

## Computer Organization

## ASIC & FPGA

## EDA Design Flow

Dr Jeff Drobman

website



[drjeffsoftware.com/classroom.html](http://drjeffsoftware.com/classroom.html)

email



[jeffrey.drobman@csun.edu](mailto:jeffrey.drobman@csun.edu)

# Index

---

- ❖ VLSI vs ASIC → slide 3
- ❖ FPGA → slide 3
- ❖ Chip Design
  - ❑ Tools → slide 48
  - ❑ Flow → slide 61

# Section

## VLSI vs ASIC

# CPU/GPU vs ASIC

**Quora**

## What is the difference between ASICs and GPUs/CPU's?



**Jeff Drobman**

Lecturer at California State University, Northridge (2016–present) · Just now

in general, ASICs use smaller functional building blocks than “cores”, which are much larger, complete units. ASIC’s are “application specific” — to a single application, and are not *programmable*.

Whereas CPU and GPU cores are software *programmable* — so *general* purpose (not application specific).

# VLSI vs ASIC vs FPGA

➤ 3 options for chips

## ❖ VLSI

### ☐ **Fully** Custom

- *Building blocks:* Designer IP (all levels)
- *Tools:* Licensed from EDA vendors

## ❖ ASIC

### ☐ **Semi** Custom

- *Building blocks:* Manufacturer IP
- *Tools:* Provided by Mfr (ASIC vendor)

## ❖ FPGA

### ☐ **Programmable** Custom (SRAM)

- *Building blocks:* logic gates (NAND, NOR)
- *Tools:* Lab Programmers, software

# Section



FPGA

# FPGA vs PLD

## ❖ FPGA

- ❑ **Field Programmable** (SRAM based)
  - Building blocks: logic gates (NAND, NOR)
  - Tools: Lab Programmers, software

## ❖ PLD

- ❑ **PLA**
  - AMD *Mach* family (merged with PAL)
- ❑ **PAL**
  - MMI invented, bought by AMD
  - AMD spun off as Vantis (bought by Lattice)
- ❑ **CPLD**
  - *Complex* PLD

# Programmable Logic – FPGA

COMP222

From Wikipedia, the free encyclopedia

FPGAs

FPGA

A **field-programmable gate array (FPGA)** is an integrated circuit designed to be configured by a customer or a designer after manufacturing – hence the term *field-programmable*. The FPGA configuration is generally specified using a hardware description language (HDL), similar to that

HDL

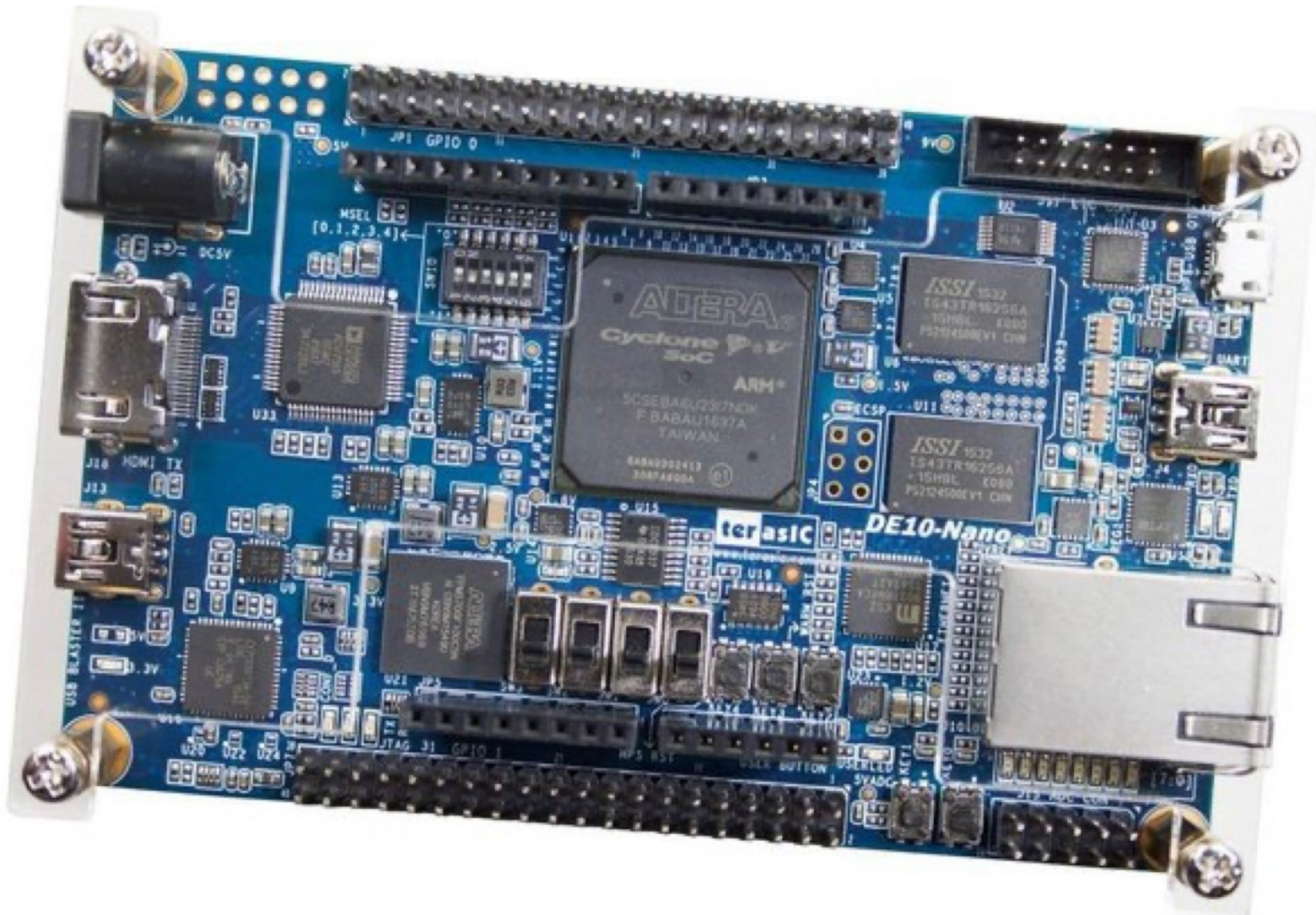


Intel





For example this is generic gaming console built using single FPGA:



# Section

---

## Chip Design

- **Tools**

# Major EDA Tools

## ❖ HDL (RTL level)

❑ Verilog

❑ VHDL

## ❖ Circuits (Analog)

❑ SPICE

VHDL = *VHSIC* HDL

## Top EDA Companies

❖ Cadence

❖ Synopsys

❖ Siemens (bought Mentor Graphics)

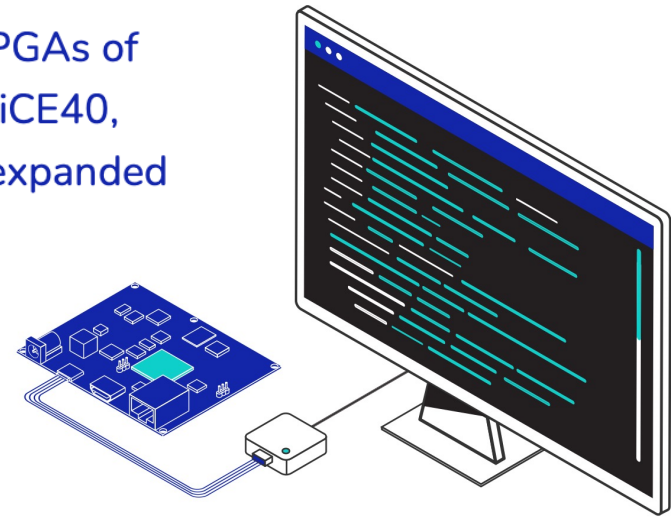
# FPGA Design



## Innovate by reaching for the open source FPGA tooling

F4PGA is a fully open source toolchain for the development of FPGAs of multiple vendors. Currently, it targets the Xilinx 7-Series, Lattice iCE40, Lattice ECP5 FPGAs, QuickLogic EOS S3 and is gradually being expanded to provide a comprehensive end-to-end FPGA synthesis flow.

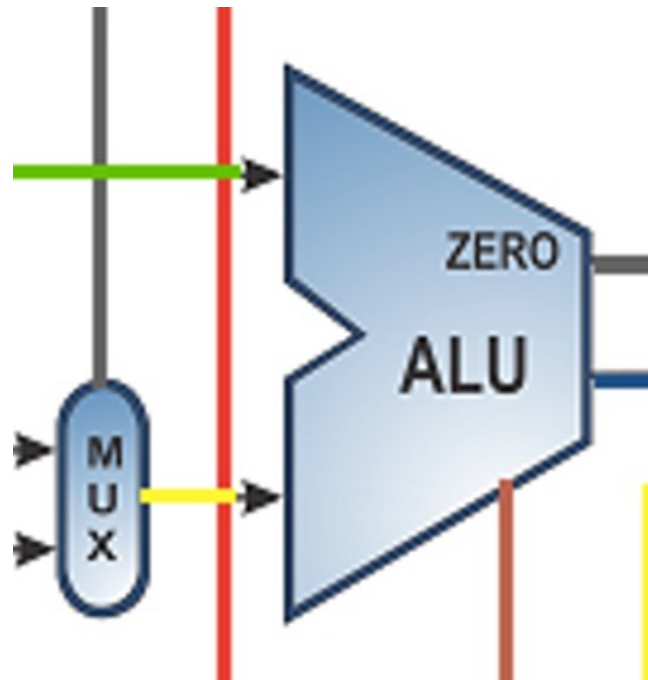
Xilinx (AMD)   Altera (Intel)



# MIPS ALU in Verilog

HDL

**Verilog HDL** is a **hardware description language** used to design and document electronic systems. Verilog HDL allows designers to design at various levels of abstraction. It is the most widely used HDL with a user community of more than 50,000 active designers.



# MIPS ALU in Verilog

HDL

Figure 8.5.15: A Verilog behavioral definition of a MIPS ALU (COD Figure B.5.15).

Structure

```
module MIPSALU (ALUctl, A, B, ALUOut, Zero);  
    input [3:0] ALUctl;  
    input [31:0] A,B;  
    output reg [31:0] ALUOut;  
    output Zero;
```

Behavior

```
    assign Zero = (ALUOut==0); //Zero is true if ALUOut is 0  
    always @(ALUctl, A, B) begin //reevaluate if these change  
        case (ALUctl)  
            0: ALUOut <= A & B;  
            1: ALUOut <= A | B;  
            2: ALUOut <= A + B;  
            6: ALUOut <= A - B;  
            7: ALUOut <= A < B ? 1 : 0;  
            12: ALUOut <= ~(A | B); // result is nor  
            default: ALUOut <= 0;  
        endcase  
    end  
endmodule
```



# Adder Verilog Code



Utsav Banerjee · [Follow](#)

VLSI Design Engineer with Verilog Expertise · 6y



## Related How do I write the verilog code for ripple carry adder?

Let the adder have adder inputs `a` & `b`, carry-in `cin`, and sum output `sout`, and carry-out `cout`. The Verilog code for the ripple carry adder is as follows:

```
1 // 1-Bit Full Adder
2
3 module full_adder (sout, cout, a, b, cin);
4     input  a, b, cin;
5     output sout, cout;
6
7     assign sout = a ^ b ^ cin;
8     assign cout = (a & b) | ((a | b) & cin);
9 endmodule
```



# Adder Verilog Code

```
10
11 // Multi-Bit Ripple Carry Adder
12
13 module rcadder (sout, cout, a, b, cin);
14     parameter INPUT_WIDTH = 16;
15     input [INPUT_WIDTH-1:0] a, b;
16     input cin;
17     output cout;
18     output [INPUT_WIDTH-1:0] sout;
19     wire [INPUT_WIDTH:0] cin_int;
20
```



# Adder Verilog Code



```

21         genvar i;
22
23         assign cin_int[0] = cin;
24         generate
25             for(i = 0; i <= INPUT_WIDTH-1; i = i + 1)
26                 begin: gen_full_adder_cells
27                     full_adder u_full_adder_inst
28                     (.sout(sout[i]), .cout(cin_int[i+1]), .a(a[i]), .b(b[i]),
29                     .cin(cin_int[i]));
30                 end
31             endgenerate
32         assign cout = cin_int[INPUT_WIDTH];
33     endmodule

```

The width of the adder is parameterized. The default value of this parameter is `INPUT_WIDTH = 16`. To instantiate this adder in your design, with a width of your choice, say 32, use the following format:

```

1 rcadder #(.INPUT_WIDTH(32)) u_rcadder_inst (.sout(sout),
    .cout(cout), .a(a), .b(b), .cin(cin));

```

# Free EDA Tools



**Devansh Rajoria** · [Follow](#)

Former Application Engineer at Synopsys (company) (2019–2019) · Updated 4y

Use :-

1. EDA playground if you want online simulation.
2. Icarus verilog for offline simulation and people who work on Linux (do some Google search and you can easily download it)
3. Also, there are limited version of ModelSim and Xilinx ISE design suite which you can use.

\*All are free :)

Hope it helps.




**Joe Zbiciak** · [Follow](#)



Processor architect for ~20 years · 1y

**Related I have the schematic and Verilog code of a microprocessor. What should I do to get it on a chip?**

Either synthesize it for an FPGA and live with this "soft" implementation, or scare up several \$million and contact an ASIC house to put together a physical design derived from your Verilog, and fab it.

Or, if it's a simple and small enough processor that you can personally lay it out by hand, maybe you can send it through [MOSIS](#)'  foundry services. They do multiproject wafers, so if you only want, say, 4 or 8 parts, that's your best bet.

MOSIS

# Z Chip 1: NOT



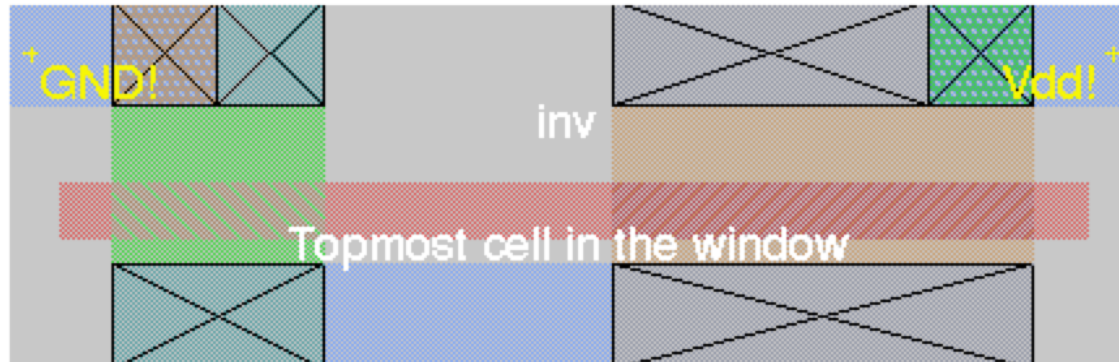
Joe Zbiciak · Follow

Studied Electrical Engineering · Upvoted by Jose Soares Augusto, Instructor on electronics/signal processing/control 30+ yrs · 2y



**Related** What is the gate density in digital electronics?

See this?



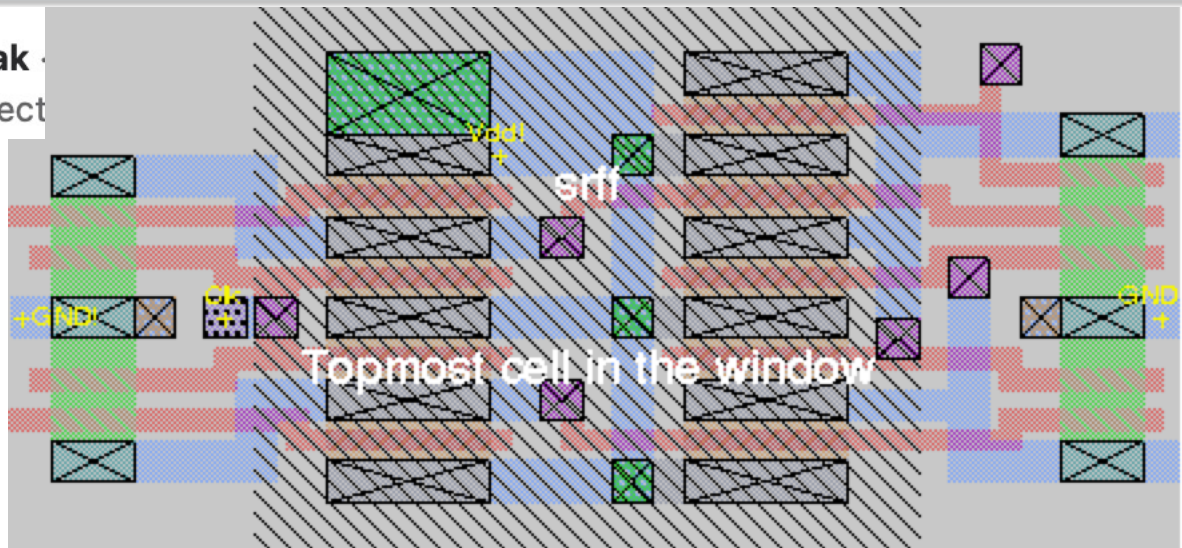
That is a very simple logic gate. It's an inverter that I designed back in 1994 or 1995, in 2 micron technology. It's not much to look at, but that's not the point.

It takes up space. The red line above is 2 microns tall. The whole gate is 14 microns tall. If I counted correctly, its about 38 microns wide. The design rules and electrical characteristics dictate the various sizes of the features and the minimum distances they must have from each other. For example, the N diffusion (green) on the left must be a minimum distance from the P diffusion (brown) on the right. The P diffusion is twice as wide as the N diffusion, because P channel resistivity is twice N channel's.

# Z Chip 2: RS F-F



Joe Zbiciak  
Studied Elect



This is a clocked set-reset flip-flop, consisting of 4 NAND gates: three are 2-input NANDs, and the fourth is a 3-input NAND whose extra input acts as an asynchronous "set" input.

It has a lot more going on per unit area. Still, various structures need to be spaced out in a particular way to stay within the design rules. For example, the blue metal-1 wires need to be at least 3 micron apart. The rectangles with Xs in them represent areas with *vias*, bits of metal that connect various layers. These need to be a multiple of 4 microns in each dimension to make room for the 2x2 "cuts" that allow metal to be deposited between the layers. The shaded area in the middle is N-well diffusion, and that needs to extend some number of microns (6?) beyond the P-diffusion for the P transistors. (The N well is present for the inverter as well; however, I didn't display it in that screen-shot.)



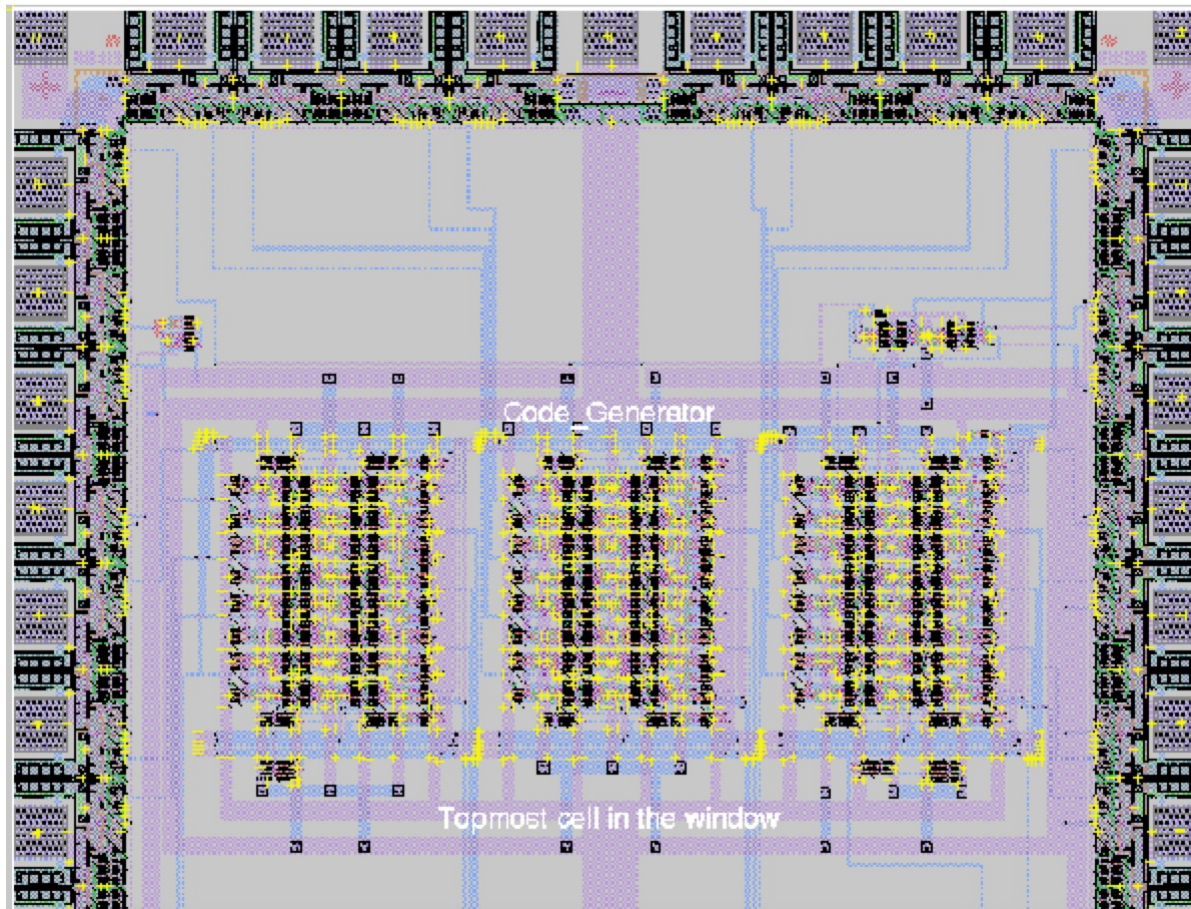
# Z Chip 3



Joe Zbiciak  
Studied Elect

**So, what is gate density?** *It's the number of gates you can fit in a certain square area of silicon.*

Because my design was very regular, and completely laid out by hand, I feel like I got a pretty decent gate density for my first shot at doing this. I probably could have gotten a *little* more density in the final design if I'd "river-routed" the vertical blue buses at the left and right edges of the block above, as I ultimately placed three of these side-by-side:

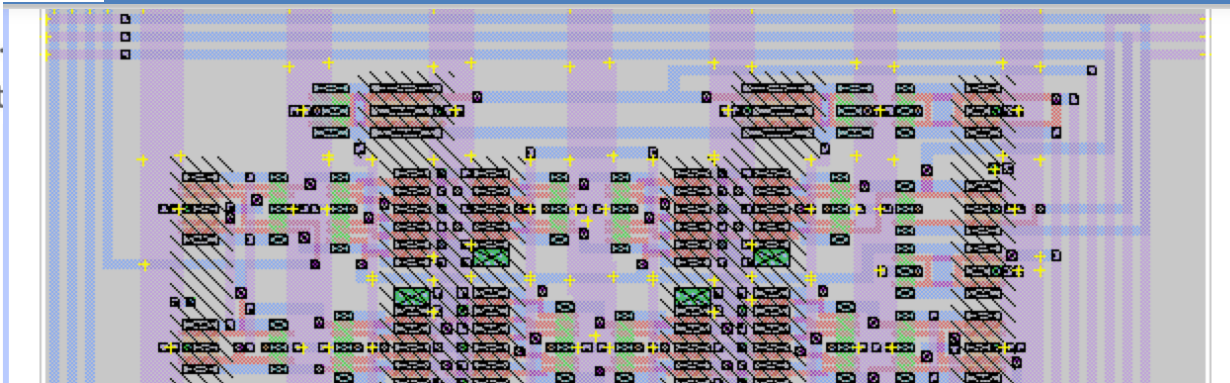






Joe Zbiciak  
Studied Elect

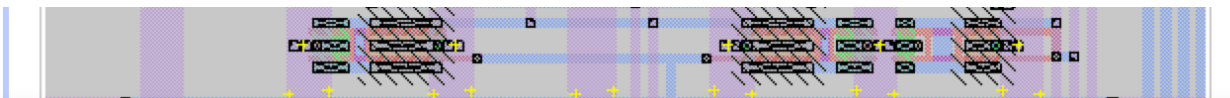
# Z Chip 4: 8 D F-F + XOR



That's 8 D flip-flops, designed as master-slave clocked set-reset flip-flop pairs clocked on opposite clocks, with XOR gates between each D flip-flop, and some clock-drivers at the top and bottom. I realize my D flip-flop is a belt-and-braces design, perhaps, but I had one chance to make this work, so I declined to optimize.



And, there's all the wiring to connect these together. Sometimes that leaves "whitespace":



# Section

---

## Chip Design

- **Analysis**





# ASIC Functional V



---

## Siemens EDA

---

### The State of IC and ASIC Functional Verification

by Daniel Payne on 02-09-2023 at 10:00 am

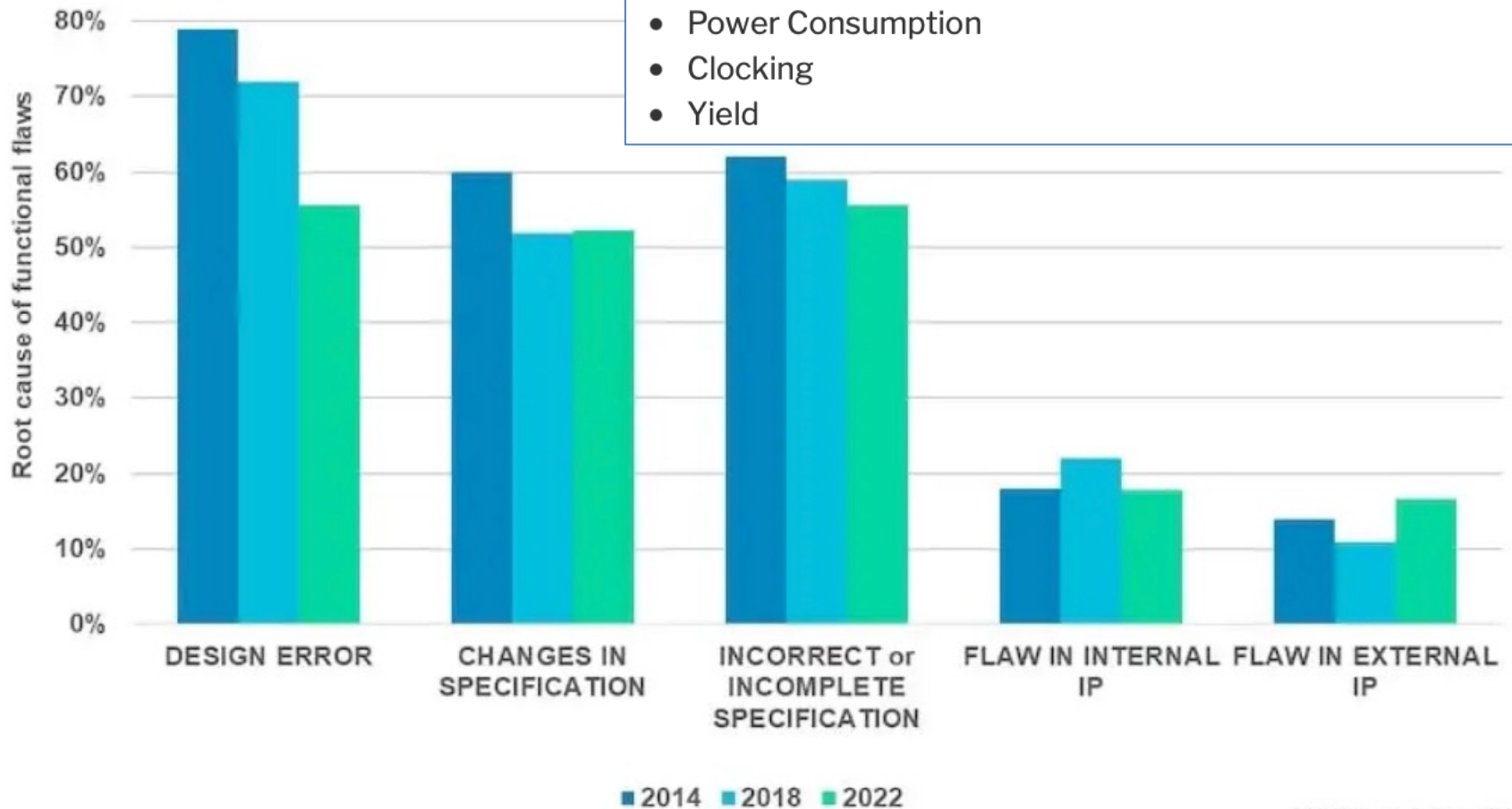
Categories: [EDA](#), [Siemens EDA](#)

Our global semiconductor market had a total value of \$547 billion in 2021, dipping to \$545 billion in 2023, then forecasted to grow to \$635 billion in 2025, according to [IBS](#), where the IC and ASIC segment would reach \$330.9 billion in 2025. Systems

# Design Errors

The top five causes for these silicon re-spins in ranked order are:

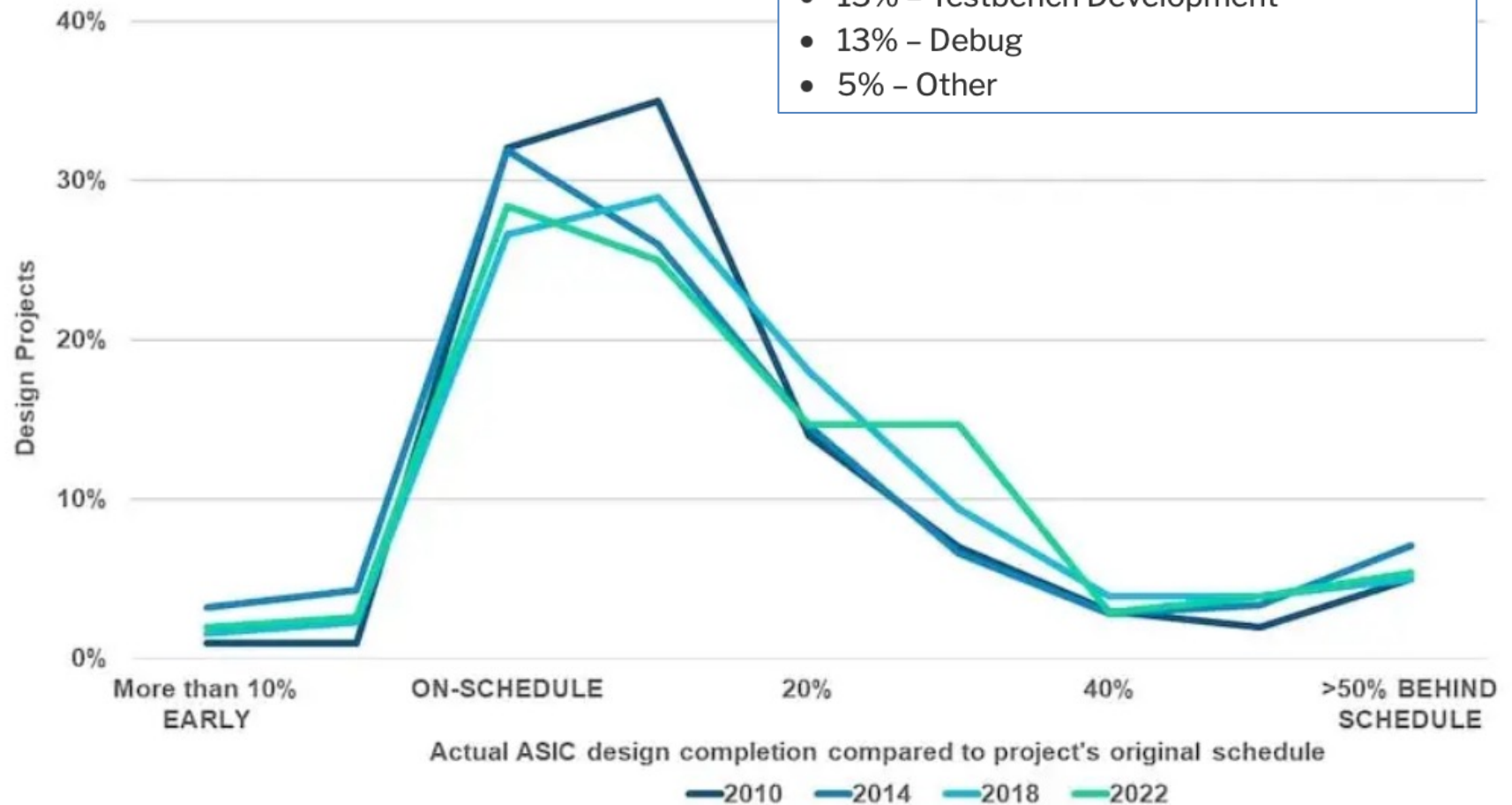
- Logic, functional
- Analog
- Power Consumption
- Clocking
- Yield



\* Multiple replies possible

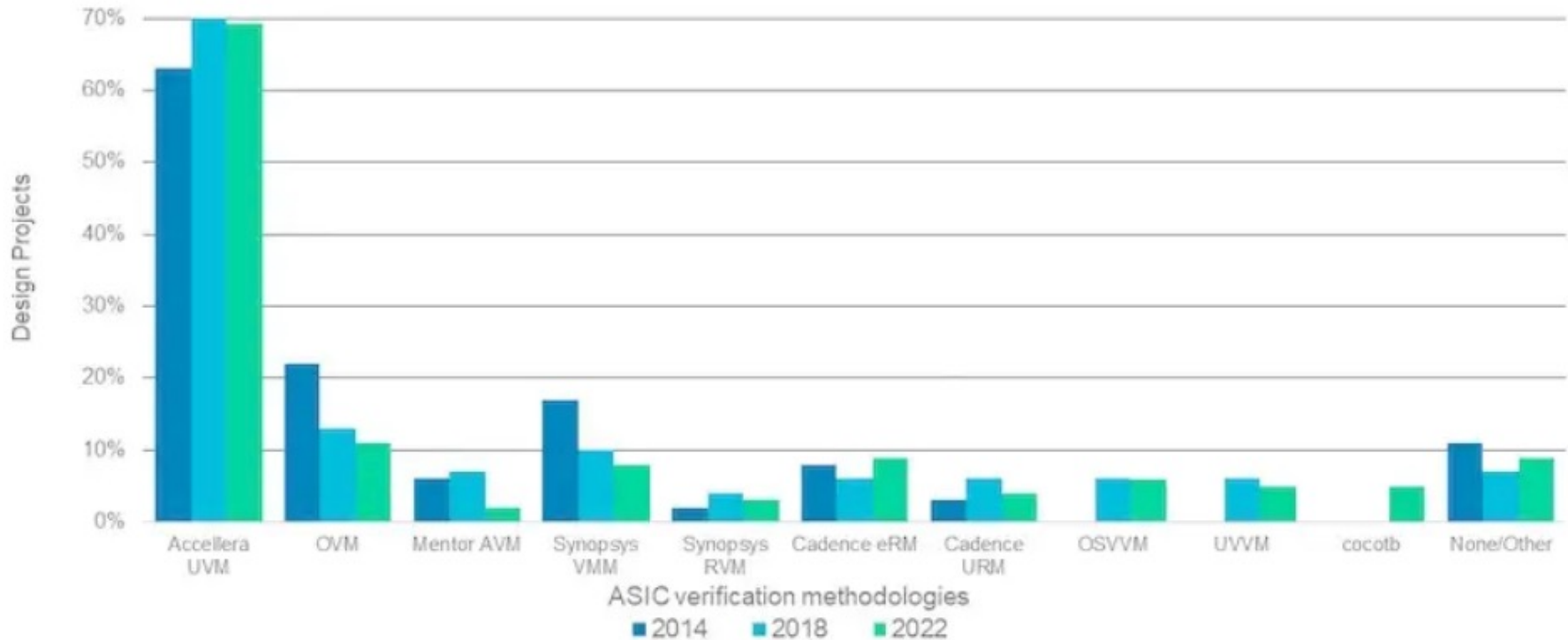
# Project Schedule

- 47% – Test planning
- 21% – Creating Test and Running Simulation
- 15% – Testbench Development
- 13% – Debug
- 5% – Other

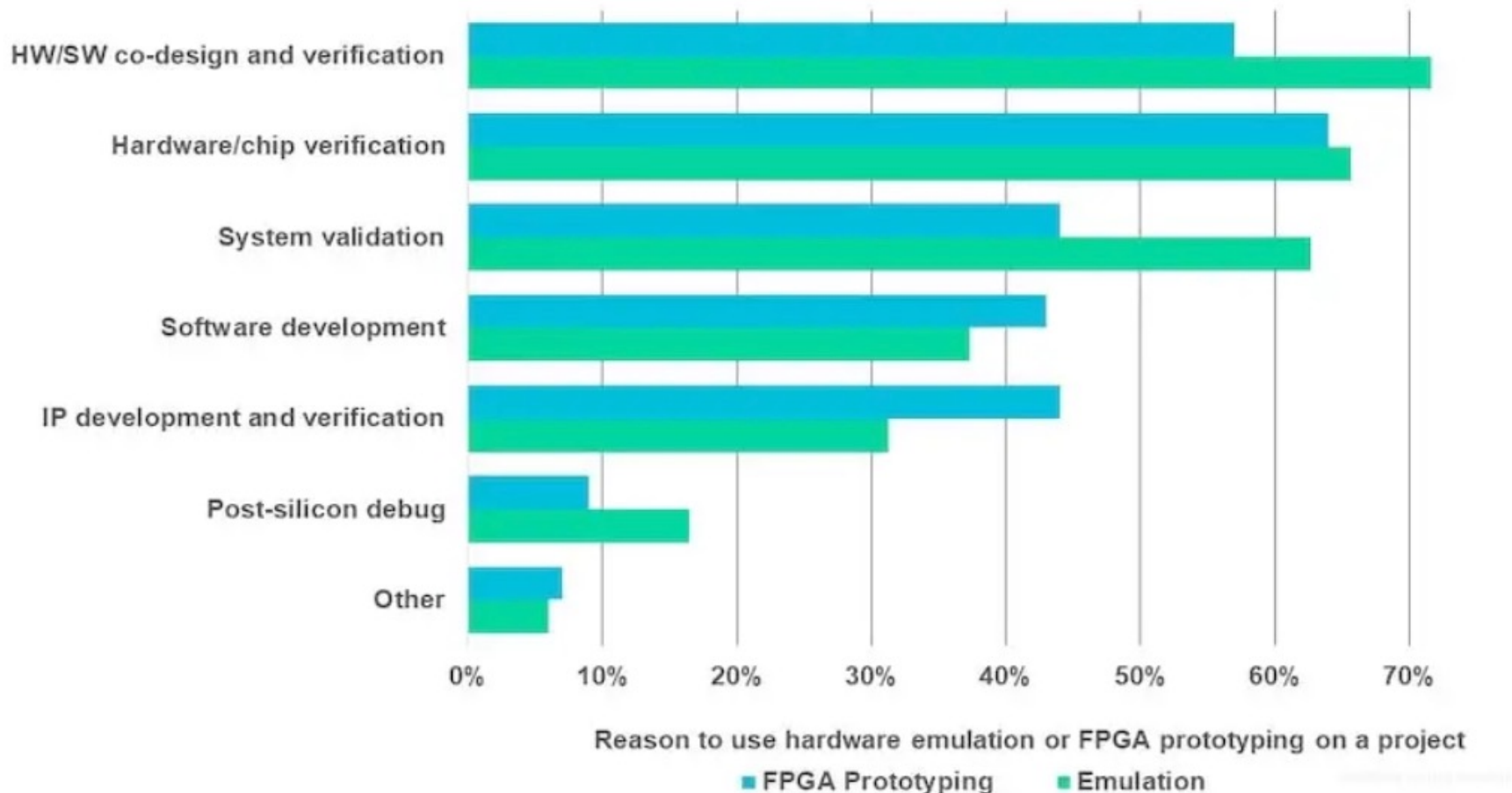


# ASIC Functional V

## ASIC verification methodologies



# FPGA HW Emulation



\* Multiple replies possible

# Section

---

## Chip Design

- **Flow**

# Chip Design Flow

## VLSI & ASIC

1. Low-Level designs
2. Schematic capture
3. RTL: **Verilog** or **VHDL**
4. Behavioral/Functional modeling (**C++**)
5. Circuit simulations: **SPICE**
6. Clock design
7. Electrical analysis
8. Timing analysis
9. Layout/Floor planning
10. Interconnect
11. Packaging
12. Test (die and packaged)
13. Laser out defects (optional)

➤ Using EDA tool sets

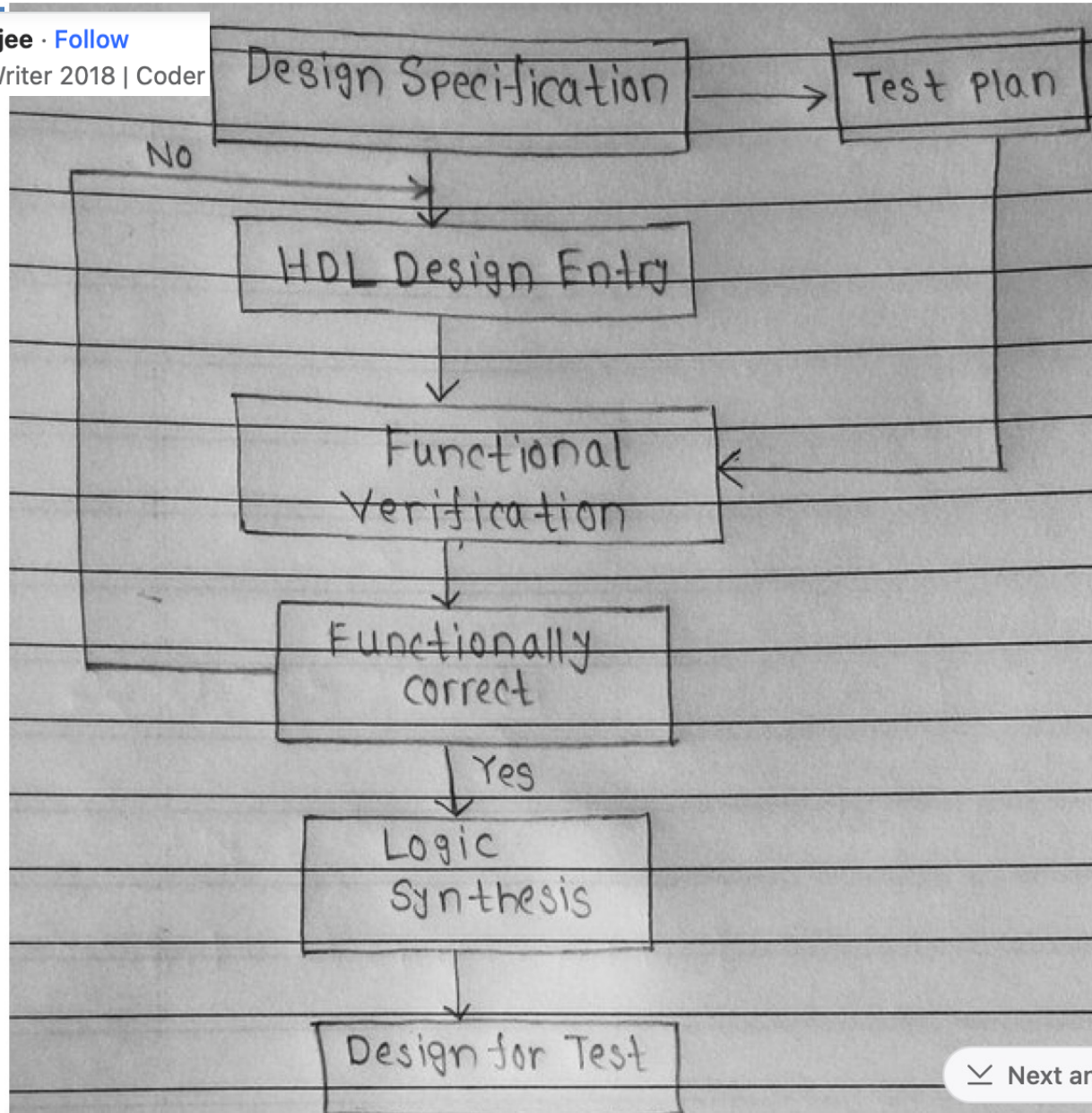


# Chip Design Flow



Sougata Bhattacharjee · Follow

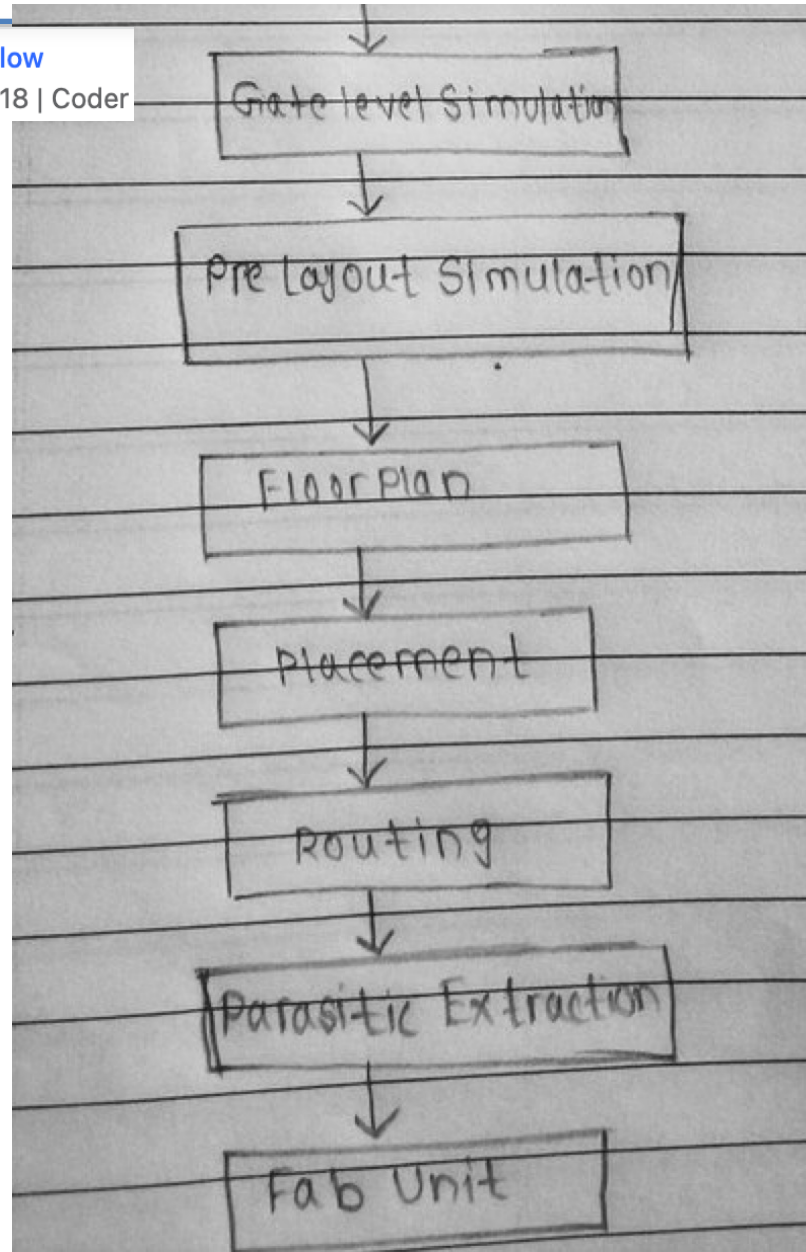
VLSI Engineer | Top Writer 2018 | Coder







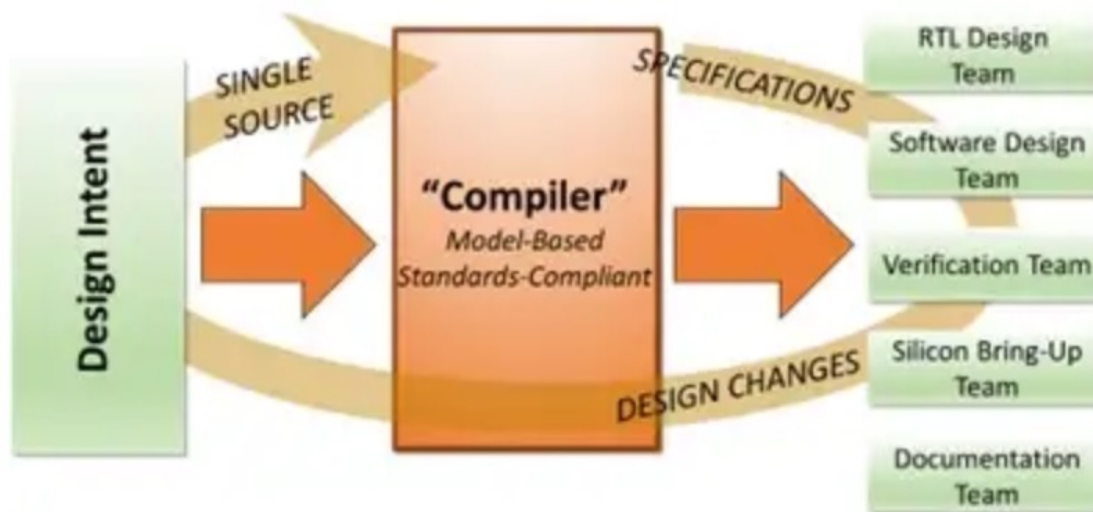
# Chip Design Flow (cont)



# Chip Design Flow

## VLSI & ASIC

The RTL should faithfully implement every nuance of the HSI specification because the slightest infidelity can be catastrophic (read “expensive”), and the software team must be *pro-active participants* in the RTL implementation to enable the richest possible customer experience. And yes, finalizing the HSI is an iterative process, sometimes extending beyond product delivery – so changes to the HSI implementation must be fast, accurate, comprehensive, and include all chip team stakeholder views. Human intervention only adds risk, consumes precious resources, and will assuredly impair productivity.



# Synopsys SoC Environment

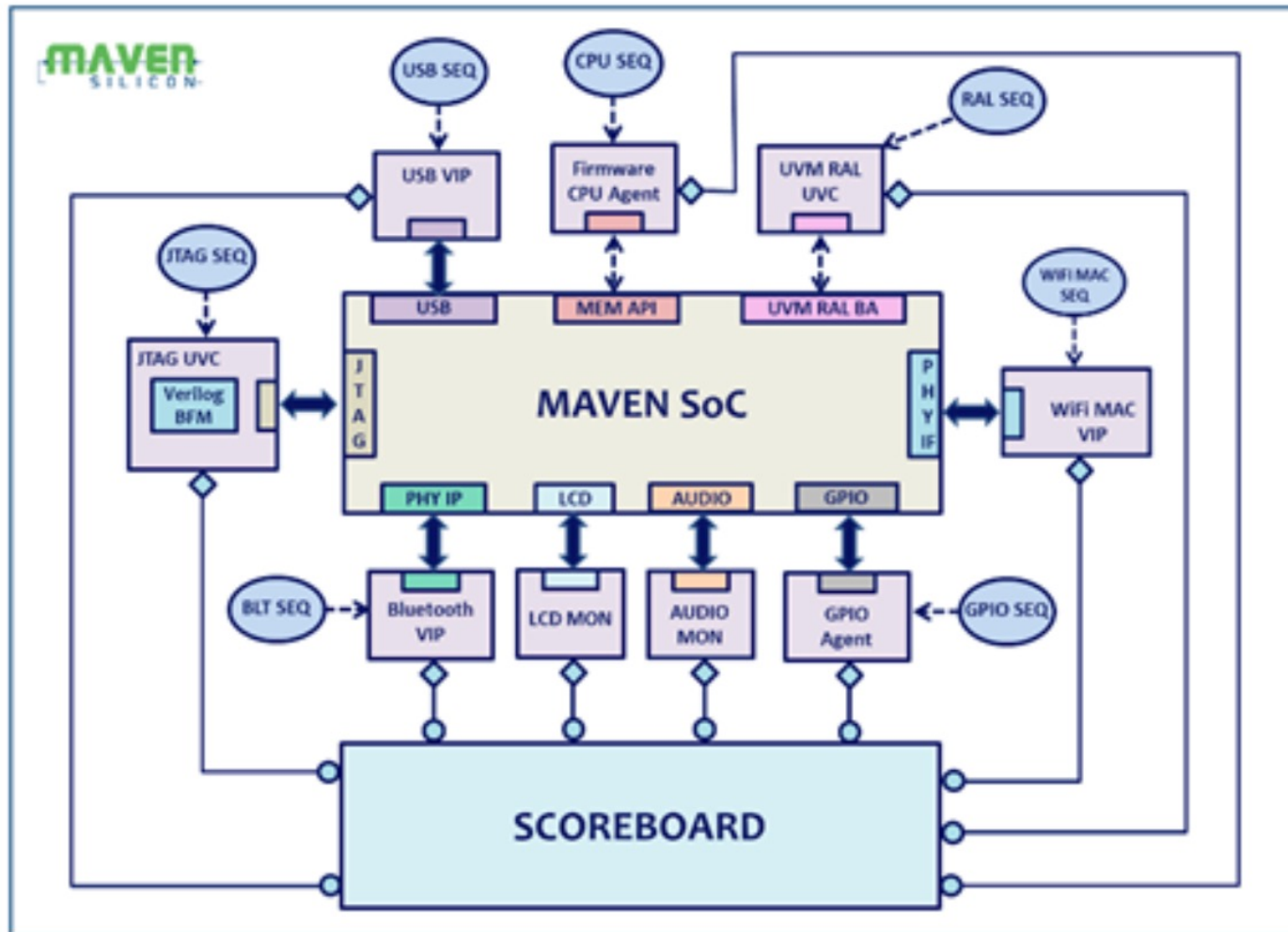
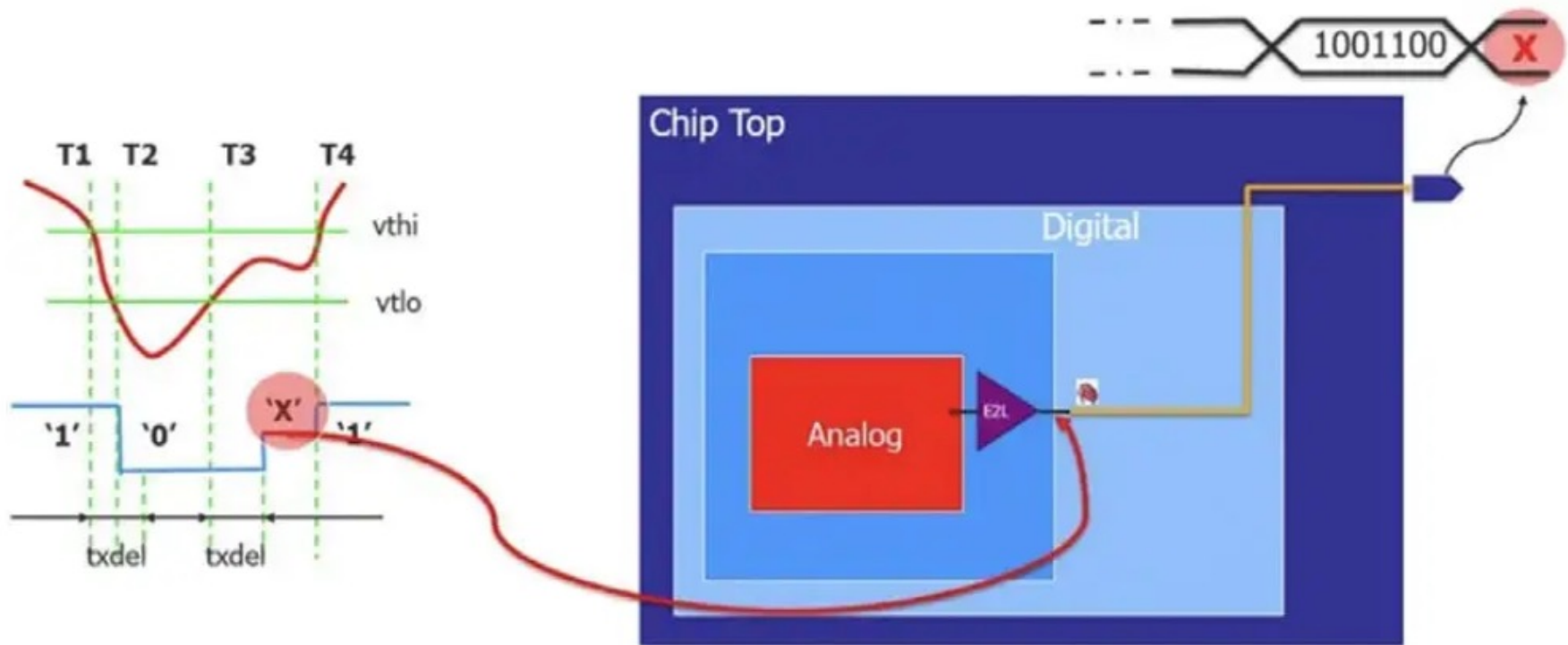


Figure 4: SoC Verification Environment

# Siemens EDA: Mixed Signal

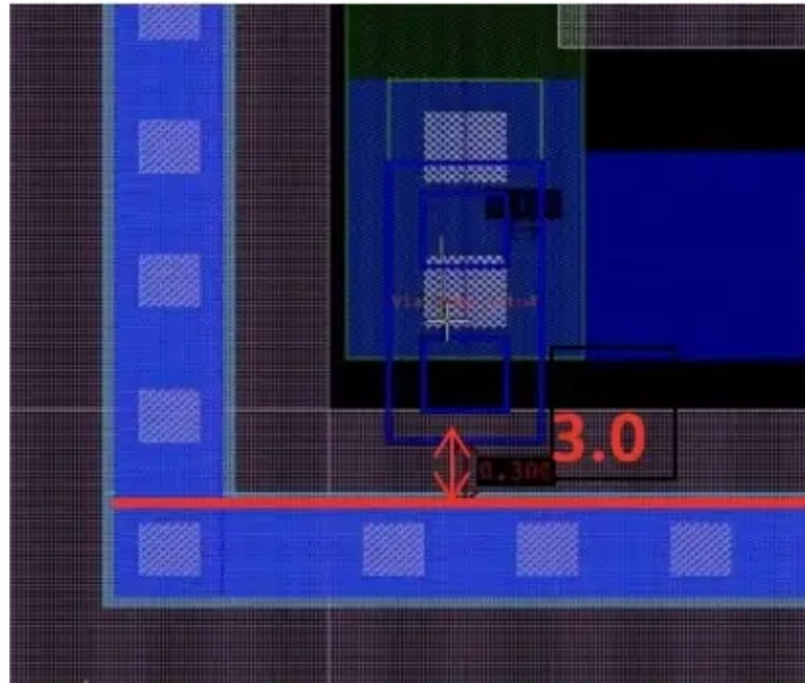


*Mixed-signal waveforms*

# Siemens EDA: Layout

## Custom IC Layout

The companion to S-Edit for layout is dubbed **L-Edit**, and it uses OpenAccess for interoperability, while also supporting PCells for parameterized layout generation. Save time by using Schematic Driven Layout (SDL) along with PCells. Decrease layout verification iterations by using the L-Edit IC Rule-Aware Layout so that you can visually see design rules appear while editing, and it will enforce correct layout.



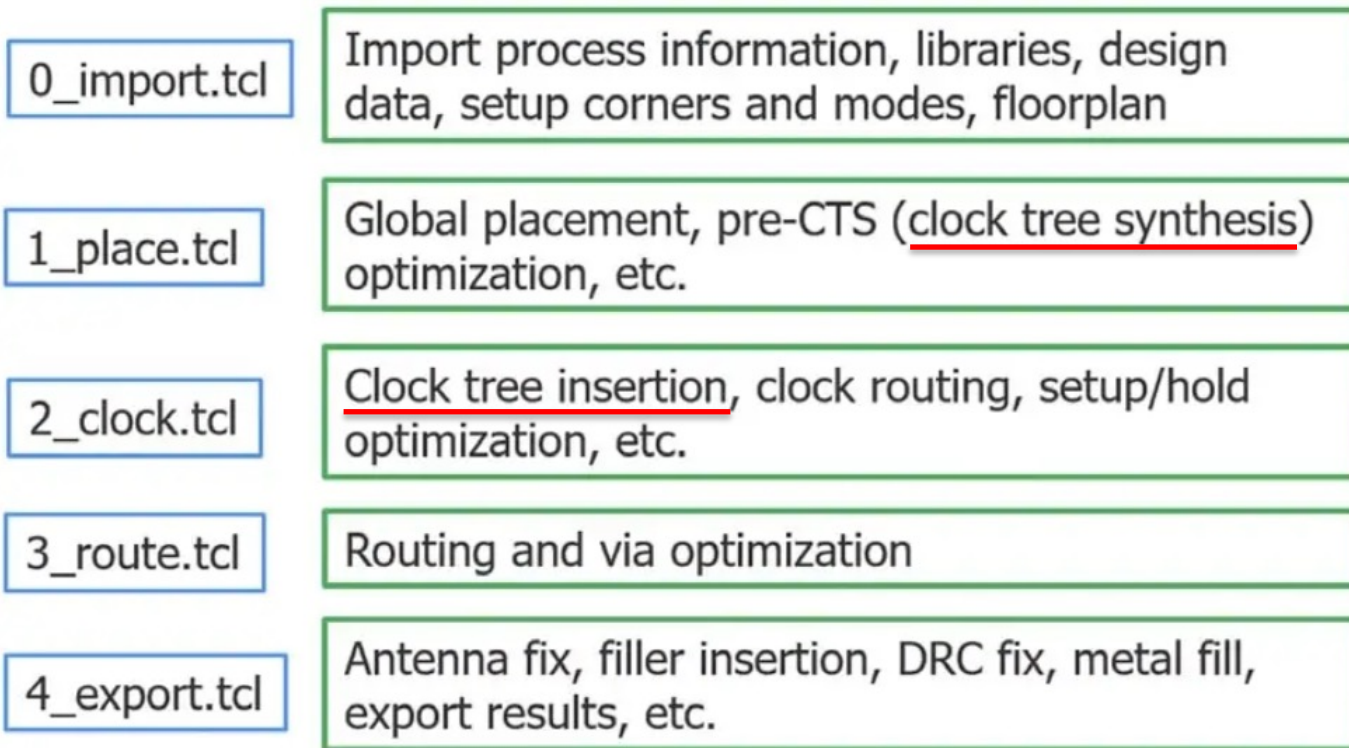
*IC Rule-Aware Layout*



# Siemens EDA: Synthesis

## - Digital Implementation

Physical synthesis comes from the [Oasys-RTL](#) tool, acquired back in [2013](#), then the placement and routing task is completed with the [Nitro](#) tool. Both tools are launched from the GUI in L-Edit, or you can use command line control if preferred. The complex P&R steps are guided by the Nitro reference flow:



*Nitro reference flow*



# EDA Trends

1. **Increased of automation:** VLSI design and verification processes are becoming more complex and time-consuming, making automation essential to improve efficiency and productivity. This includes the use of machine learning and artificial intelligence (AI) to accelerate verification and testing.
2. **Adoption of new design and verification methodologies:** With the increasing complexity of VLSI systems, traditional design and verification methodologies are becoming less effective. New approaches such as high-level synthesis and formal verification are gaining popularity.
3. **Emphasis on security and reliability:** As electronic systems become more interconnected, the need for secure and reliable designs becomes more critical. VLSI design and verification technologies will need to incorporate new techniques to ensure the security and reliability of electronic systems.
4. **Greater use of specialized hardware:** Specialized hardware, such as field-programmable gate arrays (FPGAs) and application-specific integrated circuits (ASICs), are becoming increasingly popular in VLSI design and verification. These specialized devices offer higher performance and efficiency compared to general-purpose computing systems.
5. **Increased use of virtual prototyping:** Virtual prototyping enables the creation of a virtual system model that can be used to simulate and test the design of electronic systems. This approach can reduce development time and costs, while