Semiconductors 101, unit 4

Needs, specs and IC design

Before a chip goes into a fab, it is designed in full on computers and checked many times: a large chip has more than ten billion transistors, far too many to draw or check by hand, so every step relies on design software, and some blocks are even bought. Every step from "what should it do" to handing over the design is there to avoid finding a mistake after tape-out.

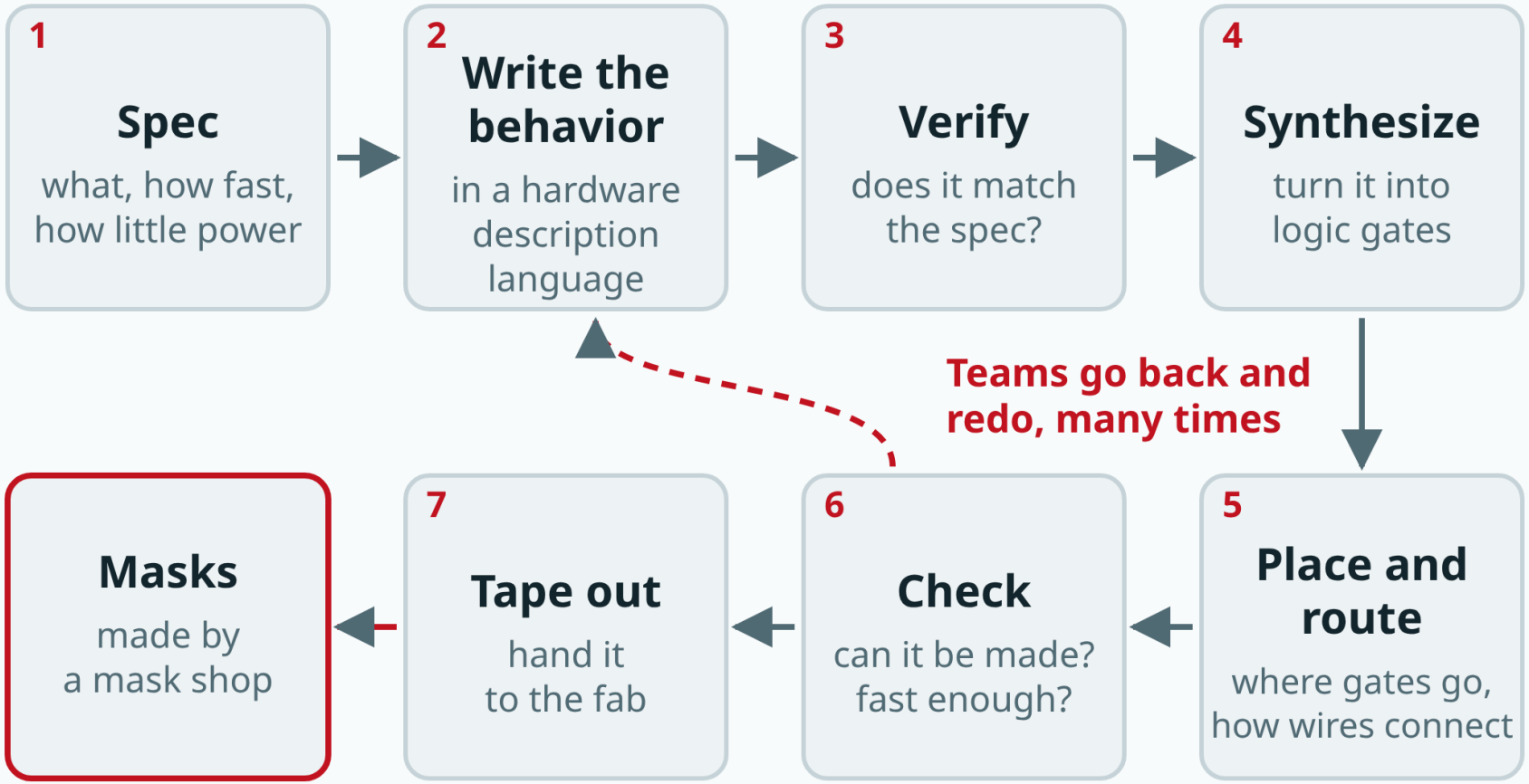
Semiconductors 101: a free course following the life of a chip
Free to use in class and for non-commercial teaching. Please credit fab24hr. Do not sell it or present it as your own work.

Goals

1 Outline the order of digital chip design, from the spec, writing the behavior, verification, synthesis and place and route to sign-off and tape-out, and which kind of design software (EDA) each step uses.

2 Explain how verification is done, setting tests from the spec, checking coverage and proving things with math, and why a mistake found after tape-out is so costly.

3 Know which blocks in a chip are often bought as semiconductor IP, how soft and hard IP differ, and how a fab's rules reach the design through the PDK.



Changing the design after tape-out means new masks: expensive, and months of extra waiting

A chip is designed on a computer and checked many times before tape-out, when it goes out to be made into masks. Teams often go back and redo steps before tape-out; after it, a change means new masks.

Step by step

1 Set the spec: what the chip must do, how fast it runs, how much power it may use at most, how big it is.

Why: Every later step works to it: the software looks for ways to meet its targets, and verification sets its tests from it.

If not: Without a spec, nobody can say whether the design is right, or whether verification is finished.

2 With the fab and process chosen, pick the IP blocks to buy, and check there's a version for that process generation.

Why: Hardly any team draws every block from scratch, and a chip can only be made with one process.

If not: Pick one that doesn't match, and you wait for someone to make a new version, or change fabs.

3 Write the behavior of your own blocks as text in a hardware description language (HDL).

Why: "Add these two numbers on every beat" is one line, while drawn as transistors it's a whole patch of the chip.

If not: Write it wrong, and the software after it will faithfully build the mistake.

Step by step

4 Verify: simulate with tests set from the spec, check coverage, and test the bought IP connected in too.

Why: On a computer you can change one line and try again; actually making a chip takes months.

If not: A bug that survives to tape-out means changing the design and making new masks, and every chip made has the same bug.

5 Synthesize: use the cell library to translate the written behavior into a circuit of connected logic gates.

Why: There are many ways to build the same thing; the software looks up its tables, compares speed, area and power, and picks one that meets the targets.

If not: By hand, nobody could connect them all, let alone compare which way is best.

6 Place and route: place the logic gates, deliver the clock and run wires through several layers, producing the layout.

Why: Placement matters: the longer a wire, the slower the signal and the more power it uses.

If not: Placed badly, wires can't get through or signals arrive late, and it has to be placed again.

Step by step

7 Sign off: check the rules, check the layout matches the circuit, and check signals arrive in time even when the fab makes transistors fast or slow (the corners).

Why: Only when everything passes does the design tape out; the masks are made from these files, and after that it can't simply be changed.

If not: A missed bug shows up only when the chips come back months later, and the only fix is new masks.

The spec is the list of requirements, and the software works to it

Design starts from the spec: what the chip must do, how fast it runs, how much power it may use at most, and how big it is. These often pull against each other: more speed often needs more circuitry and more power; the bigger the chip, the fewer fit on a wafer and the more each one costs; and for something that runs a long time on one small battery, power comes first. A phone chip has more than ten billion transistors; even counting just ten billion, drawing one a second without sleeping or eating would take over three hundred years, so every step relies on a whole set of design software called EDA (electronic design automation). The software doesn't design the chip by itself: engineers give it targets, such as how long each beat may take at most (a chip's circuits move to a steady beat called the clock); it finds a way within those limits and hands back a report, and where something fails, people work out why, change it and run it again.

From written behavior to patterns, layer by layer

A digital circuit's behavior is first written as text in a hardware description language (HDL): "add these two numbers on every beat" is one line, while drawn as transistors it's a whole patch of the chip. A synthesis tool then uses a cell library (building blocks made in advance for this process: logic gates, and small circuits that hold a 0 or a 1, each with its speed and power worked out beforehand) to translate it into a circuit of connected logic gates; there are many ways to build the same thing, and the software looks up its tables and picks one that meets the targets. Place and route then puts the logic gates onto the chip, delivers the clock to every part of it and runs wires through several layers of wiring, producing a pattern for each layer, called the layout; the longer a wire, the slower the signal and the more power it uses. All this software works from a set of files the fab provides, called a process design kit (PDK): the design rules, the transistor models and the data for each layer. If the models are off, however carefully the software calculates, the real chip won't match.

Verification: set tests from the spec, and find what hasn't been tested

Verification checks, before a chip is made, that the design does what the spec says, and the general view is that it often takes a large share of design work. The tests have to come from the spec, not from the circuit the designers wrote, or its mistakes get counted as right answers. Simulation lets a computer pretend to be the chip, feeding in inputs and comparing the outputs with the expected answers, and the computer also makes up lots of random tests; every time the design changes, all the old tests run again. Every answer being right doesn't mean there's no bug; maybe the wrong part just hasn't been tested, so coverage shows which tests are still missing. And the tests can never cover everything: adding two 32-bit numbers alone has 2 to the power of 64 possible inputs, more than five hundred years at a billion tries a second; so for small parts that matter most, formal verification uses math to prove that something always holds, for example that two signals are never switched on together.

Many blocks are bought: semiconductor IP

A large chip has more than ten billion transistors, and hardly any team draws every block from scratch: processor cores, memory interfaces and analog circuits that turn signals like sound into numbers are often bought as semiconductor IP, a block of circuit design made and verified in advance and licensed to others to put inside their own chips; what's sold is the design, not a chip. Soft IP is a circuit description in text and can be built again for another fab; hard IP is an already laid-out pattern, tied to one fab's particular process generation. Analog circuits are usually hard IP: they're sensitive to the size and placement of every transistor, so if someone else rearranges them, they may not work right. When analog circuits shrink, the benefit is small, and sometimes they get harder to make, so many analog chips are made the way chips were made several generations ago. Bought IP was verified on its own, and once it's connected, the two sides may not handle the clock, the power or restarting the same way, so the whole chip is verified again together.

Common misconceptions

People say: Designing a chip is just writing software.

In fact: A hardware description language looks a bit like programming, but a program runs one line after another, while an HDL describes a circuit with many parts working at the same time, mostly to a fixed beat (the clock). And writing it isn't the end: it still has to be verified, synthesized, placed and routed, and pass sign-off before it can be handed over.

People say: Verification, a fab's metrology and chip testing are all checks, so one of them is enough.

In fact: They check different things. Verification happens before tape-out, mostly on a computer, and checks whether the design is right; if it's wrong, every chip is wrong the same way. Metrology and inspection happen in the fab partway through and check whether key steps went right, such as film thickness and line width. Chip testing usually happens once after the wafer is finished and once after packaging, on every chip, and checks whether this chip, as made, is faulty. Think of building houses: check the blueprint before work starts, check the walls are straight while building, and turn on the taps and lights in every house before handover.

Play

Start at chapter 1, "How to Split 12": make the corners the design was simulated at for real, and see whether the design holds up

Scan with a phone to play from here

fab24hr.com/en/?story=1&utm_source=slides

Think

1 A small hearing-aid company is designing its own chip for the first time. The analog circuit that turns the microphone's sound into numbers and the clock circuit that sets the beat for the whole chip could be drawn in-house or bought as ready-made IP. How would you decide which to do yourself and which to buy? What would you ask the seller before buying?

2 Verification takes a lot of people and time, but doing less of it risks finding a mistake after tape-out. If the schedule were tight, how would you decide that "this is enough checking"?

Read

How are chips designed, and what does IC design involve?

fab24hr.com/en/blog/chip-design/

What is semiconductor IP, and why do chips contain circuits someone else designed?

fab24hr.com/en/blog/silicon-ip/

What is EDA, and what does chip design software do at each step?

fab24hr.com/en/blog/what-is-eda/

What is chip verification, and how do you know a design is right before tape-out?

fab24hr.com/en/blog/chip-verification/

Further

What is an SoC, the "system on a chip" in your phone?

Analog vs digital chips: what's the difference?

VLSI design vs working in a fab: what's the difference?

Semiconductors 101: a free course following the life of a chip

https://fab24hr.com/en/learn/design/

Written from public textbooks, industry standards, university course notes and papers; the companies, people, machine numbers and figures in the game are fictional.

Free to use in class and for non-commercial teaching. Please credit fab24hr. Do not sell it or present it as your own work.

Generic illustration made with AI help from public sources; may contain mistakes; not any company's tool