From Physics to Logic

This course aims to introduce you to the layers of abstraction of modern computer systems. We won't spend much time below the level of bits, bytes, words, and functional units, but I think you should at least be aware of the richness that exists below this level and what implications it has for the higher layers.

Simulating Classical Physics With Quantum Physics

Modern computer systems are based ultimately on quantum physics. However, they use the quantum effects of materials essentially to simulate classical physics. Most mathematical theories of computation also assume classical physics. How we built computers and how we think about computers have matched for over half of a century. To a reasonable approximation, all current computers can be thought of as Universal Turing Machines, we can think of UTMs in terms of a Newtonian game of billiards, and we can implement that game by exploiting the quantum physics of certain materials.

Starting in the early 1980s, models of computation based directly on quantum physics have been developed. We don't know yet, but these models may be more powerful than UTMs. In the mid 1990s, Peter Schor published an algorithm that factored large numbers into primes very quickly on a computational model known as a Quantum Turing Machine. That this task is believed to be difficult on classical computers (albeit without proof!) under-girds almost all of cryptography, and thus Schor's result has generated much interest. We also don't know if it is feasible to build actual quantum computers, but several groups are spending lots of money to figure out if it is possible.

The point of telling you this is to let you know that the landscape of systems (not to mention the entire world!) will change drastically if quantum computation turns out to be more powerful than classical computation and if quantum computers turn out to be implementable.

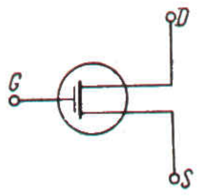
Semiconductors

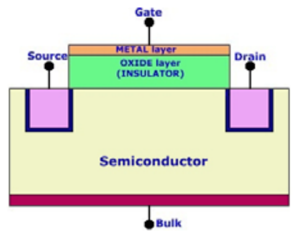
The materials from which processors, memory, and other devices are constructed are known as semiconductors. While semiconductors are created using very mundane materials (silicon is the main ingredient of sand!), sophisticated chemistry is used to "grow" giant perfect silicon crystals and to subtly "dope" them with "impurities" in order to carefully control their properties. In addition to semiconductors, ultra-pure conductors, often created using copper, and insulators, typically created using silicon oxide (sand rust!) are the core materials of computers.

Transistors

Semiconductors, insulators, and conductors are used to construct passive electronic components such as wires, resistors, capacitors, and inductors (rarely). However, the most important component built from semiconductors is an active one, the transistor. Transistors are three terminal devices, and there are many kinds. In the MOSFET, which is the typical kind of transistor in modern computer chips, the voltage on one terminal (the gate) controls how difficult it is send electrical current between two other terminals (the source and drain).

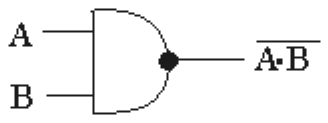
In the nonlinear circuitry of computer chips (as opposed to the linear circuitry of , say, an audio amplifier), we usually use the gate to "turn on" and "turn off" the flow of current, like a simple valve. Below, you can see the symbol of a MOSFET and a cross-section of its typical construction.

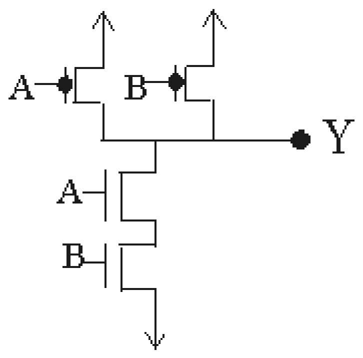




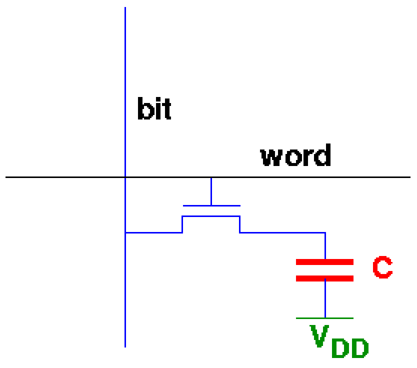
Logic and Memory

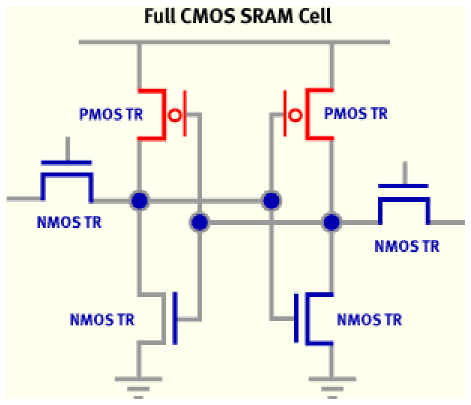
Using transistors and capacitors, we can create the combinational logic and memory that are the basis of computers. For example, the following shows the symbol for a NAND at the left and its implementation using MOSFETS at right. The solid circles (often shown as open circles as well) represent inversion. Basically, a circle at the gate of a mosfet indicates that as gate voltage decreases, it gets easier to send current from source to drain. Without a circle, decreasing voltage makes it gets harder. Using NAND gates, we can build all other combination logic gates.





We can also use transistors (and other components) to build memory cells. Below, at left we see a DRAM cell, which is what main memory on most computers consists of. At right, we see an SRAM cell, which is the typical cell in cache memories, processor register files, and other components. There is a third kind of memory cell, a latch, which is widely used in implementing processor pipelines.





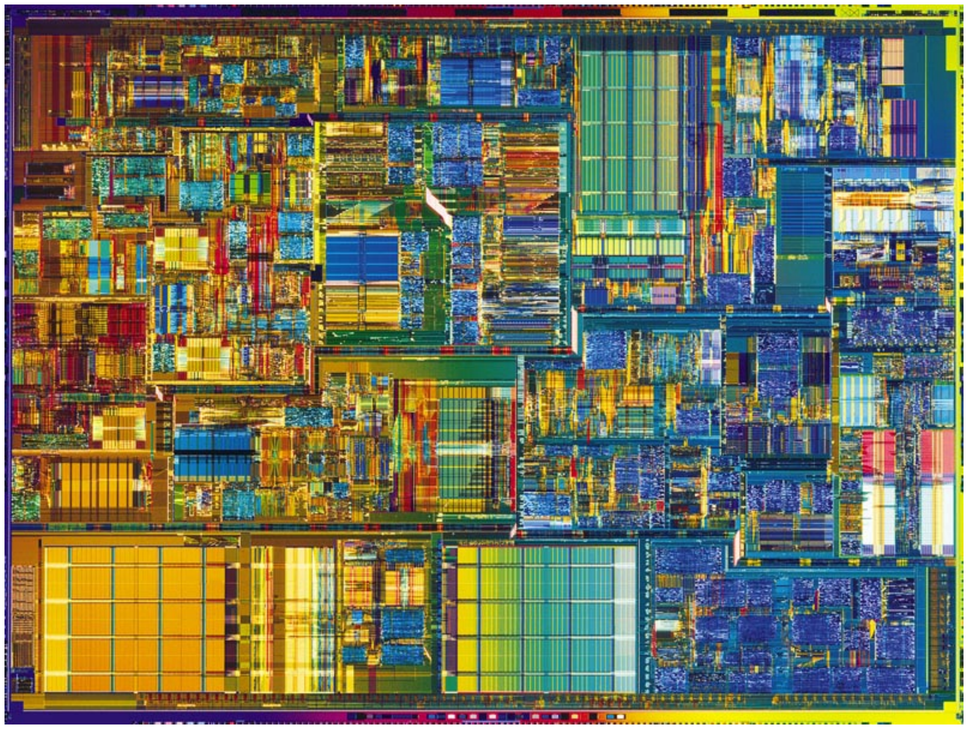
An important thing to notice is that an SRAM cell uses no capacitors but uses six transistors instead of just one. SRAM cells are much faster than DRAM cells, but also much more expensive. That's why they tend to be put in caches. An SRAM cell will also hold its bit as long as there is power. In a DRAM cell, because the bit is held in a capacitor and capacitors always leak charge, the bit must be frequently refreshed by external circuitry.

Photolithography and Chips

Transistors were first developed in the late 1940s. By the late 1950s, engineers had learned how to mass-produce individual transistors inexpensively. Around this time, Jack Kilby, an engineer at Texas Instruments, had a vision of putting building multiple transistors and passive devices on a single semiconductor substrate. He developed the first monolithic integrated circuit (IC or "chip"), inventing a technology that is still going strong today.

A chip is produced using a process called photolithography, which you can think of as a strange kind of photographic printing. If you've ever done black-and-white printing in a darkroom, you understand the gist of it already. Essentially, a negative called a mask is produced that is the inverse of the layer of material to be put on the semiconductor substrate. The substrate is coated with the material that will be deposited and then covered in a material called photoresist. The mask is then placed over the photoresist and a bright light (ultraviolet light these days) is shown on it. Next, the photoresist is developed, and then the unexposed photoresist is etched away (like fixing a photographic print). Then the next layer can be created over the current one. Chips are built up out of multiple layers like this.

Below you'll see a bird's eye view of the Intel Pentium 4 processor. This whole chip has an area of only 217 square millimeters (about the size of the nail on your big toe) yet contains about 42 million transistors.



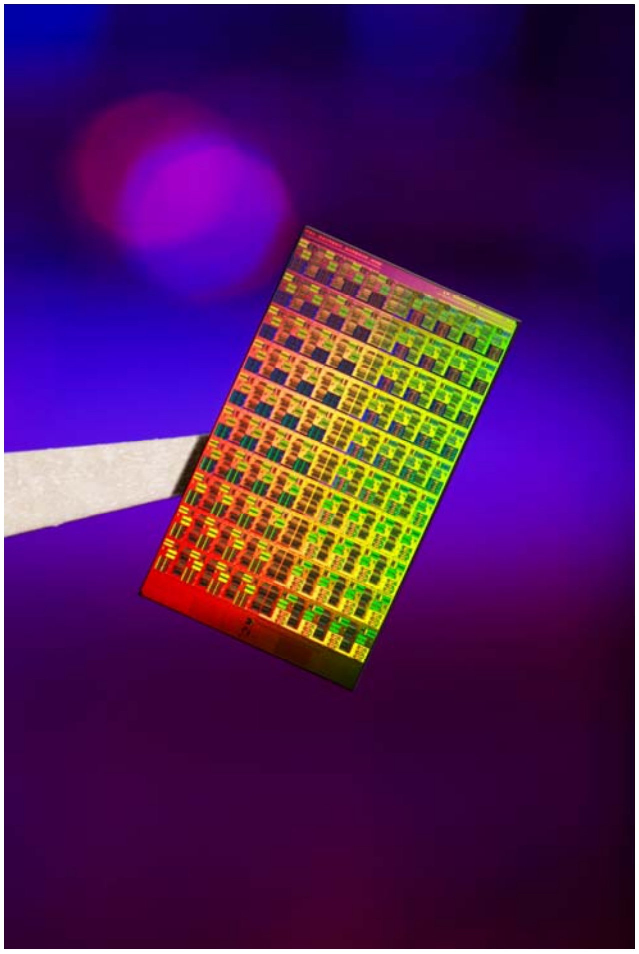
Moore's Law, Its Limits, And So Many Transistors

In the late 1960s, Gordon Moore made a plot of the number of transistors on a chip as a function of the year the chip came out and found that it was exponential. Every 18 months, the number of transistors would double, and chips would become much faster because the transistors were smaller. He predicted that this would continue, and he was right. Moore's Law still holds true, except now the doubling period is just one year. Experts expect that it will continue for at least another 10 years. The current ways that we build processors will not be able to make use of this many transistors, and thus there is much current research on how to actually exploit having billions of transistors on a chip.

Multi-core Processors and Parallelism

An important recent shift in processors has been the advent of the multi-core era. In a multi-core processor, the chip contains multiple copies of the processor. Even commonplace desktop microprocessors today have four cores (four copies of the processor). It is expected that since the number of transistors on chip will continue to scale exponentially for some time, the consequence is that the number of cores (copies of the processor) per chip will also grow exponentially for some time, probably doubling every year or two. This means that a 64 core chip will probably be commonplace in four years or so.

Each individual core of a multi-core chip is unlikely to be much faster than a current core, however. For deep technical reasons, we are shifting away from an era (the single core era) in which the increasing number and speed of transistors let us make processors that ran existing code ever faster, to a new era (the multi-core era) in which existing code will not automatically get faster. Having 64 cores does not mean that existing programs will get 64 times faster. To make programs that run even twice as fast will require significant changes in how algorithms, languages, and compiler-toolchains are built. Above all, making code performance scale with the number of cores will require significant intellectual effort on the part of the programmer, and create opportunity for programmers as well. The following image is of an experimental 80 core chip developed by Intel



A multi-core chip is a parallel computer, meaning that it can do several things simultaneously at the hardware level. Parallel computers have been around for over 30 years, but the focus of research in parallel systems to this point has largely been to support scientific computations. With parallel computers on everyone's desktop and in everyone's cell phone, existing and new ideas from parallel systems will need to be employed. The challenge is how to feed the parallelism. In the case of the above chip, the programmer may have to think about what 80 things the chip should be doing simultaneously.