrisc.org/docs/tagged-memory-v0.1/rocket-core/
…and CS152

Example: RISC-V Design Process

- Design through layers of abstractions

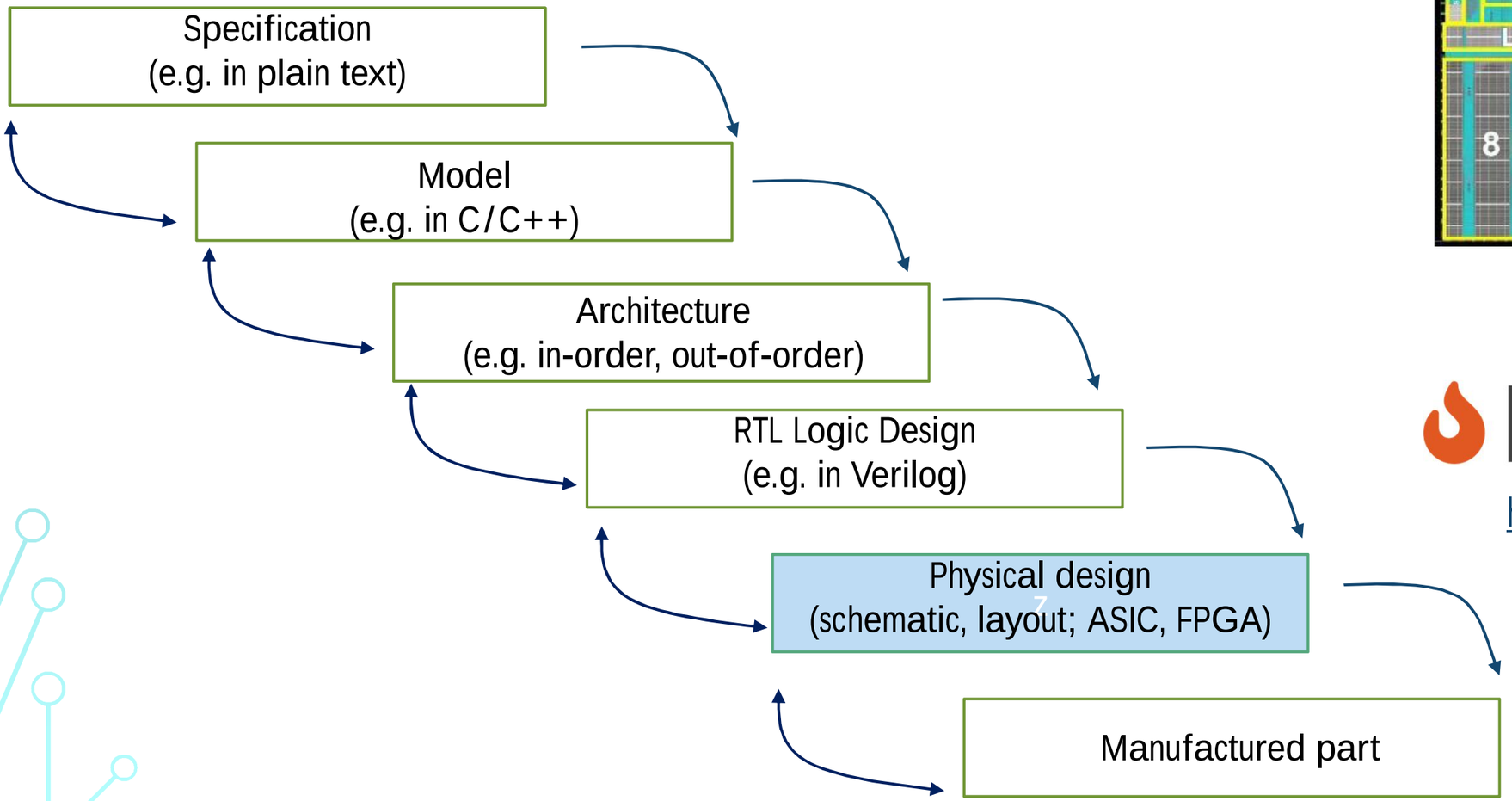


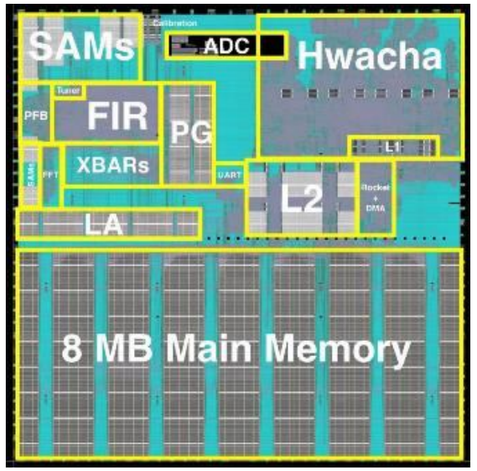


https://riscv.org/risc-v-cores/

Example: RISC-V Design Process

- Design through layers of abstractions



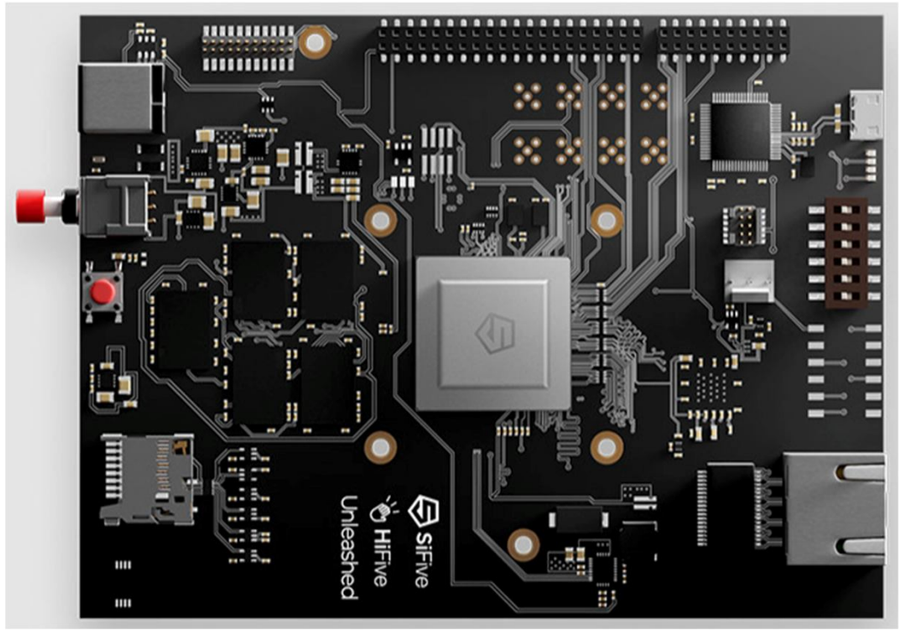


FireSim

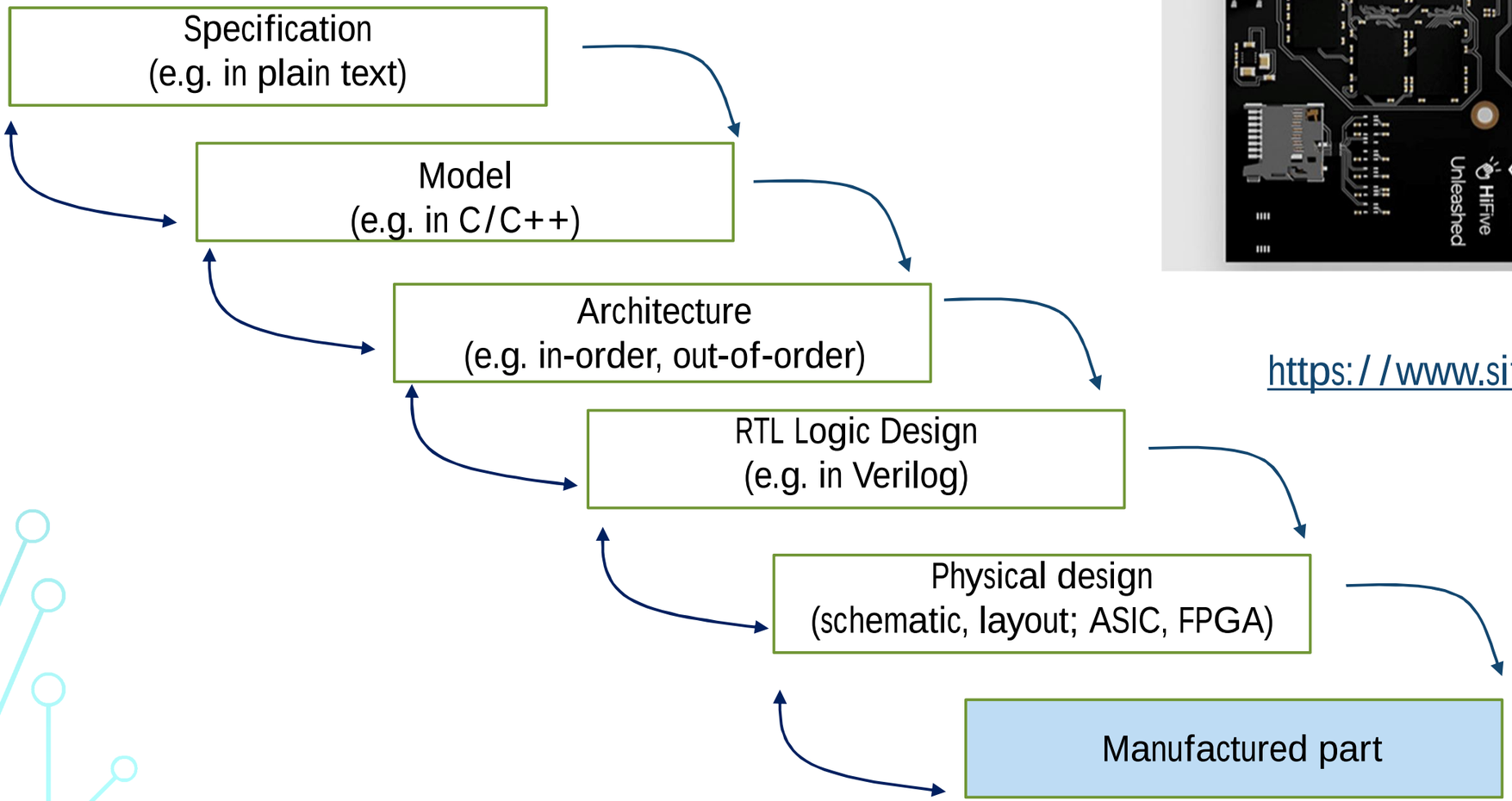
https://fires.im/

Example: RISC-V Design Process

- Design through layers of abstractions



https://www.sifive.com/boards/hifive-unleashed

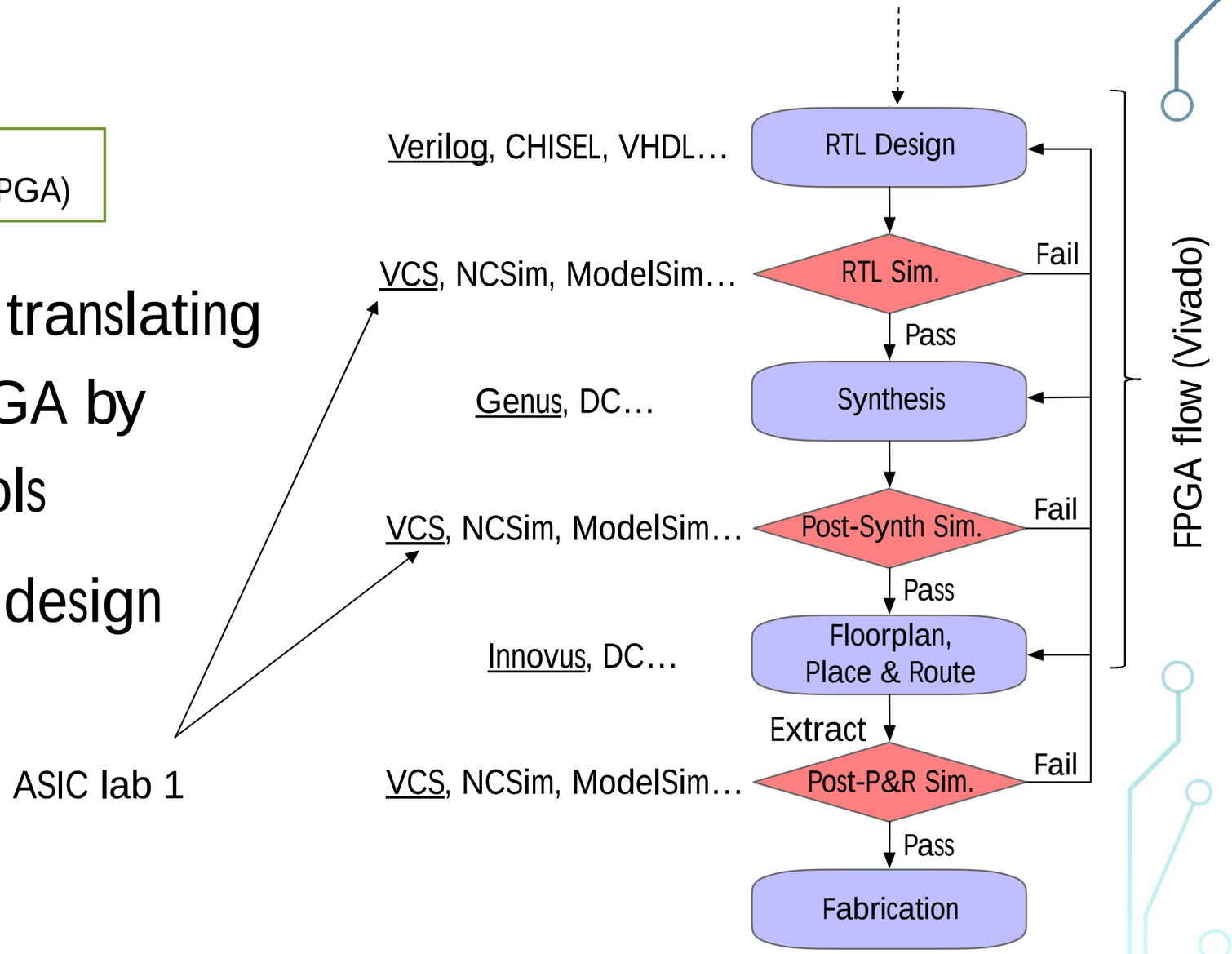


RTL → Physical Design (ASIC)

RTL Logic Design
(e.g. in Verilog)

Physical design
(schematic, layout; ASIC, FPGA)

- Labs focus on a process of translating RTL to physical ASIC or FPGA by using industry-standard tools
- Project explores the entire design stack



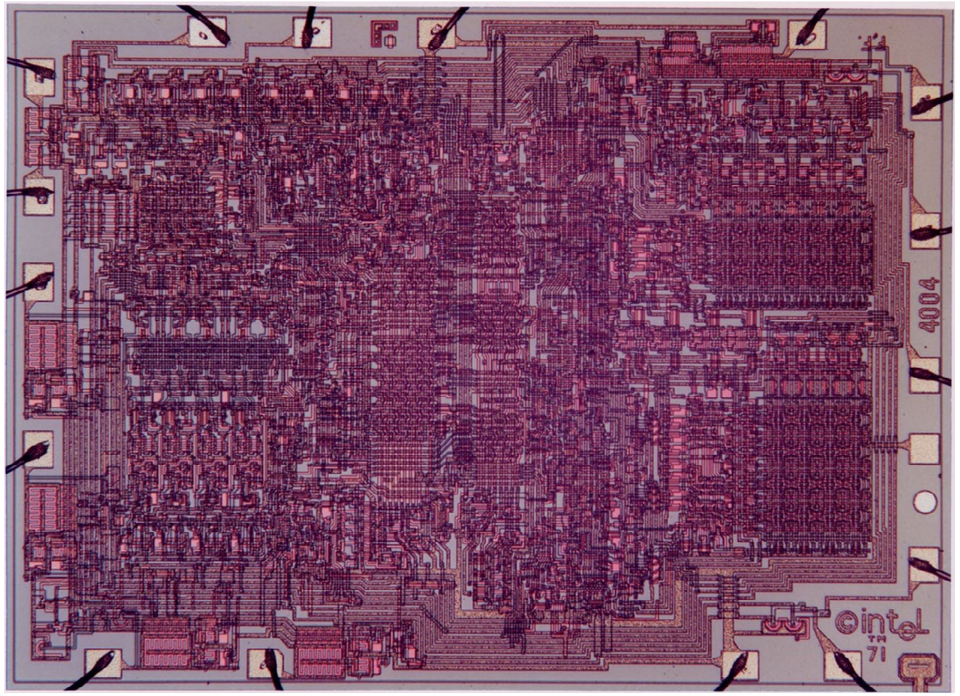
Administrivia

- You need to be enrolled in one lab in addition to the lecture
- Hw 1 is out, due on Friday; Hw 2 out this week
- Lab 3 starts on Thursday

How Did We Get Here?

IC Design in the 70's and early 80's

- Circuit design, layout, and processing tightly linked.
- Logic design and layout was all done by-hand
- Chip design was typically done by vertically integrated companies, who designed and fabricated their own chips
  - Fabs weren't that expensive back then



The Intel 4004 microprocessor, which was introduced in 1971. The 4004 contained 2300 transistors and performed 60,000 calculations per second. Courtesy: Intel.

Federico Faggin,
Ted Hoff,
Stan Mazor

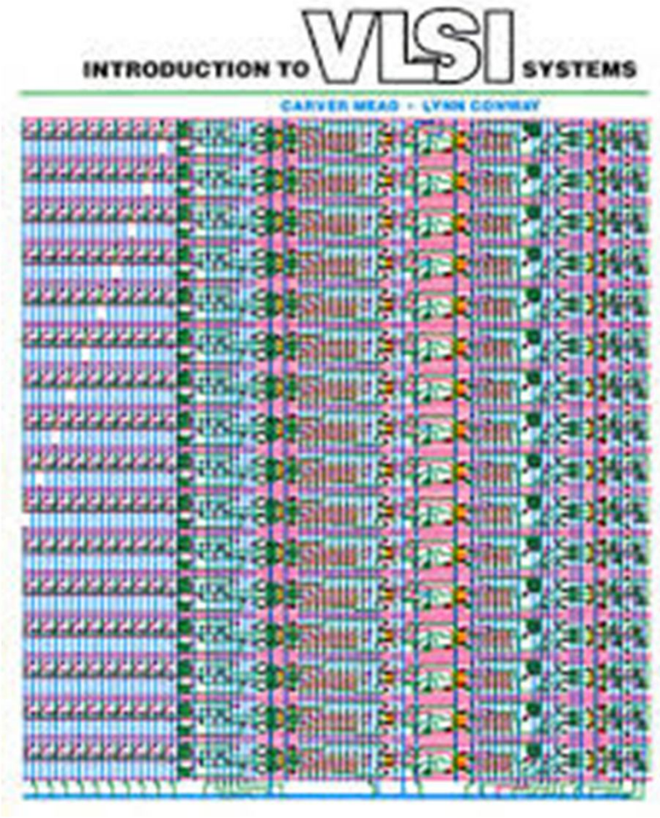
Introduced to help sell memory chips!

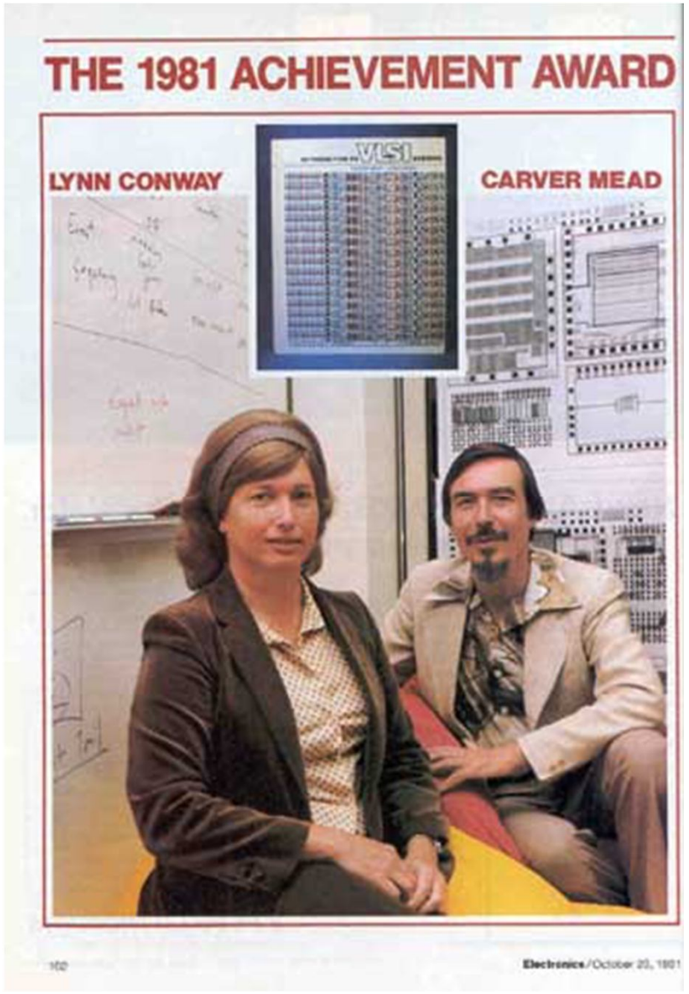
Early Design Practice

- Initially, designs were represented by hand drawings. Then masks where made by transferring drawings to rubylith.
  - Base layer of heavy transparent dimensionally stable Mylar. A thin film of deep red cellophane-like material covers the base layer. Patterns formed by cutting (often by hand) the transparent covering.

- Later transition to an electronic format (CIF, GDS) meant: Layouts easily be stored and transmitted. Written to tape and transferred to manufacturer (tape-out).Transmitted over the network (new idea back then). Software could automatically check for layout errors. Generated from a program - **huge idea.**

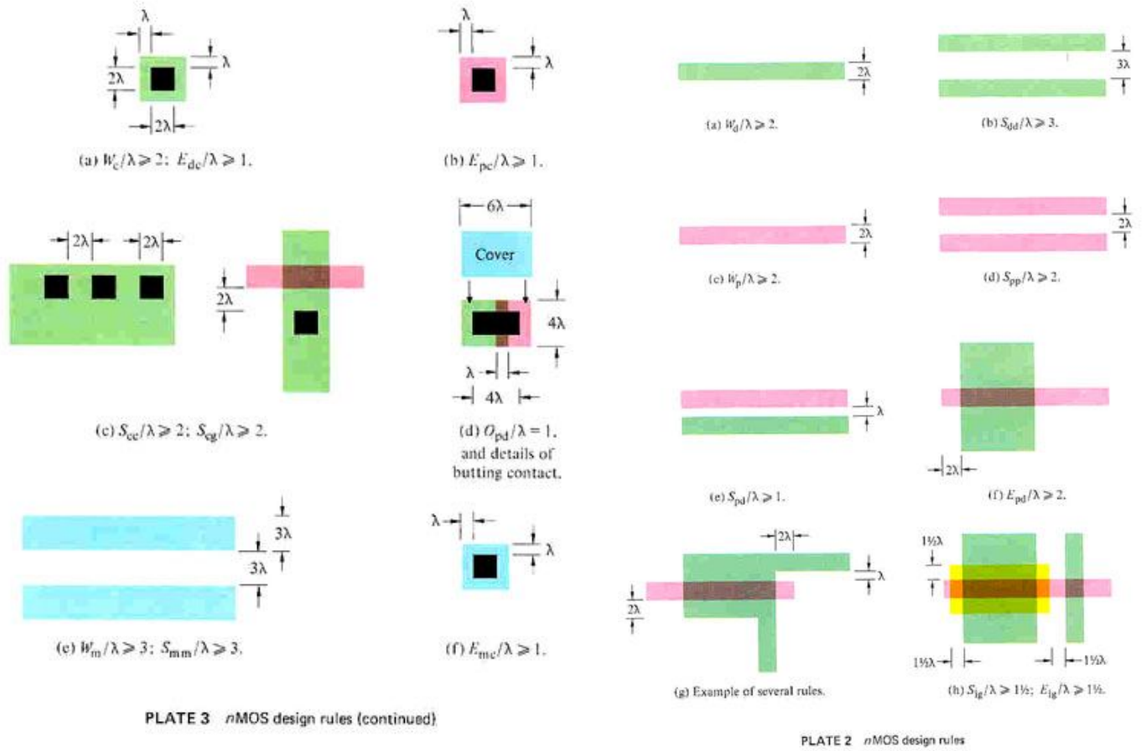
The Start of the IC Design Revolution





Geometric Design Rules

- Early on, to generate the mask information for fabrication, the designer needed intimate knowledge of the manufacturing process. Even once this knowledge was distilled to a set of "Geometric Design Rules", this set of rules was voluminous.
- Academics (C. Mead and others) came up with a much simplified set of design rules (single page description).
  - Sufficiently small set that designers could memorize. Sufficiently abstract to allow process engineers to shrink the process and preserve existing layouts. Process resolution becomes a "parameter", λ.



Key Development: Silicon Foundries

- Separate the designer from the fabricator: Modeled after the printing industry. (Very few authors actually own and run printing presses!)
- Simple standard geometric design rules where the key: these form the “contract” between the designer and manufacturer.
- Designer sends the layout, foundry manufactures the chip and send back.

- A scalable model for the industry: IC fab is expensive and complex. Amortizes the expense over many designers. Designers and companies not held back by need to develop and maintain large expensive factories. “Fabless” semiconductor companies - lots of these and very few foundries.

TSMC, Global Foundries, UMC, Samsung, SMIC, …

Computer Aided Design (1)

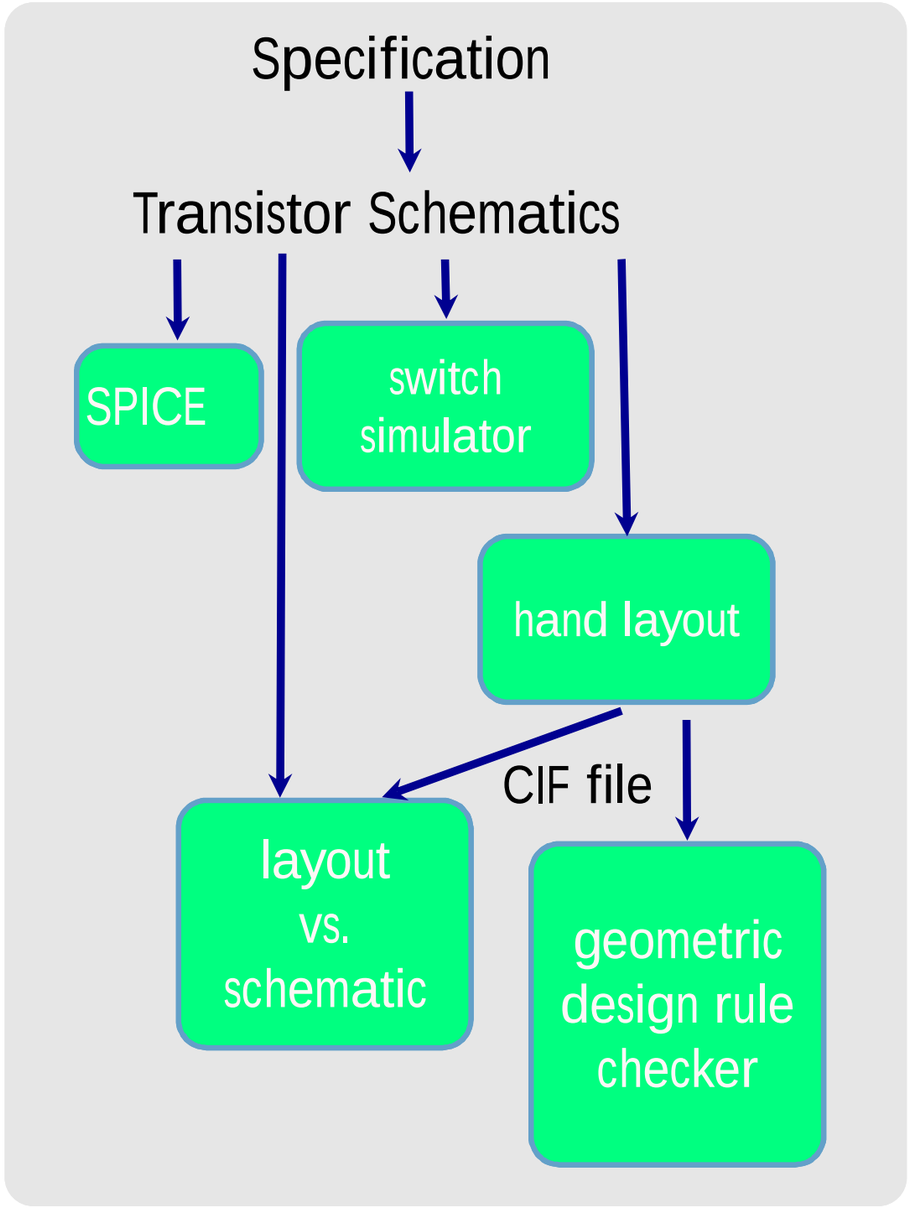
Several advances lead to the development of interactive tools for generating layout:

- Computer based layout representation (CIF, GDS).
- Advances in computer graphics (Ivan Sutherland) and display devices.
- Personal "workstation" (Xerox Alto - Chuck Thacker). "Back room" computers didn't have the necessary bandwidth to the display.
- Berkeley version - MAGIC

Early ’80’s Design Methodology and Flow

Schematic + Full-Custom Layout

- SPICE for timing,
- Switch-level simulation for overall functionality,
- Hand layout,
- No power analysis,
- Layout verified with geometric design rule checker (DRC) and later also layout versus schematic (LVS) checkers



Computer Aided Design (2)

- For some time after CIF was invented: Layout was generated by hand, then typed in as a CIF file with a text editor.
- Layout compilers
  - Soon some designers started embedding CIF primitives in conventional programming languages: LISP, pascal, fortran, C.
  - This allows designers to write programs that generated layout. Such programs could be parameterized:

```
define GENERATE_RAM(rows, columns) {
  for I from 1 to rows
   for J from 1 to columns
     (GENERATE_BITCELL)}
GENERATE_RAM(128, 32);
```

  - Lead to circuit/layout generation from higher level descriptions.
- Eventually, Cadence and Synopsys (formed out of Berkeley) built tools to generate layout from HDLs.

Implementation Alternatives

Implementation Alternative Summary

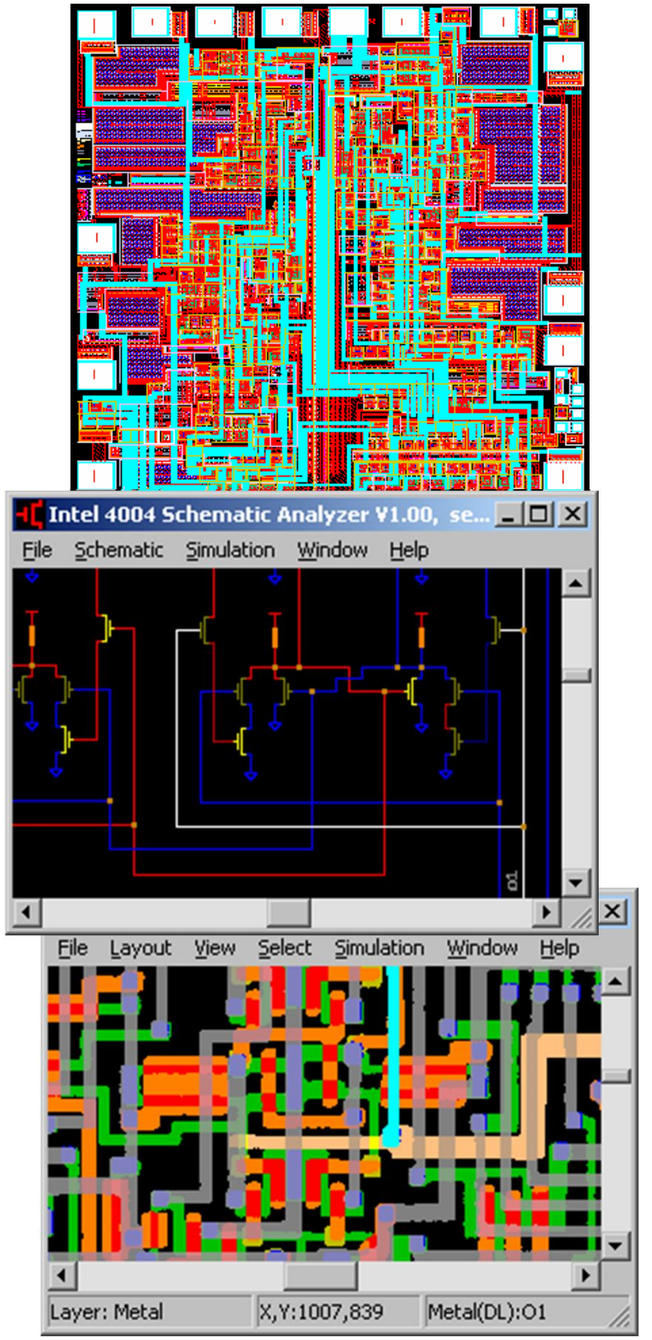
Full-custom:	Every transistors layout hand-drawn and optimized.
Standard-cell:	Logic gates and “macros” automatically placed and routed.
Gate-array (structured ASIC):	Partially prefabricated wafers with arrays of transistors customized with metal layers or vias.
FPGA:	Prefabricated chips that can be customized “in the field”
Microprocessor:	Instruction set interpreter customized through software.
Domain-specific processor:	Special instruction set interpreters (ex: DSP, NP, GPU).

These days, “ASIC” almost always means standard-cell design.

Full-Custom

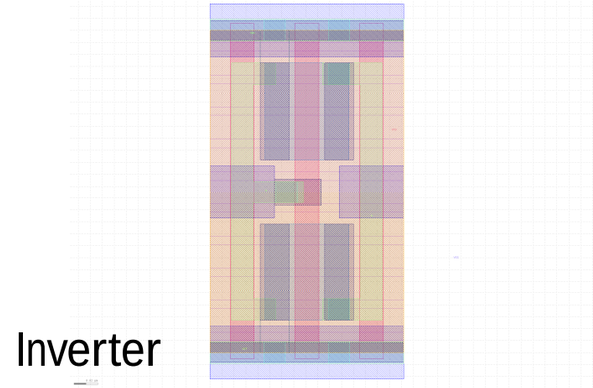
- Circuit styles and transistors are custom sized and drawn to optimize performance and power.
- High NRE (non-recurring engineering) costs
  - Time-consuming layout iterations

- Common today for **analog design**.
- In digital world – for **memories** and **arrays**.

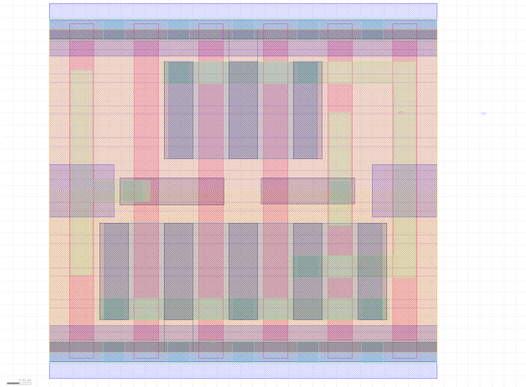


Standard Cells

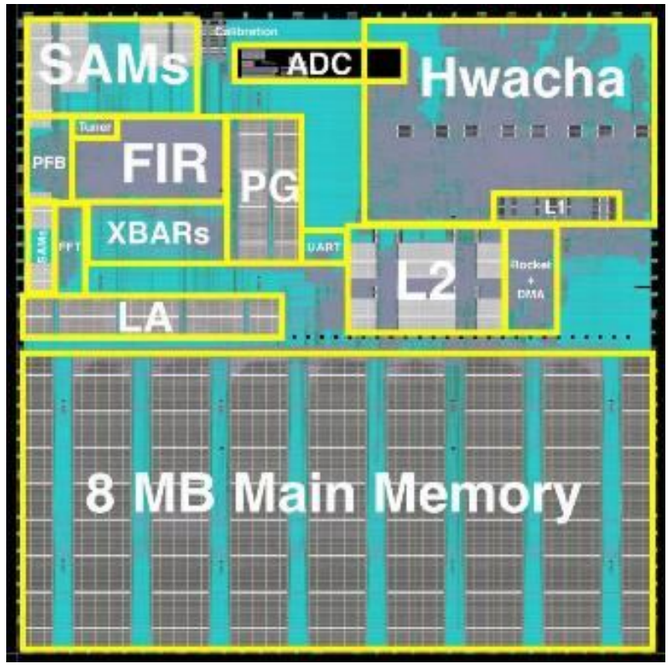
- Library of logic gates (1000-2000)
  - Combinational: NANDs, NORs, … Sequential: Flip-flops, latches, …
- Each cell comes complete with:
  - layout, schematic, symbol
  - Logic (Verilog), timing, power models.
- Chip layout is automatic, reducing NREs (usually no hand-layout).
- Memories, I/O and analog components complete the ASIC



Inverter



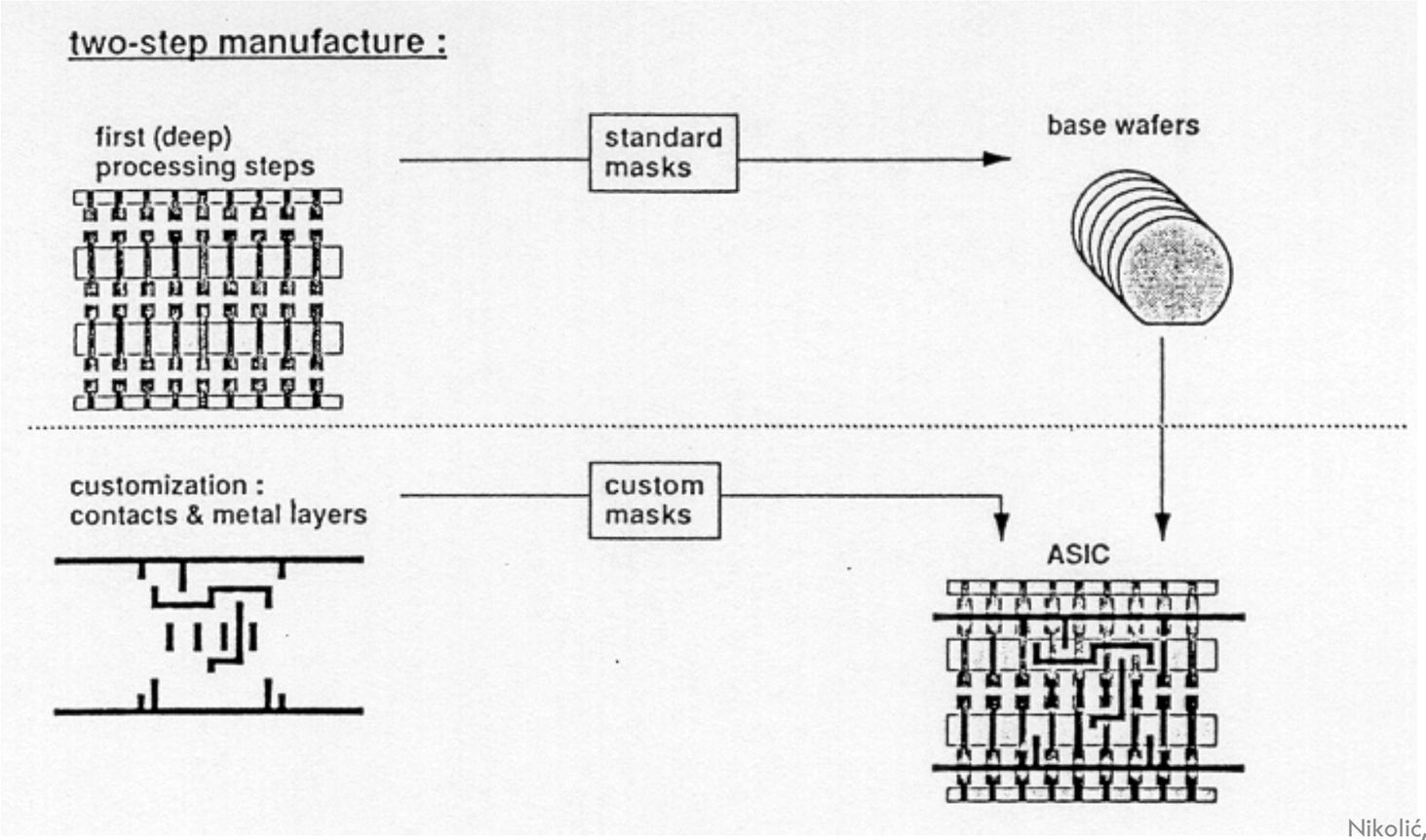
NAND2



ASAP7 – 7nm library from Arizona State University

Gate Array

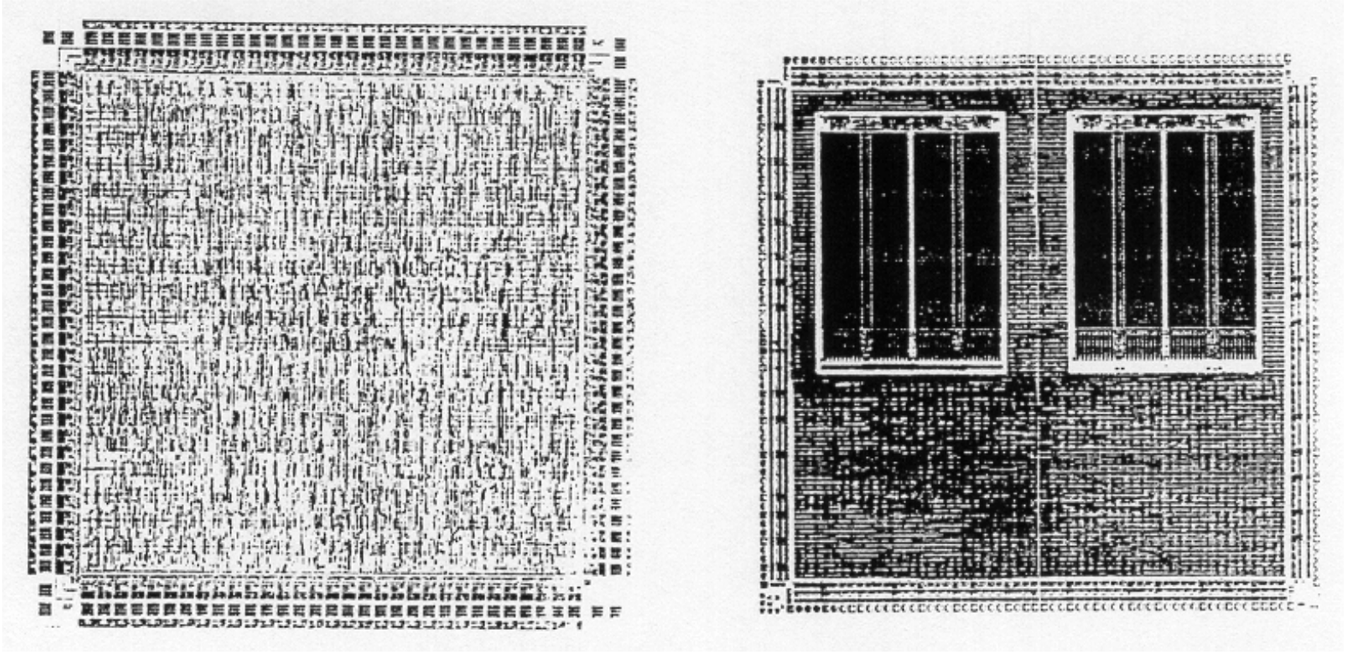
- Prefabricated wafers of, rows of transistors. Customize as needed with “back-end” metal processing (contact cuts, metal wires). Could use a different factory.
- CAD software understands how to make gates and registers.



Gate Array

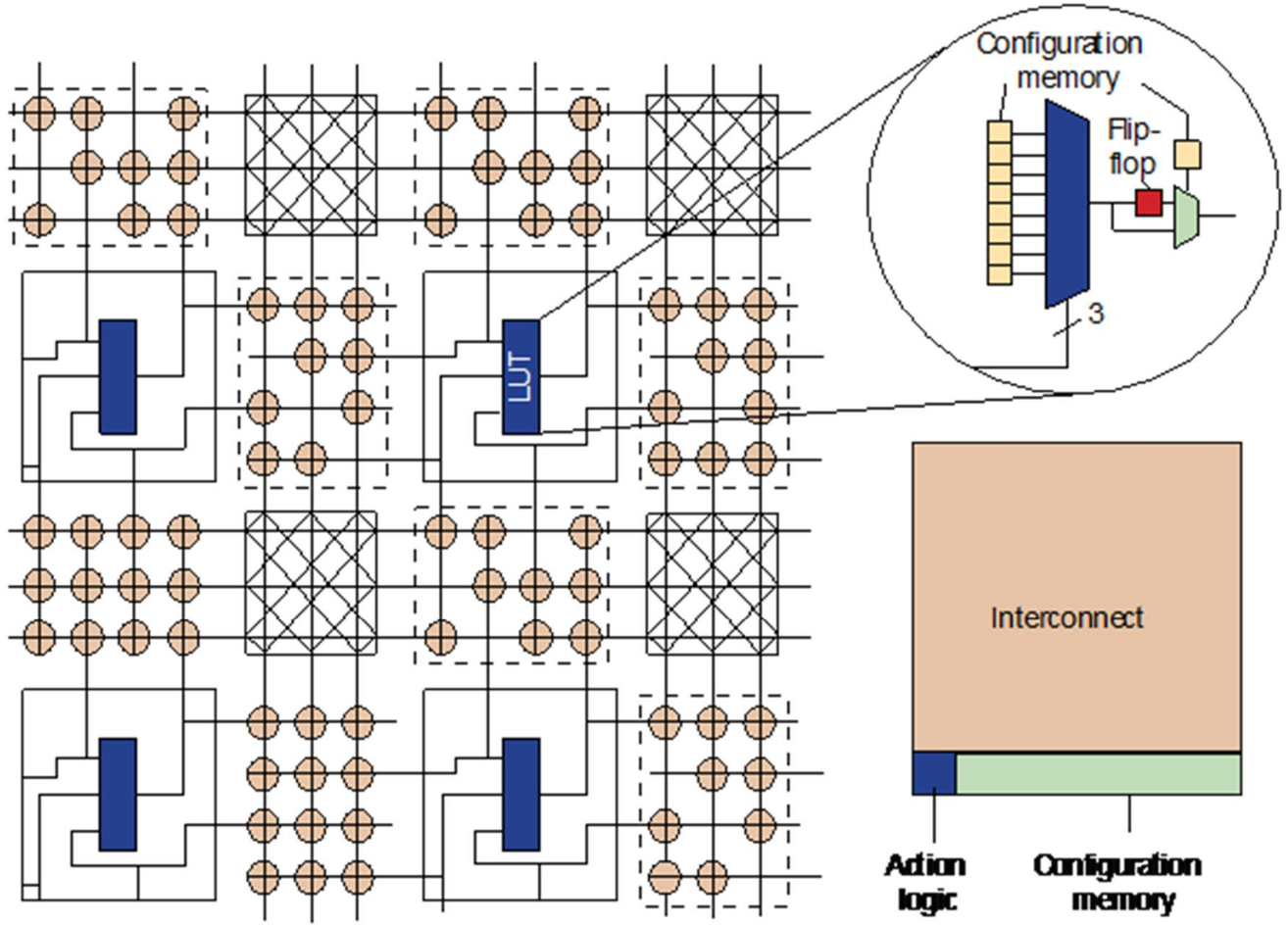
- Shifts large portion of design and mask NRE to vendor.
- Shorter design and processing times, reduced time to market for user.
- Highly structured layout with fixed size transistors leads to large sub-circuits (ex: Flip-flops) and higher per die costs.
- Memory arrays are particularly inefficient, so often prefabricated.
- Displaced by field-programmable gate arrays

Sea-of-gates, structured ASIC, master-slice.



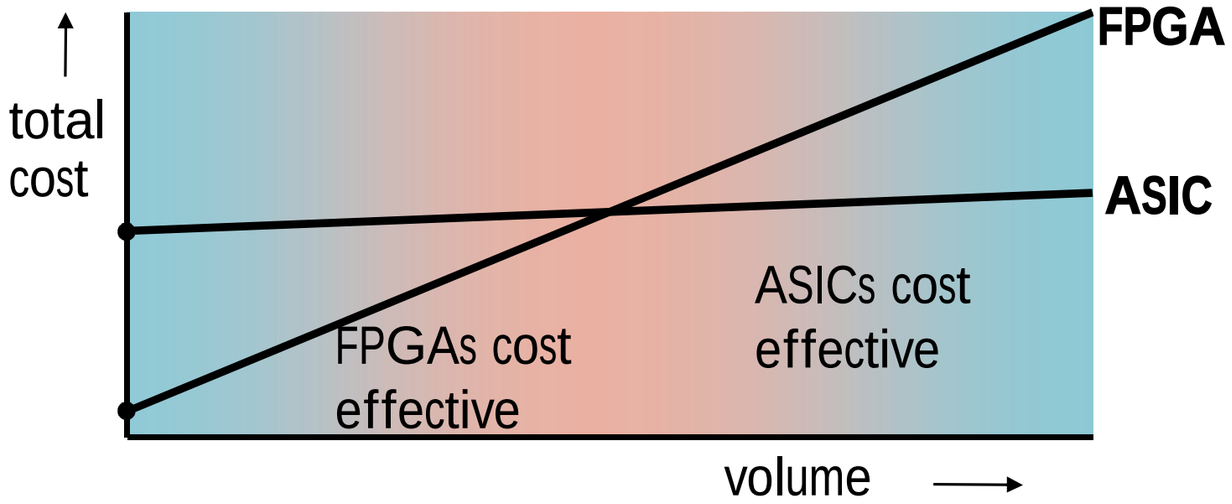
Field Programmable Gate Arrays (FPGA)

- Two-dimensional array of simple logic- and interconnection-blocks.
- Typical architecture: Look-up-tables (LUTs) implement any function of n-inputs (n=3 in this case).
- Optional flip-flop with each LUT.



- Fuses, EPROM, or Static RAM cells are used to store the "configuration".
  - Here, it determines function implemented by LUT, selection of Flip-flop, and interconnection points.
- Many FPGAs include special circuits to accelerate adder carry-chain and many special cores: RAMs, MAC, Ethernetnet, USB, PCIe, CPUs, ...

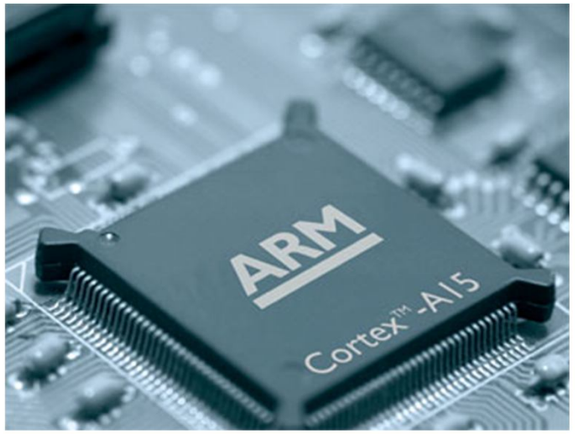
FPGA versus ASIC



- **ASIC:** Higher NRE costs (10's of $M). Relatively Low cost per die (10's of $ or less).
- **FPGAs:** Low NRE costs. Relatively low silicon efficiency ⇒ high cost per part (> 10's of $ to 1000's of $).
- **Cross-over volume** from cost effective FPGA design to ASIC was often in the 100K range.
- **ASICs often have >10x higher performance and 10x lower power for the same application.**

Microprocessors

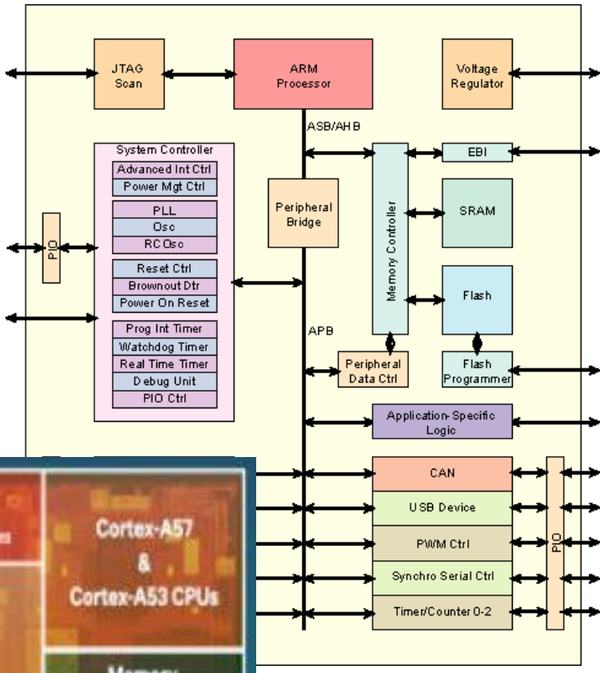
- Where relatively low performance and/or high flexibility is needed, a viable implementation alternative:
  - Software implements desired function
  - "Microcontroller", often with built in nonvolatile program memory and used as single function.
- Two "abstraction" levels:
  - Instruction Set Architecture (ISA)
  - "Synthesizable" RTL model ("soft core", available in HDL)
- Their implementation can be both ASIC or FPGA



§	Assembler
	ADD{cond}{S} Rd, Rn, <Operand2>
	ADC{cond}{S} Rd, Rn, <Operand2>
5E	QADD{cond} Rd, Rm, Rn
5E	QDADD{cond} Rd, Rm, Rn
	SUB{cond}{S} Rd, Rn, <Operand2>
	SBC{cond}{S} Rd, Rn, <Operand2>
	RSB{cond}{S} Rd, Rn, <Operand2>
	RSC{cond}{S} Rd, Rn, <Operand2>
5E	QSUB{cond} Rd, Rm, Rn
5E	QDSUB{cond} Rd, Rm, Rn
2	MUL{cond}{S} Rd, Rm, Rs
2	MLA{cond}{S} Rd, Rm, Rs, Rn
M	UMULL{cond}{S} RdLo, RdHi, Rm, Rs
M	UMLAL{cond}{S} RdLo, RdHi, Rm, Rs
6	UMAAL{cond} RdLo, RdHi, Rm, Rs

System-on-Chip (SOC)

- Brings together: standard cell blocks, custom analog blocks, processor cores, memory blocks, embedded FPGAs, …
- Standardized on-chip buses (or hierarchical interconnect) permit “easy” integration of many blocks.
  - Ex: AXI, …
- “IP Block” business model: Hard- or soft-cores available from third party vendors.
- ARM, inc. is the example. Hard- and “synthesizable” processors.
- ARM and other companies provide, Ethernet, USB controllers, analog functions, memory blocks, …



Qualcomm Snapdragon

- Pre-verified block designs, standard bus interfaces (or adapters) ease integration - lower NREs, shorten TTM.

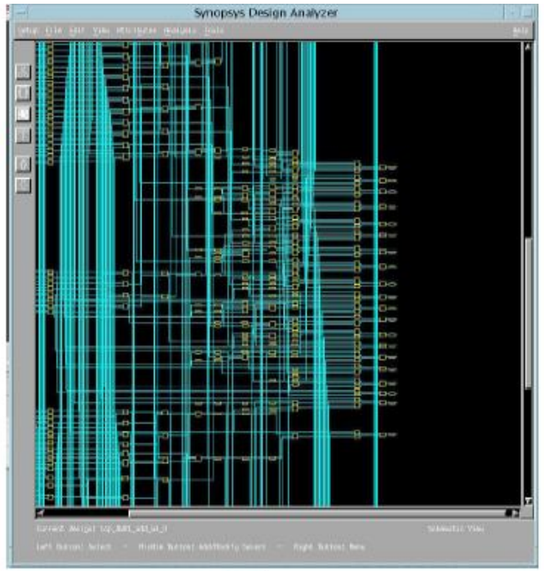
ASIC Design

Verilog to ASIC Layout Flow

- "Push-button" approach

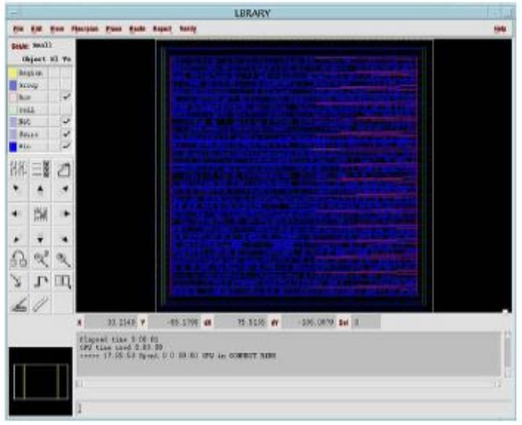
```
module adder64 (a, b, sum);
  input  [63:0] a, b;
  output [63:0] sum;

  assign sum = a + b;
endmodule
```

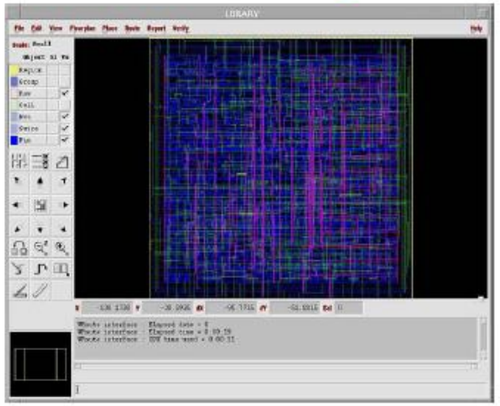


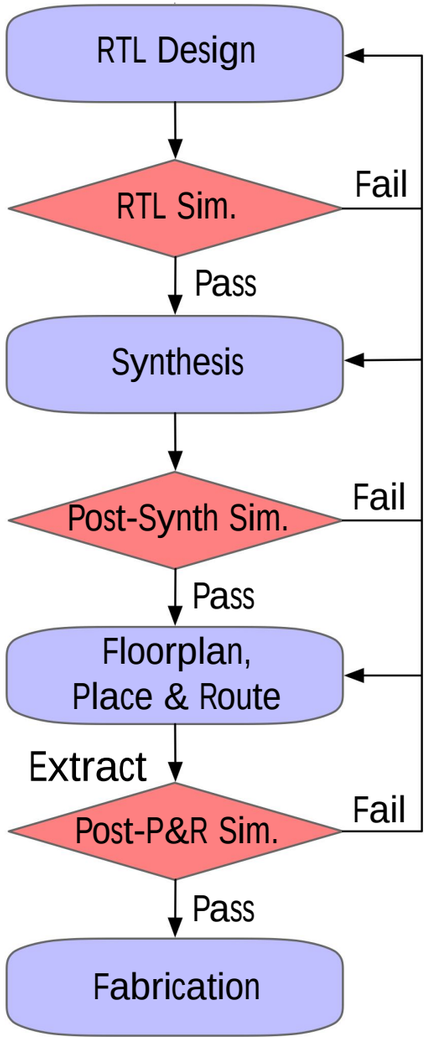
After Synthesis

After Placement

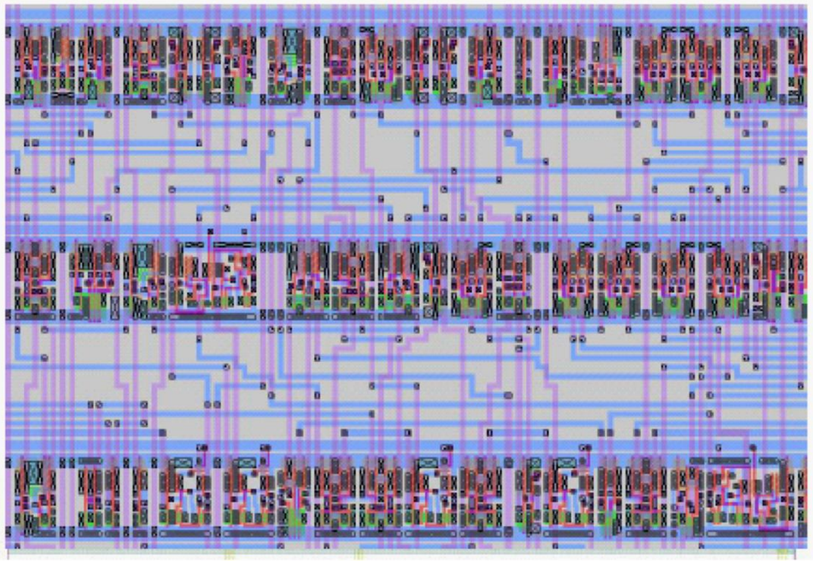


After Routing

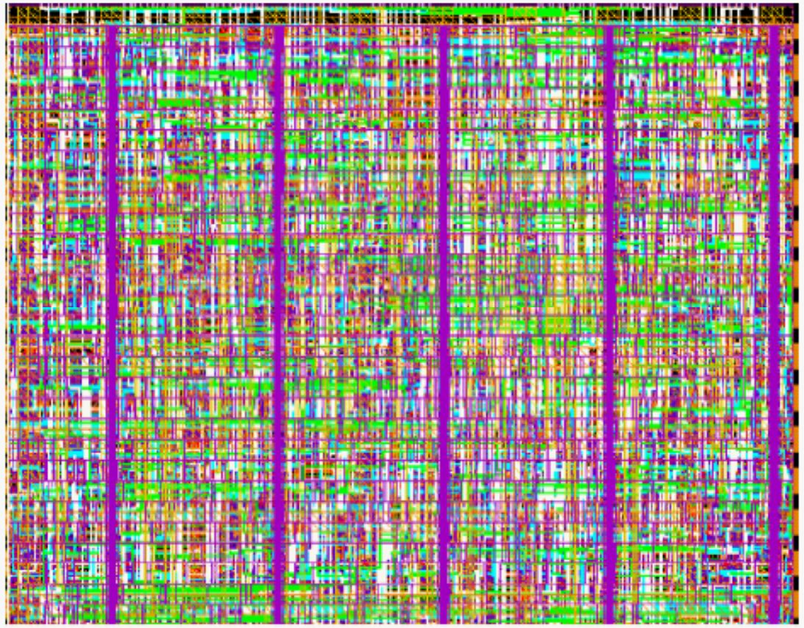




Standard cell layout methodology



Old times: 1μm, 2-metal process



Modern sub-100nm process
"Transistors are free things that fit under wires"

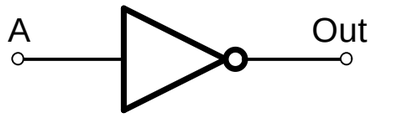
- With limited # metal layers, dedicated routing channels were needed
- Currently area dominated by wires

Standard Cells (ASAP7)

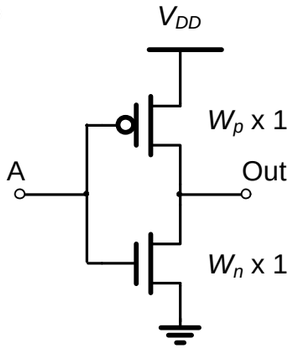
Name

1x Inverter

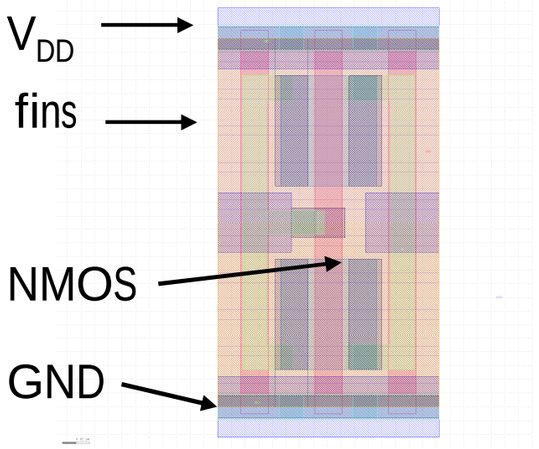
Symbol



Schematic



Layout (gds)



INVX1_ASAP7_75T_R

asap7sc7p5t_24_INVBUF_RVT_TT Cell Library: Process , Voltage 0.70, Temp 25.00

Truth Table

INPUT	OUTPUT
A	Y
0	1
1	0

Footprint

Cell Name	Area
INVx1_ASAP7_75t_R	0.69984

Pin Capacitance Information

Cell Name	Pin Cap(ff)	Max Cap(ff)
	A	Y
INVx1_ASAP7_75t_R	0.43869	46.08000

Leakage Information

Cell Name	Leakage(pW)		
	Min.	Avg	Max.
INVx1_ASAP7_75t_R	0.00000	51.15440	53.32100

Delay Information

Delay(ps) to Y rising :

Cell Name	Timing Arc(Dir)	Delay(ps)		
		Min	Mid	Max
INVx1_ASAP7_75t_R	A->Y (FR)	5.53739	36.47100	284.32000

Verilog for logic sim

```
`celldefine
module INVx1_ASAP7_75t_R (Y, A);
      output Y;
      input A;

      // Function
      not (Y, A);

      // Timing
      specify
            (A => Y) = 0;
      endspecify
endmodule
`endcelldefine
```

Netlist for Spice sim

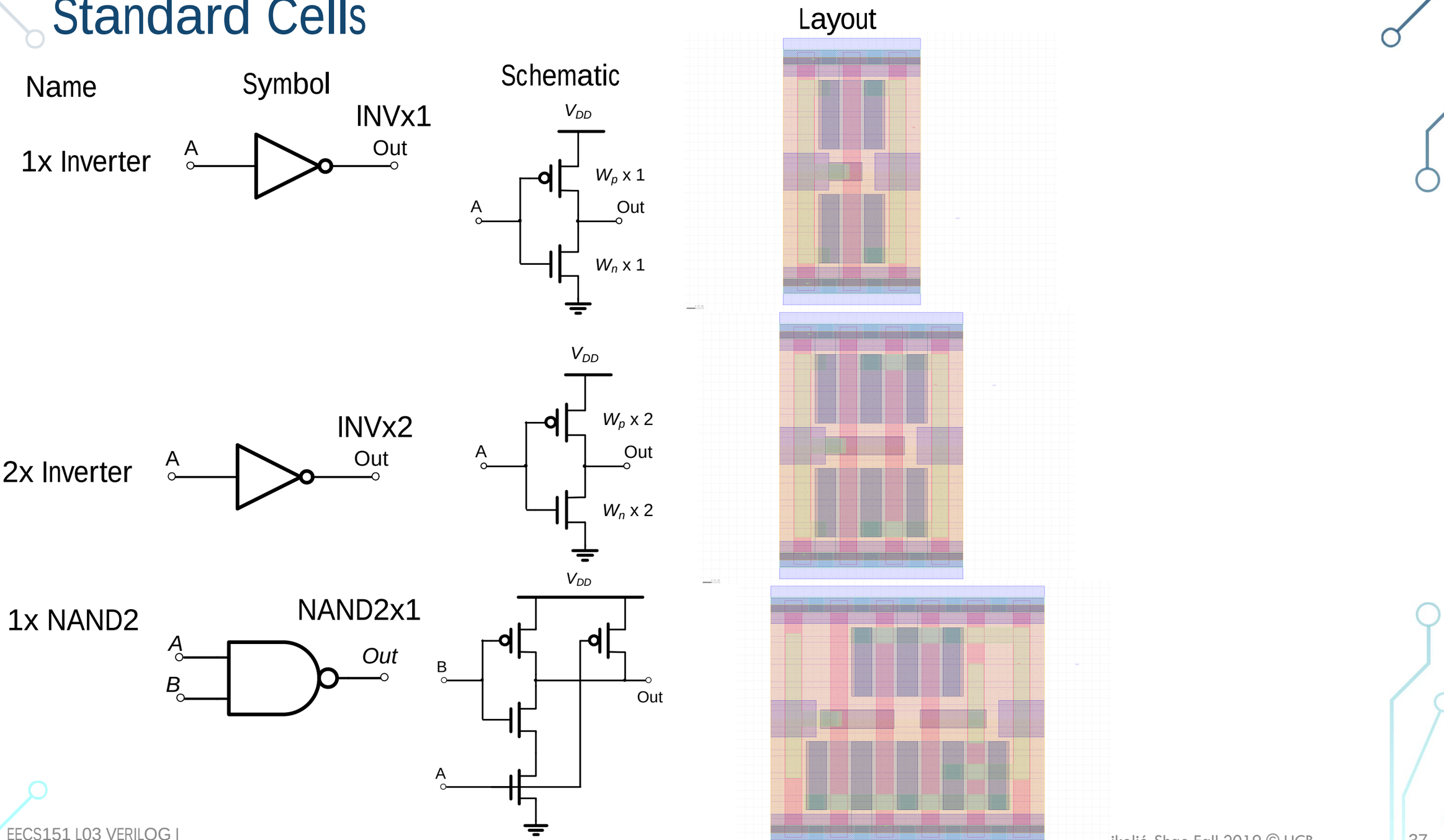
```
.subckt PM_BUFX10_ASAP7_75T_R%A 2 7 10 13 15 23 27 VSS
c21 27 VSS 0.00191995f $X=0.081 $Y=0.135
c22 23 VSS 0.0244917f $X=0.0565 $Y=0.1325
c23 13 VSS 0.00510508f $X=0.135 $Y=0.135
c24 10 VSS 0.0605852f $X=0.135 $Y=0.0675
c25 2 VSS 0.0649407f $X=0.081 $Y=0.0675
r26 23 27 1.66358 $w=1.8e-08 $l=2.45e-08 $layer=M1
$thickness=3.6e-08 $X=0.0565
…
```

.lib for synthesis

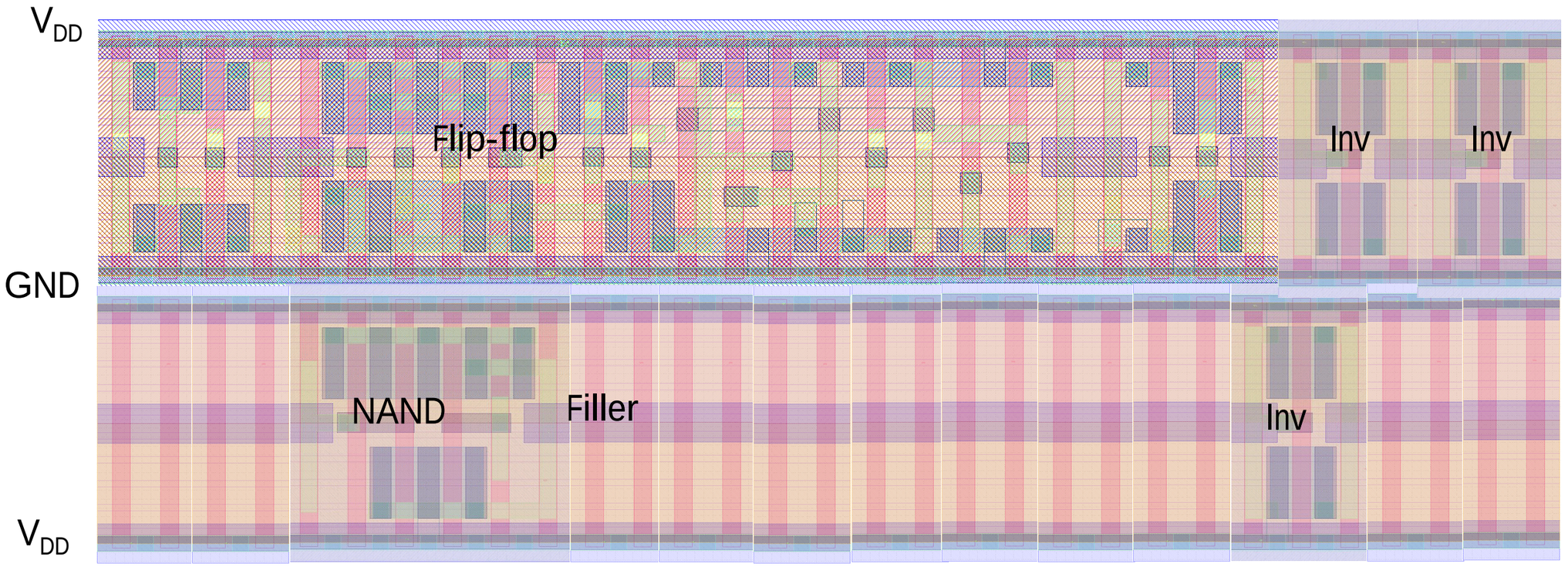
```
cell (INVx11_ASAP7_75t_R) {
   area : 3.03264;
   cell_leakage_power : 562.699;
   pg_pin (VDD) {
     pg_type : primary_power;
     voltage_name : "VDD";
   }
   pg_pin (VSS) {
     pg_type : primary_ground;
     voltage_name : "VSS";
   }
   leakage_power () {
     value : 538.867;
     when : "(A * !Y)";
     related_pg_pin : VDD;
   }
…
```

+ .lef for place and route, and other files for LVS and DRC

Standard Cells



Standard Cell Placement

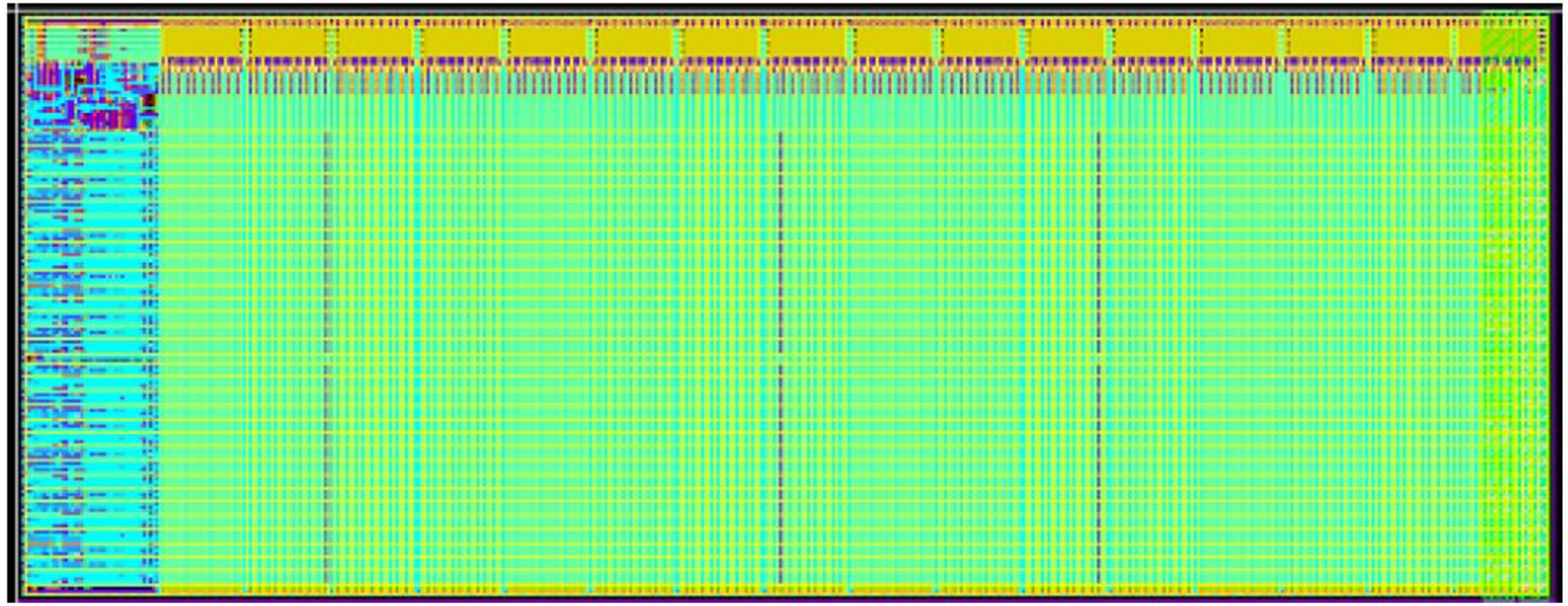


Standard cells abut and flip

Router connects inputs and outputs

Macro modules

256×32 (or 8192 bit) SRAM Generated by hard-macro module generator



- Generate highly regular structures (entire memories, multipliers, etc.) with a few lines of code
- Verilog models for memories automatically generated based on size

Quiz

True or false?

a) FPGAs are usually faster than ASICs

b) Verilog syntax is different for FPGAs and ASICs

c) One cannot change the FPGA function in a deployed product

abc

1. FFF
2. FFT
3. FTF
4. FTT
5. TFF
6. TFT
7. TTF
8. TTT

Hardware Description Languages

Hardware Description Languages

Verilog:

- Simple C-like syntax for structural and behavior hardware constructs
- Mature set of commercial tools for synthesis and simulation
- Used in EECS 151 / 251A

VHDL:

- Semantically very close to Verilog
- More syntactic overhead
- Extensive type system for “synthesis time” checking

System Verilog:

- Enhances Verilog with strong typing along with other additions

BlueSpec:

- Invented by Prof. Arvind at MIT
- Originally built within the Haskell programming language
- Now available commercially: bluespec.edu

Chisel:

- Developed at UC Berkeley
- Used in CS152, CS250
- Available at: chisel.eecs.berkeley.edu

Verilog: Brief History

- Originated at Automated Integrated Design Systems (renamed Gateway) in 1985. Acquired by Cadence in 1989.
- Invented as simulation language. Synthesis was an afterthought. Many of the basic techniques for synthesis were developed at Berkeley in the 80's and applied commercially in the 90's.
- Around the same time as the origin of Verilog, the US Department of Defense developed VHDL (A double acronym! VSIC (Very High-Speed Integrated Circuit) HDL). Because it was in the public domain it began to grow in popularity.
- Afraid of losing market share, Cadence opened Verilog to the public in 1990.
- An IEEE working group was established in 1993, and ratified IEEE Standard 1394 (Verilog) in 1995. We use IEEE Std 1364-2005.
- Verilog is the language of choice of Silicon Valley companies, initially because of high-quality tool support and its similarity to C-language syntax.
- VHDL is still popular within the government, in Europe and Japan, and some Universities.
- Most major CAD frameworks now support both.

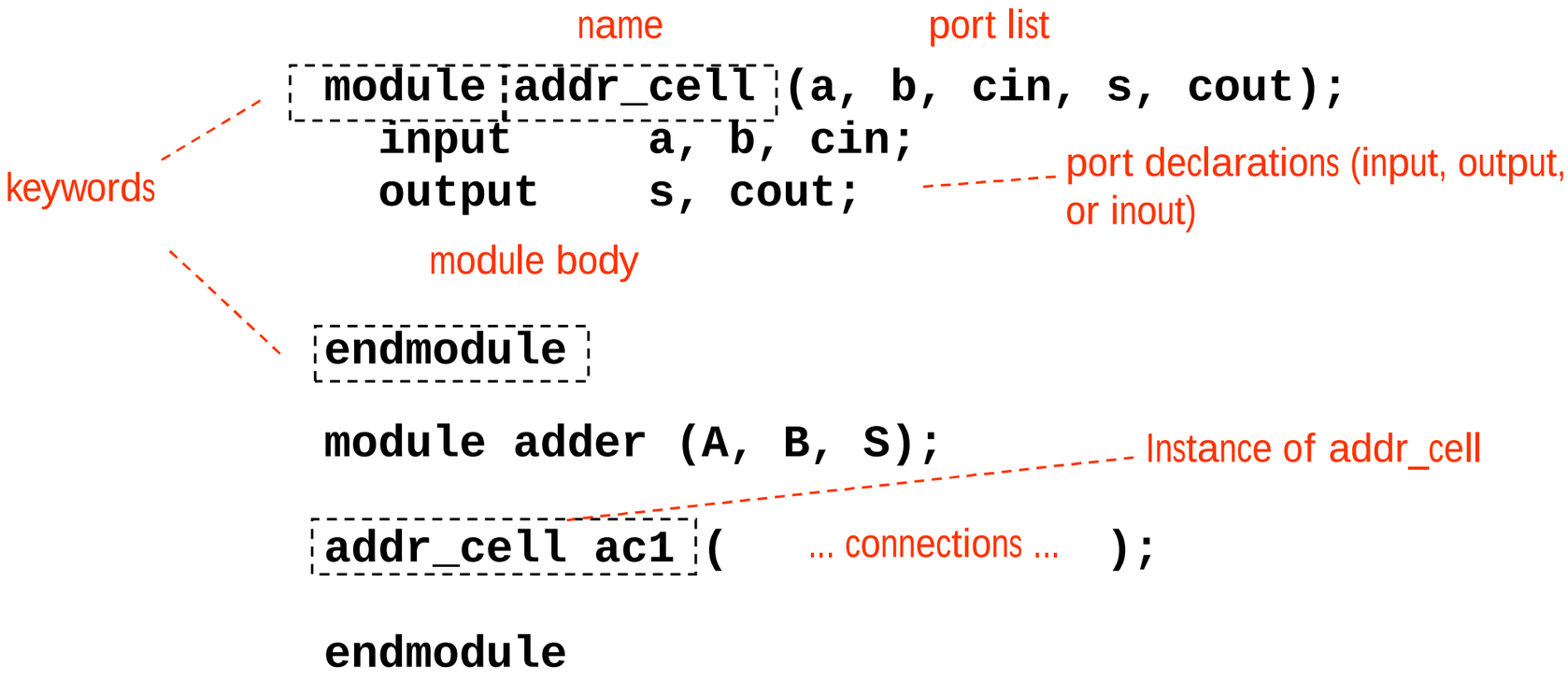
Verilog

Verilog Introduction

- A **`module`** definition describes a component in a circuit
- Two ways to describe module contents:
  - **Structural Verilog**
    - List of sub-components and how they are connected
    - Just like schematics, but using text
    - tedious to write, hard to decode
    - You get precise control over circuit details
    - May be necessary to map to special resources of the FPGA/ASIC
  - **Behavioral Verilog**
    - Describe what a component does, not how it does it
    - Synthesized into a circuit that has this behavior
    - Result is only as good as the tools
- Build up a hierarchy of modules. Top-level module is your entire design (or the environment to test your design).

Verilog Modules and Instantiation

- Modules define circuit components.
- Instantiation defines hierarchy of the design.



Note: A module is not a function in the C sense. There is no call and return mechanism. Think of it more like a hierarchical data structure.

Structural Model - XOR example

```
module xor_gate ( out, a, b );
  input     a, b;
  output    out;
  wire      aBar, bBar, t1, t2;

  not invA (aBar, a);
  not invB (bBar, b);
  and and1 (t1, a, bBar);
  and and2 (t2, b, aBar);
  or  or1 (out, t1, t2);

endmodule
```

module name

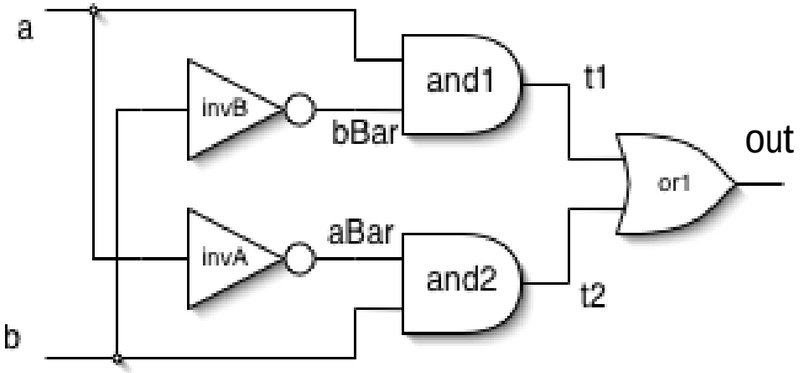
port list

port declarations

internal signal declarations

Built-in gates

instances

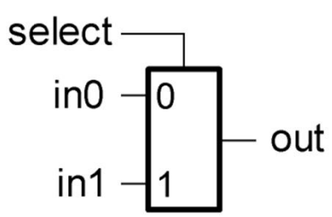


Interconnections (note output is first)

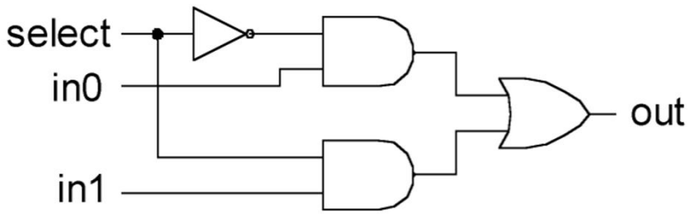
Instance name

- Notes:
  - The instantiated gates are not “executed”. They are active always.
  - xor gate already exists as a built-in (so really no need to define it).
  - Undeclared variables assumed to be wires. Don’t let this happen to you!

Structural Example: 2-to1 mux



a) 2-input mux symbol



b) 2-input mux gate-level circuit diagram

```
/* 2-input multiplexor in gates */
module mux2 (in0, in1, select, out);
   input in0,in1,select;
   output out;
   wire s0,w0,w1;

   not (s0, select);
   and (w0, s0, in0),
       (w1, select, in1);
   or  (out, w0, w1);

endmodule // mux2
```

C++ style comments

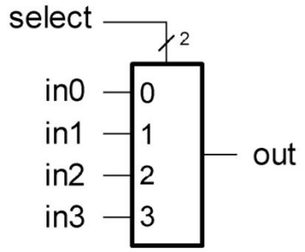
Built-ins don't need instance names

Multiple instances can share the same "master" name.

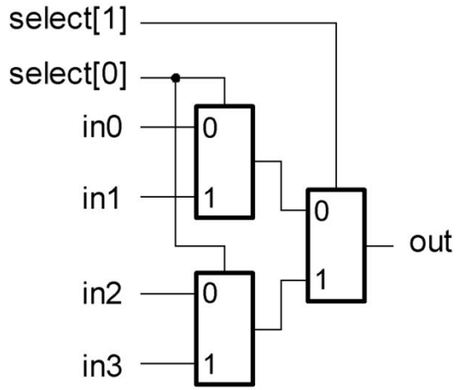
Built-ins gates can have > 2 inputs. Ex:

```
and (w0, a, b, c, d);
```

Instantiation, Signal Array, Named ports



a) 4-input mux symbol



b) 4-input mux implemented with 2-input muxes

```
/* 2-input multiplexor in gates */
module mux2 (in0, in1, select, out);
   input in0,in1,select;
   output out;
   wire s0,w0,w1;
   not (s0, select);
   and (w0, s0, in0),
       (w1, select, in1);
   or  (out, w0, w1);
endmodule // mux2
```

```
module mux4 (in0, in1, in2, in3, select, out);
input in0,in1,in2,in3;
input [1:0] select;
output out;
wire w0,w1;
  mux2
    m0 (.select(select[0]), .in0(in0), .in1(in1), .out(w0)),
    m1 (.select(select[0]), .in0(in2), .in1(in3), .out(w1)),
    m3 (.select(select[1]), .in0(w0), .in1(w1), .out(out));
endmodule // mux4
```

Signal array. Declares select[1], select[0]

Named ports. Highly recommended.

Summary

- The design flow involves translating an abstracted RTL design into physical logic gates, and verifying that it has been done correctly
- Verilog is commonly used to describe RTL
  - Structural or
  - Behavioral