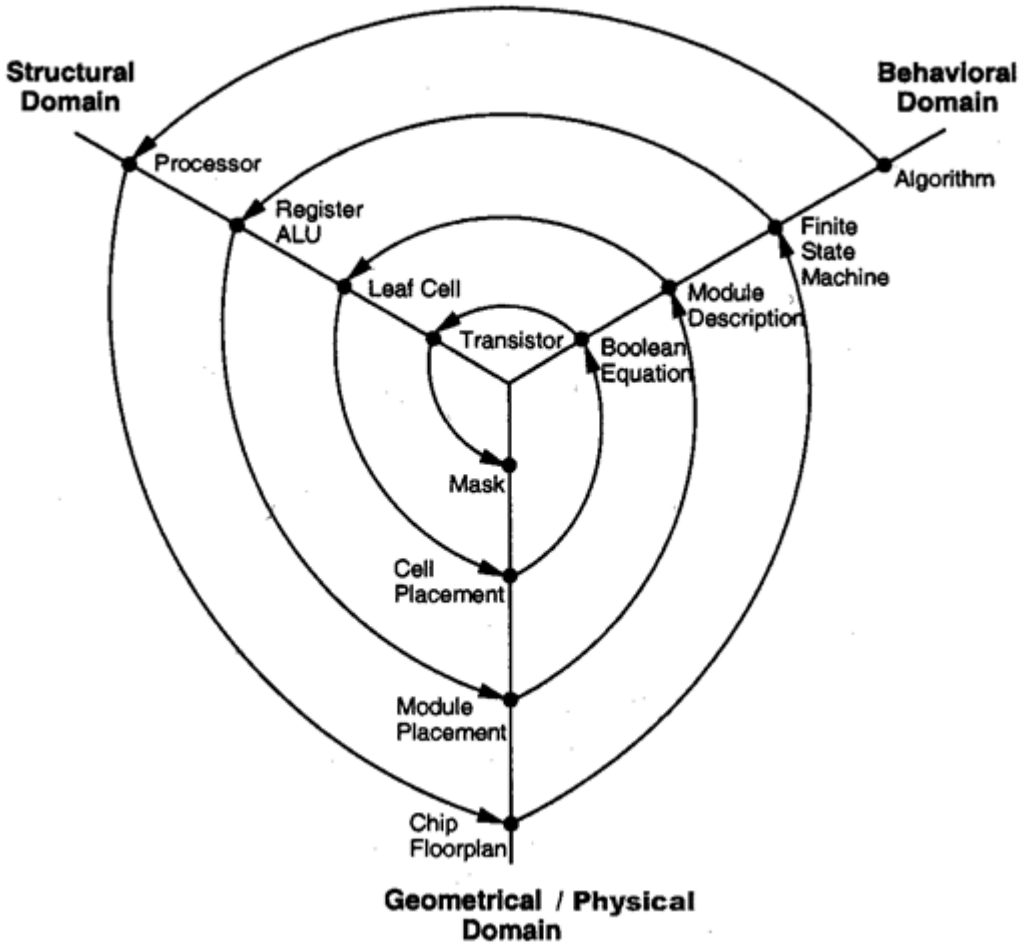
Design Domains, Modeling, Design Flows (Spiral), Hierarchy and Levels

From CMOS VLSI Design Textbook – Weste and Harris – mainly digital VLSI



From Rabaey et al CMOS VLSI textbook – mainly digital

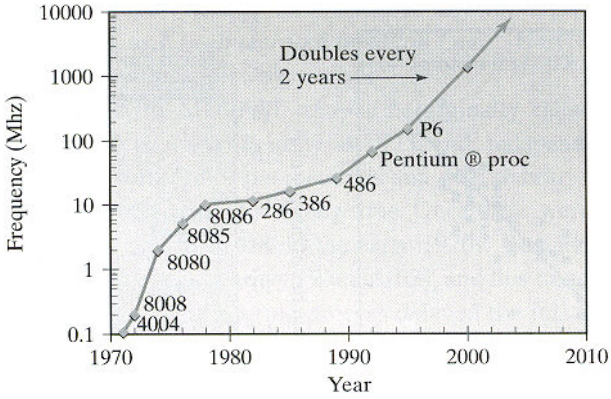


Figure 1-4 Microprocessor performance trends at the beginning of the 21st century. (Courtesy of Intel.)

The million-transistor/chip barrier was crossed in the late 1980s. Clock frequencies double every three years in the past decade and have reached into the GHz range. This is illustrated in Figure 1-4, which plots the microprocessor trends in terms of performance at the beginning of the 21st century. An important observation is that, as of now, these trends have not shown any signs of a slowdown.

It should not surprise the reader that this revolution has had a profound impact on how digital circuits are designed. Early designs were truly handcrafted. Every transistor was laid out and optimized individually and carefully fitted into its environment. This is adequately illustrated in Figure 1-5a, which shows the design of the Intel 4004 microprocessor. Obviously, this approach is not appropriate when more than a million devices have to be created and assembled. With the rapid evolution of the design technology, time to market is one of the crucial factors in the ultimate success of a component.

As a result, designers have increasingly adhered to rigid design methodologies and strategies that are more amenable to design automation. The impact of this approach is apparent from the layout of one of the later Intel microprocessors, the Pentium® 4, shown in Figure 1-5b. Instead of the individualized approach of the earlier designs, a circuit is constructed in a hierarchical way: a processor is a collection of modules, each of which consists of a number of cells on its own. Cells are reused as much as possible to reduce the design effort and to enhance the chances for a first-time-right implementation. The fact that this hierarchical approach is at all possible is the key ingredient for the success of digital circuit design and also explains why, for instance, very large-scale analog design has never caught on.

The obvious next question is why such an approach is feasible in the digital world and not (or to a lesser degree) in analog designs. The crucial concept here, and the most important one in dealing with the complexity issue, is *abstraction*. At each design level, the internal details of a complex module can be abstracted away and replaced by a *black-box view* or *model*. This model contains virtually all the information needed to deal with the block at the next level of hierarchy.

For instance, once a designer has implemented a multiplier module, its performance can be defined very accurately and can be captured in a model. In general, the performance of this multiplier is only marginally influenced by the way it is utilized in a larger system. For all purposes, therefore, it can be considered a black box with known characteristics. As there exists no compelling need for the system designer to look inside this box, design complexity is substantially reduced. The impact of this *divide-and-conquer* approach is dramatic. Instead of having to deal with a myriad of elements, the designer has to consider only a handful of components, each of which are characterized in performance and cost by a small number of parameters.

This is analogous to a software designer using a library of software routines such as input/output drivers. Someone writing a large program does not bother to look inside those library routines. The only thing he cares about is the intended result of calling one of those modules. Imagine what writing software programs would be like if you had to fetch every bit individually from the disk and ensure its correctness instead of relying on handy "file open" and "get string" operators.

Abstraction levels typically used in digital circuit design are, in order of increasing abstraction, the device, circuit, gate, functional module (e.g., adder) and system levels (e.g., processor), as illustrated in Figure 1-6. A semiconductor device is an entity with a very complex behavior. No circuit designer will ever seriously consider the solid-state physics equations governing the behavior of the device when designing a digital gate. Instead, he will use a simplified model that adequately describes the input/output behavior of the transistor. For instance, an AND gate is adequately described by its Boolean expression ($Z = A.B$), its bounding box, the position of the input and output terminals, and the delay between the inputs and the output.

This design philosophy has been the enabler for the emergence of elaborate *computer-aided design* (CAD) frameworks for digital integrated circuits—without it the current design complexity would not have been achievable. Design tools include simulation at the various complexity levels, design verification, layout generation, and design synthesis. An overview of these tools and design methodologies is given in Chapter 8.

Furthermore, to avoid the redesign and reverification of frequently used cells, such as basic gates and arithmetic and memory modules, designers most often resort to *cell libraries.* These libraries not only contain the layouts, but also provide complete documentation and characterization of the behavior of the cells. The use of cell libraries is, for instance, apparent in the layout of the Pentium® 4 processor (Figure 1-5b). The integer and floating-point unit, just to name a few, contain large sections designed using one particular cell-based approach, called *standard cell.* Logic gates are placed in rows of cells of equal height and interconnected using routing channels. The layout of such a block can be generated automatically given that a library of cells is available.

The preceding analysis demonstrates that design automation and modular design practices have effectively addressed some of the complexity issues incurred in contemporary digital design. This leads to the following pertinent question: If design automation solves all our design problems, why should we be concerned with digital circuit design at all? Will the next-generation

Digital vs. Analog Abstraction Levels

Digital Information Processing	Analog Information Processing
SIGNAL DOMAIN	
Stochastic Signal Detection/Estimation Operations	?
Simple DSP filtering	Op amp filtering functions
Multiply and Accumulate and Registers	Combine Gilbert Cell + Op amps
Large bitword multipliers	Innovative Circuit topologies
Small bitword multipliers	Basic Gilbert Cell
Large bitword adders	Innovative Op amp topologies
Small bitword adders	Various types of Op amp circuits
Single bit adders	?
Latches and Flip/Flops	? some temporary poor storage
Nands, Nors, XORs, Mux	Op Amps, Analog Mux
Inverters	Op Amps
Function vs. Physical Behavior abstraction	Stay in the physical domain
Pull up with pull down transistor circuit behavior	Ratio of load/driver transistors
Transistor and Device equations	Transistor and Device equations

Figure Analog Abstraction Levels - Razavi Analog CMOS

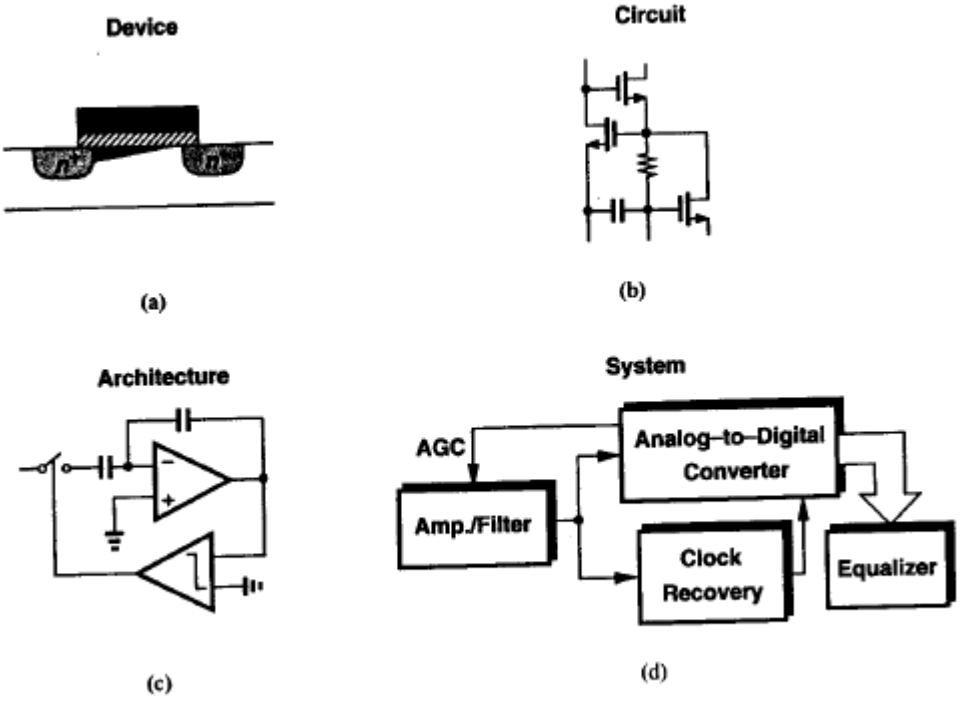


Figure 1.8 Abstraction levels in circuit design: (a) device level, (b) circuit level, (c) architecture level, (d) system level.

Note that above at System Level, the behavioral modeling of the structure is typically similar to signal modeling. For example, quantization modeling in data converters or angle modulation (eg. frequency or phase) in phase lock loops.

Figure Digital Abstraction Levels - Rabaey et al.

High level digital hardware models stay in the data domain (register values) rather than the signal domain. Signal domain models are handled in software algorithms (eg. DSP C code or HDL code) rather than in the hardware models as in analog. This separation (partition, divide and conquer) of signal models from hardware models that exists in digital design, but does not exist in analog design, is one of the key advantages of a digital design approach and explains the ongoing trend of signal processing in the digital domain replacing signal processing in the analog domain.