Silicon: From Design to Production

Shankar Viswanathan

BLU - 2019/10/16

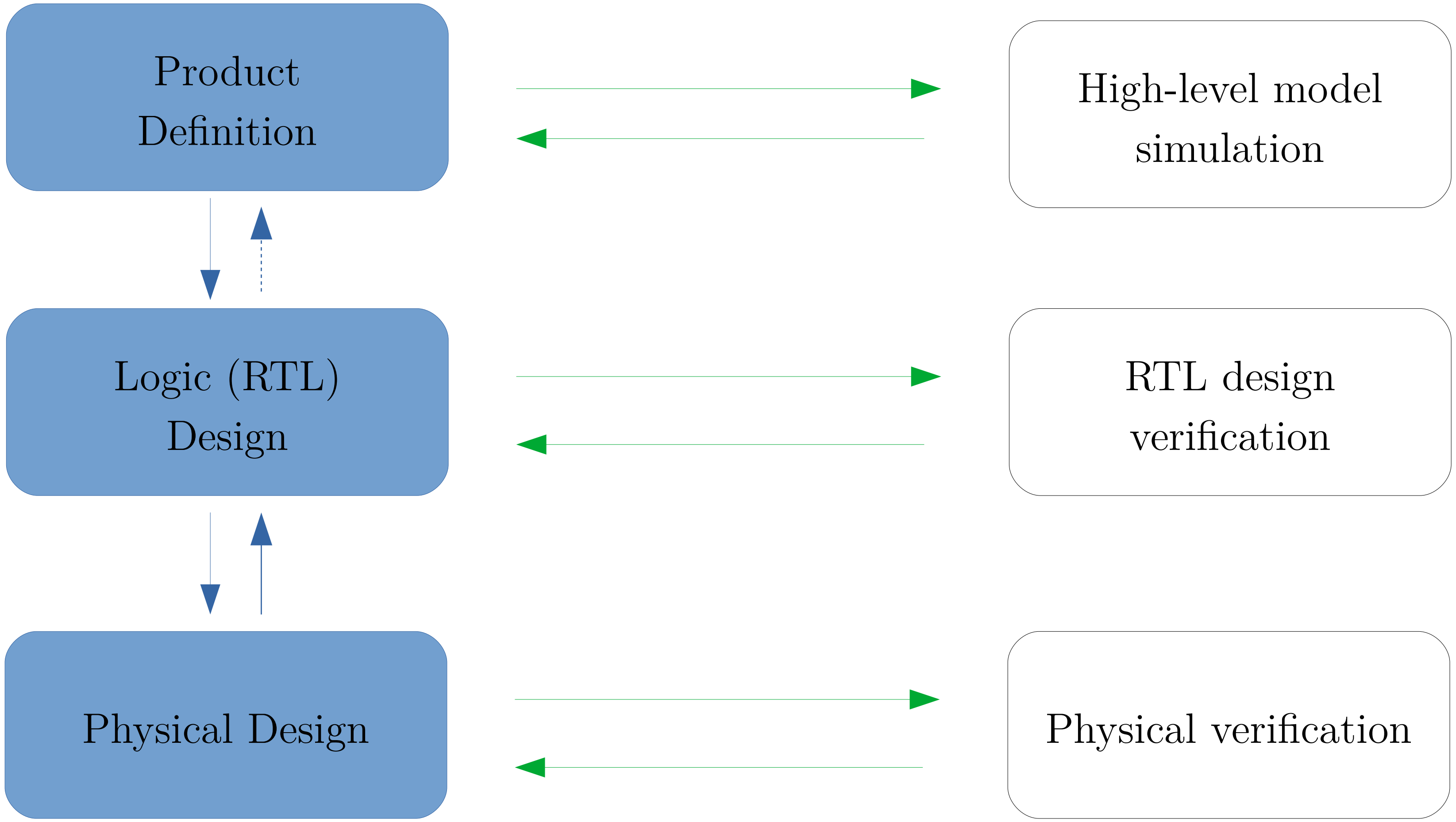
For starters ...

- This is meant as a very high-level introduction to the product development process
- I am not an expert on anything
- Ask questions at any time

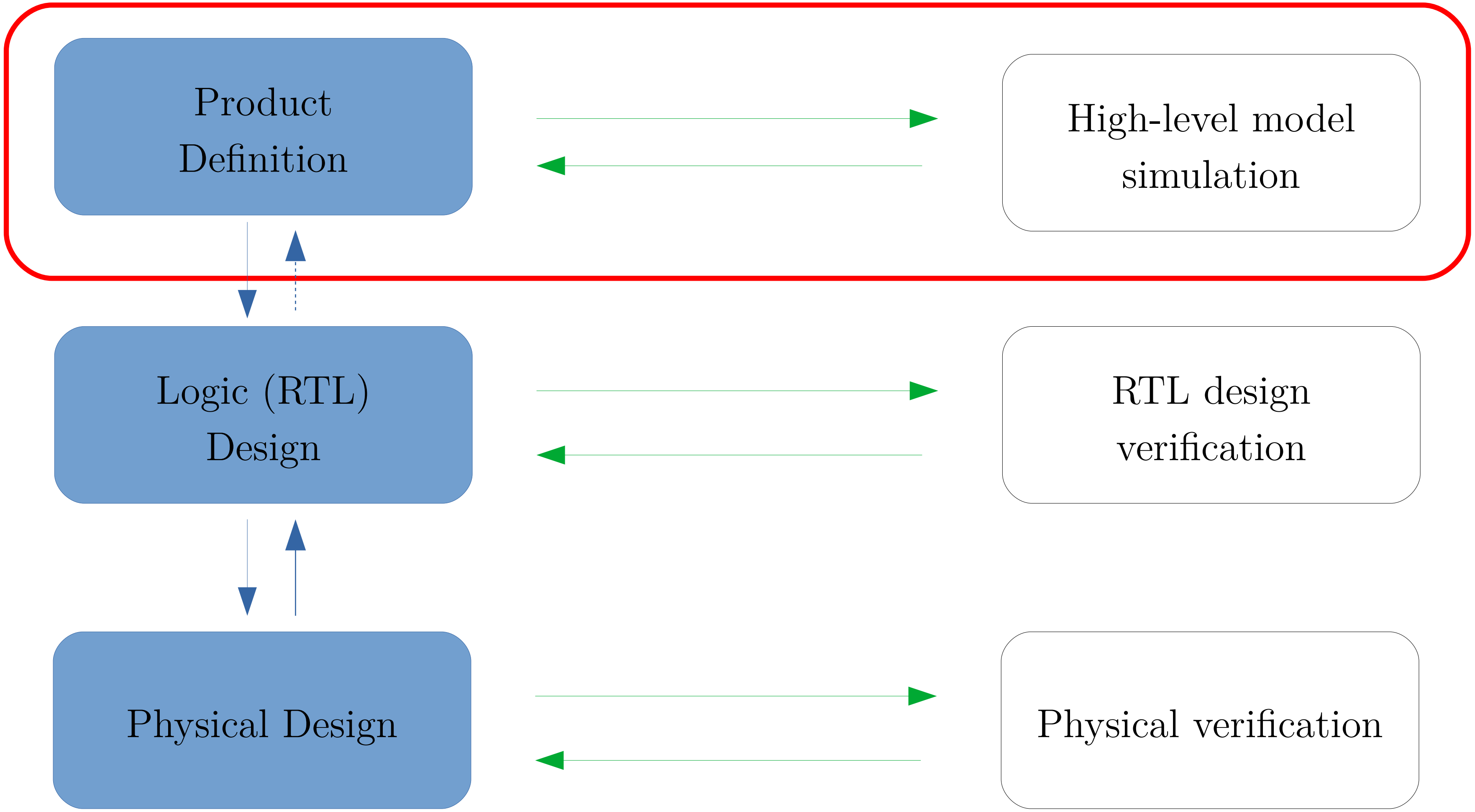
Outline

- Design process
- Silicon manufacturing and test
- Volume production

Design Flow


Product Definition
High-level model simulation
Logic (RTL) Design
RTL design verification
Physical Design
Physical verification

Design Flow


Product Definition
High-level model simulation
Logic (RTL) Design
RTL design verification
Physical Design
Physical verification

Product Definition

- Technology node
- Features
- Power
- Performance
- Chip Area (== cost)

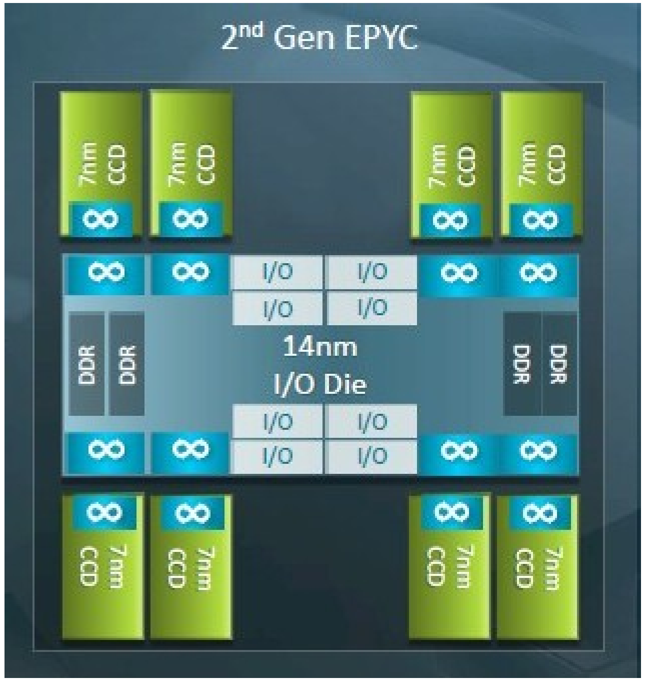
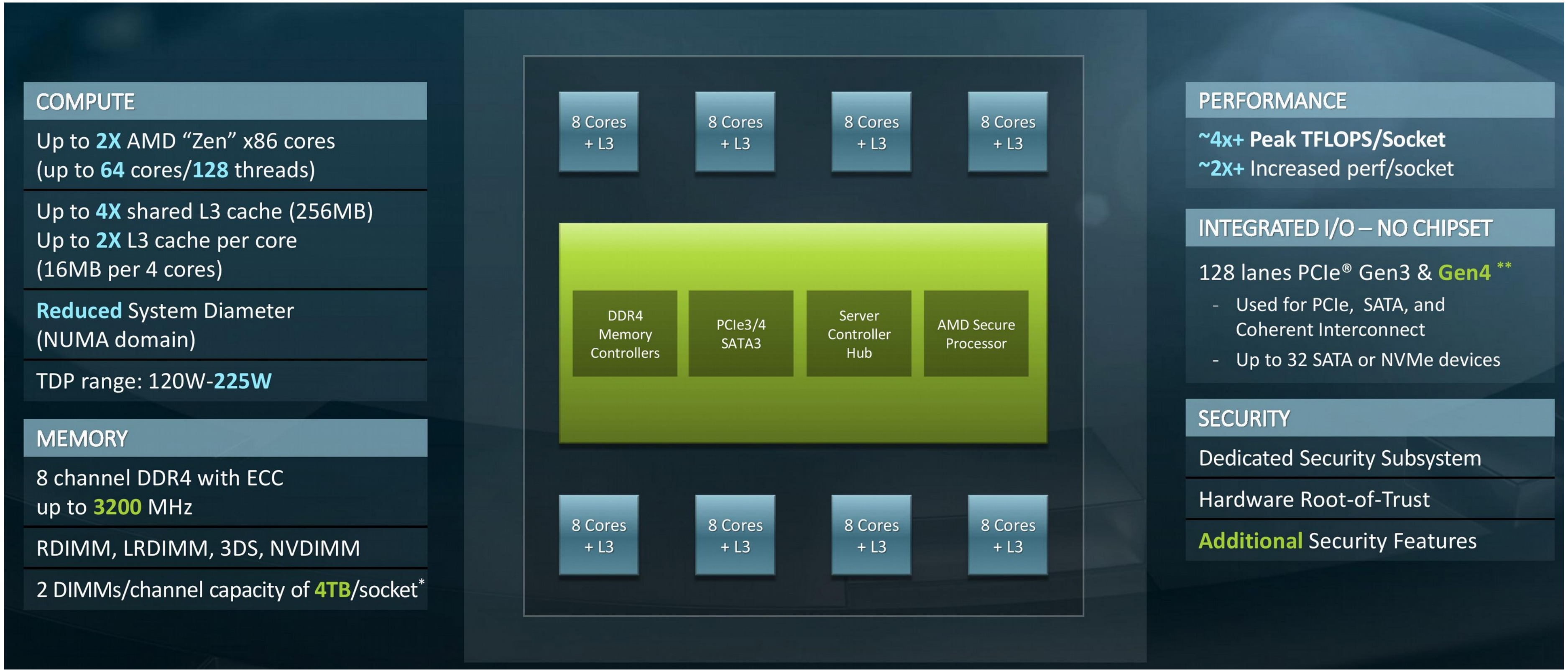


Image: amd.com

Example: AMD EPYC2



COMPUTE

Up to 2X AMD "Zen" x86 cores (up to 64 cores/128 threads)

Up to 4X shared L3 cache (256MB)
Up to 2X L3 cache per core
(16MB per 4 cores)

Reduced System Diameter (NUMA domain)

TDP range: 120W-225W

MEMORY

8 channel DDR4 with ECC up to 3200 MHz

RDIMM, LRDIMM, 3DS, NVDIMM

2 DIMMs/channel capacity of 4TB/socket*

PERFORMANCE

~4x+ Peak TFLOPS/Socket
~2x+ Increased perf/socket

INTEGRATED I/O – NO CHIPSET

128 lanes PCIe® Gen3 & Gen4 **

- Used for PCIe, SATA, and Coherent Interconnect
- Up to 32 SATA or NVMe devices

SECURITY

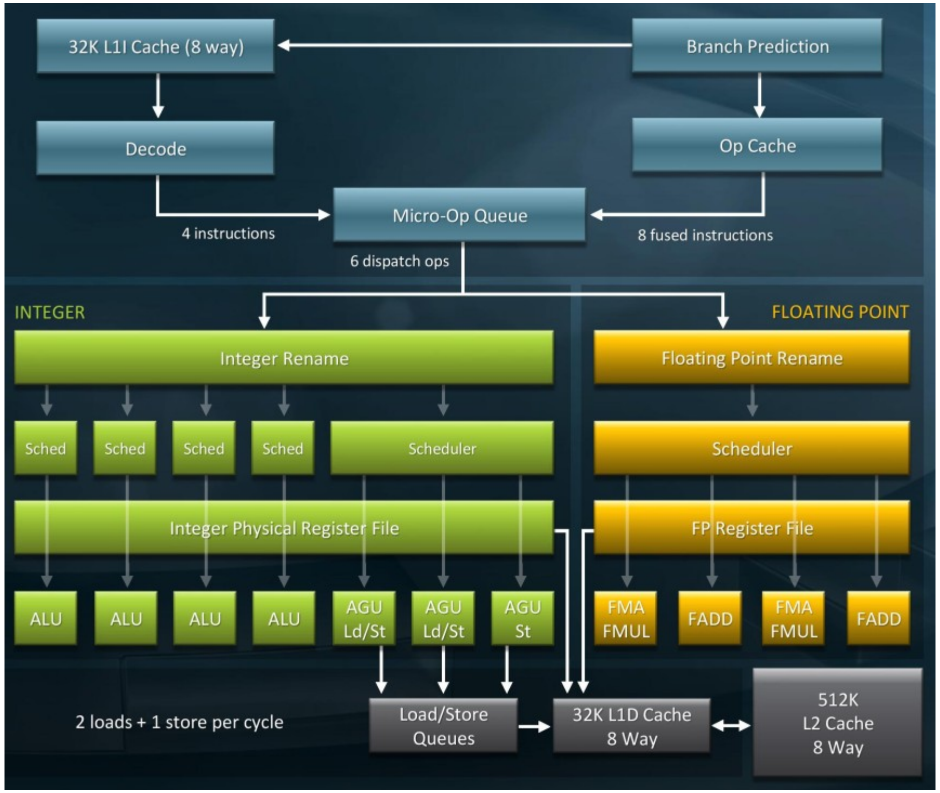
Dedicated Security Subsystem

Hardware Root-of-Trust

Additional Security Features

Image: https://www.servethehome.com/amd-epyc-7002-series-rome-delivers-a-knockout/4/

High-level design



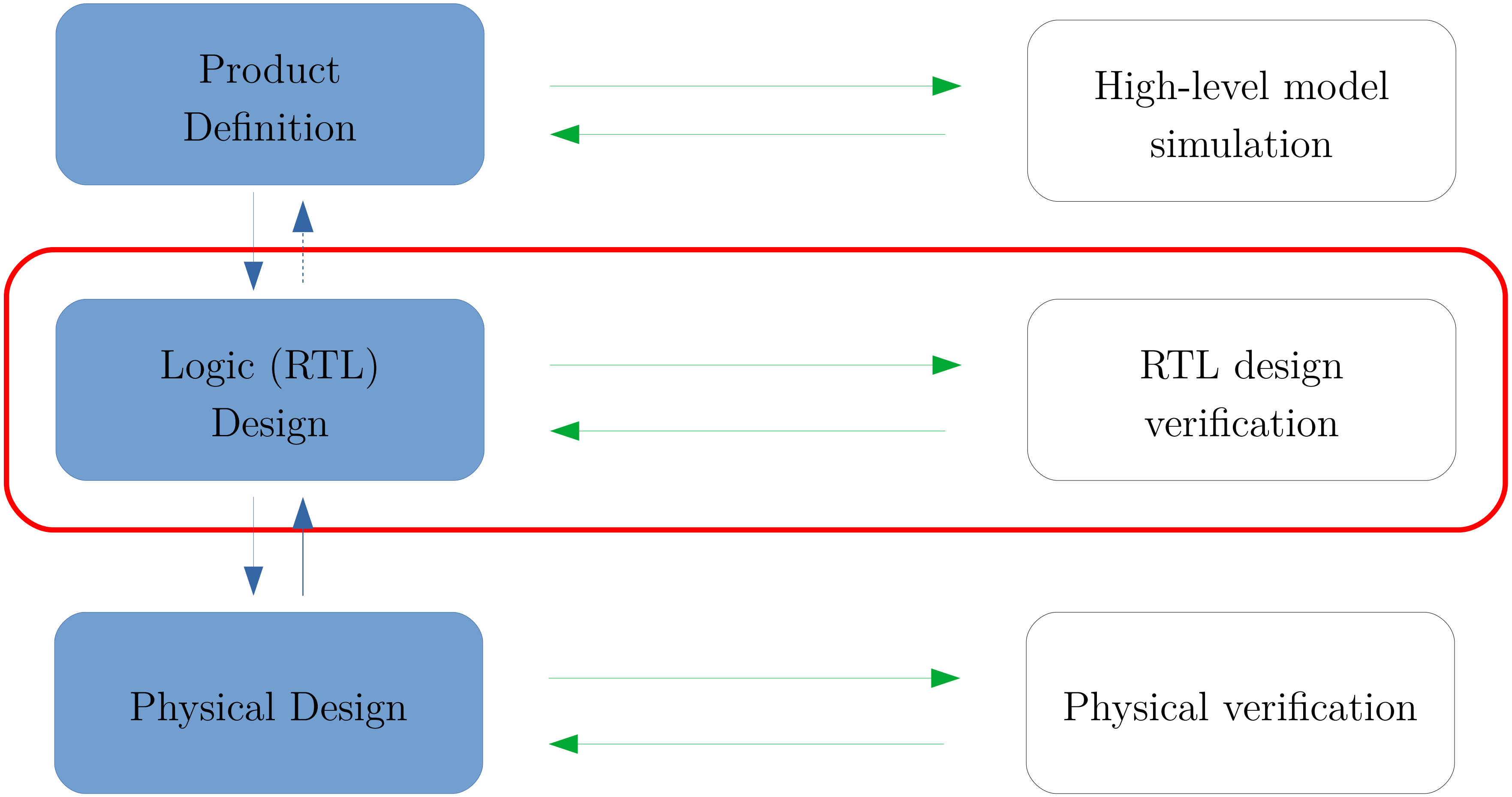
- Design features broken down into smaller logic blocks
- Pipeline structure determined

Image: https://www.amd.com/en/technologies/zen-core

High-level Models

- Performance simulator
- Power estimation simulations
- Rough floorplan & chip area estimation

Design Flow

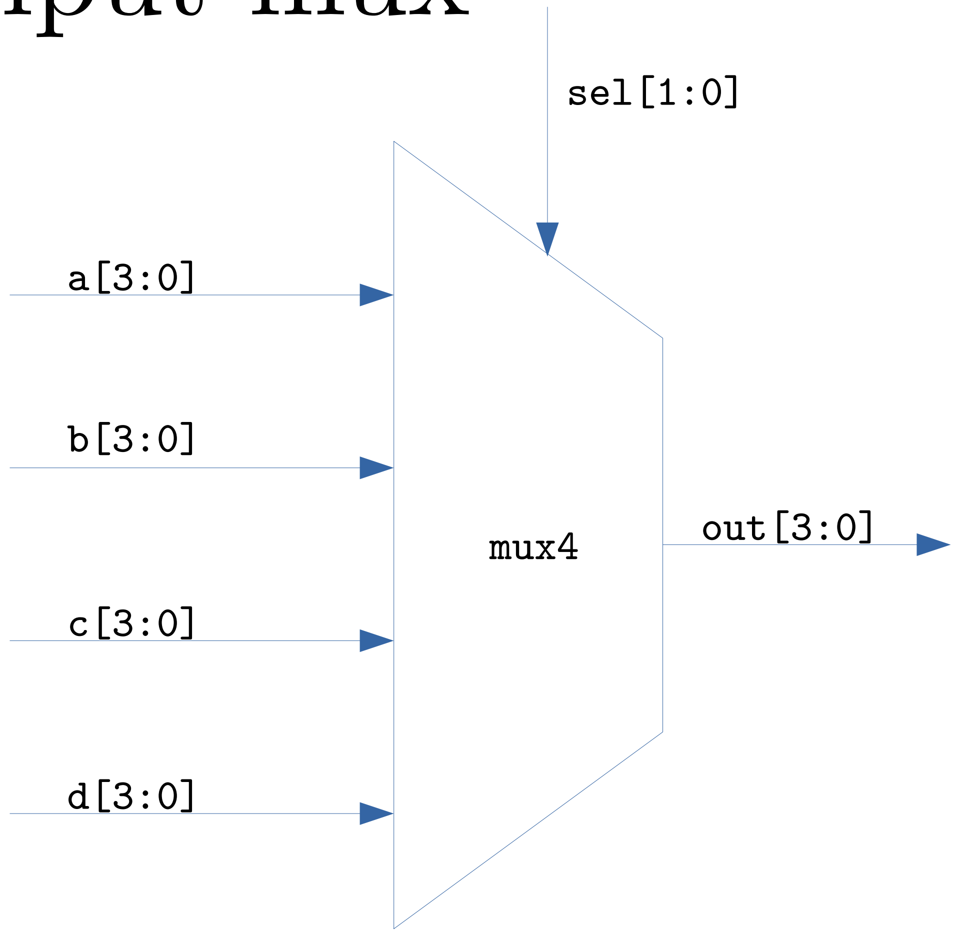

Product Definition
High-level model simulation
Logic (RTL) Design
RTL design verification
Physical Design
Physical verification

Logic Design

- Typically done using a Hardware Description Language such as Verilog
- Often uses libraries of commonly used elements like FIFOs, arrays, muxes
- Abstract behavioral models of analog elements – e.g. clock generator, high-speed IO drivers
- Needs to be aware of pipelines and timing

Verilog example: 4-input mux

```
module mux4(
         input [3:0] a,
         input [3:0] b,
         input [3:0] c,
         input [3:0] d,
         input [1:0] sel,
         output reg [3:0] out);

  always @* begin
    case(sel[1:0])
         2'b00: out[3:0] = a[3:0];
         2'b01: out[3:0] = b[3:0];
         2'b10: out[3:0] = c[3:0];
         2'b11: out[3:0] = d[3:0];
    endcase
  end
endmodule
```



Another example

```
module lshift (input d,
               input clk,
               input rst_l,
               output reg [3:0] sreg);

   always @ (posedge clk) begin
      if (!rst_l) begin
         sreg[3:0] <= 4'b0;
      end else begin
         sreg[3:0] <= {sreg[2:0], d};
      end
   end
endmodule
```

- What does this do?
- How is it different from previous mux example?

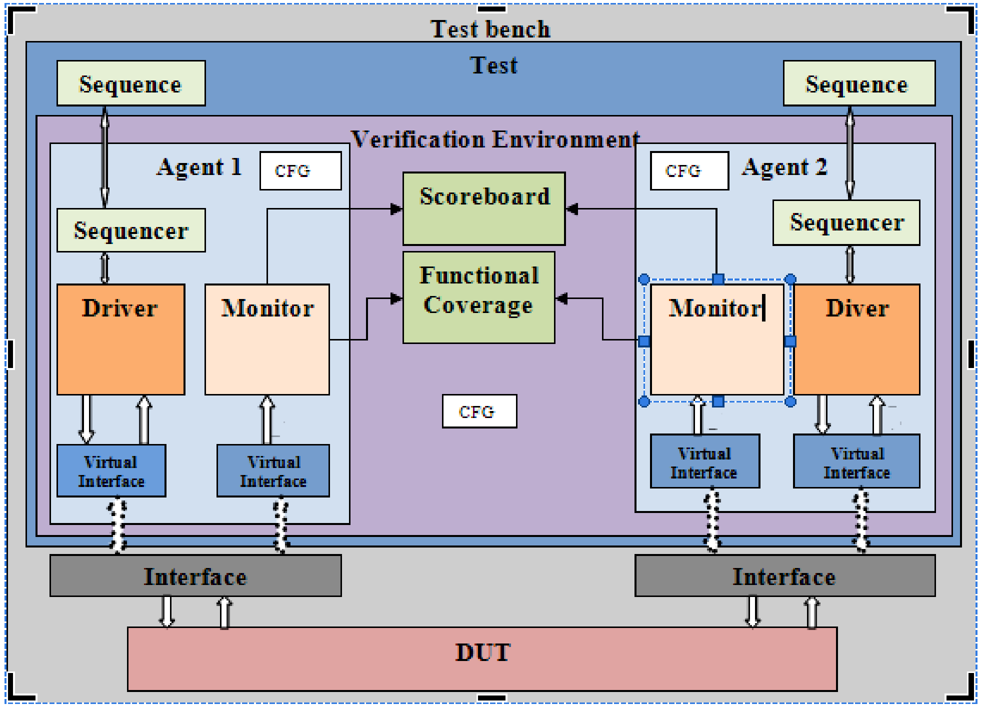
Special Consideration: DFx

- **D**esign **F**or **T**est:
  - JTAG, Scan, Loopback modes
- **D**esign **F**or **D**ebug:
  - Performance counters, HW breakpoints, Debug output pins

Design Verification

- Design verification is done to confirm that the design meets specification, i.e. is functionally correct
  - Cost of missed bugs can be very severe in terms of material cost and TTM
- Uses a mixture of simulation based checks as well as formal tools
- Unit-level as well as chip-level testbenches are constructed to run through different scenarios
  - Use both directed and constrained random tests to drive input
- Checkers and assertions are used to catch illegal conditions
- Extensive use of functional coverage and code coverage
- Generally 3 verification engineers for every RTL design engineer on the team

Testbench examples



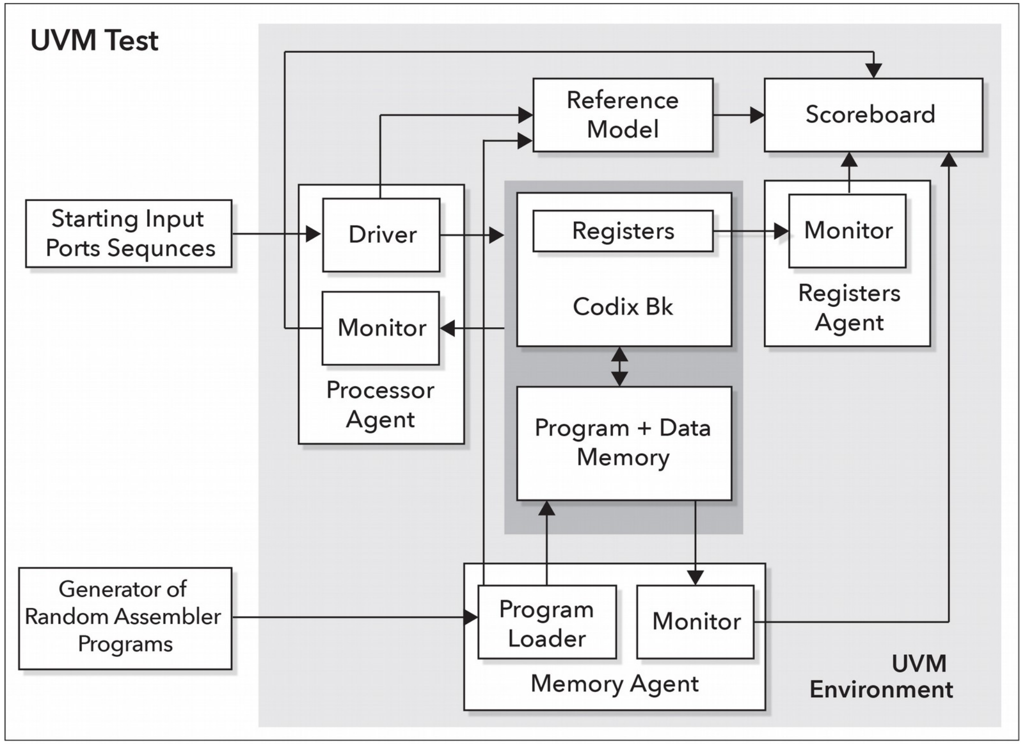


Image: https://www.researchgate.net/figure/UVM-test-bench-Architecture-All-complex-test-benches-may-be-architected-as-shown-in-the_fig1_303759959

Image: https://verificationacademy.com/verification-horizons/june-2017-volume-13-issue-2/automation-and-reuse-in-risc-v-verification-flow

Simple Testbench: mux

```
module mux_tb_top;

    reg  [3:0] a, b, c,
    d;

    wire [3:0] out;

    reg  [1:0] sel;

    reg        clk;

    integer    i;


    mux4 mux4_i0(

        .a (a),

        .b (b),

        .c (c),

        .d (d),

        .sel (sel),

        .out (out));
```

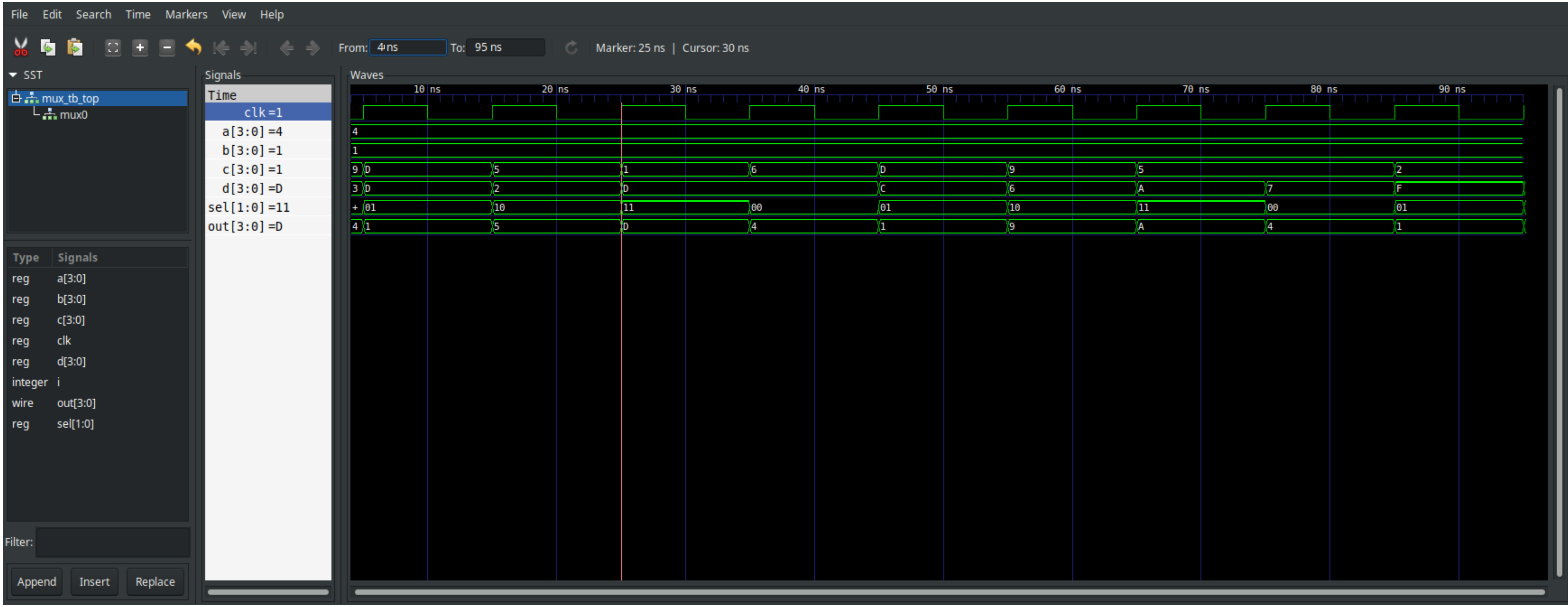
```
initial begin

    $dumpfile("dump.vcd");

    $dumpvars(0, mux_tb_top);


    sel[1:0] = 2'b0;

    a[3:0] = $urandom;

    b[3:0] = $urandom;

    c[3:0] = $urandom;

    d[3:0] = $urandom;

    clk = 0;

    i = 0;

    $monitor ("[%0t] sel=0x%0h a=0x%0h
    b=0x%0h c=0x%0h d=0x%0h out=0x%0h",
    $time, sel, a, b, c, d, out);

end
```

```
    always #5 clk = ~clk;

    always @ (posedge clk) begin

       sel <= sel + 1;

       c[3:0] <= $urandom;

       d[3:0] <= $urandom;

       i = i + 1;

    end


    always @(i) begin

       if (i == 10)

          $finish;

    end


endmodule // mux_tb_top
```

Simulation Log

```
[0] sel=0x0 a=0x4 b=0x1 c=0x9 d=0x3 out=0x4
[5] sel=0x1 a=0x4 b=0x1 c=0xd d=0xd out=0x1
[15] sel=0x2 a=0x4 b=0x1 c=0x5 d=0x2 out=0x5
[25] sel=0x3 a=0x4 b=0x1 c=0x1 d=0xd out=0xd
[35] sel=0x0 a=0x4 b=0x1 c=0x6 d=0xd out=0x4
[45] sel=0x1 a=0x4 b=0x1 c=0xd d=0xc out=0x1
[55] sel=0x2 a=0x4 b=0x1 c=0x9 d=0x6 out=0x9
[65] sel=0x3 a=0x4 b=0x1 c=0x5 d=0xa out=0xa
[75] sel=0x0 a=0x4 b=0x1 c=0x5 d=0x7 out=0x4
[85] sel=0x1 a=0x4 b=0x1 c=0x2 d=0xf out=0x1
[95] sel=0x2 a=0x4 b=0x1 c=0x2 d=0xe out=0x2
```

Waveform view



From: 4 ns To: 95 ns Marker: 25 ns | Cursor: 30 ns

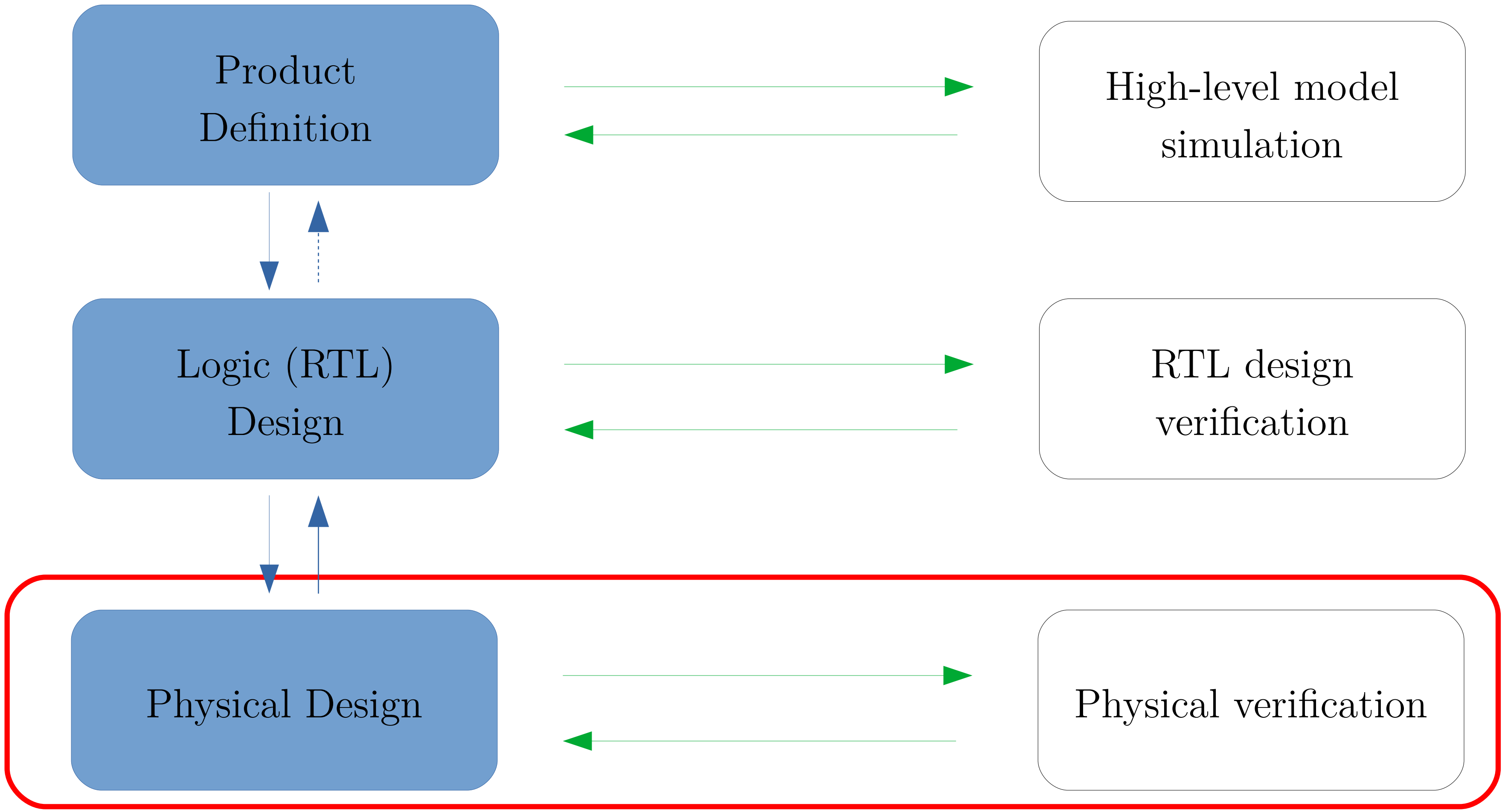
```
Time
clk=1
a[3:0]=4
b[3:0]=1
c[3:0]=1
d[3:0]=D
sel[1:0]=11
out[3:0]=D
```

```
10 ns      20 ns      30 ns      40 ns      50 ns      60 ns      70 ns      80 ns      90 ns
4
1
9 D        5          1          6          D          9          5                     2
3 D        2          D                     C          6          A          7          F
+ 01       10         11         00         01         10         11         00         01
4 1        5          D          4          1          9          A          4          1
```

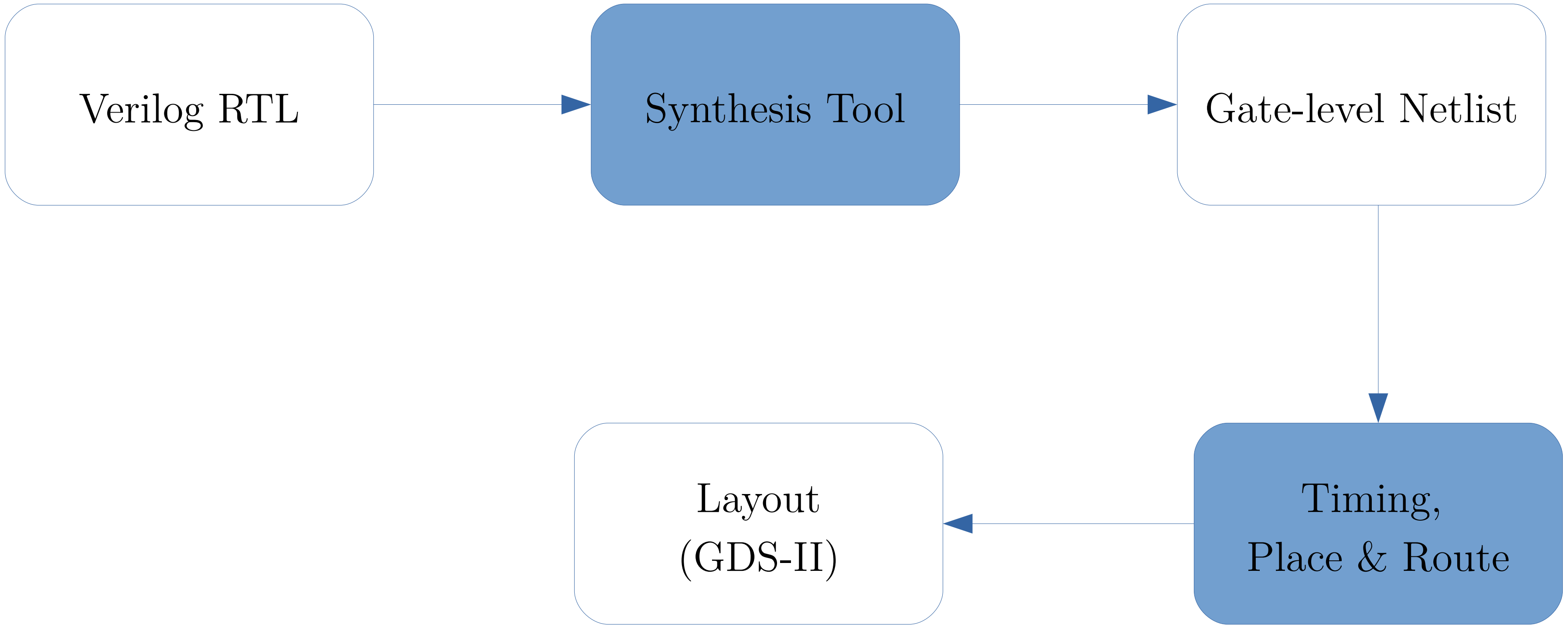
Type	Signals
reg	a[3:0]
reg	b[3:0]
reg	c[3:0]
reg	clk
reg	d[3:0]
integer	i
wire	out[3:0]
reg	sel[1:0]

Filter:

Design Flow


Product Definition
High-level model simulation
Logic (RTL) Design
RTL design verification
Physical Design
Physical verification

Physical Design Flow

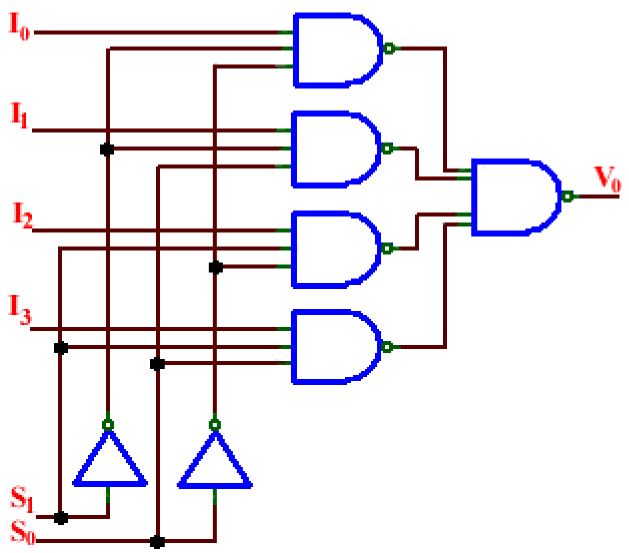


Logic Synthesis

- Converts RTL description into logic gates
- Modern tools first minimize the logic and then map it to library of gate elements
- Takes timing targets and other size or power constraints as inputs
- Result is a netlist file that is a text representation of a schematic

Synthesis example

- 4:1 mux constructed from NAND gates and inverters

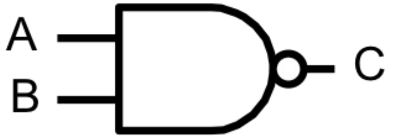


Timing, Place & Route

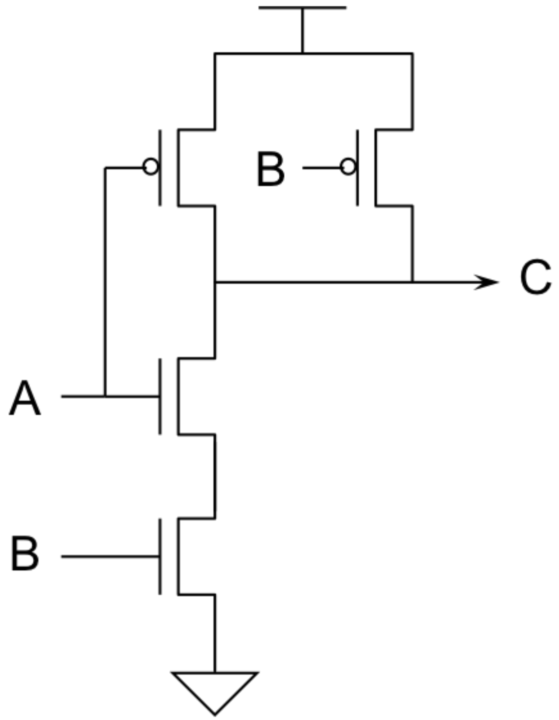
- Takes each gate and places corresponding layout cell in the block
- Routes interconnect (wires) between cells based on connectivity and timing constraints
- Clock tree generation
- Optimize and iterate for most compact layout that satisfies all constraints

Example: NAND gate

```
C = ~(A & B);
```



A	B	C
0	0	1
0	1	1
1	0	1
1	1	0



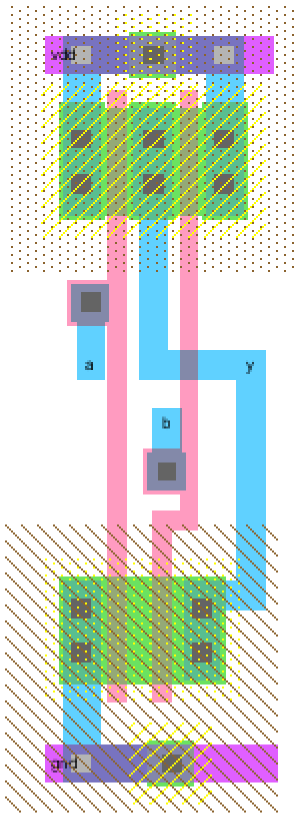


Image: Electric User's Manual

Layout example: 4:1 mux

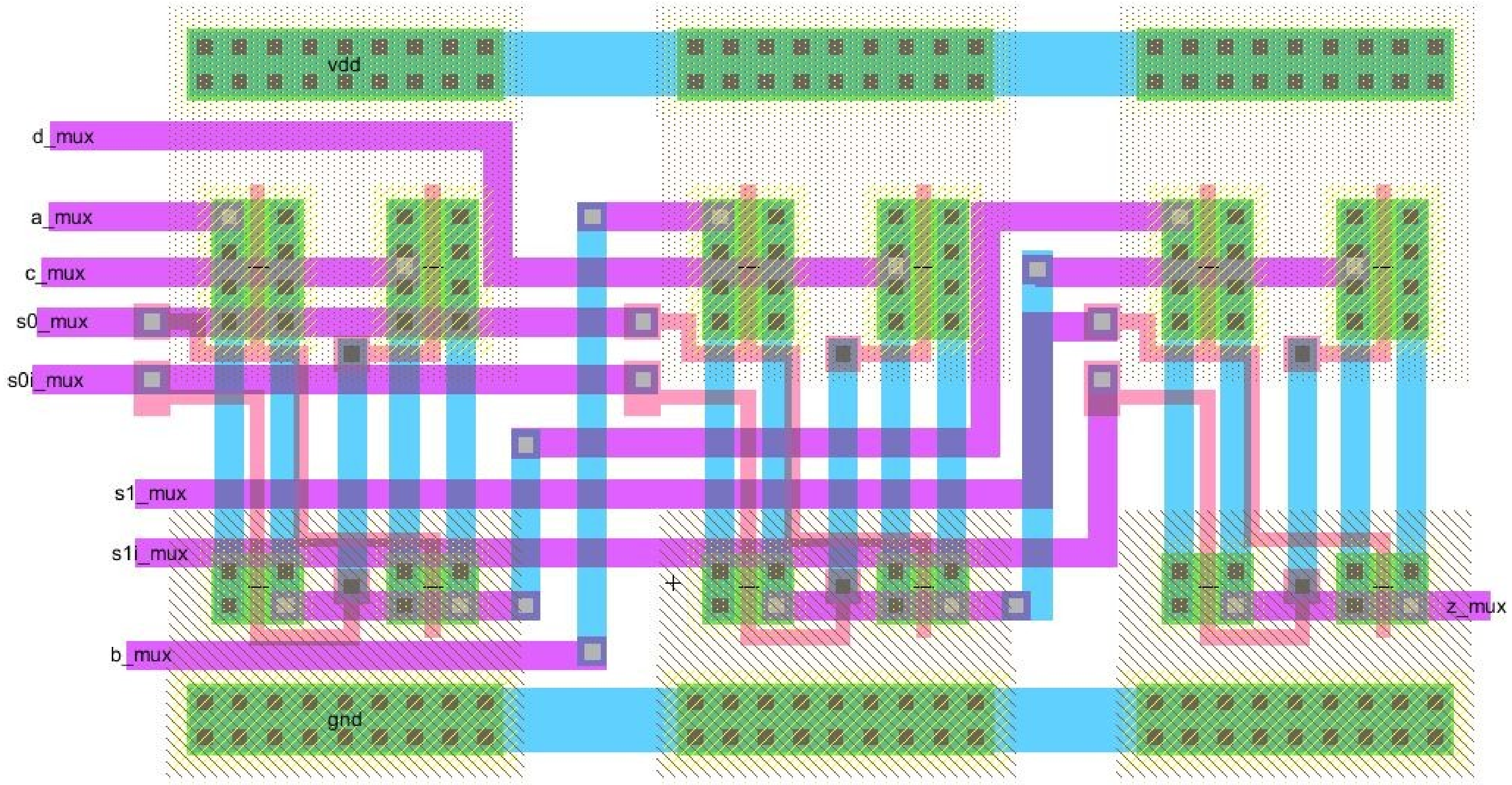


Image credit: http://cmosedu.com/jbaker/courses/ee421L/f13/students/lopeztel/proj/proj.html

Fullchip layout: MOS 6502

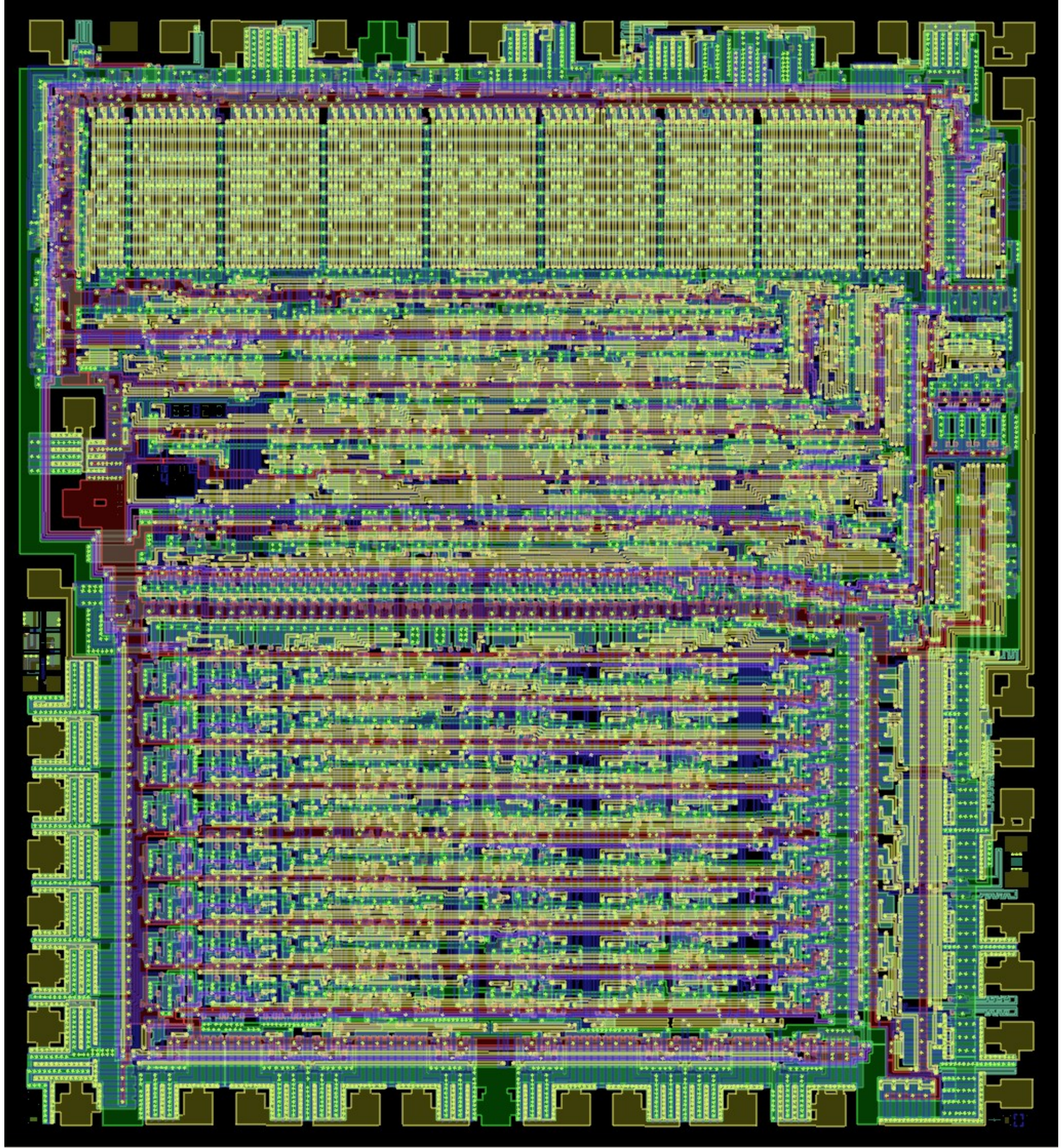


Image: http://www.visual6502.org

Physical Verification

- Run timing simulations
- Check if final netlist matches original RTL intent (logical equivalence)
- Check equivalence of netlist to final layout (LVS)
- Ensure final layout meets design rules (DRC)

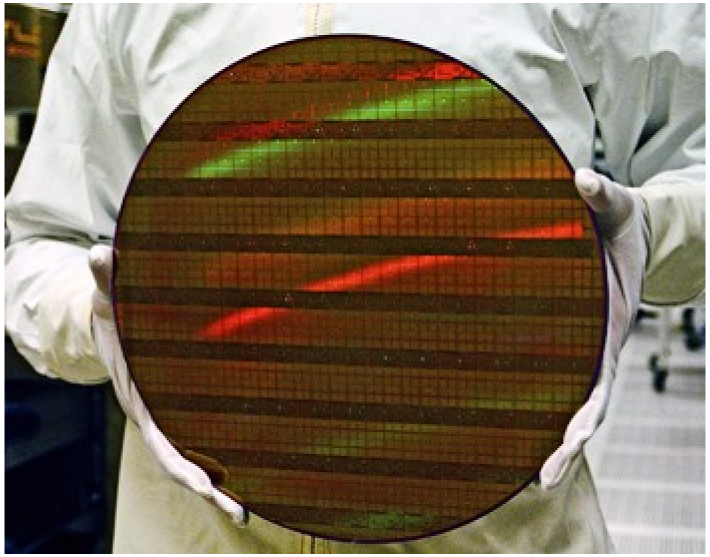
Tapeout

- Final verified GDSII mask released to fab for initial sample production
- Cake and ice cream celebration ensues

In parallel ...

- Microcode & firmware design and testing
- Software development: firmware, BIOS, compilers, drivers etc.
- Advanced use case testing using FPGAs or emulators
- Test pattern generation for chip testing

Fabrication



Images: https://www.wired.com/2010/10/inside-a-state-of-the-art-cleanroom/

Wafer testing

- Wafers are first tested for basic defects – bad die are marked
- Wafer is cut into individual dies and good dies are sent for packaging
- Packaged chip is tested again for packaging related defects

Automated testing

- ATE equipment allows testing of the chip in a highly controlled environment
- Uses pre-generated pattern of inputs and compares the output to the known "golden" output values

Teradyne UltraFLex Tester

Initial bringup & validation

- Initial samples are taken to special labs for exhaustive electrical and functional testing
- Chip is placed in a PCB along with other components
- FW and SW are loaded and booted up on the system
- Any issues found are analyzed and debugged
- Power and performance are characterized across different parts
- Major issues may necessitate design changes :-(

Power/Performance characterization

- Major benchmarks and special microbenchmarks are run and settings tuned to get maximum performance across workloads
- Power measurements are analyzed to get optimal point on frequency vs. voltage curve (perf/watt optimization)
- Clock and power gating heuristics are improved to get reduced power under idle or low stress workloads

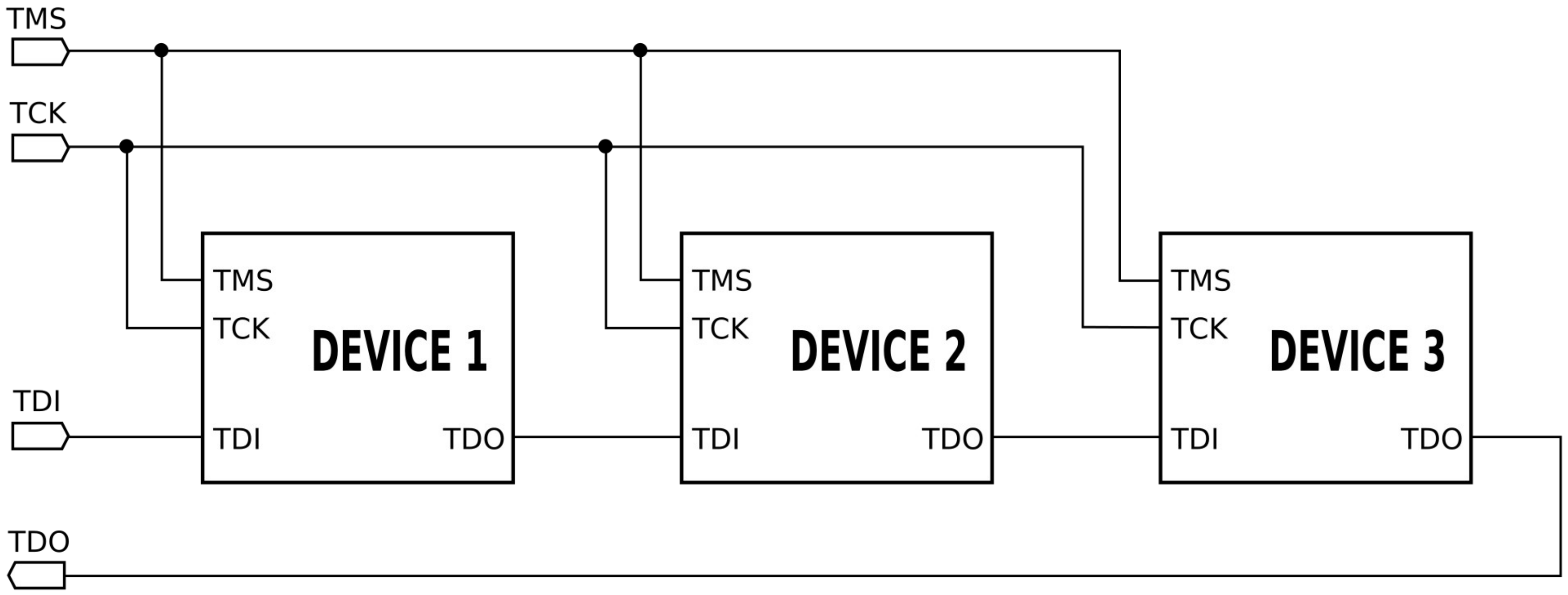
Compliance testing

- Exhaustive checks to ensure adherence to industry (or de-facto) standards
  - USB, DDR, PCIe etc.
  - “Plugfest” held to connect various 3rd party devices
- ISA compatibility: AMD64, RISC-V, MIPS etc.

Debugging Silicon

- Chip is a blackbox – can only look at I/Os
  - Oscilloscopes and logic analyzers can help look at the external I/Os
- JTAG based hardware debuggers help on-chip debugging using DFD features built into the chip:
  - HW breakpoints allow stopping execution on certain events
  - Debug triggers and scan dumps help get better insight into internal state in the chip
- Looking at performance counters in the chip can help identify certain performance bottlenecks and lead to better software optimization

JTAG



- Daisy-chained devices can be individual chips on a PCB or modules within the same chip
- In test mode, functional mode clocks are stopped, TCK clock is fed and TMS is enabled.
- JTAG commands are sent via the TDI pin. Output is observed on the TDO pin

Volume Production

- Automated high-volume testing setup to test EVERY chip prior to shipment
  - ATE testing at various temperature & humidity conditions
  - System-level test for longer software-based tests
- Chips are sorted and binned based on what passed vs failed
  - E.g. a chip that failed some test(s) at 3GHz but passed all tests at 2.8GHz would be binned as a 2.8GHz part
- Failures are analyzed for yield improvement
- Feedback provided to next-generation designs

Shipment

- Chips are packaged and boxed for retail sale
- Also put into trays or reels for automated pick-and-place machines for PCB assembly

More questions?