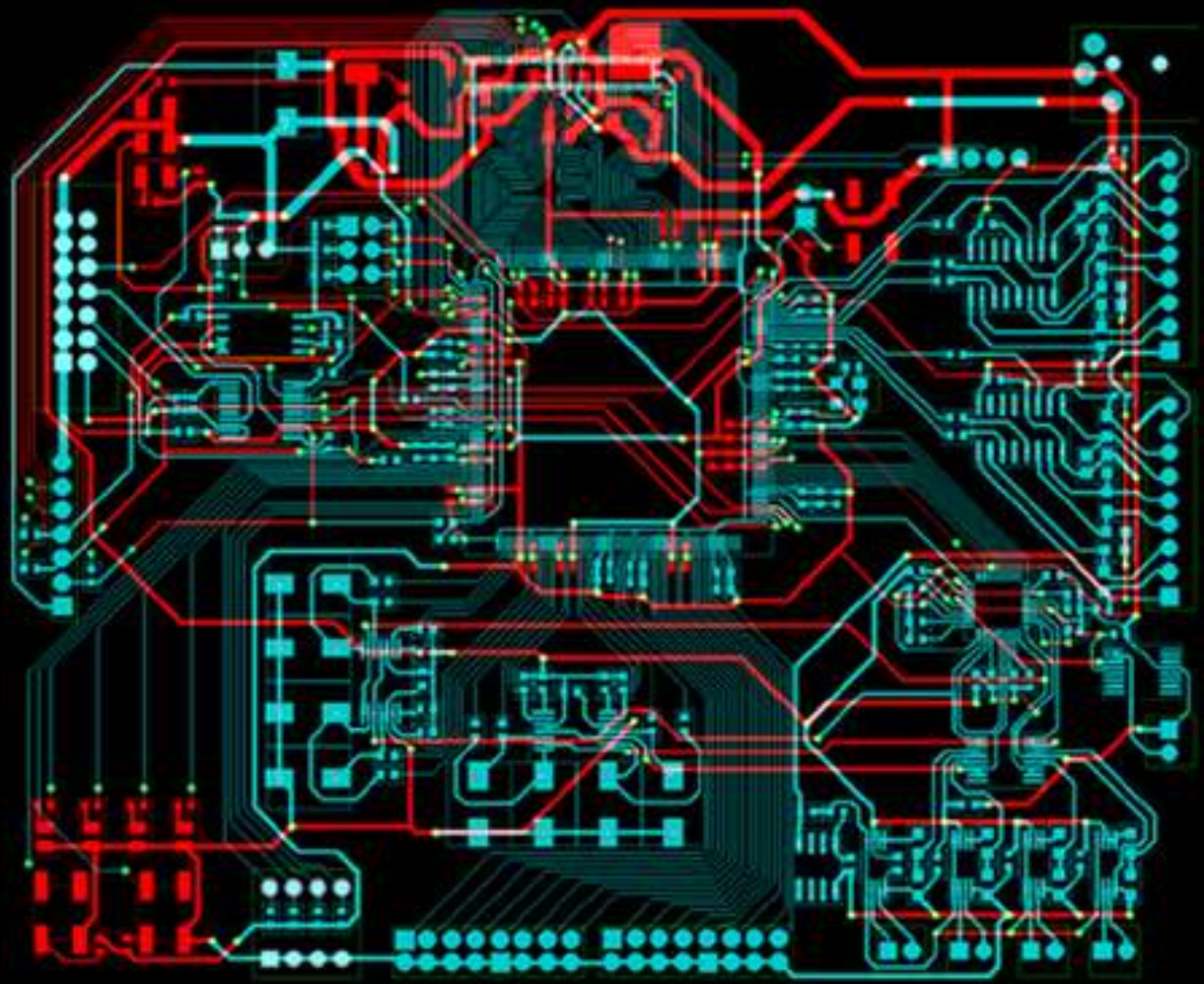
Electronic Design Automation & Gate Arrays



Yadiel Nelson

Marvin Childs

First Edition, 2012

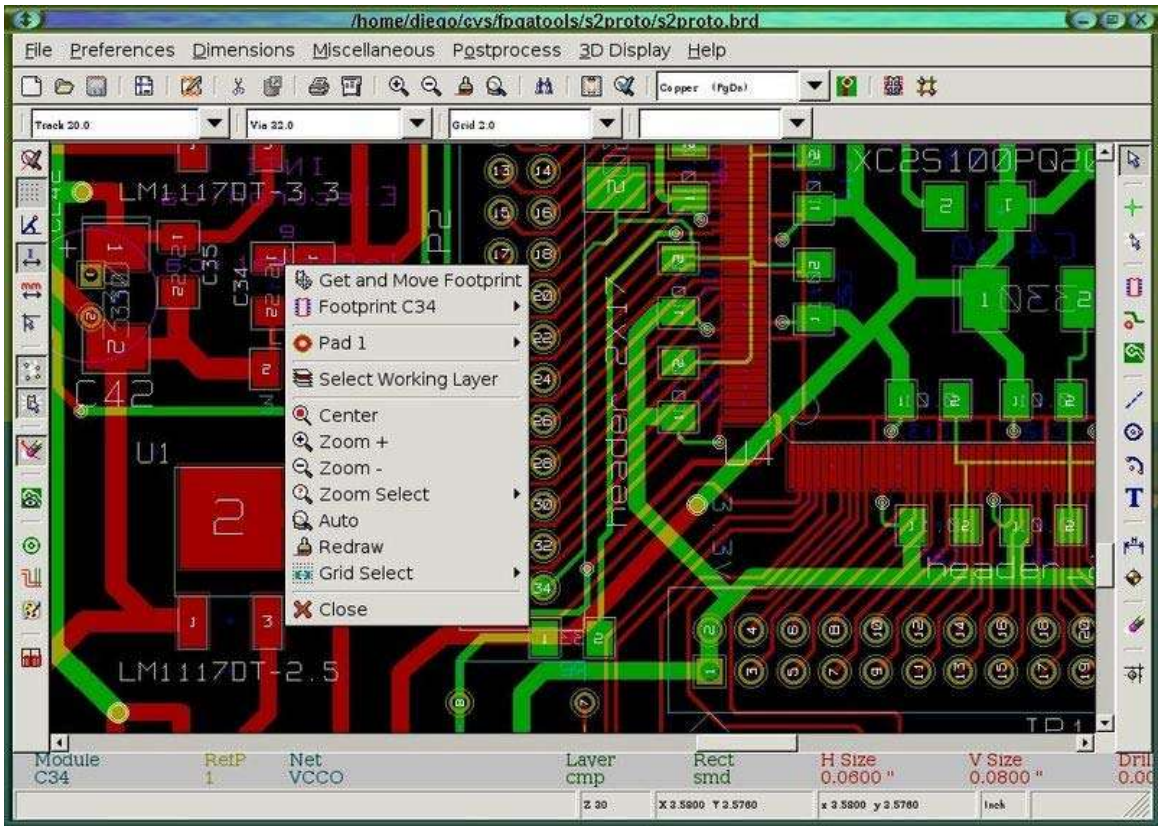
ISBN 978-81-323-1345-8

Published by:
College Publishing House
4735/22 Prakashdeep Bldg,
Ansari Road, Darya Ganj,
Delhi - 110002
Email: info@wtbooks.com

Table of Contents

Chapter 1

Introduction to Electronic Design Automation



PCB layout program

Electronic design automation (**EDA** or **ECAD**) is a category of software tools for designing electronic systems such as printed circuit boards and integrated circuits. The tools work together in a design flow that chip designers use to design and analyze entire semiconductor chips.

History

Early days

Before EDA, integrated circuits were designed by hand, and manually laid out. Some advanced shops used geometric software to generate the tapes for the Gerber photoplotter, but even those copied digital recordings of mechanically-drawn components. The process was fundamentally graphic, with the translation from electronics to graphics done manually. The best known company from this era was Calma, whose GDSII format survives.

By the mid-70s, developers started to automate the design, and not just the drafting. The first placement and routing (Place and route) tools were developed. The proceedings of the Design Automation Conference cover much of this era.

The next era began about the time of the publication of "Introduction to VLSI Systems" by Carver Mead and Lynn Conway in 1980. This ground breaking text advocated chip design with programming languages that compiled to silicon. The immediate result was a considerable increase in the complexity of the chips that could be designed, with improved access to design verification tools that used logic simulation. Often the chips were easier to lay out and more likely to function correctly, since their designs could be simulated more thoroughly prior to construction. Although the languages and tools have evolved, this general approach of specifying the desired behavior in a textual programming language and letting the tools derive the detailed physical design remains the basis of digital IC design today.

The earliest EDA tools were produced academically. One of the most famous was the "Berkeley VLSI Tools Tarball", a set of UNIX utilities used to design early VLSI systems. Still widely used is the Espresso heuristic logic minimizer and Magic.

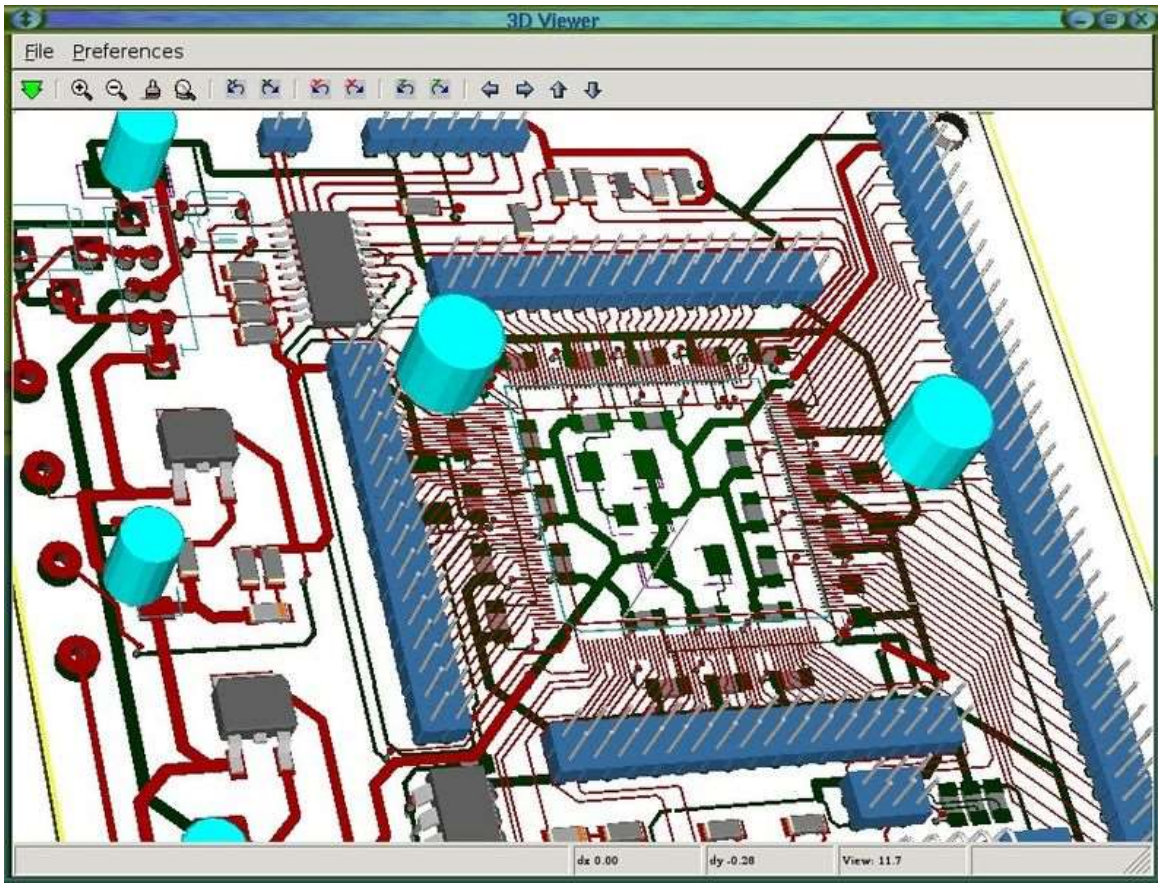
Another crucial development was the formation of MOSIS, a consortium of universities and fabricators that developed an inexpensive way to train student chip designers by producing real integrated circuits. The basic concept was to use reliable, low-cost, relatively low-technology IC processes, and pack a large number of projects per wafer, with just a few copies of each projects' chips. Cooperating fabricators either donated the processed wafers, or sold them at cost, seeing the program as helpful to their own long-term growth.

Birth of commercial EDA

1981 marks the beginning of EDA as an industry. For many years, the larger electronic companies, such as Hewlett Packard, Tektronix, and Intel, had pursued EDA internally. In 1981, managers and developers spun out of these companies to concentrate on EDA as a business. Daisy Systems, Mentor Graphics, and Valid Logic Systems were all founded around this time, and collectively referred to as **DMV**. Within a few years there were

many companies specializing in EDA, each with a slightly different emphasis. The first trade show for EDA was held at the Design Automation Conference in 1984.

In 1986, Verilog, a popular high-level design language, was first introduced as a hardware description language by Gateway Design Automation. In 1987, the U.S. Department of Defense funded creation of VHDL as a specification language. Simulators quickly followed these introductions, permitting direct simulation of chip designs: executable specifications. In a few more years, back-ends were developed to perform logic synthesis.



3D PCB layout

Current status

Current digital flows are extremely modular. The front ends produce standardized design descriptions that compile into invocations of "cells,", without regard to the cell technology. Cells implement logic or other electronic functions using a particular integrated circuit technology. Fabricators generally provide libraries of components for their production processes, with simulation models that fit standard simulation tools. Analog EDA tools are far less modular, since many more functions are required, they interact more strongly, and the components are (in general) less ideal.

EDA for electronics has rapidly increased in importance with the continuous scaling of semiconductor technology. Some users are foundry operators, who operate the semiconductor fabrication facilities, or "fabs", and design-service companies who use EDA software to evaluate an incoming design for manufacturing readiness. EDA tools are also used for programming design functionality into FPGAs.

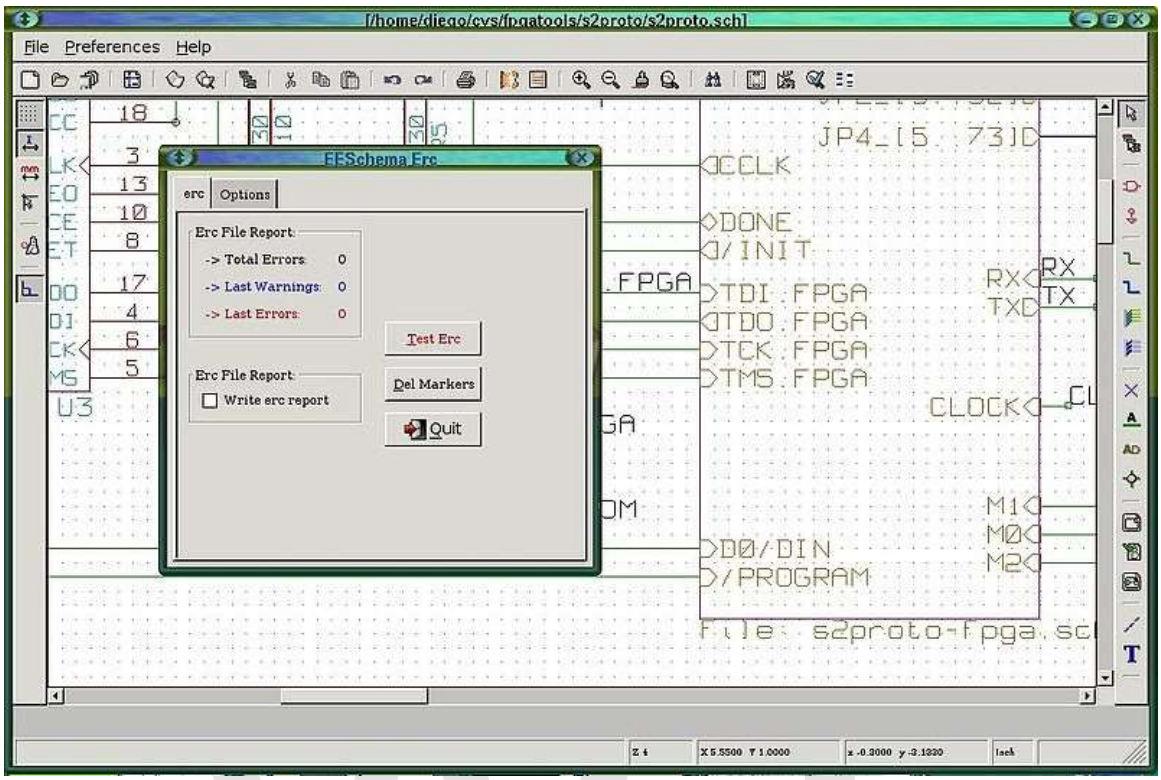
Software focuses

Design

- High-level synthesis(syn. behavioural synthesis, algorithmic synthesis) For digital chips
- Logic synthesis translation of abstract, logical language such as Verilog or VHDL into a discrete netlist of logic-gates
- Schematic Capture For standard cell digital, analog, rf like Capture CIS in Orcad by CADENCE and ISIS in Proteus
- Layout like Layout in Orcad by Cadence, ARES in Proteus

Simulation

- Transistor simulation – low-level transistor-simulation of a schematic/layout's behavior, accurate at device-level.
- Logic simulation – digital-simulation of an RTL or gate-netlist's digital (boolean 0/1) behavior, accurate at boolean-level.
- **Behavioral Simulation** – high-level simulation of a design's architectural operation, accurate at cycle-level or interface-level.
- Hardware emulation – Use of special purpose hardware to emulate the logic of a proposed design. Can sometimes be plugged into a system in place of a yet-to-be-built chip; this is called **in-circuit emulation**.
- Technology CAD simulate and analyze the underlying process technology. Electrical properties of devices are derived directly from device physics.
- Electromagnetic field solvers, or just field solvers, solve Maxwell's equations directly for cases of interest in IC and PCB design. They are known for being slower but more accurate than the layout extraction above.



Schematic capture program

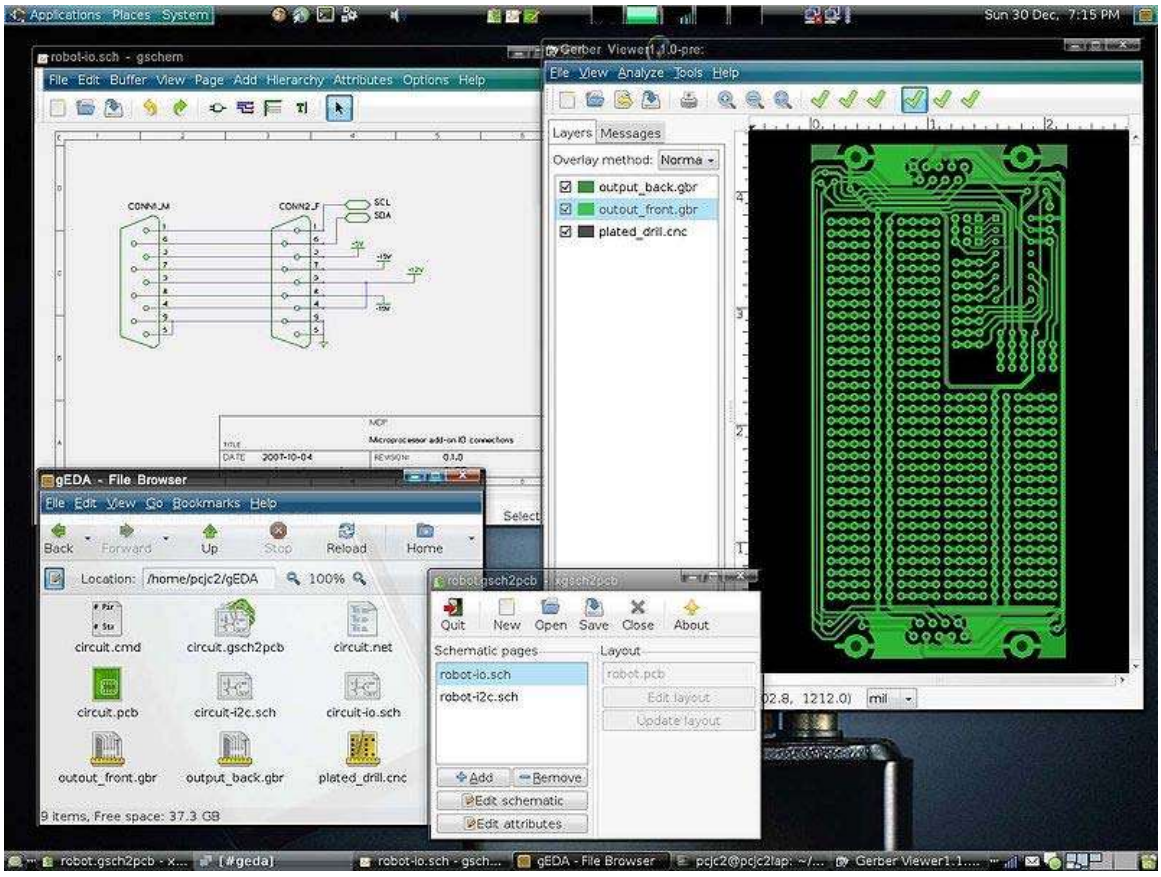
Analysis and verification

- Functional verification
- Clock Domain Crossing Verification (CDC check): Similar to linting, but these checks/tools specialize in detecting and reporting potential issues like data loss, meta-stability due to use of multiple clock domains in the design.
- Formal verification, also model checking: Attempts to prove, by mathematical methods, that the system has certain desired properties, and that certain undesired effects (such as deadlock) cannot occur.
- Equivalence checking: algorithmic comparison between a chip's RTL-description and synthesized gate-netlist, to ensure functional equivalence at the *logical* level.
- Static timing analysis: Analysis of the timing of a circuit in an input-independent manner, hence finding a worst case over all possible inputs.
- Physical verification, PV: checking if a design is physically manufacturable, and that the resulting chips will not have any function-preventing physical defects, and will meet original specifications.

Manufacturing preparation

- Mask data preparation, MDP: generation of actual lithography photomask used to physically manufacture the chip.

- Resolution enhancement techniques, RET – methods of increasing of quality of final photomask.
- Optical proximity correction, OPC – up-front compensation for diffraction and interference effects occurring later when chip is manufactured using this mask.
- Mask generation – generation of flat mask image from hierarchical design.
- Automatic test pattern generation, ATPG – generates pattern-data to systematically exercise as many logic-gates, and other components, as possible.
- Built-in self-test, or BIST – installs self-contained test-controllers to automatically test a logic (or memory) structure in the design



PCB layout and schematic for connector design

Companies

Top companies

- $3.73 billion - Synopsys
- $2.06 billion - Cadence
- $1.18 billion - Mentor Graphics

- $233 million - Magma Design Automation
- $157 million - Zuken Inc.

Note: Market caps current as of October, 2010. EEsof should likely be on this list, but does not have a market cap as it is the EDA division of Agilent.

Acquisitions

Many of the EDA companies acquire small companies with software or other technology that can be adapted to their core business. Most of the market leaders are rather incestuous amalgamations of many smaller companies. This trend is helped by the tendency of software companies to design tools as accessories that fit naturally into a larger vendor's suite of programs (on digital circuitry, many new tools incorporate analog design, and mixed systems. This is happening because there is now a trend to place entire electronic systems on a single chip.

Chapter 2

High-Level Synthesis & Logic Synthesis

High-level synthesis

High-level synthesis (HLS), sometimes referred to as *C synthesis, electronic system level (ESL) synthesis, algorithmic synthesis, or behavioral synthesis*, is an automated design process that interprets an algorithmic description of a desired behavior and creates hardware that implements that behavior . The starting point of a high-level synthesis flow is ANSI C/C++/SystemC code. The code is analyzed, architecturally constrained, and scheduled to create a register transfer level hardware design language (HDL), which is then in turn commonly synthesized to the gate level by the use of a logic synthesis tool. The goal of HLS is to let hardware designers efficiently build and verify hardware, by giving them better control over optimization of their design architecture, and through the nature of allowing the designer to describe the design at a higher level of tools while the tool does the RTL implementation. Verification of the RTL is an important part of the process.

Hardware design can be created at a variety of levels of abstraction. The commonly used levels of abstraction are gate level, register transfer level (RTL), and algorithmic level.

While logic synthesis uses an RTL description of the design, high-level synthesis works at a higher level of abstraction, starting with an algorithmic description in a high-level language such as SystemC and Ansi C/C++. The designer typically develops the module functionality and the interconnect protocol. The high-level synthesis tools handle the micro-architecture and transform untimed or partially timed functional code into fully timed RTL implementations, automatically creating cycle-by-cycle detail for hardware implementation. The (RTL) implementations are then used directly in a conventional logic synthesis flow to create a gate-level implementation.

History

Early academic work extracted scheduling, allocation, and binding as the basic steps for high-level-synthesis. Scheduling partitions the algorithm in control steps that are used to define the states in the FSM. Each control step contains one small section of the algorithm that can be performed in a single clock cycle in the hardware. Allocation and

binding maps the instructions and variables to the hardware components, multiplexors, registers and wires of the data path.

First generation behavioral synthesis was introduced by Synopsys in 1994 as Behavioral Compiler and used Verilog or VHDL as input languages. The abstraction level used was partially timed (clocked) processes. Tools based on behavioral Verilog or VHDL were not widely adopted in part because neither languages nor the partially timed abstraction were well suited to modeling behavior at a high level. 10 years later, in early 2004, Synopsys end-of-lifed Behavioral Compiler.

In 2004, there emerged a number of next generation commercial high-level synthesis products (also called behaviorial synthesis or algorithmic synthesis at the time) which provided synthesis of circuits specified at C level to a register transfer level (RTL) specification. Synthesizing from the popular C language offered accrued abstraction, expressive power and coding flexibility while tying with existing flows and legacy models. This language shift, combined with other technical advances was a key enabler for successful industrial usage. High-level synthesis tools are used for complex ASIC and FPGA design.

High-level synthesis was primarily adopted in Japan and Europe in the early years. As of late 2008, there was an emerging adoption in the United States.

Source Input

The most common source inputs for high level synthesis are based on standards languages such as ANSI C/C++ and SystemC.

High level synthesis typically also includes a bit-accurate executable specification as input, since to derive an efficient hardware implementation, additional information is needed on what is an acceptable Mean-Square Error or Bit-Error Rate etc. For example, if the designer starts with a FIR filter written using the "double" floating type, before he or she can derive an efficient hardware implementation, they need to perform numerical refinement to arrive at a fixed-point implementation. The refinement requires additional information on the level of quantization noise that can be tolerated, the valid input ranges etc. This bit-accurate specification makes the high level synthesis source specification functionally complete.

Process stages

The high-level synthesis process consists of a number of activities. Various high-level synthesis tools perform these activities in different orders using different algorithms. Some high-level synthesis tools combine some of these activities or perform them iteratively to converge on the desired solution.

- *Lexical processing*

- *Algorithm optimization*
- *Control/Dataflow analysis*
- *Library processing*
- *Resource allocation*
- *Scheduling*
- *Functional unit binding*
- *Register binding*
- *Output processing*
- *Input Rebundling*

Functionality

Architectural constraints

Synthesis constraints for the architecture can automatically be applied based on the design analysis. These constraints can be broken into

- Hierarchy
- Interface
- Memory
- Loop
- Low-level timing constraints
- iteration

Interface synthesis

Interface Synthesis refers to the ability to accept pure C/C++ description as its input, then use automated interface synthesis technology to control the timing and communications protocol on the design interface. This enables interface analysis and exploration of a full range of hardware interface options such as streaming, single- or dual-port RAM plus various handshaking mechanisms. With interface synthesis the designer does not embed interface protocols in the source description. Examples might be: direct connection, one line, 2 line handshake, FIFO.

Logic synthesis

In electronics, **logic synthesis** is a process by which an abstract form of desired circuit behavior (typically register transfer level (RTL)) is turned into a design implementation in terms of logic gates. Common examples of this process include synthesis of HDLs, including VHDL and Verilog. Some tools can generate bitstreams for programmable logic devices such as PALs or FPGAs, while others target the creation of ASICs. Logic synthesis is one aspect of electronic design automation.

History of logic synthesis

The roots of logic synthesis can be traced to the treatment of logic by George Boole (1815 to 1864), in what is now termed Boolean algebra. In 1938, Claude Shannon showed that the two-valued Boolean algebra can describe the operation of switching circuits. In the early days, logic design involved manipulating the truth table representations as Karnaugh maps. The Karnaugh map-based minimization of logic is guided by a set of rules on how entries in the maps can be combined. A human designer can typically only work with Karnaugh maps containing up to four to six variables.

The first step toward automation of logic minimization was the introduction of the Quine–McCluskey algorithm that could be implemented on a computer. This exact minimization technique presented the notion of prime implicants and minimum cost covers that would become the cornerstone of two-level minimization. Nowadays, the much more efficient Espresso heuristic logic minimizer has become the standard tool for this operation. Another area of early research was in state minimization and encoding of finite state machines (FSMs), a task that was the bane of designers. The applications for logic synthesis lay primarily in digital computer design. Hence, IBM and Bell Labs played a pivotal role in the early automation of logic synthesis. The evolution from discrete logic components to programmable logic arrays (PLAs) hastened the need for efficient two-level minimization, since minimizing terms in a two-level representation reduces the area in a PLA.

However, two-level logic circuits are of limited importance in a very-large-scale integration (VLSI) design; most designs use multiple levels of logic. As a matter of fact, almost any circuit representation in RTL or Behavioural Description is a multi-level representation. An early system that was used to design multilevel circuits was LSS from IBM. It used local transformations to simplify logic. Work on LSS and the Yorktown Silicon Compiler spurred rapid research progress in logic synthesis in the 1980s. Several universities contributed by making their research available to the public, most notably SIS from University of California, Berkeley, RASP from University of California, Los Angeles and BOLD from University of Colorado, Boulder. Within a decade, the technology migrated to commercial logic synthesis products offered by electronic design automation companies.

High-level synthesis or behavioral synthesis

With a goal of increasing designer productivity, research efforts on the synthesis of circuits specified at the behavioral level have led to the emergence of commercial solutions in 2004, which are used for complex ASIC and FPGA design. These tools automatically synthesize circuits specified at C level to a register transfer level (RTL) specification, which can be used as input to a gate-level logic synthesis flow. Today, high-level synthesis, also known as ESL synthesis and behavioral synthesis, essentially refers to circuit synthesis from high level Languages like ANSI C/C++ or SystemC etc., whereas Logic Synthesis refers to synthesis from structural or functional description to RTL.

Multi-level logic minimization

Typical practical implementations of a logic function utilize a multi-level network of logic elements. Starting from an RTL description of a design, the synthesis tool constructs a corresponding multilevel Boolean network.

Next, this network is optimized using several technology-independent techniques before technology-dependent optimizations are performed. The typical cost function during technology-independent optimizations is total literal count of the factored representation of the logic function (which correlates quite well with circuit area).

Finally, technology-dependent optimization transforms the technology-independent circuit into a network of gates in a given technology. The simple cost estimates are replaced by more concrete, implementation-driven estimates during and after technology mapping. Mapping is constrained by factors such as the available gates (logic functions) in the technology library, the drive sizes for each gate, and the delay, power, and area characteristics of each gate.

Commercial tool for logic synthesis

Software tools for logic synthesis targeting ASICs

- *Design Compiler* by Synopsys
- *Encounter RTL Compiler* by Cadence Design Systems
    - *BuildGates*, an older product by Cadence Design Systems, humorously named after Bill Gates
- *TalusDesign* by Magma Design Automation
- *BooleDozer:* Logic synthesis tool by IBM (internal IBM EDA tool)

Software tools for logic synthesis targeting FPGAs

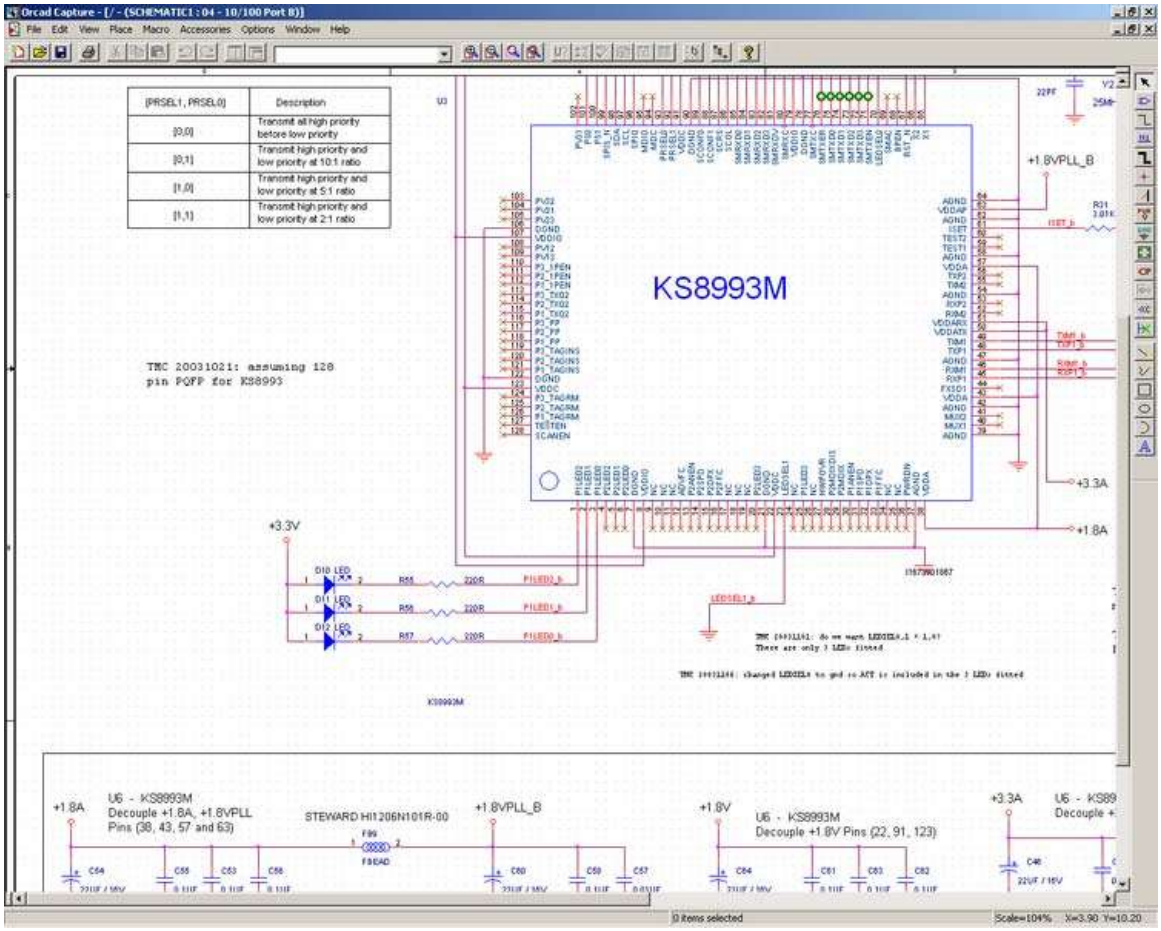
- *Encounter RTL Compiler* by Cadence Design Systems
- *LeonardoSpectrum and Precision (RTL / Physical)* by Mentor Graphics

- *Synplify (PRO / Premier)* by Synplicity
- *BlastFPGA* by Magma Design Automation
- *Quartus II integrated Synthesis by Altera*
- *XST (delivered within ISE) by Xilinx*
- *DesignCompiler Ultra* and *IC Compiler* by Synopsys
- *IspLever* by Lattice Semiconductor

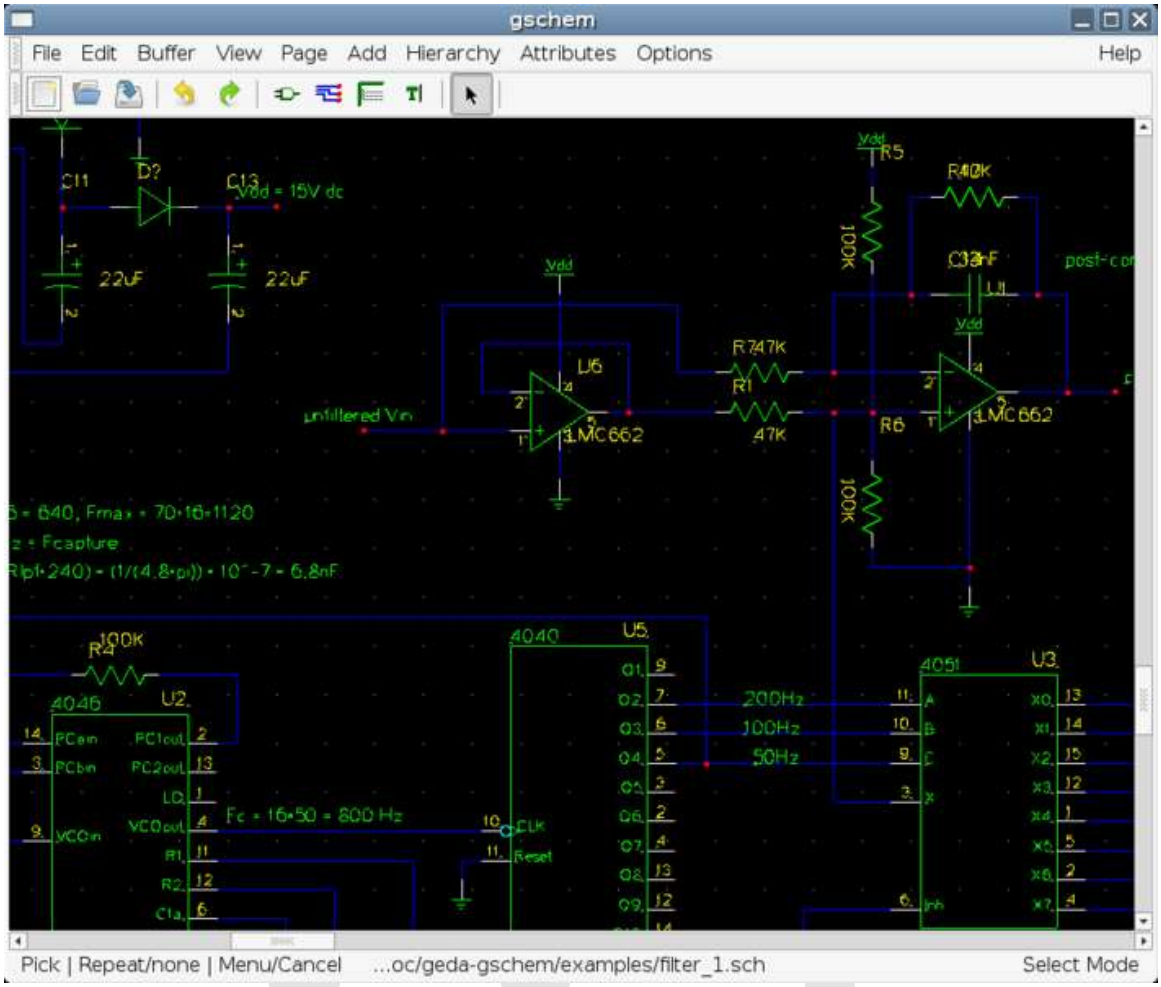
Chapter 3

Schematic Capture & Placement

Schematic capture



Orcad schematic capture program



gschem open source schematic capture. Part of the gEDA suite

Schematic capture or **schematic entry** is a step in the design cycle of electronic design automation (EDA) at which the electronic diagram, or electronic schematic of the designed electronic circuit is created by a designer. This is done interactively with the help of a schematic capture tool also known as schematic editor.

The circuit design is the very first step of actual design of an electronic circuit. Typically sketches are drawn on paper, and then entered into a computer using a schematic editor. Therefore schematic entry is said to be a front-end operation of several others in the design flow.

Despite the complexity of modern components – huge ball grid arrays and tiny passive components – schematic capture is easier today than it has been for many years. CAD software is easier to use and is available in full-featured expensive packages, very capable mid-range packages that sometimes have free versions and completely free versions that are either open source or directly linked to a printed circuit board fabrication company.

In past years, schematic diagrams with largely discrete components were fairly readable however with the newer high pin-count parts and with the almost universal use of standard letter or A4 sized paper, schematics have become less so. Many times, there will be a single large part on a page with nothing but pin reference keys to connect it to other pages.

Readability levels can be enhanced by using buses and superbuses, related pins can be connected into a common bus and routed to other pages. Buses don't need to be just the traditional address or data bus directly linked pins. A bus grouping can also be used for related uses, such as all analog input or all communications related pin functions.

Other Considerations

After the circuit design is captured in a schematic, most EDA tools allow the design to be simulated.

Schematic capture involves not only entering the circuits into the CAD system, but also generally calls for decisions that may seem more appropriate for later in the design, such as package choice. Although you may be able to change the package later, many PCB CAD systems ask you to choose both the part and package when placing it into the schematic capture program.

This also brings into play such considerations as prototyping and assembly. In a high-volume assembly environment, there will be plenty of opportunities for DFM analysis. However, in a rapid prototyping environment such as at assembly houses specializing in low-volume/high-mix and quick turnaround times, the pick and place machines are programmed directly from the board layout files. Careful package selection during schematic capture will save time during the assembly and debug process.

With new parts, the CAD system may not have your chosen component in its parts library, so you may need to create the parts library yourself. Again, you may at the time not be overly concerned with the package, but careful creation of the part library will save time and risk later.

After the circuit design is captured in a schematic, then the PCB layout can begin.

Schematic Capture Tips

To make schematics easier to read there are a few tips that can increase organization and readability.

- Schematics should read left to right, just like a book. Generally, inputs on the left, outputs on the right.

- Naming signals / nets will dramatically increase the readability and traceability of your schematics. It will be a great help during the pcb design phase of your project.
- No 4-way ties. A 4-way tie can cause confusion and should be avoided at all times (This 'tip' is often contended among design professionals).
- Break the schematic into logical building blocks. This is easily accomplished with multiple pages, each having a dedicated functionality.
- Use all capital letters as much as possible, especially for small text such as net names. This improves legibility, and is required by most drafting standards.
- To maximize portability, net names should consist of letters, numbers, and underscores. Use of characters such as spaces, pluses or minuses, or case sensitive names (i.e. two or more names who only differ in case) is discouraged, as it can cause subtle problems when exporting a netlist for simulation or layout. For example, net names in Verilog cannot have spaces; a netlist export tool would have to "escape" these characters to allow a Verilog simulation, which could lead to confusion if there are later errors referencing these nets.

Placement (EDA)

Placement is an essential step in electronic design automation - the portion of the physical design flow that assigns exact locations for various circuit components within the chip's core area. An inferior placement assignment will not only affect the chip's performance but might also make it nonmanufacturable by producing excessive wirelength, which is beyond available routing resources. Consequently, a placer must perform the assignment while optimizing a number of objectives to ensure that a circuit meets its performance demands. Typical placement objectives include

- Total wirelength: Minimizing the total wirelength, or the sum of the length of all the wires in the design, is the primary objective of most existing placers. This not only helps minimize chip size, and hence cost, but also minimizes power and delay, which are proportional to the wirelength (This assumes long wires have additional buffering inserted; all modern design flows do this.)
- Timing: The clock cycle of a chip is determined by the delay of its longest path, usually referred to as the critical path. Given a performance specification, a placer must ensure that no path exists with delay exceeding the maximum specified delay.
- Congestion: While it is necessary to minimize the total wirelength to meet the total routing resources, it is also necessary to meet the routing resources within various local regions of the chip's core area. A congested region might lead to excessive routing detours, or make it impossible to complete all routes.

- Power: Power minimization typically involves distributing the locations of cell components so as to reduce the overall power consumption, alleviate hot spots, and smooth temperature gradients.
- A secondary objective is placement runtime minimization.

Placement within the EDA design flow

A placer takes a given synthesized circuit netlist together with a technology library and produces a valid placement layout. The layout is optimized according to the aforementioned objectives and ready for cell resizing and buffering — a step essential for timing and signal integrity satisfaction. Clock-tree synthesis and routing follow, completing the physical design process. In many cases, parts of, or the entire, physical design flow are iterated a number of times until design closure is achieved.

In the case of application-specific integrated circuits, or ASICs, the chip's core layout area comprises a number of fixed height rows, with either some or no space between them. Each row consists of a number of sites which can be occupied by the circuit components. A free site is a site that is not occupied by any component. Circuit components are either standard cells, macro blocks, or I/O pads. Standard cells have a fixed height equal to a row's height, but have variable widths. The width of a cell is an integral number of sites. On the other hand, blocks are typically larger than cells and have variable heights that can stretch a multiple number of rows. Some blocks can have preassigned locations — say from a previous floorplanning process — which limit the placer's task to assigning locations for just the cells. In this case, the blocks are typically referred to by fixed blocks. Alternatively, some or all of the blocks may not have preassigned locations. In this case, they have to be placed with the cells in what is commonly referred to as mixed-mode placement.

In addition to ASICs, placement retains its prime importance in gate array structures such as field-programmable gate arrays (FPGAs). In FPGAs, placement maps the circuit's subcircuits into programmable FPGA logic blocks in a manner that guarantees the completion of the subsequent stage of routing.

Basic techniques

- Analytical techniques approximate the wirelength objective using quadratic or nonlinear formulations.
- The advent of min-cut partitioners paved the way to the introduction of min-cut placers.
- Another thread of placement techniques started with the proposal of simulated annealing as a general combinatorial optimization technique.

Chapter 4

SPICE

SPICE 1

Original author(s)	Laurence Nagel
Initial release	1973
Written in	Fortran
Type	Electronic circuit simulation

SPICE 2

Initial release	1975
Stable release	2G.6 / 1983
Written in	Fortran
Type	Electronic circuit simulation

SPICE 3

Original author(s)	Thomas Quarles
Initial release	1989
Stable release	3f.4 / July 1993
Written in	C
Type	Electronic circuit simulation

SPICE (Simulation Program with Integrated Circuit Emphasis) is a general-purpose open source analog electronic circuit simulator. It is a powerful program that is used in integrated circuit and board-level design to check the integrity of circuit designs and to predict circuit behavior.

Introduction

Integrated circuits, unlike board-level designs composed of discrete parts, are impossible to breadboard before manufacture. Further, the high costs of photolithographic masks and other manufacturing prerequisites make it essential to design the circuit to be as close to perfect as possible before the integrated circuit is first built. Simulating the circuit with SPICE is the industry-standard way to verify circuit operation at the transistor level before committing to manufacturing an integrated circuit.

Board-level circuit designs can often be breadboarded for testing. Even with a breadboard, some circuit properties may not be accurate compared to the final printed wiring board, such as parasitic resistances and capacitances. These parasitic components can often be estimated more accurately using SPICE simulation. Also, designers may want more information about the circuit than is available from a single mock-up. For instance, circuit performance is affected by component manufacturing tolerances. In these cases it is common to use SPICE to perform Monte Carlo simulations of the effect of component variations on performance, a task which is impractical using calculations by hand for a circuit of any appreciable complexity.

Circuit simulation programs, of which SPICE and derivatives are the most prominent, take a text netlist describing the circuit elements (transistors, resistors, capacitors, etc.) and their connections, and translate this description into equations to be solved. The general equations produced are nonlinear differential algebraic equations which are solved using implicit integration methods, Newton's method and sparse matrix techniques.

Origins

SPICE was developed at the Electronics Research Laboratory of the University of California, Berkeley by Laurence Nagel with direction from his research advisor, Prof. Donald Pederson. SPICE1 was largely a derivative of the CANCER program, which Nagel had worked on under Prof. Ronald Rohrer. CANCER was an acronym for "Computer Analysis of Nonlinear Circuits, Excluding Radiation," a hint to Berkeley's liberalism of 1960s: at these times many circuit simulators were developed under the United States Department of Defense contracts that required the capability to evaluate the radiation hardness of a circuit. When Nagel's original advisor, Prof. Rohrer, left Berkeley, Prof. Pederson became his advisor. Pederson insisted that CANCER, a proprietary program, be rewritten enough that restrictions could be removed and the program could be put in the public domain.

SPICE1 was first presented at a conference in 1973. SPICE1 was coded in FORTRAN and used nodal analysis to construct the circuit equations. Nodal analysis has limitations in representing inductors, floating voltage sources and the various forms of controlled sources. SPICE1 had relatively few circuit elements available and used a fixed-timestep transient analysis. The real popularity of SPICE started with SPICE2 in 1975. SPICE2,

also coded in FORTRAN, was a much-improved program with more circuit elements, variable timestep transient analysis using either the trapezoidal (second order Adams-Moulton method) or the Gear integration method (also known as BDF), equation formulation via modified nodal analysis (avoiding the limitations of nodal analysis), and an innovative FORTRAN-based memory allocation system developed by another graduate student, Ellis Cohen. The last FORTRAN version of SPICE was 2G.6 in 1983. SPICE3 was developed by Thomas Quarles (with A. Richard Newton as advisor) in 1989. It is written in C, uses the same netlist syntax, and added X Window System plotting.

As an early open source program, SPICE was widely distributed and used. Its ubiquity became such that "to SPICE a circuit" remains synonymous with circuit simulation. SPICE source code was from the beginning distributed by UC Berkeley for a nominal charge (to cover the cost of magnetic tape). The license originally included distribution restrictions for countries not considered friendly to the USA, but the source code is currently covered by the BSD license.

SPICE inspired and served as a basis for many other circuit simulation programs, in academia, in industry, and in commercial products. The first commercial version of SPICE was ISPICE, an interactive version on a timeshare service, National CSS. The most prominent commercial versions of SPICE include HSPICE (originally commercialized by Shawn and Kim Hailey of Meta Software, but now owned by Synopsys) and PSPICE (now owned by Cadence Design Systems). The academic spinoffs of SPICE include XSPICE, developed at Georgia Tech, which added mixed analog/digital "code models" for behavioral simulation, and Cider (previously CODECS, from UC Berkeley/Oregon State Univ.) which added semiconductor device simulation. The integrated circuit industry adopted SPICE quickly, and until commercial versions became well developed many IC design houses had proprietary versions of SPICE. Today a few IC manufacturers, typically the larger companies, have groups continuing to develop SPICE-based circuit simulation programs. Among these are ADICE at Analog Devices, LTspice at Linear Technology, Mica at Freescale Semiconductor, and TISPICE at Texas Instruments. (Other companies maintain internal circuit simulators which are not directly based upon SPICE, among them PowerSpice at IBM, Titan at Qimonda, Lynx at Intel Corporation, and Pstar at NXP Semiconductor.)

Program features and structure

SPICE became popular because it contained the analyses and models needed to design integrated circuits of the time, and was robust enough and fast enough to be practical to use. Precursors to SPICE often had a single purpose: The BIAS program, for example, did simulation of bipolar transistor circuit operating points; the SLIC program did only small-signal analyses. SPICE combined operating point solutions, transient analysis, and various small-signal analyses with the circuit elements and device models needed to successfully simulate many circuits.

Analyses

SPICE2 included these analyses:

- AC analysis (linear small-signal frequency domain analysis)
- DC analysis (nonlinear quiescent point calculation)
- DC transfer curve analysis (a sequence of nonlinear operating points calculated while sweeping an input voltage or current, or a circuit parameter)
- Noise analysis (a small signal analysis done using an adjoint matrix technique which sums uncorrelated noise currents at a chosen output point)
- Transfer function analysis (a small-signal input/output gain and impedance calculation)
- Transient analysis (time-domain large-signal solution of nonlinear differential algebraic equations)

Since SPICE is generally used to model nonlinear circuits, the small signal analyses are necessarily preceded by a quiescent point calculation at which the circuit is linearized. SPICE2 also contained code for other small-signal analyses: sensitivity analysis, pole-zero analysis, and small-signal distortion analysis. Analysis at various temperatures was done by automatically updating semiconductor model parameters for temperature, allowing the circuit to be simulated at temperature extremes.

Other circuit simulators have since added many analyses beyond those in SPICE2 to address changing industry requirements. Parametric sweeps were added to analyze circuit performance with changing manufacturing tolerances or operating conditions. Loop gain and stability calculations were added for analog circuits. Harmonic balance or time-domain steady state analyses were added for RF and switched-capacitor circuit design. However, a public-domain circuit simulator containing the modern analyses and features needed to become a successor in popularity to SPICE has not yet emerged.

Device models

SPICE2 included many semiconductor device compact models: three levels of MOSFET model, a combined Ebers–Moll and Gummel-Poon bipolar model, a JFET model, and a model for a junction diode. In addition, it had many other elements: resistors, capacitors, inductors (including coupling), independent voltage and current sources, ideal transmission lines, and voltage and current controlled sources.

SPICE3 added more sophisticated MOSFET models, which were required due to advances in semiconductor technology. In particular, the BSIM family of models were added, which were also developed at UC Berkeley.

Commercial and industrial SPICE simulators have added many other device models as technology advanced and earlier models became inaccurate. To attempt standardization of these models so that a set of model parameters may be used in different simulators, an industry working group was formed, the Compact Model Council, to choose, maintain

and promote the use of standard models. The standard models today include BSIM3, BSIM4, BSIMSOI, PSP, HICUM, and MEXTRAM.

Input and output: Netlists, schematic capture and plotting

SPICE2 took a text netlist as input and produced line-printer listings as output, which fit with the computing environment in 1975. These listings were either columns of numbers corresponding to calculated outputs (typically voltages or currents), or line-printer character "plots". SPICE3 retained the netlist for circuit description, but allowed analyses to be controlled from a command-line interface similar to the C shell. SPICE3 also added basic X-Window plotting, as UNIX and engineering workstations became common.

Vendors and various free software projects have added schematic capture front-ends to SPICE, allowing a schematic diagram of the circuit to be drawn and the netlist to be automatically generated. Also, graphical user interfaces were added for selecting the simulations to be done and manipulating the voltage and current output vectors. In addition, very capable graphing utilities have been added to see waveforms and graphs of parametric dependencies. Several free versions of these extended programs are available, some as introductory limited packages, and some without restrictions.

Chapter 5

Logic Simulation & Hardware Emulation

Logic simulation

Logic simulation is the use of a computer program to simulate the operation of a digital circuit. Logic simulation is the primary tool used for verifying the logical correctness of a hardware design. In many cases logic simulation is the first activity performed in the process of taking a hardware design from concept to realization. Modern hardware description languages are both simulatable and synthesizable.

Levels of abstraction

Because simulation is a general technique, a hardware design can be simulated at a variety of levels of abstraction. Often it is useful to simulate a model at several levels of abstraction in the same simulation run. The commonly used levels of abstraction are gate level, register transfer level (RTL), and behavioral (or algorithmic) level. However, it is possible to incorporate lower levels like transistor level or even lower physical levels as well as higher levels such as transaction levels or domain-specific levels.

Advantages of logic simulation

Simulation is the key activity in the design verification process. That is not to say that it is an ideal process. It has some very positive attributes:

- It is a natural way for the designer to get feedback about their design. Because it is just running a program – the design itself – the designer interacts with it using the vocabulary and abstractions of the design. There is no layer of translation to obscure the behavior of the design.
- The level of effort required to debug and then verify the design is proportional to the maturity of the design. That is, early in the design's life, bugs and incorrect behavior are usually found quickly. As the design matures, it takes longer to find the errors. This is beneficial early in the design process. It becomes more problematic later.
- Simulation is completely general. Any hardware design can be simulated. The only limits are time and computer resources.

Prospective way to accelerate logic simulation is using distributed and parallel computations.

Limitations of logic simulation

On the negative side, simulation has two drawbacks, one of which is glaring:

- There is (usually) no way to know when you are done. It is not feasible to completely test, via simulation, all possible states and inputs of any non-trivial system.
- Simulation can take an inordinately large amount of computing resources, since typically it uses a single processor to reproduce the behavior of many (perhaps millions of) parallel hardware processes.

Every design project must answer the question “have we simulated enough to find all the bugs?” and every project manager has taped out his design knowing that the truthful answer to that question is either “no” or “I don’t know”. It is this fundamental problem with simulation that has caused so much effort to be spent looking for both tools to help answer the question and formal alternatives to simulation.

Code coverage, functional coverage and logic coverage tools have all been developed to help gauge the completeness of simulation testing. None are complete solutions, though they all help. Formal alternatives have been less successful. Just like in the general software world, where proving programs correct has proven intractable, formal methods for verifying hardware designs have still not proven general enough to replace simulation. That is not surprising, since it is the same problem.

The second drawback motivates most of the research and development in simulation. That is, simulation is always orders of magnitude slower than the system being simulated. If a hardware system runs at 1GHz, a simulation of that system might run at 10-1000 Hz, depending on the level of the simulation and the size of the system. That is a slowdown of from 10^6 to 10^8! Consequently, many people have spent a lot of time and effort finding ways to speed up logic simulation.

Event simulation versus cycle simulation

Event simulation allows the design to contain simple timing information – the delay needed for a signal to travel from one place to another. During simulation, signal changes are tracked in form of events. A change at a certain time triggers an event after a certain delay. Events are sorted by the time when they will occur, and when all events for a particular time have been handled, the simulated time is advanced to the time of the next scheduled event. How fast an event simulation runs depends on the number of events to be processed (the amount of activity in the model).

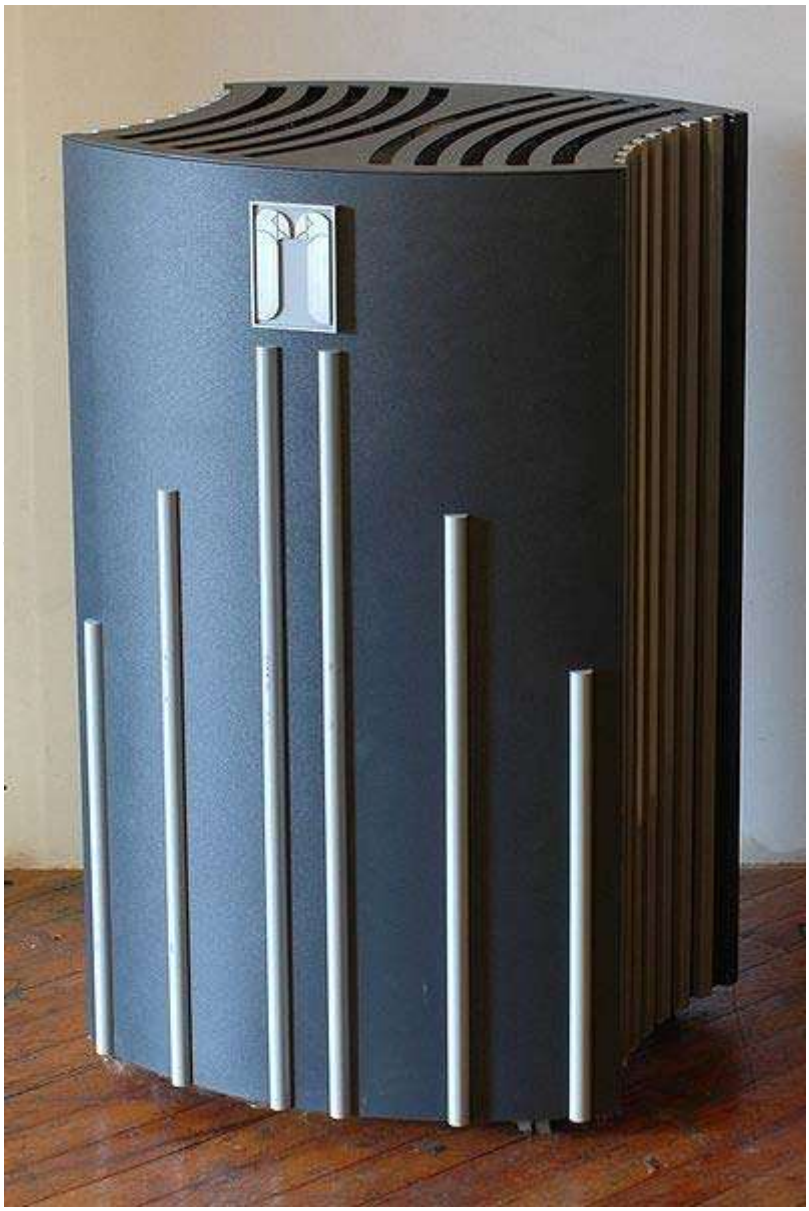
In cycle simulation, it is not possible to specify delays. A cycle-accurate model is used, and every gate is evaluated in every cycle. Cycle simulation therefore runs at a constant speed, regardless of activity in the model. Optimized implementations may make take advantage of low model activity to speed up simulation by skipping evaluation of gates whose inputs didn't change.

While event simulation can provide some feedback regarding signal timing, it is not a replacement for static timing analysis. In comparison to event simulation, cycle simulation tends to be faster, to scale better, and to be better suited for hardware acceleration / emulation. However, chip design trends point to event simulation gaining relative performance due to activity factor reduction in the circuit (due to techniques such as clock gating and power gating, which are becoming much more commonly used in an effort to reduce power dissipation). In these cases, since event simulation only simulates necessary events, performance may no longer be a disadvantage over cycle simulation. Event simulation also has the advantage of greater flexibility, handling design features difficult to handle with cycle simulation, such as asynchronous logic and incommensurate clocks. Due to these considerations, almost all commercial logic simulators have an event based capability, even if they primarily rely on cycle based techniques.

Summary

Considering both the advantages and disadvantages of logic simulation, it really is quite a good tool for verifying the correctness of a hardware design. Despite its drawbacks, simulation remains the first choice for proving correctness of a design before fabrication, and its value has been well established.

Hardware emulation



Ikos NSIM-64 Hardware simulation accelerator

In integrated circuit design, **hardware emulation** is the process of imitating the behavior of one or more pieces of hardware (typically a system under design) with another piece of hardware, typically a special purpose emulation system. The goal is normally debugging of the system being designed. Often an emulator is fast enough to be plugged into a working target system in place of a yet-to-be-built chip, so the whole system can be debugged with live data. This is a specific case of in-circuit emulation.

Sometimes hardware emulation can be confused with hardware devices such as expansion cards with hardware processors that assist functions of software emulation, such as older daughterboards with x86 chips to allow x86 OSes to run on motherboards of different processor families.

Introduction

The largest fraction of silicon integrated circuit respins are due to (at least in part) functional errors. Thus, comprehensive functional verification is key to reducing development costs and delivering a product on time. Functional verification of a design is most often performed using logic simulation and/or prototyping. There are advantages and disadvantages to each and often both are used. Logic simulation is easy, accurate, flexible, and low cost. However, simulation is often not fast enough for large designs and almost always too slow to run application software against the hardware design. FPGA-based prototypes are fast and inexpensive. But the time required to implement a large design into several FPGAs can be very long and is error-prone. Changes to fix design flaws also take a long time to implement and may require board wiring changes. Since FPGA prototypes have little debugging capability, probing signals inside the FPGAs in real time is very difficult, if not impossible, and recompiling FPGAs to move probes takes too long. The usual compromise is to use simulation early in the verification process when bugs and fixes are frequent, and prototyping at the end of the development cycle when the design is basically complete and speed is needed to get sufficient testing to uncover any remaining system-level bugs. Prototyping is also popular for testing software.

Simulation acceleration can address the performance shortcomings of simulation to an extent. Here the design is mapped into a hardware accelerator to run much faster and the testbench (and any behavioral design code) continues to run on the simulator on the workstation. A high-bandwidth, low latency channel connects the workstation to the accelerator to exchange signal data between testbench and design. By Amdahl's law, the slowest device in the chain will determine the speed achievable. Normally, this is the testbench in the simulator. With a very efficient testbench (written in C or transaction-based), the channel may become the bottleneck. In some cases, a transaction-level testbench is able to feed as much data to the design being emulated as "live" stimulus.

In-circuit emulation improves greatly on FPGA prototyping’s long time to implement and change designs, and provides a comprehensive, efficient debugging capability. While it takes weeks or months to implement an FPGA prototype, it takes only days to implement emulation. And design changes take but a few hours or less. Emulation does this at the expense of running speed and cost compared to FPGA prototypes. Looking at emulation from the other direction, it improves on acceleration’s performance by substituting "live" stimulus for the simulated testbench. This stimulus can come from a target system (the product being developed), or from test equipment. At 10,000 to 100,000 times the speed of simulation, emulation is often the only technique that can deliver the speed necessary to test application software while still providing a comprehensive hardware debug environment.

Debugging simulations vs. emulations/prototyping

It is worth noting that simulation and prototyping involve two different styles of execution. Simulation executes the RTL code serially while a prototype executes fully in parallel. This leads to differences in debugging. In simulation:

- The user can set a breakpoint and stop simulation to inspect the design state, interact with the design, and resume simulation.
- The user can stop execution “mid-cycle” as it were with only part of the code executed.
- The user can see any signal in the design and the contents of any memory location at any time.
- The user can even back up time (if they saved checkpoint(s)) and re-run.

With a prototype:

- The user employs a logic analyzer for visibility, and so can see only a limited number of signals which they determined ahead of time (by clipping on probes).
- The target does not stop when the logic analyzer triggers, so each time the user changes the probes or trigger condition, they have to reset the environment and start again from the beginning.

Acceleration and emulation are more like prototyping and silicon in terms of RTL execution and debugging since the entire design executes simultaneously as it will in the silicon. Since the same hardware is often used to provide both simulation acceleration and in-circuit emulation, these systems provide a blend of these two very different debugging styles.

High end hardware emulators provide a debugging environment with many features that can be found in logic simulators, and in some cases even surpass their debugging capabilities:

- The user can set a breakpoint and stop emulation to inspect the design state, interact with the design, and resume emulation. The emulator always stops on cycle boundaries.
- The user has visibility to any signal or memory contents in the design without the need to set up probes before the run. While visibility is provided also for past time, the amount of time that it can show in the past might be limited in some cases to the depth of the emulator's trace memory.
- The user can even back up time (if they saved checkpoint(s)) and re-run.

Emulation and 2-state logic

Another difference between simulation and acceleration and emulation is a consequence of accelerators using hardware for implementation – they have only two logic states – acting the way the silicon will when fabricated. This implies:

- They are not useful for analyzing X-state initialization.
- They cannot analyze strength resolution, or at least this must be done statically at compile time.
- Emulators do not model precise circuit timing, and hence they will probably not find any race conditions or setup and hold time violations.

These tasks are properly carried out during logic simulation or with a static timing analysis tool.

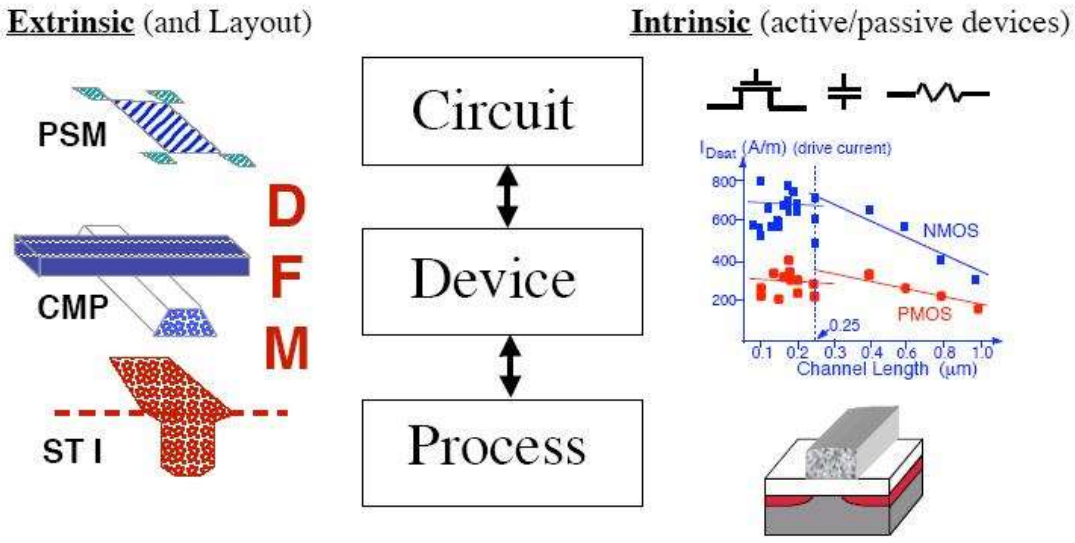
Emulation versus prototyping

A key distinction between an emulator and an FPGA prototyping system is that the emulator provides a rich debug environment while a prototyping system has little to no debug capability and is primarily used after the design is debugged to create multiple copies for system analysis and software development.

Chapter 6

Technology CAD & Electromagnetic Field Solver

Technology CAD (or **Technology Computer Aided Design**, or **TCAD**) is a branch of electronic design automation that models semiconductor fabrication and semiconductor device operation. The modeling of the fabrication is termed Process TCAD, while the modeling of the device operation is termed Device TCAD. Included are the modelling of process steps (such as diffusion and ion implantation), and modelling of the behavior of the electrical devices based on fundamental physics, such as the doping profiles of the devices. TCAD may also include the creation of *compact models* (such as the well known SPICE transistor models), which try to capture the electrical behavior of such devices but do not generally derive them from the underlying physics. (However, the SPICE simulator itself is usually considered as part of ECAD rather than TCAD.)



Hierarchy of technology CAD tools building from the process level to circuits. Left side icons show typical manufacturing issues; right side icons reflect MOS scaling results based on TCAD

Technology files and design rules are essential building blocks of the integrated circuit design process. Their accuracy and robustness over process technology, its variability and the operating conditions of the IC--environmental, parasitic interactions and testing, including adverse conditions such as electro-static discharge--are critical in determining performance, yield and reliability. Development of these technology and design rule files involves an iterative process that crosses boundaries of technology and device development, product design and quality assurance. Modeling and simulation play a critical role in support of many aspects of this evolution process.

The goals of TCAD start from the physical description of integrated circuit devices, considering both the physical configuration and related device properties, and build the links between the broad range of physics and electrical behavior models that support circuit design. Physics-based modeling of devices, in distributed and lumped forms, is an essential part of the IC process development. It seeks to quantify the underlying understanding of the technology and abstract that knowledge to the device design level, including extraction of the key parameters that support circuit design and statistical metrology. Although the emphasis here is on Metal Oxide Semiconductor (MOS) transistors--the workhorse of the IC industry--it is useful to briefly overview the development history of the modeling tools and methodology that has set the stage for the present state-of-the-art.

History

The evolution of technology computer-aided design (TCAD)--the synergistic combination of process, device and circuit simulation and modeling tools—finds its roots in bipolar technology, starting in the late 1960s, and the challenges of junction isolated, double-and triple-diffused transistors. These devices and technology were the basis of the first integrated circuits; nonetheless, many of the scaling issues and underlying physical effects are integral to IC design, even after four decades of IC development. With these early generations of IC, process variability and parametric yield were an issue—a theme that will reemerge as a controlling factor in future IC technology as well.

Process control issues--both for the intrinsic devices and all the associated parasitics--presented formidable challenges and mandated the development of a range of advanced physical models for process and device simulation. Starting in the late 1960s and into the 1970s, the modeling approaches exploited were dominantly one- and two-dimensional simulators. While TCAD in these early generations showed exciting promise in addressing the physics-oriented challenges of bipolar technology, the superior scalability and power consumption of MOS technology revolutionized the IC industry. By the mid-1980s, CMOS became the dominant driver for integrated electronics. Nonetheless, these early TCAD developments set the stage for their growth and broad deployment as an essential toolset that has leveraged technology development through the VLSI and ULSI eras which are now the mainstream.

IC development for more than a quarter-century has been dominated by the MOS technology. In the 1970s and 1980s NMOS was favored owing to speed and area advantages, coupled with technology limitations and concerns related to isolation, parasitic effects and process complexity. During that era of NMOS-dominated LSI and the emergence of VLSI, the fundamental scaling laws of MOS technology were codified and broadly applied. It was also during this period that TCAD reached maturity in terms of realizing robust process modeling (primarily one-dimensional) which then became an integral technology design tool, used universally across the industry. At the same time device simulation, dominantly two-dimensional owing to the nature of MOS devices, became the work-horse of technologists in the design and scaling of devices. The transition from NMOS to CMOS technology resulted in the necessity of tightly-coupled and fully-2D simulators for process and device simulations. This third generation of TCAD tools became critical to address the full complexity of twin-well CMOS technology (see Figure 3a), including issues of design rules and parasitic effects such as latchup. An abbreviated but prospective view of this period, through the mid-1980s, is given in; and from the point of view of how TCAD tools were used in the design process.

Modern TCAD

Today the requirements for and use of TCAD cross-cut a very broad landscape of design automation issues, including many fundamental physical limits. At the core are still a host of process and device modeling challenges that support intrinsic device scaling and parasitic extraction. These applications include technology and design rule development, extraction of compact models and more generally design for manufacturability (DFM).. The dominance of interconnects for giga-scale integration (transistor counts in O(billion)) and clocking frequencies in O (10 gigahertz)) have mandated the development of tools and methodologies that embrace patterning by electro-magnetic simulations—both for optical patterns and electronic and optical interconnect performance modeling—as well as circuit-level modeling. This broad range of issues at the device and interconnect levels, including links to underlying patterning and processing technologies, is summarized in Figure 1 and provides a conceptual framework for the discussion that now follows.

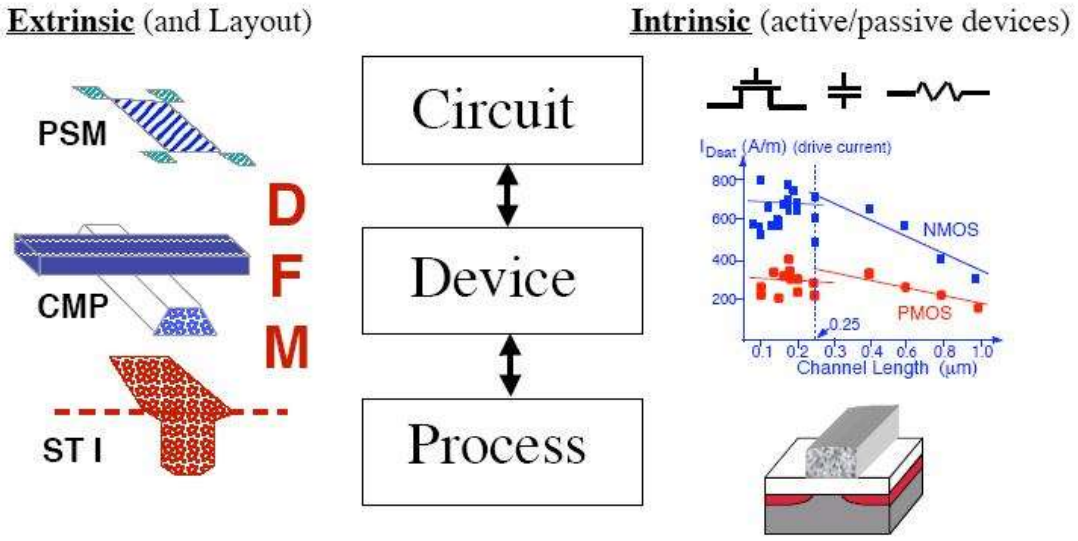


Figure 1: Hierarchy of technology CAD tools building from the process level to circuits. Left side icons show typical manufacturing issues; right side icons reflect MOS scaling results based on TCAD

Figure 1 depicts a hierarchy of process, device and circuit levels of simulation tools. On each side of the boxes indicating modeling level are icons that schematically depict representative applications for TCAD. The left side gives emphasis to Design For Manufacturing (DFM) issues such as: shallow-trench isolation (STI), extra features required for phase-shift masking (PSM) and challenges for multi-level interconnects that include processing issues of chemical-mechanical planarization (CMP), and the need to consider electro-magnetic effects using electromagnetic field solvers. The right side icons show the more traditional hierarchy of expected TCAD results and applications: complete process simulations of the intrinsic devices, predictions of drive current scaling and extraction of technology files for the complete set of devices and parasitics.

Figure 2 again looks at TCAD capabilities but this time more in the context of design flow information and how this relates to the physical layers and modeling of the electronic design automation (EDA) world. Here the simulation levels of process and device modeling are considered as integral capabilities (within TCAD) that together provide the "mapping" from mask-level information to the functional capabilities needed at the EDA level such as compact models ("technology files") and even higher-level behavioral models. Also shown is the extraction and electrical rule checking (ERC); this indicates that many of the details that to date have been embedded in analytical formulations, may in fact also be linked to the deeper TCAD level in order to support the growing complexity of technology scaling.

TCAD Providers

Current major suppliers of TCAD tools include Synopsys, Silvaco and Crosslight. The open source GSS, Archimedes and Aeneas has some of the capabilities of the commercial products. TCAD Central maintains an information resource for available TCAD software.

Electromagnetic field solver

Electromagnetic field solvers (or sometimes just **field solvers**) are specialized programs that solve (a subset of) Maxwell's equations directly. They form a part of the field of electronic design automation, or EDA, and are commonly used in the design of integrated circuits and printed circuit boards. They are used when a solution from first principles is needed, or the highest accuracy is required.

Introduction

The extraction of parasitic circuit models is important for various aspects of physical verification such as timing, signal integrity, substrate coupling, and power grid analysis. As circuit speeds and densities have increased, the need has grown to account accurately for parasitic effects for larger and more complicated interconnect structures. In addition, the electromagnetic complexity has grown as well, from resistance and capacitance, to inductance, and now even full electromagnetic wave propagation. This increase in complexity has also grown for the analysis of passive devices such as integrated inductors. Electromagnetic behavior is governed by Maxwell's equations, and all parasitic extraction requires solving some form of Maxwell's equations. That form may be a simple analytic parallel plate capacitance equation, or may involve a full numerical solution for a complicated 3D geometry with wave propagation. In layout extraction, analytic formulas for simple or simplified geometry can be used where accuracy is less important than speed, but when the geometric configuration is not simple and accuracy demands do not allow simplification, numerical solution of the appropriate form of Maxwell's equations must be employed.

The appropriate form of Maxwell's equations is typically solved by one of two classes of methods. The first uses a differential form of the governing equations and requires the discretization (meshing) of the entire domain in which the electromagnetic fields reside. Two of the most common approaches in this first class are the finite difference (FD) and finite element (FEM) method. The resultant linear algebraic system (matrix) that must be solved is large but sparse (contains very few non-zero entries). Sparse linear solution methods, such as sparse factorization, conjugate-gradient, or multigrid methods can be used to solve these systems, the best of which require CPU time and memory of O(N) time, where N is the number of elements in the discretization. However most problems in electronic design automation (EDA) are open problems, also called exterior problems,

and since the fields decrease slowly towards infinity, these methods can require extremely large N.

The second class of methods are integral equation methods which instead require a discretization of only the sources of electromagnetic field. Those sources can be physical quantities, such as the surface charge density for the capacitance problem, or mathematical abstractions resulting from the application of Green's theorem. When the sources exist only on two-dimensional surfaces for three-dimensional problems, the method is often called a boundary element method(BEM). For open problems, the sources of the field exist in a much smaller domain than the fields themselves, and thus the size of linear systems generated by integral equations methods are much smaller than FD or FEM, as illustrated for a small portion of two signal lines in Figure 1. Integral equation methods, however, generate dense (all entries are nonzero) linear systems which makes such methods preferable to FD or FEM only for small problems. Such systems require O(N^2) memory to store and O(N^3) to solve via direct Gaussian elimination or at best O(N^2) if solved iteratively. Increasing circuit speeds and densities require the solution of increasingly complicated interconnect, making dense integral equation approaches unsuitable due to these high growth rates of computational cost with increasing problem size.

In the past two decades, much work has gone into improving both the differential and integral equation approaches, as well as new approaches based on random-walk methods. Methods of truncating the discretization required by the FD and FEM approaches has greatly reduced the number of elements required. Integral equation approaches have become particularly popular for interconnect extraction due to sparsification techniques, also sometimes called matrix compression, acceleration, or matrix-free techniques, which have brought nearly O(N) growth in storage and solution time to integral equation methods.

In the IC industry, sparsified integral equation techniques are typically used to solve capacitance and inductance extraction problems. The random-walk methods have become quite mature for capacitance extraction. For problems requiring the solution of the full Maxwell's equations (full-wave), both differential and integral equation approaches are common.

Chapter 7

Elements of Electronic Design Automation

Circuit design

The process of **circuit design** can cover systems ranging from complex electronic systems all the way down to the individual transistors within an integrated circuit. For simple circuits the design process can often be done by one person without needing a planned or structured design process, but for more complex designs, teams of designers following a systematic approach with intelligently guided computer simulation are becoming increasingly common.

In integrated circuit design automation, the term "circuit design" often refers the step of the design cycle which outputs the schematics of the integrated circuit. Typically this is the step between logic design and physical design.

Formal circuit design usually involves the following stages:

- sometimes, writing the requirement specification after liaising with the customer
- writing a technical proposal to meet the requirements of the customer specification
- synthesising on paper a schematic circuit diagram, an abstract electrical or electronic circuit that will meet the specifications
- calculating the component values to meet the operating specifications under specified conditions
- performing simulations to verify the correctness of the design
- building a breadboard or other prototype version of the design and testing against specification
- making any alterations to the circuit to achieve compliance
- choosing a method of construction as well as all the parts and materials to be used
- presenting component and layout information to draughtspersons, and layout and mechanical engineers, for prototype production
- testing or type-testing a number of prototypes to ensure compliance with customer requirements
- signing and approving the final manufacturing drawings

- post-design services (obsolescence of components etc.)

Specification

The process of circuit design begins with the specification, which states the functionality that the finished design must provide, but does not indicate how it is to be achieved . The initial specification is basically a technically detailed description of what the customer wants the finished circuit to achieve and can include a variety of electrical requirements, such as what signals the circuit will receive, what signals it must output, what power supplies are available and how much power it is permitted to consume. The specification can (and normally does) also set some of the physical parameters that the design must meet, such as size, weight, moisture resistance, temperature range, thermal output, vibration tolerance and acceleration tolerance.

As the design process progresses the designer(s) will frequently return to the specification and alter it to take account of the progress of the design. This can involve tightening specifications that the customer has supplied, and adding tests that the circuit must pass in order to be accepted. These additional specifications will often be used in the verification of a design. Changes that conflict with or modify the customer's original specifications will almost always have to be approved by the customer before they can be acted upon.

Correctly identifying the customer needs can avoid a condition known as 'design creep' which occurs in the absence of realistic initial expectations, and later by failing to communicate fully with the client during the design process. It can be defined in terms of its results; "at one extreme is a circuit with more functionality than necessary, and at the other is a circuit having an incorrect functionality". (DeMers, 1997) Nevertheless some changes can be expected and it is good practice to keep options open for as long as possible because it's easier to remove spare elements from the circuit later on than it is to put them in.

Design

The design process involves moving from the specification at the start, to a plan that contains all the information needed to be physically constructed at the end, this normally happens by passing through a number of stages, although in very simple circuit it may be done in a single step. The process normally begins with the conversion of the specification into a block diagram of the various functions that the circuit must perform, at this stage the contents of each block are not considered, only what each block must do, this is sometimes referred to as a "black box" design. This approach allows the possibly very complicated task to be broken into smaller tasks which may either by tackled in sequence or divided amongst members of a design team.

Each block is then considered in more detail, still at an abstract stage, but with a lot more focus on the details of the electrical functions to be provided. At this or later stages it is

common to require a large amount of research or mathematical modeling into what is and is not feasible to achieve. The results of this research may be fed back into earlier stages of the design process, for example if it turns out one of the blocks cannot be designed within the parameters set for it, it may be necessary to alter other blocks instead. At this point it is also common to start considering both how to demonstrate that the design does meet the specifications, and how it is to be tested (which can include self diagnostic tools).

Finally the individual circuit components are chosen to carry out each function in the overall design, at this stage the physical layout and electrical connections of each component are also decided, this layout commonly taking the form of artwork for the production of a printed circuit board or Integrated circuit. This stage is typically extremely time consuming because of the vast array of choices available. A practical constraint on the design at this stage is that of standardization, while a certain value of component may be calculated for use in some location in a circuit, if that value cannot be purchased from a supplier, then the problem has still not been solved. To avoid this a certain amount of 'catalog engineering' can be applied to solve the more mundane tasks within an overall design.

Costs

Proper design philosophy and structure incorporates economic and technical considerations and keeps them in balance at all times, and right from the start. Balance is the key concept here; just as many delays and pitfalls can come from ill considered cost cutting as with cost overruns. Good accounting tools (and a design culture that fosters their use) is imperative for a successful project. "Manufacturing costs shrink as design costs soar," is often quoted as a truism in circuit design, particularly for ICs.

Verification and testing

Once a circuit has been designed, it must be both verified and tested. Verification is the process of going through each stage of a design and ensuring that it will do what the specification requires it to do. This is frequently a highly mathematical process and can involve large-scale computer simulations of the design. In any complicated design it is very likely that problems will be found at this stage and may involve a large amount of the design work be redone in order to fix them.

Testing is the real-world counterpart to verification, testing involves physically building at least a prototype of the design and then (in combination with the test procedures in the specification or added to it) checking the circuit really does do what it was designed to.

Prototyping

Prototyping is a means of exploring ideas before an investment is made in them. Depending on the scope of the prototype and the level of detail required, prototypes can

be built at any time during the project. Sometimes they are created early in the project, during the planning and specification phase, commonly using a process known as breadboarding; that's when the need for exploration is greatest, and when the time investment needed is most viable. Later in the cycle packaging mock-ups are used to explore appearance and usability, and occasionally a circuit will need to be modified to take these factors into account.

Results

As circuit design is the process of working out the physical form that an electronic circuit will take, the result of the circuit design process is the instructions on how to construct the physical electronic circuit. This will normally take the form of blueprints describing the size, shape, connectors, etc in use, and artwork or CAM file for manufacturing a printed circuit board or Integrated circuit.

Documentation

Any commercial design will normally also include an element of documentation, the precise nature of this documentation will vary according to the size and complexity of the circuit as well as the country in which it is to be used. As a bare minimum the documentation will normally include at least the specification and testing procedures for the design and a statement of compliance with current regulations. In the EU this last item will normally take the form of a CE Declaration listing the European directives complied with and naming an individual responsible for compliance.

Circuit diagram

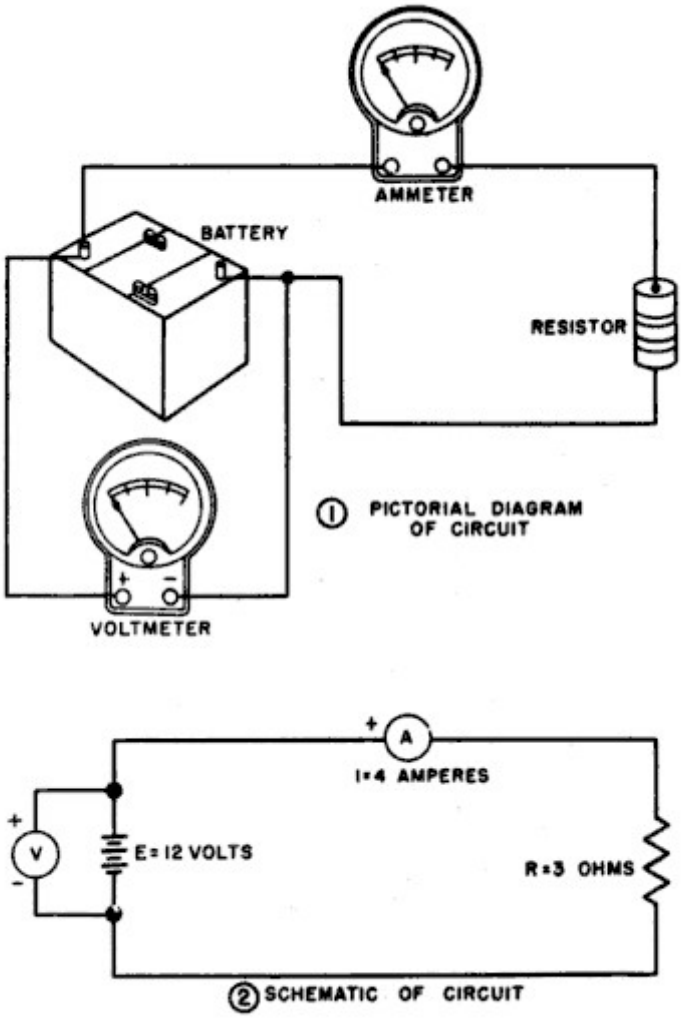
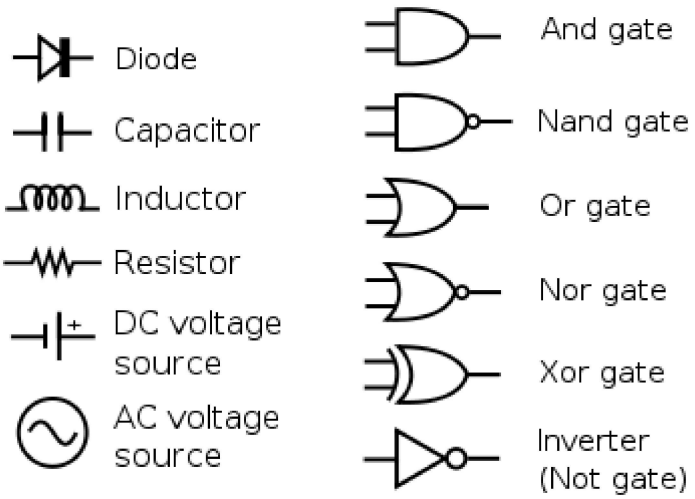
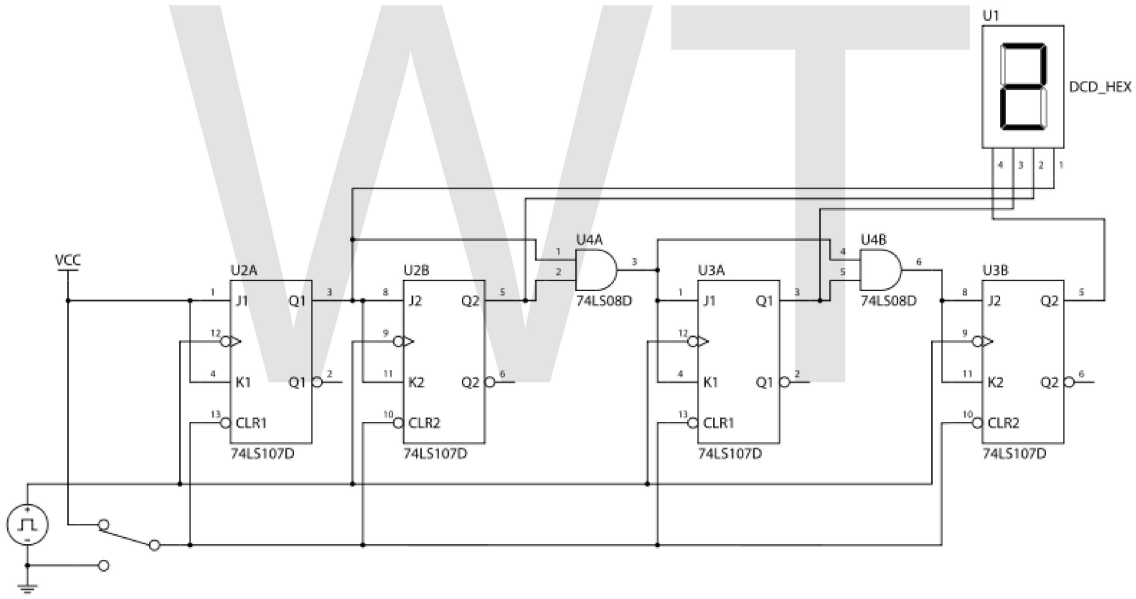


Figure 48. Diagram of a basic circuit.

Comparison of pictorial and schematic styles of circuit diagrams



Common schematic diagram symbols (US symbols)



The circuit diagram for a four-bit TTL counter, a type of state machine

A **circuit diagram** (also known as an **electrical diagram**, **elementary diagram**, or **electronic schematic**) is a simplified conventional graphical representation of an electrical circuit. A pictorial circuit diagram uses simple images of components, while a schematic diagram shows the components of the circuit as simplified standard symbols; both types show the connections between the devices, including power and signal connections. Arrangement of the components interconnections on the diagram does not correspond to their physical locations in the finished device.

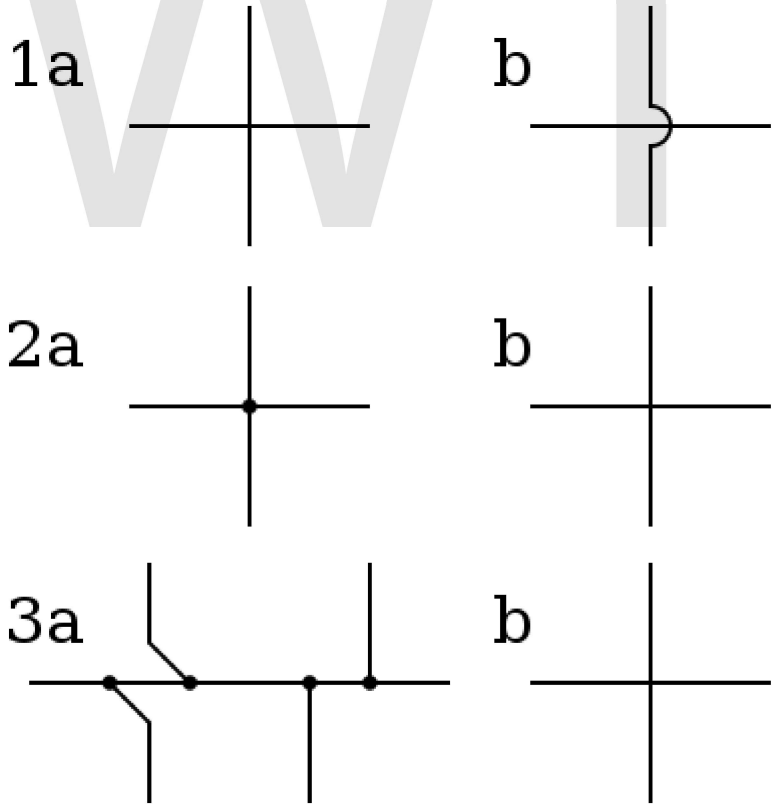
Unlike a block diagram or layout diagram, a circuit diagram shows the actual wire connections being used. The diagram does not show the physical arrangement of

components. A drawing meant to depict what the physical arrangement of the wires and the components they connect is called "artwork" or "layout" or the "physical design."

Circuit diagrams are used for the design (circuit design), construction (such as PCB layout), and maintenance of electrical and electronic equipment.

Symbols

Circuit diagram symbols have differed from country to country and have changed over time, but are now to a large extent internationally standardized. Simple components often had symbols intended to represent some feature of the physical construction of the device. For example, the symbol for a resistor shown here dates back to the days when that component was made from a long piece of wire wrapped in such a manner as to not produce inductance, which would have made it a coil. These wirewound resistors are now used only in high-power applications, smaller resistors being cast from *carbon composition* (a mixture of carbon and filler) or fabricated as an insulating tube or chip coated with a metal film. The internationally standardized symbol for a resistor is therefore now simplified to an oblong, sometimes with the value in ohms written inside, instead of the zig-zag symbol. A less common symbol is simply a series of peaks on one side of the line representing the conductor, rather than back-and-forth as shown here.



Schematic wire junctions:
1. Old style: (a) connection, (b) no connection.

2. One CAD style: (a) connection, (b) no connection.
3. Alternative CAD Style: (a) connection, (b) no connection.

The linkages between leads were once simple crossings of lines; one wire insulated from and "jumping over" another was indicated by it making a little semicircle over the other line. With the arrival of computerized drafting, a connection of two intersecting wires was shown by a crossing with a dot or "blob", and a crossover of insulated wires by a simple crossing without a dot. However, there was a danger of confusing these two representations if the dot was drawn too small or omitted. Modern practice is to avoid using the "crossover with dot" symbol, and to draw the wires meeting at two points instead of one. It is also common to use a hybrid style, showing connections as a cross with a dot while insulated crossings use the semicircle.

On a circuit diagram, the symbols for components are labelled with a descriptor or reference designator matching that on the list of parts. For example, C1 is the first capacitor, L1 is the first inductor, Q1 is the first transistor, and R1 is the first resistor (note that this is not written as a subscript, as in R_1, L_1,…). Often the value or type designation of the component is given on the diagram beside the part, but detailed specifications would go on the parts list.

Detailed rules for reference designations are provided in the International standard IEC 61346.

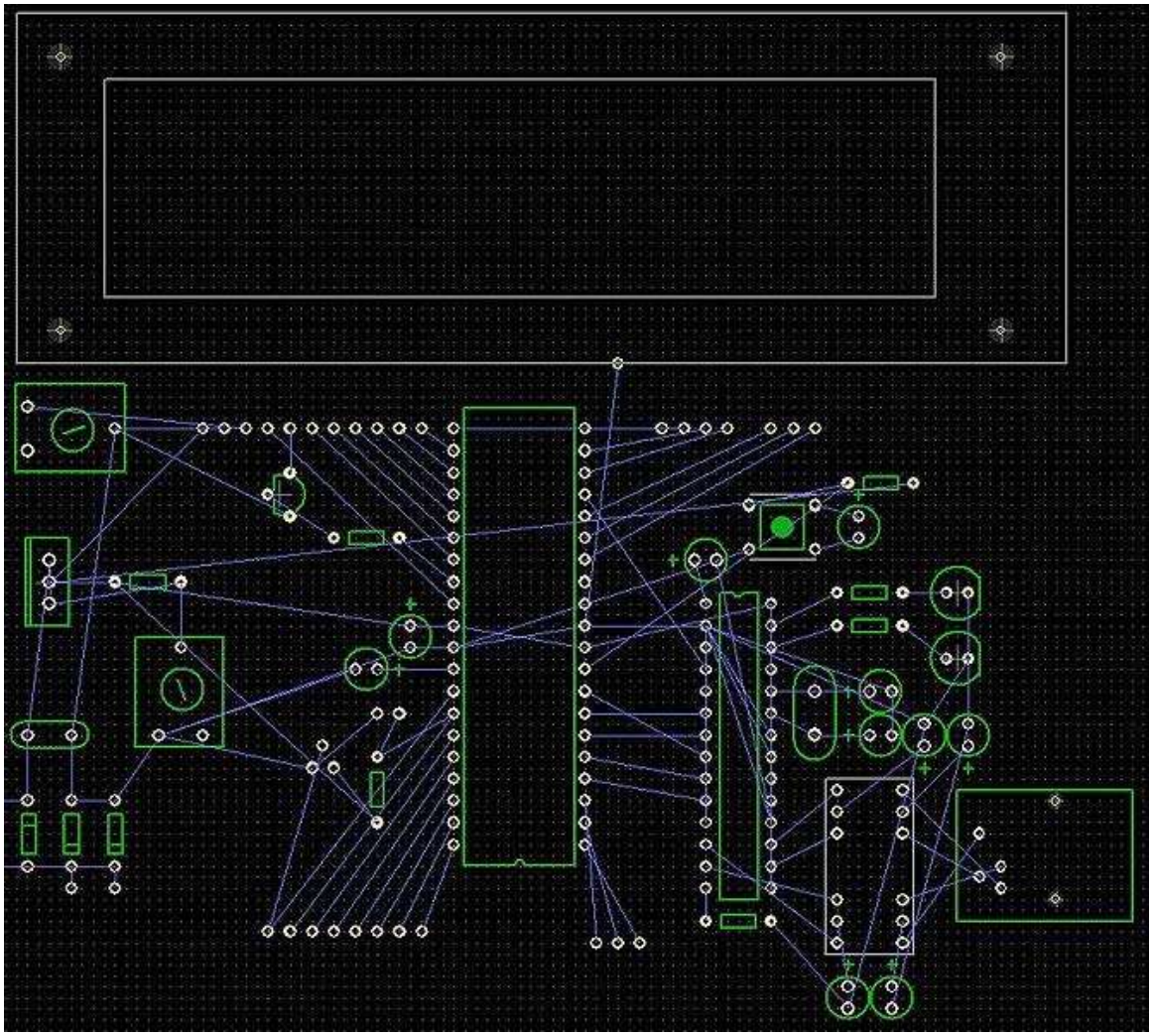
Organization of drawings

It is a usual although not universal convention that schematic drawings are organized on the page from left to right and top to bottom in the same sequence as the flow of the main signal or power path. For example, a schematic for a radio receiver might start with the antenna input at the left of the page and end with the loudspeaker at the right. Positive power supply connections for each stage would be shown towards the top of the page, with grounds, negative supplies, or other return paths towards the bottom. Schematic drawings intended for maintenance may have the principle signal paths highlighted to assist in understanding the signal flow through the circuit. More complex devices have multi-page schematics and must rely on cross-reference symbols to show the flow of signals between the different sheets of the drawing.

Detailed rules for the preparation of circuit diagrams (and other document kinds used in electrotechnology) are provided in the International standard IEC 61082-1.

Relay logic line diagrams (also called ladder logic diagrams) use another common standardized convention for organizing schematic drawings, with a vertical power supply "rail" on the left and another on the right, and components strung between them like the rungs of a ladder.

Artwork



A rat's nest

Once the schematic has been made, it is converted into a layout that can be fabricated onto a printed circuit board (PCB). The layout is usually started by the process of schematic capture. The result is what is known as a rat's nest. The rat's nest is a jumble of wires (lines) criss-crossing each other to their destination nodes. These wires are routed either manually or by the use of electronics design automation (EDA) tools. The EDA tools arrange and rearrange the placement of components and find paths for tracks to connect various nodes. This results in the final layout artwork for the integrated circuit or printed circuit board.

A generalized design flow would be as:

> Schematic → Schematic capture → Rat's nest → Routing → Artwork → PCB development & etching → Component mounting → Testing

Design closure

Design closure is the process by which a VLSI design is modified from its initial description to meet a growing list of design constraints and objectives.

Introduction

Every chip starts off as someone's idea of a good thing: "If we can make a part that performs function X, we will all be rich!" Once the concept is established, someone from marketing says "To make this chip profitably, it must cost $C and run at frequency F." Someone from manufacturing says "To meet this chip's targets, it must have a yield of Y%." Someone from packaging says "It must fit in the P package and dissipate no more than W watts." Eventually, the team generates an extensive list of all the constraints and objectives they must meet to manufacture a product that can be sold profitably. The management then forms a design team, which consists of chip architects, logic designers, functional verification engineers, physical designers, and timing engineers, and assigns them to create a chip to the specifications.

Constraints vs Objectives

The distinction between constraints and objectives is straightforward: a constraint is a design target that must be met for the design to be successful. For example, a chip may be required to run at a specific frequency so it can interface with other components in a system. In contrast, an objective is a design target where more (or less) is better. For example, yield is generally an objective, which is maximized to lower manufacturing cost. For the purposes of design closure, the distinction between constraints and objectives is not important; here we uses the words interchangeably.

Evolution of the Design Closure Flow

Designing a chip used to be a much simpler task. In the early days of VLSI, a chip consisted of a few thousand logic circuits that performed a simple function at speeds of a few MHz. Design closure was simple: if all of the necessary circuits and wires "fit", the chip would perform the desired function.

Modern design closure has grown orders of magnitude more complex. Modern logic chips can have tens to hundreds of millions of logic elements switching at speeds of several GHz. This improvement has been driven by Moore's law of scaling of technology, and has introduced many new design considerations. As a result, a modern VLSI designer must consider the performance of a chip against a list of dozens of design constraints and objectives including performance, power, signal integrity, reliability, and yield. In response to this growing list of constraints, the design closure flow has evolved from a simple linear list of tasks to a very complex, highly iterative flow such as the following simplified ASIC design flow:

Reference ASIC Design Flow

- Concept phase: Functional objectives and architecture of a chip are developed.
- Logic design: Architecture is implemented in a register transfer level (RTL) language, then simulated to verify that it performs the desired functions. This includes functional verification.
- Floorplanning: The RTL of the chip is assigned to gross regions of the chip, input/output (I/O) pins are assigned and large objects (arrays, cores, etc.) are placed.
- Logic synthesis: The RTL is mapped into a gate-level netlist in the target technology of the chip.
- Design for Testability: The test structures like scan chains are inserted.
- Placement: The gates in the netlist are assigned to nonoverlapping locations on the chip.
- Logic/placement refinement: Iterative logical and placement transformations to close performance and power constraints.
- Clock insertion: Balanced buffered clock trees are introduced into the design.
- Routing: The wires that connect the gates in the netlist are added.
- Postwiring optimization: Remaining performance, noise, and yield violations are removed.
- Design for manufacturability: The design is modified, where possible, to make it as easy as possible to produce.
- Signoff checks: Since errors are expensive, time consuming and hard to spot, extensive error checking is the rule, making sure the mapping to logic was done correctly, and checking that the manufacturing rules were followed faithfully.
- Tapeout and mask generation: the design data is turned into photomasks in mask data preparation.

Evolution of design constraints

The purpose of the flow is to take a design from concept phase to working chip. The complexity of the flow is a direct result of the addition and evolution of the list of design closure constraints. To understand this evolution it is important to understand the life cycle of a design constraint. In general, design constraints influence the design flow via the following five-stage evolution:

- Early warnings: Before chip issues begin occurring, academics and industry visionaries make dire predictions about the future impact of some new technology effect.
- Hardware problems: Sporadic hardware failures start showing up in the field due to the new effect. Postmanufacturing redesign and hardware re-spins are required to get the chip to function.
- Trial and error: Constraints on the effect are formulated and used to drive postdesign checking. Violations of the constraint are fixed manually.

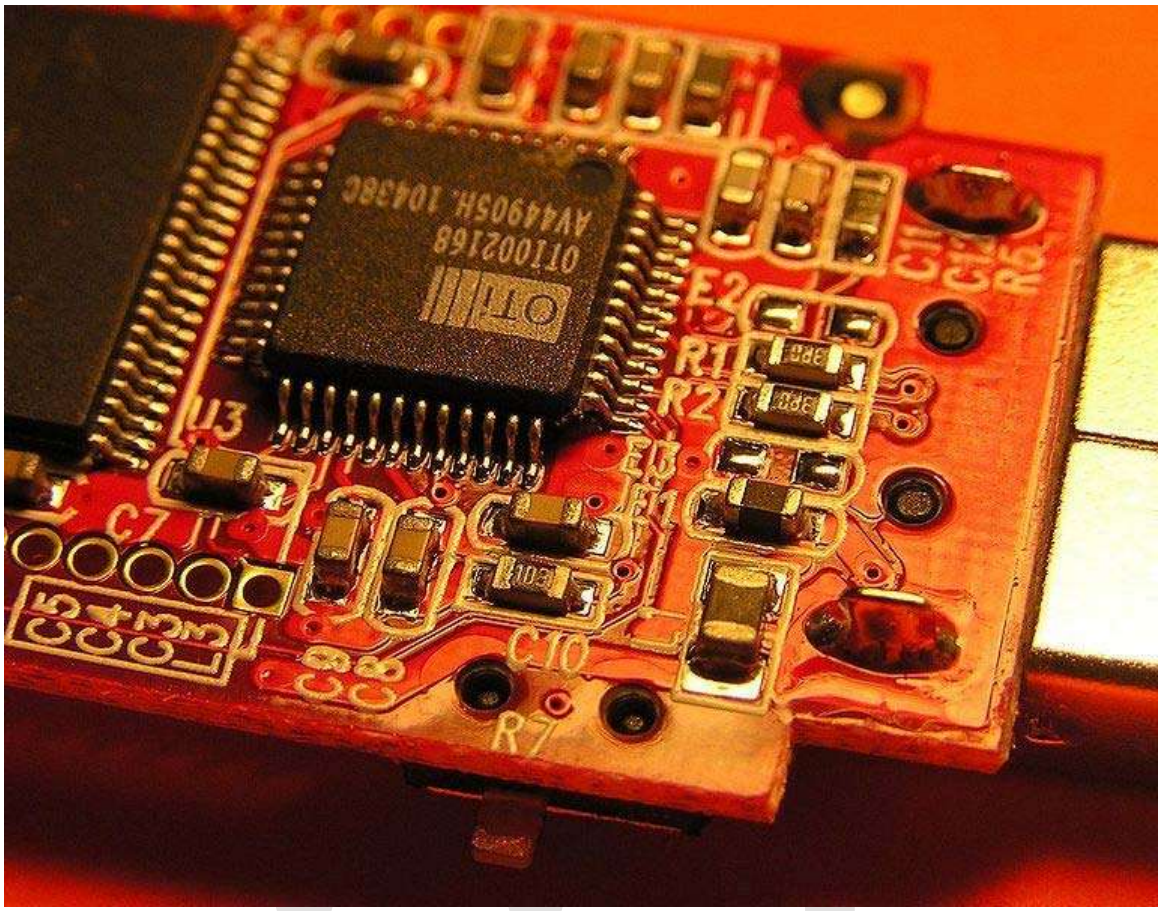
- Find and repair: Large number of violations of the constraint drives the creation of automatic postdesign analysis and repair flows.
- Predict and prevent: Constraint checking moves earlier in the flow using predictive estimations of the effect. These drive optimizations to prevent violations of the constraint.

A good example of this evolution can be found in the signal integrity constraint. In the mid-1990s (180 nm node), industry visionaries were describing the impending dangers of coupling noise long before chips were failing. By the mid-late 1990s, noise problems were cropping up in advanced microprocessor designs. By 2000, automated noise analysis tools were available and were used to guide manual fix-up. The total number of noise problems identified by the analysis tools identified by the flow quickly became too many to correct manually. In response, CAD companies developed the noise avoidance flows that are currently in use in the industry.

At any point in time, the constraints in the design flow are at different stages of their life cycle. At the time of this writing, for example, performance optimization is the most mature and is well into the fifth phase with the widespread use of timing-driven design flows. Power- and defect-oriented yield optimization is well into the fourth phase; power supply integrity, a type of noise constraint, is in the third phase; circuit-limited yield optimization is in the second phase, etc. A list of the first-phase impending constraint crises can always by found in the International Technology Roadmap for Semiconductors (ITRS) 15-year-outlook technology roadmaps.

As a constraint matures in the design flow, it tends to work its way from the end of the flow to the beginning. As it does this, it also tends to increase in complexity and in the degree that it contends with other constraints. Constraints tend to move up in the flow due to one of the basic paradoxes of design: accuracy vs. influence. Specifically, the earlier in a design flow a constraint is addressed, the more flexibility there is to address the constraint. Ironically, the earlier one is in a design flow, the more difficult it is to predict compliance. For example, an architectural decision to pipeline a logic function can have a far greater impact on total chip performance than any amount of postrouting fix-up. At the same time, accurately predicting the performance impact of such a change before the chip logic is synthesized, let alone placed or routed, is very difficult. This paradox has shaped the evolution of the design closure flow in several ways. First, it requires that the design flow is no longer composed of a linear set of discrete steps. In the early stages of VLSI it was sufficient to break the design into discrete stages, i.e., first do logic synthesis, then do placement, then do routing. As the number and complexity of design closure constraints has increased, the linear design flow has broken down. In the past if there were too many timing constraint violations left after routing, it was necessary to loop back, modify the tool settings slightly, and reexecute the previous placement steps. If the constraints were still not met, it was necessary to reach further back in the flow and modify the chip logic and repeat the synthesis and placement steps. This type of looping is both time consuming and unable to guarantee convergence i.e., it is possible to loop back in the flow to correct one constraint violation only to find that the correction induced another unrelated violation.

Surface-mount technology



Surface mount components on a flash drive's circuit board. The small rectangular chips with numbers are resistors, while the unmarked small rectangular chips are capacitors. The visible capacitors and resistors are predominantly a combination of 0805 and 0603 package sizes, while the smallest chip capacitors are 0402 size.

Surface mount technology (**SMT**) is a method for constructing electronic circuits in which the components (SMC, or Surface Mounted Components) are mounted directly onto the surface of printed circuit boards (PCBs). Electronic devices so made are called *surface mount devices* or **SMD**s. In the industry it has largely replaced the through-hole technology construction method of fitting components with wire leads into holes in the circuit board.

An SMT component is usually smaller than its through-hole counterpart because it has either smaller leads or no leads at all. It may have short pins or leads of various styles, flat contacts, a matrix of solder balls (BGAs), or terminations on the body of the component.

History

Surface mount technology was developed in the 1960s and became widely used in the late 1980s. Much of the pioneering work in this technology was by IBM. The design approach first demonstrated by IBM in 1960 in a small-scale computer was later applied in the Launch Vehicle Digital Computer used in the Instrument Unit that guided all Saturn IV and Saturn V vehicles. Components were mechanically redesigned to have small metal tabs or end caps that could be directly soldered to the surface of the PCB. Components became much smaller and component placement on both sides of a board became far more common with surface mounting than through-hole mounting, allowing much higher circuit densities. Often only the solder joints hold the parts to the board, although parts on the bottom or "second" side of the board are temporarily secured with a dot of adhesive as well. Surface mounted devices (**SMD**s) are usually made physically small and lightweight for this reason. Surface mounting lends itself well to a high degree of automation, reducing labor cost and greatly increasing production rates. SMDs can be one-quarter to one-tenth the size and weight, and one-half to one-quarter the cost of equivalent through-hole parts.

SMD Terms

SMp DIALECT	Expanded Form
SMD	Surface Mount Devices (active, passive and electromechanical components)
SMT	Surface Mount Technology (assembling and montage technology)
SMA	Surface Mount Assembly (module assembled with SMT)
SMC	Surface Mount Components (components for SMT)
SMP	Surface Mount Packages (SMD case forms)
SME	Surface Mount Equipment (SMT assembling machines)
SO	Small Outline (4 to 28 pins)
VSO	Very Small Outline (40 pins)
SOP	Small Outline Package (case)
SOD	Small Outline Diode
SOT	Small Outline Transistor
SOIC	Small Outline Integrated Circuit
CC	Chip Carrier
LCC	Leadless Chip Carrier
PLCC	Plastic Leadless Chip Carrier
LCCC	Leadless Ceramic Chip Carrier
MELF	Metal Electrode Face Bonding

MINI MELF Mini Metal Electrode Face Bonding
MICRO MELF Micro Metal Electrode Face Bonding

Assembly techniques

Assembly line of SMT placement machines

Where components are to be placed, the printed circuit board has flat, usually tin-lead, silver, or gold plated copper pads without holes, called *solder pads*. Solder paste, a sticky mixture of flux and tiny solder particles, is first applied to all the solder pads with a stainless steel or nickel stencil using a screen printing process. After screen printing, the boards then proceed to the pick-and-place machines, where they are placed on a conveyor belt. The components to be placed on the boards are usually delivered to the production line in either paper/plastic tapes wound on reels or plastic tubes. Some large integrated circuits are delivered in static-free trays. Numerical control pick-and-place machines remove the parts from the tapes, tubes or trays and place them on the PCB.

The boards are then conveyed into the reflow soldering oven. They first enter a pre-heat zone, where the temperature of the board and all the components is gradually, uniformly raised. The boards then enter a zone where the temperature is high enough to melt the solder particles in the solder paste, bonding the component leads to the pads on the circuit board. The surface tension of the molten solder helps keep the components in place, and if the solder pad geometries are correctly designed, surface tension automatically aligns

the components on their pads. There are a number of techniques for reflowing solder. One is to use infrared lamps; this is called infrared reflow. Another is to use a hot gas convection. Another technology which is becoming popular again is special fluorocarbon liquids with high boiling points which use a method called vapor phase reflow. Due to environmental concerns, this method was falling out of favor until lead-free legislation was introduced which requires tighter controls on soldering. Currently, at the end of 2008, convection soldering is the most popular reflow technology using either standard air or nitrogen gas. Each method has its advantages and disadvantages. With infrared reflow, the board designer must lay the board out so that short components don't fall into the shadows of tall components. Component location is less restricted if the designer knows that vapor phase reflow or convection soldering will be used in production. Following reflow soldering, certain irregular or heat-sensitive components may be installed and soldered by hand, or in large scale automation, by focused infrared beam (FIB) or localized convection equipment.

If the circuit board is double sided then this printing, placement, reflow process may be repeated using either solder paste or glue to hold the components in place. If glue is used then the parts must be soldered later using a wave soldering process.

After soldering, the boards may be washed to remove flux residues and any stray solder balls that could short out closely spaced component leads. Rosin flux is removed with fluorocarbon solvents, high flash point hydrocarbon solvents, or low flash solvents e.g. limonene (derived from orange peels) which require extra rinsing or drying cycles. Water soluble fluxes are removed with deionized water and detergent, followed by an air blast to quickly remove residual water. However, most electronic assemblies are made using a "No-Clean" process where the flux residues are designed to be left on the circuit board [Benign]. This saves the cost of cleaning, speeds up the whole process, and reduces waste.

Finally, the boards are visually inspected for missing or misaligned components and solder bridging. If needed, they are sent to a rework station where a human operator corrects any errors. They are then sent to the testing stations (in-circuit testing and/or functional testing) to verify that they operate correctly.

Main advantages

The main advantages of SMT over the older through-hole technique are:

- Smaller components. Smallest is currently 0.4 x 0.2 mm. (.01" x .005" - 01005)
- Much higher number of components and many more connections per component.
- Fewer holes need to be drilled through abrasive boards.
- Simpler automated assembly.
- Small errors in component placement are corrected automatically (the surface tension of the molten solder pulls the component into alignment with the solder pads).
- Components can be placed on both sides of the circuit board.

- Lower resistance and inductance at the connection (leading to better performance for high frequency parts).
- Better mechanical performance under shake and vibration conditions.
- SMT parts generally cost less than through-hole parts.
- Fewer unwanted RF signal effects in SMT parts when compared to leaded parts, yielding better predictability of component characteristics.
- Faster assembly. Some placement machines are capable of placing more than 136,000 components per hour.

Main disadvantages

- The manufacturing processes for SMT are much more sophisticated than through-hole boards, raising the initial cost and time of setting up for production.
- Manual prototype assembly or component-level repair is more difficult (more so without a steady hand and the right tools) given the very small sizes and lead spacings of many SMDs.
- SMDs can't be used directly with breadboards (a quick snap-and-play prototyping tool), requiring either a custom PCB for every prototype or the mounting of the SMD upon a pin-leaded carrier. For prototyping around a specific SMD component, a less-expensive breakout board may be used. Additionally, stripboard style protoboards can be used, some of which include pads for standard sized SMD comportments.
- SMDs' solder connections may be damaged by potting compounds going through thermal cycling.

Reworking defective SMD components

Tweezer soldering iron used on '0805' SMD component

Defective surface mount components can be repaired in two ways: by using soldering irons (depends on the kind and number of connections) or using a professional rework system. In most cases a rework system is the first choice because the human influence on the rework result is very low. Generally, two essential soldering methods can be distinguished: infrared soldering and soldering with hot gas.

Benefits and disadvantages of different soldering methods

Infrared soldering:

During infrared soldering, the energy for heating up the solder joint will be transmitted by long or short wave electromagnetic radiation.

Benefits

- Easy setup
- No compressed air required
- No component-specific nozzles (low costs)
- Fast reaction of infrared source (depends on used system)

Disadvantages

- Central areas will be heated more than peripheral areas
- Temperature can hardly be controlled, peaks cannot be ruled out
- Covering of the neighboured components is necessary to prevent damage, which requires additional time for every board
- Surface temperature depends on the component's reflection characteristics: dark surfaces will be heated more than lighter surfaces
- The temperature additionally depends on the surface shape. Convective loss of energy will reduce the temperature of the component
- No reflow atmosphere possible

Conventional hot gas soldering

During hot gas soldering, the energy for heating up the solder joint will be transmitted by a gaseous medium. This can be air or inert gas (nitrogen).

Benefits

- Simulating reflow oven atmosphere
- Switching between hot gas and nitrogen (economic use)
- Standard and component-specific nozzles allow high reliability and reduced process time
- Allow reproducible soldering profiles
- Efficient heating, large heat amounts can be transmitted
- Even heating of the affected board area
- Temperature of the component will never exceed the adjusted gas temperature
- Rapid cool down after reflow, resulting in small-grained solder joints (depends on used system)

Disadvantages

- Thermal capacity of the heat generator results in slow reaction whereby thermal profiles can be distorted (depends on used system)

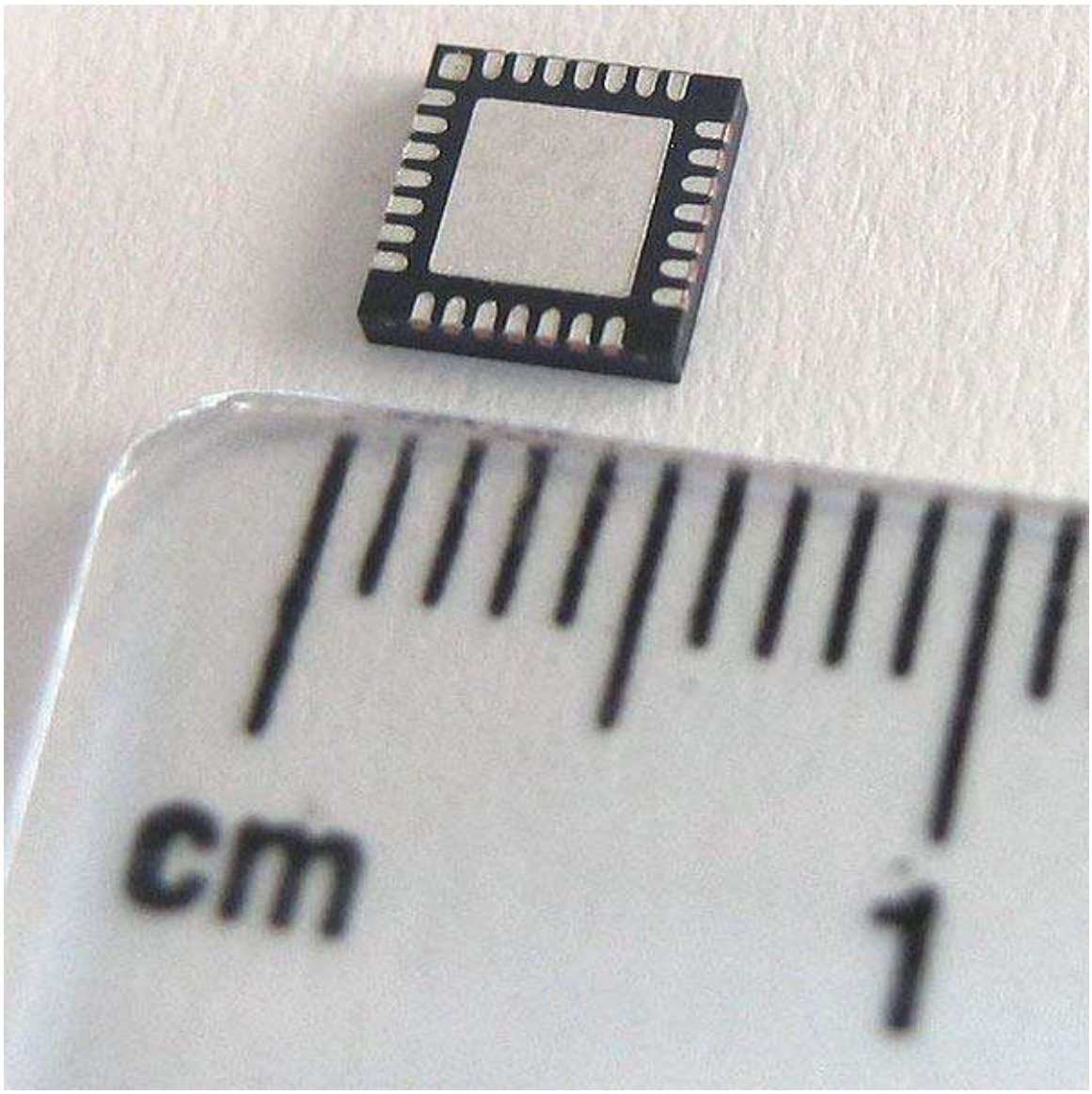
A rework process usually undoes some type of error, either human or machine-generated, and includes the following steps:

- Melt solder and component removal
- Residual solder removal
- Printing of solder paste on PCB, direct component printing or dispensing
- Placement and reflow of new component.

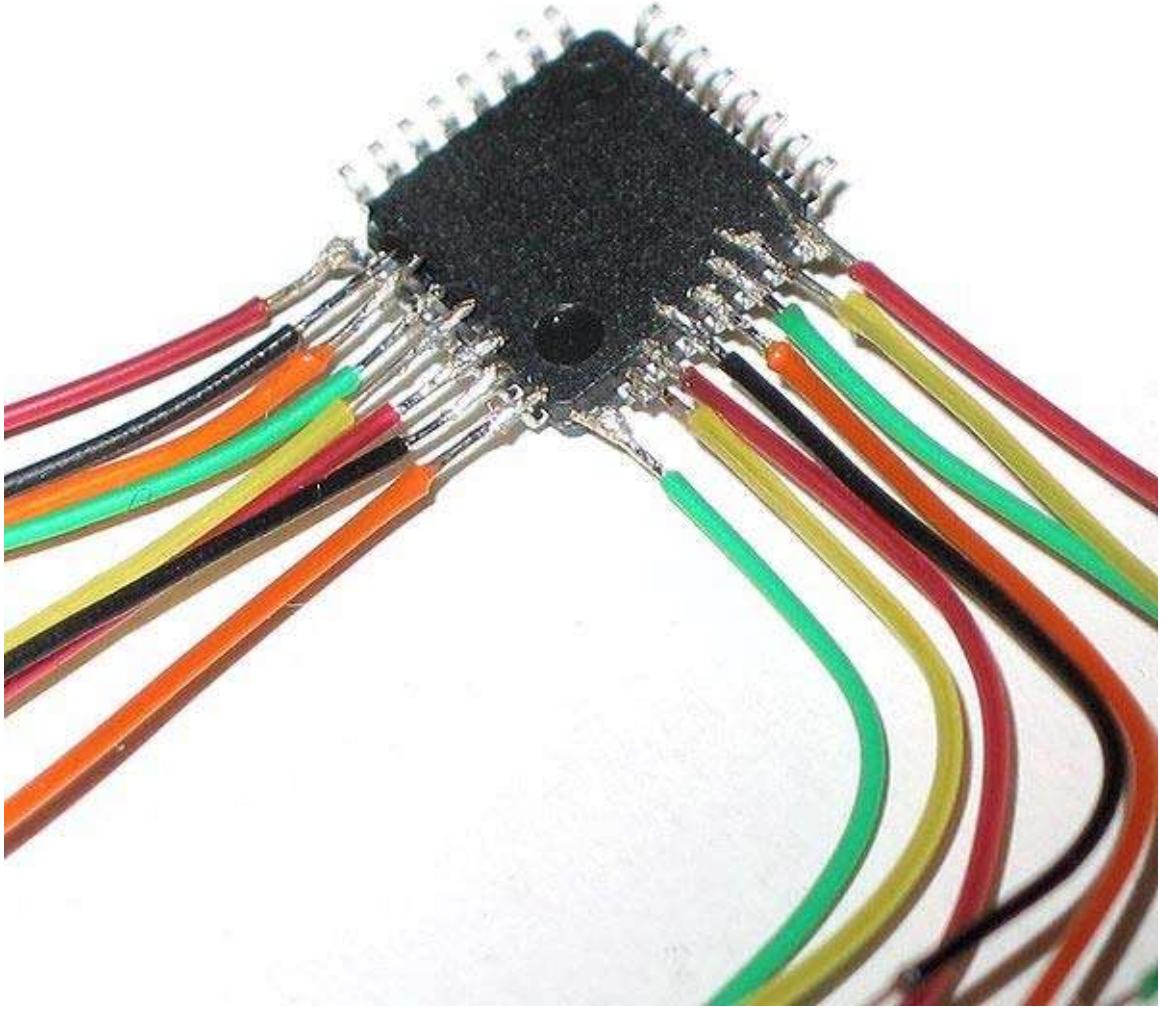
Sometimes hundreds or thousands of the same part need to be repaired. Such errors, if due to assembly, are often caught during the process, however a whole new level of rework arises when component failure is discovered too late, and perhaps unnoticed until

the end user experiences them. Rework may also be used if high-value products require revisions, and re-engineering, perhaps to change a single firmware based component, may revive a once obsolete product. These tasks require a rework operation specifically designed to repair/replace components in volume.

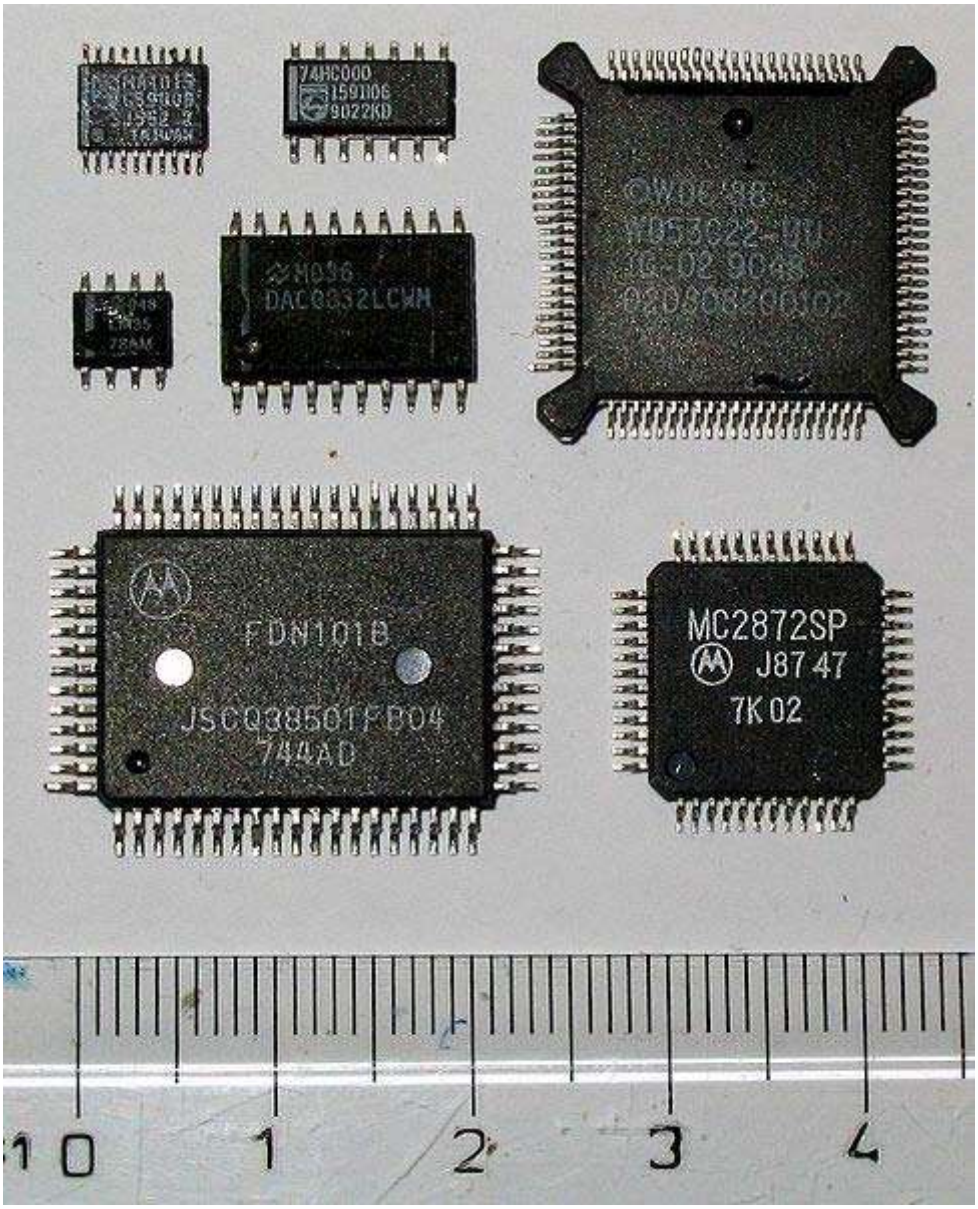
Package sizes



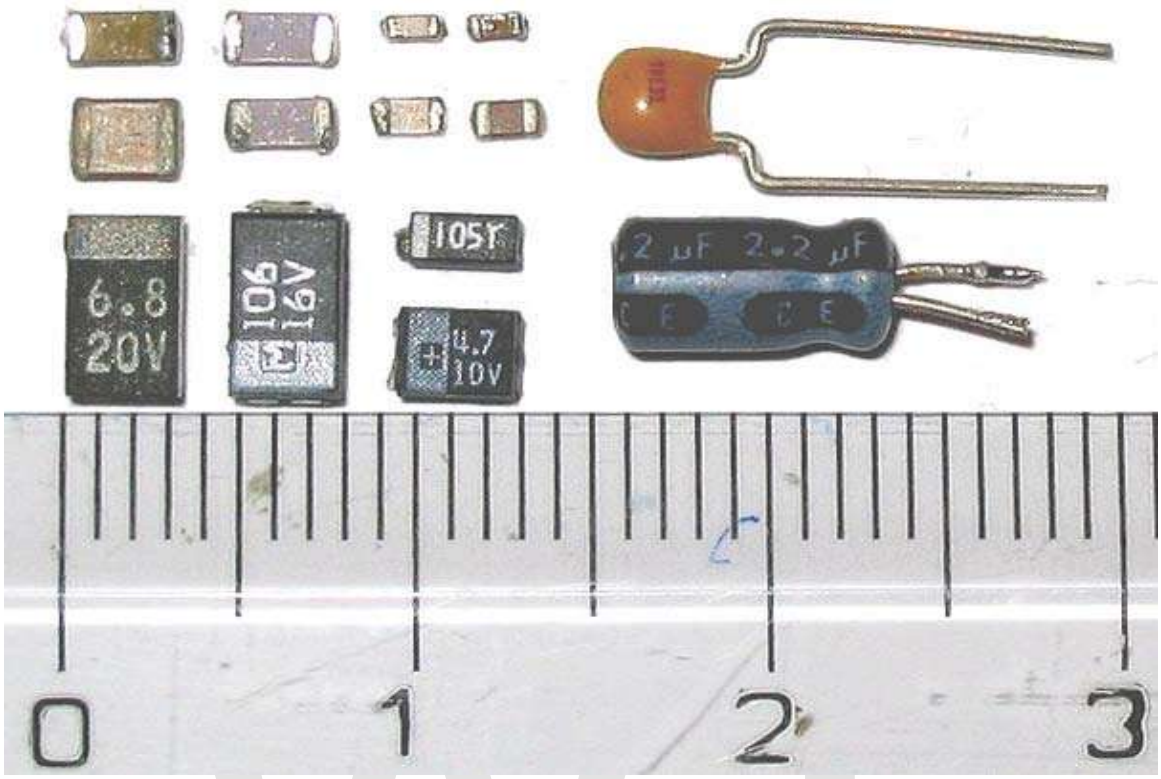
MLP package 28-pin chip, upside down to show contacts



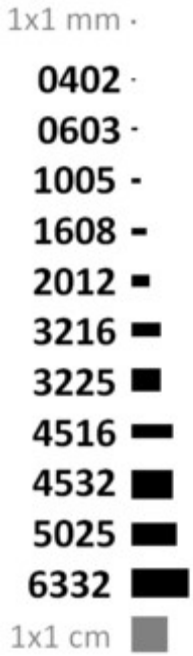
32-pin MQFP chip with manually-soldered wires attached for prototyping. The same effect can be achieved using commercially available breakout boards.



Various SMD chips, desoldered



SMD capacitors (on the left) with two through-hole capacitors (on the right)



Example of typical metric sizes (grey squares shown for reference)

Surface mount components are usually smaller than their counterparts with leads, and are designed to be handled by machines rather than by humans. The electronics industry has standardized package shapes and sizes (the leading standardisation body is JEDEC). These include:

- **Two-terminal packages**
  - Rectangular passive components (mostly resistors and capacitors):
    - 01005 (0402 metric) : 0.016" × 0.008" (0.4 mm × 0.2 mm) Typical power rating for resistors 1/32 watt
    - 0201 (0603 metric) : 0.024" × 0.012" (0.6 mm × 0.3 mm) Typical power rating for resistors 1/20 watt
    - 0402 (1005 metric) : 0.04" × 0.02" (1.0 mm × 0.5 mm) Typical power rating for resistors 1/16 watt
    - 0603 (1608 metric) : 0.063" × 0.031" (1.6 mm × 0.8 mm) Typical power rating for resistors 1/16 watt
    - 0805 (2012 metric) : 0.08" × 0.05" (2.0 mm × 1.25 mm) Typical power rating for resistors 1/10 or 1/8 watt
    - 1206 (3216 metric) : 0.126" × 0.063" (3.2 mm × 1.6 mm) Typical power rating for resistors 1/4 watt
    - 1210 (3225 metric) : 0.126" × 0.1" (3.2 mm × 2.5 mm) Typical power rating for resistors 1/2 watt
    - 1808 (4516 metric) : 0.177" × 0.063" (4.5 mm × 1.6 mm)
    - 1812 (4532 metric) : 0.18" × 0.12" (4.5 mm × 3.2 mm) Typical power rating for resistors 1/2 watt
    - 2010 (5025 metric) : 0.2" × 0.1" (5.0 mm × 2.5 mm) Typical power rating for resistors 1/2 watt
    - 2512 (6332 metric) : 0.25" × 0.12" (6.35 mm × 3.0 mm) Typical power rating for resistors 1 watt
  - Tantalum capacitors :
    - EIA 3216-12 (Kemet S, AVX S): 3.2 mm × 1.6 mm × 1.2 mm
    - EIA 3216-18 (Kemet A, AVX A): 3.2 mm × 1.6 mm × 1.8 mm
    - EIA 3528-12 (Kemet T, AVX T): 3.5 mm × 2.8 mm × 1.2 mm
    - EIA 3528-21 (Kemet B, AVX B): 3.5 mm × 2.8 mm × 2.1 mm
    - EIA 6032-15 (Kemet U, AVX W): 6.0 mm × 3.2 mm × 1.5 mm
    - EIA 6032-28 (Kemet C, AVX C): 6.0 mm × 3.2 mm × 2.8 mm
    - EIA 7260-38 (Kemet E, AVX V): 7.2 mm × 6.0 mm × 3.8 mm
    - EIA 7343-20 (Kemet V, AVX Y): 7.3 mm × 4.3 mm × 2.0 mm
    - EIA 7343-31 (Kemet D, AVX D): 7.3 mm × 4.3 mm × 3.1 mm
    - EIA 7343-43 (Kemet X, AVX E): 7.3 mm × 4.3 mm × 4.3 mm
  - Aluminium capacitors :
    - (Panasonic/CDE A, Chemi-Con B): 3.3 mm × 3.3 mm
    - (Panasonic B, Chemi-Con D): 4.3 mm × 4.3 mm
    - (Panasonic C, Chemi-Con E): 5.3 mm × 5.3 mm
    - (Panasonic D, Chemi-Con F): 6.6 mm × 6.6 mm
    - (Panasonic E/F, Chemi-Con H): 8.3 mm × 8.3 mm
    - (Panasonic G, Chemi-Con J): 10.3 mm × 10.3 mm

        - (Chemi-Con K): 13.0 mm × 13.0 mm
        - (Panasonic H): 13.5 mm × 13.5 mm
        - (Panasonic J, Chemi-Con L): 17.0 mm × 17.0 mm
        - (Panasonic K, Chemi-Con M): 19.0 mm × 19.0 mm
    - SOD: Small Outline Diode
        - SOD-523: 1.25 × 0.85 × 0.65 mm
        - SOD-323: 1.7 × 1.25 × 0.95 mm
        - SOD-123: 3.68 × 1.17 × 1.60 mm
        - SOD-80C: 3.50 × 1.50 × More info
    - MELF — Metal Electrode Leadless Face — (mostly resistors and diodes): Barrel shaped components, dimensions do not match those of rectangular references for identical codes.
        - MicroMelf (MMU) Size 0102: *L*:2.2 mm *D*:1.1 mm (solder pad fits rectangular 0805) 1/5 watt (0.2 W) 100 V
        - MiniMelf (MMA) Size 0204: *L*:3.6 mm *D*:1.4 mm (solder pad fits rectangular 1206) 1/4 watt (0.25 W) 200 V
        - Melf (MMB) Size 0207: *L*:5.8 mm *D*:2.2 mm 1 watt (1.0 W) 500 V
- **Three terminal packages**
    - SOT: small-outline transistor, with three terminals
        - SOT-223: 6.7 mm × 3.7 mm × 1.8 mm body - four terminals, one of which is a large heat-transfer pad
        - SOT-89: 4.5 mm × 2.5 mm × 1.5 mm body - three terminals, one of which extends to a large heat-transfer pad
        - SOT-23: 2.9 mm × 1.3/1.75 mm × 1.3 mm body - three terminals for a transistor
        - SOT-323 (SC-70): 2 mm × 1.25 mm × 0.95 mm body - three terminals
        - SOT-416 (SC-75): 1.6 mm × 0.8 mm × 0.8 mm body - three terminals
        - SOT-723: 1.2 mm × 0.8 mm × 0.5 mm body - three terminals - flat lead
        - SOT-883 (SC-101): 1 mm × 0.6 mm × 0.5 mm body - three terminals - leadless
    - DPAK (TO-252): discrete packaging. Developed by Motorola to house higher powered devices. Comes in three- or five-terminal versions
    - D2PAK (TO-263) - bigger than the DPAK; basically a surface mount equivalent of the TO220 through-hole package. Comes in 3, 5, 6, 7, 8 or 9-terminal versions
    - D3PAK (TO-268) - even larger than D2PAK
- **Five and six terminal packages**
    - SOT: small-outline transistor, with more than three terminals
        - SOT-23-5: 2.9 mm × 1.3/1.75 mm × 1.3 mm body - five terminals
        - SOT-23-6: 2.9 mm × 1.3/1.75 mm × 1.3 mm body - six terminals
        - SOT-23-8: 2.9 mm × 1.3/1.75 mm × 1.3 mm body - eight terminals

        - SOT-363 (SC-88, SC-70-6): 2 mm × 1.25 mm × 0.95 mm body - six terminals
        - SOT-666: 1.6 mm × 1.25 mm × 0.55 mm body - six terminals
- **Packages with higher terminal count** *(drawings of most of the following packages can be found on)*
    - Dual-in-line
        - Small-outline integrated circuit (SOIC) - small-outline integrated circuit, dual-in-line, 8 or more pins, gull-wing lead form, pin spacing 1.27 mm
        - J-Leaded Small Outline Package (SOJ) - the same as SOIC except J-leaded
        - TSOP - thin small-outline package, thinner than SOIC with smaller pin spacing of 0.5 mm
        - SSOP - Shrink Small-Outline Package, pin spacing of 0.635 mm or in some cases 0.8 mm
        - TSSOP - Thin Shrink Small-Outline package.
        - QSOP - Quarter-Size Small-Outline package, with pin spacing of 0.635 mm
        - VSOP - Very Small Outline Package, even smaller than QSOP; 0.4, 0.5 mm or 0.65 mm pin spacing
    - Quad-in-line
        - PLCC - plastic leaded chip carrier, square, J-lead, pin spacing 1.27 mm
        - QFP - Quad Flat Package, various sizes, with pins on all four sides
        - LQFP - Low-profile Quad Flat Package, 1.4 mm high, varying sized and pins on all four sides
        - PQFP - plastic quad flat-pack, a square with pins on all four sides, 44 or more pins
        - CQFP - ceramic quad flat-pack, similar to PQFP
        - MQFP - Metric Quad Flat Pack, a QFP package with metric pin distribution
        - TQFP - thin quad flat pack, a thinner version of PQFP
        - QFN - quad flat pack, no-leads, smaller footprint than leaded equivalent
        - LCC - Leadless Chip Carrier, contacts are recessed vertically to "wick-in" solder. Common in aviation electronics because of robustness to mechanical vibration.
        - MLP - Leadframe package with a 0.5 mm contact pitch, no leads
        - PQFN - power quad flat-pack, no-leads, with exposed die-pad[s] for heatsinking
    - Grid arrays
        - PGA - Pin grid array.
        - BGA - ball grid array, with a square or rectangular array of solder balls on one surface, ball spacing typically 1.27 mm

        - LGA - Same Manufacture process of BGA, has Advantage being cooler than BGA by quick Heat Deception feature. No Balls only Patch.
        - FBGA - fine pitch ball grid array, with a square or rectangular array of solder balls on one surface
        - LFBGA - low profile fine pitch ball grid array, with a square or rectangular array of solder balls on one surface, ball spacing typically 0.8 mm
        - TFBGA - thin fine pitch ball grid array, with a square or rectangular array of solder balls on one surface, ball spacing typically 0.5 mm
        - CGA - column grid array, circuit package in which the input and output points are high temperature solder cylinders or columns arranged in a grid pattern.
        - CCGA - ceramic column grid array, circuit package in which the input and output points are high temperature solder cylinders or columns arranged in a grid pattern. The body of the component is ceramic.
        - µBGA - micro-BGA, with ball spacing less than 1 mm
        - LLP - Lead Less Package, a package with metric pin distribution (0.5 mm pitch).
    - Non-packaged devices (although surface mount, these devices require specific process for assembly):
        - COB - chip-on-board; a bare silicon chip, that is usually an integrated circuit, is supplied without a package (usually a lead frame overmolded with epoxy) and is attached, often with epoxy, directly to a circuit board. The chip is then wire bonded and protected from mechanical damage and contamination by an epoxy "glob-top".
        - COF - chip-on-flex; a variation of COB, where a chip is mounted directly to a flex circuit.
        - COG - chip-on-glass; a variation of COB, where a chip is mounted directly to a piece of glass - typically an LCD.

There are often subtle variations in package details from manufacturer to manufacturer, and even though standard designations are used, designers need to confirm dimensions when laying out printed circuit boards.

Distributed element model

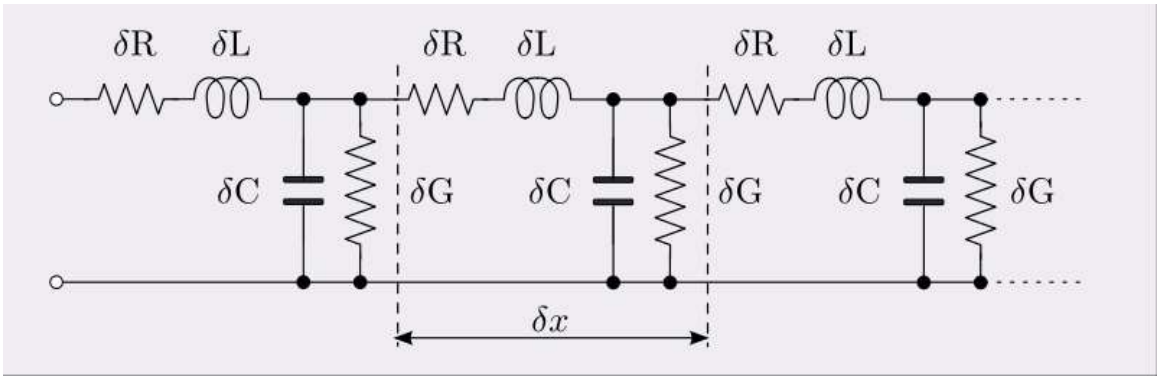


Fig.1 Transmission line. The distributed element model applied to a transmission line.

In electrical engineering, the **distributed element model** or **transmission line model** of electrical circuits assumes that the attributes of the circuit (resistance, capacitance, and inductance) are distributed continuously throughout the material of the circuit. This is in contrast to the more common lumped element model, which assumes that these values are lumped into electrical components that are joined by perfectly conducting wires. In the distributed element model, each circuit element is infinitesimally small, and the wires connecting elements are not assumed to be perfect conductors; that is, they have impedance. Unlike the lumped element model, it assumes non-uniform current along each branch and non-uniform voltage along each node. The distributed model is used at high frequencies where the wavelength approaches the physical dimensions of the circuit, making the lumped model inaccurate.

Applications

The distributed element model is more accurate but more complex than the lumped element model. The use of infinitesimals will often require the application of calculus whereas circuits analysed by the lumped element model can be solved with linear algebra. The distributed model is consequently only usually applied when accuracy calls for its use. Where this point is depends on the accuracy required in a specific application, but essentially, it needs to be used in circuits where the wavelengths of the signals have become comparable to the physical dimensions of the components. An often quoted engineering rule of thumb (not to be taken too literally because there are many exceptions) is that parts larger than one tenth of a wavelength will usually need to be analysed as distributed elements.

Transmission lines

Transmission lines are a common example of the use of the distributed model. Its use is dictated because the length of the line will usually be many wavelengths of the circuit's operating frequency. Even for the low frequencies used on power transmission lines, one

tenth of a wavelength is still only about 500 kilometres at 60 Hz. Transmission lines are usually represented in terms of the primary line constants as shown in figure 1. From this model the behaviour of the circuit is described by the secondary line constants which can be calculated from the primary ones.

The primary line constants are normally taken to be constant with position along the line leading to a particularly simple analysis and model. However, this is not always the case, variations in physical dimensions along the line will cause variations in the primary constants, that is, they have now to be described as functions of distance. Most often, such a situation represents an unwanted deviation from the ideal, such as a manufacturing error, however, there are a number of components where such longitudinal variations are deliberately introduced as part of the function of the component. A well known example of this is the horn antenna.

Where reflections are present on the line, quite short lengths of line can exhibit effects that are simply not predicted by the lumped element model. A quarter wavelength line, for instance, will transform the terminating impedance into its dual. This can be a wildly different impedance.

High frequency transistors

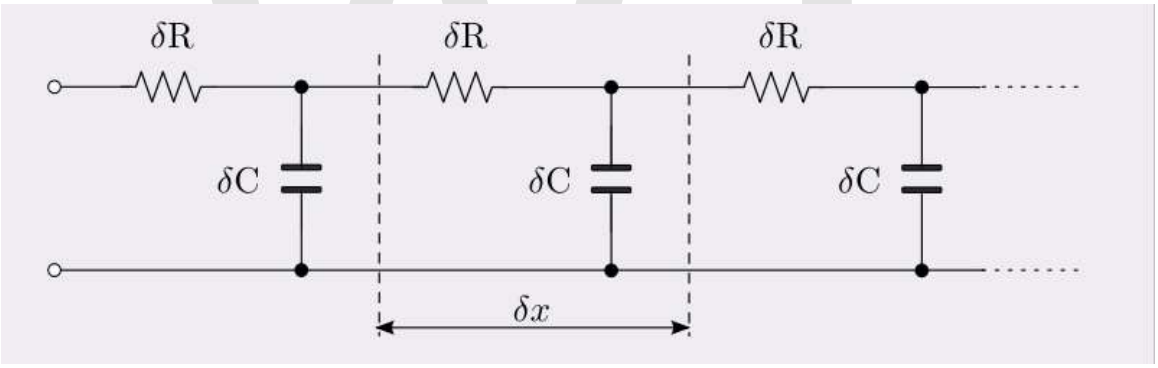


Fig.2. The base region of a bipolar junction transistor can be modelled as a simplified transmission line.

Another example of the use of distributed elements is in the modelling of the base region of a bipolar junction transistor at high frequencies. The analysis of charge carriers crossing the base region is not accurate when the base region is simply treated as a lumped element. A more successful model is a simplified transmission line model which includes distributed bulk resistance of the base material and distributed capacitance to the substrate. This model is represented in figure 2.

Resistivity measurements

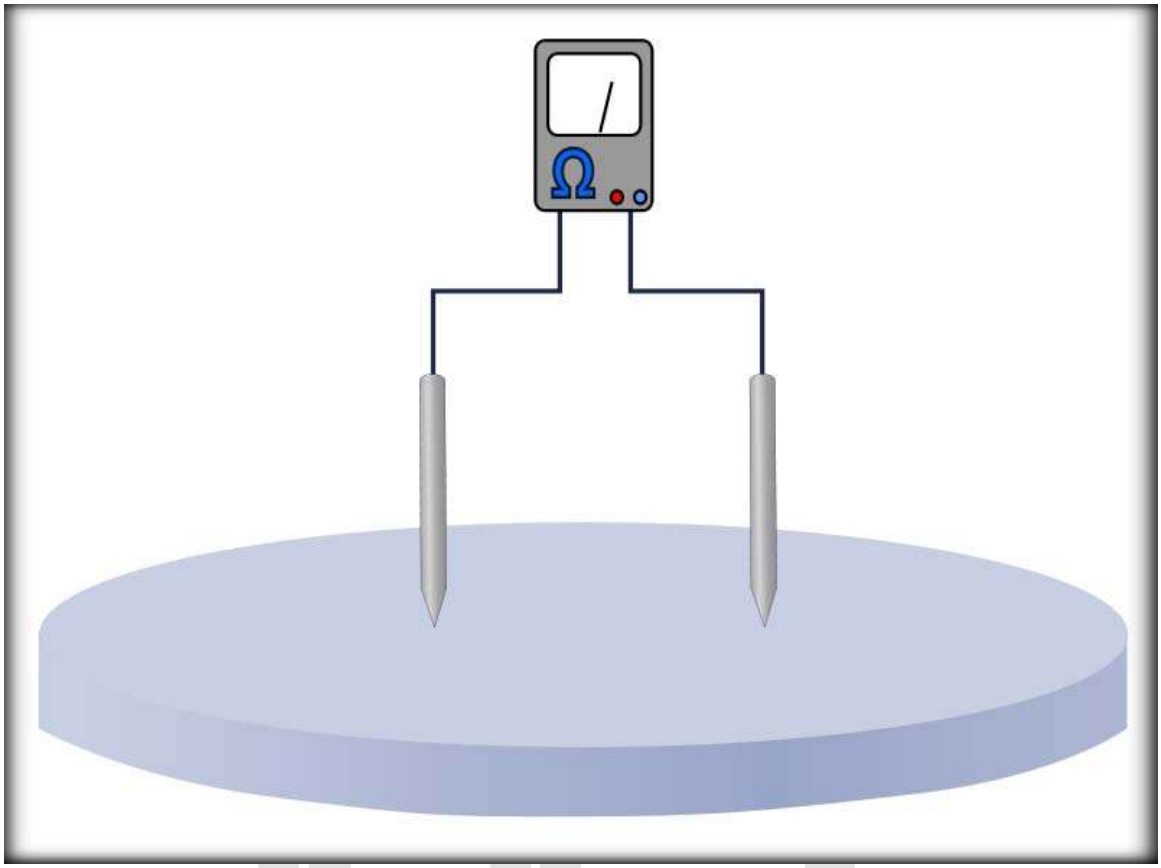


Fig. 3. Simplified arrangement for measuring resistivity of a bulk material with surface probes.

In many situations it is desired to measure resistivity of a bulk material by applying an electrode array at the surface. Amongst the fields that use this technique are geophysics (because it avoids having to dig into the substrate) and the semiconductor industry also use it (for the similar reason that it is non-intrusive) for testing bulk silicon wafers. The basic arrangement is shown in figure 3, although normally more electrodes would be used. To form a relationship between the voltage and current measured on the one hand, and the resistivity of the material on the other, it is necessary to apply the distributed element model by considering the material to be an array of infinitesimal resistor elements. Unlike the transmission line example, the need to apply the distributed element model arises from the geometry of the setup, and not from any wave propagation considerations.

The model used here needs to be truly 3-dimensional (transmission line models are usually described by elements of a one-dimensional line). It is also possible that the resistances of the elements will be functions of the co-ordinates, indeed, in the geophysical application it may well be that regions of changed resistivity are the very things that it is desired to detect.

Inductor windings

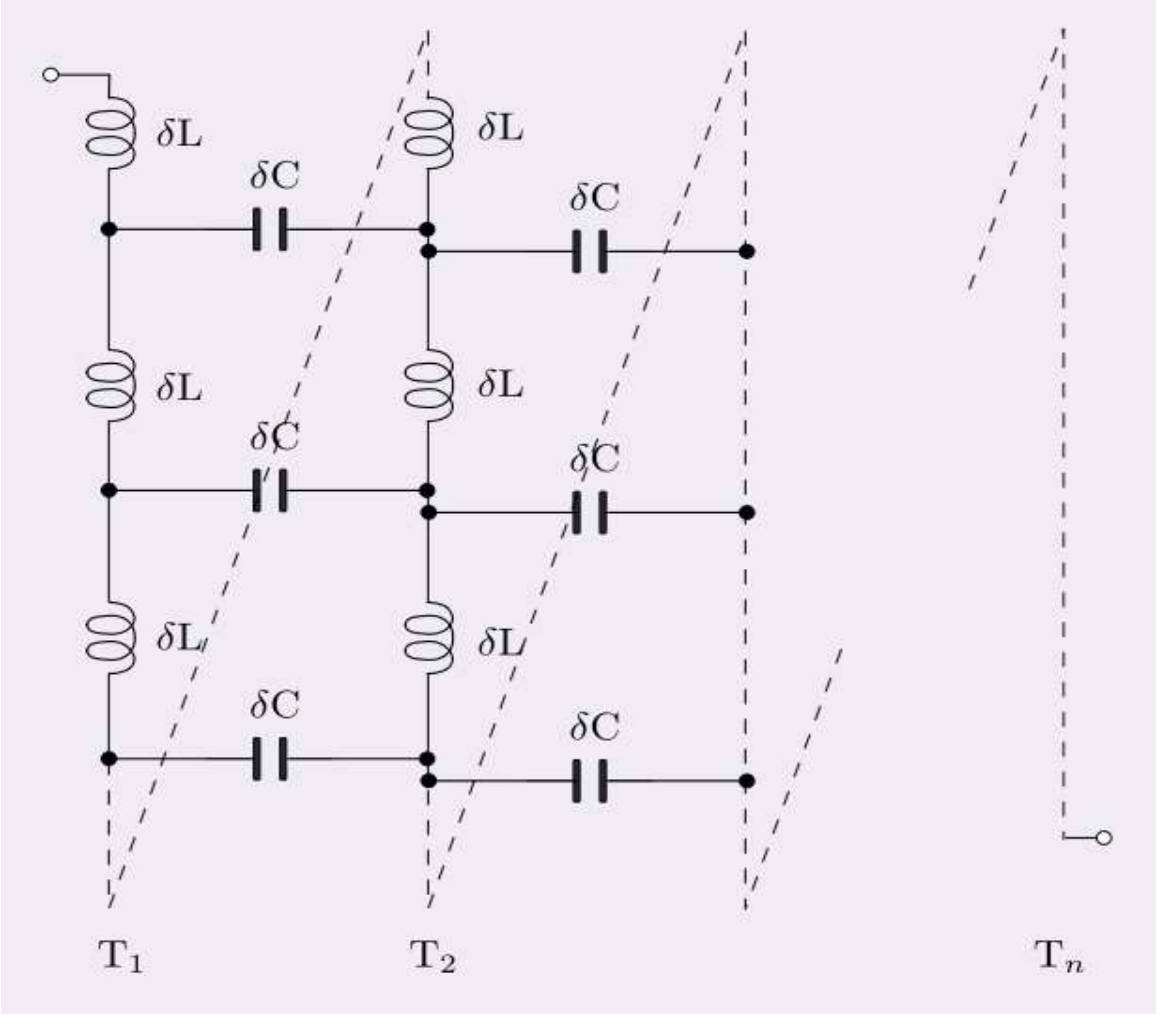
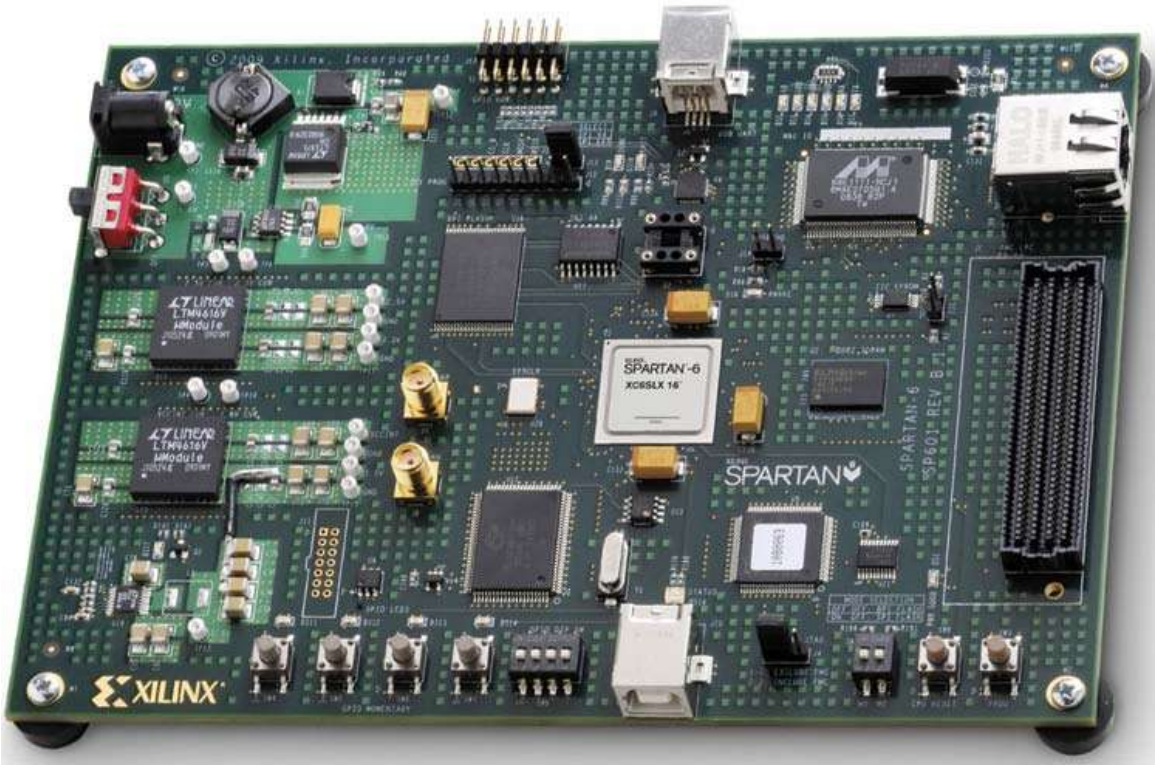


Fig. 4. A possible distributed element model of an inductor. A more accurate model will also require series resistance elements with the inductance elements.

Another example where a simple one-dimensional model will not suffice is the windings of an inductor. Coils of wire have capacitance between adjacent turns (and also more remote turns as well, but the effect progressively diminishes). For a single layer solenoid, the distributed capacitance will mostly lie between adjacent turns as shown in figure 4 between turns T_1 and T_2, but for multiple layer windings and more accurate models distributed capacitance to other turns must also be considered. This model is fairly difficult to deal with in simple calculations and for the most part is avoided. The most common approach is to roll up all the distributed capacitance into one lumped element in parallel with the inductance and resistance of the coil. This lumped model works successfully at low frequencies but falls apart at high frequencies where the usual practice is to simply measure (or specify) an overall *Q* for the inductor without associating a specific equivalent circuit.

Chapter 8

Field-Programmable Gate Array



An example of a Xilinx Spartan 6 FPGA programming/evaluation board

An Altera Stratix IV GX FPGA

A **Field-programmable Gate Array** (**FPGA**) is an integrated circuit designed to be configured by the customer or designer after manufacturing—hence "field-programmable". The FPGA configuration is generally specified using a hardware description language (HDL), similar to that used for an application-specific integrated circuit (ASIC) (circuit diagrams were previously used to specify the configuration, as they were for ASICs, but this is increasingly rare). FPGAs can be used to implement any logical function that an ASIC could perform. The ability to update the functionality after shipping, partial re-configuration of the portion of the design and the low non-recurring engineering costs relative to an ASIC design (notwithstanding the generally higher unit cost), offer advantages for many applications.

FPGAs contain programmable logic components called "logic blocks", and a hierarchy of reconfigurable interconnects that allow the blocks to be "wired together"—somewhat like many (changeable) logic gates that can be inter-wired in (many) different configurations. Logic blocks can be configured to perform complex combinational functions, or merely simple logic gates like AND and XOR. In most FPGAs, the logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memory.

In addition to digital functions, some FPGAs have analog features. The most common analog feature is programmable slew rate and drive strength on each output pin, allowing the engineer to set slow rates on lightly loaded pins that would otherwise ring unacceptably, and to set stronger, faster rates on heavily loaded pins on high-speed channels that would otherwise run too slow. Another relatively common analog feature is differential comparators on input pins designed to be connected to differential signaling channels. A few "mixed signal FPGAs" have integrated peripheral Analog-to-Digital

Converters (ADCs) and Digital-to-Analog Converters (DACs) with analog signal conditioning blocks allowing them to operate as a system-on-a-chip. Such devices blur the line between an FPGA, which carries digital ones and zeros on its internal programmable interconnect fabric, and field-programmable analog array (FPAA), which carries analog values on its internal programmable interconnect fabric.

History

The FPGA industry sprouted from programmable read-only memory (PROM) and programmable logic devices (PLDs). PROMs and PLDs both had the option of being programmed in batches in a factory or in the field (field programmable), however programmable logic was hard-wired between logic gates.

In the late 1980s the Naval Surface Warfare Department funded an experiment proposed by Steve Casselman to develop a computer that would implement 600,000 reprogrammable gates. Casselman was successful and a patent related to the system was issued in 1992.

Some of the industry's foundational concepts and technologies for programmable logic arrays, gates, and logic blocks are founded in patents awarded to David W. Page and LuVerne R. Peterson in 1985.

Xilinx Co-Founders, Ross Freeman and Bernard Vonderschmitt, invented the first commercially viable field programmable gate array in 1985 – the XC2064. The XC2064 had programmable gates and programmable interconnects between gates, the beginnings of a new technology and market. The XC2064 boasted a mere 64 configurable logic blocks (CLBs), with two 3-input lookup tables (LUTs). More than 20 years later, Freeman was entered into the National Inventors Hall of Fame for his invention.

Xilinx continued unchallenged and quickly growing from 1985 to the mid-1990s, when competitors sprouted up, eroding significant market-share. By 1993, Actel was serving about 18 percent of the market.

The 1990s were an explosive period of time for FPGAs, both in sophistication and the volume of production. In the early 1990s, FPGAs were primarily used in telecommunications and networking. By the end of the decade, FPGAs found their way into consumer, automotive, and industrial applications.

FPGAs got a glimpse of fame in 1997, when Adrian Thompson, a researcher working at the University of Sussex, merged genetic algorithm technology and FPGAs to create a sound recognition device. Thomson's algorithm configured an array of 10 x 10 cells in a Xilinx FPGA chip to discriminate between two tones, utilising analogue features of the digital chip. The application of genetic algorithms to the configuration of devices like FPGA's is now referred to as Evolvable hardware

Modern developments

A recent trend has been to take the coarse-grained architectural approach a step further by combining the logic blocks and interconnects of traditional FPGAs with embedded microprocessors and related peripherals to form a complete "system on a programmable chip". This work mirrors the architecture by Ron Perlof and Hana Potash of Burroughs Advanced Systems Group which combined a reconfigurable CPU architecture on a single chip called the SB24. That work was done in 1982. Examples of such hybrid technologies can be found in the Xilinx Virtex-II PRO and Virtex-4 devices, which include one or more PowerPC processors embedded within the FPGA's logic fabric. The Atmel FPSLIC is another such device, which uses an AVR processor in combination with Atmel's programmable logic architecture. The Actel SmartFusion devices incorporate an ARM architecture Cortex-M3 hard processor core (with up to 512kB of flash and 64kB of RAM) and analog peripherals such as a multi-channel ADC and DACs to their flash-based FPGA fabric.

An alternate approach to using hard-macro processors is to make use of soft processor cores that are implemented within the FPGA logic.

As previously mentioned, many modern FPGAs have the ability to be reprogrammed at "run time," and this is leading to the idea of reconfigurable computing or reconfigurable systems — CPUs that reconfigure themselves to suit the task at hand. The Mitrion Virtual Processor from Mitrionics is an example of a reconfigurable soft processor, implemented on FPGAs. However, it does not support dynamic reconfiguration at runtime, but instead adapts itself to a specific program.

Additionally, new, non-FPGA architectures are beginning to emerge. Software-configurable microprocessors such as the Stretch S5000 adopt a hybrid approach by providing an array of processor cores and FPGA-like programmable cores on the same chip.

Gates

- 1987: 9,000 gates, Xilinx
- 1992: 600,000, Naval Surface Warfare Department
- Early 2000s: Millions

Market size

- 1985: First commercial FPGA technology invented by Xilinx
- 1987: $14 million
- ~1993: >$385 million
- 2005: $1.9 billion
- 2010 estimates: $2.75 billion

FPGA design starts

- 10,000
- 2005: 80,000
- 2008: 90,000

FPGA comparisons

Historically, FPGAs have been slower, less energy efficient and generally achieved less functionality than their fixed ASIC counterparts. A study has shown that designs implemented on FPGAs need on average 18 times as much area, draw 7 times as much dynamic power, and are 3 times slower than the corresponding ASIC implementations.

An Altera Cyclone II FPGA, on an Altera teraSIC DE1 Prototyping board

Advantages include the ability to re-program in the field to fix bugs, and may include a shorter time to market and lower non-recurring engineering costs. Vendors can also take a middle road by developing their hardware on ordinary FPGAs, but manufacture their final version so it can no longer be modified after the design has been committed.

Xilinx claims that several market and technology dynamics are changing the ASIC/FPGA paradigm:

- Integrated circuit costs are rising aggressively
- ASIC complexity has lengthened development time
- R&D resources and headcount are decreasing
- Revenue losses for slow time-to-market are increasing
- Financial constraints in a poor economy are driving low-cost technologies

These trends make FPGAs a better alternative than ASICs for a larger number of higher-volume applications than they have been historically used for, to which the company attributes the growing number of FPGA design starts.

Some FPGAs have the capability of partial re-configuration that lets one portion of the device be re-programmed while other portions continue running.

Versus complex programmable logic devices

The primary differences between CPLDs (Complex Programmable Logic Devices) and FPGAs are architectural. A CPLD has a somewhat restrictive structure consisting of one or more programmable sum-of-products logic arrays feeding a relatively small number of clocked registers. The result of this is less flexibility, with the advantage of more predictable timing delays and a higher logic-to-interconnect ratio. The FPGA architectures, on the other hand, are dominated by interconnect. This makes them far more flexible (in terms of the range of designs that are practical for implementation within them) but also far more complex to design for.

Another notable difference between CPLDs and FPGAs is the presence in most FPGAs of higher-level embedded functions (such as adders and multipliers) and embedded memories, as well as to have logic blocks implement decoders or mathematical functions.

Security considerations

With respect to security, FPGAs have both advantages and disadvantages as compared to ASICs or secure microprocessors. FPGAs' flexibility makes malicious modifications during fabrication a lower risk. For many FPGAs, the loaded design is exposed while it is loaded (typically on every power-on). To address this issue, some FPGAs support bitstream encryption.

Applications

Applications of FPGAs include digital signal processing, software-defined radio, aerospace and defense systems, ASIC prototyping, medical imaging, computer vision, speech recognition, cryptography, bioinformatics, computer hardware emulation, radio astronomy, metal detection and a growing range of other areas.

FPGAs originally began as competitors to CPLDs and competed in a similar space, that of glue logic for PCBs. As their size, capabilities, and speed increased, they began to take over larger and larger functions to the state where some are now marketed as full systems on chips (SoC). Particularly with the introduction of dedicated multipliers into FPGA architectures in the late 1990s, applications which had traditionally been the sole reserve of DSPs began to incorporate FPGAs instead.

FPGAs especially find applications in any area or algorithm that can make use of the massive parallelism offered by their architecture. One such area is code breaking, in particular brute-force attack, of cryptographic algorithms.

FPGAs are increasingly used in conventional high performance computing applications where computational kernels such as FFT or Convolution are performed on the FPGA instead of a microprocessor.

The inherent parallelism of the logic resources on an FPGA allows for considerable computational throughput even at a low MHz clock rates. The flexibility of the FPGA allows for even higher performance by trading off precision and range in the number format for an increased number of parallel arithmetic units. This has driven a new type of processing called reconfigurable computing, where time intensive tasks are offloaded from software to FPGAs.

The adoption of FPGAs in high performance computing is currently limited by the complexity of FPGA design compared to conventional software and the turn-around times of current design tools.

Traditionally, FPGAs have been reserved for specific vertical applications where the volume of production is small. For these low-volume applications, the premium that companies pay in hardware costs per unit for a programmable chip is more affordable than the development resources spent on creating an ASIC for a low-volume application. Today, new cost and performance dynamics have broadened the range of viable applications.

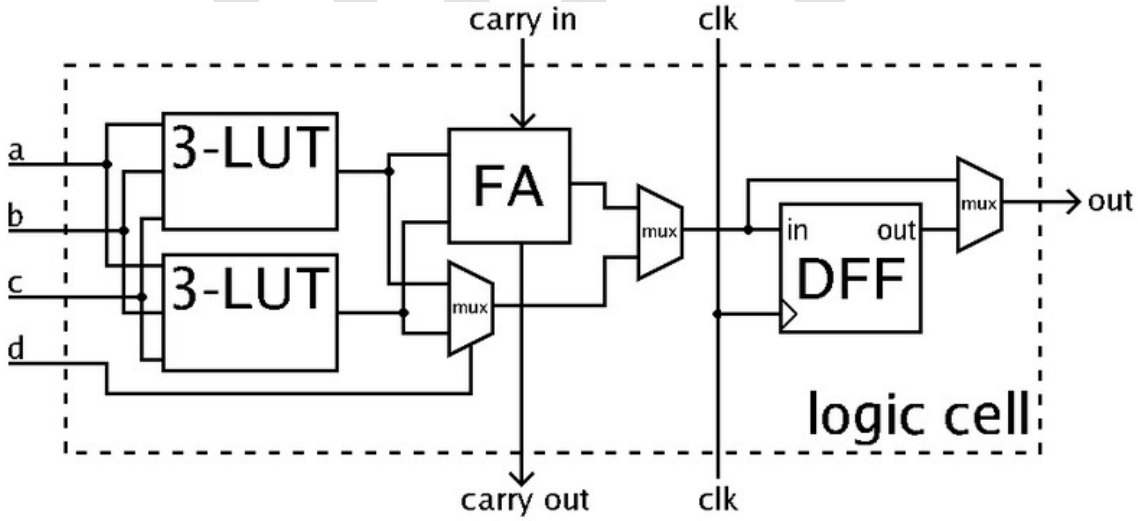
Architecture

The most common FPGA architecture consists of an array of logic blocks (called Configurable Logic Block, CLB, or Logic Array Block, LAB, depending on vendor), I/O pads, and routing channels. Generally, all the routing channels have the same width

(number of wires). Multiple I/O pads may fit into the height of one row or the width of one column in the array.

An application circuit must be mapped into an FPGA with adequate resources. While the number of CLBs/LABs and I/Os required is easily determined from the design, the number of routing tracks needed may vary considerably even among designs with the same amount of logic. For example, a crossbar switch requires much more routing than a systolic array with the same gate count. Since unused routing tracks increase the cost (and decrease the performance) of the part without providing any benefit, FPGA manufacturers try to provide just enough tracks so that most designs that will fit in terms of LUTs and IOs can be routed. This is determined by estimates such as those derived from Rent's rule or by experiments with existing designs.

In general, a logic block (CLB or LAB) consists of a few logical cells (called ALM, LE, Slice etc). A typical cell consists of a 4-input Lookup table (LUT), a Full adder (FA) and a D-type flip-flop, as shown below. The LUTs are in this figure split into two 3-input LUTs. In *normal mode* those are combined into a 4-input LUT through the left mux. In *arithmetic* mode, their outputs are fed to the FA. The selection of mode is programmed into the middle mux. The output can be either synchronous or asynchronous, depending on the programming of the mux to the right, in the figure example. In practice, entire or parts of the FA are put as functions into the LUTs in order to save space.



Simplified example illustration of a logic cell

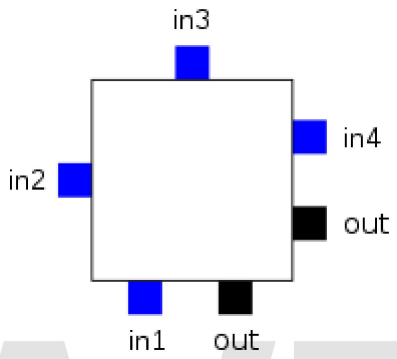
ALMs and Slices usually contains 2 or 4 structures similar to the example figure, with some shared signals.

CLBs/LABs typically contains a few ALMs/LEs/Slices.

In recent years, manufacturers have started moving to 6-input LUTs in their high performance parts, claiming increased performance.

Since clock signals (and often other high-fanout signals) are normally routed via special-purpose dedicated routing networks in commercial FPGAs, they and other signals are separately managed.

For this example architecture, the locations of the FPGA logic block pins are shown below.



Logic Block Pin Locations

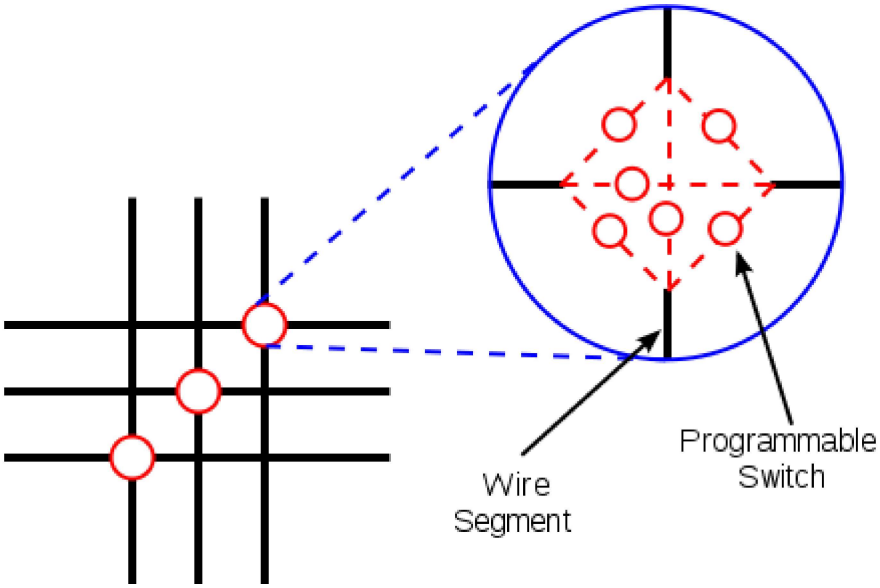
Each input is accessible from one side of the logic block, while the output pin can connect to routing wires in both the channel to the right and the channel below the logic block.

Each logic block output pin can connect to any of the wiring segments in the channels adjacent to it.

Similarly, an I/O pad can connect to any one of the wiring segments in the channel adjacent to it. For example, an I/O pad at the top of the chip can connect to any of the W wires (where W is the channel width) in the horizontal channel immediately below it.

Generally, the FPGA routing is unsegmented. That is, each wiring segment spans only one logic block before it terminates in a switch box. By turning on some of the programmable switches within a switch box, longer paths can be constructed. For higher speed interconnect, some FPGA architectures use longer routing lines that span multiple logic blocks.

Whenever a vertical and a horizontal channel intersect, there is a switch box. In this architecture, when a wire enters a switch box, there are three programmable switches that allow it to connect to three other wires in adjacent channel segments. The pattern, or topology, of switches used in this architecture is the planar or domain-based switch box topology. In this switch box topology, a wire in track number one connects only to wires in track number one in adjacent channel segments, wires in track number 2 connect only to other wires in track number 2 and so on. The figure below illustrates the connections in a switch box.



Switch box topology

Modern FPGA families expand upon the above capabilities to include higher level functionality fixed into the silicon. Having these common functions embedded into the silicon reduces the area required and gives those functions increased speed compared to building them from primitives. Examples of these include multipliers, generic DSP blocks, embedded processors, high speed IO logic and embedded memories.

FPGAs are also widely used for systems validation including pre-silicon validation, post-silicon validation, and firmware development. This allows chip companies to validate their design before the chip is produced in the factory, reducing the time-to-market.

FPGA design and programming

To define the behavior of the FPGA, the user provides a hardware description language (HDL) or a schematic design. The HDL form is more suited to work with large structures because it's possible to just specify them numerically rather than having to draw every piece by hand. However, schematic entry can allow for easier visualisation of a design.

Then, using an electronic design automation tool, a technology-mapped netlist is generated. The netlist can then be fitted to the actual FPGA architecture using a process called place-and-route, usually performed by the FPGA company's proprietary place-and-route software. The user will validate the map, place and route results via timing analysis, simulation, and other verification methodologies. Once the design and validation process is complete, the binary file generated (also using the FPGA company's proprietary software) is used to (re)configure the FPGA.

Going from schematic/HDL source files to actual configuration: The source files are fed to a software suite from the FPGA/CPLD vendor that through different steps will produce

a file. This file is then transferred to the FPGA/CPLD via a serial interface (JTAG) or to an external memory device like an EEPROM.

The most common HDLs are VHDL and Verilog, although in an attempt to reduce the complexity of designing in HDLs, which have been compared to the equivalent of assembly languages, there are moves to raise the abstraction level through the introduction of alternative languages. National Instrument's LabVIEW graphical programming language (sometimes referred to as "G") has an FPGA add-in module available to target and program FPGA hardware.

To simplify the design of complex systems in FPGAs, there exist libraries of predefined complex functions and circuits that have been tested and optimized to speed up the design process. These predefined circuits are commonly called *IP cores*, and are available from FPGA vendors and third-party IP suppliers (rarely free, and typically released under proprietary licenses). Other predefined circuits are available from developer communities such as OpenCores (typically released under free and open source licenses such as the GPL, BSD or similar license), and other sources.

In a typical design flow, an FPGA application developer will simulate the design at multiple stages throughout the design process. Initially the RTL description in VHDL or Verilog is simulated by creating test benches to simulate the system and observe results. Then, after the synthesis engine has mapped the design to a netlist, the netlist is translated to a gate level description where simulation is repeated to confirm the synthesis proceeded without errors. Finally the design is laid out in the FPGA at which point propagation delays can be added and the simulation run again with these values back-annotated onto the netlist.

Basic process technology types

- SRAM - based on static memory technology. In-system programmable and re-programmable. Requires external boot devices. CMOS.
- Antifuse - One-time programmable. CMOS.
- PROM - Programmable Read-Only Memory technology. One-time programmable because of plastic packaging.
- EPROM - Erasable Programmable Read-Only Memory technology. One-time programmable but with window, can be erased with ultraviolet (UV) light. CMOS.
- EEPROM - Electrically Erasable Programmable Read-Only Memory technology. Can be erased, even in plastic packages. Some but not all EEPROM devices can be in-system programmed. CMOS.
- Flash - Flash-erase EPROM technology. Can be erased, even in plastic packages. Some but not all flash devices can be in-system programmed. Usually, a flash cell is smaller than an equivalent EEPROM cell and is therefore less expensive to manufacture. CMOS.
- Fuse - One-time programmable. Bipolar.

Major manufacturers

Xilinx and Altera are the current FPGA market leaders and long-time industry rivals. Together, they control over 80 percent of the market, with Xilinx alone representing over 50 percent.

Both Xilinx and Altera provide free Windows and Linux design software.

Other competitors include Lattice Semiconductor (SRAM based with integrated configuration Flash, instant-on, low power, live reconfiguration), Actel (antifuse, flash-based, mixed-signal), SiliconBlue Technologies (extremely low power SRAM-based FPGAs with option integrated nonvolatile configuration memory), Achronix (RAM based, 1.5 GHz fabric speed) who will be building their chips on Intels' state-of-the art 22 nm process, and QuickLogic (handheld focused CSSP, no general purpose FPGAs).

In March 2010, Tabula announced their new FPGA technology that uses time-multiplexed logic and interconnect for greater potential cost savings for high-density applications.

Chapter 9

Application-Specific Integrated Circuit

An **application-specific integrated circuit** (ASIC) is an integrated circuit (IC) customized for a particular use, rather than intended for general-purpose use. For example, a chip designed solely to run a cell phone is an ASIC. Application-specific standard products (ASSPs) are intermediate between ASICs and industry standard integrated circuits like the 7400 or the 4000 series.

As feature sizes have shrunk and design tools improved over the years, the maximum complexity (and hence functionality) possible in an ASIC has grown from 5,000 gates to over 100 million. Modern ASICs often include entire 32-bit processors, memory blocks including ROM, RAM, EEPROM, Flash and other large building blocks. Such an ASIC is often termed a SoC (system-on-a-chip). Designers of digital ASICs use a hardware description language (HDL), such as Verilog or VHDL, to describe the functionality of ASICs.

Field-programmable gate arrays (FPGA) are the modern-day technology for building a breadboard or prototype from standard parts; programmable logic blocks and programmable interconnects allow the same FPGA to be used in many different applications. For smaller designs and/or lower production volumes, FPGAs may be more cost effective than an ASIC design even in production. The non-recurring engineering cost of an ASIC can run into the millions of dollars.

History

The initial ASICs used gate array technology. Ferranti produced perhaps the first gate-array, the ULA (Uncommitted Logic Array), around 1980. An early successful commercial application was the ULA circuitry found in the 8-bit ZX81 and ZX Spectrum low-end personal computers, introduced in 1981 and 1982. These were used by Sinclair Research (UK) essentially as a low-cost I/O solution aimed at handling the computer's graphics. Some versions of ZX81/Timex Sinclair 1000 used just four chips (ULA, 2Kx8 RAM, 8Kx8 ROM, Z80A CPU) to implement an entire mass-market personal computer with built-in BASIC interpreter.

Customization occurred by varying the metal interconnect mask. ULAs had complexities of up to a few thousand gates. Later versions became more generalized, with different

base dies customised by both metal and polysilicon layers. Some base dies include RAM elements.

Standard cell design

In the mid 1980s, a designer would choose an ASIC manufacturer and implement their design using the design tools available from the manufacturer. While third-party design tools were available, there was not an effective link from the third-party design tools to the layout and actual semiconductor process performance characteristics of the various ASIC manufacturers. Most designers ended up using factory-specific tools to complete the implementation of their designs. A solution to this problem, which also yielded a much higher density device, was the implementation of Standard Cells. Every ASIC manufacturer could create functional blocks with known electrical characteristics, such as propagation delay, capacitance and inductance, that could also be represented in third-party tools. Standard Cell design is the utilization of these functional blocks to achieve very high gate density and good electrical performance. Standard cell design fits between Gate Array and Full Custom design in terms of both its NRE (Non-Recurring Engineering) and recurring component cost.

By the late 1990s, logic synthesis tools became available. Such tools could compile HDL descriptions into a gate-level netlist. Standard-cell Integrated Circuits (ICs) are designed in the following conceptual stages, although these stages overlap significantly in practice.

1. A team of design engineers starts with a non-formal understanding of the required functions for a new ASIC, usually derived from Requirements analysis.
2. The design team constructs a description of an ASIC to achieve these goals using an HDL. This process is analogous to writing a computer program in a high-level language. This is usually called the RTL (Register transfer level) design.
3. Suitability for purpose is verified by functional verification. This may include such techniques as logic simulation, formal verification, emulation, or creating an equivalent pure software model. Each technique has advantages and disadvantages, and often several methods are used.
4. Logic synthesis transforms the RTL design into a large collection of lower-level constructs called standard cells. These constructs are taken from a standard-cell library consisting of pre-characterized collections of gates (such as 2 input nor, 2 input nand, inverters, etc.). The standard cells are typically specific to the planned manufacturer of the ASIC. The resulting collection of standard cells, plus the needed electrical connections between them, is called a gate-level netlist.
5. The gate-level netlist is next processed by a placement tool which places the standard cells onto a region representing the final ASIC. It attempts to find a placement of the standard cells, subject to a variety of specified constraints.
6. The routing tool takes the physical placement of the standard cells and uses the netlist to create the electrical connections between them. Since the search space is large, this process will produce a “sufficient” rather than “globally-optimal” solution. The output is a file which can be used to create a set of photomasks

enabling a semiconductor fabrication facility (commonly called a 'fab') to produce physical ICs.

7. Given the final layout, circuit extraction computes the parasitic resistances and capacitances. In the case of a digital circuit, this will then be further mapped into delay information, from which the circuit performance can be estimated, usually by static timing analysis. This, and other final tests such as design rule checking and power analysis (collectively called signoff) are intended to ensure that the device will function correctly over all extremes of the process, voltage and temperature. When this testing is complete the photomask information is released for chip fabrication.

These steps, implemented with a level of skill common in the industry, almost always produce a final device that correctly implements the original design, unless flaws are later introduced by the physical fabrication process.

The design steps (or flow) are also common to standard product design. The significant difference is that Standard Cell design uses the manufacturer's cell libraries that have been used in potentially hundreds of other design implementations and therefore are of much lower risk than full custom design. Standard Cells produce a design density that is cost effective, and they can also integrate IP cores and SRAM (Static Random Access Memory) effectively, unlike Gate Arrays.

Gate array design

Gate array design is a manufacturing method in which the diffused layers, i.e. transistors and other active devices, are predefined and wafers containing such devices are held in stock prior to metallization — in other words, unconnected. The physical design process then defines the interconnections of the final device. For most ASIC manufacturers, this consists of from two to as many as nine metal layers, each metal layer running perpendicular to the one below it. Non-recurring engineering costs are much lower, as photolithographic masks are required only for the metal layers, and production cycles are much shorter, as metallization is a comparatively quick process.

Gate array ASICs are always a compromise as mapping a given design onto what a manufacturer held as a stock wafer never gives 100% utilization. Often difficulties in routing the interconnect require migration onto a larger array device with consequent increase in the piece part price. These difficulties are often a result of the layout software used to develop the interconnect.

Pure, logic-only gate array design is rarely implemented by circuit designers today, having been replaced almost entirely by field-programmable devices, such as field-programmable gate arrays (FPGAs), which can be programmed by the user and thus offer minimal tooling charges (non-recurring engineering (NRE)), only marginally increased piece part cost, and comparable performance. Today, gate arrays are evolving into structured ASICs that consist of a large IP core like a CPU, DSP unit, peripherals, standard interfaces, integrated memories SRAM, and a block of reconfigurable,

uncommited logic. This shift is largely because ASIC devices are capable of integrating such large blocks of system functionality and "system-on-a-chip" requires far more than just logic blocks.

In their frequent usages in the field, the terms "gate array" and "semi-custom" are synonymous. Process engineers more commonly use the term "semi-custom", while "gate-array" is more commonly used by logic (or gate-level) designers.

Full-custom design

By contrast, full-custom ASIC design defines all the photolithographic layers of the device. Full-custom design is used for both ASIC design and for standard product design.

The benefits of full-custom design usually include reduced area (and therefore recurring component cost), performance improvements, and also the ability to integrate analog components and other pre-designed — and thus fully verified — components, such as microprocessor cores that form a system-on-chip.

The disadvantages of full-custom design can include increased manufacturing and design time, increased non-recurring engineering costs, more complexity in the computer-aided design (CAD) system, and a much higher skill requirement on the part of the design team.

For digital-only designs, however, "standard-cell" cell libraries, together with modern CAD systems, can offer considerable performance/cost benefits with low risk. Automated layout tools are quick and easy to use and also offer the possibility to "hand-tweak" or manually optimize any performance-limiting aspect of the design.

Structured design

Structured ASIC design (also referred to as "platform ASIC design"), is a relatively new term in the industry, resulting in some variation in its definition. However, the basic premise of a structured ASIC is that both manufacturing cycle time and design cycle time are reduced compared to cell-based ASIC, by virtue of there being pre-defined metal layers (thus reducing manufacturing time) and pre-characterization of what is on the silicon (thus reducing design cycle time). One definition states that

> In a "structured ASIC" design, the logic mask-layers of a device are predefined by the ASIC vendor (or in some cases by a third party). Design differentiation and customization is achieved by creating custom metal layers that create custom connections between predefined lower-layer logic elements. "Structured ASIC" technology is seen as bridging the gap between field-programmable gate arrays and "standard-cell" ASIC designs. Because only a small number of chip layers must be custom-produced, "structured ASIC" designs have much smaller non-

recurring expenditures (NRE) than "standard-cell" or "full-custom" chips, which require that a full mask set be produced for every design.

This is effectively the same definition as a gate array. What makes a structured ASIC different is that in a gate array, the predefined metal layers serve to make manufacturing turnaround faster. In a structured ASIC, the use of predefined metallization is primarily to reduce cost of the mask sets as well as making the design cycle time significantly shorter. For example, in a cell-based or gate-array design the user must often design power, clock, and test structures themselves; these are predefined in most structured ASICs and therefore can save time and expense for the designer compared to gate-array. Likewise, the design tools used for structured ASIC can be substantially lower cost and easier (faster) to use than cell-based tools, because they do not have to perform all the functions that cell-based tools do. In some cases, the structured ASIC vendor requires that customized tools for their device (e.g., custom physical synthesis) be used, also allowing for the design to be brought into manufacturing more quickly.

One other important aspect about structured ASIC is that it allows intellectual property (IP) that is common to certain applications or industry segments to be "built in", rather than "designed in". By building the IP directly into the architecture the designer can again save both time and money compared to designing IP into a cell-based ASIC.

Cell libraries, IP-based design, hard and soft macros

Cell libraries of logical primitives are usually provided by the device manufacturer as part of the service. Although they will incur no additional cost, their release will be covered by the terms of a non-disclosure agreement (NDA) and they will be regarded as intellectual property by the manufacturer. Usually their physical design will be pre-defined so they could be termed "hard macros".

What most engineers understand as "intellectual property" are IP cores, designs purchased from a third-party as sub-components of a larger ASIC. They may be provided as an HDL description (often termed a "soft macro"), or as a fully routed design that could be printed directly onto an ASIC's mask (often termed a hard macro). Many organizations now sell such pre-designed cores — CPUs, Ethernet, USB or telephone interfaces — and larger organizations may have an entire department or division to produce cores for the rest of the organization. Indeed, the wide range of functions now available is a significant factor in the phenomenal increase in electronics in the late 1990s and early 2000s; as a core takes a lot of time and investment to create, its re-use and further development cuts product cycle times dramatically and creates better products. Additionally, organizations such as OpenCores are collecting free IP cores paralleling the open source movement in software.

Soft macros are often process-independent, i.e., they can be fabricated on a wide range of manufacturing processes and different manufacturers. Hard macros are process-limited and usually further design effort must be invested to migrate (port) to a different process or manufacturer.

Multi-project wafers

Some manufacturers offer Multi-Project Wafers (MPW) as a method of obtaining low cost prototypes. Often called shuttles, these MPW, containing several designs, run at regular, scheduled intervals on a "cut and go" basis, usually with very little liability on the part of the manufacturer. The contract involves the assembly and packaging of a handful of devices. The service usually involves the supply of a physical design data base i.e. masking information or Pattern Generation (PG) tape. The manufacturer is often referred to as a "silicon foundry" due to the low involvement it has in the process. .

ASIC suppliers

There are two different types of ASIC suppliers, IDM and fabless. An IDM supplier's ASIC product is based in large part on proprietary technology such as design tools, IP, packaging, and usually although not necessarily the process technology. Fabless ASIC suppliers rely almost exclusively on outside suppliers for their technology. The classification can be confusing since several IDM's are also fabless semiconductor companies.

IDM ASIC suppliers

- Avago Technologies
- Elmos Semiconductor
- Cavium Networks
- Fujitsu
- Freescale
- HITACHI
- IBM
- Infineon Technologies
- LSI Corporation
- Marvell Semiconductor
- NEC
- NXP Semiconductors
- ON Semiconductor
- Renesas
- Samsung
- STMicroelectronics
- Texas Instruments
- Toshiba
- TSMC

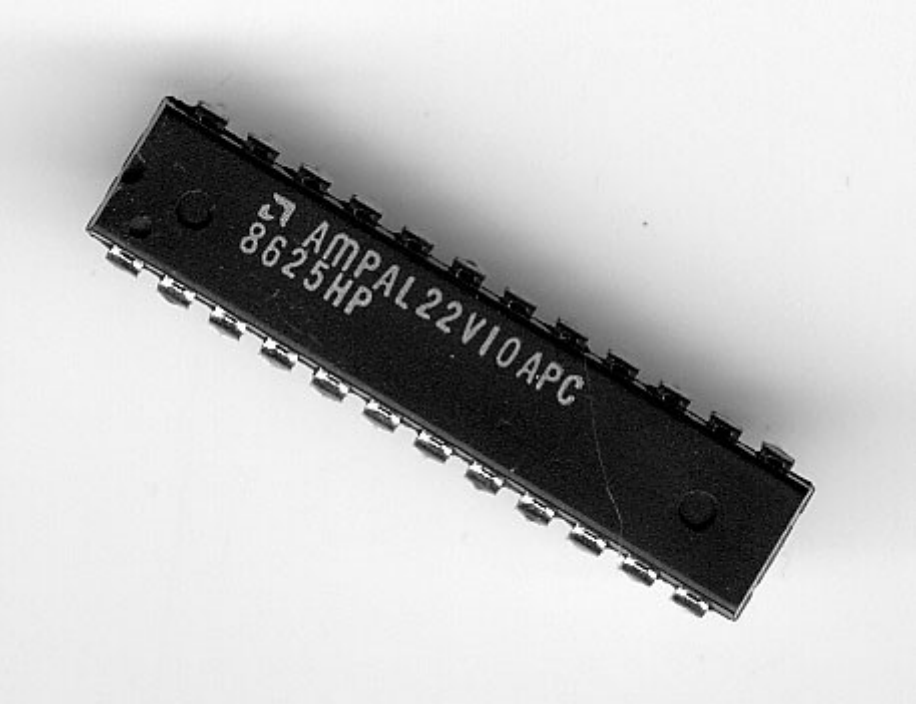
Fabless ASIC suppliers

- Alchip
- Aeroflex Colorado Springs

- Brite Semiconductor
- Custom Silicon Solutions
- eASIC
- eSilicon
- Faraday Technology
- Hong Kong Science and Technology Parks Corporation
- JVD Inc.
- Marvell Semiconductor
- MOSIS
- Nvidia
- Open-Silicon
- PMC Sierra
- Qualcomm
- Socle
- System to ASIC

Chapter 10

Programmable Array Logic



AMD 22V10 in 24-pin DIP

MMI PAL 16R6 in 20-pin DIP

The term **Programmable Array Logic** (PAL) is used to describe a family of programmable logic device semiconductors used to implement logic functions in digital circuits introduced by Monolithic Memories, Inc. (MMI) in March 1978. MMI obtained a registered trademark on the term PAL for use in "Programmable Semiconductor Logic Circuits". The trademark is currently held by Lattice Semiconductor.

PAL devices consisted of a small PROM (programmable read-only memory) core and additional output logic used to implement particular desired logic functions with few components.

Using specialized machines, PAL devices were "field-programmable". Each PAL device was "one-time programmable" (OTP), meaning that it could not be updated and reused after its initial programming. (MMI also offered a similar family called HAL, or "hard array logic", which were like PAL devices except that they were mask-programmed at the factory.)

Early history

Before PALs were introduced, designers of digital logic circuits would use small-scale integration (SSI) components, such as those in the 7400 series TTL (transistor-transistor logic) family; the 7400 family included a variety of logic building blocks, such as gates (NOT, NAND, NOR, AND, OR), multiplexers (MUXes) and demultiplexers (DEMUXes), flip flops (D-type, JK, etc.) and others. One PAL device would typically replace dozens of such "discrete" logic packages, so the SSI business went into decline as the PAL business took off. PALs were used advantageously in many products, such as minicomputers, as documented in Tracy Kidder's best-selling book "The Soul of a New Machine."

PALs were not the first commercial programmable logic devices; Signetics had been selling its field programmable logic array (FPLA) since 1975. These devices were completely unfamiliar to most circuit designers and were perceived to be too difficult to use. The FPLA had a relatively slow maximum operating speed (due to having both programmable-AND and programmable-OR arrays), was expensive, and had a poor reputation for testability. Another factor limiting the acceptance of the FPLA was the large package, a 600-mil (0.6", or 15.24 mm) wide 28-pin dual in-line package (DIP).

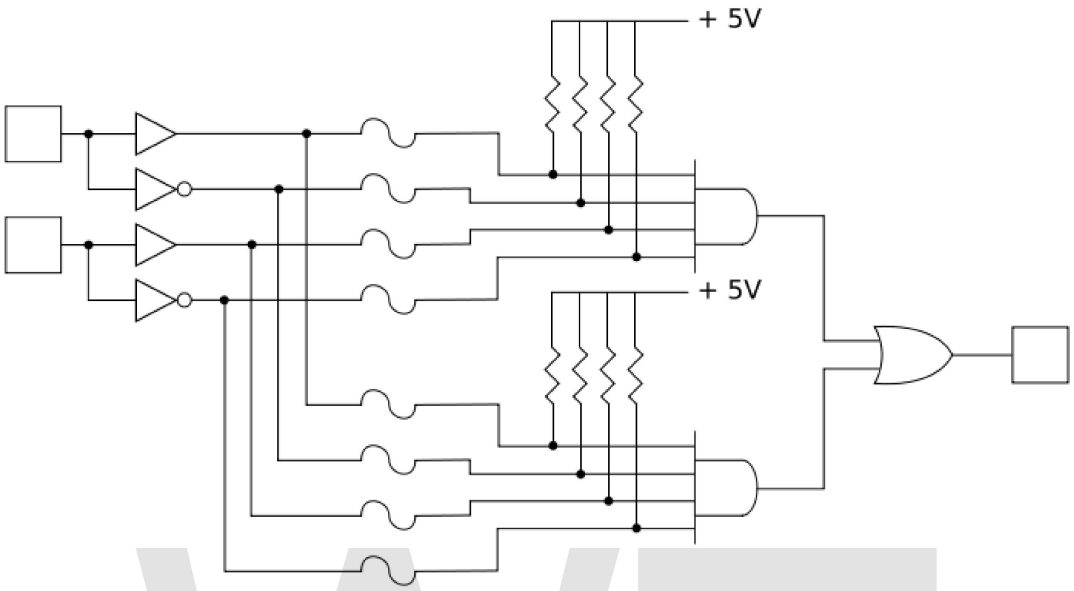
The project to create the PAL device was managed by John Birkner and the actual PAL circuit was designed by H. T. Chua. In a previous job, Birkner had developed a 16-bit processor using 80 standard logic devices. His experience with standard logic led him to believe that user programmable devices would be more attractive to users if the devices were designed to replace standard logic. This meant that the package sizes had to be more typical of the existing devices, and the speeds had to be improved. The PAL met these requirements and was a huge success and were "second sourced" by National Semiconductor, Texas Instruments, and Advanced Micro Devices.

Process technologies

Early PALs were 20-pin DIP components fabricated in silicon using bipolar transistor technology with one-time programmable (OTP) titanium-tungsten programming fuses. Later devices were manufactured by Lattice Semiconductor and Advanced Micro Devices using CMOS technology.

The original 20 and 24-pin PALs were described by MMI as medium-scale integration (MSI) devices.

PAL architecture



Simplified programmable logic device

The programmable elements (shown as a fuse) connect both the true and complemented inputs to the AND gates. These AND gates, also known as *product terms*, are ORed together to form a *sum-of-products* logic array.

The PAL architecture consists of two main components: a logic plane and output logic macrocells.

Programmable logic plane

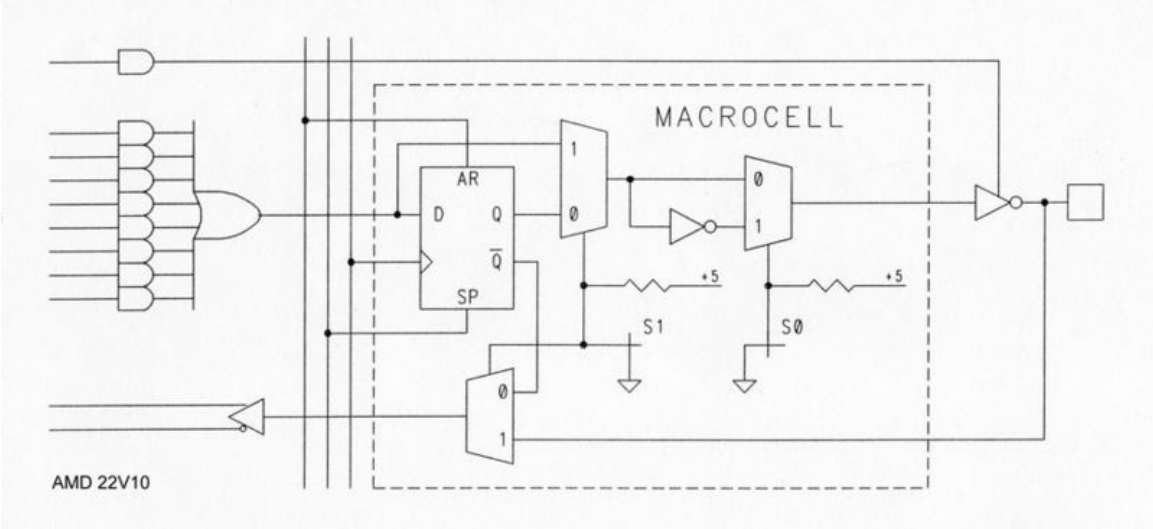
The programmable logic plane is a programmable read-only memory (PROM) array that allows the signals present on the devices pins (or the logical complements of those signals) to be routed to an output logic macrocell.

PAL devices have arrays of transistor cells arranged in a "fixed-OR, programmable-AND" plane used to implement "sum-of-products" binary logic equations for each of the outputs in terms of the inputs and either synchronous or asynchronous feedback from the outputs.

Output logic

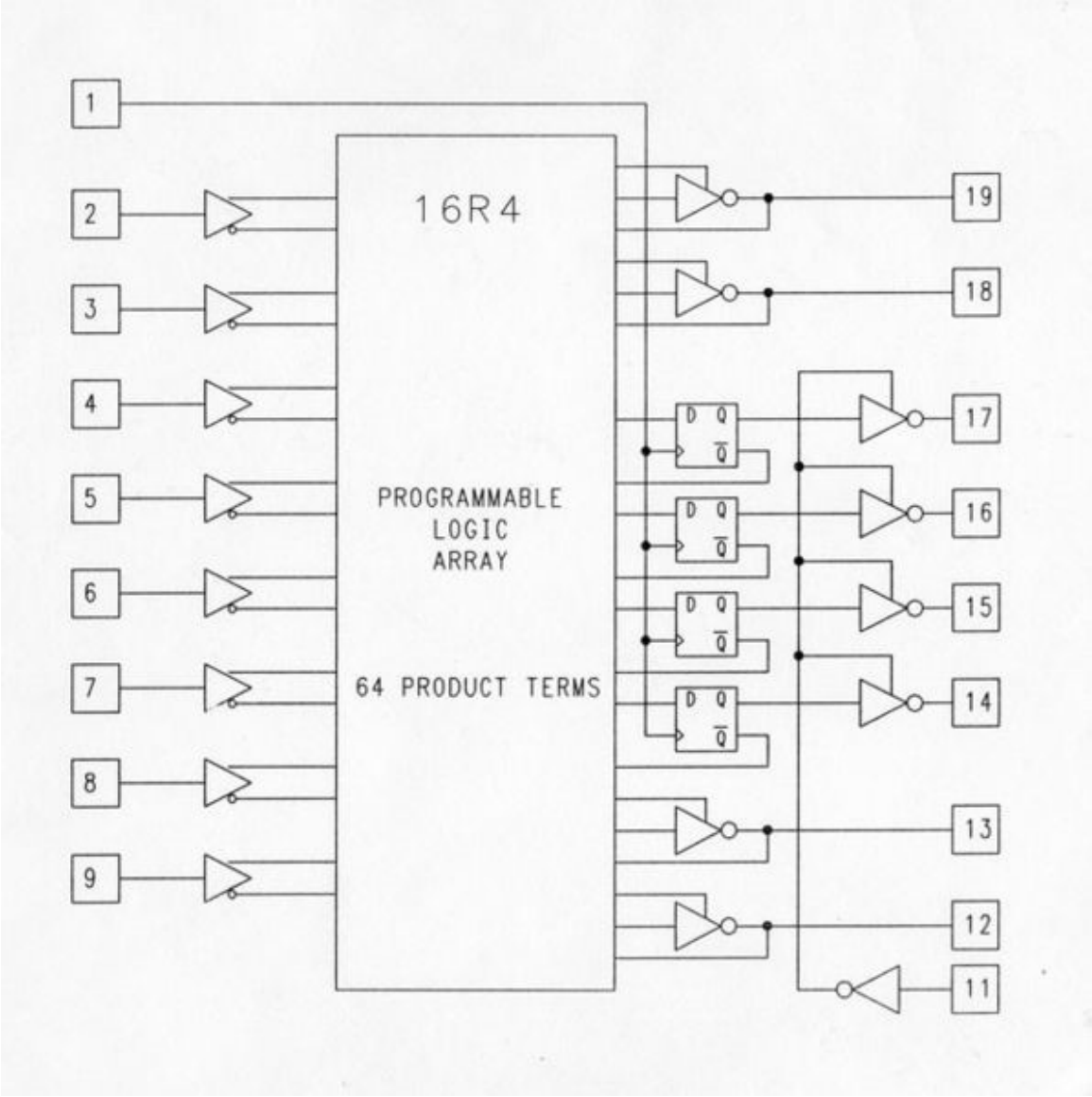
The early 20-pin PALs had 10 inputs and 8 outputs. The outputs were active low and could be registered or combinational. Members of the PAL family were available with various output structures called "output logic macrocells" or OLMCs. Prior to the introduction of the "V" (for "variable") series, the types of OLMCs available in each PAL

were fixed at the time of manufacture. (The PAL16L8 had 8 combinational outputs and the PAL16R8 had 8 registered outputs. The PAL16R6 had 6 registered and 2 combinational while the PAL16R4 had 4 of each.) Each output could have up to 8 product terms (effectively AND gates), however the combinational outputs used one of the terms to control a bidirectional output buffer. There were other combinations that had fewer outputs with more product term per output and were available with active high outputs. The 16X8 family or registered devices had an XOR gate before the register. There were also similar 24-pin versions of these PALs.

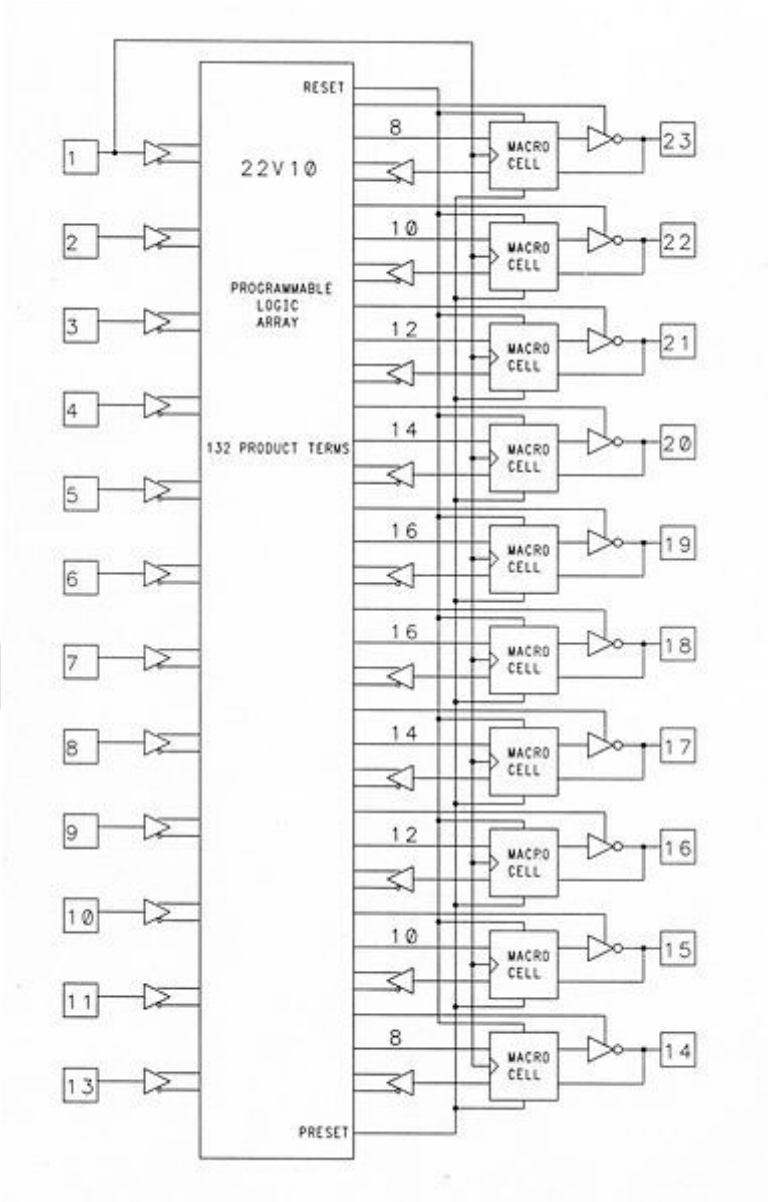


AMD 22V10 Output Macrocell

This fixed output structure often frustrated designers attempting to optimize the utility of PAL devices because output structures of different types were often required by their applications. (For example, one could not get 5 registered outputs with 3 active high combinational outputs.) So, in June 1983 AMD introduced the 22V10, a 24 pin device with 10 output logic macrocells. Each macrocell could be configured by the user to be combinational or registered, active high or active low. The number of product term allocated to an output varied from 8 to 16. This one device could replace all of the 24 pin fixed function PAL devices. Members of the PAL "V" ("variable") series included the PAL16V8, PAL20V8 and PAL22V10.



PAL 16R4 Block Diagram



AMD 22V10 Block Diagram

Programming PALs

PALs were programmed electrically using binary patterns (as JEDEC ASCII/hexadecimal files) and a special electronic programming system available from either the manufacturer or a third-party, such as DATA/IO. In addition to single-unit device programmers, device feeders and gang programmers were often used when more than just a few PALs needed to be programmed. (For large volumes, electrical programming costs could be eliminated by having the manufacturer fabricate a custom metal mask used to program the customers' patterns at the time of manufacture; MMI used the term "hard array logic" (HAL) to refer to devices programmed in this way.)

Programming languages

```
PAL16R4 PAL              PAL DESIGN SPECIFICATION
CNT4SC
4 bit counter with synchronous clear
Michael Holley and Dave Pellerin
Clk  Clear  NC  NC  NC  NC  NC  NC  NC  GND
 OE  NC     NC /Q3 /Q2 /Q1 /Q0  NC  NC  VCC

  Q3 :=  Clear
       + /Q3 * /Q2 * /Q1 * /Q0
       +  Q3 *  Q0
       +  Q3 *  Q1
       +  Q3 *  Q2

  Q2 :=  Clear
       + /Q2 * /Q1 * /Q0
       +  Q2 *  Q0
       +  Q2 *  Q1

  Q1 :=  Clear
       + /Q1 * /Q0
       +  Q1 *  Q0

  Q0 :=  Clear
       + /Q0

FUNCTION TABLE
OE Clear Clk   /Q0 /Q1 /Q2 /Q3
------------------------------
 L   H    C     L   L   L   L
 L   L    C     H   L   L   L
 L   L    C     L   H   L   L
 L   L    C     H   H   L   L
 L   L    C     L   L   H   L
 L   H    C     L   L   L   L
------------------------------
```

PALASM design of a 4-bit counter

Though some engineers programmed PAL devices by manually editing files containing the binary fuse pattern data, most opted to design their logic using a hardware description language (HDL) such as Data I/O's ABEL, Logical Devices' CUPL, or MMI's PALASM. These were computer-assisted design (CAD) (now referred to as "electronic design automation") programs which translated (or "compiled") the designers' logic equations into binary fuse map files used to program (and often test) each device.

PALASM

The PALASM (from "PAL assembler") language was used to express boolean equations for the outputs pins in a text file which was then converted to the 'fuse map' file for the programming system using a vendor-supplied program; later the option of translation from schematics became common, and later still, 'fuse maps' could be 'synthesized' from an HDL (hardware description language,) such as Verilog.

The PALASM compiler was written by MMI in FORTRAN IV on an IBM 370/168. MMI made the source code available to users at no cost. By 1983, MMI customers ran versions on the DEC PDP-11, Data General NOVA, Hewlett-Packard HP2100, MDS800 and others.

ABEL

Data I/O Corporation released ABEL.

CUPL

Logical Devices, Inc. released the Universal Compiler for Programmable Logic (CUPL), which ran under MSDOS on the IBM PC and is currently available as an integrated development package for Microsoft Windows.

Device programmers

Popular device programmers included Data I/O Corporation's Model 60A Logic Programmer and Model 2900.

One of the very first PAL Programmers was the Structured Design "SD-20". They had the PALASM software built-in and only required a CRT terminal to enter the equations and view the fuse plots. After fusing, the outputs of the PAL could be verified if test vectors were entered in the source file.

Successors

After MMI succeeded with the 20-pin PAL parts introduced circa 1978, AMD introduced the 24-pin 22V10 PAL with additional features. After buying out MMI (circa 1987), AMD spun off a consolidated operation as Vantis, and that business was acquired by Lattice Semiconductor in 1989.

Altera introduced the EP300 (first CMOS PAL) in 1983 and later moved into the FPGA business.

Lattice Semiconductor introduced the generic array logic (GAL) family in 1985, with functional equivalents of the "V" series PALs that used reprogrammable logic planes

based on EEPROM (electrically eraseable programmable read-only memory) technology. National Semiconductor was a "second source" of GAL parts. AMD introduced a similar family called PALCE. In general one GAL part is able to function as any of the similar family PAL devices. For example the 16V8 GAL is able to replace the 16L8, 16H8, 16H6, 16H4, 16H2 and 16R8 PALs (and many others besides).

ICT (International CMOS Technology) introduced the PEEL 18CV8 in 1986. The 20-pin CMOS EEPROM part could be used in place of any of the registered-output bipolar PALs and used much less power.

Larger-scale programmable logic devices were introduced by Atmel, Lattice Semiconductor, and others. These devices extended the PAL architecture by including multiple logic planes and/or burying logic macrocells within the logic plane(s). The term "complex programmable logic device" (CPLD) was introduced to differentiate these devices from their PAL and GAL predecessors, which were then sometimes referred to as "simple programmable logic devices" or SPLDs.

Another large programmable logic device is the "field-programmable gate array" or FPGA. This term is often used to describe devices currently made by Altera and Xilinx.

Chapter 11

Programmable Logic Device

A **programmable logic device** or PLD is an electronic component used to build reconfigurable digital circuits. Unlike a logic gate, which has a fixed function, a PLD has an undefined function at the time of manufacture. Before the PLD can be used in a circuit it must be programmed, that is, reconfigured.

Using a ROM as a PLD

Before PLDs were invented, read-only memory (ROM) chips were used to create arbitrary combinational logic functions of a number of inputs. Consider a ROM with *m* inputs (the address lines) and *n* outputs (the data lines). When used as a memory, the ROM contains 2^m words of *n* bits each. Now imagine that the inputs are driven not by an *m*-bit address, but by *m* independent logic signals. Theoretically, there are 2^m possible Boolean functions of these *m* signals, but the structure of the ROM allows just 2^n of these functions to be produced at the output pins. The ROM therefore becomes equivalent to *n* separate logic circuits, each of which generates a chosen function of the *m* inputs.

The advantage of using a ROM in this way is that any conceivable function of the *m* inputs can be made to appear at any of the *n* outputs, making this the most general-purpose combinational logic device available. Also, PROMs (programmable ROMs), EPROMs (ultraviolet-erasable PROMs) and EEPROMs (electrically erasable PROMs) are available that can be programmed using a standard PROM programmer without requiring specialised hardware or software. However, there are several disadvantages:

- they are usually much slower than dedicated logic circuits,
- they cannot necessarily provide safe "covers" for asynchronous logic transitions so the PROM's outputs may glitch as the inputs switch,
- they consume more power,
- they are often more expensive than programmable logic, especially if high speed is required.

Since most ROMs do not have input or output registers, they cannot be used stand-alone for sequential logic. An external TTL register was often used for sequential designs such as state machines. Common EPROMs, for example the 2716, are still sometimes used in

this way by hobby circuit designers, who often have some lying around. This use is sometimes called a 'poor man's PAL'.

Early programmable logic

In 1969, Motorola offered the XC157, a mask-programmed gate array with 12 gates and 30 uncommitted input/output pins.

In 1970, Texas Instruments developed a mask-programmable IC based on the IBM read-only associative memory or ROAM. This device, the TMS2000, was programmed by altering the metal layer during the production of the IC. The TMS2000 had up to 17 inputs and 18 outputs with 8 JK flip flop for memory. TI coined the term Programmable Logic Array for this device.

In 1971, General Electric Company (GE) was developing a programmable logic device based on the new PROM technology. This experimental device improved on IBM's ROAM by allowing multilevel logic. Intel had just introduced the floating-gate UV erasable PROM so the researcher at GE incorporated that technology. The GE device was the first erasable PLD ever developed, predating the Altera EPLD by over a decade. GE obtained several early patents on programmable logic devices.

In 1973 National Semiconductor introduced a mask-programmable PLA device (DM7575) with 14 inputs and 8 outputs with no memory registers. This was more popular than the TI part but cost of making the metal mask limited its use. The device is significant because it was the basis for the field programmable logic array produced by Signetics in 1975, the 82S100. (Intersil actually beat Signetics to market but poor yield doomed their part.)

In 1974 GE entered into an agreement with Monolithic Memories to develop a mask-programmable logic device incorporating the GE innovations. The device was named the 'Programmable Associative Logic Array' or PALA. The MMI 5760 was completed in 1976 and could implement multilevel or sequential circuits of over 100 gates. The device was supported by a GE design environment where Boolean equations would be converted to mask patterns for configuring the device. The part was never brought to market.

PAL

MMI introduced a breakthrough device in 1978, the Programmable Array Logic or PAL. The architecture was simpler than that of Signetics FPLA because it omitted the programmable OR array. This made the parts faster, smaller and cheaper. They were available in 20 pin 300 mil DIP packages while the FPLAs came in 28 pin 600 mil packages. The PAL Handbook demystified the design process. The PALASM design software (PAL Assembler) converted the engineers' Boolean equations into the fuse pattern required to program the part. The PAL devices were soon second-sourced by National Semiconductor, Texas Instruments and AMD.

After MMI succeeded with the 20-pin PAL parts, AMD introduced the 24-pin 22V10 PAL with additional features. After buying out MMI (1987), AMD spun off a consolidated operation as Vantis, and that business was acquired by Lattice Semiconductor in 1999.

There are also PLA's : Programmable Logic Array.

GALs

Lattice GAL 16V8 and 20V8

An innovation of the PAL was the **generic array logic** device, or **GAL**, invented by Lattice Semiconductor in 1985. This device has the same logical properties as the PAL but can be erased and reprogrammed. The GAL is very useful in the prototyping stage of a design, when any bugs in the logic can be corrected by reprogramming. GALs are programmed and reprogrammed using a PAL programmer, or by using the in-circuit programming technique on supporting chips.

Lattice GALs combine CMOS and electrically erasable (E^2) floating gate technology for a high-speed, low-power logic device.

A similar device called a **PEEL** (**programmable electrically erasable logic**) was introduced by the International CMOS Technology (ICT) corporation.

CPLDs

PALs and GALs are available only in small sizes, equivalent to a few hundred logic gates. For bigger logic circuits, complex PLDs or CPLDs can be used. These contain the equivalent of several PALs linked by programmable interconnections, all in one integrated circuit. CPLDs can replace thousands, or even hundreds of thousands, of logic gates.

Some CPLDs are programmed using a PAL programmer, but this method becomes inconvenient for devices with hundreds of pins. A second method of programming is to solder the device to its printed circuit board, then feed it with a serial data stream from a personal computer. The CPLD contains a circuit that decodes the data stream and configures the CPLD to perform its specified logic function.

Each manufacturer has a proprietary name for this programming system. For example, Lattice Semiconductor calls it "in-system programming". However, these proprietary systems are beginning to give way to a standard from the Joint Test Action Group, JTAG.

FPGAs

While PALs were busy developing into GALs and CPLDs (all discussed above), a separate stream of development was happening. This type of device is based on gate array technology and is called the field-programmable gate array (FPGA). Early examples of FPGAs are the 82s100 array, and 82S105 sequencer, by Signetics, introduced in the late 1970s. The 82S100 was an array of AND terms. The 82S105 also had flip flop functions.

FPGAs use a grid of logic gates, and once stored, the data doesn't change, similar to that of an ordinary gate array. The term "field-programmable" means the device is programmed by the customer, not the manufacturer.

FPGAs are usually programmed after being soldered down to the circuit board, in a manner similar to that of larger CPLDs. In most larger FPGAs the configuration is volatile, and must be re-loaded into the device whenever power is applied or different functionality is required. Configuration is typically stored in a configuration PROM or EEPROM. EEPROM versions may be in-system programmable (typically via JTAG).

The difference between FPGAs and CPLDs is that FPGAs are internally based on Look-up tables (LUTs) whereas CPLDs form the logic functions with sea-of-gates (e.g. sum of

products). CPLDs are meant for simpler designs while FPGAs are meant for more complex designs. In general, CPLDs are a good choice for wide combinational logic applications, whereas FPGAs are more suitable for large state machines (i.e. microprocessors).

Other variants

At present, much interest exists in reconfigurable systems. These are microprocessor circuits that contain some fixed functions and other functions that can be altered by code running on the processor. Designing self-altering systems requires engineers to learn new methods, and that new software tools be developed.

PLDs are being sold now that contain a microprocessor with a fixed function (the so-called *core*) surrounded by programmable logic. These devices let designers concentrate on adding new features to designs without having to worry about making the microprocessor work.

How PLDs retain their configuration

A PLD is a combination of a logic device and a memory device. The memory is used to store the pattern that was given to the chip during programming. Most of the methods for storing data in an integrated circuit have been adapted for use in PLDs. These include:

- Silicon antifuses
- SRAM
- EPROM or EEPROM cells
- Flash memory

Silicon antifuses are the storage elements used in the PAL, the first type of PLD. These are connections that are made by applying a voltage across a modified area of silicon inside the chip. They are called antifuses because they work in the opposite way to normal fuses, which begin life as connections until they are broken by an electric current.

SRAM, or static RAM, is a volatile type of memory, meaning that its contents are lost each time the power is switched off. SRAM-based PLDs therefore have to be programmed every time the circuit is switched on. This is usually done automatically by another part of the circuit.

An EPROM cell is a MOS (metal-oxide-semiconductor) transistor that can be switched on by trapping an electric charge permanently on its gate electrode. This is done by a PAL programmer. The charge remains for many years and can only be removed by exposing the chip to strong ultraviolet light in a device called an EPROM eraser.

Flash memory is non-volatile, retaining its contents even when the power is switched off. It can be erased and reprogrammed as required. This makes it useful for PLD memory.

As of 2005, most CPLDs are electrically programmable and erasable, and non-volatile. This is because they are too small to justify the inconvenience of programming internal SRAM cells every time they start up, and EPROM cells are more expensive due to their ceramic package with a quartz window.

PLD programming languages

Many PAL programming devices accept input in a standard file format, commonly referred to as 'JEDEC files'.They are analogous to software compilers. The languages used as source code for logic compilers are called hardware description languages, or HDLs.

PALASM and ABEL are frequently used for low-complexity devices, while Verilog and VHDL are popular higher-level description languages for more complex devices. The more limited ABEL is often used for historical reasons, but for new designs VHDL is more popular, even for low-complexity designs.

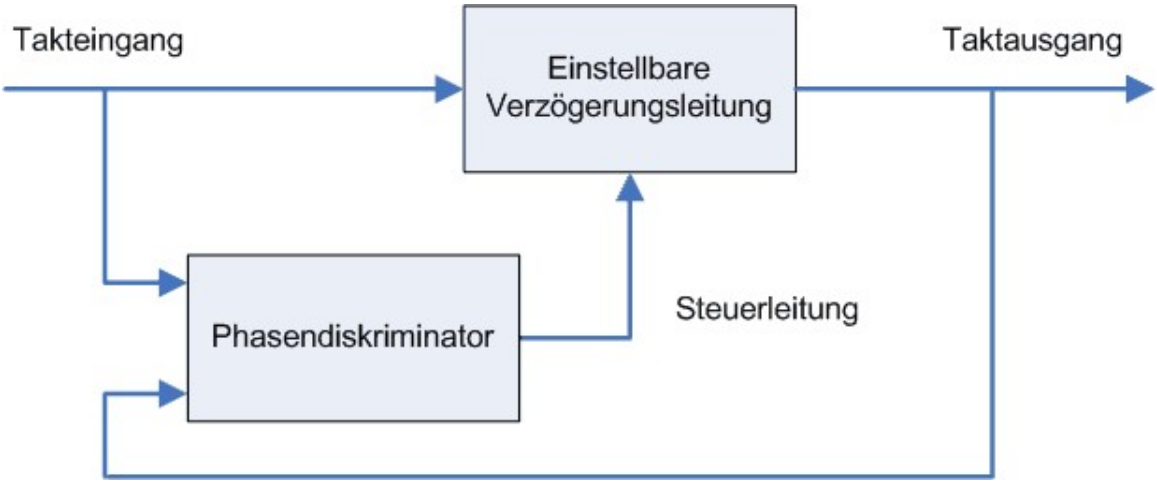
PLD programming devices

A device programmer is used to transfer the boolean logic pattern into the programmable device. In the early days of programmable logic, every PLD manufacturer also produced a specialized device programmer for its family of logic devices. Later, universal device programmers came onto the market that supported several logic device families from different manufacturers. Today's device programmers usually can program common PLDs (mostly PAL/GAL equivalents) from all existing manufacturers. Common file formats used to store the boolean logic pattern (fuses) are JEDEC, Altera POF (Programmable Object File), or Xilinx BITstream.

Chapter 12

Delay-Locked Loop & Complex Programmable Logic Device

Delay-Locked Loop



In electronics, a **delay-locked loop** (DLL) is a digital circuit similar to a phase-locked loop (PLL), with the main difference being the absence of an internal voltage-controlled oscillator. A DLL can be used to change the phase of a clock signal (a signal with a periodic waveform), usually to enhance the *clock rise*-to-*data output valid* timing characteristics of integrated circuits (such as DRAM devices). DLLs can also be used for clock recovery (CDR). From the outside, a DLL can be seen as a negative-delay gate placed in the clock path of a digital circuit.

Another way to view the difference between a DLL and a PLL is that a DLL is a first order loop and a PLL is a second order loop. A DLL compares the phase of one of its outputs to the input clock to generate an error signal which is then integrated and fed back as the control to all of the delay elements. The integration allows the error to go to zero while keeping the control signal, and thus the delays, where they need to be for phase lock. Since the control signal directly impacts the phase this is all that is required. A PLL compares the phase of its oscillator with the incoming signal to generate an error signal which is then integrated to create a control signal for the voltage-controlled

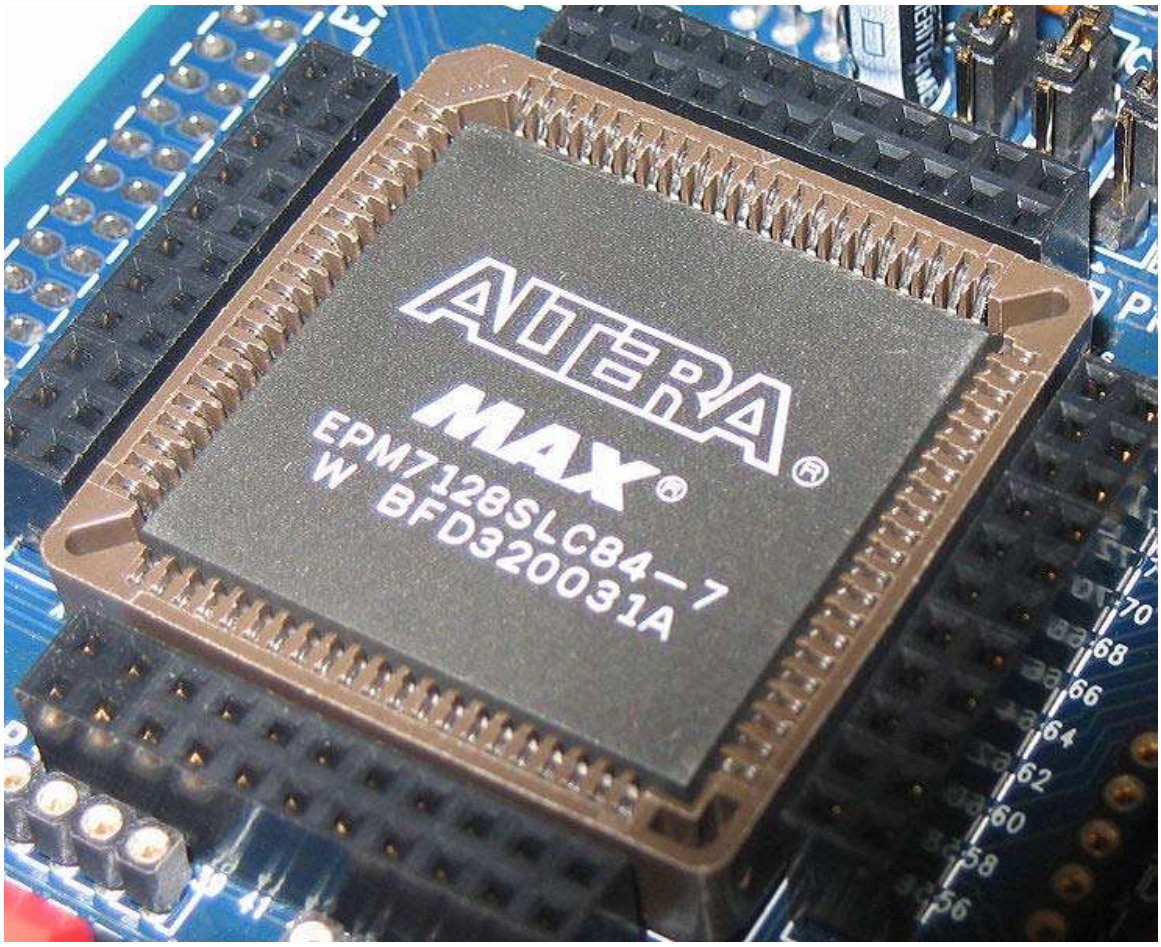
oscillator. The control signal impacts the frequency of the oscillator, and phase is the integral of frequency, so a second integration is unavoidably performed by the oscillator itself. A first order feedback system is significantly easier to stabilize than a second order feedback system, which is a major advantage of DLLs.

The main component of a DLL is a delay chain composed of many delay gates connected front-to-back. The input of the chain (and thus of the DLL) is connected to the clock that is to be negatively delayed. A multiplexer is connected to each stage of the delay chain; the selector of this multiplexer is automatically updated by a control circuit to produce the negative delay effect. The output of the DLL is the resulting, negatively delayed clock signal.

The phase shift can be specified either in absolute terms (in delay chain gate units), or as a proportion of the clock period, or both.

Compared to phase-locked loops, delay-locked loops are a relatively recent innovation, first found in Dr. Combes' work in the early 1990s, then popularized by Xilinx in their Virtex family of FPGA products.

Complex Programmable Logic Device



An Altera MAX 7000-series CPLD with 2500 gates

A **complex programmable logic device (CPLD)** is a programmable logic device with complexity between that of PALs and FPGAs, and architectural features of both. The building block of a CPLD is the macrocell, which contains logic implementing disjunctive normal form expressions and more specialized logic operations.

Features in common with PALs:

- Non-volatile configuration memory. Unlike many FPGAs, an external configuration ROM isn't required, and the CPLD can function immediately on system start-up.
- For many legacy CPLD devices, routing constrains most logic blocks to have input and output signals connected to external pins, reducing opportunities for internal state storage and deeply layered logic. This is usually not a factor for larger CPLDs and newer CPLD product families.

Features in common with FPGAs:

- Large number of gates available. CPLDs typically have the equivalent of thousands to tens of thousands of logic gates, allowing implementation of moderately complicated data processing devices. PALs typically have a few hundred gate equivalents at most, while FPGAs typically range from tens of thousands to several million.
- Some provisions for logic more flexible than sum-of-product expressions, including complicated feedback paths between macro cells, and specialized logic for implementing various commonly-used functions, such as integer arithmetic.

The most noticeable difference between a large CPLD and a small FPGA is the presence of on-chip non-volatile memory in the CPLD. This distinction is rapidly becoming less relevant, as several of the latest FPGA products also offer models with embedded configuration memory.

The characteristic of non-volatility makes the CPLD the device of choice in modern digital designs to perform 'boot loader' functions before handing over control to other devices not having this capability. A good example is where a CPLD is used to load configuration data for an FPGA from non-volatile memory.

CPLDs were an evolutionary step from even smaller devices that preceded them, PLAs (first shipped by Signetics), and PALs. These in turn were preceded by standard logic products, that offered no programmability and were "programmed" by wiring several standard logic chips together.

The main distinction between FPGA and CPLD device architectures is that FPGAs are internally based on Look-up tables (LUTs) while CPLDs form the logic functions with sea-of-gates (e.g. sum of products).

Chapter 13

Partial Re-Configuration & Rent's Rule

Partial Re-Configuration

Partial Reconfiguration is the process of configuring a portion of a field programmable gate array while the other part is still running/operating.

Hardware, like software, can be designed modularly, by creating subcomponents and then higher-level components to instantiate them. In many cases it is useful to be able to swap out one or several of these subcomponents while the FPGA is still operating.

Normally, reconfiguring an FPGA requires it to be held in reset while an external controller reloads a design onto it. Partial reconfiguration allows for critical parts of the design to continue operating while a controller either on the FPGA or off of it loads a partial design into a reconfigurable module. Partial reconfiguration also can be used to save space for multiple designs by only storing the partial designs that change between designs.

A common example for when partial reconfiguration would be useful is the case of a communication device. If the device is controlling multiple connections, some of which require encryption, it would be useful to be able to load different encryption cores without bringing the whole controller down.

Partial reconfiguration is not supported on all FPGAs. A special software flow with emphasis on modular design is required. Typically the design modules are built along well defined boundaries inside the FPGA that require the design to be specially mapped to the internal hardware.

From the functionality of the design, partial reconfiguration can be divided into two groups:

- **dynamic partial reconfiguration**, also known as an **active** partial reconfiguration - permits to change the part of the device while the rest of an FPGA is still running;
- **static partial reconfiguration -** the device is not active during the reconfiguration process. While the partial data is sent into the FPGA, the rest of

the device is stopped (in the shutdown mode) and brought up after the configuration is completed.

There are two styles of partial reconfiguration of FPGA devices from Xilinx: **module-based** and **difference-based**.

Module-based partial reconfiguration permits to reconfigure distinct modular parts of the design. To ensure the communication across the reconfigurable module boundaries, special bus macros ought to be prepared. It works as a fixed routing bridge that connects the reconfigurable module with the rest part of the design. Module-based partial reconfiguration requires to perform a set of specific guidelines during at the stage of design specification. Finally for each reconfigurable module of the design, separate bit-stream is created. Such a bit-stream is used to perform the partial reconfiguration of an FPGA.

Difference-based partial reconfiguration can be used when a small change is made to the design. It is especially useful in case of changing LUT equations or dedicated memory blocks content. The partial bit-stream contains only information about differences between the current design structure (that resides in the FPGA) and the new content of an FPGA. There are two ways of difference-based reconfiguration known as a front-end and back-end. The first one is based on the modification of the design in the hardware description languages (HDLs). It is clear that such a solution requires full repeating of the synthesis and implementation processes. The back-end difference-based partial reconfiguration permits to make changes at the implementation stage of the prototyping flow. Therefore there is no need for re-synthesis of the design. The usage of both methods (either front-end and back-end) leads to creation of a partial bit-stream that can be used for a partial reconfiguration of the FPGA.

Rent's Rule

Rent's rule pertains to the organization of computing logic, specifically the relationship between the number of external signal connections to a logic block (i.e., the number of "pins") with the number of logic gates in the logic block, and has been applied to circuits ranging from small digital circuits to mainframe computers.

E.F. Rent's discovery and first publications

In the 1960's, E.F. Rent, an IBM employee, found a remarkable trend between the number of pins (terminals T) at the boundaries of integrated circuit designs at IBM and the number of internal components (g), such as logic gates or standard cells. On a log − log plot, these datapoints were on a straight line, implying a power-law relation $T = tg^p$ where t and p are constants ($p < 1.0$, and generally $0.5 < p < 0.8$). Rent disclosed his

findings in IBM-internal memoranda, but the relation was described in 1971 by Landman and Russo. They performed a hierarchical circuit partitioning in such a way that at each hierarchical level (top-down) the least number of interconnections had to be cut to partition the circuit (in more or less equal parts). At each partitioning step, they noted the number of terminals and the number of components in each partition and then partitioned the sub-partitions further. They found the power law rule applied to the resulting T versus g plot and named it "Rent's rule". It is crucial to recognise that Rent's rule is an empirical result based on observations of existing designs, and therefore it is less applicable to the analysis of non-traditional circuit architectures. Having said that, it does provide a useful framework with which to compare similar architectures.

Theoretical basis

Christie and Stroobandt later derived Rent's rule theoretically for homogeneous systems and pointed out that the amount of optimization achieved in placement is reflected by the parameter p, the "Rent exponent", which also depends on the circuit topology. In particular, values $p < 1$ correspond to a greater fraction of short interconnects. The constant t in Rent's rule can be viewed as the average number of terminals required by a single logic block since $T = t$ when $g = 1$.

Special cases and applications

Random arrangement of logic blocks typically have $p = 1$. Larger values are impossible since the maximum number of terminals for any region containing g logic components in a homogeneous system is given by $T = tg$. Lower bounds on p depend on the interconnection topology since it is generally impossible to make all wires short. This lower bound p^* is often called the "intrinsic Rent exponent", a notion first introduced by Hagen et al. It can be used to characterize optimal placements and also measure the interconnection complexity of a circuit. Higher (intrinsic) Rent exponent values correspond to a higher topological complexity. One extreme example ($p = 0$) is a long chain of logic blocks, while a clique has $p = 1$. In realistic 2D circuits, p^* ranges from 0.5 for highly-regular circuits (such as SRAM) to 0.75 for random logic.

System performance analysis tools such as BACPAC typically use Rent's rule to calculate expected wiring lengths and wiring demands.

Estimating Rent's exponent

To estimate Rent's exponent, one can use top-down partitioning, as used in min-cut placement. For every partition, count the number of terminals connected to the partition and compare it to the number of logic blocks in the partition. Rent's exponent can then be found by fitting these datapoints on a log-log plot, resulting in an exponent p'. For optimally partitioned circuits, $p' = p^*$ but this is no longer the case for practical

(heuristic) partitioning approaches. For partitioning-based placement algorithms $p^* \leq p' \leq p$.

Region II of Rent's rule

Landman and Russo found a deviation of Rent's rule near the "far end", i.e., for partitions with a large number of blocks, which is known as "Region II" of Rent's Rule . A similar deviation exists at for small partitions, and has been found by Stroobandt who called it Region III.

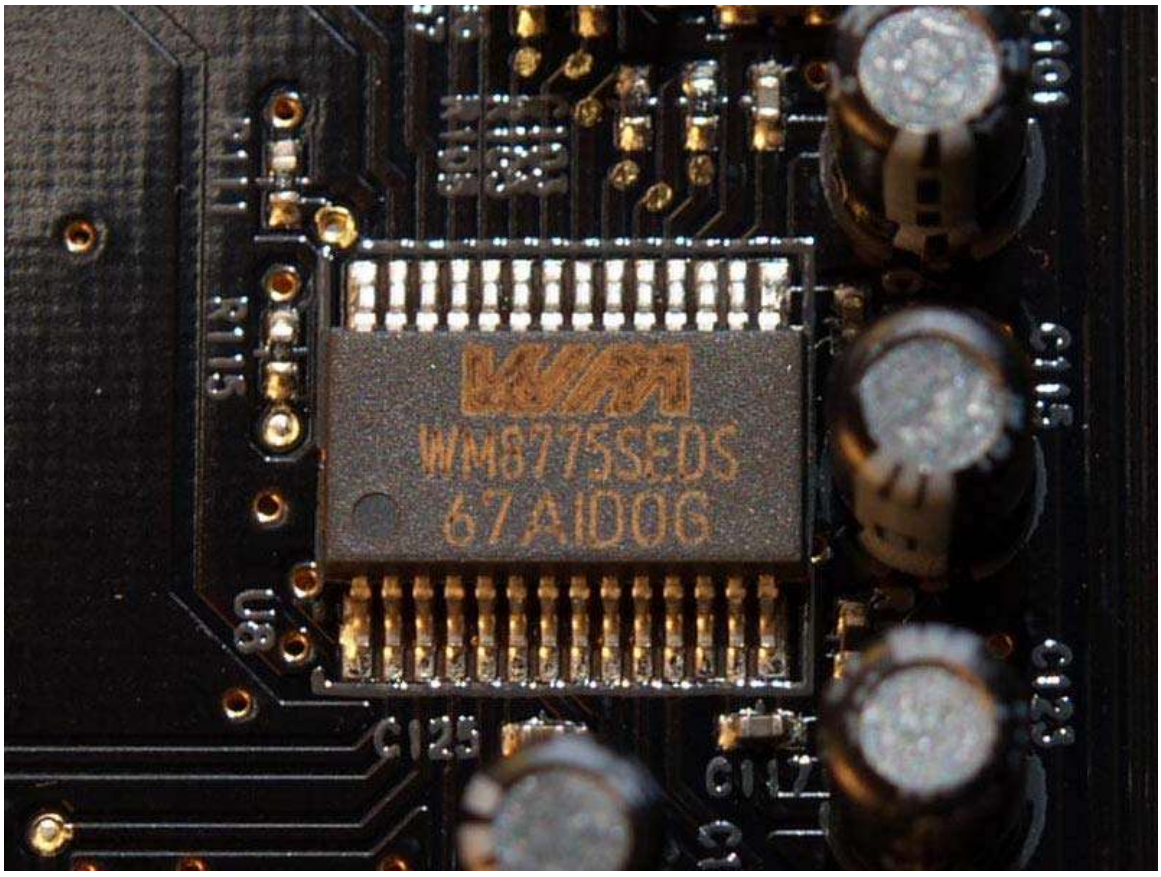
Rentian wirelength estimation

Another IBM employee, Donath, discovered that Rent's rule can be used to estimate the average wirelength and the wirelength distribution in VLSI chips. This motivated the System Level Interconnect Prediction workshop, founded in 1999, and an entire community working on wirelength prediction. The resulting wirelength estimates have been improved significantly since then and are now used for "technology exploration." The use of Rent's rule allows to perform such estimates *a priori* (i.e., before actual placement) and thus predict the properties of future technologies (clock frequencies, number of routing layers needed, area, power) based on limited information about future circuits and technologies.

A comprehensive overview of work based on Rent's rule has been published by Stroobandt.

Chapter 14

Analog-to-Digital Converter



4-channel stereo multiplexed analog-to-digital converter WM8775SEDS made by Wolfson Microelectronics placed on a X-Fi Fatal1ty Pro sound card.

An **analog-to-digital converter** (abbreviated **ADC**, **A/D** or **A to D**) is a device which converts a continuous quantity to a discrete time digital representation. An ADC may also provide an isolated measurement. The reverse operation is performed by a digital-to-analog converter (**DAC**).

Typically, an ADC is an electronic device that converts an input analog voltage or current to a digital number proportional to the magnitude of the voltage or current. However,

some non-electronic or only partially electronic devices, such as rotary encoders, can also be considered ADCs.

The digital output may use different coding schemes. Typically the digital output will be a two's complement binary number that is proportional to the input, but there are other possibilities. An encoder, for example, might output a Gray code.

Concepts

Resolution

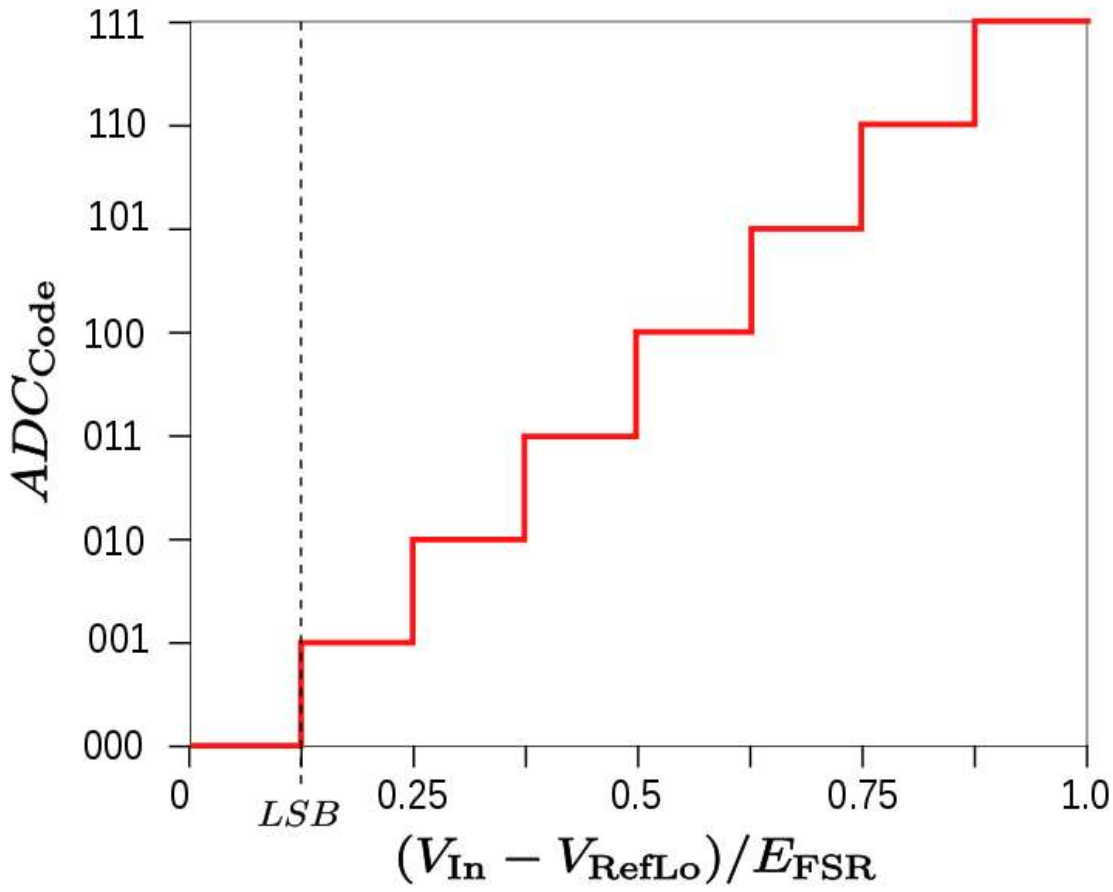


Fig. 1. An 8-level ADC coding scheme

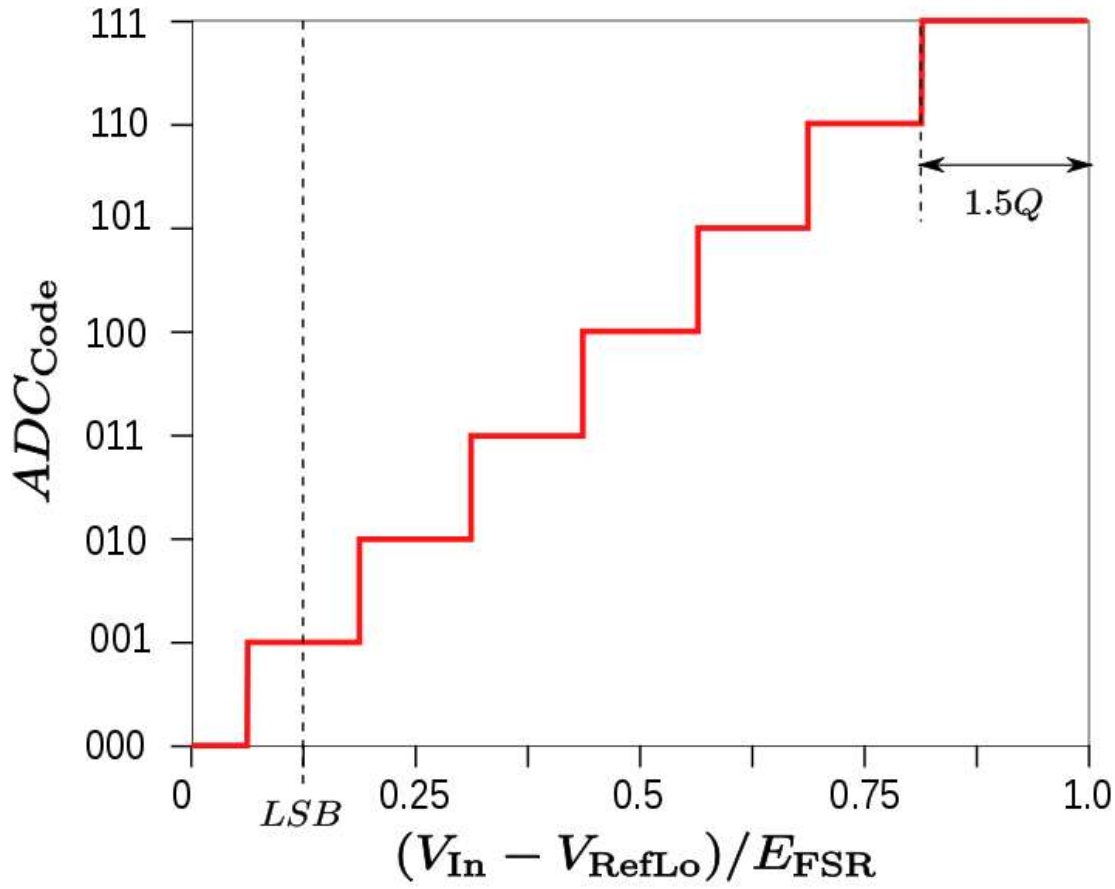


Fig. 2. An 8-level ADC coding scheme. As in figure 1 but with mid-tread coding.

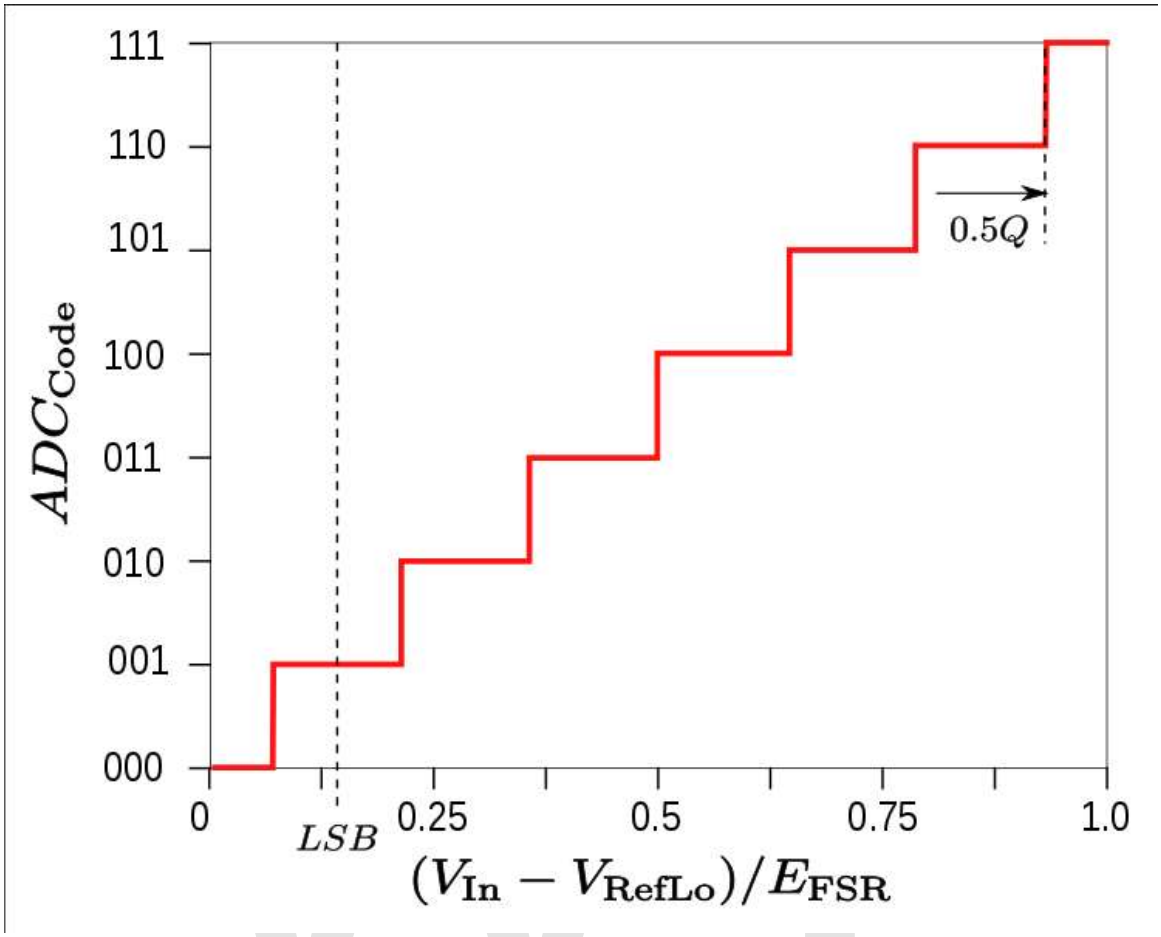


Fig. 3. An 8-level ADC mid-tread coding scheme. As in figure 2 but with equal half-*LSB* intervals at the highest and lowest codes. Note that *LSB* is now slightly larger than in figures 1 and 2.

The resolution of the converter indicates the number of discrete values it can produce over the range of analog values. The values are usually stored electronically in binary form, so the resolution is usually expressed in bits. In consequence, the number of discrete values available, or "levels", is usually a power of two. For example, an ADC with a resolution of 8 bits can encode an analog input to one in 256 different levels, since $2^8 = 256$. The values can represent the ranges from 0 to 255 (i.e. unsigned integer) or from −128 to 127 (i.e. signed integer), depending on the application.

Resolution can also be defined electrically, and expressed in volts. The minimum change in voltage required to guarantee a change in the output code level is called the *LSB* (least significant bit, since this is the voltage represented by a change in the LSB). The resolution *Q* of the ADC is equal to the *LSB* voltage. The voltage resolution of an ADC is equal to its overall voltage measurement range divided by the number of discrete voltage intervals:

$$Q = \frac{E_{\mathrm{FSR}}}{N},$$

where N is the number of voltage intervals and E_{FSR} is the full scale voltage range. E_{FSR} is given by

$$E_{\mathrm{FSR}} = V_{\mathrm{RefHi}} - V_{\mathrm{RefLow}},$$

where V_{RefHi} and V_{RefLow} are the upper and lower extremes, respectively, of the voltages that can be coded.

Normally, the number of voltage intervals is given by

$$N = 2^M,$$

where M is the ADC's resolution in bits.

That is, one voltage interval is assigned per code level. However, figure 3 shows a situation where

$$N = 2^M - 1$$

Some examples:

- Example 1
  - Coding scheme as in figure 1
  - Full scale measurement range = 0 to 10 volts
  - ADC resolution is 12 bits: 2^{12} = 4096 quantization levels (codes)
  - ADC voltage resolution, Q = (10 V − 0 V) / 4096 = 10 V / 4096 ≈ 0.00244 V ≈ 2.44 mV.

- Example 2
  - Coding scheme as in figure 2
  - Full scale measurement range = -10 to +10 volts
  - ADC resolution is 14 bits: 2^{14} = 16384 quantization levels (codes)
  - ADC voltage resolution is, Q = (10 V − (−10 V)) / 16384 = 20 V / 16384 ≈ 0.00122 V ≈ 1.22 mV.

- Example 3
  - Coding scheme as in figure 3
  - Full scale measurement range = 0 to 7 volts
  - ADC resolution is 3 bits: 2^3 = 8 quantization levels (codes)
  - ADC voltage resolution is, Q = (7 V − 0 V)/7 = 7 V/7 = 1 V = 1000 mV

In most ADCs, the smallest output code ("0" in an unsigned system) represents a voltage range which is 0.5Q, that is, half the ADC voltage resolution (Q). The largest code

represents a range of 1.5Q as in figure 2 (if this were 0.5Q also, the result would be as figure 3). The other $N-2$ codes are all equal in width and represent the ADC voltage resolution (Q) calculated above. Doing this centers the code on an input voltage that represents the M th division of the input voltage range. This practice is called "mid-tread" operation. This type of ADC can be modeled mathematically as:

$$ADC_{\text{Code}} = \text{round}\left(\left(\frac{2^M}{V_{\text{RefHi}} - V_{\text{RefLow}}}\right) \cdot (V_{\text{In}} - V_{\text{RefLow}})\right)$$

The exception to this convention seems to be the Microchip PIC processor, where all M steps are equal width, as shown in figure 1. This practice is called "Mid-Rise with Offset" operation.

$$ADC_{\text{Code}} = \text{floor}\left(\left(\frac{2^M}{V_{\text{RefHi}} - V_{\text{RefLow}}}\right) \cdot (V_{\text{In}} - V_{\text{RefLow}})\right)$$

In practice, the useful resolution of a converter is limited by the best signal-to-noise ratio (SNR) that can be achieved for a digitized signal. An ADC can resolve a signal to only a certain number of bits of resolution, called the effective number of bits (ENOB). One effective bit of resolution changes the signal-to-noise ratio of the digitized signal by 6 dB, if the resolution is limited by the ADC. If a preamplifier has been used prior to A/D conversion, the noise introduced by the amplifier can be an important contributing factor towards the overall SNR.

Response type

Linear ADCs

Most ADCs are of a type known as linear The term *linear* implies here the range of the input values that map to each output value has a linear relationship with the output value, i.e., that the output value k is used for the range of input values from

$$m(k + b)$$

to

$$m(k + 1 + b),$$

where m and b are constants. Here b is typically 0 or −0.5. When $b = 0$, the ADC is referred to as *mid-rise*, and when $b = -0.5$ it is referred to as *mid-tread*.

Non-linear ADCs

If the probability density function of a signal being digitized is uniform, then the signal-to-noise ratio relative to the quantization noise is the best possible. Because this is often not the case, it is usual to pass the signal through its cumulative distribution function (CDF) before the quantization. This is good because the regions that are more important get quantized with a better resolution. In the dequantization process, the inverse CDF is needed.

This is the same principle behind the companders used in some tape-recorders and other communication systems, and is related to entropy maximization.

For example, a voice signal has a Laplacian distribution. This means that the region around the lowest levels, near 0, carries more information than the regions with higher amplitudes. Because of this, logarithmic ADCs are very common in voice communication systems to increase the dynamic range of the representable values while retaining fine-granular fidelity in the low-amplitude region.

An eight-bit A-law or the μ-law logarithmic ADC covers the wide dynamic range and has a high resolution in the critical low-amplitude region, that would otherwise require a 12-bit linear ADC.

Accuracy

An ADC has several sources of errors. Quantization error and (assuming the ADC is intended to be linear) non-linearity are intrinsic to any analog-to-digital conversion. There is also a so-called *aperture error* which is due to a clock jitter and is revealed when digitizing a time-variant signal (not a constant value).

These errors are measured in a unit called the *LSB*, which is an abbreviation for least significant bit. In the above example of an eight-bit ADC, an error of one LSB is 1/256 of the full signal range, or about 0.4%.

Quantization error

Quantization error (or quantization noise) is the difference between the original signal and the digitized signal. Hence, The magnitude of the quantization error at the sampling instant is between zero and half of one LSB. Quantization error is due to the finite resolution of the digital representation of the signal, and is an unavoidable imperfection in all types of ADCs.

Non-linearity

All ADCs suffer from non-linearity errors caused by their physical imperfections, causing their output to deviate from a linear function (or some other function, in the case of a

deliberately non-linear ADC) of their input. These errors can sometimes be mitigated by calibration, or prevented by testing.

Important parameters for linearity are integral non-linearity (INL) and differential non-linearity (DNL). These non-linearities reduce the dynamic range of the signals that can be digitized by the ADC, also reducing the effective resolution of the ADC.

Aperture error

Imagine that we are digitizing a sine wave $x(t) = A\sin(2\pi f_0 t)$. Provided that the actual sampling time *uncertainty* due to the *clock jitter* is Δt, the error caused by this phenomenon can be estimated as $E_{ap} \leq |x'(t)\Delta t| \leq 2A\pi f_0 \Delta t$.

The error is zero for DC, small at low frequencies, but significant when high frequencies have high amplitudes. This effect can be ignored if it is drowned out by the *quantizing error*. Jitter requirements can be calculated using the following formula: $\Delta t < \frac{1}{2^q \pi f_0}$, where q is a number of ADC bits.

ADC resolution in bit	input frequency						
	1 Hz	44.1 kHz	192 kHz	1 MHz	10 MHz	100 MHz	1 GHz
8	1243 µs	28.2 ns	6.48 ns	1.24 ns	124 ps	12.4 ps	1.24 ps
10	311 µs	7.05 ns	1.62 ns	311 ps	31.1 ps	3.11 ps	0.31 ps
12	77.7 µs	1.76 ns	405 ps	77.7 ps	7.77 ps	0.78 ps	0.08 ps
14	19.4 µs	441 ps	101 ps	19.4 ps	1.94 ps	0.19 ps	0.02 ps
16	4.86 µs	110 ps	25.3 ps	4.86 ps	0.49 ps	0.05 ps	–
18	1.21 µs	27.5 ps	6.32 ps	1.21 ps	0.12 ps	–	–
20	304 ns	6.88 ps	1.58 ps	0.16 ps	–	–	–
24	19.0 ns	0.43 ps	0.10 ps	–	–	–	–
32	74.1 ps	–	–	–	–	–	–

This table shows, for example, that it is not worth using a precise 24-bit ADC for sound recording if there is not an *ultra low jitter* clock. One should consider taking this phenomenon into account before choosing an ADC.

Clock jitter is caused by phase noise. The resolution of ADCs with a digitization bandwidth between 1 MHz and 1 GHz is limited by jitter.

When sampling audio signals at 44.1 kHz, the anti-aliasing filter should have eliminated all frequencies above 22 kHz. The input frequency (in this case, 22 kHz), not the ADC clock frequency, is the determining factor with respect to jitter performance.

Sampling rate

The analog signal is continuous in time and it is necessary to convert this to a flow of digital values. It is therefore required to define the rate at which new digital values are sampled from the analog signal. The rate of new values is called the *sampling rate* or *sampling frequency* of the converter.

A continuously varying bandlimited signal can be sampled (that is, the signal values at intervals of time T, the sampling time, are measured and stored) and then the original signal can be *exactly* reproduced from the discrete-time values by an interpolation formula. The accuracy is limited by quantization error. However, this faithful reproduction is only possible if the sampling rate is higher than twice the highest frequency of the signal. This is essentially what is embodied in the Shannon-Nyquist sampling theorem.

Since a practical ADC cannot make an instantaneous conversion, the input value must necessarily be held constant during the time that the converter performs a conversion (called the *conversion time*). An input circuit called a sample and hold performs this task—in most cases by using a capacitor to store the analog voltage at the input, and using an electronic switch or gate to disconnect the capacitor from the input. Many ADC integrated circuits include the sample and hold subsystem internally.

Aliasing

All ADCs work by sampling their input at discrete intervals of time. Their output is therefore an incomplete picture of the behaviour of the input. There is no way of knowing, by looking at the output, what the input was doing between one sampling instant and the next. If the input is known to be changing slowly compared to the sampling rate, then it can be assumed that the value of the signal between two sample instants was somewhere between the two sampled values. If, however, the input signal is changing rapidly compared to the sample rate, then this assumption is not valid.

If the digital values produced by the ADC are, at some later stage in the system, converted back to analog values by a digital to analog converter or DAC, it is desirable that the output of the DAC be a faithful representation of the original signal. If the input signal is changing much faster than the sample rate, then this will not be the case, and spurious signals called *aliases* will be produced at the output of the DAC. The frequency of the aliased signal is the difference between the signal frequency and the sampling rate. For example, a 2 kHz sine wave being sampled at 1.5 kHz would be reconstructed as a 500 Hz sine wave. This problem is called *aliasing*.

To avoid aliasing, the input to an ADC must be low-pass filtered to remove frequencies above half the sampling rate. This filter is called an *anti-aliasing* filter, and is essential for a practical ADC system that is applied to analog signals with higher frequency content.

Although aliasing in most systems is unwanted, it should also be noted that it can be exploited to provide simultaneous down-mixing of a band-limited high frequency signal.

Dither

In A-to-D converters, performance can usually be improved using dither. This is a very small amount of random noise (white noise) which is added to the input before conversion. Its amplitude is set to be twice the value of the least significant bit. Its effect is to cause the state of the LSB to randomly oscillate between 0 and 1 in the presence of very low levels of input, rather than sticking at a fixed value. Rather than the signal simply getting cut off altogether at this low level (which is only being quantized to a resolution of 1 bit), it extends the effective range of signals that the A-to-D converter can convert, at the expense of a slight increase in noise - effectively the quantization error is diffused across a series of noise values which is far less objectionable than a hard cutoff. The result is an accurate representation of the signal over time. A suitable filter at the output of the system can thus recover this small signal variation.

An audio signal of very low level (with respect to the bit depth of the ADC) sampled without dither sounds extremely distorted and unpleasant. Without dither the low level may cause the least significant bit to "stick" at 0 or 1. With dithering, the true level of the audio may be calculated by averaging the actual quantized sample with a series of other samples [the dither] that are recorded over time.

A virtually identical process, also called dither or dithering, is often used when quantizing photographic images to a fewer number of bits per pixel—the image becomes noisier but to the eye looks far more realistic than the quantized image, which otherwise becomes banded. This analogous process may help to visualize the effect of dither on an analogue audio signal that is converted to digital.

Dithering is also used in integrating systems such as electricity meters. Since the values are added together, the dithering produces results that are more exact than the LSB of the analog-to-digital converter.

Note that dither can only increase the resolution of a sampler, it cannot improve the linearity, and thus accuracy does not necessarily improve.

Oversampling

Usually, signals are sampled at the minimum rate required, for economy, with the result that the quantization noise introduced is white noise spread over the whole pass band of the converter. If a signal is sampled at a rate much higher than the Nyquist frequency and then digitally filtered to limit it to the signal bandwidth there are the following advantages:

- digital filters can have better properties (sharper rolloff, phase) than analogue filters, so a sharper anti-aliasing filter can be realised and then the signal can be downsampled giving a better result
- a 20-bit ADC can be made to act as a 24-bit ADC with 256× oversampling
- the signal-to-noise ratio due to quantization noise will be higher than if the whole available band had been used. With this technique, it is possible to obtain an effective resolution larger than that provided by the converter alone
- The improvement in SNR is 3 dB (equivalent to 0.5 bits) per octave of oversampling which is not sufficient for many applications. Therefore, oversampling is usually coupled with noise shaping. With noise shaping, the improvement is 6L+3 dB per octave where L is the order of loop filter used for noise shaping. e.g. - a 2nd order loop filter will provide an improvement of 15 dB/octave.

Relative speed and precision

The speed of an ADC varies by type. The Wilkinson ADC is limited by the clock rate which is processable by current digital circuits. Currently, frequencies up to 300 MHz are possible. The conversion time is directly proportional to the number of channels. For a successive approximation ADC, the conversion time scales with the logarithm of the number of channels. Thus for a large number of channels, it is possible that the successive approximation ADC is faster than the Wilkinson. However, the time consuming steps in the Wilkinson are digital, while those in the successive approximation are analog. Since analog is inherently slower than digital, as the number of channels increases, the time required also increases. Thus there are competing processes at work. Flash ADCs are certainly the fastest type of the three. The conversion is basically performed in a single parallel step. For an 8-bit unit, conversion takes place in a few tens of nanoseconds.

There is, as expected, somewhat of a trade off between speed and precision. Flash ADCs have drifts and uncertainties associated with the comparator levels, which lead to poor uniformity in channel width. Flash ADCs have a resulting poor linearity. For successive approximation ADCs, poor linearity is also apparent, but less so than for flash ADCs. Here, non-linearity arises from accumulating errors from the subtraction processes. Wilkinson ADCs are the best of the three. These have the best differential non-linearity. The other types require channel smoothing in order to achieve the level of the Wilkinson.

The sliding scale principle

The sliding scale or randomizing method can be employed to greatly improve the channel width uniformity and differential linearity of any type of ADC, but especially flash and successive approximation ADCs. Under normal conditions, a pulse of a particular amplitude is always converted to a certain channel number. The problem lies in that channels are not always of uniform width, and the differential linearity decreases proportionally with the divergence from the average width. The sliding scale principle uses an averaging effect to overcome this phenomenon. A random, but known analog

voltage is added to the input pulse. It is then converted to digital form, and the equivalent digital version is subtracted, thus restoring it to its original value. The advantage is that the conversion has taken place at a random point. The statistical distribution of the final channel numbers is decided by a weighted average over a region of the range of the ADC. This in turn desensitizes it to the width of any given channel.

ADC structures

These are the most common ways of implementing an electronic ADC:

- A **direct conversion ADC** or **flash ADC** has a bank of comparators sampling the input signal in parallel, each firing for their decoded voltage range. The comparator bank feeds a logic circuit that generates a code for each voltage range. Direct conversion is very fast, capable of gigahertz sampling rates, but usually has only 8 bits of resolution or fewer, since the number of comparators needed, $2^N - 1$, doubles with each additional bit, requiring a large expensive circuit. ADCs of this type have a large die size, a high input capacitance, high power dissipation, and are prone to produce glitches on the output (by outputting an out-of-sequence code). Scaling to newer submicrometre technologies does not help as the device mismatch is the dominant design limitation. They are often used for video, wideband communications or other fast signals in optical storage.

- A **successive-approximation ADC** uses a comparator to reject ranges of voltages, eventually settling on a final voltage range. Successive approximation works by constantly comparing the input voltage to the output of an internal digital to analog converter (DAC, fed by the current value of the approximation) until the best approximation is achieved. At each step in this process, a binary value of the approximation is stored in a successive approximation register (SAR). The SAR uses a reference voltage (which is the largest signal the ADC is to convert) for comparisons. For example if the input voltage is 60 V and the reference voltage is 100 V, in the 1st clock cycle, 60 V is compared to 50 V (the reference, divided by two. This is the voltage at the output of the internal DAC when the input is a '1' followed by zeros), and the voltage from the comparator is positive (or '1') (because 60 V is greater than 50 V). At this point the first binary digit (MSB) is set to a '1'. In the 2nd clock cycle the input voltage is compared to 75 V (being halfway between 100 and 50 V: This is the output of the internal DAC when its input is '11' followed by zeros) because 60 V is less than 75 V, the comparator output is now negative (or '0'). The second binary digit is therefore set to a '0'. In the 3rd clock cycle, the input voltage is compared with 62.5 V (halfway between 50 V and 75 V: This is the output of the internal DAC when its input is '101' followed by zeros). The output of the comparator is negative or '0' (because 60 V is less than 62.5 V) so the third binary digit is set to a 0. The fourth clock cycle similarly results in the fourth digit being a '1' (60 V is greater than 56.25 V, the DAC output for '1001' followed by zeros). The result of this would be in the binary form 1001. This is also called *bit-weighting conversion*, and is similar to a binary search. The analogue value is rounded to the nearest binary value below,

meaning this converter type is mid-rise. Because the approximations are successive (not simultaneous), the conversion takes one clock-cycle for each bit of resolution desired. The clock frequency must be equal to the sampling frequency multiplied by the number of bits of resolution desired. For example, to sample audio at 44.1 kHz with 32 bit resolution, a clock frequency of over 1.4 MHz would be required. ADCs of this type have good resolutions and quite wide ranges. They are more complex than some other designs.

- A **ramp-compare ADC** produces a saw-tooth signal that ramps up or down then quickly returns to zero. When the ramp starts, a timer starts counting. When the ramp voltage matches the input, a comparator fires, and the timer's value is recorded. Timed ramp converters require the least number of transistors. The ramp time is sensitive to temperature because the circuit generating the ramp is often just some simple oscillator. There are two solutions: use a clocked counter driving a DAC and then use the comparator to preserve the counter's value, or calibrate the timed ramp. A special advantage of the ramp-compare system is that comparing a second signal just requires another comparator, and another register to store the voltage value. A very simple (non-linear) ramp-converter can be implemented with a microcontroller and one resistor and capacitor. Vice versa, a filled capacitor can be taken from an integrator, time-to-amplitude converter, phase detector, sample and hold circuit, or peak and hold circuit and discharged. This has the advantage that a slow comparator cannot be disturbed by fast input changes.

- The **Wilkinson ADC** was designed by D. H. Wilkinson in 1950. The Wilkinson ADC is based on the comparison of an input voltage with that produced by a charging capacitor. The capacitor is allowed to charge until its voltage is equal to the amplitude of the input pulse. (A comparator determines when this condition has been reached.) Then, the capacitor is allowed to discharge linearly, which produces a ramp voltage. At the point when the capacitor begins to discharge, a gate pulse is initiated. The gate pulse remains on until the capacitor is completely discharged. Thus the duration of the gate pulse is directly proportional to the amplitude of the input pulse. This gate pulse operates a linear gate which receives pulses from a high-frequency oscillator clock. While the gate is open, a discrete number of clock pulses pass through the linear gate and are counted by the address register. The time the linear gate is open is proportional to the amplitude of the input pulse, thus the number of clock pulses recorded in the address register is proportional also. Alternatively, the charging of the capacitor could be monitored, rather than the discharge.

- An **integrating ADC** (also **dual-slope** or **multi-slope** ADC) applies the unknown input voltage to the input of an integrator and allows the voltage to ramp for a fixed time period (the run-up period). Then a known reference voltage of opposite polarity is applied to the integrator and is allowed to ramp until the integrator output returns to zero (the run-down period). The input voltage is computed as a function of the reference voltage, the constant run-up time period, and the

measured run-down time period. The run-down time measurement is usually made in units of the converter's clock, so longer integration times allow for higher resolutions. Likewise, the speed of the converter can be improved by sacrificing resolution. Converters of this type (or variations on the concept) are used in most digital voltmeters for their linearity and flexibility.

- A **delta-encoded ADC** or Counter-ramp has an up-down counter that feeds a digital to analog converter (DAC). The input signal and the DAC both go to a comparator. The comparator controls the counter. The circuit uses negative feedback from the comparator to adjust the counter until the DAC's output is close enough to the input signal. The number is read from the counter. Delta converters have very wide ranges, and high resolution, but the conversion time is dependent on the input signal level, though it will always have a guaranteed worst-case. Delta converters are often very good choices to read real-world signals. Most signals from physical systems do not change abruptly. Some converters combine the delta and successive approximation approaches; this works especially well when high frequencies are known to be small in magnitude.

- A **pipeline ADC** (also called **subranging quantizer**) uses two or more steps of subranging. First, a coarse conversion is done. In a second step, the difference to the input signal is determined with a digital to analog converter (DAC). This difference is then converted finer, and the results are combined in a last step. This can be considered a refinement of the successive approximation ADC wherein the feedback reference signal consists of the interim conversion of a whole range of bits (for example, four bits) rather than just the next-most-significant bit. By combining the merits of the successive approximation and flash ADCs this type is fast, has a high resolution, and only requires a small die size.

- A **Sigma-Delta ADC** (also known as a Delta-Sigma ADC) oversamples the desired signal by a large factor and filters the desired signal band. Generally, a smaller number of bits than required are converted using a Flash ADC after the filter. The resulting signal, along with the error generated by the discrete levels of the Flash, is fed back and subtracted from the input to the filter. This negative feedback has the effect of noise shaping the error due to the Flash so that it does not appear in the desired signal frequencies. A digital filter (decimation filter) follows the ADC which reduces the sampling rate, filters off unwanted noise signal and increases the resolution of the output (sigma-delta modulation, also called delta-sigma modulation).

- A **Time-interleaved ADC** uses M parallel ADCs where each ADC sample data every M:th cycle of the effective sample clock. The result is that the sample rate is increased M times compared to what each individual ADC can manage. In practice, the individual differences between the M ADCs degrade the overall performance reducing the SFDR. However, technologies exist to correct for these time-interleaving mismatch errors.

- An **ADC with intermediate FM stage** first uses a voltage-to-frequency converter to converts the desired signal into an oscillating signal with a frequency proportional to the voltage of the desired signal, and then uses a frequency counter to convert that frequency into a digital count proportional to the desired signal voltage. Longer integration times allow for higher resolutions. Likewise, the speed of the converter can be improved by sacrificing resolution. The two parts of the ADC may be widely separated, with the frequency signal passed through a opto-isolator or transmitted wirelessly. Some such ADCs use sine wave or square wave frequency modulation; others use pulse-frequency modulation. Such ADCs were once the most popular way to show a digital display of the status of a remote analog sensor.

There can be other ADCs that use a combination of electronics and other technologies:

- A **Time-stretch analog-to-digital converter (TS-ADC)** digitizes a very wide bandwidth analog signal, that cannot be digitized by a conventional electronic ADC, by time-stretching the signal prior to digitization. It commonly uses a photonic preprocessor frontend to time-stretch the signal, which effectively slows the signal down in time and compresses its bandwidth. As a result, an electronic backend ADC, that would have been too slow to capture the original signal, can now capture this slowed down signal. For continuous capture of the signal, the frontend also divides the signal into multiple segments in addition to time-stretching. Each segment is individually digitized by a separate electronic ADC. Finally, a digital signal processor rearranges the samples and removes any distortions added by the frontend to yield the binary data that is the digital representation of the original analog signal.

Commercial analog-to-digital converters

These are usually integrated circuits.

Most converters sample with 6 to 24 bits of resolution, and produce fewer than 1 megasample per second. Thermal noise generated by passive components such as resistors masks the measurement when higher resolution is desired. For audio applications and in room temperatures, such noise is usually a little less than 1 μV (microvolt) of white noise. If the Most Significant Bit corresponds to a standard 2 volts of output signal, this translates to a noise-limited performance that is less than 20~21 bits, and obviates the need for any dithering. Mega- and gigasample per second converters are available, though (Feb 2002). Megasample converters are required in digital video cameras, video capture cards, and TV tuner cards to convert full-speed analog video to digital video files. Commercial converters usually have ±0.5 to ±1.5 LSB error in their output.

In many cases the most expensive part of an integrated circuit is the pins, because they make the package larger, and each pin has to be connected to the integrated circuit's silicon. To save pins, it is common for slow ADCs to send their data one bit at a time

over a serial interface to the computer, with the next bit coming out when a clock signal changes state, say from zero to 5 V. This saves quite a few pins on the ADC package, and in many cases, does not make the overall design any more complex (even microprocessors which use memory-mapped I/O only need a few bits of a port to implement a serial bus to an ADC).

Commercial ADCs often have several inputs that feed the same converter, usually through an analog multiplexer. Different models of ADC may include sample and hold circuits, instrumentation amplifiers or differential inputs, where the quantity measured is the difference between two voltages.

Applications

Application to music recording

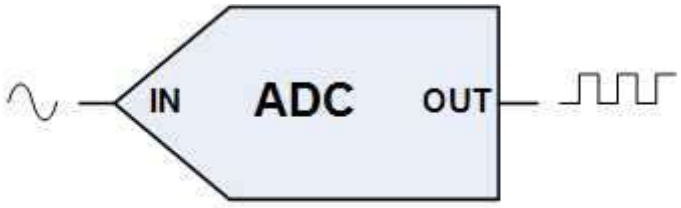
ADCs are integral to current music reproduction technology. Since much music production is done on computers, when an analog recording is used, an ADC is needed to create the PCM data stream that goes onto a compact disc or digital music file.

The current crop of AD converters utilized in music can sample at rates up to 192 kilohertz. High bandwidth headroom allows the use of cheaper or faster anti-aliasing filters of less severe filtering slopes. The proponents of oversampling assert that such shallower anti-aliasing filters produce less deleterious effects on sound quality, exactly because of their gentler slopes. Others prefer entirely filterless AD conversion, arguing that aliasing is less detrimental to sound perception than pre-conversion brickwall filtering. Considerable literature exists on these matters, but commercial considerations often play a significant role. Most high-profile recording studios record in 24-bit/192-176.4 kHz PCM or in DSD formats, and then downsample or decimate the signal for Red-Book CD production (44.1 kHz or at 48 kHz for commonly used for radio/TV broadcast applications).

Digital Signal Processing

AD converters are used virtually everywhere where an analog signal has to be processed, stored, or transported in digital form. Fast video ADCs are used, for example, in TV tuner cards. Slow on-chip 8, 10, 12, or 16 bit ADCs are common in microcontrollers. Very fast ADCs are needed in digital oscilloscopes, and are crucial for new applications like software defined radio.

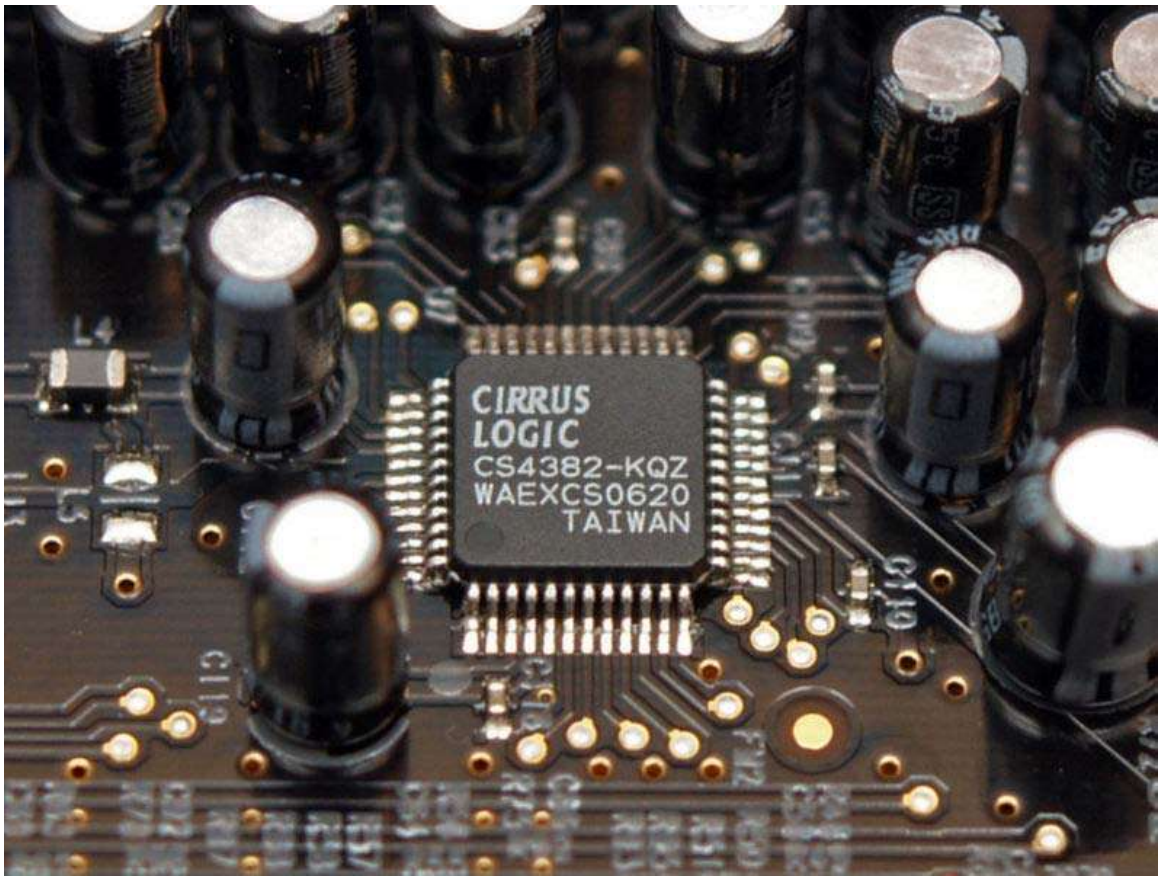
Electrical Symbol



ELECTRICAL SYMBOL FOR ANALOG TO DIGITAL CONVERTER (ADC)

Chapter 15

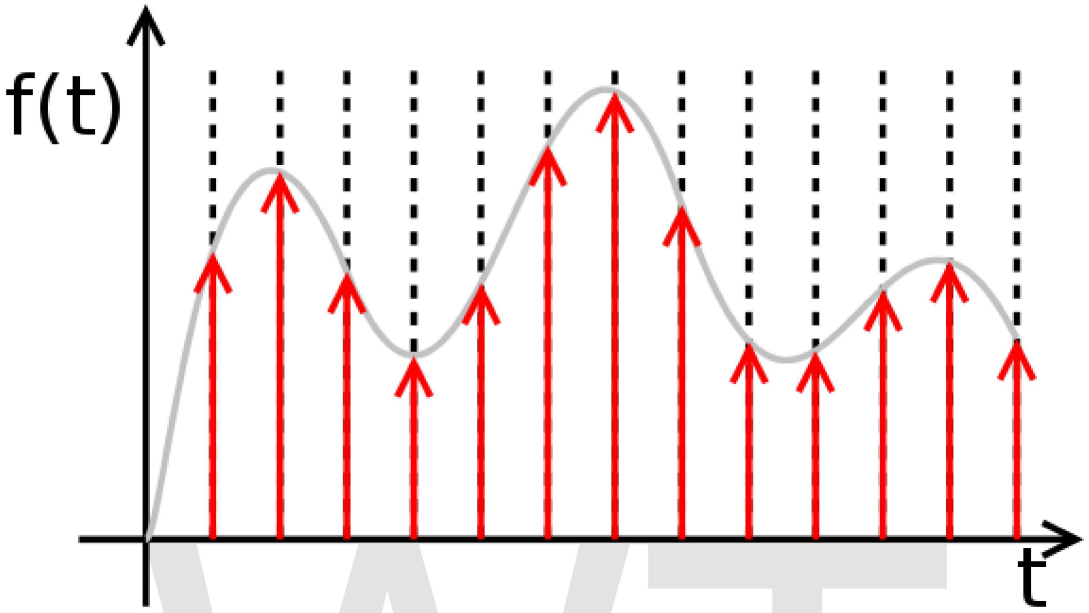
Digital-to-Analog Converter



8-channel digital-to-analog converter Cirrus Logic CS4382 as used in a soundcard

In electronics, a **digital-to-analog converter** (**DAC** or **D-to-A**) is a device that converts a digital (usually binary) code to an analog signal (current, voltage, or electric charge). An analog-to-digital converter (ADC) performs the reverse operation.

Basic ideal operation



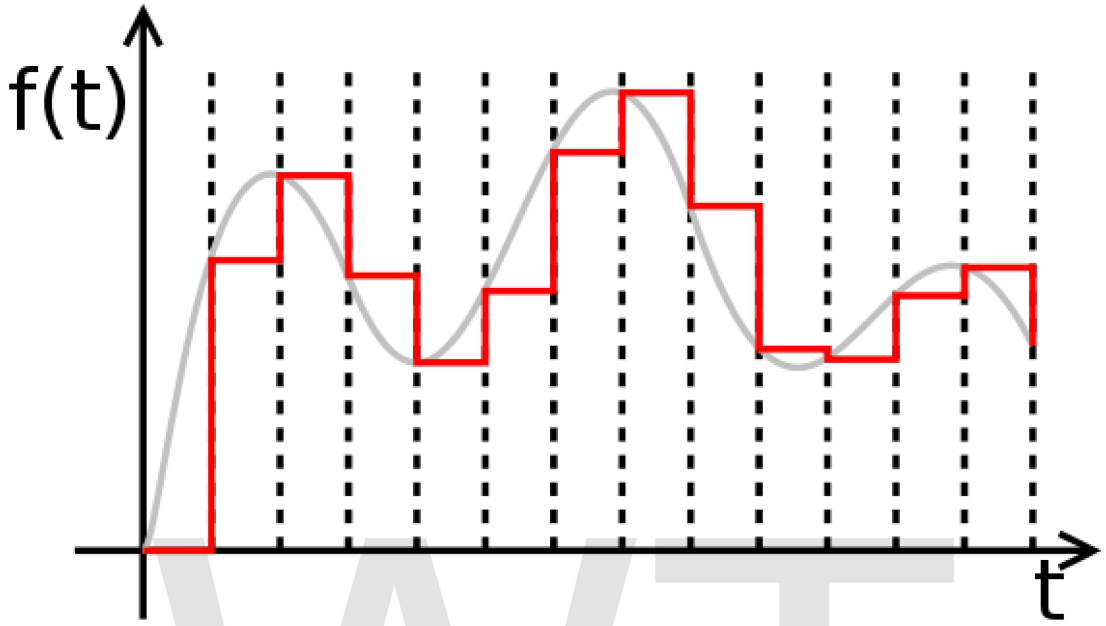
Ideally sampled signal

A DAC converts an abstract finite-precision number (usually a fixed-point binary number) into a concrete physical quantity (e.g., a voltage or a pressure). In particular, DACs are often used to convert finite-precision time series data to a continually varying physical signal.

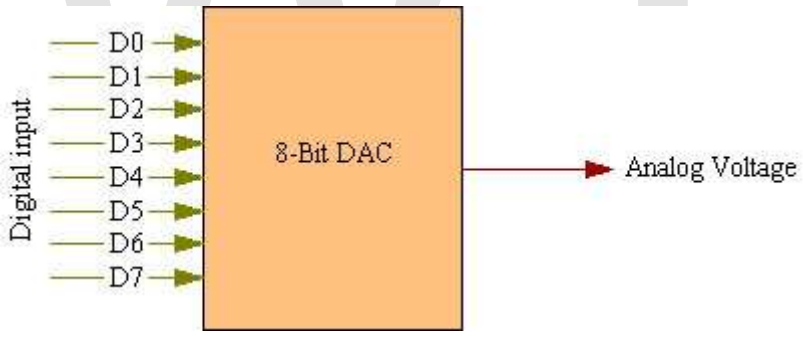
A typical DAC converts the abstract numbers into a concrete sequence of impulses that are then processed by a reconstruction filter using some form of interpolation to fill in data between the impulses. Other DAC methods (e.g., methods based on Delta-sigma modulation) produce a pulse-density modulated signal that can then be filtered in a similar way to produce a smoothly varying signal.

By the Nyquist–Shannon sampling theorem, sampled data can be reconstructed perfectly provided that its bandwidth meets certain requirements (e.g., a baseband signal with bandwidth less than the Nyquist frequency; BUT requires an infinite number of samples. The finite number used in real life cause other problems especially with the D/A reconstruction of the original signal. However, even with an ideal reconstruction filter, digital sampling introduces quantization error that makes perfect reconstruction practically impossible. Increasing the digital resolution (i.e., increasing the number of bits used in each sample) or introducing sampling dither can reduce this error.

Practical operation



Piecewise constant output of a conventional practical DAC.



A simplified functional diagram of an 8-bit DAC

Instead of impulses, usually the sequence of numbers update the analogue voltage at uniform sampling intervals.

These numbers are written to the DAC, typically with a clock signal that causes each number to be latched in sequence, at which time the DAC output voltage changes rapidly from the previous value to the value represented by the currently latched number. The effect of this is that the output voltage is *held* in time at the current value until the next input number is latched resulting in a piecewise constant or 'staircase' shaped output. This is equivalent to a zero-order hold operation and has an effect on the frequency response of the reconstructed signal.

The fact that DACs output a sequence of piecewise constant values (known as zero-order hold in sample data textbooks) or rectangular pulses causes multiple harmonics above the Nyquist frequency. Usually, these are removed with a low pass filter acting as a reconstruction filter in applications that require it.

Applications

Audio

Top-loading CD player and external digital-to-analog converter

Most modern audio signals are stored in digital form (for example MP3s and CDs) and in order to be heard through speakers they must be converted into an analog signal. DACs are therefore found in CD players, digital music players, and PC sound cards.

Specialist standalone DACs can also be found in high-end hi-fi systems. These normally take the digital output of a compatible CD player or dedicated transport and convert the signal into an analog line-level output that can then be fed into an amplifier to drive speakers.

Similar digital-to-analog converters can be found in digital speakers such as USB speakers, and in sound cards.

VOIP (Voice over IP) Phone, Data transmission over the Internet is done digitally so in order for voice to be transmitted it must be converted to digital using an Analog-to-Digital Converter and be converted into analog again using a DAC so the voice it can be heard on the other end.

Video

Video signals from a digital source, such as a computer, must be converted to analog form if they are to be displayed on an analog monitor. As of 2007, analog inputs are more commonly used than digital, but this may change as flat panel displays with DVI and/or HDMI connections become more widespread. A video DAC is, however, incorporated in any digital video player with analog outputs. The DAC is usually integrated with some memory (RAM), which contains conversion tables for gamma correction, contrast and brightness, to make a device called a RAMDAC.

A device that is distantly related to the DAC is the digitally controlled potentiometer, used to control an analog signal digitally.

Mechanical

An unusual application of digital-to-analog conversion was the whiffletree electromechanical digital-to-analog convertor linkage in the IBM Selectric typewriter.

DAC types

The most common types of electronic DACs are:

- The pulse-width modulator, the simplest DAC type. A stable current or voltage is switched into a low-pass analog filter with a duration determined by the digital input code. This technique is often used for electric motor speed control, but has many other applications as well.
- Oversampling DACs or interpolating DACs such as the delta-sigma DAC, use a pulse density conversion technique. The oversampling technique allows for the use of a lower resolution DAC internally. A simple 1-bit DAC is often chosen because the oversampled result is inherently linear. The DAC is driven with a pulse-density modulated signal, created with the use of a low-pass filter, step nonlinearity (the actual 1-bit DAC), and negative feedback loop, in a technique called delta-sigma modulation. This results in an effective high-pass filter acting on the quantization (signal processing) noise, thus steering this noise out of the low frequencies of interest into the megahertz frequencies of little interest, which is called noise shaping. The quantization noise at these high frequencies is removed or greatly attenuated by use of an analog low-pass filter at the output (sometimes a simple RC low-pass circuit is sufficient). Most very high resolution

DACs (greater than 16 bits) are of this type due to its high linearity and low cost. Higher oversampling rates can relax the specifications of the output low-pass filter and enable further suppression of quantization noise. Speeds of greater than 100 thousand samples per second (for example, 192 kHz) and resolutions of 24 bits are attainable with delta-sigma DACs. A short comparison with pulse-width modulation shows that a 1-bit DAC with a simple first-order integrator would have to run at 3 THz (which is physically unrealizable) to achieve 24 meaningful bits of resolution, requiring a higher-order low-pass filter in the noise-shaping loop. A single integrator is a low-pass filter with a frequency response inversely proportional to frequency and using one such integrator in the noise-shaping loop is a first order delta-sigma modulator. Multiple higher order topologies (such as MASH) are used to achieve higher degrees of noise-shaping with a stable topology.

- The binary-weighted DAC, which contains one resistor or current source for each bit of the DAC connected to a summing point. These precise voltages or currents sum to the correct output value. This is one of the fastest conversion methods but suffers from poor accuracy because of the high precision required for each individual voltage or current. Such high-precision resistors and current sources are expensive, so this type of converter is usually limited to 8-bit resolution or less.
- The R-2R ladder DAC which is a binary-weighted DAC that uses a repeating cascaded structure of resistor values R and 2R. This improves the precision due to the relative ease of producing equal valued-matched resistors (or current sources). However, wide converters perform slowly due to increasingly large RC-constants for each added R-2R link.
- The thermometer-coded DAC, which contains an equal resistor or current-source segment for each possible value of DAC output. An 8-bit thermometer DAC would have 255 segments, and a 16-bit thermometer DAC would have 65,535 segments. This is perhaps the fastest and highest precision DAC architecture but at the expense of high cost. Conversion speeds of >1 billion samples per second have been reached with this type of DAC.
- Hybrid DACs, which use a combination of the above techniques in a single converter. Most DAC integrated circuits are of this type due to the difficulty of getting low cost, high speed and high precision in one device.
    - The segmented DAC, which combines the thermometer-coded principle for the most significant bits and the binary-weighted principle for the least significant bits. In this way, a compromise is obtained between precision (by the use of the thermometer-coded principle) and number of resistors or current sources (by the use of the binary-weighted principle). The full binary-weighted design means 0% segmentation, the full thermometer-coded design means 100% segmentation.

DAC performance

DACs are very important to system performance. The most important characteristics of these devices are:

- **Resolution**: This is the number of possible output levels the DAC is designed to reproduce. This is usually stated as the number of bits it uses, which is the base two logarithm of the number of levels. For instance a 1 bit DAC is designed to reproduce 2 (2^1) levels while an 8 bit DAC is designed for 256 (2^8) levels. Resolution is related to the **effective number of bits** (ENOB) which is a measurement of the actual resolution attained by the DAC.
- **Maximum sampling frequency**: This is a measurement of the maximum speed at which the DACs circuitry can operate and still produce the correct output. As stated in the Nyquist–Shannon sampling theorem, a signal must be sampled at over twice the frequency of the desired signal. For instance, to reproduce signals in all the audible spectrum, which includes frequencies of up to 20 kHz, it is necessary to use DACs that operate at over 40 kHz. The CD standard samples audio at 44.1 kHz, thus DACs of this frequency are often used. A common frequency in cheap computer sound cards is 48 kHz — many work at only this frequency, offering the use of other sample rates only through (often poor) internal resampling.
- **Monotonicity**: This refers to the ability of a DAC's analog output to move only in the direction that the digital input moves (i.e., if the input increases, the output doesn't dip before asserting the correct output.) This characteristic is very important for DACs used as a low frequency signal source or as a digitally programmable trim element.
- **THD+N**: This is a measurement of the distortion and noise introduced to the signal by the DAC. It is expressed as a percentage of the total power of unwanted harmonic distortion and noise that accompany the desired signal. This is a very important DAC characteristic for dynamic and small signal DAC applications.
- **Dynamic range**: This is a measurement of the difference between the largest and smallest signals the DAC can reproduce expressed in decibels. This is usually related to DAC resolution and noise floor.

Other measurements, such as phase distortion and jitter, can also be very important for some applications.

Bits	Color limit	Frequency	Examples
10	1.024 colors	54 MHz	
12		54 MHz	Sony NS-575p
12	4.096 colors	108 MHz	
12		150 MHz	NeoDigits Helios X5000
12		216 MHz	Philips BDP9000 (Blu-ray)
12		297 MHz	Toshiba HD-XE1
12		216 MHz	Samsung BD-P1200 (Blu-ray)
14	16.384 colors	108 MHz	Pioneer Elite, Black Finish, DV79AVI
14		216 MHz	Marantz DV9600, Sony DVPNS9100ES
16		149 MHz	NeuNeo HVD108

DAC figures of merit

- Static performance:
  - Differential nonlinearity (DNL) shows how much two adjacent code analog values deviate from the ideal 1LSB step
  - Integral nonlinearity (INL) shows how much the DAC transfer characteristic deviates from an ideal one. That is, the ideal characteristic is usually a straight line; INL shows how much the actual voltage at a given code value differs from that line, in LSBs (1LSB steps).
  - Gain
  - Offset
  - Noise is ultimately limited by the thermal noise generated by passive components such as resistors. For audio applications and in room temperatures, such noise is usually a little less than 1 μV (microvolt) of white noise. This limits performance to less than 20~21 bits even in 24-bit DACs.
- Frequency domain performance
  - Spurious-free dynamic range (SFDR) indicates in dB the ratio between the powers of the converted main signal and the greatest undesired spur
  - Signal to noise and distortion ratio (SNDR) indicates in dB the ratio between the powers of the converted main signal and the sum of the noise and the generated harmonic spurs
  - i-th harmonic distortion (HDi) indicates the power of the i-th harmonic of the converted main signal
  - Total harmonic distortion (THD) is the sum of the powers of all HDi
  - If the maximum DNL error is less than 1 LSB, then D/A converter is guaranteed to be monotonic.

However, many monotonic converters may have a maximum DNL greater than 1 LSB.

- Time domain performance:
  - Glitch energy
  - Response uncertainty
  - Time nonlinearity (TNL)