Digital Electronics

Layton Stanford

First Edition, 2012

ISBN 978-81-323-2846-9

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

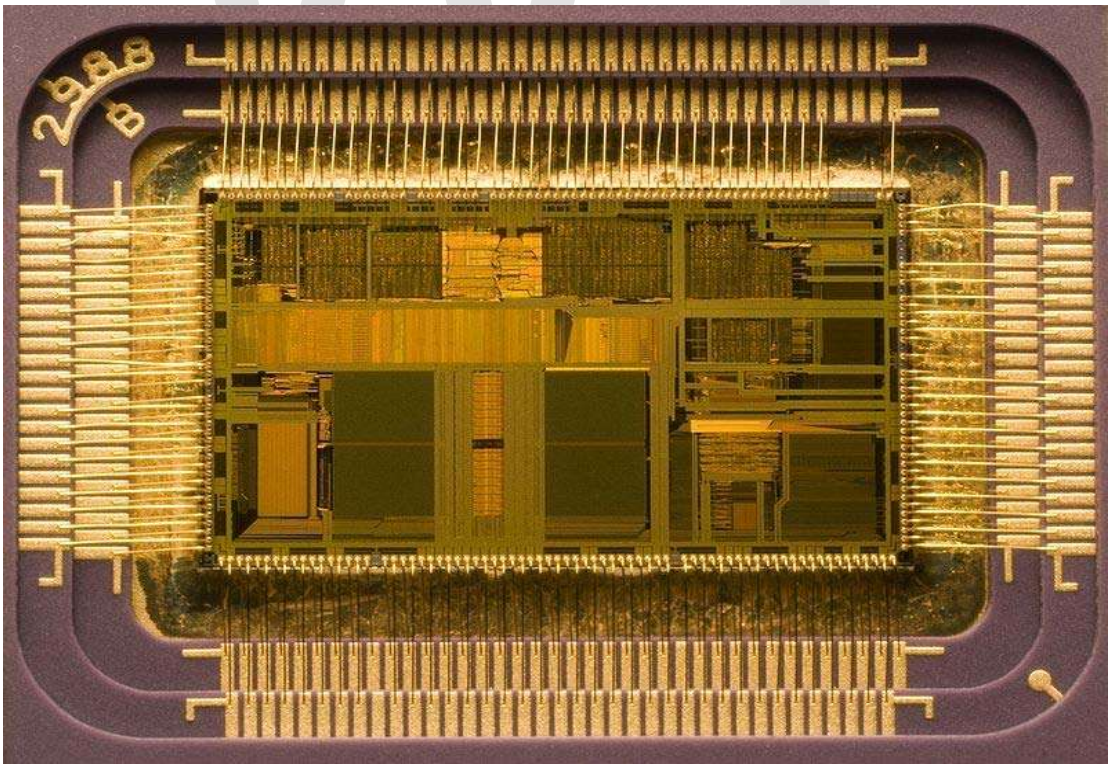
Table of Contents

Chapter- 1

Digital Electronics

Digital electronics represent signals by discrete bands of analog levels, rather than by a continuous range. All levels within a band represent the same signal state. Relatively small changes to the analog signal levels due to manufacturing tolerance, signal attenuation or parasitic noise do not leave the discrete envelope, and as a result are ignored by signal state sensing circuitry.

In most cases the number of these states is two, and they are represented by two voltage bands: one near zero volts and a higher level near the supply voltage, corresponding to the "false" ("0") and "true" ("1") values of the boolean domain respectively.



Intel 80486DX2

Hitachi J100

Digital techniques are useful because it is easier to get an electronic device to switch into one of a number of known states than to accurately reproduce a continuous range of values.

Digital electronic circuits are usually made from large assemblies of logic gates, simple electronic representations of Boolean logic functions.

Advantages

One advantage of digital circuits when compared to analog circuits is that signals represented digitally can be transmitted without degradation due to noise. For example, a continuous audio signal, transmitted as a sequence of 1s and 0s, can be reconstructed without error provided the noise picked up in transmission is not enough to prevent identification of the 1s and 0s. An hour of music can be stored on a compact disc as about 6 billion binary digits.

In a digital system, a more precise representation of a signal can be obtained by using more binary digits to represent it. While this requires more digital circuits to process the signals, each digit is handled by the same kind of hardware. In an analog system, additional resolution requires fundamental improvements in the linearity and noise characteristics of each step of the signal chain.

Computer-controlled digital systems can be controlled by software, allowing new functions to be added without changing hardware. Often this can be done outside of the factory by updating the product's software. So, the product's design errors can be corrected after the product is in a customer's hands.

Information storage can be easier in digital systems than in analog ones. The noise-immunity of digital systems permits data to be stored and retrieved without degradation. In an analog system, noise from aging and wear degrade the information stored. In a digital system, as long as the total noise is below a certain level, the information can be recovered perfectly.

Disadvantages

In some cases, digital circuits use more energy than analog circuits to accomplish the same tasks, thus producing more heat. In portable or battery-powered systems this can limit use of digital systems.

For example, battery-powered cellular telephones often use a low-power analog front-end to amplify and tune in the radio signals from the base station. However, a base station has grid power and can use power-hungry, but very flexible software radios. Such base stations can be easily reprogrammed to process the signals used in new cellular standards.

Digital circuits are sometimes more expensive, especially in small quantities.

Most useful digital systems must translate from continuous analog signals to discrete digital signals. This causes quantization errors. Quantization error can be reduced if the system stores enough digital data to represent the signal to the desired degree of fidelity. The Nyquist-Shannon sampling theorem provides an important guideline as to how much digital data is needed to accurately portray a given analog signal.

In some systems, if a single piece of digital data is lost or misinterpreted, the meaning of large blocks of related data can completely change. Because of the cliff effect, it can be difficult for users to tell if a particular system is right on the edge of failure, or if it can tolerate much more noise before failing.

Digital fragility can be reduced by designing a digital system for robustness. For example, a parity bit or other error management method can be inserted into the signal path. These schemes help the system detect errors, and then either correct the errors, or at least ask for a new copy of the data. In a state-machine, the state transition logic can be designed to catch unused states and trigger a reset sequence or other error recovery routine.

Digital memory and transmission systems can use techniques such as error detection and correction to use additional data to correct any errors in transmission and storage.

On the other hand, some techniques used in digital systems make those systems more vulnerable to single-bit errors. These techniques are acceptable when the underlying bits are reliable enough that such errors are highly unlikely. A single-bit error in audio data stored directly as linear pulse code modulation (such as on a CD-ROM) causes, at worst, a single click. Instead, many people use audio compression to save storage space and download time, even though a single-bit error may corrupt the entire song.

Analog issues in digital circuits

Digital circuits are made from analog components. The design must assure that the analog nature of the components doesn't dominate the desired digital behavior. Digital systems must manage noise and timing margins, parasitic inductances and capacitances, and filter power connections.

Bad designs have intermittent problems such as "glitches", vanishingly-fast pulses that may trigger some logic but not others, "runt pulses" that do not reach valid "threshold" voltages, or unexpected ("undecoded") combinations of logic states.

Additionally, where clocked digital systems interface to analogue systems or systems that are driven from a different clock, the digital system can be subject to metastability where a change to the input violates the set-up time for a digital input latch. This situation will self-resolve, but will take a random time, and while it persists can result in invalid signals being propagated within the digital system for a short time.

Since digital circuits are made from analog components, digital circuits calculate more slowly than low-precision analog circuits that use a similar amount of space and power. However, the digital circuit will calculate more repeatably, because of its high noise immunity. On the other hand, in the high-precision domain (for example, where 14 or more bits of precision are needed), analog circuits require much more power and area than digital equivalents.

Construction

A digital circuit is often constructed from small electronic circuits called logic gates that can be used to create combinational logic. Each logic gate represents a function of boolean logic. A logic gate is an arrangement of electrically controlled switches, better known as transistors.

Each logic symbol is represented by a different shape. The actual set of shapes was introduced in 1984 under IEEE\ANSI standard 91-1984. "The logic symbol given under this standard are being increasingly used now and have even started appearing in the literature published by manufacturers of digital integrated circuits."

The output of a logic gate is an electrical flow or voltage, that can, in turn, control more logic gates.

Logic gates often use the fewest number of transistors in order to reduce their size, power consumption and cost, and increase their reliability.

Integrated circuits are the least expensive way to make logic gates in large volumes. Integrated circuits are usually designed by engineers using electronic design automation software.

Another form of digital circuit is constructed from lookup tables, (many sold as "programmable logic devices", though other kinds of PLDs exist). Lookup tables can perform the same functions as machines based on logic gates, but can be easily reprogrammed without changing the wiring. This means that a designer can often repair design errors without changing the arrangement of wires. Therefore, in small volume products, programmable logic devices are often the preferred solution. They are usually designed by engineers using electronic design automation software.

When the volumes are medium to large, and the logic can be slow, or involves complex algorithms or sequences, often a small microcontroller is programmed to make an embedded system. These are usually programmed by software engineers.

When only one digital circuit is needed, and its design is totally customized, as for a factory production line controller, the conventional solution is a programmable logic controller, or PLC. These are usually programmed by electricians, using ladder logic.

Structure of digital systems

Engineers use many methods to minimize logic functions, in order to reduce the circuit's complexity. When the complexity is less, the circuit also has fewer errors and less electronics, and is therefore less expensive.

The most widely used simplification is a minimization algorithm like the Espresso heuristic logic minimizer within a CAD system, although historically, binary decision

diagrams, an automated Quine–McCluskey algorithm, truth tables, Karnaugh Maps, and Boolean algebra have been used.

Representations are crucial to an engineer's design of digital circuits. Some analysis methods only work with particular representations.

The classical way to represent a digital circuit is with an equivalent set of logic gates. Another way, often with the least electronics, is to construct an equivalent system of electronic switches (usually transistors). One of the easiest ways is to simply have a memory containing a truth table. The inputs are fed into the address of the memory, and the data outputs of the memory become the outputs.

For automated analysis, these representations have digital file formats that can be processed by computer programs. Most digital engineers are very careful to select computer programs ("tools") with compatible file formats.

To choose representations, engineers consider types of digital systems. Most digital systems divide into "combinational systems" and "sequential systems." A combinational system always presents the same output when given the same inputs. It is basically a representation of a set of logic functions, as already discussed.

A sequential system is a combinational system with some of the outputs fed back as inputs. This makes the digital machine perform a "sequence" of operations. The simplest sequential system is probably a flip flop, a mechanism that represents a binary digit or "bit".

Sequential systems are often designed as state machines. In this way, engineers can design a system's gross behavior, and even test it in a simulation, without considering all the details of the logic functions.

Sequential systems divide into two further subcategories. "Synchronous" sequential systems change state all at once, when a "clock" signal changes state. "Asynchronous" sequential systems propagate changes whenever inputs change. Synchronous sequential systems are made of well-characterized asynchronous circuits such as flip-flops, that change only when the clock changes, and which have carefully designed timing margins.

The usual way to implement a synchronous sequential state machine is to divide it into a piece of combinational logic and a set of flip flops called a "state register." Each time a clock signal ticks, the state register captures the feedback generated from the previous state of the combinational logic, and feeds it back as an unchanging input to the combinational part of the state machine. The fastest rate of the clock is set by the most time-consuming logic calculation in the combinational logic.

The state register is just a representation of a binary number. If the states in the state machine are numbered (easy to arrange), the logic function is some combinational logic that produces the number of the next state.

In comparison, asynchronous systems are very hard to design because all possible states, in all possible timings must be considered. The usual method is to construct a table of the minimum and maximum time that each such state can exist, and then adjust the circuit to minimize the number of such states, and force the circuit to periodically wait for all of its parts to enter a compatible state (this is called "self-resynchronization"). Without such careful design, it is easy to accidentally produce asynchronous logic that is "unstable", that is, real electronics will have unpredictable results because of the cumulative delays caused by small variations in the values of the electronic components. Certain circuits (such as the synchronizer flip-flops, switch debouncers, arbiters, and the like which allow external unsynchronized signals to enter synchronous logic circuits) are inherently asynchronous in their design and must be analyzed as such.

As of 2005, almost all digital machines are synchronous designs because it is much easier to create and verify a synchronous design—the software currently used to simulate digital machines does not yet handle asynchronous designs. However, asynchronous logic is thought to be superior, if it can be made to work, because its speed is not constrained by an arbitrary clock; instead, it runs at the maximum speed of its logic gates. Building an asynchronous circuit using faster parts makes the circuit faster.

Many digital systems are data flow machines. These are usually designed using synchronous register transfer logic, using hardware description languages such as VHDL or Verilog.

In register transfer logic, binary numbers are stored in groups of flip flops called registers. The outputs of each register are a bundle of wires called a "bus" that carries that number to other calculations. A calculation is simply a piece of combinational logic. Each calculation also has an output bus, and these may be connected to the inputs of several registers. Sometimes a register will have a multiplexer on its input, so that it can store a number from any one of several buses. Alternatively, the outputs of several items may be connected to a bus through buffers that can turn off the output of all of the devices except one. A sequential state machine controls when each register accepts new data from its input.

In the 1980s, some researchers discovered that almost all synchronous register-transfer machines could be converted to asynchronous designs by using first-in-first-out synchronization logic. In this scheme, the digital machine is characterized as a set of data flows. In each step of the flow, an asynchronous "synchronization circuit" determines when the outputs of that step are valid, and presents a signal that says, "grab the data" to the stages that use that stage's inputs. It turns out that just a few relatively simple synchronization circuits are needed.

The most general-purpose register-transfer logic machine is a computer. This is basically an automatic binary abacus. The control unit of a computer is usually designed as a microprogram run by a microsequencer. A microprogram is much like a player-piano roll. Each table entry or "word" of the microprogram commands the state of every bit that controls the computer. The sequencer then counts, and the count addresses the memory or

combinational logic machine that contains the microprogram. The bits from the microprogram control the arithmetic logic unit, memory and other parts of the computer, including the microsequencer itself.

In this way, the complex task of designing the controls of a computer is reduced to a simpler task of programming a collection of much simpler logic machines.

Computer architecture is a specialized engineering activity that tries to arrange the registers, calculation logic, buses and other parts of the computer in the best way for some purpose. Computer architects have applied large amounts of ingenuity to computer design to reduce the cost and increase the speed and immunity to programming errors of computers. An increasingly common goal is to reduce the power used in a battery-powered computer system, such as a cell-phone. Many computer architects serve an extended apprenticeship as microprogrammers.

"Specialized computers" are usually a conventional computer with a special-purpose microprogram.

Automated design tools

To save costly engineering effort, much of the effort of designing large logic machines has been automated. The computer programs are called "electronic design automation tools" or just "EDA."

Simple truth table-style descriptions of logic are often optimized with EDA that automatically produces reduced systems of logic gates or smaller lookup tables that still produce the desired outputs. The most common example of this kind of software is the Espresso heuristic logic minimizer.

Most practical algorithms for optimizing large logic systems use algebraic manipulations or binary decision diagrams, and there are promising experiments with genetic algorithms and annealing optimizations.

To automate costly engineering processes, some EDA can take state tables that describe state machines and automatically produce a truth table or a function table for the combinational logic of a state machine. The state table is a piece of text that lists each state, together with the conditions controlling the transitions between them and the belonging output signals.

It is common for the function tables of such computer-generated state-machines to be optimized with logic-minimization software such as Minilog.

Often, real logic systems are designed as a series of sub-projects, which are combined using a "tool flow." The tool flow is usually a "script," a simplified computer language that can invoke the software design tools in the right order.

Tool flows for large logic systems such as microprocessors can be thousands of commands long, and combine the work of hundreds of engineers.

Writing and debugging tool flows is an established engineering specialty in companies that produce digital designs. The tool flow usually terminates in a detailed computer file or set of files that describe how to physically construct the logic. Often it consists of instructions to draw the transistors and wires on an integrated circuit or a printed circuit board.

Parts of tool flows are "debugged" by verifying the outputs of simulated logic against expected inputs. The test tools take computer files with sets of inputs and outputs, and highlight discrepancies between the simulated behavior and the expected behavior.

Once the input data is believed correct, the design itself must still be verified for correctness. Some tool flows verify designs by first producing a design, and then scanning the design to produce compatible input data for the tool flow. If the scanned data matches the input data, then the tool flow has probably not introduced errors.

The functional verification data are usually called "test vectors." The functional test vectors may be preserved and used in the factory to test that newly constructed logic works correctly. However, functional test patterns don't discover common fabrication faults. Production tests are often designed by software tools called "test pattern generators". These generate test vectors by examining the structure of the logic and systematically generating tests for particular faults. This way the fault coverage can closely approach 100%, provided the design is properly made testable.

Once a design exists, and is verified and testable, it often needs to be processed to be manufacturable as well. Modern integrated circuits have features smaller than the wavelength of the light used to expose the photoresist. Manufacturability software adds interference patterns to the exposure masks to eliminate open-circuits, and enhance the masks' resolution and contrast.

Design for testability

"There are several reasons for testing a logic circuit. When the circuit is first developed, it is necessary to verify that the design circuit meets the required functional and timing specifications. When multiple copies of a correctly designed circuit are being manufactured, it is essential to test each copy to ensure that the manufacturing process has not introduced any flaws.

A large logic machine (say, with more than a hundred logical variables) can have an astronomical number of possible states. Obviously, in the factory, testing every state is impractical if testing each state takes a microsecond, and there are more states than the number of microseconds since the universe began. Unfortunately, this ridiculous-sounding case is typical.

Fortunately, large logic machines are almost always designed as assemblies of smaller logic machines. To save time, the smaller sub-machines are isolated by permanently-installed "design for test" circuitry, and are tested independently.

One common test scheme known as "scan design" moves test bits serially (one after another) from external test equipment through one or more serial shift registers known as "scan chains". Serial scans have only one or two wires to carry the data, and minimize the physical size and expense of the infrequently-used test logic.

After all the test data bits are in place, the design is reconfigured to be in "normal mode" and one or more clock pulses are applied, to test for faults (e.g. stuck-at low or stuck-at high) and capture the test result into flip-flops and/or latches in the scan shift register(s). Finally, the result of the test is shifted out to the block boundary and compared against the predicted "good machine" result.

In a board-test environment, serial to parallel testing has been formalized with a standard called "JTAG" (named after the "Joint Test Action Group" that proposed it).

Another common testing scheme provides a test mode that forces some part of the logic machine to enter a "test cycle." The test cycle usually exercises large independent parts of the machine.

Trade-offs

Several numbers determine the practicality of a system of digital logic. Engineers explored numerous electronic devices to get an ideal combination of fanout, speed, low cost and reliability.

The cost of a logic gate is crucial. In the 1930s, the earliest digital logic systems were constructed from telephone relays because these were inexpensive and relatively reliable. After that, engineers always used the cheapest available electronic switches that could still fulfill the requirements.

The earliest integrated circuits were a happy accident. They were constructed not to save money, but to save weight, and permit the Apollo Guidance Computer to control an inertial guidance system for a spacecraft. The first integrated circuit logic gates cost nearly $50 (in 1960 dollars, when an engineer earned $10,000/year). To everyone's surprise, by the time the circuits were mass-produced, they had become the least-expensive method of constructing digital logic. Improvements in this technology have driven all subsequent improvements in cost.

With the rise of integrated circuits, reducing the absolute number of chips used represented another way to save costs. The goal of a designer is not just to make the simplest circuit, but to keep the component count down. Sometimes this results in slightly more complicated designs with respect to the underlying digital logic but nevertheless reduces the number of components, board size, and even power consumption.

For example, in some logic families, NAND gates are the simplest digital gate to build. All other logical operations can be implemented by NAND gates. If a circuit already required a single NAND gate, and a single chip normally carried four NAND gates, then the remaining gates could be used to implement other logical operations like logical and. This could eliminate the need for a separate chip containing those different types of gates.

The "reliability" of a logic gate describes its mean time between failure (MTBF). Digital machines often have millions of logic gates. Also, most digital machines are "optimized" to reduce their cost. The result is that often, the failure of a single logic gate will cause a digital machine to stop working.

Digital machines first became useful when the MTBF for a switch got above a few hundred hours. Even so, many of these machines had complex, well-rehearsed repair procedures, and would be nonfunctional for hours because a tube burned-out, or a moth got stuck in a relay. Modern transistorized integrated circuit logic gates have MTBFs greater than 82 billion hours (8.2×10^{10}) hours, and need them because they have so many logic gates.

Fanout describes how many logic inputs can be controlled by a single logic output without exceeding the current ratings of the gate. The minimum practical fanout is about five. Modern electronic logic using CMOS transistors for switches have fanouts near fifty, and can sometimes go much higher.

The "switching speed" describes how many times per second an inverter (an electronic representation of a "logical not" function) can change from true to false and back. Faster logic can accomplish more operations in less time. Digital logic first became useful when switching speeds got above fifty hertz, because that was faster than a team of humans operating mechanical calculators. Modern electronic digital logic routinely switches at five gigahertz (5×10^9 hertz), and some laboratory systems switch at more than a terahertz (1×10^{12} hertz).

Logic families

Design started with relays. Relay logic was relatively inexpensive and reliable, but slow. Occasionally a mechanical failure would occur. Fanouts were typically about ten, limited by the resistance of the coils and arcing on the contacts from high voltages.

Later, vacuum tubes were used. These were very fast, but generated heat, and were unreliable because the filaments would burn out. Fanouts were typically five to seven, limited by the heating from the tubes' current. In the 1950s, special "computer tubes" were developed with filaments that omitted volatile elements like silicon. These ran for hundreds of thousands of hours.

The first semiconductor logic family was resistor-transistor logic. This was a thousand times more reliable than tubes, ran cooler, and used less power, but had a very low fan-in of three. Diode-transistor logic improved the fanout up to about seven, and reduced the

power. Some DTL designs used two power-supplies with alternating layers of NPN and PNP transistors to increase the fanout.

Transistor transistor logic (TTL) was a great improvement over these. In early devices, fanout improved to ten, and later variations reliably achieved twenty. TTL was also fast, with some variations achieving switching times as low as twenty nanoseconds. TTL is still used in some designs.

Emitter coupled logic is very fast but uses a lot of power. It was extensively used for high-performance computers made up of many medium-scale components (such as the Illiac IV).

Modern integrated circuits mostly use variations of CMOS, which is fast, small and low-power. Fanouts of forty or more are possible, with some speed penalty.

Non-electronic logic

It is possible to construct non-electronic digital mechanisms. In principle, any technology capable of representing discrete states and representing logic operations could be used to build mechanical logic. MIT students Erlyne Gee, Edward Hardebeck, Danny Hillis (co-author of The Connection Machine), Margaret Minsky and brothers Barry and Brian Silverman, built two working computers from Tinker toys, string, a brick, and a sharpened pencil. The Tinkertoy computer is in the Boston Museum of Science.

Hydraulic, pneumatic and mechanical versions of logic gates exist and are used in situations where electricity cannot be used. The first two types are considered under the heading of fluidics. One application of fluidic logic is in military hardware that is likely to be exposed to a nuclear electromagnetic pulse (nuclear EMP, or NEMP) that would destroy electrical circuits.

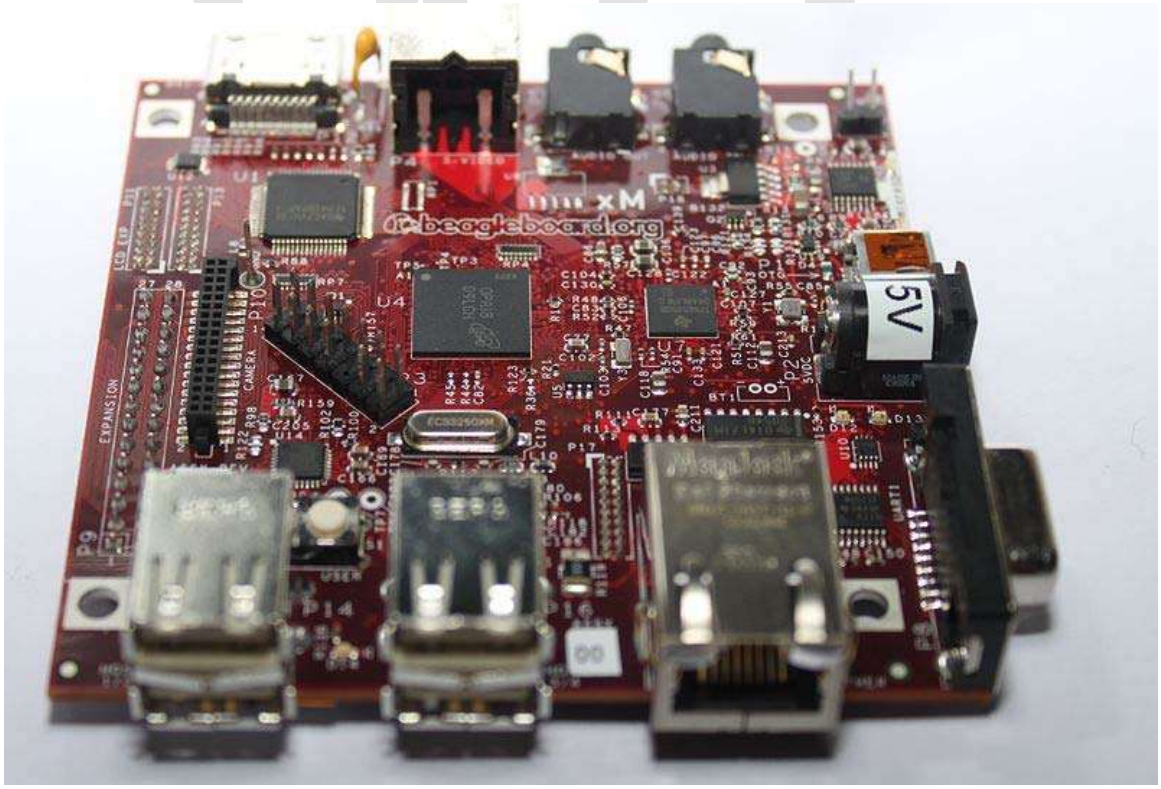
Mechanical logic is frequently used in inexpensive controllers, such as those in washing machines. Famously, the first computer design, by Charles Babbage, was designed to use mechanical logic. Mechanical logic might also be used in very small computers that could be built by nanotechnology.

Another example is that if two particular enzymes are required to prevent the construction of a particular protein, this is the equivalent of a biological "NAND" gate.

Recent developments

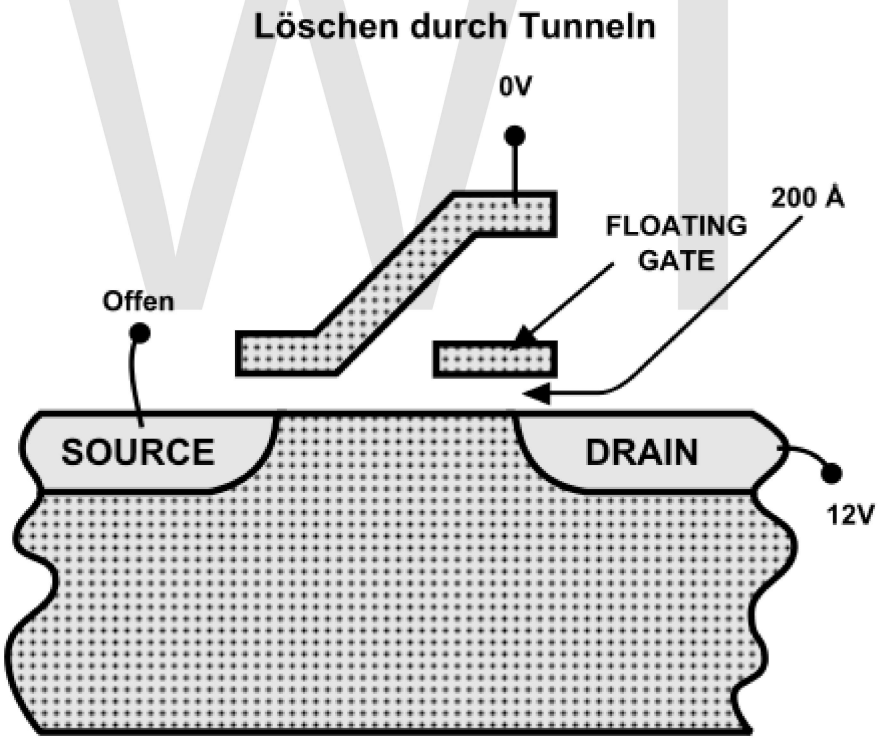
The discovery of superconductivity has enabled the development of Rapid Single Flux Quantum (RSFQ) circuit technology, which uses Josephson junctions instead of transistors. Most recently, attempts are being made to construct purely optical computing systems capable of processing digital information using nonlinear optical elements.

Almacenaje sucesivo Fifo



BeagleBoard xM.

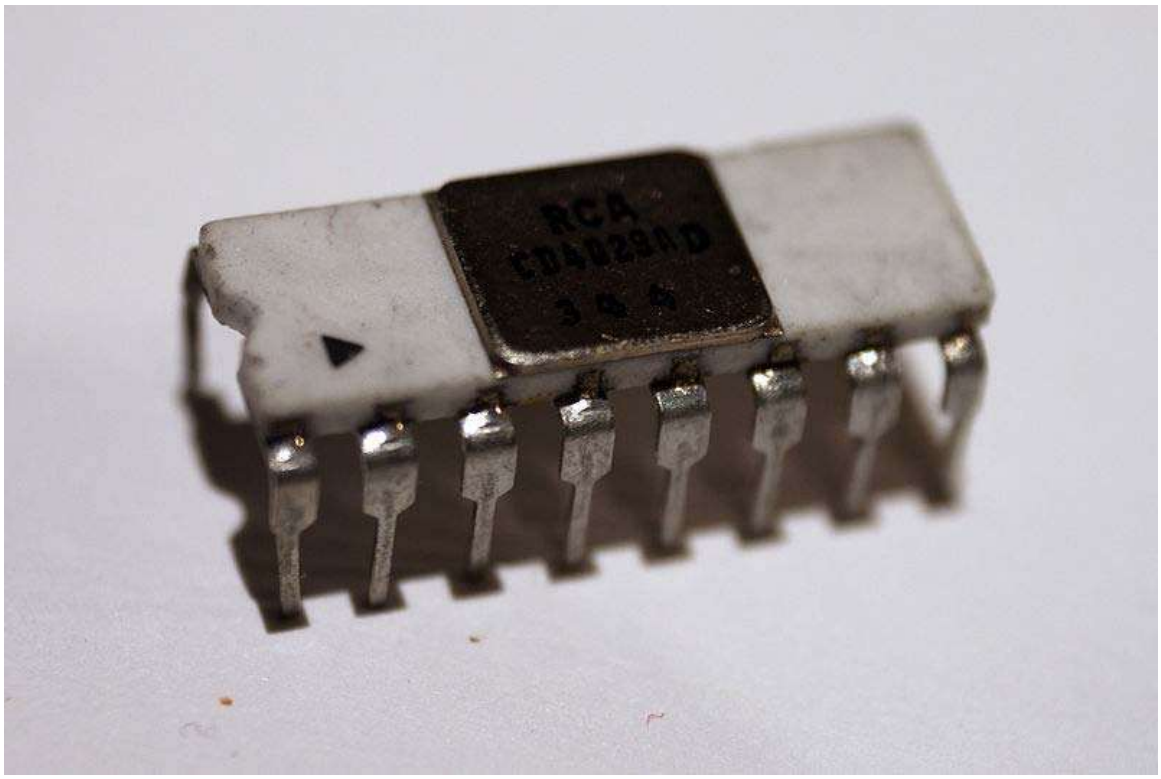
Ferrite core Nova



Flash erase german

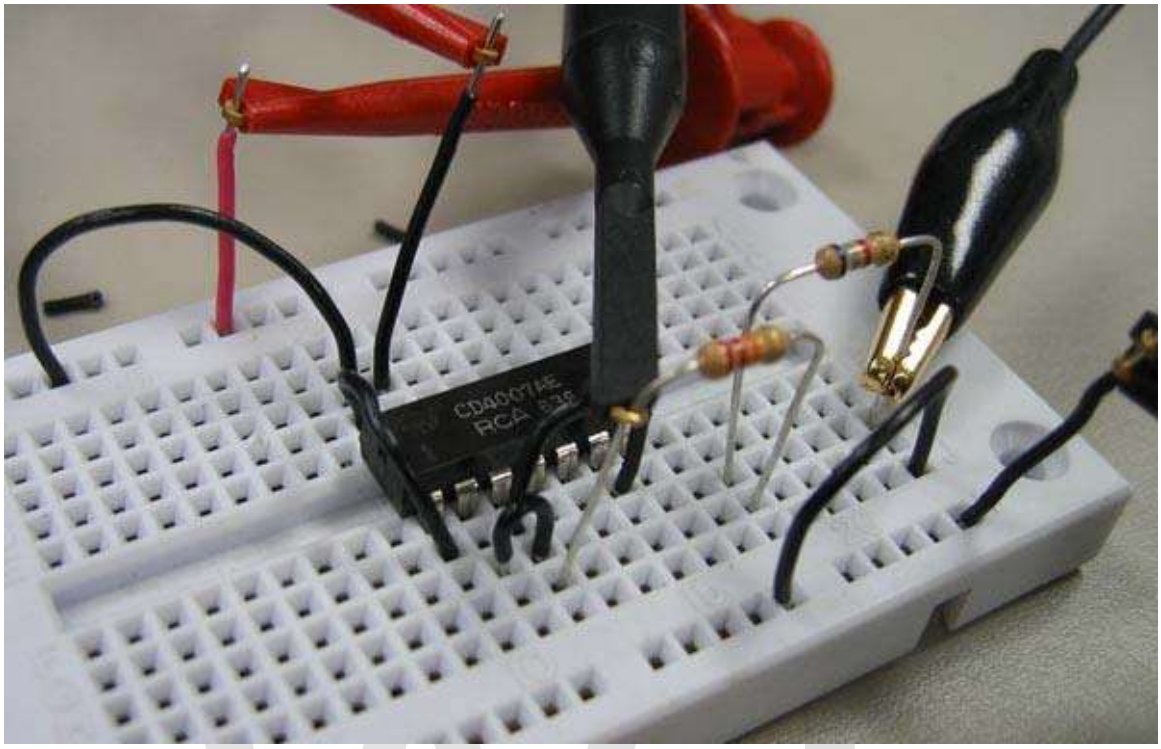
Chapter- 2

4000 Series



A very early CD4029 counter IC, manufactured by RCA.

The **4000 series** (originally: CD4000 series, but now includes HEF4000 series, etc.) is a family of industry standard integrated circuits which implement a variety of logic functions using Complementary Metal–Oxide–Semiconductor technology. They were introduced by RCA as *CD4000 COS/MOS* in 1968, as a lower power and more versatile alternative to the 7400 series of TTL logic chips. Almost all IC manufacturers active during the era fabricated chips from this series. RCA sometimes advertised the line as COSMOS, standing for COmplementary Symmetry Metal-Oxide Semiconductor. The naming system followed the RCA convention of CA for analog, CD for digital, but did not relate to the Texas Instruments SN7400 series numbering scheme.



The CD4007 on a breadboard

For many years, the 4000 series devices could not operate at speeds as fast as the popular 7400 TTL chips, but had the advantage of much lower power consumption, the ability to operate over a much wider range of supply voltages (3 V to 15 V), and simpler circuit design due to the vastly increased fanout. However their slower speed (initially only capable of about 1 MHz operation, compared with TTL's 10 MHz) limited their applications to static or slow speed designs. Later, new fabrication technology largely overcame the speed problems, while retaining backward compatibility with most circuit designs. Although all semiconductors can be damaged by electrostatic discharge, the high impedance of CMOS inputs makes them more susceptible than bipolar transistor-based, TTL, devices. Eventually, the advantages of CMOS (especially the later series such as 74HC) edged out the older TTL chips, but at the same time ever increasing LSI techniques edged out the modular chip approach to design. The 4000 series is still widely available, but perhaps less important than it was two decades ago.

The series was extended in the late 1970s and 1980s to include new types which implemented new or more greatly integrated functions, or were better versions of existing chips in the 4000 series. Most of these newer chips were given 45xx and 45xxx designations, but are usually still regarded by engineers as part of the 4000 series.

In the 1990s, some manufacturers (e.g. Texas Instruments) ported the 4000 series to their newer HCMOS technology with devices such as the 74HCT4060 providing equivalent functionality to a 4060 IC but with greater speed.

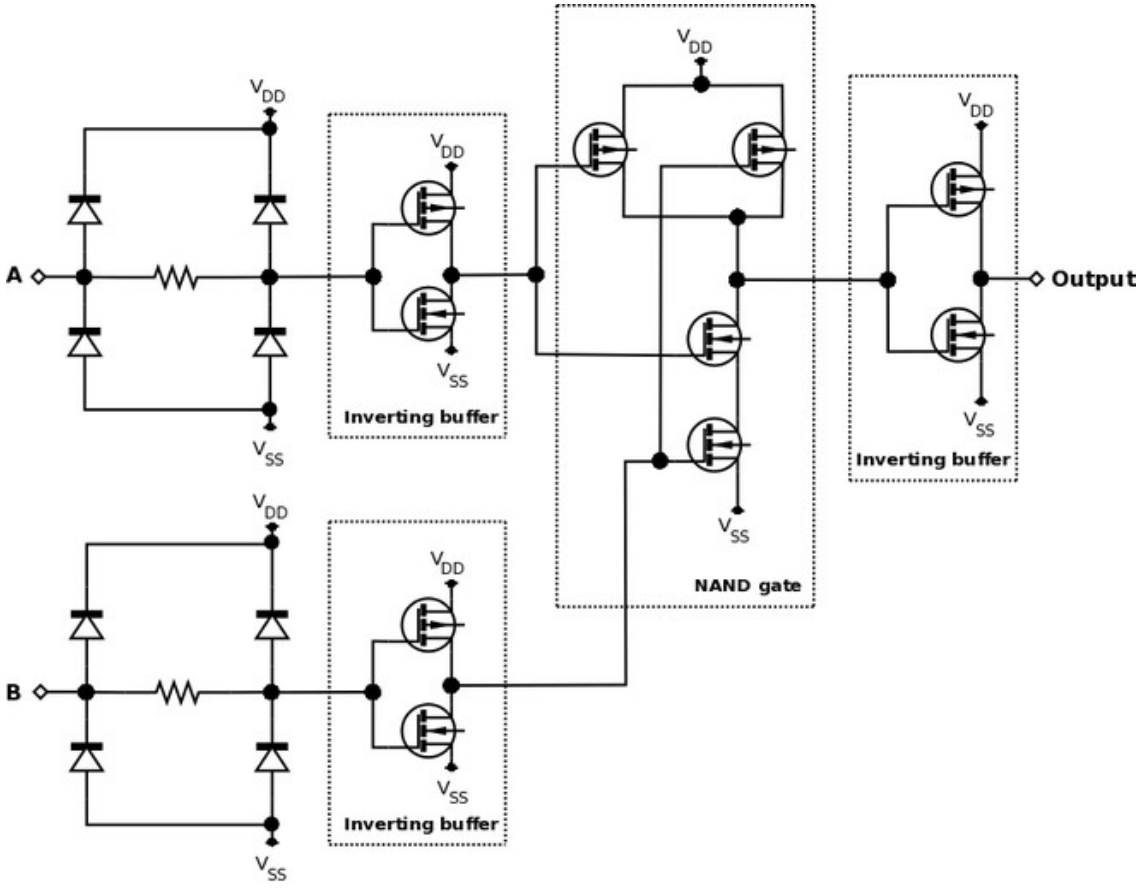
The 4000 series integrated circuits have been used in space satellites for many decades. As of 2000, people building satellites still use 4000 series integrated circuits.

Design considerations

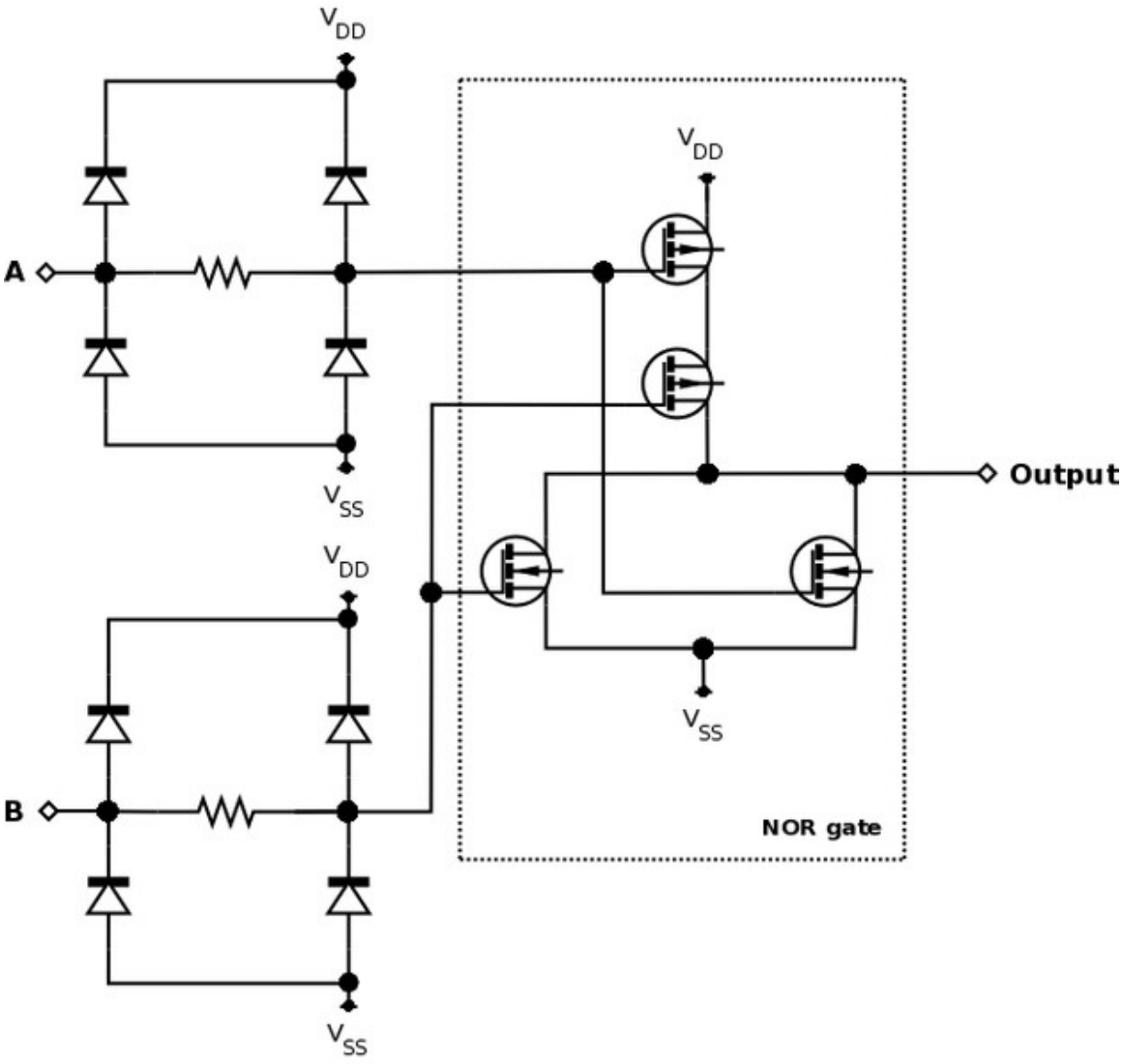
The original 4000 series was available in two versions: The A series was unbuffered, while the B series featured buffered inputs and outputs (in the form of additional, simple logic gates). The buffered outputs are able to source or sink more current than the unbuffered outputs, thus eliminating the need for discrete switching transistors in some designs. The buffered versions also have faster output switching times, as the signal rise time of the buffered output stage is faster than that of an unbuffered device, however the overall propagation delay through the buffered versions is higher due to the additional circuitry. The buffered devices are less susceptible to output oscillation with slow-changing inputs so designers have to weigh up the pros and cons of using buffered or unbuffered parts according to the nature of the circuit in which the devices are being used. The additional input and output gates on the buffered parts also make them marginally less susceptible to damage by electrostatic discharge (ESD).

Although the original designation for unbuffered and buffered parts was the addition of an 'A' or 'B' suffix to the part code (eg: 4000A = unbuffered, 4000B = buffered), some manufacturers (eg: Texas Instruments) later changed to using UB (unbuffered) and B (buffered) suffixes (eg: 4000UB and 4000B).

The diagrams below show the construction differences between a simple buffered and unbuffered CMOS NOR logic gate. Note that the logic gate at the core of the buffered part is actually a NAND gate, but the overall function of the complete circuit is a NOR gate due to the logic inversions performed by the buffers. (A negated NAND becomes a NOR as defined by De Morgan's laws in Boolean Algebra.) The clamping diodes on the inputs are to offer some protection against ESD.



Buffered CMOS two input NOR gate



Unbuffered CMOS two input NOR gate

The 4000 series permits the use of "cookbook design" at least for slow design, where standard circuit elements can be created, shared, and connected to other circuits with few, if any, connection difficulties. This greatly speeds the design of new hardware by reusing standard approaches to circuit design. In contrast, TTL circuits, while similarly modular, often require much more careful interfacing, since the limited fanout (and fan-in) require that the loading of each output be carefully considered. Some later TTL families, like 74LS reduce this problem with fanouts of 20. It is also much easier to prototype LSI designs using the 4000 series and get repeatable and transferable results when moving to the more integrated design.

Some care needs to be taken with the design of circuits using CMOS chips. Many parts offer multiple logic gates in a single package and it is common to not need or use all of them. An engineer who forgets to 'tie off' (connect the unused gate inputs to VSS or VDD

through pull-down or pull-up resistors) may find the chip draws excessive current. The problem is caused by biasing in each gate. With the inputs disconnected, the gate will bias itself into a linear mode where the outputs are partially switched; this leaves the output buffer drawing a great deal of current since it isn't fully on or off, creating a low resistance current path between the power supply rails.

Example common 4000 series chips

- 4000 - Dual 3-Input NOR Gate and Inverter
- 4001 - Quad 2-Input NOR Gate
- 4002 - Dual 4-Input NOR Gate OR Gate
- 4008 - 4-Bit Full Adder
- 4010 - hex non-inverting buffer
- 4011 - Quad 2-Input NAND Gate

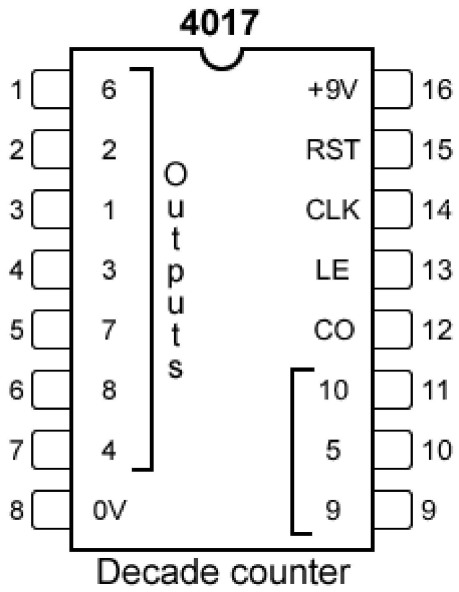
Notable parts

A few parts are notable in the 4000 series because of their level of integration compared to other chips. This list is intentionally incomplete and is meant to provide a sample of the more interesting parts in the series. Devices useful for switching analog signals (such as the 4066, and 4051 to 4053) have continued to enjoy popularity in some audio designs (although non-4000 series chips, often with less distortion, are now available).

4017 decade counter

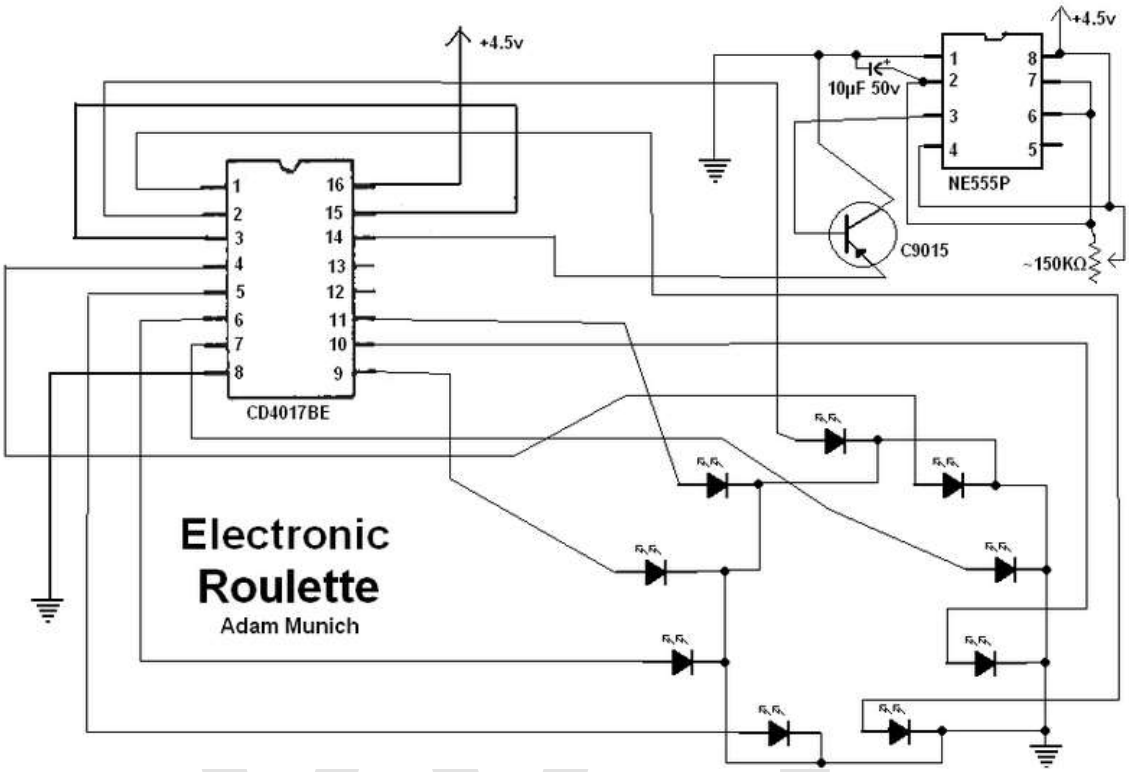
The **4017 IC** is a 16-pin CMOS decade counter from the 4000 series. It takes clock pulses from the clock input, and makes one of the ten outputs come on in sequence each time a clock pulse arrives.

Pinout



Pin number	Name	Purpose
1	6	The 6th sequential output
2	2	The 2nd sequential output
3	1	The 1st sequential output
4	3	The 3rd sequential output
5	7	The 7th sequential output
6	8	The 8th sequential output
7	4	The 4th sequential output
8	0 V, V_{DD}	The connection to the 0 V rail
9	9	The 9th sequential output
10	5	The 5th sequential output
11	10	The 10th sequential output
12	CO	**C**arry **o**ut output - outputs high on counts 0 to 4, outputs low on counts 5 to 9 (thus a transition from low to high occurs when counting from 9 back to 0)
13	LE	**L**atch **e**nable - latches on the current output when high (i.e. the chip counts when LE is low)
14	CLK	**Cl**oc**k** in
15	RST	**Reset** - sets output 1 high and outputs 2 through 10 low, when taken high
16	+9 V, V_{CC}	The connection to the $+V_{CC}$ rail (voltage between +3 V and +15 V)

Example: Electronic Roulette



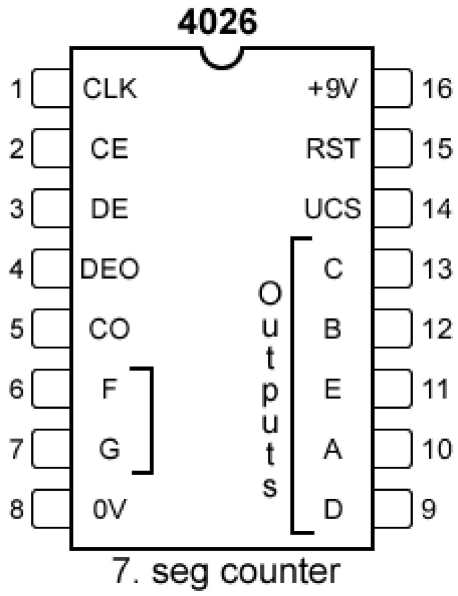
Electronic Roulette circuit diagram.

The circuit diagram on the right shows how to create a game of roulette using the 4017 decade counter and various other electronic parts. The switch stops the roulette, and the resistor adjusts the spin speed.

4026 counter and display decoder

The **4026 IC** is a 16-pin CMOS seven-segment counter from the 4000 series. It counts clock pulses and returns the output in a form which can be displayed on a seven-segment display. This avoids using a binary-coded decimal to seven-segment decoder, but it can only be used to display the (decimal) digits 0-9.

Pinout



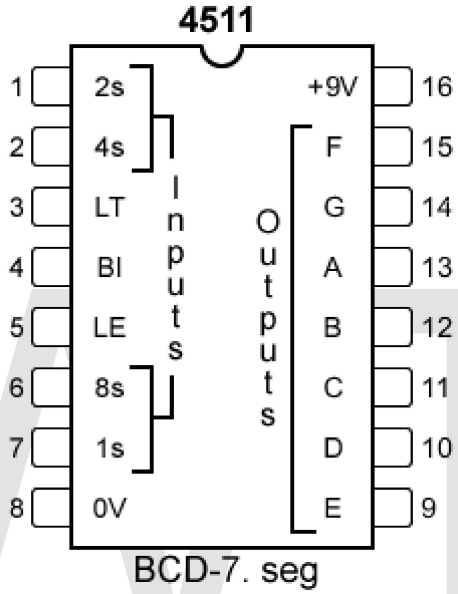
Pin number	Name	Purpose
1	CLK	**C**lock in
2	CI	**C**lock **i**nhibit - when low, clock pulses increment the seven-segment
3	DE	**D**isplay **e**nable - the chip outputs to the seven-segment when this is high (i.e. when it's low, the seven-segment is off) - useful to conserve battery life, for instance
4	DEO	**D**isplay **e**nable **o**ut - for chaining 4026s
5	CO	**C**arry **o**ut output - Is high when changing from 9 to 0. It provides an output at 1/10 of the clock frequency, to drive the clock input of another 4026 to provide multi-digit counting.
6	F	Output for the seven-segment's **F** input
7	G	Output for the seven-segment's **G** input
8	V_{DD}	The connection to the 0 V rail
9	D	Output for the seven-segment's **D** input
10	A	Output for the seven-segment's **A** input
11	E	Output for the seven-segment's **E** input
12	B	Output for the seven-segment's **B** input
13	C	Output for the seven-segment's **C** input
14	UCS	**U**ngated **C**-**s**egment - an output for the seven-segment's **C** input which isn't affected by the **DE** input. This output is high unless the count is 2, when it goes low.
15	RST	**R**ese**t** - resets all outputs to low when taken high

16	V_{SS}	The connection to the +9 V rail

4511 BCD to seven-segment decoder

The **4511 IC** is a 16-pin CMOS BCD to seven-segment decoder from the 4000 series. It takes the binary-coded decimal from a binary counter and decodes it to drive a seven-segment display.

Pinout

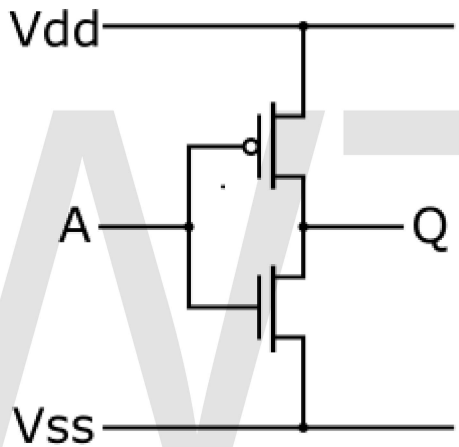


Pin number	Name	Purpose
1	2s	Input for the 2s digit from the binary counter
2	4s	Input for the 4s digit from the binary counter
3	LT	**L**amp **t**est - when low, the chip takes all the segments on the display high (to test connections, etc.)
4	BI	**B**lanking **i**nput - when low, the chip doesn't output to the display - to conserve battery life, for instance
5	LE	**L**atch **e**nable - latches on the current output when high (i.e. the inputs change the output when LE is low)
6	8s	Input for the 8s digit from the binary counter
7	1s	Input for the 1s digit from the binary counter
8	0 V, V_{DD}	The connection to the 0 V rail
9	E	Output for the seven-segment's **E** input
10	D	Output for the seven-segment's **D** input

11	C	Output for the seven-segment's **C** input
12	B	Output for the seven-segment's **B** input
13	A	Output for the seven-segment's **A** input
14	G	Output for the seven-segment's **G** input
15	F	Output for the seven-segment's **F** input
16	+9 V, V_{CC}	The connection to the +9 V rail

Chapter- 3

CMOS



CMOS inverter (NOT logic gate)

Complementary metal–oxide–semiconductor (**CMOS**) is a technology for constructing integrated circuits. CMOS technology is used in microprocessors, microcontrollers, static RAM, and other digital logic circuits. CMOS technology is also used for several analog circuits such as image sensors, data converters, and highly integrated transceivers for many types of communication. Frank Wanlass patented CMOS in 1967 (US patent 3,356,858).

CMOS is also sometimes referred to as **complementary-symmetry metal–oxide–semiconductor** (or COS-MOS). The words "complementary-symmetry" refer to the fact that the typical digital design style with CMOS uses complementary and symmetrical pairs of p-type and n-type metal oxide semiconductor field effect transistors (MOSFETs) for logic functions.

Two important characteristics of CMOS devices are high noise immunity and low static power consumption. Significant power is only drawn when the transistors in the CMOS device are switching between on and off states. Consequently, CMOS devices do not

produce as much waste heat as other forms of logic, for example transistor-transistor logic (TTL) or NMOS logic. CMOS also allows a high density of logic functions on a chip. It was primarily for this reason that CMOS became the most used technology to be implemented in VLSI chips.

The phrase "metal–oxide–semiconductor" is a reference to the physical structure of certain field-effect transistors, having a metal gate electrode placed on top of an oxide insulator, which in turn is on top of a semiconductor material. Aluminum was once used but now the material is polysilicon. Other metal gates have made a comeback with the advent of high-k dielectric materials in the CMOS process, as announced by IBM and Intel for the 45 nanometer node and beyond.

Technical details

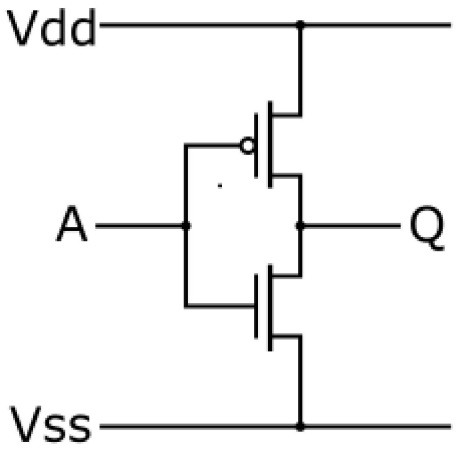
"CMOS" refers to both a particular style of digital circuitry design, and the family of processes used to implement that circuitry on integrated circuits (chips). CMOS circuitry dissipates less power than logic families with resistive loads. Since this advantage has increased and grown more important, CMOS processes and variants have come to dominate, thus the vast majority of modern integrated circuit manufacturing is on CMOS processes. As of 2010, CPUs with the best performance per watt each year have been CMOS static logic since 1976.

CMOS circuits use a combination of p-type and n-type metal–oxide–semiconductor field-effect transistors (MOSFETs) to implement logic gates and other digital circuits found in computers, telecommunications equipment, and signal processing equipment. Although CMOS logic can be implemented with discrete devices (for instance, in an introductory circuits class), typical commercial CMOS products are integrated circuits composed of millions of transistors of both types on a rectangular piece of silicon of between 10 & 400mm^2. These devices are commonly called "chips", although within the industry they are also referred to as "die" (singular) or "dice", or "dies" (plural).

Composition

The main principle behind CMOS circuits that allows them to implement logic gates is the use of p-type and n-type metal–oxide–semiconductor field-effect transistors to create paths to the output from either the voltage source or ground. When a path to output is created from the voltage source, the circuit is said to be pulled up. The other circuit state occurs when a path to output is created from ground and the output pulled down to the ground potential.

Inversion



Static CMOS Inverter

CMOS circuits are constructed in such a way that all PMOS transistors must have either an input from the voltage source or from another PMOS transistor. Similarly, all NMOS transistors must have either an input from ground or from another NMOS transistor. The composition of a PMOS transistor creates low resistance between its source and drain contacts when a low gate voltage is applied and high resistance when a high gate voltage is applied. On the other hand, the composition of an NMOS transistor creates high resistance between source and drain when a low gate voltage is applied and low resistance when a high gate voltage is applied. CMOS accomplishes current reduction by complementing every nMOSFET with a pMOSFET and connecting both gates and both drains together. A high voltage on the gates will cause the nMOSFET to conduct and the pMOSFET not to conduct while a low voltage on the gates causes the reverse. During the switching time as the voltage goes from one state to another, both MOSFETs will conduct briefly. This arrangement greatly reduces power consumption and heat generation.

The image on the right shows what happens when an input is connected to both a PMOS transistor (top of diagram) and an NMOS transistor (bottom of diagram). When the voltage of input A is low, the NMOS transistor's channel is in a high resistance state. This limits the current that can flow from Q to ground. The PMOS transistor's channel is in a low resistance state and much more current can flow from the supply to the output. Because the resistance between the supply voltage and Q is low, the voltage drop between the supply voltage and Q due to a current drawn from Q is small. The output therefore registers a high voltage.

On the other hand, when the voltage of input A is high, the PMOS transistor is in an OFF (high resistance) state so it would limit the current flowing from the positive supply to the output, while the NMOS transistor is in an ON (low resistance) state, allowing the output to drain to ground. Because the resistance between Q and ground is low, the voltage drop

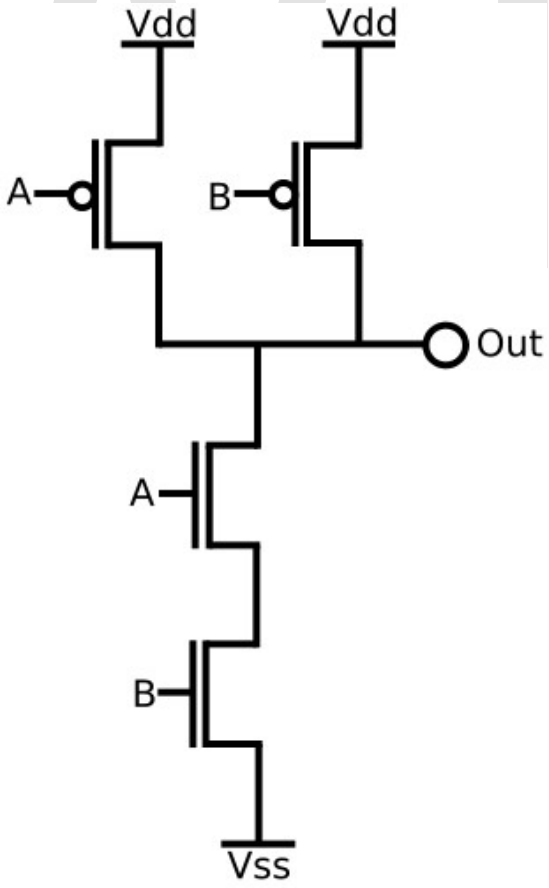
due to a current drawn into Q placing Q above ground is small. This low drop results in the output registering a low voltage.

In short, the outputs of the PMOS and NMOS transistors are complementary such that when the input is low, the output is high, and when the input is high, the output is low. Because of this behavior of input and output, the CMOS circuits' output is the inversion of the input.

Duality

An important characteristic of a CMOS circuit is the duality that exists between its PMOS transistors and NMOS transistors. A CMOS circuit is created to allow a path always to exist from the output to either the power source or ground. To accomplish this, the set of all paths to the voltage source must be the complement of the set of all paths to ground. This can be easily accomplished by defining one in terms of the NOT of the other. Due to the De Morgan's laws based logic, the PMOS transistors in parallel have corresponding NMOS transistors in series while the PMOS transistors in series have corresponding NMOS transistors in parallel.

Logic



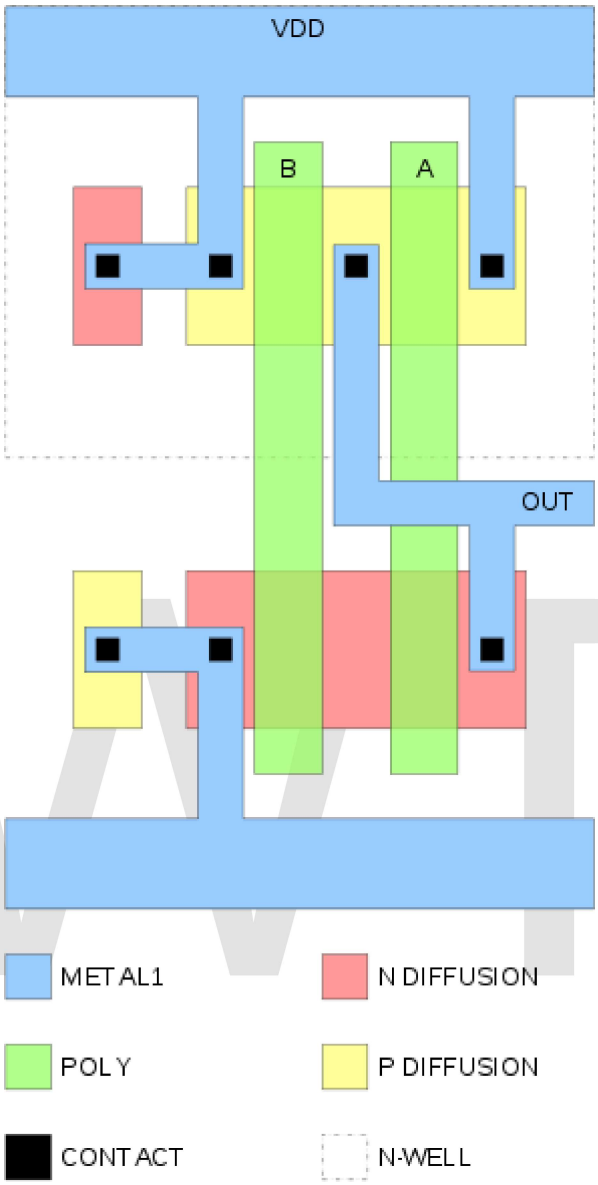
NAND gate in CMOS logic

More complex logic functions such as those involving AND and OR gates require manipulating the paths between gates to represent the logic. When a path consists of two transistors in series, both transistors must have low resistance to the corresponding supply voltage, modeling an AND. When a path consists of two transistors in parallel, either one or both of the transistors must have low resistance to connect the supply voltage to the output, modeling an OR.

Shown on the right is a circuit diagram of a NAND gate in CMOS logic. If both of the A and B inputs are high, then both the NMOS transistors (bottom half of the diagram) will conduct, neither of the PMOS transistors (top half) will conduct, and a conductive path will be established between the output and V_{ss} (ground), bringing the output low. If either of the A or B inputs is low, one of the NMOS transistors will not conduct, one of the PMOS transistors will, and a conductive path will be established between the output and V_{dd} (voltage source), bringing the output high.

An advantage of CMOS over NMOS is that *both* low-to-high and high-to-low output transitions are fast since the pull-up transistors have low resistance when switched on, unlike the load resistors in NMOS logic. In addition, the output signal swings the full voltage between the low and high rails. This strong, more nearly symmetric response also makes CMOS more resistant to noise.

Example: NAND gate in physical layout

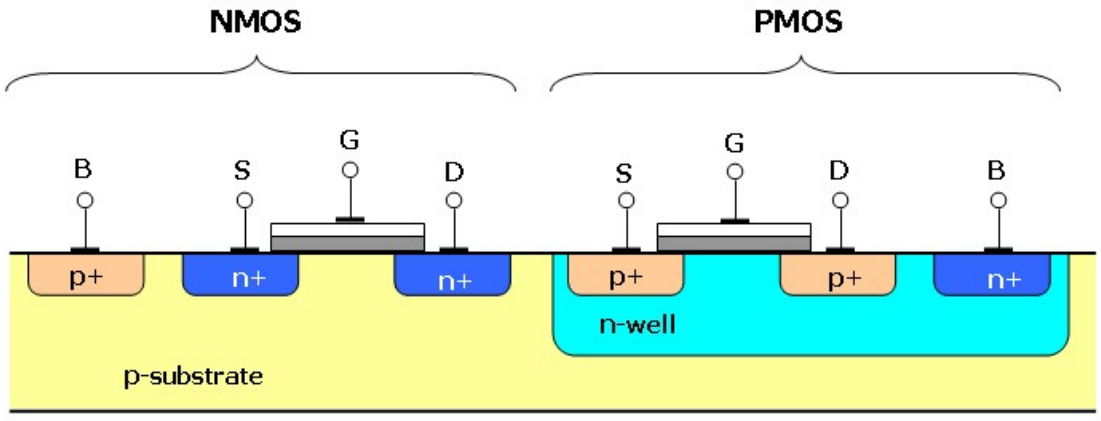


The physical layout of a NAND circuit. The larger regions of N-type diffusion and P-type diffusion are part of the transistors. The two smaller regions on the left are taps to prevent latchup.

This example shows a NAND logic device drawn as a physical representation as it would be manufactured. The physical layout perspective is a "bird's eye view" of a stack of layers. The circuit is constructed on a P-type substrate. The polysilicon, diffusion, and n-well are referred to as "base layers" and are actually inserted into trenches of the P-type substrate. The contacts penetrate an insulating layer between the base layers and the first layer of metal (metal1) making a connection.

The inputs to the NAND (illustrated in green color) are in polysilicon. The CMOS transistors (devices) are formed by the intersection of the polysilicon and diffusion; N diffusion for the N device & P diffusion for the P device (illustrated in salmon and yellow coloring respectively). The output ("out") is connected together in metal (illustrated in cyan coloring). Connections between metal and polysilicon or diffusion are made through contacts (illustrated as black squares). The physical layout example matches the NAND logic circuit given in the previous example.

The N device is manufactured on a P-type substrate while the P device is manufactured in an N-type well (n-well). A P-type substrate "tap" is connected to V_{SS} and an N-type n-well tap is connected to V_{DD} to prevent latchup.



Cross section of two transistors in a CMOS gate, in an N-well CMOS process

Power: switching and leakage

CMOS logic dissipates less power than NMOS logic circuits because CMOS dissipates power only when switching ("dynamic power"). On a typical ASIC in a modern 90 nanometer process, switching the output might take 120 picoseconds, and happen once every ten nanoseconds. NMOS logic dissipates power whenever the output is low ("static power"), because there is a current path from V_{dd} to V_{ss} through the load resistor and the n-type network.

Static CMOS gates are very power efficient because they dissipate nearly zero power when idle. Earlier, the power consumption of CMOS devices was not the major concern while designing chips. Factors like speed and area dominated the design parameters. As the CMOS technology moved below sub-micron levels the power consumption per unit area of the chip has risen tremendously.

Broadly classifying, power dissipation in CMOS circuits occurs because of two components:

1. Static dissipation

1.a. Sub threshold condition when the transistors are off. Both NMOS and PMOS transistors have a gate–source threshold voltage, below which the current (called *sub threshold* current) through the device drops exponentially. Historically, CMOS designs operated at supply voltages much larger than their threshold voltages (V_{dd} might have been 5 V, and V_{th} for both NMOS and PMOS might have been 700 mV). A special type of the CMOS transistor with near zero threshold voltage is the native transistor.

1.b. Tunnelling current through gate oxide. SiO_2 is a very good insulator, but at very small thickness levels electrons can tunnel across the very thin insulation; the probability drops off exponentially with oxide thickness. Tunnelling current becomes very important for transistors below 130nm technology with gate oxides of 20Å or thinner.

1.c. Leakage current through reverse biased diodes. Small reverse leakage currents are formed due to formation of reverse bias between diffusion regions and wells (for e.g., p-type diffusion vs. n-well), wells and substrate (for e.g., n-well vs. p-substrate). In modern process diode leakage is very small compared to sub threshold and tunnelling currents, so these may be neglected during power calculations.

1.d. Contention current in ratioed circuit

2. Dynamic Dissipation

2.a. Charging and discharging of load capacitances. CMOS circuits dissipate power by charging the various load capacitances (mostly gate and wire capacitance, but also drain and some source capacitances) whenever they are switched. In one complete cycle of CMOS logic, current flows from V_{DD} to the load capacitance to charge it and then flows from the charged load capacitance to ground during discharge. Therefore in one complete charge/discharge cycle, a total of $Q=C_L V_{DD}$ is thus transferred from V_{DD} to ground. Multiply by the switching frequency on the load capacitances to get the current used, and multiply by voltage again to get the characteristic switching power dissipated by a CMOS device: $P = CV^2f$.

Since most gates do not operate/switch at every clock cycle, they are often accompanied by a factor α, called the activity factor. Now, the dynamic power dissipation may be re-written as $P = \alpha CV^2f$.

A clock in a system has an activity factor α=1, since it rises and falls every cycle. Most data has an activity factor of 0.5. If correct load capacitance is estimated on a node together with its activity factor, the dynamic power dissipation at that node can be calculated effectively.

2.b. Short circuit power dissipation Since there is a finite rise/fall time for both pMOS and nMOS, during transition, say, from off to on, both the transistors will be on for a small period of time in which current will find a path directly from V_{DD} to ground, hence

creating a short circuit current. Short circuit power dissipation increases with rise and fall time of the transistors.

An additional form of power consumption became significant in the 1990s as wires on chip became narrower and the long wires became more resistive. CMOS gates at the end of those resistive wires see slow input transitions. During the middle of these transitions, both the NMOS and PMOS logic networks are partially conductive, and current flows directly from V_{dd} to V_{SS}. The power thus used is called *crowbar* power. Careful design which avoids weakly driven long skinny wires has ameliorated this effect, and crowbar power is nearly always substantially smaller than switching power.

To speed up designs, manufacturers have switched to constructions that have lower voltage thresholds but because of this a modern NMOS transistor with a V_{th} of 200 mV has a significant subthreshold leakage current. Designs (e.g. desktop processors) which include vast numbers of circuits which are not actively switching still consume power because of this leakage current. Leakage power is a significant portion of the total power consumed by such designs. Further technology advances that use even thinner gate dielectrics have an additional leakage component because of current tunnelling through the extremely thin gate dielectric. Using high-k dielectrics instead of silicon dioxide that is the conventional gate dielectric allows similar device performance, but with a thicker gate insulator, thus avoiding this current. Leakage power reduction using new material and system designs is critical to sustaining scaling of CMOS.

Analog CMOS

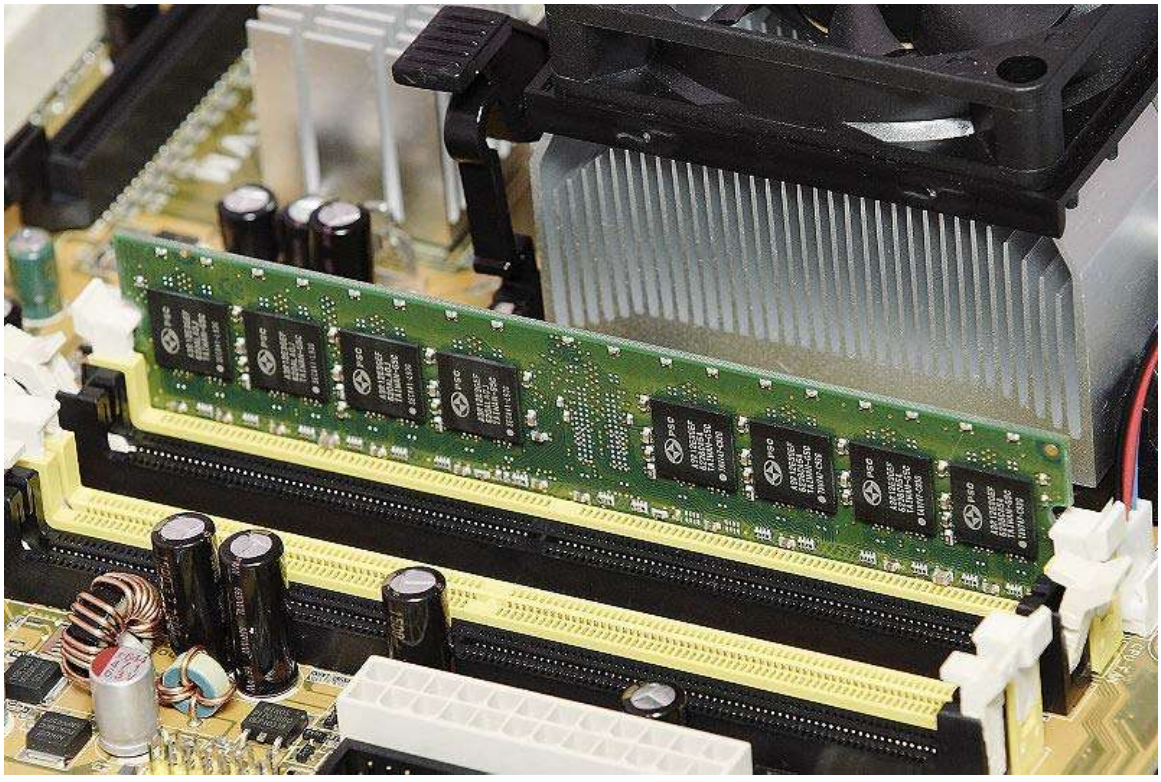
Besides digital applications, CMOS technology is also used in analog applications. For example, there are CMOS operational amplifier ICs available in the market. Transmission gates may be used instead of signal relays. CMOS technology is also widely used for RF circuits all the way to microwave frequencies, in mixed-signal (analog+digital) applications.

Temperature range

Conventional CMOS devices work over a range of −55 °C to +125 °C. There were theoretical indications as early as August 2008 that silicon CMOS will work down to −233 °C (40 K). Functioning temperatures near 40 K have since been achieved using overclocked AMD Phenom II processors with a combination of liquid nitrogen and liquid helium cooling.

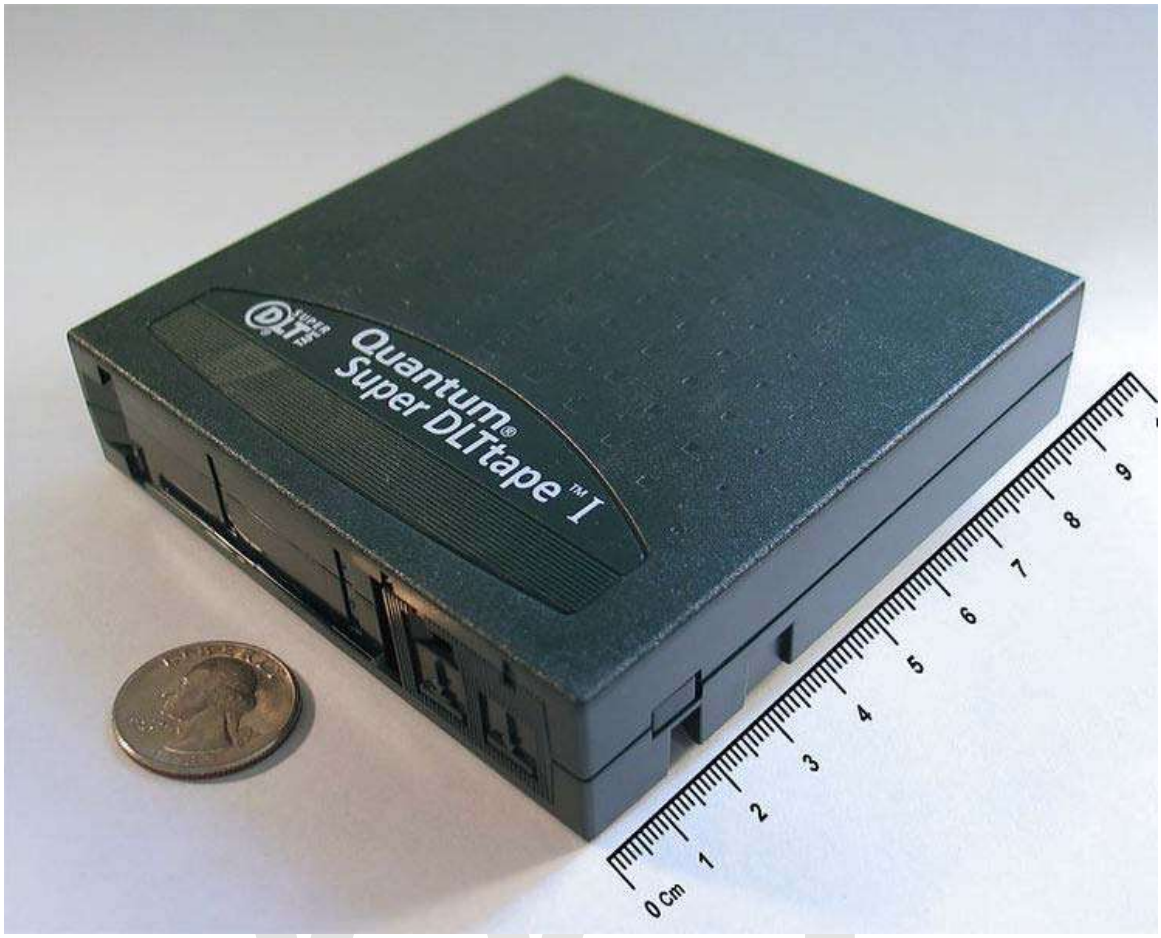
Chapter- 4

Computer Data Storage



1 GB of SDRAM mounted in a personal computer. An example of *primary* storage.

40 GB PATA hard disk drive (HDD); when connected to a computer it serves as *secondary* storage.



160 GB SDLT tape cartridge, an example of *off-line* storage. When used within a robotic tape library, it is classified as *tertiary* storage instead.

Computer data storage, often called **storage** or **memory**, refers to computer components and recording media that retain digital data used for computing for some interval of time. Computer data storage provides one of the core functions of the modern computer, that of information retention. It is one of the fundamental components of all modern computers, and coupled with a central processing unit (CPU, a processor), implements the basic computer model used since the 1940s.

In contemporary usage, *memory* usually refers to a form of semiconductor storage known as random-access memory, typically DRAM (Dynamic-RAM) but *memory* can refer to other forms of fast but temporary storage. Similarly, *storage* today more commonly refers to storage devices and their media not directly accessible by the CPU (secondary or tertiary storage) — typically hard disk drives, optical disc drives, and other devices slower than RAM but more permanent. Historically, *memory* has been called *main memory*, *real storage* or *internal memory* while storage devices have been referred to as *secondary storage*, *external memory* or *auxiliary/peripheral storage*.

The contemporary distinctions are helpful, because they are also fundamental to the architecture of computers in general. The distinctions also reflect an important and significant technical difference between memory and mass storage devices, which has been blurred by the historical usage of the term *storage*.

Many different forms of storage, based on various natural phenomena, have been invented. So far, no practical universal storage medium exists, and all forms of storage have some drawbacks. Therefore a computer system usually contains several kinds of storage, each with an individual purpose.

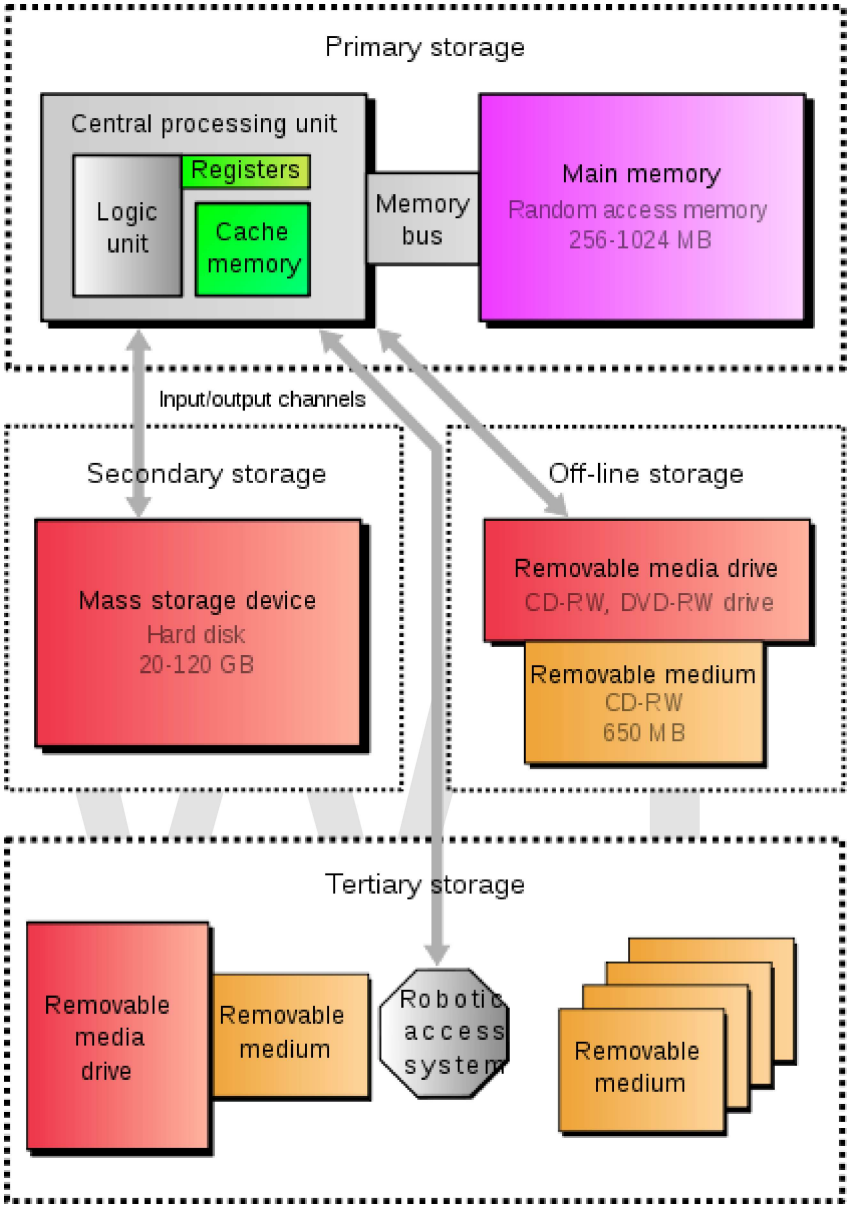
A digital computer represents data using the binary numeral system. Text, numbers, pictures, audio, and nearly any other form of information can be converted into a string of bits, or binary digits, each of which has a value of 1 or 0. The most common unit of storage is the byte, equal to 8 bits. A piece of information can be handled by any computer whose storage space is large enough to accommodate *the binary representation of the piece of information*, or simply data. For example, using eight million bits, or about one megabyte, a typical computer could store a short novel.

Traditionally the most important part of every computer is the central processing unit (CPU, or simply a processor), because it actually operates on data, performs any calculations, and controls all the other components.

Without a significant amount of memory, a computer would merely be able to perform fixed operations and immediately output the result. It would have to be reconfigured to change its behavior. This is acceptable for devices such as desk calculators or simple digital signal processors. Von Neumann machines differ in that they have a memory in which they store their operating instructions and data. Such computers are more versatile in that they do not need to have their hardware reconfigured for each new program, but can simply be reprogrammed with new in-memory instructions; they also tend to be simpler to design, in that a relatively simple processor may keep state between successive computations to build up complex procedural results. Most modern computers are von Neumann machines.

In practice, almost all computers use a variety of memory types, organized in a storage hierarchy around the CPU, as a trade-off between performance and cost. Generally, the lower a storage is in the hierarchy, the lesser its bandwidth and the greater its access latency is from the CPU. This traditional division of storage to primary, secondary, tertiary and off-line storage is also guided by cost per bit.

Hierarchy of storage



Various forms of storage, divided according to their distance from the central processing unit. The fundamental components of a general-purpose computer are arithmetic and logic unit, control circuitry, storage space, and input/output devices. Technology and capacity as in common home computers around 2005.

Primary storage

Primary storage (or **main memory** or **internal memory**), often referred to simply as **memory**, is the only one directly accessible to the CPU. The CPU continuously reads instructions stored there and executes them as required. Any data actively operated on is also stored there in uniform manner.

Historically, early computers used delay lines, Williams tubes, or rotating magnetic drums as primary storage. By 1954, those unreliable methods were mostly replaced by magnetic core memory. Core memory remained dominant until the 1970s, when advances in integrated circuit technology allowed semiconductor memory to become economically competitive.

This led to modern random-access memory (RAM). It is small-sized, light, but quite expensive at the same time. (The particular types of RAM used for primary storage are also volatile, i.e. they lose the information when not powered).

As shown in the diagram, traditionally there are two more sub-layers of the primary storage, besides main large-capacity RAM:

- Processor registers are located inside the processor. Each register typically holds a word of data (often 32 or 64 bits). CPU instructions instruct the arithmetic and logic unit to perform various calculations or other operations on this data (or with the help of it). Registers are the fastest of all forms of computer data storage.
- Processor cache is an intermediate stage between ultra-fast registers and much slower main memory. It's introduced solely to increase performance of the computer. Most actively used information in the main memory is just duplicated in the cache memory, which is faster, but of much lesser capacity. On the other hand it is much slower, but much larger than processor registers. Multi-level hierarchical cache setup is also commonly used—*primary cache* being smallest, fastest and located inside the processor; *secondary cache* being somewhat larger and slower.

Main memory is directly or indirectly connected to the central processing unit via a *memory bus*. It is actually two buses (not on the diagram): an address bus and a data bus. The CPU firstly sends a number through an address bus, a number called memory address, that indicates the desired location of data. Then it reads or writes the data itself using the data bus. Additionally, a memory management unit (MMU) is a small device between CPU and RAM recalculating the actual memory address, for example to provide an abstraction of virtual memory or other tasks.

As the RAM types used for primary storage are volatile (cleared at start up), a computer containing only such storage would not have a source to read instructions from, in order to start the computer. Hence, non-volatile primary storage containing a small startup program (BIOS) is used to bootstrap the computer, that is, to read a larger program from non-volatile *secondary* storage to RAM and start to execute it. A non-volatile technology used for this purpose is called ROM, for read-only memory (the terminology may be somewhat confusing as most ROM types are also capable of *random access*).

Many types of "ROM" are not literally *read only*, as updates are possible; however it is slow and memory must be erased in large portions before it can be re-written. Some embedded systems run programs directly from ROM (or similar), because such programs are rarely changed. Standard computers do not store non-rudimentary programs in ROM,

rather use large capacities of secondary storage, which is non-volatile as well, and not as costly.

Recently, *primary storage* and *secondary storage* in some uses refer to what was historically called, respectively, *secondary storage* and *tertiary storage*.

Secondary storage

A hard disk drive with protective cover removed.

Secondary storage (also known as external memory or auxiliary storage), differs from primary storage in that it is not directly accessible by the CPU. The computer usually uses its input/output channels to access secondary storage and transfers the desired data using intermediate area in primary storage. Secondary storage does not lose the data when the device is powered down—it is non-volatile. Per unit, it is typically also two orders of magnitude less expensive than primary storage. Consequently, modern computer systems typically have two orders of magnitude more secondary storage than primary storage and data is kept for a longer time there.

In modern computers, hard disk drives are usually used as secondary storage. The time taken to access a given byte of information stored on a hard disk is typically a few thousandths of a second, or milliseconds. By contrast, the time taken to access a given byte of information stored in random access memory is measured in billionths of a second, or nanoseconds. This illustrates the significant access-time difference which distinguishes solid-state memory from rotating magnetic storage devices: hard disks are

typically about a million times slower than memory. Rotating optical storage devices, such as CD and DVD drives, have even longer access times. With disk drives, once the disk read/write head reaches the proper placement and the data of interest rotates under it, subsequent data on the track are very fast to access. As a result, in order to hide the initial seek time and rotational latency, data is transferred to and from disks in large contiguous blocks.

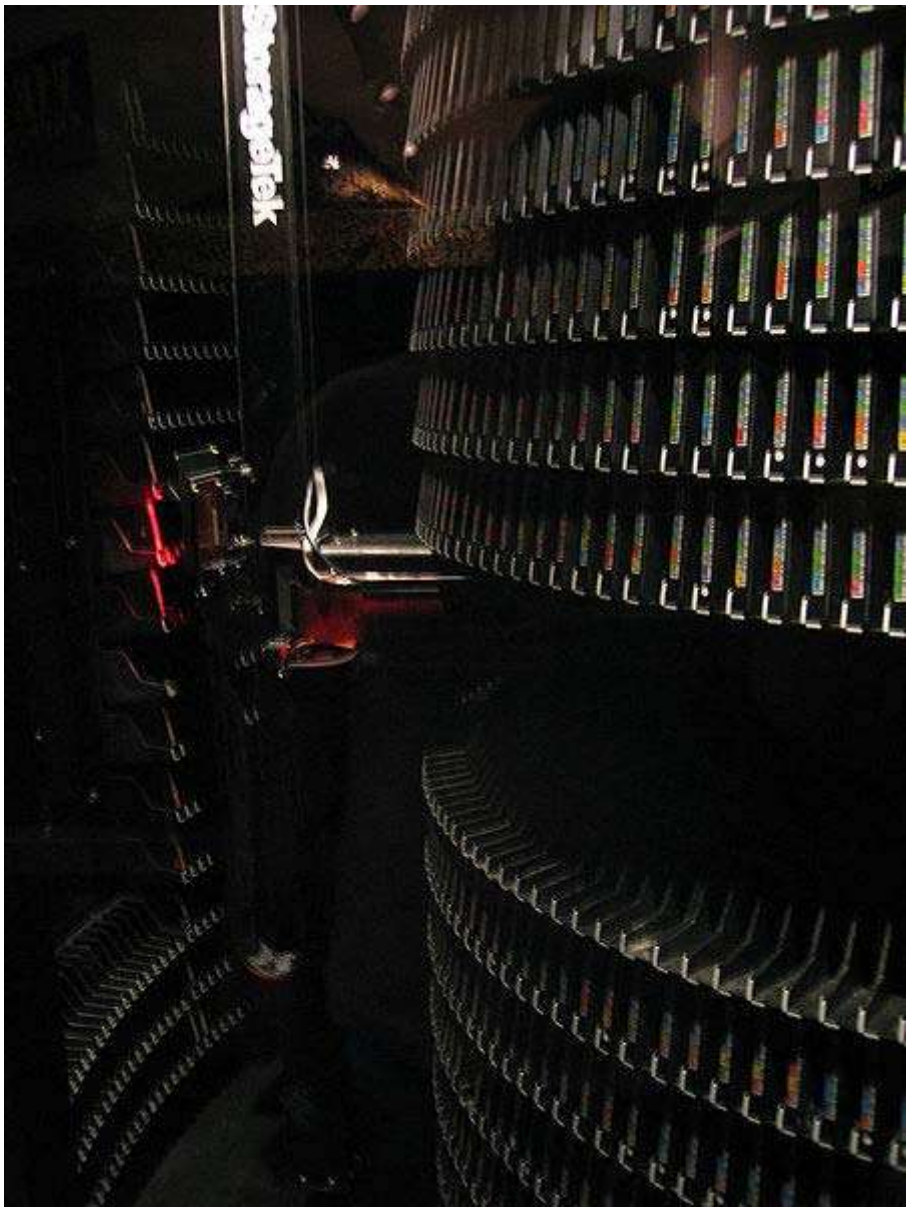
When data reside on disk, block access to hide latency offers a ray of hope in designing efficient external memory algorithms. Sequential or block access on disks is orders of magnitude faster than random access, and many sophisticated paradigms have been developed to design efficient algorithms based upon sequential and block access . Another way to reduce the I/O bottleneck is to use multiple disks in parallel in order to increase the bandwidth between primary and secondary memory.

Some other examples of secondary storage technologies are: flash memory (e.g. USB flash drives or keys), floppy disks, magnetic tape, paper tape, punched cards, standalone RAM disks, and Iomega Zip drives.

The secondary storage is often formatted according to a file system format, which provides the abstraction necessary to organize data into files and directories, providing also additional information (called metadata) describing the owner of a certain file, the access time, the access permissions, and other information.

Most computer operating systems use the concept of virtual memory, allowing utilization of more primary storage capacity than is physically available in the system. As the primary memory fills up, the system moves the least-used chunks (*pages*) to secondary storage devices (to a swap file or page file), retrieving them later when they are needed. As more of these retrievals from slower secondary storage are necessary, the more the overall system performance is degraded.

Tertiary storage



Large tape library. Tape cartridges placed on shelves in the front, robotic arm moving in the back. Visible height of the library is about 180 cm.

Tertiary storage or **tertiary memory**, provides a third level of storage. Typically it involves a robotic mechanism which will *mount* (insert) and *dismount* removable mass storage media into a storage device according to the system's demands; this data is often copied to secondary storage before use. It is primarily used for archival of rarely accessed information since it is much slower than secondary storage (e.g. 5–60 seconds vs. 1-10 milliseconds). This is primarily useful for extraordinarily large data stores, accessed without human operators. Typical examples include tape libraries and optical jukeboxes.

When a computer needs to read information from the tertiary storage, it will first consult a catalog database to determine which tape or disc contains the information. Next, the computer will instruct a robotic arm to fetch the medium and place it in a drive. When the computer has finished reading the information, the robotic arm will return the medium to its place in the library.

Off-line storage

Off-line storage is a computer data storage on a medium or a device that is not under the control of a processing unit. The medium is recorded, usually in a secondary or tertiary storage device, and then physically removed or disconnected. It must be inserted or connected by a human operator before a computer can access it again. Unlike tertiary storage, it cannot be accessed without human interaction.

Off-line storage is used to transfer information, since the detached medium can be easily physically transported. Additionally, in case a disaster, for example a fire, destroys the original data, a medium in a remote location will probably be unaffected, enabling disaster recovery. Off-line storage increases general information security, since it is physically inaccessible from a computer, and data confidentiality or integrity cannot be affected by computer-based attack techniques. Also, if the information stored for archival purposes is accessed seldom or never, off-line storage is less expensive than tertiary storage.

In modern personal computers, most secondary and tertiary storage media are also used for off-line storage. Optical discs and flash memory devices are most popular, and to much lesser extent removable hard disk drives. In enterprise uses, magnetic tape is predominant. Older examples are floppy disks, Zip disks, or punched cards.

Characteristics of storage

A 1GB DDR RAM memory module (detail)

Storage technologies at all levels of the storage hierarchy can be differentiated by evaluating certain core characteristics as well as measuring characteristics specific to a particular implementation. These core characteristics are volatility, mutability, accessibility, and addressibility. For any particular implementation of any storage technology, the characteristics worth measuring are capacity and performance.

Volatility

Will retain the stored information even if it is not constantly supplied with electric power. It is suitable for long-term storage of information.

Volatile memory

: Requires constant power to maintain the stored information. The fastest memory technologies of today are volatile ones (not a universal rule). Since primary storage is required to be very fast, it predominantly uses volatile memory.

Differentiation

Dynamic random access memory

: A form of volatile memory which also requires the stored information to be periodically re-read and re-written, or refreshed, otherwise it would vanish.

Static memory

: A form of volatile memory similar to DRAM with the exception that it never needs to be refreshed as long as power is applied. (It loses its content if power is removed).

Mutability

Read/write storage or mutable storage

: Allows information to be overwritten at any time. A computer without some amount of read/write storage for primary storage purposes would be useless for many tasks. Modern computers typically use read/write storage also for secondary storage.

Read only storage

: Retains the information stored at the time of manufacture, and **write once storage** (Write Once Read Many) allows the information to be written only once at some point after manufacture. These are called **immutable storage**. Immutable storage is used for tertiary and off-line storage. Examples include CD-ROM and CD-R.

Slow write, fast read storage

: Read/write storage which allows information to be overwritten multiple times, but with the write operation being much slower than the read operation. Examples include CD-RW and flash memory.

Accessibility

Random access

: Any location in storage can be accessed at any moment in approximately the same amount of time. Such characteristic is well suited for primary and secondary storage.

Sequential access

: The accessing of pieces of information will be in a serial order, one after the other; therefore the time to access a particular piece of information depends upon which piece of information was last accessed. Such characteristic is typical of off-line storage.

Addressability

Location-addressable

: Each individually accessible unit of information in storage is selected with its numerical memory address. In modern computers, location-addressable storage

usually limits to primary storage, accessed internally by computer programs, since location-addressability is very efficient, but burdensome for humans.

File addressable
: Information is divided into *files* of variable length, and a particular file is selected with human-readable directory and file names. The underlying device is still location-addressable, but the operating system of a computer provides the file system abstraction to make the operation more understandable. In modern computers, secondary, tertiary and off-line storage use file systems.

Content-addressable
: Each individually accessible unit of information is selected based on the basis of (part of) the contents stored there. Content-addressable storage can be implemented using software (computer program) or hardware (computer device), with hardware being faster but more expensive option. Hardware content addressable memory is often used in a computer's CPU cache.

Capacity

Raw capacity
: The total amount of stored information that a storage device or medium can hold. It is expressed as a quantity of bits or bytes (e.g. 10.4 megabytes).

Memory storage density
: The compactness of stored information. It is the storage capacity of a medium divided with a unit of length, area or volume (e.g. 1.2 megabytes per square inch).

Performance

Latency
: The time it takes to access a particular location in storage. The relevant unit of measurement is typically nanosecond for primary storage, millisecond for secondary storage, and second for tertiary storage. It may make sense to separate read latency and write latency, and in case of sequential access storage, minimum, maximum and average latency.

Throughput
: The rate at which information can be read from or written to the storage. In computer data storage, throughput is usually expressed in terms of megabytes per second or MB/s, though bit rate may also be used. As with latency, read rate and write rate may need to be differentiated. Also accessing media sequentially, as opposed to randomly, typically yields maximum throughput.

Energy use

- Storage devices that reduce fan usage, automatically shut-down during inactivity, and low power hard drives can reduce energy consumption 90 percent.
- 2.5 inch hard disk drives often consume less power than larger ones. Low capacity solid-state drives have no moving parts and consume less power than hard disks. Also, memory may use more power than hard disks.

Fundamental storage technologies

As of 2008, the most commonly used data storage technologies are semiconductor, magnetic, and optical, while paper still sees some limited usage. Some other fundamental storage technologies have also been used in the past or are proposed for development.

Semiconductor

Semiconductor memory uses semiconductor-based integrated circuits to store information. A semiconductor memory chip may contain millions of tiny transistors or capacitors. Both *volatile* and *non-volatile* forms of semiconductor memory exist. In modern computers, primary storage almost exclusively consists of dynamic volatile semiconductor memory or dynamic random access memory. Since the turn of the century, a type of non-volatile semiconductor memory known as flash memory has steadily gained share as off-line storage for home computers. Non-volatile semiconductor memory is also used for secondary storage in various advanced electronic devices and specialized computers.

Magnetic

Magnetic storage uses different patterns of magnetization on a magnetically coated surface to store information. Magnetic storage is *non-volatile*. The information is accessed using one or more read/write heads which may contain one or more recording transducers. A read/write head only covers a part of the surface so that the head or medium or both must be moved relative to another in order to access data. In modern computers, magnetic storage will take these forms:

- Magnetic disk
  - Floppy disk, used for off-line storage
  - Hard disk drive, used for secondary storage
- Magnetic tape data storage, used for tertiary and off-line storage

In early computers, magnetic storage was also used for primary storage in a form of magnetic drum, or core memory, core rope memory, thin-film memory, twistor memory or bubble memory. Also unlike today, magnetic tape was often used for secondary storage.

Optical

Optical storage, the typical optical disc, stores information in deformities on the surface of a circular disc and reads this information by illuminating the surface with a laser diode and observing the reflection. Optical disc storage is *non-volatile*. The deformities may be permanent (read only media), formed once (write once media) or reversible (recordable or read/write media). The following forms are currently in common use:

- CD, CD-ROM, DVD, BD-ROM: Read only storage, used for mass distribution of digital information (music, video, computer programs)
- CD-R, DVD-R, DVD+R, BD-R: Write once storage, used for tertiary and off-line storage
- CD-RW, DVD-RW, DVD+RW, DVD-RAM, BD-RE: Slow write, fast read storage, used for tertiary and off-line storage
- Ultra Density Optical or UDO is similar in capacity to BD-R or BD-RE and is slow write, fast read storage used for tertiary and off-line storage.

Magneto-optical disc storage is optical disc storage where the magnetic state on a ferromagnetic surface stores information. The information is read optically and written by combining magnetic and optical methods. Magneto-optical disc storage is *non-volatile*, *sequential access*, slow write, fast read storage used for tertiary and off-line storage.

3D optical data storage has also been proposed.

Paper

Paper data storage, typically in the form of paper tape or punched cards, has long been used to store information for automatic processing, particularly before general-purpose computers existed. Information was recorded by punching holes into the paper or cardboard medium and was read mechanically (or later optically) to determine whether a particular location on the medium was solid or contained a hole. A few technologies allow people to make marks on paper that are easily read by machine—these are widely used for tabulating votes and grading standardized tests. Barcodes made it possible for any object that was to be sold or transported to have some computer readable information securely attached to it.

Uncommon

Vacuum tube memory
: A Williams tube used a cathode ray tube, and a Selectron tube used a large vacuum tube to store information. These primary storage devices were short-lived in the market, since Williams tube was unreliable and Selectron tube was expensive.

Electro-acoustic memory
: Delay line memory used sound waves in a substance such as mercury to store information. Delay line memory was dynamic volatile, cycle sequential read/write storage, and was used for primary storage.

Optical tape
: is a medium for optical storage generally consisting of a long and narrow strip of plastic onto which patterns can be written and from which the patterns can be read back. It shares some technologies with cinema film stock and optical discs, but is compatible with neither. The motivation behind developing this technology was the possibility of far greater storage capacities than either magnetic tape or optical discs.

Phase-change memory

uses different mechanical phases of Phase Change Material to store information in an X-Y addressable matrix, and reads the information by observing the varying electrical resistance of the material. Phase-change memory would be non-volatile, random access read/write storage, and might be used for primary, secondary and off-line storage. Most rewritable and many write once optical disks already use phase change material to store information.

Holographic data storage
: stores information optically inside crystals or photopolymers. Holographic storage can utilize the whole volume of the storage medium, unlike optical disc storage which is limited to a small number of surface layers. Holographic storage would be non-volatile, sequential access, and either write once or read/write storage. It might be used for secondary and off-line storage.

Molecular memory
: stores information in polymer that can store electric charge. Molecular memory might be especially suited for primary storage. The theoretical storage capacity of molecular memory is 10 terabits per square inch.

Related technologies

Network connectivity

A secondary or tertiary storage may connect to a computer utilizing computer networks. This concept does not pertain to the primary storage, which is shared between multiple processors in a much lesser degree.

- **Direct-attached storage** (DAS) is a traditional mass storage, that does not use any network. This is still a most popular approach. This term was coined lately, together with NAS and SAN.
- **Network-attached storage** (NAS) is mass storage attached to a computer which another computer can access at file level over a local area network, a private wide area network, or in the case of online file storage, over the Internet. NAS is commonly associated with the NFS and CIFS/SMB protocols.
- **Storage area network** (SAN) is a specialized network, that provides other computers with storage capacity. The crucial difference between NAS and SAN is the former presents and manages file systems to client computers, whilst the latter provides access at block-addressing (raw) level, leaving it to attaching systems to manage data or file systems within the provided capacity. SAN is commonly associated with Fibre Channel networks.

Robotic storage

Large quantities of individual magnetic tapes, and optical or magneto-optical discs may be stored in robotic tertiary storage devices. In tape storage field they are known as tape libraries, and in optical storage field optical jukeboxes, or optical disk libraries per analogy. Smallest forms of either technology containing just one drive device are referred to as autoloaders or autochangers.

Robotic-access storage devices may have a number of slots, each holding individual media, and usually one or more picking robots that traverse the slots and load media to built-in drives. The arrangement of the slots and picking devices affects performance. Important characteristics of such storage are possible expansion options: adding slots, modules, drives, robots. Tape libraries may have from 10 to more than 100,000 slots, and provide terabytes or petabytes of near-line information. Optical jukeboxes are somewhat smaller solutions, up to 1,000 slots.

Robotic storage is used for backups, and for high-capacity archives in imaging, medical, and video industries. Hierarchical storage management is a most known archiving strategy of automatically *migrating* long-unused files from fast hard disk storage to libraries or jukeboxes. If the files are needed, they are *retrieved* back to disk.

Chapter- 5

Delay Line Memory

Delay line memory was a form of computer memory used on some of the earliest digital computers. Like many modern forms of electronic computer memory, delay line memory was a refreshable memory, but as opposed to modern random-access memory, delay line memory was serial-access. In the earliest forms of delay line memory, information introduced to the memory in the form of electric pulses was transduced into mechanical waves that propagated relatively slowly through a medium, such as a cylinder filled with a liquid like mercury, a magnetostrictive coil, or a piezoelectric crystal. The propagation medium could support the propagation of hundreds or thousands of pulses at any one time. Upon reaching the other end of the propagation medium, the waves were re-transduced into electric pulses, amplified, shaped, and reintroduced to the propagation medium at the beginning, thus refreshing the memory. Accessing a desired part of the propagation medium's memory contents required waiting for the pulses of interest to reach the end of the medium, a wait typically on the order of microseconds. Use of a delay line for a computer memory was invented by J. Presper Eckert in the mid-1940s for use in computers such as the EDVAC and the UNIVAC I.

Genesis in radar

The basic concept of the delay line originated with World War II radar research, as a system to reduce clutter from reflections from the ground and other "fixed" objects.

A radar system consists largely of an antenna, a transmitter, a receiver, and a display of some sort. The antenna is connected to the transmitter, which sends out a brief pulse of radio energy before being disconnected again. The antenna is then connected to the receiver, which amplifies any reflected signals, and sends them to the display. Objects farther from the radar return echos later in time than those located closer to the radar, which the display indicates visually.

Non-moving objects at a fixed distance from the antenna always return a signal after the same delay. This would appear as a fixed spot on the display, making detection of other targets in the area more difficult. Early radars simply aimed their beams away from the ground in order to avoid the majority of this "clutter". This was not an ideal situation by

any means; it required careful setup and aiming which was not very easy for smaller mobile radars, and did nothing to remove other sources of clutter like reflections off certain terrain features.

To filter these returns out, two pulses were compared, and returns with common timing are removed. To do this, the signal sent from the receiver to the display was split in two, with one path leading directly to the display, and the second leading to a delay unit. The delay was carefully tuned to delay the signals some multiple of the time between pulses (the pulse repetition frequency), that way the delayed signal from an earlier pulse would exit the delay unit at the same time as a newer pulse was being received from the antenna. One of the signals was then inverted, typically the one from the delay, and the two signals were then combined and sent to the display. Any signal that was at the same location was nullified by the inverted signal from a previous pulse, leaving only the moving objects on the display.

Mercury memory of UNIVAC I (1951)

Several different types of delay systems were invented for this purpose, with one common principle being that the information was stored acoustically in a medium. MIT experimented with a number of systems including glass, quartz, steel and lead. The Japanese deployed a system consisting of a quartz element with a powdered glass coating that reduced surface waves that interfered with proper reception. The United States Naval Research Laboratory used steel rods wrapped into a helix, but this was useful only for low frequencies under 1 MHz. Raytheon used a magnesium alloy originally developed for making bells.

The first practical de-cluttering system based on the concept was developed by J. Presper Eckert at the University of Pennsylvania's Moore School of Electrical Engineering. His solution used a column of mercury with piezo crystal transducers (a combination of speaker and microphone) at either end. Signals from the radar amplifier were sent to the

piezo at one end of the tube, which would cause the transducer to pulse and generate a small wave in the mercury. The wave would quickly travel to the far end of the tube, where it would be read back out by the other piezo, inverted, and sent to the display. Careful mechanical arrangement was needed to ensure the delay time matched the inter-pulse timing of the particular radar being used.

All of these systems were suitable for conversion into a computer memory. The key was to recycle the signals within the memory system so they would not disappear after traveling through the delay. This was relatively easy to arrange with simple electronics.

Acoustic delay lines

Mercury delay lines

After the war Eckert turned his attention to computer development, which was a topic of some interest at the time. One problem with practical development was the lack of a suitable memory device, and Eckert's work on the radar delays meant he had a major advantage over other researchers in this regard.

For a computer application the timing was still critical, but for a different reason. Conventional computers have a natural "cycle time" needed to complete an operation, the start and end of which typically consist of reading or writing memory. Thus the delay lines had to be timed such that the pulses would arrive at the receiver just as the computer was ready to read it. Typically many pulses would be "in flight" through the delay, and the computer would count the pulses by comparing to a master clock to find the particular bit it was looking for.

Mercury was used because the acoustic impedance of mercury is almost exactly the same as that of the piezoelectric quartz crystals; this minimized the energy loss and the echoes when the signal was transmitted from crystal to medium and back again. The high speed of sound in mercury (1450 m/s) meant that the time needed to wait for a pulse to arrive at the receiving end was less than it would have been with a slower medium, such as air, but it also meant that the total number of pulses that could be stored in any reasonably sized column of mercury was limited. Other technical drawbacks of mercury included its weight, its cost, and its toxicity. Moreover, to get the acoustic impedances to match as closely as possible, the mercury had to be kept at a constant temperature. The system heated the mercury to a uniform above-room temperature setting of 40 °C (100 °F), which made servicing the tubes hot and uncomfortable work. (Alan Turing proposed the use of gin as an ultrasonic delay medium, claiming that it had the necessary acoustic properties.)

A considerable amount of engineering was needed to maintain a "clean" signal inside the tube. Large transducers were used to generate a very tight "beam" of sound that would not touch the walls of the tube, and care had to be taken to eliminate reflections off the far end of the tubes. The tightness of the beam then required considerable tuning to make sure the two piezos were pointed directly at each other. Since the speed of sound changes

with temperature (because of the change in density with temperature) the tubes were heated in large ovens to keep them at a precise temperature. Other systems instead adjusted the computer clock rate according to the ambient temperature to achieve the same effect.

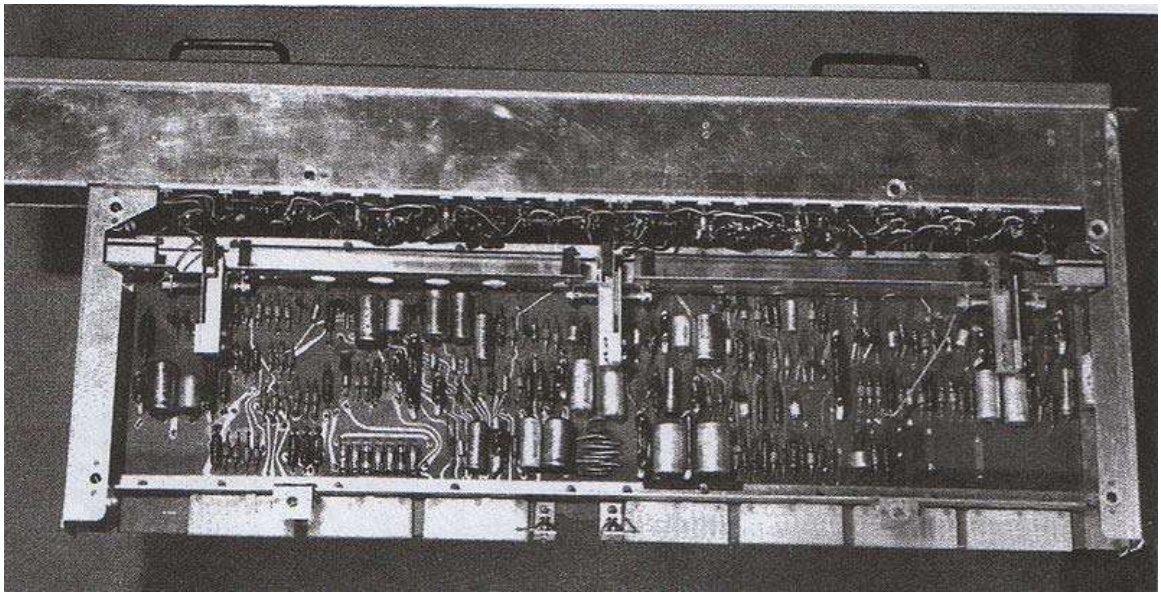
EDSAC, designed to be the first stored-program digital computer, began operation with 512 35-bit words of memory, stored in 32 delay lines holding 576 bits each (a 36th bit was added to every word as a start/stop indicator). In the UNIVAC I this was reduced somewhat, each column stored 120 bits (although the term "bit" was not in popular use at the time), requiring seven large memory units with 18 columns each to make up a 1000-word store. Combined with their support circuitry and amplifiers, the memory subsystem formed its own walk-in room. The average access time was about 222 microseconds, which was considerably faster than the mechanical systems used on earlier computers.

CSIRAC, completed in November 1949, also used delay line memory.

Magnetostrictive delay lines

A later version of the delay line used metal wires as the storage medium. Transducers were built by applying the magnetostrictive effect; small pieces of a magnetostrictive material, typically nickel, were attached to either side of the end of the wire, inside an electromagnet. When bits from the computer entered the magnets the nickel would contract or expand (based on the polarity) and twist the end of the wire. The resulting torsional wave would then move down the wire just as the sound wave did down the mercury column. In most cases the entire wire was made of the same material.

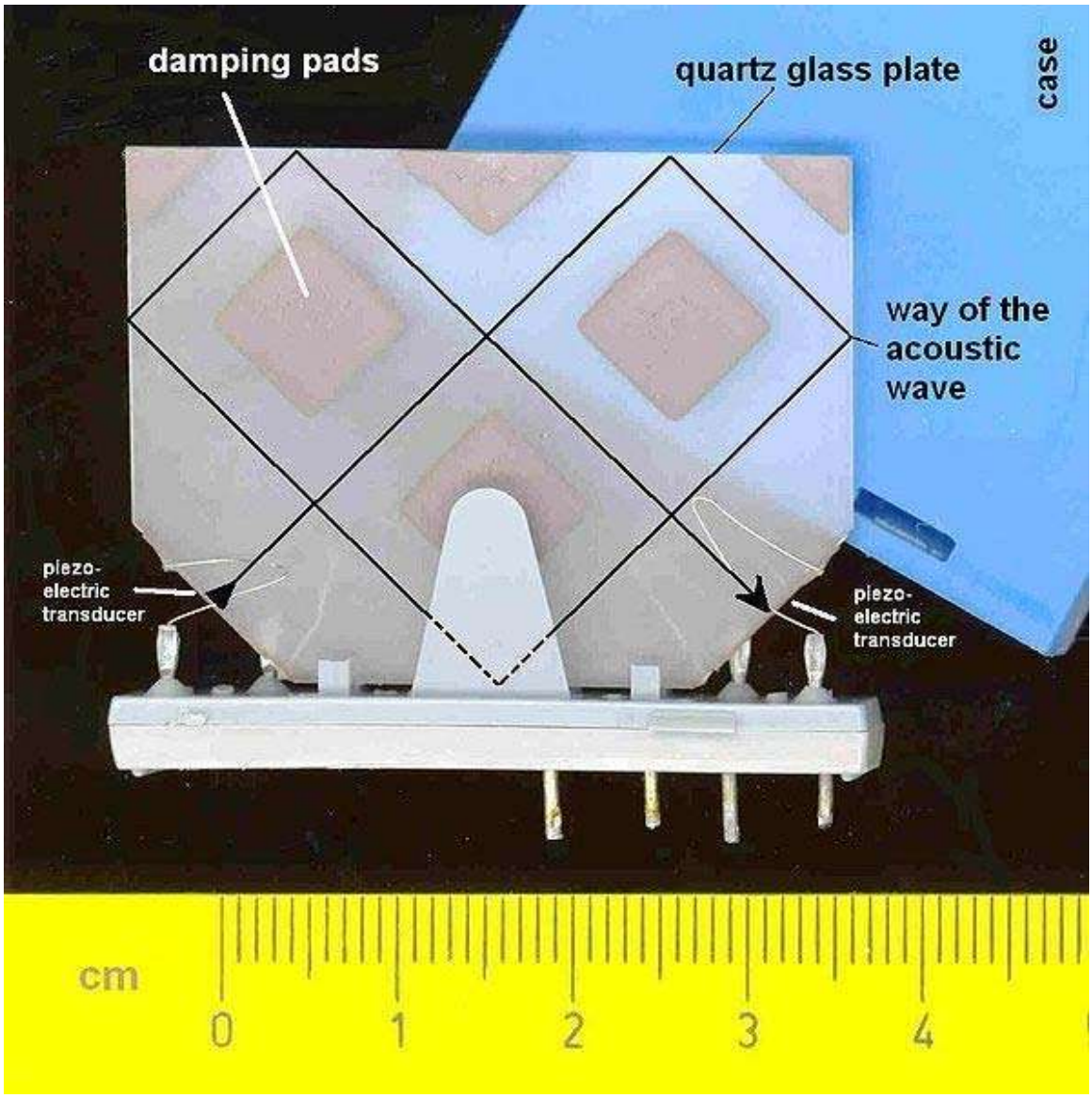
Unlike the compressive wave, however, the torsional waves are considerably more resistant to problems caused by mechanical imperfections, so much so that the wires could be wound into a loose coil and pinned to a board. Due to their ability to be coiled, the wire-based systems could be built as "long" as needed, and tended to hold considerably more data per unit; 1k units were typical on a board only 1 foot square. Of course this also meant that the time needed to find a particular bit was somewhat longer as it traveled through the wire, and access times on the order of 500 microseconds were typical.



100 microsecond delay line store

Delay line memory was far less expensive and far more reliable per bit than flip-flops made from tubes, and yet far faster than a latching relay. It was used right into the late 1960s, notably on British commercial machines like the LEO I, Highgate Wood Telephone Exchange, and various Ferranti machines. Delay line memory was also used for video memory in early terminals, where one delay line would typically store 4 lines of characters. (4 lines x 40 characters per line x 6 bits per character= 960 bits in one delay line) They were also used very successfully in several models of early desktop electronic calculator, including the Friden EC130 (1964) and EC132, the Olivetti Programma 101 desktop programmable calculator introduced in 1965, and the Litton Monroe Epic 2000 and 3000 programmable calculators of 1967.

Piezoelectric delay lines



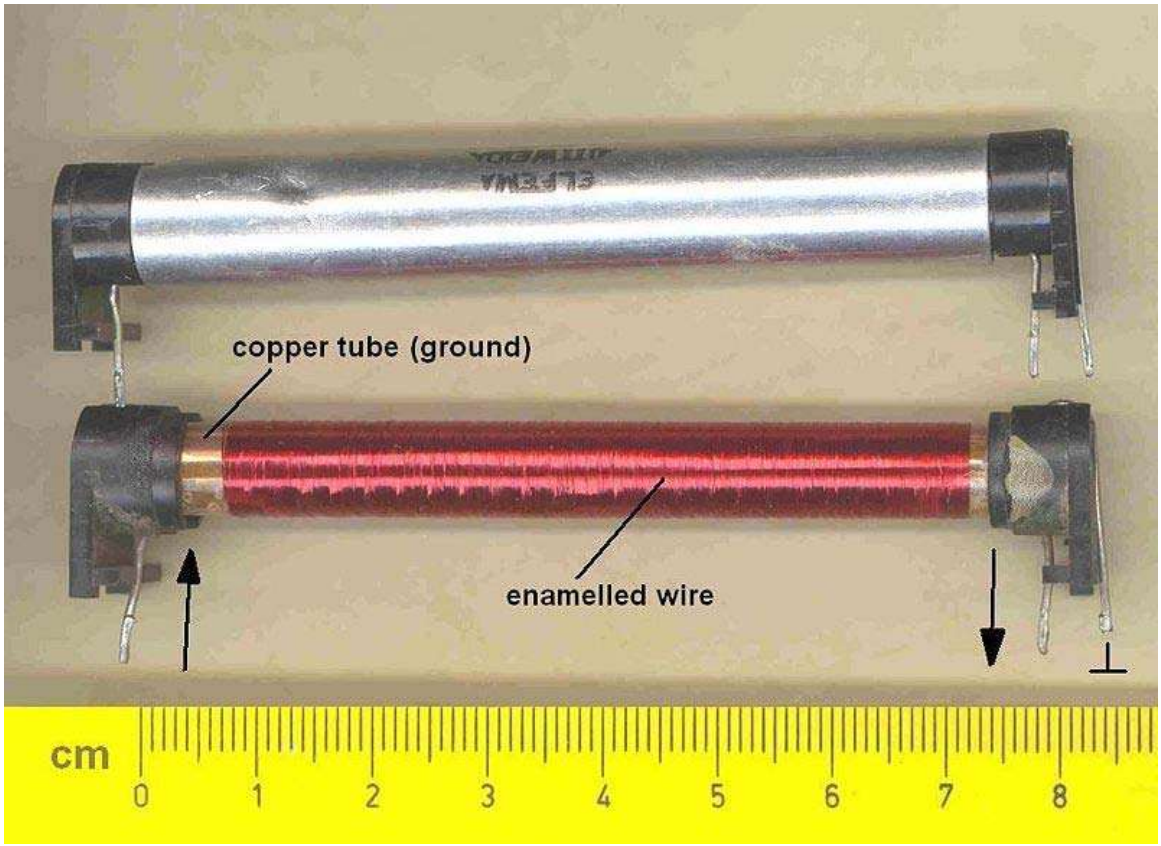
An ultrasonic delay line from a Colour-TV; it delays the colour signal for 64µs
Manufacturer: VEB ELFEMA Mittweida (GDR) in 1980

A similar solution to the magnetostrictive system was to use delay lines made entirely of a piezo material, typically quartz. Current fed into one end of the crystal would generate a compressive wave that would flow to the other end where it could be read out. In effect, piezoelectric delays simply replaced the mercury and transducers of a conventional mercury delay line with a single unit combining both. However these solutions were fairly rare; building crystals of the required quality in large sizes was not easy, limiting them to small sizes, and thus small amounts of data storage.

A better and more widespread use of piezoelectric delays was in European television sets. The European PAL standard for color broadcasts compares the signal from two

subsequent lines in order to avoid color shifting due to small phase shifts. By comparing two lines, one inverted, the shifting is averaged out and returns a signal more closely matching the original even under interference. In order to compare the two lines, a piezo delay tuned to the timing of the lines, 64 μs, is inserted in the signal path. The delay unit is shaped to "fold" the beam multiple times through the crystal, greatly reducing its length and producing a small square-ish device.

Electric delay lines



Electric delay line (450ns), consisting of enamelled copper wire, wound around a metal tube

Electric delay lines are used for shorter delay times (ns to several μs). They consist of a long electric line or are made of discrete inductors and capacitors, which are arranged in a chain. To shorten the total length of the line it can be wound around a metal tube, getting some more capacitance against ground and also more inductance due to the wire windings, which are laying close together.

Other examples are:

- short coaxial or microstrip lines for phase matching in high frequency circuits or antennas

- hollow resonator lines in magnetrons and klystrons as helices in travelling wave tubes to match the velocity of the electrons to the velocity of the electromagnetic waves
- undulators in free electron lasers

Another way to create a delay time is to implement a delay line in an integrated circuit storage device. This can be done digitally or with a discrete analogue method. The analogue one uses bucket-brigade devices or charge coupled devices (CCD), which transport a stored electric charge stepwise from one end to the other. Both digital and analog methods are bandwidth limited at the upper end to the half of the clock frequency, which determines the steps of transportation.

In modern computers operating at gigahertz speeds, millimeter differences in the length of conductors in a parallel data bus can cause data-bit skew, which can lead to data corruption or reduced processing performance. This is remedied by making all conductor paths of similar length, delaying the arrival time for what would otherwise be shorter travel distances by using zig-zagging traces.

Chapter- 6

Digital Signal Processing

Digital signal processing (**DSP**) is concerned with the representation of signals by a sequence of numbers or symbols and the processing of these signals. Digital signal processing and analog signal processing are subfields of signal processing. DSP includes subfields like: audio and speech signal processing, sonar and radar signal processing, sensor array processing, spectral estimation, statistical signal processing, digital image processing, signal processing for communications, control of systems, biomedical signal processing, seismic data processing, etc.

The goal of DSP is usually to measure, filter and/or compress continuous real-world analog signals. The first step is usually to convert the signal from an analog to a digital form, by *sampling* it using an analog-to-digital converter (ADC), which turns the analog signal into a stream of numbers. However, often, the required output signal is another analog output signal, which requires a digital-to-analog converter (DAC). Even if this process is more complex than analog processing and has a discrete value range, the application of computational power to digital signal processing allows for many advantages over analog processing in many applications, such as error detection and correction in transmission as well as data compression.

DSP algorithms have long been run on standard computers, on specialized processors called digital signal processors (DSPs), or on purpose-built hardware such as application-specific integrated circuit (ASICs). Today there are additional technologies used for digital signal processing including more powerful general purpose microprocessors, field-programmable gate arrays (FPGAs), digital signal controllers (mostly for industrial apps such as motor control), and stream processors, among others.

Signal sampling

With the increasing use of computers the usage of and need for digital signal processing has increased. In order to use an analog signal on a computer it must be digitized with an analog-to-digital converter. Sampling is usually carried out in two stages, discretization and quantization. In the discretization stage, the space of signals is partitioned into equivalence classes and quantization is carried out by replacing the signal with

representative signal of the corresponding equivalence class. In the quantization stage the representative signal values are approximated by values from a finite set.

The Nyquist–Shannon sampling theorem states that a signal can be exactly reconstructed from its samples if the sampling frequency is greater than twice the highest frequency of the signal; but requires an infinite number of samples . In practice, the sampling frequency is often significantly more than twice that required by the signal's limited bandwidth.

A digital-to-analog converter is used to convert the digital signal back to analog. The use of a digital computer is a key ingredient in digital control systems.

DSP domains

In DSP, engineers usually study digital signals in one of the following domains: time domain (one-dimensional signals), spatial domain (multidimensional signals), frequency domain, autocorrelation domain, and wavelet domains. They choose the domain in which to process a signal by making an informed guess (or by trying different possibilities) as to which domain best represents the essential characteristics of the signal. A sequence of samples from a measuring device produces a time or spatial domain representation, whereas a discrete Fourier transform produces the frequency domain information, that is the frequency spectrum. Autocorrelation is defined as the cross-correlation of the signal with itself over varying intervals of time or space.

Time and space domains

The most common processing approach in the time or space domain is enhancement of the input signal through a method called filtering. Digital filtering generally consists of some linear transformation of a number of surrounding samples around the current sample of the input or output signal. There are various ways to characterize filters; for example:

- A "linear" filter is a linear transformation of input samples; other filters are "non-linear". Linear filters satisfy the superposition condition, i.e. if an input is a weighted linear combination of different signals, the output is an equally weighted linear combination of the corresponding output signals.

- A "causal" filter uses only previous samples of the input or output signals; while a "non-causal" filter uses future input samples. A non-causal filter can usually be changed into a causal filter by adding a delay to it.

- A "time-invariant" filter has constant properties over time; other filters such as adaptive filters change in time.

- Some filters are "stable", others are "unstable". A stable filter produces an output that converges to a constant value with time, or remains bounded within a finite

interval. An unstable filter can produce an output that grows without bounds, with bounded or even zero input.

- A "finite impulse response" (FIR) filter uses only the input signals, while an "infinite impulse response" filter (IIR) uses both the input signal and previous samples of the output signal. FIR filters are always stable, while IIR filters may be unstable.

Filters can be represented by block diagrams which can then be used to derive a sample processing algorithm to implement the filter using hardware instructions. A filter may also be described as a difference equation, a collection of zeroes and poles or, if it is an FIR filter, an impulse response or step response.

The output of a digital filter to any given input may be calculated by convolving the input signal with the impulse response.

Frequency domain

Signals are converted from time or space domain to the frequency domain usually through the Fourier transform. The Fourier transform converts the signal information to a magnitude and phase component of each frequency. Often the Fourier transform is converted to the power spectrum, which is the magnitude of each frequency component squared.

The most common purpose for analysis of signals in the frequency domain is analysis of signal properties. The engineer can study the spectrum to determine which frequencies are present in the input signal and which are missing.

In addition to frequency information, phase information is often needed. This can be obtained from the Fourier transform. With some applications, how the phase varies with frequency can be a significant consideration.

Filtering, particularly in non-realtime work can also be achieved by converting to the frequency domain, applying the filter and then converting back to the time domain. This is a fast, O(n log n) operation, and can give essentially any filter shape including excellent approximations to brickwall filters.

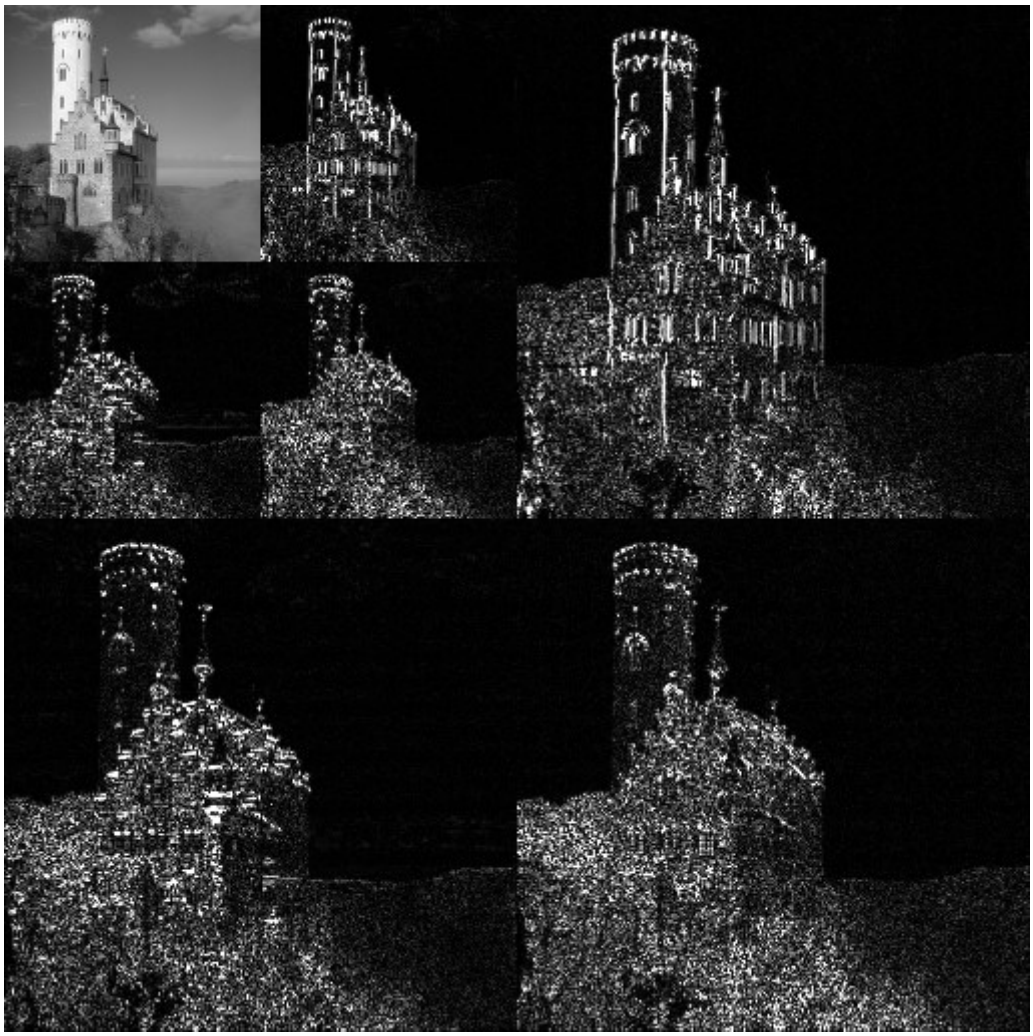
There are some commonly used frequency domain transformations. For example, the cepstrum converts a signal to the frequency domain through Fourier transform, takes the logarithm, then applies another Fourier transform. This emphasizes the frequency components with smaller magnitude while retaining the order of magnitudes of frequency components.

Frequency domain analysis is also called *spectrum-* or *spectral analysis*.

Z-plane analysis

Whereas analog filters are usually analysed in terms of transfer functions in the s plane using Laplace transforms, digital filters are analysed in the z plane in terms of Z-transforms. A digital filter may be described in the z plane by its characteristic collection of zeroes and poles.

Wavelet



An example of the 2D discrete wavelet transform that is used in JPEG2000. The original image is high-pass filtered, yielding the three large images, each describing local changes in brightness (details) in the original image. It is then low-pass filtered and downscaled, yielding an approximation image; this image is high-pass filtered to produce the three smaller detail images, and low-pass filtered to produce the final approximation image in the upper-left.

In numerical analysis and functional analysis, a **discrete wavelet transform** (DWT) is any wavelet transform for which the wavelets are discretely sampled. As with other

wavelet transforms, a key advantage it has over Fourier transforms is temporal resolution: it captures both frequency *and* location information (location in time).

Applications

The main applications of DSP are audio signal processing, audio compression, digital image processing, video compression, speech processing, speech recognition, digital communications, RADAR, SONAR, seismology and biomedicine. Specific examples are speech compression and transmission in digital mobile phones, room correction of sound in hi-fi and sound reinforcement applications, weather forecasting, economic forecasting, seismic data processing, analysis and control of industrial processes, medical imaging such as CAT scans and MRI, MP3 compression, computer graphics, image manipulation, hi-fi loudspeaker crossovers and equalization, and audio effects for use with electric guitar amplifiers.

Implementation

Digital signal processing is often implemented using specialised microprocessors such as the DSP56000, the TMS320, or the SHARC. These often process data using fixed-point arithmetic, although some versions are available which use floating point arithmetic and are more powerful. For faster applications FPGAs might be used. Beginning in 2007, multicore implementations of DSPs have started to emerge from companies including Freescale and Stream Processors, Inc. For faster applications with vast usage, ASICs might be designed specifically. For slow applications, a traditional slower processor such as a microcontroller may be adequate. Also a growing number of DSP applications are now being implemented on Embedded Systems using powerful PCs with a Multi-core processor.

Chapter- 7

Finite-state Machine

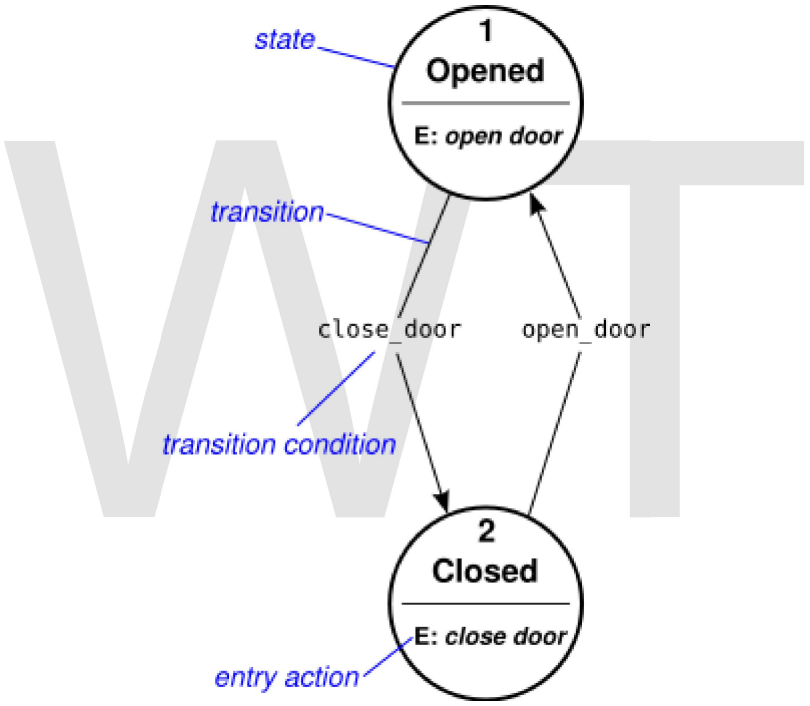


Fig. 1 Example of a simple finite state machine

A **finite-state machine** (**FSM**) or **finite-state automaton** (plural: *automata*), or simply a **state machine**, is a mathematical abstraction sometimes used to design digital logic or computer programs. It is a behavior model composed of a finite number of states, transitions between those states, and actions, similar to a flow graph in which one can inspect the way logic runs when certain conditions are met. It has finite internal memory, an input feature that reads symbols in a sequence, one at a time without going backward; and an output feature, which may be in the form of a user interface, once the model is implemented. The operation of an FSM begins from one of the states (called a *start state*), goes through transitions depending on input to different states and can end in any of those available, however only a certain set of states mark a successful flow of operation (called *accept states*).

Finite-state machines can solve a large number of problems, among which are electronic design automation, communication protocol design, parsing and other engineering applications. In biology and artificial intelligence research, state machines or hierarchies of state machines are sometimes used to describe neurological systems and in linguistics—to describe the grammars of natural languages.

Concepts and vocabulary

A current *state* is determined by past states of the system. As such, it can be said to record information about the past, i.e., it reflects the input changes from the system start to the present moment. The number and names of the states typically depend on the different possible states of the memory, e.g. if the memory is three bits long, there are 8 possible states. A *transition* indicates a state change and is described by a condition that would need to be fulfilled to enable the transition. An *action* is a description of an activity that is to be performed at a given moment. There are several action types:

Entry action
: which is performed *when entering* the state

Exit action
: which is performed *when exiting* the state

Input action
: which is performed depending on present state and input conditions

Transition action
: which is performed when performing a certain transition

An FSM can be represented using a state diagram (or state transition diagram) as in figure 1 above. Besides this, several state transition table types are used. The most common representation is shown below: the combination of current state (e.g. B) and input (e.g. Y) shows the next state (e.g. C). The complete actions information can be added only using footnotes. An FSM definition including the full actions information is possible using state tables.

State transition table

Current state → **Input ↓**	**State A**	**State B**	**State C**
Input X	...	...	...
Input Y	...	State C	...
Input Z	...	...	...

In addition to their use in modeling reactive systems presented here, finite state automata are significant in many different areas, including electrical engineering, linguistics, computer science, philosophy, biology, mathematics, and logic. Finite state machines are a class of automata studied in automata theory and the theory of computation. In computer science, finite state machines are widely used in modeling of application

behavior, design of hardware digital systems, software engineering, compilers, network protocols, and the study of computation and languages.

Classification

There are two different groups of state machines: Acceptors/Recognizers and Transducers.

Acceptors and recognizers

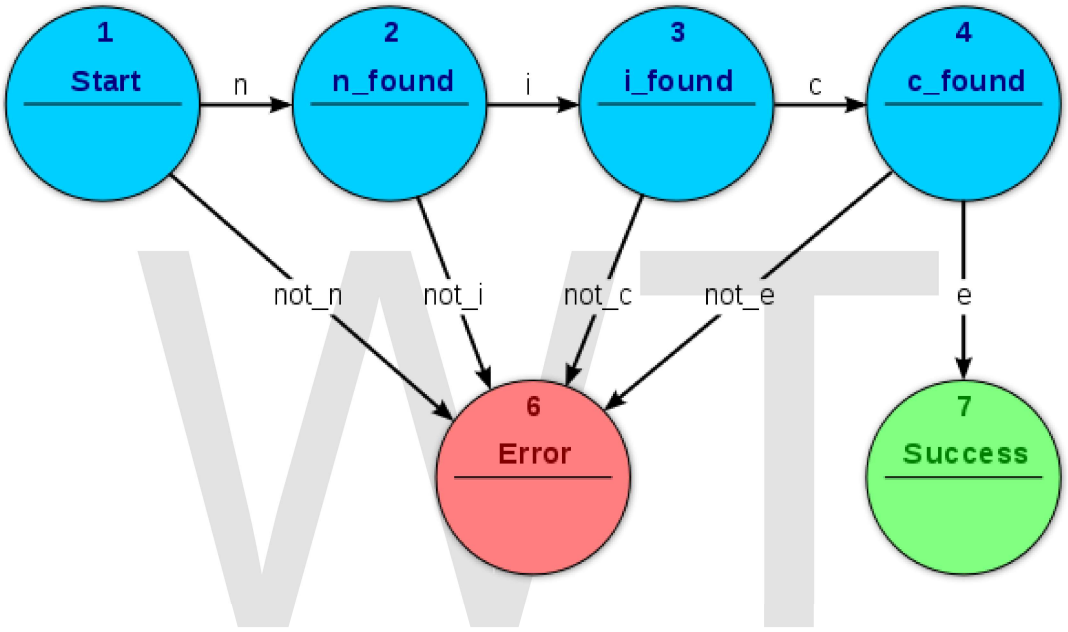


Fig. 2 Acceptor FSM: parsing the word "nice"

Acceptors and **recognizers** (also **sequence detectors**) produce a binary output, saying either *yes* or *no* to answer whether the input is accepted by the machine or not. All states of the FSM are said to be either accepting or not accepting. At the time when all input is processed, if the current state is an accepting state, the input is accepted; otherwise it is rejected. As a rule the input are symbols (characters); actions are not used. The example in figure 2 shows a finite state machine which accepts the word "nice". In this FSM the only accepting state is number 7.

The machine can also be described as defining a language, which would contain every word accepted by the machine but none of the rejected ones; we say then that the language is *accepted* by the machine. By definition, the languages accepted by FSMs are the regular languages—that is, a language is regular if there is some FSM that accepts it.

Start state

The start state is usually shown drawn with an arrow "pointing at it from any where" (Sipser (2006) p. 34).

Accept (or final) states

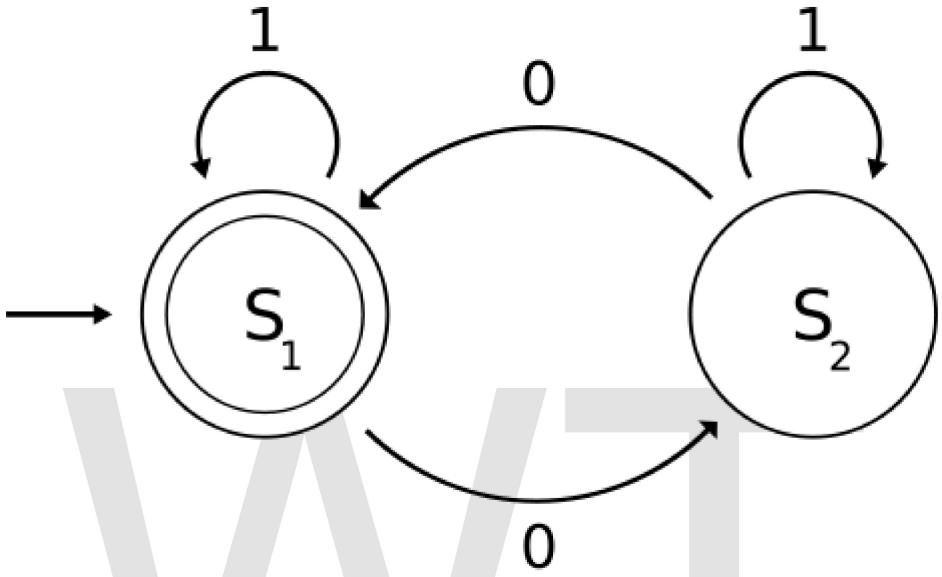


Fig. 3: Representation of a finite-state machine; this example shows one that determines whether a binary number has an odd or even number of 0's, where S_1 is an **accepting state**.

Accept states (also referred to as **accepting** or **final** states) are those at which the machine reports that the input string, as processed so far, is a member of the language it accepts. It is usually represented by a double circle.

An example of an accepting state appears in the diagram to the right: a deterministic finite automaton (DFA) that detects whether the binary input string contains an even number of 0's.

S_1 (which is also the start state) indicates the state at which an even number of 0's has been input. S_1 is therefore an accepting state. This machine will finish in an accept state, if the binary string contains an even number of 0's (including any binary string containing no 0's). Examples of strings accepted by this DFA are epsilon (the empty string), 1, 11, 11..., 00, 010, 1010, 10110, etc...

Transducers

Transducers generate output based on a given input and/or a state using actions. They are used for control applications and in the field of computational linguistics. Here two types are distinguished:

Moore machine
: The FSM uses only entry actions, i.e., output depends only on the state. The advantage of the Moore model is a simplification of the behaviour. Consider an elevator door. The state machine recognizes two commands: "command_open" and "command_close" which trigger state changes. The entry action (E:) in state "Opening" starts a motor opening the door, the entry action in state "Closing" starts a motor in the other direction closing the door. States "Opened" and "Closed" stop the motor when fully opened or closed. They signal to the outside world (e.g., to other state machines) the situation: "door is open" or "door is closed".

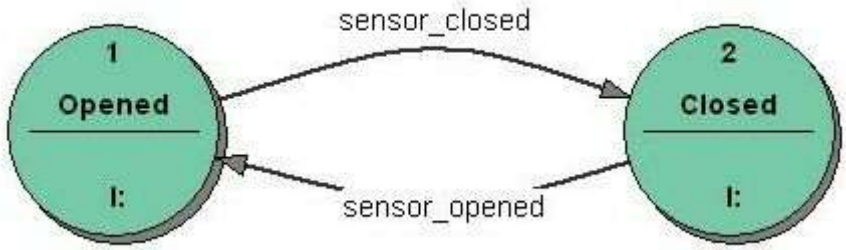


Fig. 4 Transducer FSM: Mealy model example

Mealy machine
: The FSM uses only input actions, i.e., output depends on input and state. The use of a Mealy FSM leads often to a reduction of the number of states. The example in figure 4 shows a Mealy FSM implementing the same behaviour as in the Moore example (the behaviour depends on the implemented FSM execution model and will work, e.g., for virtual FSM but not for event driven FSM). There are two input actions (I:): "start motor to close the door if command_close arrives" and "start motor in the other direction to open the door if command_open arrives". The "opening" and "closing" intermediate states are not shown.

In practice mixed models are often used.

More details about the differences and usage of Moore and Mealy models, including an executable example, can be found in the external technical note "Moore or Mealy model?"

Determinism

A further distinction is between **deterministic** (DFA) and **non-deterministic** (NFA, GNFA) automata. In deterministic automata, every state has exactly one transition for

each possible input. In non-deterministic automata, an input can lead to one, more than one or no transition for a given state. This distinction is relevant in practice, but not in theory, as there exists an algorithm which can transform any NFA into a more complex DFA with identical functionality.

The FSM with only one state is called a combinatorial FSM and uses only input actions. This concept is useful in cases where a number of FSM are required to work together, and where it is convenient to consider a purely combinatorial part as a form of FSM to suit the design tools.

UML state machines

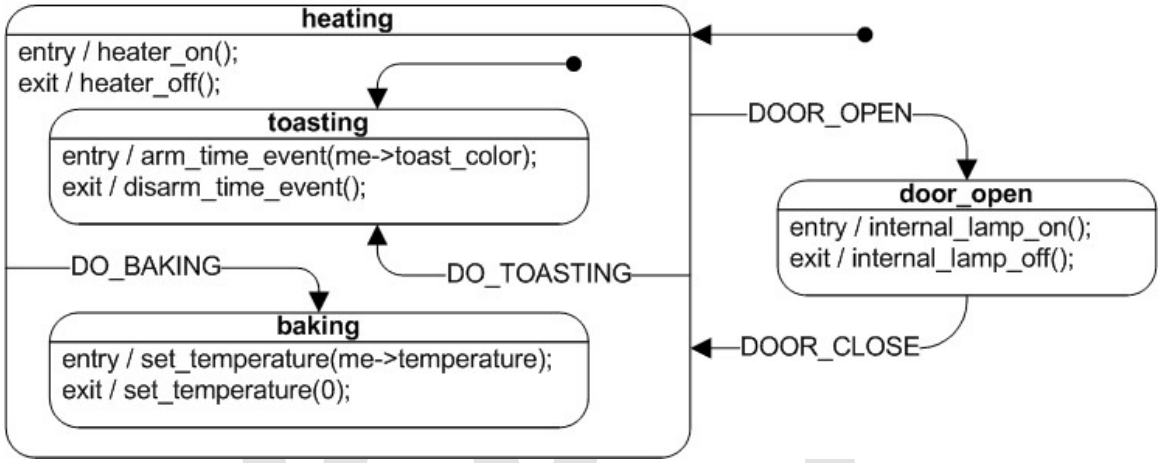


Fig. 5 UML state machine example (a toaster oven)

The Unified Modeling Language has a very rich semantics and notation for describing state machines. UML state machines overcome the limitations of traditional finite state machines while retaining their main benefits. UML state machines introduce the new concepts of hierarchically nested states and orthogonal regions, while extending the notion of actions. UML state machines have the characteristics of both Mealy machines and Moore machines. They support actions that depend on both the state of the system and the triggering event, as in Mealy machines, as well as entry and exit actions, which are associated with states rather than transitions, as in Moore machines.

Alternative semantics

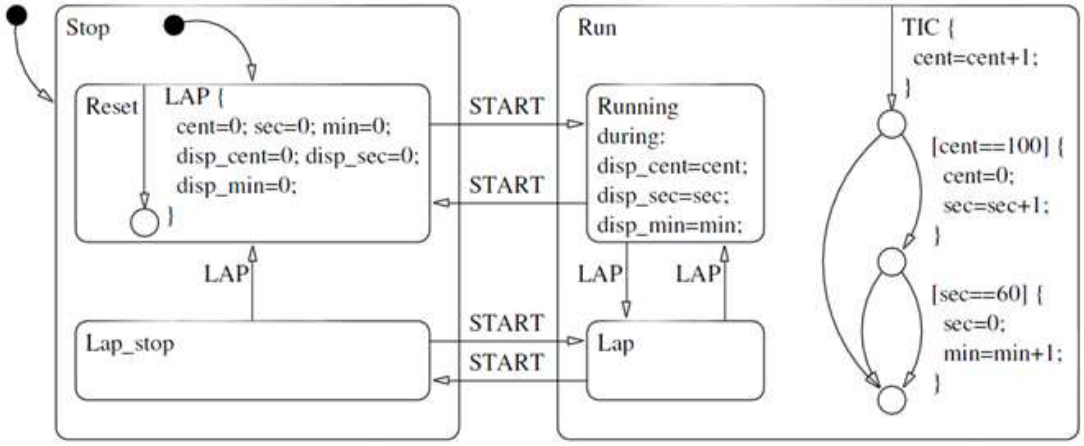


Fig. 6 Model of a simple stopwatch

There are other sets of semantics available to represent state machines. For example, there are tools for modeling and designing logic for embedded controllers. They combine hierarchical state machines, flow graphs, and truth tables into one language, resulting in a different formalism and set of semantics. Figure 6 illustrates this mix of state machines and flow graphs with a set of states to represent the state of a stopwatch and a flow graph to control the ticks of the watch. These charts, like Harel's original state machines, support hierarchically nested states, orthogonal regions, state actions, and transition actions.

FSM logic

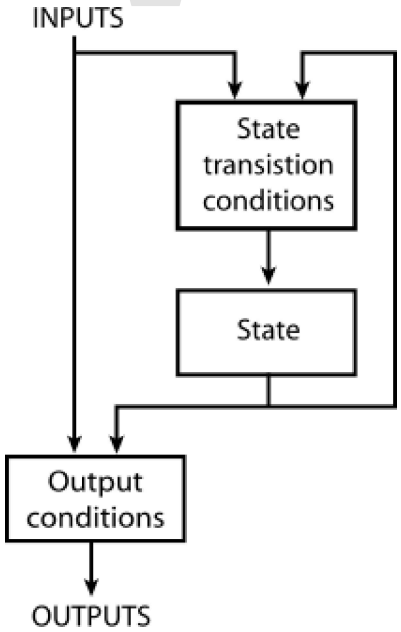


Fig. 7 FSM Logic (Mealy)

The next state and output of an FSM is a function of the input and of the current state. The FSM logic is shown in Figure 7.

Mathematical model

In accordance to the general classification, the following formal definitions are found:

- A *deterministic finite state machine* or *acceptor deterministic finite state machine* is a quintuple (Σ,S,s_0,δ,F), where:
  - Σ is the input alphabet (a finite, non-empty set of symbols).
  - S is a finite, non-empty set of states.
  - s_0 is an initial state, an element of S.
  - δ is the state-transition function: $\delta : S \times \Sigma \rightarrow S$(in a nondeterministic finite state machine it would be $\delta : S \times \Sigma \rightarrow \mathcal{P}(S)$, i.e., δ would return a set of states).
  - F is the set of final states, a (possibly empty) subset of S.

For both deterministic and non-deterministic FSMs, it is conventional to allow δ to be a partial function, i.e. $\delta(q,x)$ does not have to be defined for every combination of $q \in S$and $x \in \Sigma$. If an FSM M is in a state q, the next symbol is x and $\delta(q,x)$ is not defined, then M can announce an error (i.e. reject the input).

- A *finite state transducer* is a sextuple (Σ,Γ,S,s_0,δ,ω), where:
  - Σ is the input alphabet (a finite non empty set of symbols).
  - Γ is the output alphabet (a finite, non-empty set of symbols).
  - S is a finite, non-empty set of states.
  - s_0 is the initial state, an element of S. In a nondeterministic finite state machine, s_0 is a set of initial states.
  - δ is the state-transition function: $\delta : S \times \Sigma \rightarrow S$.
  - ω is the output function.

If the output function is a function of a state and input alphabet ($\omega : S \times \Sigma \rightarrow \Gamma$) that definition corresponds to the **Mealy model**, and can be modelled as a Mealy machine. If the output function depends only on a state ($\omega : S \rightarrow \Gamma$) that definition corresponds to the **Moore model**, and can be modelled as a Moore machine. A finite-state machine with no output function at all is known as a semiautomaton or transition system.

Optimization

Optimizing an FSM means finding the machine with the minimum number of states that performs the same function. The fastest known algorithm doing this is the Hopcroft minimization algorithm. Other techniques include using an implication table, or the Moore reduction procedure. Additionally, acyclic FSAs can be optimized using a simple bottom up algorithm.

Implementation

Hardware applications

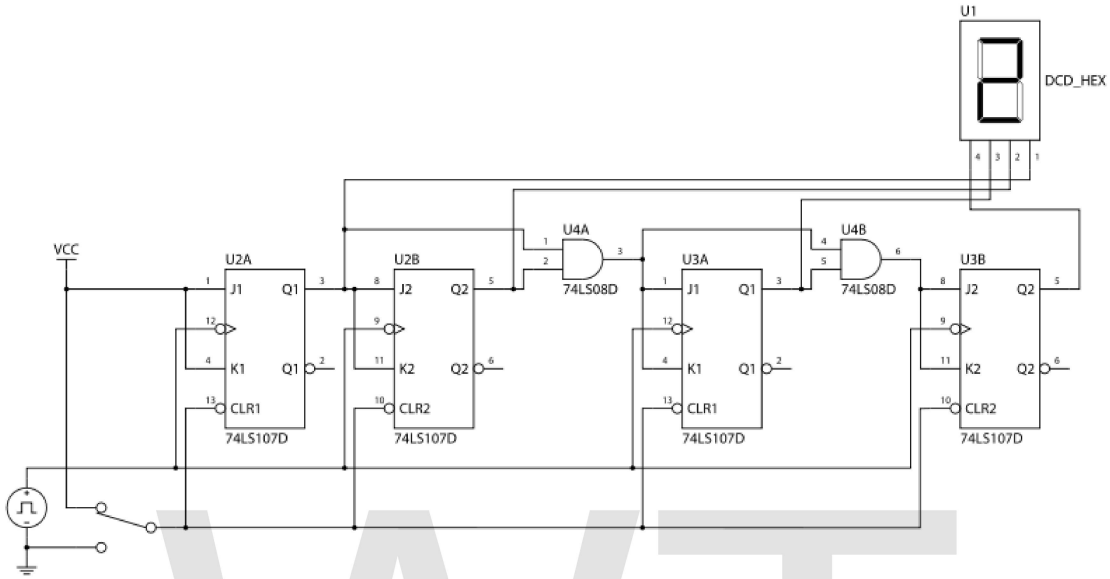


Fig. 8 The circuit diagram for a 4-bit TTL counter, a type of state machine

In a digital circuit, an FSM may be built using a programmable logic device, a programmable logic controller, logic gates and flip flops or relays. More specifically, a hardware implementation requires a register to store state variables, a block of combinational logic which determines the state transition, and a second block of combinational logic that determines the output of an FSM. One of the classic hardware implementations is the Richards controller.

Mealy and Moore machines produce logic with asynchronous output, because there is a propagation delay between the flip-flop and output. This causes slower operating frequencies in FSM. A Mealy or Moore machine can be convertible to a FSM which output is directly from a flip-flop, which makes the FSM run at higher frequencies. This kind of FSM is sometimes called Medvedev FSM. A counter is the simplest form of this kind of FSM.

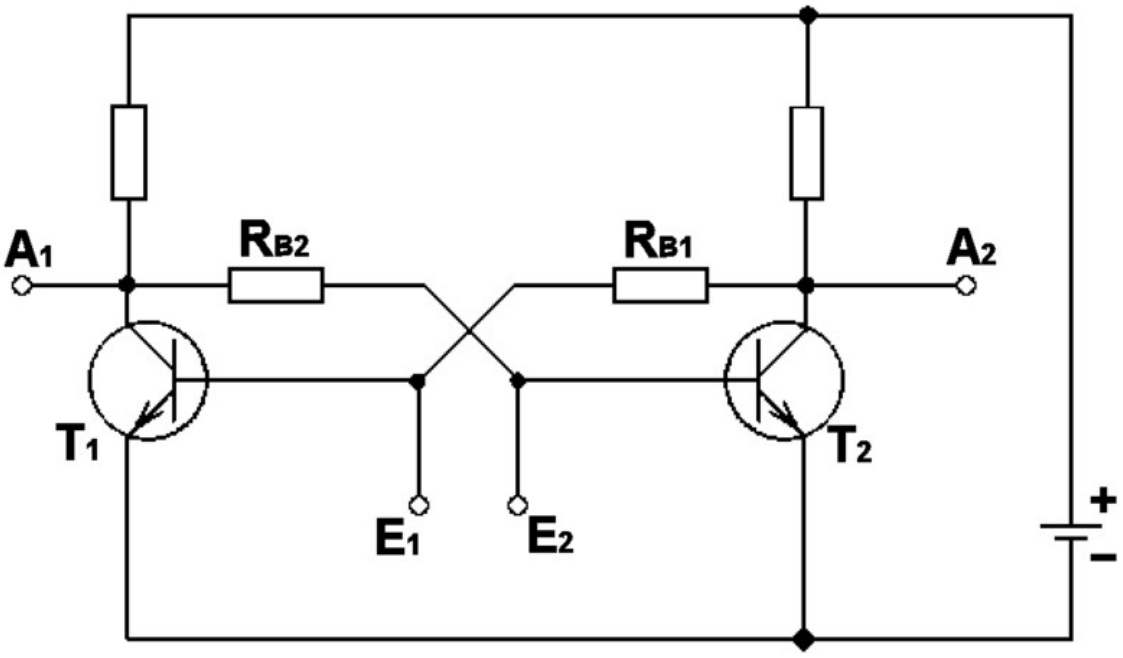
Software applications

The following concepts are commonly used to build software applications with finite state machines:

- Automata-based programming
- Event driven FSM
- Virtual FSM (VFSM)

Chapter- 8

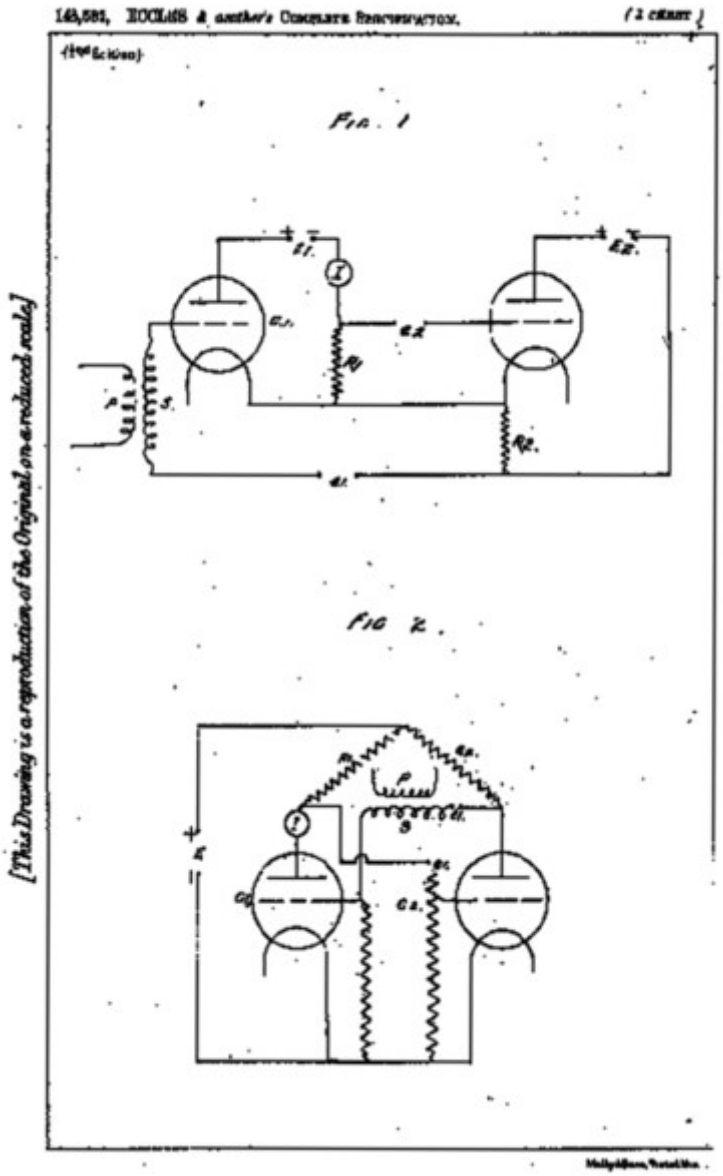
Flip-flop



A traditional flip-flop circuit based on bipolar junction transistors

In electronics, a **flip-flop** is a circuit that has two stable states and can be used to store state information. The circuit can be made to change state by signals applied to one or more control inputs and will have one or two outputs. A circuit incorporating flip-flops has the attribute of *state*; its output depends not only on its current input, but also on its previous inputs. Such a circuit is described as sequential logic. Where a single input is provided, the circuit changes state every time a pulse appears on the input signal. Since the flip-flop retains the state after the signal pulses are removed, one type of flip-flop circuit is also called a "latch". Other types of flip-flops may have inputs that set a particular state, set the opposite state, or change states, depending on which input is pulsed.

Flip-flops are used as data storage elements, for counting of pulses, and for synchronizing randomly-timed input signals to some reference timing signal. Flip-flops are a fundamental building block of digital electronics systems used in computers, communications, and many other types of systems.



Flip-flop schematics from the Eccles and Jordan patent filed 1918, one drawn as a cascade of amplifiers with a positive feedback path, and the other as a symmetric cross-coupled pair

History

The first electronic flip-flop was invented in 1918 by William Eccles and F. W. Jordan. It was initially called the *Eccles–Jordan trigger circuit* and consisted of two active

elements (vacuum tubes). Such circuits and their transistorized versions were common in computers even after the introduction of integrated circuits, though flip-flops made from logic gates are also common now.

Early flip-flops were known variously as trigger circuits or multivibrators. A multivibrator is a two-state circuit; they come in several varieties, based on whether each state is stable or not: an *astable multivibrator* is not stable in either state, so it acts as a relaxation oscillator; a *monostable multivibrator* makes a pulse while in the unstable state, then returns to the stable state, and is known as a *one-shot*; a *bistable multivibrator* has two stable states, and this is the one usually known as a flip-flop. However, this terminology has been somewhat variable, historically. For example:

- 1942 – multivibrator implies astable: "The multivibrator circuit (Fig. 7-6) is somewhat similar to the flip-flop circuit, but the coupling from the anode of one valve to the grid of the other is by a condenser only, so that the coupling is not maintained in the steady state."
- 1942 – multivibrator as a particular flip-flop circuit: "Such circuits were known as 'trigger' or 'flip-flop' circuits and were of very great importance. The earliest and best known of these circuits was the multivibrator."
- 1943 – flip-flop as one-shot pulse generator: "It should be noted that an essential difference between the two-valve flip-flop and the multivibrator is that the flip-flop has one of the valves biased to cutoff."
- 1949 – monostable as flip-flop: "Monostable multivibrators have also been called 'flip-flops'."
- 1949 – monostable as flip-flop: "... a flip-flop is a monostable multivibrator and the ordinary multivibrator is an astable multivibrator."

According to P. L. Lindley, a JPL engineer, the flip-flop types discussed below (RS, D, T, JK) were first discussed in a 1954 UCLA course on computer design by Montgomery Phister, and then appeared in his book *Logical Design of Digital Computers.* Lindley was at the time working at Hughes Aircraft under Dr. Eldred Nelson, who had coined the term JK for a flip-flop which changed states when both inputs were on. The other names were coined by Phister. They differ slightly from some of the definitions given below. Lindley explains that he heard the story of the JK flip-flop from Dr. Eldred Nelson, who is responsible for coining the term while working at Hughes Aircraft. Flip-flops in use at Hughes at the time were all of the type that came to be known as J-K. In designing a logical system, Dr. Nelson assigned letters to flip-flop inputs as follows: #1: A & B, #2: C & D, #3: E & F, #4: G & H, #5: J & K.

Implementation

Flip-flops can be either simple (transparent) or clocked; the transparent ones are commonly called latches.

The word *latch* is mainly used for storage elements, while clocked devices are described as **flip-flops**.

Simple flip-flops can be built around a pair of cross-coupled inverting elements: vacuum tubes, bipolar transistors, field effect transistors, inverters, and inverting logic gates have all been used in practical circuits. Clocked devices are specially designed for synchronous systems; such devices ignore their inputs except at the transition of a dedicated clock signal (known as clocking, pulsing, or strobing). Clocking causes the flip-flop to either change or retain its output signal based upon the values of the input signals at the transition. Some flip-flops change output on the rising edge of the clock, others on the falling edge.

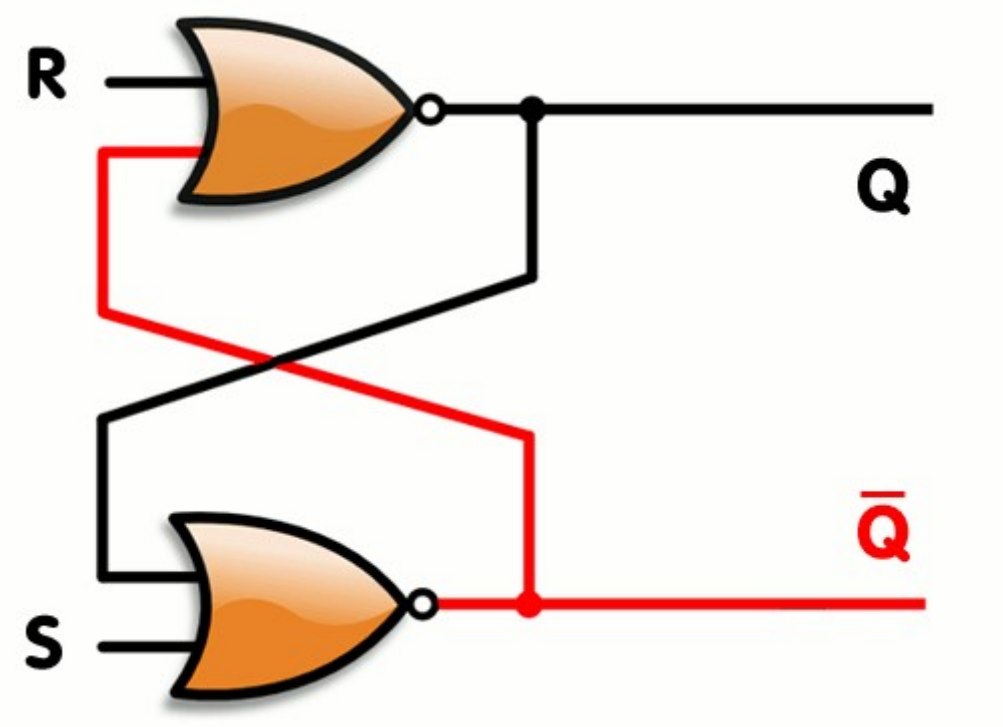
Since the elementary amplifying stages are inverting, two stages can be connected in succession (as a cascade) to form the needed non-inverting amplifier. In this configuration, each amplifier may be considered as an active inverting feedback network for the other inverting amplifier. Thus the two stages are connected in a non-inverting loop although the circuit diagram is usually drawn as a symmetric cross-coupled pair (both the drawings are initially introduced in the Eccles–Jordan patent).

Flip-flop types

Flip-flops can be divided into common types: the **RS** ("set-reset"), **D** ("data" or "delay"), **T** ("toggle"), and **JK** types are the common ones. The behavior of a particular type can be described by what is termed the characteristic equation, which derives the "next" (i.e., after the next clock pulse) output, Q_{next}, in terms of the input signal(s) and/or the current output, Q.

Simple set-reset latches

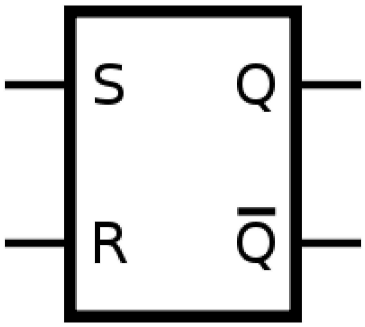
SR NOR latch



An RS latch, constructed from a pair of cross-coupled NOR gates. Red and black mean logical '1' and '0', respectively.

When using static gates as building blocks, the most fundamental latch is the simple *SR latch*, where S and R stand for *set* and *reset*. It can be constructed from a pair of cross-coupled NOR logic gates. The stored bit is present on the output marked Q.

While the S and R inputs are both low, feedback maintains the Q and Q outputs in a constant state, with Q the complement of Q. If S (*Set*) is pulsed high while R (*Reset*) is held low, then the Q output is forced high, and stays high when S returns to low; similarly, if R is pulsed high while S is held low, then the Q output is forced low, and stays low when R returns to low.



The symbol for an SR NOR latch

SR latch operation

S	R	Action
0	0	No Change
0	1	Q = 0
1	0	Q = 1
1	1	Restricted combination

The R = S = 1 combination is called a **restricted combination** or a **forbidden state** because, as both NOR gates then output zeros, it breaks the logical equation Q = **not** Q. The combination is also inappropriate in circuits where *both* inputs may go low *simultaneously* (i.e. a transition from *restricted* to *keep*). The output would lock at either 1 or 0 depending on the propagation time relations between the gates (a race condition). In certain implementations, it could also lead to longer ringings (damped oscillations) before the output settles, and thereby result in undetermined values (errors) in high-frequency digital circuits. Although this condition is usually avoided, it can be useful in some applications.

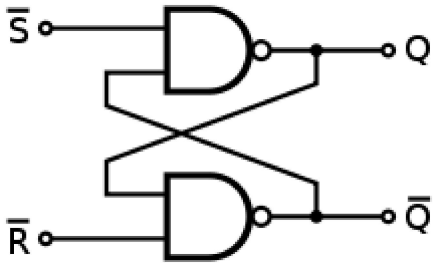
To overcome the restricted combination, one can add gates to the inputs that would convert `(S,R) = (1,1)` to one of the non-restricted combinations. That can be:

- Q = 1 (1,0) – referred to as an *S-latch*
- Q = 0 (0,1) – referred to as an *R-latch*
- Keep state (0,0) – referred to as an *E-latch*

Alternatively, the restricted combination can be made to *toggle* the output. The result is the JK latch.

Characteristic: Q+ = R'Q + R'S or Q+ = R'Q + S.

SR NAND latch

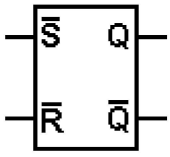


SR latch

This is an alternate model of the simple SR latch built with NAND (not AND) logic gates. *Set* and *reset* now become active low signals, denoted S and R respectively. Otherwise, operation is identical to that of the SR latch. Historically, SR-latches have been predominant despite the notational inconvenience of active-low inputs. This is because NAND gates are cheaper to produce than NOR gates in the diode-transistor logic (DTL), transistor-transistor logic (TTL) families, and complementary metal–oxide semiconductor (CMOS) logic families.

SR latch operation

S	R	Action
0	0	Restricted combination
0	1	Q = 1
1	0	Q = 0
1	1	No Change



Symbol for an SR NAND latch

JK latch

The JK latch is much less used than the JK flip-flop. The JK latch follows the following state table:

JK latch truth table

J	K	Q_{next}	Comment
0	0	Q	No change
0	1	0	Reset
1	0	1	Set

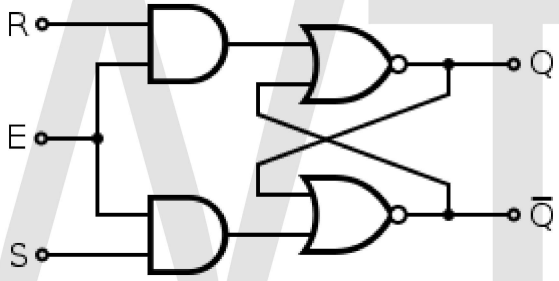
1 1 Q Toggle

Hence, the JK latch is an SR latch that is made to *toggle* its output when passed the restricted combination of 11. Unlike the JK Flip-Flop, in the JK latch, this is not a useful state because the speed of the toggling is not directed by a clock.

Gated latches and conditional transparency

Latches are designed to be *transparent.* That is, input signal changes cause immediate changes in output; when several *transparent* latches follow each other, using the same clock signal, signals can propagate through all of them at once. Alternatively, additional logic can be added to a simple transparent latch to make it *non-transparent* or *opaque* when another input (an "enable" input) is not asserted. By following a *transparent-high* latch with a *transparent-low* (or *opaque-high*) latch, a master–slave flip-flop is implemented.

Gated SR latch



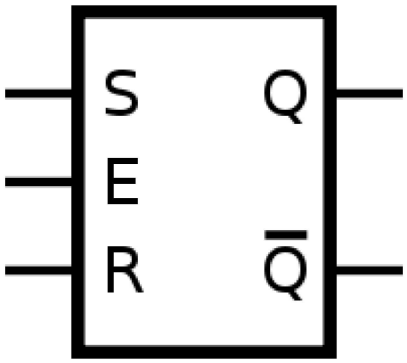
A gated SR latch circuit diagram constructed from NOR gates.

A *synchronous SR latch* (sometimes *clocked SR flip-flop*) can be made by adding a second level of NAND gates to the inverted SR latch (or a second level of AND gates to the direct SR latch). The extra gates further invert the inputs so the simple SR latch becomes a gated SR latch (and a simple SR latch would transform into a gated SR latch with inverted enable).

With E high (*enable* true), the signals can pass through the input gates to the encapsulated latch; all signal combinations except for (0,0) = *hold* then immediately reproduce on the (Q,Q) output, i.e. the latch is *transparent*.

With E low (*enable* false) the latch is *closed (opaque)* and remains in the state it was left the last time E was high.

The *enable* input is sometimes a clock signal, but more often a read or write strobe.

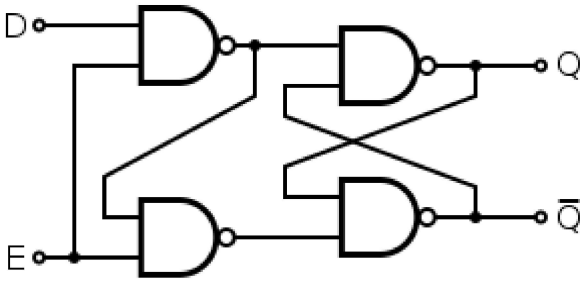


Symbol for a gated SR latch

Gated SR latch operation

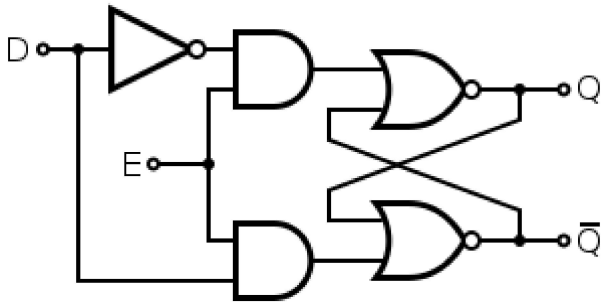
E/C	Action
0	No action (keep state)
1	The same as non-clocked SR latch

Gated D latch



A D-type transparent latch based on SR NAND latch

- **D** Input
- **E** Enable/clock
- **Q** Output
- **Q** Inverse of Q



A gated D latch based on SR NOR latch

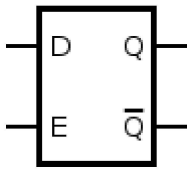
This latch exploits the fact that in the two active input combinations (01 and 10) of a gated SR latch R is the complement of S. The input NAND stage converts the two D input states (0 and 1) to these two input combinations for the next SR latch by inverting the data input signal. The low state of the *enable* signal produces the inactive "11" combination. Thus a gated D-latch may be considered as a *one-input synchronous SR latch*. This configuration prevents from applying the restricted combination to the inputs. It is also known as *transparent latch*, *data latch*, or simply *gated latch*. It has a *data* input and an *enable* signal (sometimes named *clock*, or *control*). The word *transparent* comes from the fact that, when the enable input is on, the signal propagates directly through the circuit, from the input D to the output Q.

Transparent latches are typically used as I/O ports or in asynchronous systems, or in synchronous two-phase systems (synchronous systems that use a two-phase clock), where two latches operating on different clock phases prevent data transparency as in a master–slave flip-flop.

Latches are available as integrated circuits, usually with multiple latches per chip. For example, 74HC75 is a quadruple transparent latch in the 7400 series.

Gated D latch truth table

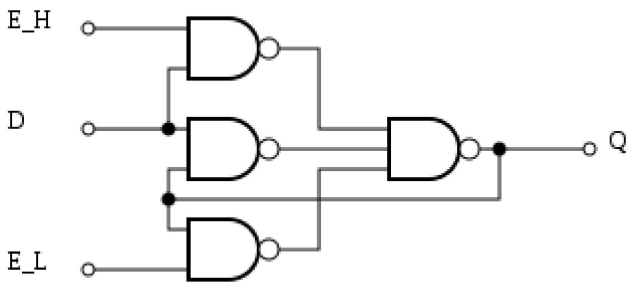
E/C	D	Q	Q	Comment
0	X	Q_{prev}	Q_{prev}	No change
1	0	0	1	Reset
1	1	1	0	Set



Symbol for a gated D latch

The truth table shows that when the *e*nable/*c*lock input is 0, the D input has no effect on the output. When E/C is high, the output equals D.

Earle latch



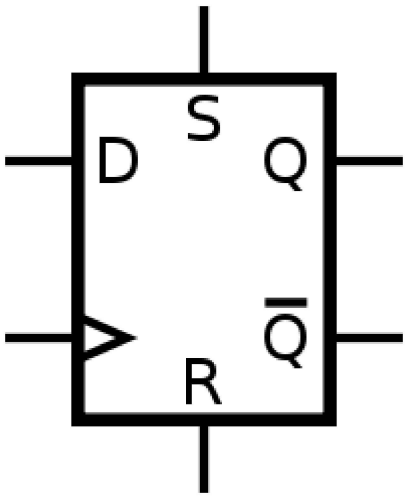
Earle latch uses complementary Enable inputs: Enable active Low (E_L) and Enable active H (E_H)

The classic gated latch designs have some undesirable characteristics. They require double-rail logic or an inverter. The input-to-output propagation may take up to three gate delays. The input-to-output propagation is not constant – some outputs take two gate delays while others take three.

Designers looked for alternatives. A successful alternative is the Earle latch. It requires only a single data input, and its output takes a constant two gate delays. In addition, the two gate levels of the Earle latch can be merged with the last two gate levels of the circuits driving the latch. Merging the latch function can implement the latch with no additional gate delays.

The Earle latch is hazard free. If the middle NAND gate is omitted, then one gets the **polarity hold latch**, which is commonly used because it demands less logic. Intentionally skewing the clock signal can avoid the hazard.

D flip-flop



D flip-flop symbol

The D flip-flop is the most common flip-flop in use today. It is better known as *data* or *delay* flip-flop (as its output Q looks like a delay of input D).

The Q output takes on the state of the D input at the moment of a positive edge at the clock pin (or negative edge if the clock input is active low). It is called the D flip-flop for this reason, since the output takes the value of the D input or *data* input, and *delays* it by one clock cycle. The D flip-flop can be interpreted as a primitive memory cell, zero-order hold, or delay line. Whenever the clock pulses, the value of Q_{next} is D and Q_{prev} otherwise.

Truth table:

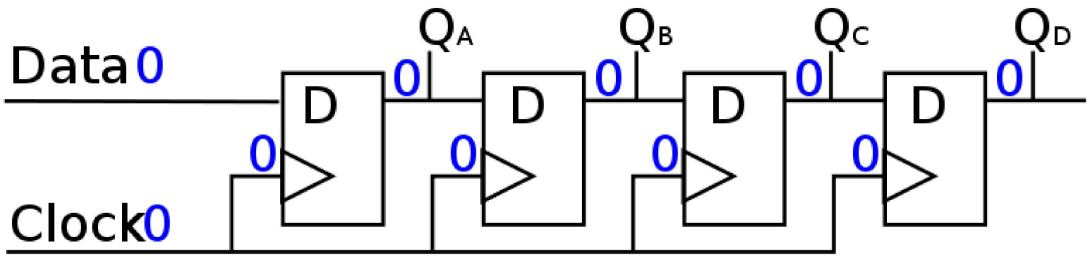
Clock	**D**	**Q**	**Q_{prev}**
Rising edge	0	0	X
Rising edge	1	1	X
Non-Rising	X	Q_{prev}	

('X' denotes a *Don't care* condition, meaning the signal is irrelevant)

Most D-type flip-flops in ICs have the capability to be forced to the set or reset state (which ignores the D and clock inputs), much like an SR flip-flop. Usually, the illegal S = R = 1 condition is resolved in D-type flip-flops. By setting S = R = 0, the flip-flop can be used as described above.

Inputs				**Outputs**	
S	**R**	**D**	**>**	**Q**	**Q'**
0	1	X	X	0	1

1	0	X	X	1	0
1	1	X	X	1	1

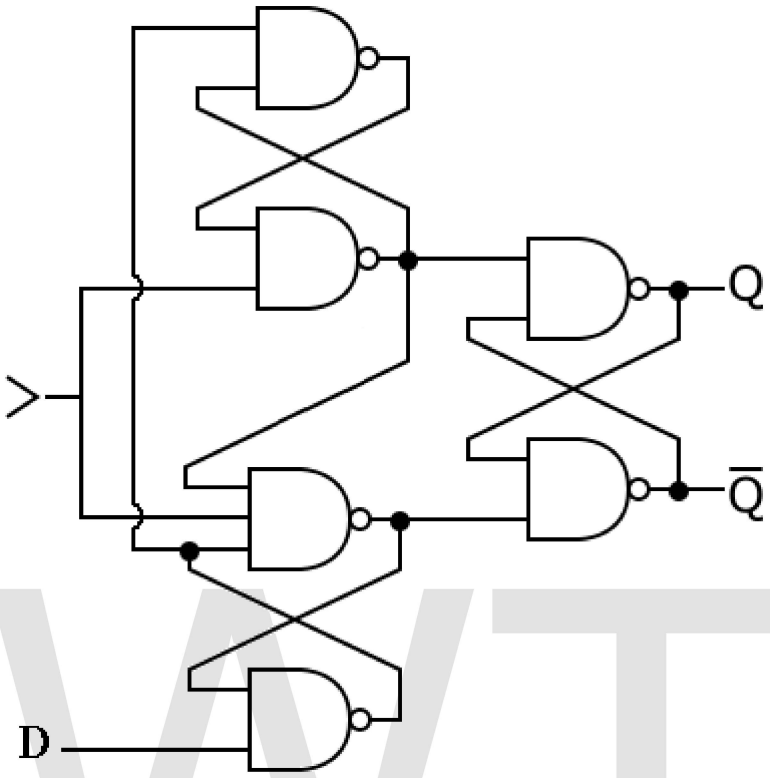


4-bit serial-in, serial-out (SISO) shift register

These flip-flops are very useful, as they form the basis for shift registers, which are an essential part of many electronic devices. The advantage of the D flip-flop over the D-type "transparent latch" is that the signal on the D input pin is captured the moment the flip-flop is clocked, and subsequent changes on the D input will be ignored until the next clock event. An exception is that some flip-flops have a "reset" signal input, which will reset Q (to zero), and may be either asynchronous or synchronous with the clock.

The above circuit shifts the contents of the register to the right, one bit position on each active transition of the clock. The input X is shifted into the leftmost bit position.

Classical positive-edge-triggered D flip-flop



A positive-edge-triggered D flip-flop

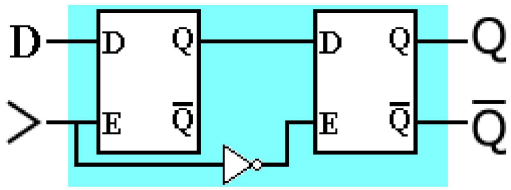
This clever circuit consists of two stages implemented by SR NAND latches. The input stage (the two latches on the left) processes the clock and data signals to ensure correct input signals for the output stage (the single latch on the right). If the clock is low, both the output signals of the input stage are high regardless of the data input; the output latch is unaffected and it stores the previous state. When the clock signal changes from low to high, only one of the output voltages (depending on the data signal) goes low and sets/resets the output latch: if D = 0, the lower output becomes low; if D = 1, the upper output becomes low. If the clock signal continues staying high, the outputs keep their states regardless of the data input and force the output latch to stay in the corresponding state as the input logical zero remains active while the clock is high. Hence the role of the output latch is to store the data only while the clock is low.

The circuit is closely related to the gated D latch as both the circuits convert the two D input states (0 and 1) to two input combinations (01 and 10) for the output SR latch by inverting the data input signal (both the circuits split the single D signal in two complementary S and R signals). The difference is that in the gated D latch simple NAND logical gates are used while in the positive-edge-triggered D flip-flop SR NAND latches are used for this purpose. The role of these latches is to "lock" the active output producing low voltage (a logical zero); thus the positive-edge-triggered D flip-flop can be thought as of a gated D latch with latched input gates.

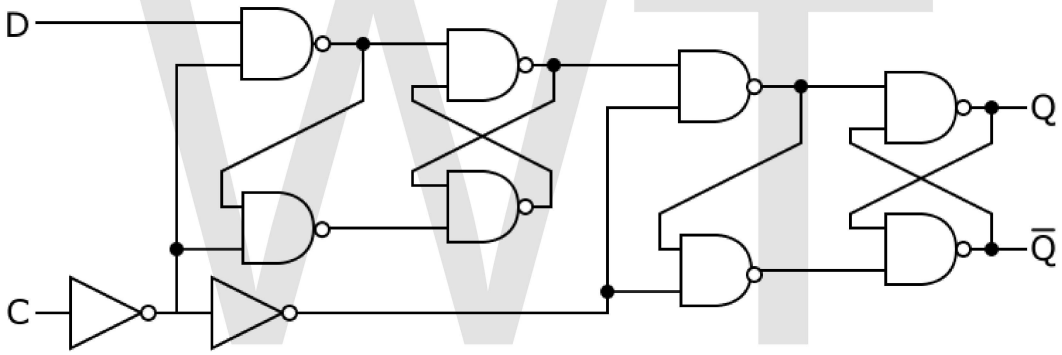
Master–slave pulse-triggered D flip-flop

A master–slave D flip-flop is created by connecting two gated D latches in series, and inverting the *enable* input to one of them. It is called master–slave because the second latch in the series only changes in response to a change in the first (master) latch.

The term *pulse-triggered* means that data is entered on the rising edge of the clock pulse, but the output does not reflect the change until the falling edge of the clock pulse.



A master–slave D flip-flop. It responds on the negative edge of the *enable* input (usually a clock)

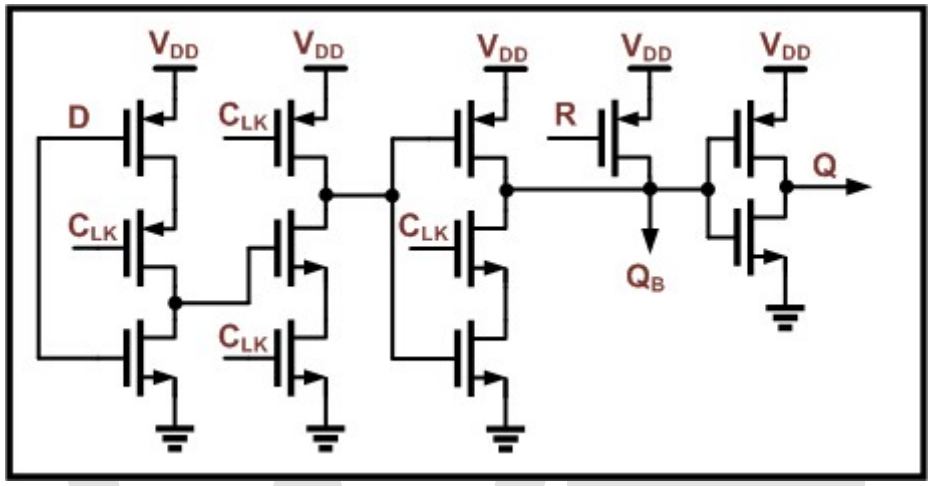


An implementation of a master–slave D flip-flop that is triggered on the positive edge of the clock

For a positive-edge triggered master–slave D flip-flop, when the clock signal is low (logical 0) the "enable" seen by the first or "master" D latch (the inverted clock signal) is high (logical 1). This allows the "master" latch to store the input value when the clock signal transitions from low to high. As the clock signal goes high (0 to 1) the inverted "enable" of the first latch goes low (1 to 0) and the value seen at the input to the master latch is "locked". Nearly simultaneously, the twice inverted "enable" of the second or "slave" D latch transitions from low to high (0 to 1) with the clock signal. This allows the signal captured at the rising edge of the clock by the now "locked" master latch to pass through the "slave" latch. When the clock signal returns to low (1 to 0), the output of the "slave" latch is "locked", and the value seen at the last rising edge of the clock is held while the "master" latch begins to accept new values in preparation for the next rising clock edge.

By removing the leftmost inverter in the circuit at side, a D-type flip flop that strobes on the *falling edge* of a clock signal can be obtained. This has a truth table like this:

D	Q	>	Q_{next}
0	X	Falling	0
1	X	Falling	1



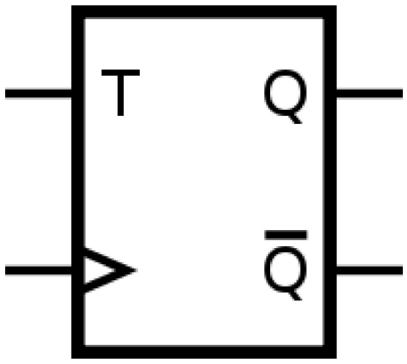
A CMOS IC implementation of a True Single Phase edge-triggered flip-flop with reset

Edge-triggered dynamic D flip-flop

A more efficient way to make a D flip-flop is not so easy to understand, but it works the same way. While the master–slave D flip-flop is also triggered on the edge of a clock, its components are each triggered by clock levels. The "edge-triggered D flip-flop" does not have the master–slave properties.

Edge-triggered D flip-flops are often implemented in integrated high-speed operations using dynamic logic. This means that the digital output is stored on parasitic device capacitance while the device is not transitioning. This design of dynamic flip flops also enable simple resetting since the reset operation can be performed by simply discharging one or more internal nodes. A common dynamic flip-flop variety is the True Single Phase Clock (TSPC) which performs the flip flop operation with little power and at high speeds. However these types of dynamic flip-flops will not work at static or low clocked speeds, since given enough time the parasitic capacitance will discharge through leakage paths and will cause the logic levels to enter invalid states.

T flip-flop



A circuit symbol for a T-type flip-flop

If the T input is high, the T flip-flop changes state ("toggles") whenever the clock input is strobed. If the T input is low, the flip-flop holds the previous value. This behavior is described by the characteristic equation:

$$Q_{next} = T \oplus Q = T\overline{Q} + \overline{T}Q$$ (expanding the XOR operator)

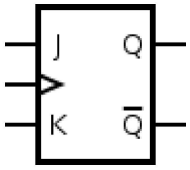
and can be described in a truth table:

T flip-flop operation

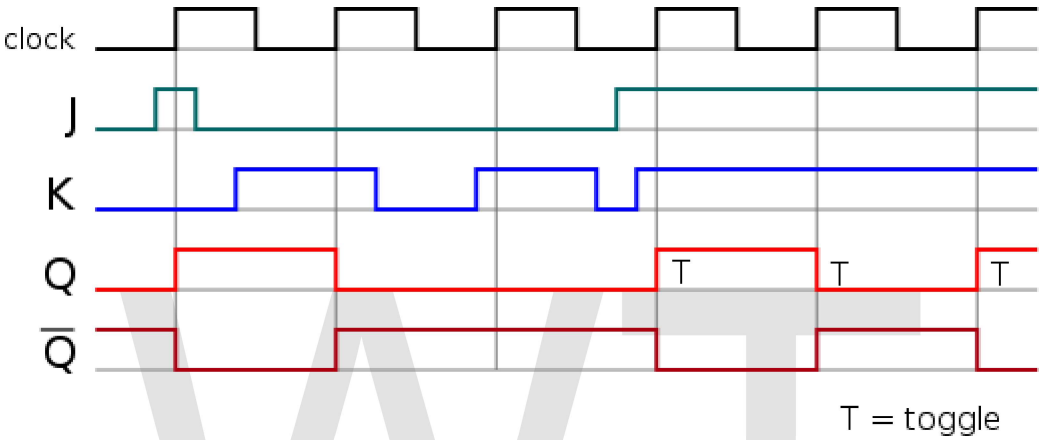
Characteristic table				**Excitation table**			
T	***Q***	Q_{next}	**Comment**	***Q***	Q_{next}	***T***	**Comment**
0	0	0	hold state (no clk)	0	0	0	No change
0	1	1	hold state (no clk)	1	1	0	No change
1	0	1	toggle	0	1	1	Complement
1	1	0	toggle	1	0	1	Complement

When T is held high, the toggle flip-flop divides the clock frequency by two; that is, if clock frequency is 4 MHz, the output frequency obtained from the flip-flop will be 2 MHz. This "divide by" feature has application in various types of digital counters. A T flip-flop can also be built using a JK flip-flop (J & K pins are connected together and act as T) or D flip-flop (T input and $Q_{previous}$ is connected to the D input through an XOR gate). A T flip-flop can also be built using an edge-triggered D flip-flop with its D input fed from its own inverted output.

JK flip-flop



A circuit symbol for a positive-edge-triggered JK flip-flop



JK flip-flop timing diagram

The JK flip-flop augments the behavior of the SR flip-flop (J=Set, K=Reset) by interpreting the S = R = 1 condition as a "flip" or toggle command. Specifically, the combination J = 1, K = 0 is a command to set the flip-flop; the combination J = 0, K = 1 is a command to reset the flip-flop; and the combination J = K = 1 is a command to toggle the flip-flop, i.e., change its output to the logical complement of its current value. Setting J = K = 0 does NOT result in a D flip-flop, but rather, will hold the current state. To synthesize a D flip-flop, simply set K equal to the complement of J. The JK flip-flop is therefore a universal flip-flop, because it can be configured to work as an SR flip-flop, a D flip-flop, or a T flip-flop.

NOTE: The flip-flop is positive-edge triggered (rising clock pulse) as seen in the timing diagram.

The characteristic equation of the JK flip-flop is:

$$Q_{next} = J\overline{Q} + \overline{K}Q$$

and the corresponding truth table is:

JK Flip Flop operation

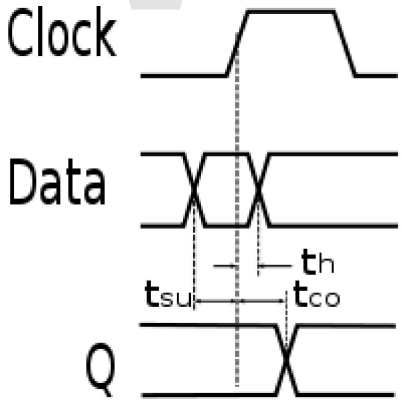
Characteristic table				Excitation table				
J	**K**	**Q_{next}**	**Comment**	**Q**	**Q_{next}**	**J**	**K**	**Comment**
0	0	Q	hold state	0	0	0	X	No change
0	1	0	reset	0	1	1	X	Set
1	0	1	set	1	0	X	1	Reset
1	1	Q	toggle	1	1	X	0	No change

Metastability

Flip-flops are prone to a problem called metastability, which can happen when two inputs, such as data and clock or clock and reset, are changing at about the same time, such that the resulting state would depend on the order of the input events. When the order is not clear, within appropriate timing constraints, the result is that the output may behave unpredictably, taking many times longer than normal to settle to one state or the other, or even oscillating several times before settling. Theoretically, the time to settle down is not bounded. In a computer system, this metastability can cause corruption of data or a program crash, if the state is not stable before another circuit uses its value; in particular, if two different logical paths use the output of a flip-flop, one path can interpret it as a 0 and the other as a 1 when it has not resolved to stable state, putting the machine into an inconsistent state.

Timing considerations

Setup and hold times



Flip-flop setup, hold and clock-to-output timing parameters

Setup time is the minimum amount of time the data signal should be held steady **before** the clock event so that the data are reliably sampled by the clock. This applies to synchronous circuits such as the flip-flop.

Hold time is the minimum amount of time the data signal should be held steady **after** the clock event so that the data are reliably sampled. This applies to synchronous circuits such as the flip-flop.

To summarize: Setup time -> Clock flank -> Hold time.

The metastability in flip-flops can be avoided by ensuring that the data and control inputs are held valid and constant for specified periods before and after the clock pulse, called the **setup time** (t_{su}) and the **hold time** (t_h) respectively. These times are specified in the data sheet for the device, and are typically between a few nanoseconds and a few hundred picoseconds for modern devices.

Unfortunately, it is not always possible to meet the setup and hold criteria, because the flip-flop may be connected to a real-time signal that could change at any time, outside the control of the designer. In this case, the best the designer can do is to reduce the probability of error to a certain level, depending on the required reliability of the circuit. One technique for suppressing metastability is to connect two or more flip-flops in a chain, so that the output of each one feeds the data input of the next, and all devices share a common clock. With this method, the probability of a metastable event can be reduced to a negligible value, but never to zero. The probability of metastability gets closer and closer to zero as the number of flip-flops connected in series is increased.

So-called metastable-hardened flip-flops are available, which work by reducing the setup and hold times as much as possible, but even these cannot eliminate the problem entirely. This is because metastability is more than simply a matter of circuit design. When the transitions in the clock and the data are close together in time, the flip-flop is forced to decide which event happened first. However fast we make the device, there is always the possibility that the input events will be so close together that it cannot detect which one happened first. It is therefore logically impossible to build a perfectly metastable-proof flip-flop.

Propagation delay

Another important timing value for a flip-flop (F/F) is the clock-to-output delay (common symbol in data sheets: t_{CO}) or propagation delay (t_P), which is the time the flip-flop takes to change its output after the clock edge. The time for a high-to-low transition (t_{PHL}) is sometimes different from the time for a low-to-high transition (t_{PLH}).

When cascading F/Fs which share the same clock (as in a shift register), it is important to ensure that the t_{CO} of a preceding F/F is longer than the hold time (t_h) of the following flip-flop, so data present at the input of the succeeding F/F is properly "shifted in" following the active edge of the clock. This relationship between t_{CO} and t_h is normally guaranteed if the F/Fs are physically identical. Furthermore, for correct operation, it is easy to verify that the clock period has to be greater than the sum $t_{su} + t_h$.

Generalizations

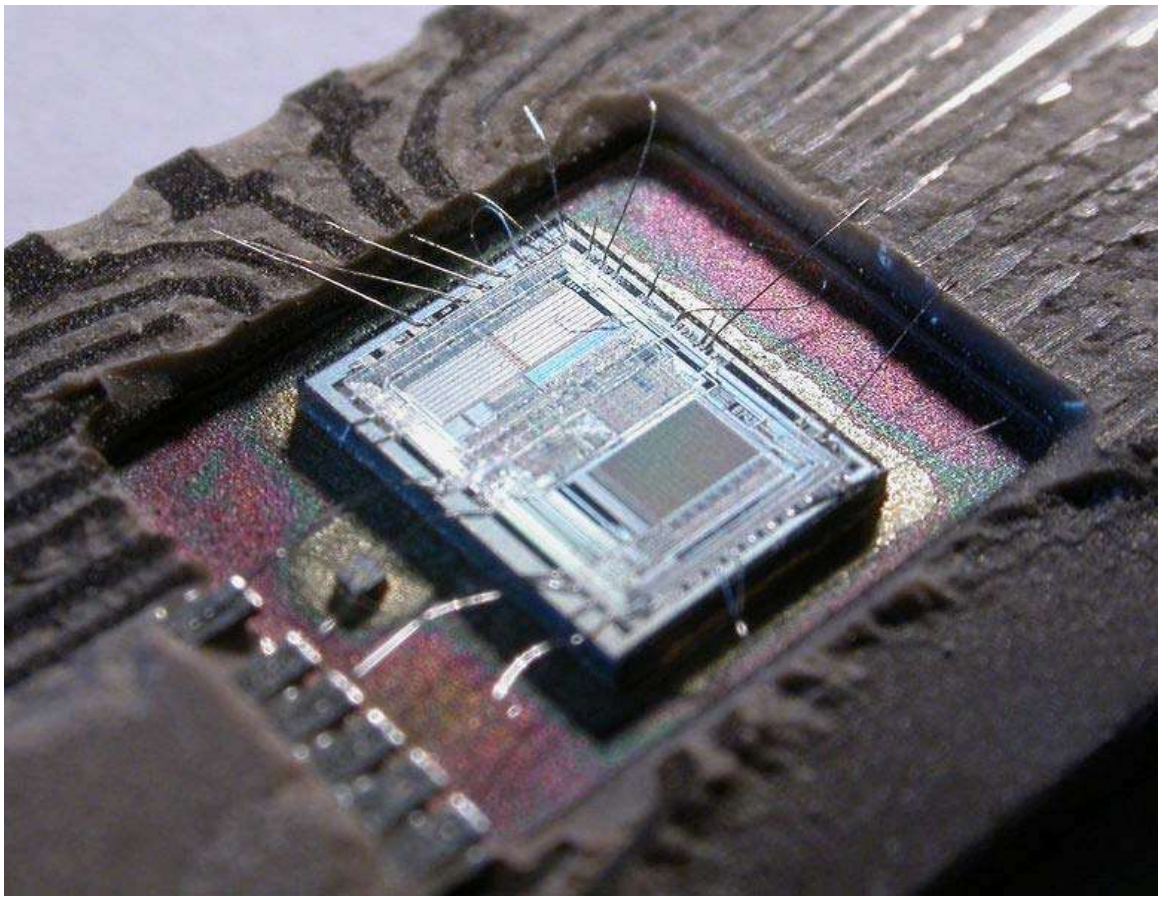
Flip-flops can be generalized in at least two ways: by making them 1-of-N instead of 1-of-2, and by adapting them to logic with more than two states. In the special cases of 1-of-3 encoding, or multi-valued ternary logic, these elements may be referred to as *flip-flap-flops*.

In a conventional flip-flop, exactly one of the two complementary outputs is high. This can be generalized to a memory element with N outputs, exactly one of which is high (alternatively, where exactly one of N is low). The output is therefore always a one-hot (respectively *one-cold*) representation. The construction is similar to a conventional cross-coupled flip-flop; each output, when high, inhibits all the other outputs. Alternatively, more or less conventional flip-flops can be used, one per output, with additional circuitry to make sure only one at a time can be true.

Another generalization of the conventional flip-flop is a memory element for multi-valued logic. In this case the memory element retains exactly one of the logic states until the control inputs induce a change. In addition, a multiple-valued clock can also be used, leading to new possible clock transitions.

Chapter- 9

Microcontroller



The die from an Intel 8742, an 8-bit microcontroller that includes a CPU running at 12 MHz, 128 bytes of RAM, 2048 bytes of EPROM, and I/O in the same chip.

A **microcontroller** (sometimes abbreviated **µC**, **uC** or **MCU**) is a small computer on a single integrated circuit containing a processor core, memory, and programmable input/output peripherals. Program memory in the form of NOR flash or OTP ROM is also often included on chip, as well as a typically small amount of RAM. Microcontrollers are

designed for embedded applications, in contrast to the microprocessors used in personal computers or other general purpose applications.

Microcontrollers are used in automatically controlled products and devices, such as automobile engine control systems, implantable medical devices, remote controls, office machines, appliances, power tools, and toys. By reducing the size and cost compared to a design that uses a separate microprocessor, memory, and input/output devices, microcontrollers make it economical to digitally control even more devices and processes. Mixed signal microcontrollers are common, integrating analog components needed to control non-digital electronic systems.

Some microcontrollers may use four-bit words and operate at clock rate frequencies as low as 4 kHz, for low power consumption (milliwatts or microwatts). They will generally have the ability to retain functionality while waiting for an event such as a button press or other interrupt; power consumption while sleeping (CPU clock and most peripherals off) may be just nanowatts, making many of them well suited for long lasting battery applications. Other microcontrollers may serve performance-critical roles, where they may need to act more like a digital signal processor (DSP), with higher clock speeds and power consumption.

Embedded design

A microcontroller can be considered a self-contained system with a processor, memory and peripherals and can be used as an embedded system. The majority of microcontrollers in use today are embedded in other machinery, such as automobiles, telephones, appliances, and peripherals for computer systems. These are called embedded systems. While some embedded systems are very sophisticated, many have minimal requirements for memory and program length, with no operating system, and low software complexity. Typical input and output devices include switches, relays, solenoids, LEDs, small or custom LCD displays, radio frequency devices, and sensors for data such as temperature, humidity, light level etc. Embedded systems usually have no keyboard, screen, disks, printers, or other recognizable I/O devices of a personal computer, and may lack human interaction devices of any kind.

Interrupts

Microcontrollers must provide real time (predictable, though not necessarily fast) response to events in the embedded system they are controlling. When certain events occur, an interrupt system can signal the processor to suspend processing the current instruction sequence and to begin an interrupt service routine (ISR, or "interrupt handler"). The ISR will perform any processing required based on the source of the interrupt before returning to the original instruction sequence. Possible interrupt sources are device dependent, and often include events such as an internal timer overflow, completing an analog to digital conversion, a logic level change on an input such as from a button being pressed, and data received on a communication link. Where power consumption is important as in battery operated devices, interrupts may also wake a

microcontroller from a low power sleep state where the processor is halted until required to do something by a peripheral event.

Programs

Microcontroller programs must fit in the available on-chip program memory, since it would be costly to provide a system with external, expandable, memory. Compilers and assemblers are used to convert high-level language and assembler language codes into a compact machine code for storage in the microcontroller's memory. Depending on the device, the program memory may be permanent, read-only memory that can only be programmed at the factory, or program memory may be field-alterable flash or erasable read-only memory.

Other microcontroller features

Microcontrollers usually contain from several to dozens of general purpose input/output pins (GPIO). GPIO pins are software configurable to either an input or an output state. When GPIO pins are configured to an input state, they are often used to read sensors or external signals. Configured to the output state, GPIO pins can drive external devices such as LEDs or motors.

Many embedded systems need to read sensors that produce analog signals. This is the purpose of the analog-to-digital converter (ADC). Since processors are built to interpret and process digital data, i.e. 1s and 0s, they are not able to do anything with the analog signals that may be sent to it by a device. So the analog to digital converter is used to convert the incoming data into a form that the processor can recognize. A less common feature on some microcontrollers is a digital-to-analog converter (DAC) that allows the processor to output analog signals or voltage levels.

In addition to the converters, many embedded microprocessors include a variety of timers as well. One of the most common types of timers is the Programmable Interval Timer (PIT). A PIT may either count down from some value to zero, or up to the capacity of the count register, overflowing to zero. Once it reaches zero, it sends an interrupt to the processor indicating that it has finished counting. This is useful for devices such as thermostats, which periodically test the temperature around them to see if they need to turn the air conditioner on, the heater on, etc.

Time Processing Unit (TPU) is a sophisticated timer. In addition to counting down, the TPU can detect input events, generate output events, and perform other useful operations.

A dedicated Pulse Width Modulation (PWM) block makes it possible for the CPU to control power converters, resistive loads, motors, etc., without using lots of CPU resources in tight timer loops.

Universal Asynchronous Receiver/Transmitter (UART) block makes it possible to receive and transmit data over a serial line with very little load on the CPU. Dedicated

on-chip hardware also often includes capabilities to communicate with other devices (chips) in digital formats such as I2C and Serial Peripheral Interface (SPI).

Higher integration

In contrast to general-purpose CPUs, micro-controllers may not implement an external address or data bus as they integrate RAM and non-volatile memory on the same chip as the CPU. Using fewer pins, the chip can be placed in a much smaller, cheaper package.

Integrating the memory and other peripherals on a single chip and testing them as a unit increases the cost of that chip, but often results in decreased net cost of the embedded system as a whole. Even if the cost of a CPU that has integrated peripherals is slightly more than the cost of a CPU and external peripherals, having fewer chips typically allows a smaller and cheaper circuit board, and reduces the labor required to assemble and test the circuit board.

A micro-controller is a single integrated circuit, commonly with the following features:

- central processing unit - ranging from small and simple 4-bit processors to complex 32- or 64-bit processors
- volatile memory (RAM) for data storage
- ROM, EPROM, EEPROM or Flash memory for program and operating parameter storage
- discrete input and output bits, allowing control or detection of the logic state of an individual package pin
- serial input/output such as serial ports (UARTs)
- other serial communications interfaces like I²C, Serial Peripheral Interface and Controller Area Network for system interconnect
- peripherals such as timers, event counters, PWM generators, and watchdog
- clock generator - often an oscillator for a quartz timing crystal, resonator or RC circuit
- many include analog-to-digital converters, some include digital-to-analog converters
- in-circuit programming and debugging support

This integration drastically reduces the number of chips and the amount of wiring and circuit board space that would be needed to produce equivalent systems using separate chips. Furthermore, on low pin count devices in particular, each pin may interface to several internal peripherals, with the pin function selected by software. This allows a part to be used in a wider variety of applications than if pins had dedicated functions. Micro-controllers have proved to be highly popular in embedded systems since their introduction in the 1970s.

Some microcontrollers use a Harvard architecture: separate memory buses for instructions and data, allowing accesses to take place concurrently. Where a Harvard architecture is used, instruction words for the processor may be a different bit size than

the length of internal memory and registers; for example: 12-bit instructions used with 8-bit data registers.

The decision of which peripheral to integrate is often difficult. The microcontroller vendors often trade operating frequencies and system design flexibility against time-to-market requirements from their customers and overall lower system cost. Manufacturers have to balance the need to minimize the chip size against additional functionality.

Microcontroller architectures vary widely. Some designs include general-purpose microprocessor cores, with one or more ROM, RAM, or I/O functions integrated onto the package. Other designs are purpose built for control applications. A micro-controller instruction set usually has many instructions intended for bit-wise operations to make control programs more compact. For example, a general purpose processor might require several instructions to test a bit in a register and branch if the bit is set, where a micro-controller could have a single instruction to provide that commonly-required function.

Microcontrollers typically do not have a math coprocessor, so floating point arithmetic is performed by software.

Volumes

About 55% of all CPUs sold in the world are 8-bit microcontrollers and microprocessors. According to Semico, over four billion 8-bit microcontrollers were sold in 2006.

A typical home in a developed country is likely to have only four general-purpose microprocessors but around three dozen microcontrollers. A typical mid-range automobile has as many as 30 or more microcontrollers. They can also be found in many electrical devices such as washing machines, microwave ovens, and telephones.

A PIC 18F8720 **microcontroller** in an 80-pin TQFP package.

Manufacturers have often produced special versions of their microcontrollers in order to help the hardware and software development of the target system. Originally these included EPROM versions that have a "window" on the top of the device through which program memory can be erased by ultraviolet light, ready for reprogramming after a programming ("burn") and test cycle. Since 1998, EPROM versions are rare and have been replaced by EEPROM and flash, which are easier to use (can be erased electronically) and cheaper to manufacture.

Other versions may be available where the ROM is accessed as an external device rather than as internal memory, however these are becoming increasingly rare due to the widespread availability of cheap microcontroller programmers.

The use of field-programmable devices on a microcontroller may allow field update of the firmware or permit late factory revisions to products that have been assembled but not yet shipped. Programmable memory also reduces the lead time required for deployment of a new product.

Where hundreds of thousands of identical devices are required, using parts programmed at the time of manufacture can be an economical option. These "mask programmed" parts have the program laid down in the same way as the logic of the chip, at the same time.

Programming environments

Microcontrollers were originally programmed only in assembly language, but various high-level programming languages are now also in common use to target microcontrollers. These languages are either designed specially for the purpose, or versions of general purpose languages such as the C programming language. Compilers for general purpose languages will typically have some restrictions as well as enhancements to better support the unique characteristics of microcontrollers. Some microcontrollers have environments to aid developing certain types of applications. Microcontroller vendors often make tools freely available to make it easier to adopt their hardware.

Many microcontrollers are so quirky that they effectively require their own non-standard dialects of C, such as SDCC for the 8051, which prevent using standard tools (such as code libraries or static analysis tools) even for code unrelated to hardware features. Interpreters are often used to hide such low level quirks.

Interpreter firmware is also available for some microcontrollers. For example, BASIC on the early microcontrollers Intel 8052; BASIC and FORTH on the Zilog Z8 as well as some modern devices. Typically these interpreters support interactive programming.

Simulators are available for some microcontrollers, such as in Microchip's MPLAB environment and the Revolution Education PICAXE range. These allow a developer to analyze what the behavior of the microcontroller and their program should be if they were using the actual part. A simulator will show the internal processor state and also that of the outputs, as well as allowing input signals to be generated. While on the one hand most simulators will be limited from being unable to simulate much other hardware in a system, they can exercise conditions that may otherwise be hard to reproduce at will in the physical implementation, and can be the quickest way to debug and analyze problems.

Recent microcontrollers are often integrated with on-chip debug circuitry that when accessed by an in-circuit emulator via JTAG, allow debugging of the firmware with a debugger.

Types of microcontrollers

As of 2008 there are several dozen microcontroller architectures and vendors including:

- Parallax Propeller
- Freescale 68HC11 (8-bit)
- Intel 8051
- Silicon Laboratories Pipelined 8051 Microcontrollers
- ARM processors (from many vendors) using ARM7 or Cortex-M3 cores are generally microcontrollers
- STMicroelectronics STM8 (8-bit), ST10 (16-bit) and STM32 (32-bit)

- Atmel AVR (8-bit), AVR32 (32-bit), and AT91SAM (32-bit)
- Freescale ColdFire (32-bit) and S08 (8-bit)
- Hitachi H8, Hitachi SuperH (32-bit)
- Hyperstone E1/E2 (32-bit, First full integration of RISC and DSP on one processor core [1996])
- Infineon Microcontroller: 8, 16, 32 Bit microcontrollers for automotive and industrial applications
- MIPS (32-bit PIC32)
- NEC V850 (32-bit)
- PIC (8-bit PIC16, PIC18, 16-bit dsPIC33 / PIC24)
- PowerPC ISE
- PSoC (Programmable System-on-Chip)
- Rabbit 2000 (8-bit)
- Texas Instruments Microcontrollers MSP430 (16-bit), C2000 (32-bit), and Stellaris (32-bit)
- Toshiba TLCS-870 (8-bit/16-bit)
- Zilog eZ8 (16-bit), eZ80 (8-bit)

and many others, some of which are used in very narrow range of applications or are more like applications processors than microcontrollers. The microcontroller market is extremely fragmented, with numerous vendors, technologies, and markets. Note that many vendors sell (or have sold) multiple architectures.

Interrupt latency

In contrast to general-purpose computers, microcontrollers used in embedded systems often seek to optimize interrupt latency over instruction throughput. Issues include both reducing the latency, and making it be more predictable (to support real-time control).

When an electronic device causes an interrupt, the intermediate results (registers) have to be saved before the software responsible for handling the interrupt can run. They must also be restored after that software is finished. If there are more registers, this saving and restoring process takes more time, increasing the latency. Ways to reduce such context/restore latency include having relatively few registers in their central processing units (undesirable because it slows down most non-interrupt processing substantially), or at least having the hardware not save them all (this fails if the software then needs to compensate by saving the rest "manually"). Another technique involves spending silicon gates on "shadow registers": one or more duplicate registers used only by the interrupt software, perhaps supporting a dedicated stack.

Other factors affecting interrupt latency include:

- Cycles needed to complete current CPU activities. To minimize those costs, microcontrollers tend to have short pipelines (often three instructions or less), small write buffers, and ensure that longer instructions are continuable or

restartable. RISC design principles ensure that most instructions take the same number of cycles, helping avoid the need for most such continuation/restart logic.

- The length of any critical section that needs to be interrupted. Entry to a critical section restricts concurrent data structure access. When a data structure must be accessed by an interrupt handler, the critical section must block that interrupt. Accordingly, interrupt latency is increased by however long that interrupt is blocked. When there are hard external constraints on system latency, developers often need tools to measure interrupt latencies and track down which critical sections cause slowdowns.
    - One common technique just blocks all interrupts for the duration of the critical section. This is easy to implement, but sometimes critical sections get uncomfortably long.
    - A more complex technique just blocks the interrupts that may trigger access to that data structure. This often based on interrupt priorities, which tend to not correspond well to the relevant system data structures. Accordingly, this technique is used mostly in very constrained environments.
    - Processors may have hardware support for some critical sections. Examples include supporting atomic access to bits or bytes within a word, or other atomic access primitives like the LDREX/STREX exclusive access primitives introduced in the ARMv6 architecture.
- Interrupt nesting. Some microcontrollers allow higher priority interrupts to interrupt lower priority ones. This allows software to manage latency by giving time-critical interrupts higher priority (and thus lower and more predictable latency) than less-critical ones.
- Trigger rate. When interrupts occur back-to-back, microcontrollers may avoid an extra context save/restore cycle by a form of tail call optimization.

Lower end microcontrollers tend to support fewer interrupt latency controls than higher end ones.

History

The first single-chip microprocessor was the 4-bit Intel 4004 released in 1971. With the Intel 8008 and more capable microprocessors available over the next several years.

These however all required external chip(s) to implement a working system, raising total system cost, and making it impossible to economically computerize appliances.

The first computer system on a chip optimized for control applications was the Intel 8048, released in 1975, with both RAM and ROM on the same chip. This chip would find its way into over one billion PC keyboards, and other numerous applications. At this time Intels President, Luke J. Valenter, stated that the (Microcontroller) was one of the most successful in the companies history, and expanded the division's budget over 25%.

Most microcontrollers at this time had two variants. One had an erasable EPROM program memory, which was significantly more expensive than the PROM variant which was only programmable once.

In 1993, the introduction of EEPROM memory allowed microcontrollers (beginning with the Microchip PIC16x84)) to be electrically erased quickly without an expensive package as required for EPROM, allowing both rapid prototyping, and In System Programming.

The same year, Atmel introduced the first microcontroller using Flash memory.

Other companies rapidly followed suit, with both memory types.

Cost has plummeted over time, with the cheapest 8-bit microcontrollers being available for under $0.25 in quantity (thousands) in 2009, and some 32-bit microcontrollers around $1 for similar quantities.

Nowadays microcontrollers are low cost and readily available for hobbyists, with large online communities around certain processors.

In the future, MRAM could potentially be used in microcontrollers as it has infinite endurance and its incremental semiconductor wafer process cost is relatively low.

Microcontroller embedded memory technology

Since the emergence of microcontrollers, many different memory technologies have been used. Almost all microcontrollers have at least two different kinds of memory, a non-volatile memory for storing firmware and a read-write memory for temporary data.

Data

From the earliest microcontrollers to today, six-transistor SRAM is almost always used as the read/write working memory, with a few more transistors per bit used in the register file. MRAM could potentially replace it as it is 4-10 times denser which would make it more cost effective.

In addition to the SRAM, some microcontrollers also have internal EEPROM for data storage; and even ones that do not have any (or not enough) are often connected to external serial EEPROM chip (such as the BASIC Stamp) or external serial flash memory chip.

A few recent microcontrollers beginning in 2003 have "self-programmable" flash memory.

Firmware

The earliest microcontrollers used hard-wired or mask ROM to store firmware. Later microcontrollers (such as the early versions of the Freescale 68HC11 and early PIC microcontrollers) had quartz windows that allowed ultraviolet light in to erase the EPROM.

The Microchip PIC16C84, introduced in 1993, was the first microcontroller to use EEPROM to store firmware

Also in 1993, Atmel introduced the first microcontroller using NOR Flash memory to store firmware.

PSoC microcontrollers, introduced in 2002, store firmware in SONOS flash memory.

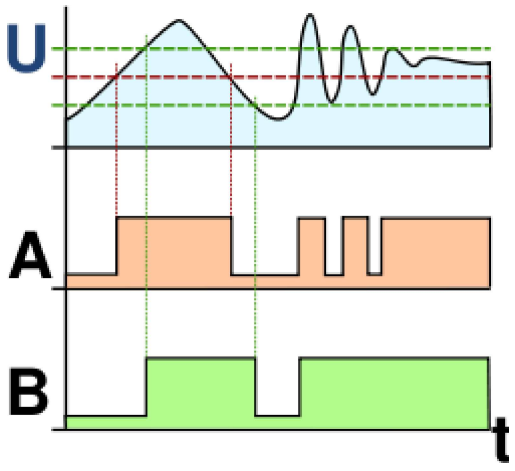
MRAM could potentially be used to store firmware.

Chapter- 10

Schmitt Trigger

In electronics, **Schmitt trigger** is a generic name of *threshold circuits* with positive feedback having a loop gain > 1. The circuit is named "trigger" because the output retains its value until the input changes sufficiently to trigger a change: in the non-inverting configuration, when the input is higher than a certain chosen threshold, the output is high; when the input is below a different (lower) chosen threshold, the output is low; when the input is between the two, the output retains its value. This dual threshold action is called *hysteresis* and implies that the Schmitt trigger possess memory and can act as a bistable circuit (latch). There is a close relation between the two kinds of circuits that actually are the same: a Schmitt trigger can be converted into a latch and v.v., a latch can be converted into a Schmitt trigger.

Schmitt trigger devices are typically used in open loop configurations for noise immunity and closed loop negative feedback configurations to implement bistable regulators, triangle/square wave generators, etc.



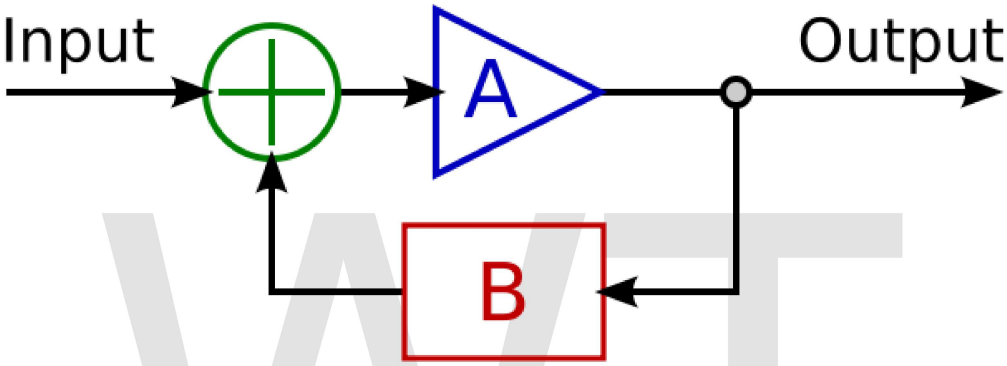
The effect of using a Schmitt trigger (B) instead of a comparator (A).

Invention

The Schmitt trigger was invented by US scientist Otto H. Schmitt in 1934 while he was still a graduate student, later described in his doctoral dissertation (1937) as a "thermionic trigger". It was a direct result of Schmitt's study of the neural impulse propagation in squid nerves.

Implementation

Fundamental idea



Schmitt trigger is a system with an avalanche-like positive feedback ($B < 1$; $B.A > 1$), in which the output helps the input

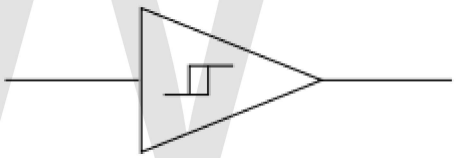
Circuits with hysteresis are based on the fundamental positive feedback idea: *any active circuit can be made to behave as a Schmitt trigger by applying a positive feedback so that the loop gain is more than one*. The positive feedback is introduced by adding a part of the output voltage to the input voltage; so, these circuits contain an *attenuator* (the B box in the figure on the right) and a *summer* (the circle with "+" inside) in addition to an amplifier acting as a comparator. There are three specific techniques for implementing this general idea. The first two of them are dual versions (series and parallel) of the general positive feedback system. In these configurations, the output voltage increases the effective difference input voltage of the comparator by *decreasing the threshold* or by *increasing the circuit input voltage*; the threshold and memory properties are incorporated in one element. In the third technique, the threshold and memory properties are separated.

Dynamic threshold (series feedback): *when the input voltage crosses the threshold in some direction the very circuit changes its own threshold to the opposite direction*. For this purpose, it subtracts a part of its output voltage from the threshold (it is equal to adding voltage to the input voltage). Thus the output affects the threshold and does not impact on the input voltage. These circuits are implemented by a differential amplifier with *series positive feedback* where the input is connected to the inverting input and the

output - to the non-inverting input. In this arrangement, attenuation and summation are separated: a voltage divider acts as an attenuator and the loop acts as a simple series voltage summer. Examples: the classic transistor emitter-coupled Schmitt trigger, op-amp inverting Schmitt trigger, etc.

Modified input voltage (parallel feedback): *when the input voltage crosses the threshold in some direction the circuit changes the very input voltage in the same direction* (now it adds a part of its output voltage directly to the input voltage). Thus the output "helps" the input voltage and does not affect the threshold. These circuits can be implemented by a single-ended non-inverting amplifier with *parallel positive feedback* where the input and the output sources are connected through resistors to the input. The two resistors form a weighted parallel summer incorporating both the attenuation and summation. Examples: the less familiar collector-base coupled Schmitt trigger, op-amp non-inverting Schmitt trigger, etc.

Two different unidirectional thresholds *are assigned in this case to two separate open-loop comparators (without hysteresis) driving an RS trigger (2-input memory cell)*. The trigger is toggled high when the input voltage crosses down to up the high threshold and low when the input voltage crosses up to down the low threshold. Again, there is a positive feedback but now it is concentrated only in the memory cell. Example: 555 timer, switch debounce circuit.



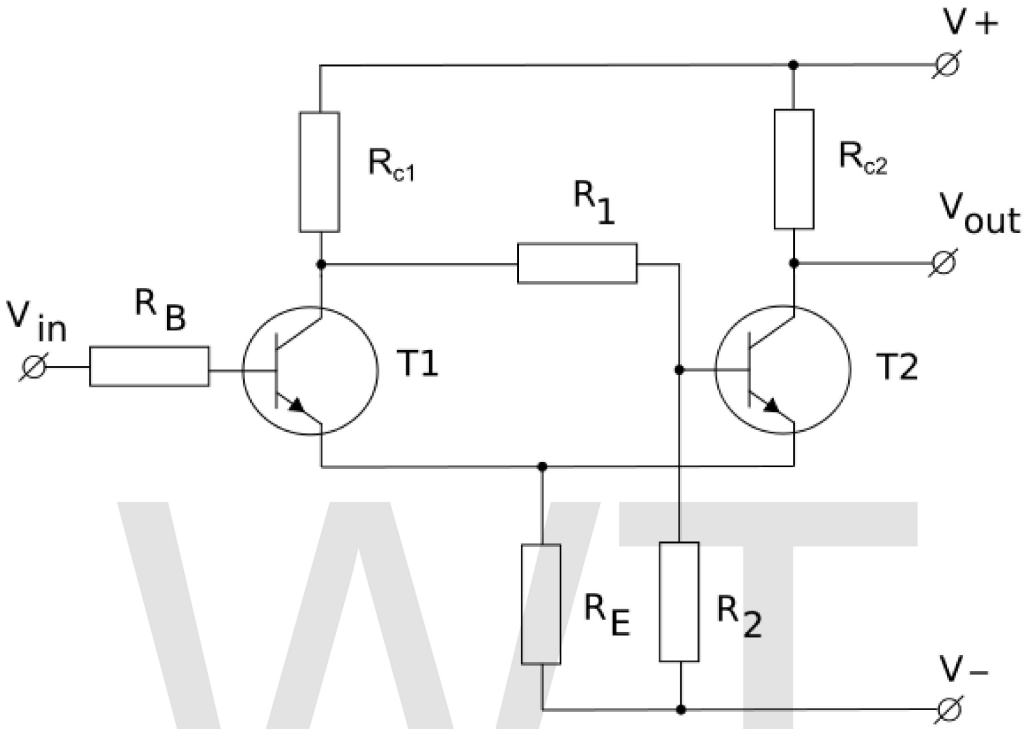
The symbol of Schmitt trigger

Some circuits and elements exhibiting negative resistance can also act as Schmitt triggers: negative impedance converters (NIC), neon lamps, tunnel diodes (e.g., a diode with an "N"-shaped current–voltage characteristic in the first quadrant), etc. In the last case, an oscillating input will cause the diode to move from one rising leg of the "N" to the other and back again as the input crosses the rising and falling switching thresholds.

The symbol for Schmitt triggers in circuit diagrams is a triangle with a symbol inside representing its ideal hysteresis curve.

Transistor Schmitt triggers

Classic emitter-coupled circuit



Schmitt trigger implemented by two emitter-coupled transistor stages

The original Schmitt trigger is based on the dynamic threshold idea that is implemented by a voltage divider with a switchable upper leg (the collector resistors R_{c1} and R_{c2}) and a steady lower leg (R_E). T1 acts as a comparator with a differential input (T1 base-emitter junction) consisting of an inverting (T1 base) and a non-inverting (T1 emitter) inputs. The input voltage is applied to the inverting input; the output voltage of the voltage divider is applied to the non-inverting input thus determining its threshold. The comparator output drives the second common collector stage T2 (an *emitter follower*) through the voltage follower R_1-R_2. The emitter-coupled transistors T1 and T2 actually compose an electronic double throw switch that switches over the upper legs of the voltage divider and changes the threshold in a different (to the input voltage) direction.

This configuration can be considered as a differential amplifier with series positive feedback between its non-inverting input (T2 base) and output (T1 collector) that forces the transition process. There is also a smaller negative feedback introduced by the emitter resistor R_E. To make the positive feedback dominate over the negative one and to obtain a hysteresis, the proportion between the two collector resistors is chosen $R_{c1} > R_{c2}$. Thus less current flows through and less voltage drop is across R_E when T1 is switched on than in the case when T2 is switched on. As a result, the circuit has two different thresholds in regard to ground (V- in the picture).

Operation

Initial state. For NPN transistors as shown, imagine the input voltage is below the shared emitter voltage (high threshold for concreteness) so that T1 base-emitter junction is backward-biased and T1 does not conduct. T2 base voltage is determined by the mentioned divider so that T2 is conducting and the trigger output is in the low state. The two resistors R_{c2} and R_E form another voltage divider that determines the high threshold. Neglecting V_{BE}, the high threshold value is approximately

$$V_{HT} = \frac{R_E}{R_E + R_{c2}} V_+ .$$

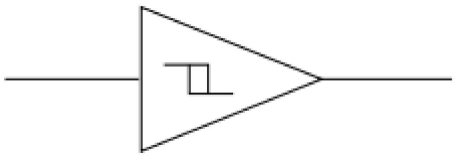
The output voltage is low but well above the ground. It is approximately equal to the high threshold and may not be low enough to be a logical zero for next digital circuits. This may require additional shifting circuit following the trigger circuit.

Crossing up the high threshold. When the input voltage (T1 base voltage) rises slightly above the voltage across the emitter resistor R_E (the high threshold), T1 begins conducting. Its collector voltage goes down and T2 begins going cut-off, because the voltage divider now provides lower T2 base voltage. The common emitter voltage follows this change and goes down thus making T1 conduct more. The current begins steering from the right leg of the circuit to the left one. Although T1 is more conducting, it passes less current through R_E (since $R_{c1} > R_{c2}$); the emitter voltage continues dropping and the effective T1 base-emitter voltage continuously increases. This avalanche-like process continues until T1 becomes completely turned on (saturated) and T2 turned off. The trigger is transitioned to the high state and the output (T2 collector) voltage is close to V+. Now, the two resistors R_{c1} and R_E form a voltage divider that determines the low threshold. Its value is approximately

$$V_{LT} = \frac{R_E}{R_E + R_{c1}} V_+ .$$

Crossing down the low threshold. With the trigger now in the high state, if the input voltage lowers enough (below the low threshold), T1 begins cutting-off. Its collector current reduces; as a result, the shared emitter voltage lowers slightly and T1 collector voltage rises significantly. R_1-R_2 voltage divider conveys this change to T2 base voltage and it begins conducting. The voltage across R_E rises, further reducing the T1 base-emitter potential in the same avalanche-like manner, and T1 ceases to conduct. T2 becomes completely turned-on (saturated) and the output voltage becomes low again.

Variations

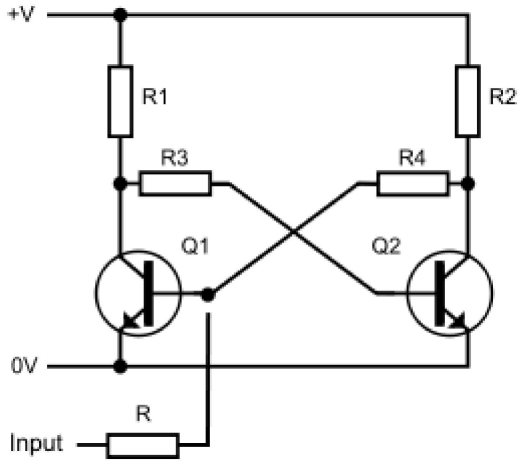


The symbol of inverting Schmitt trigger

Non-inverting circuit. The classic non-inverting Schmitt trigger can be turned into an inverting trigger by taking V_{out} from the emitters instead from T2 collector. In this configuration, the output voltage is equal to the dynamic threshold (the shared emitter voltage) and both the output levels stay away from the supply rails. Another disadvantage is that the load changes the thresholds; so, it has to be high enough. The base resistor R_B is obligatory to prevent the impact of the input voltage through T1 base-emitter junction on the emitter voltage.

Direct-coupled circuit. To simplify the circuit, the voltage divider R_1-R_2 can be omitted connecting T1 collector directly to T2 base. The base resistor R_B can be omitted as well so that the input voltage source drives directly T1 base. In this case, the common emitter voltage and T1 collector voltage are not suitable for outputs. Only T2 collector should be used as an output since, when the input voltage exceedes the high threshold and T1 saturates, its base-emitter junction is forward-biased and transfers the input voltage variations directly to the emitters. As a result, the common emitter voltage and T1 collector voltage follow the input voltage. This situation is typical for over-driven transistor differential amplifiers and ECL gates.

Collector-base coupled circuit



BJT bistable collector-base coupled circuit can be converted to a Schmitt trigger by connecting an additional base resistor to some of the bases

Like every latch, the fundamental collector-base coupled bistable circuit possesses a hysteresis. So, it can be converted to a Schmitt trigger by connecting an additional base resistor R to some of the inputs (Q1 base in the figure). The two resistors R and R_4 form a parallel voltage summer (the circle in the block diagram above) that sums output (Q2 collector) voltage and the input voltage, and drives the single-ended transistor "comparator" Q1. When the base voltage crosses the threshold ($V_{BE0} \infty 0.65$ V) in some direction, a part of Q2 collector voltage is added in the same direction to the input voltage. Thus the output modifies the input voltage by means of parallel positive feedback and does not affect the threshold (the base-emitter voltage).

Comparison between emitter- and collector-coupled circuit

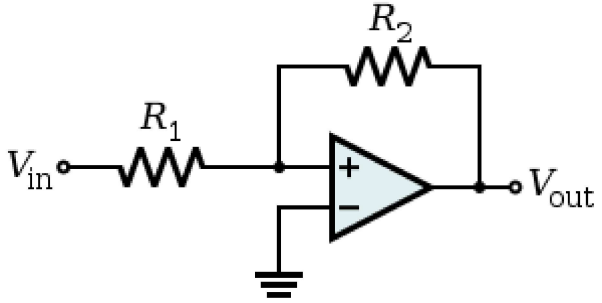
The emitter-coupled version has the advantage that the input transistor is backward-biased when the input voltage is quite below the high threshold; so, the transistor is surely cut-off. It was important when germanium transistors were used for implementing the circuit and this advantage has determined its popularity. The input base resistor can be omitted since the emitter resistor limits the current when the input base-emitter junction is forward-biased.

The emitter-coupled Schmitt trigger has not low enough level at output *logical zero* and needs an additional output shifting circuit. The collector-coupled trigger has extremely low (almost zero) output level at output *logical zero*.

Op-amp implementations

Schmitt triggers are commonly implemented using an operational amplifier or the more dedicated comparator. An open-loop op-amp and comparator may be considered as an analog-digital device having analog inputs and a digital output that extracts the sign of the voltage difference between its two inputs. The positive feedback is applied by adding a part of the output voltage to the input voltage in series or parallel manner. Due to the extremely high op-amp gain, the loop gain is also high enough and provides the avalanche-like process.

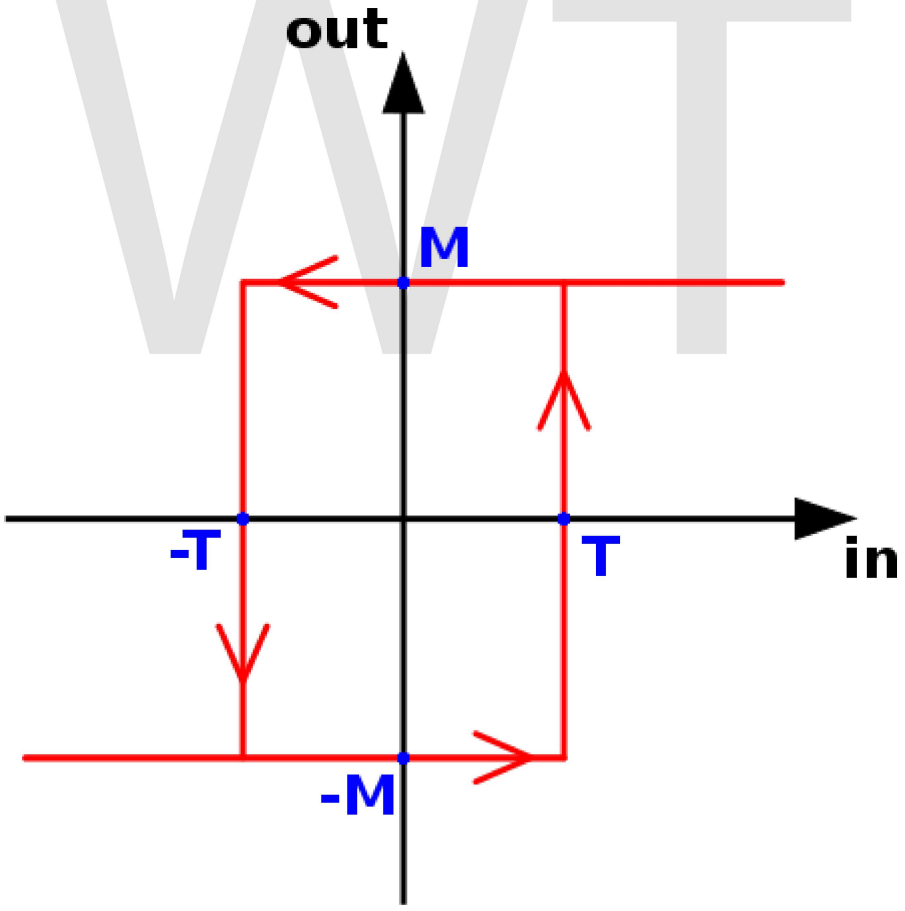
Non-inverting Schmitt trigger



Schmitt trigger implemented by a non-inverting comparator

In this circuit, the two resistors R_1 and R_2 form a parallel voltage summer. It adds a part of the output voltage to the input voltage thus "helping" it during and after switching that occurs when the resulting voltage is near the ground. This *parallel positive feedback* creates the needed hysteresis that is controlled by the proportion between the resistances of R_1 and R_2. The output of the parallel voltage summer is single-ended (it produces voltage in respect to ground); so, the circuit does not need an amplifier with a differential input. Since the conventional op-amps usually have a differential input, the inverting input is grounded (not used).

The output voltage always has the same sign as the *op-amp input voltage* but it not always has the same sign as the *circuit input voltage* (the signs of the two input voltages can differ). When the circuit input voltage is above the high threshold or below the low threshold, the output voltage has the same sign as the *circuit input voltage* (the circuit is non-inverting). It acts like a comparator that switches at a different point depending on whether the output of the comparator is high or low. When the circuit input voltage is between the thresholds, the output voltage is undefined; it depends on the last state (the circuit behaves as an elementary latch).

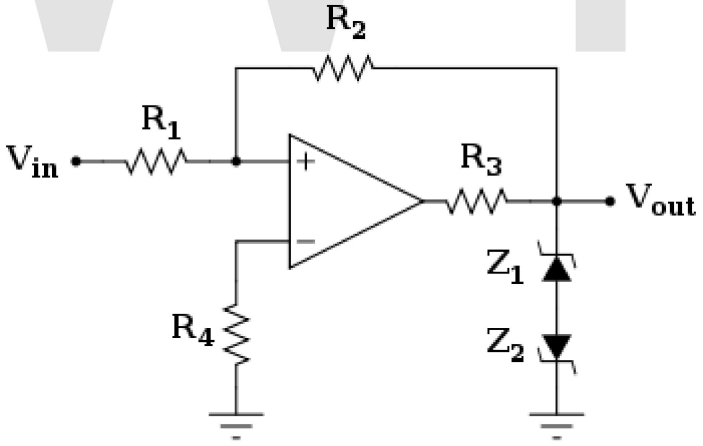


Typical hysteresis curve (Non-inverting) (which matches the curve shown on a Schmitt trigger symbol)

For instance, if the Schmitt trigger is currently in the high state, the output will be at the positive power supply rail (+V_S). The output voltage V_+ of the resistive summer can be found by applying the superposition theorem:

$$V_+ = \frac{R_2}{R_1 + R_2} \cdot V_{\text{in}} + \frac{R_1}{R_1 + R_2} \cdot V_s$$

The comparator will switch when V_+=0. Then $R_2 \cdot V_{\text{in}} = -R_1 \cdot V_s$(the same result can be obtained by applying the current conservation principle). So V_{in} must drop below $-\frac{R_1}{R_2}V_s$ to get the output to switch. Once the comparator output has switched to $-V_S$, the threshold becomes $+\frac{R_1}{R_2}V_s$ to switch back to high. So this circuit creates a switching band centered around zero, with trigger levels $\pm\frac{R_1}{R_2}V_s$ (it can be shifted to the left or the right by applying a bias voltage to the inverting input). The input voltage must rise above the top of the band, and then below the bottom of the band, for the output to switch on (plus) and then back off (minus). If R_1 is zero or R_2 is infinity (i.e., an open circuit), the band collapses to zero width, and it behaves as a standard comparator. The transfer characteristic is shown in the picture on the right. The value of the threshold T is given by $\frac{R_1}{R_2}V_s$ and the maximum value of the output M is the power supply rail.



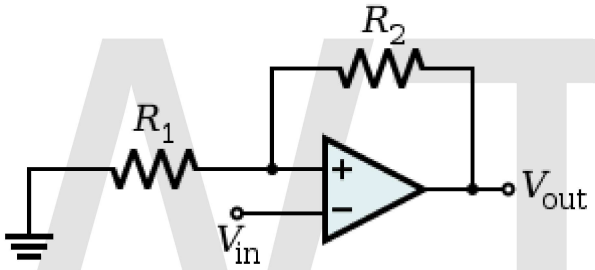
A practical Schmitt trigger configuration with precise thresholds

A unique property of circuits with parallel positive feedback is the impact on the input source. In circuits with negative parallel feedback (e.g., an inverting amplifier), the virtual ground at the inverting input separates the input source from the op-amp output. Here there is no permanent virtual ground (there is only an instant one during the transition) and the steady op-amp output voltage is applied through R_1 - R_2 network to the

input source. It is interesting fact that the op-amp output passes an opposite current through the input source (it injects current into the source when the input voltage is positive and it draws current from the source when it is negative).

A practical Schmitt trigger with precise thresholds is shown in the figure on the left. The transfer characteristic has exactly the same shape of the previous basic configuration, and the threshold values are the same as well. On the other hand, in the previous case, the output voltage was depending on the power supply, while now it is defined by the Zener diodes (which could also be replaced with a single double-anode Zener diode). In this configuration, the output levels can be modified by appropriate choice of Zener diode, and these levels are resistant to power supply fluctuations (i.e., they increase the PSRR of the comparator). The resistor R_3 is there to limit the current through the diodes, and the resistor R_4 minimizes the input voltage offset caused by the comparator's input leakage currents.

Inverting Schmitt trigger



Schmitt trigger implemented by an inverting comparator

In the inverting version, the attenuation and summation are separated. The two resistors R_1 and R_2 act only as a "pure" attenuator (voltage divider). The input loop acts as a simple series voltage summer that adds a part of the output voltage in series to the circuit input voltage. This *series positive feedback* creates the needed hysteresis that is controlled by the proportion between the resistances of R_1 and the whole resistance (R_1 and R_2). The effective voltage applied to the op-amp input is floating; so, the op-amp must have a differential input.

The circuit is named *inverting* since the output voltage always has an opposite sign to the input voltage when it is out of the hysteresis cycle (when the input voltage is above the high threshold or below the low threshold). However, if the input voltage is within the hysteresis cycle (between the high and low thresholds), the circuit can be inverting as well as non-inverting. The output voltage is undefined; it depends on the last state and the circuit behaves as an elementary latch.

To compare the two versions, the circuit operation will be considered at the same conditions as above. If the Schmitt trigger is currently in the high state, the output will be at the positive power supply rail ($+V_S$). The output voltage V_+ of the voltage divider is:

$$V_+ = \frac{R_1}{R_1 + R_2} \cdot V_s$$

The comparator will switch when $V_{in} = V_+$. So V_{in} must exceed above this voltage to get the output to switch. Once the comparator output has switched to $-V_S$, the threshold becomes $-\frac{R_1}{R_1 + R_2} V_s$ to switch back to high. So this circuit creates a switching band centered around zero, with trigger levels $\pm\frac{R_1}{R_1 + R_2} V_s$ (it can be shifted to the left or the right by connecting R_1 to bias voltage). The input voltage must rise above the top of the band, and then below the bottom of the band, for the output to switch off (minus) and then back on (plus). If R_1 is infinity or R_2 is zero (i.e., an short circuit), the band collapses to zero width, and it behaves as a standard comparator.

In contrast with the parallel version, this circuit does not impact on the input source since the source is separated from the voltage divider output by the high op-amp input differential impedance.

Applications

Schmitt triggers are typically used in open loop configurations for noise immunity and closed loop configurations to implement function generators.

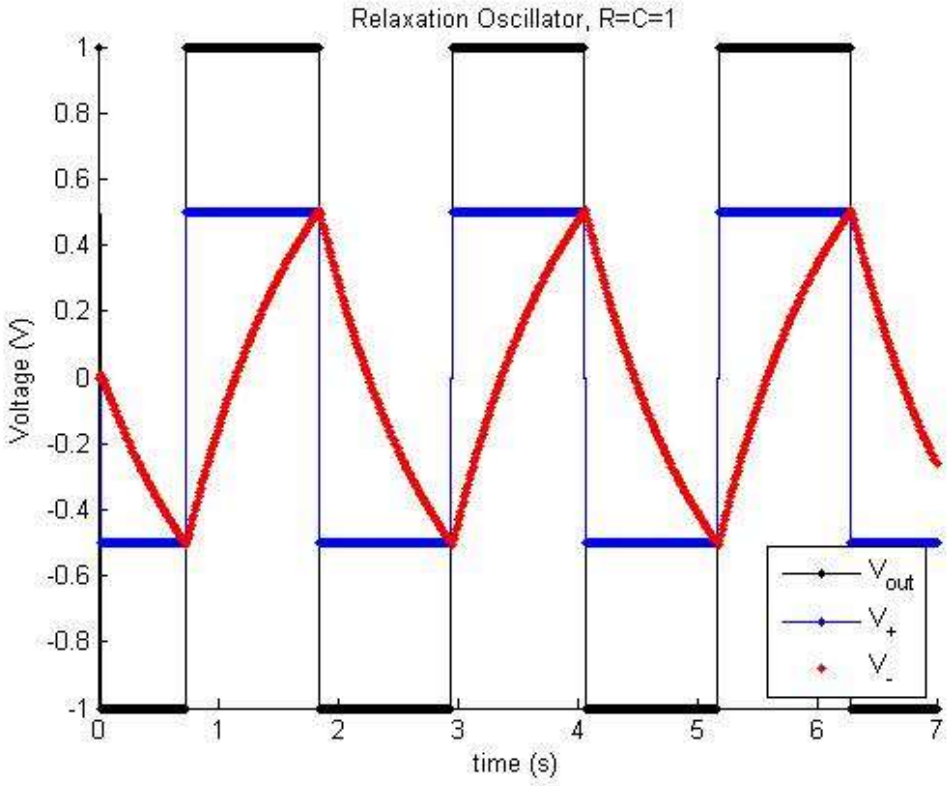
Noise immunity

One application of a Schmitt trigger is to increase the noise immunity in a circuit with only a single input threshold. With only one input threshold, a noisy input signal near that threshold could cause the output to switch rapidly back and forth from noise alone. A noisy Schmitt Trigger input signal near one threshold can cause only one switch in output value, after which it would have to move beyond the other threshold in order to cause another switch.

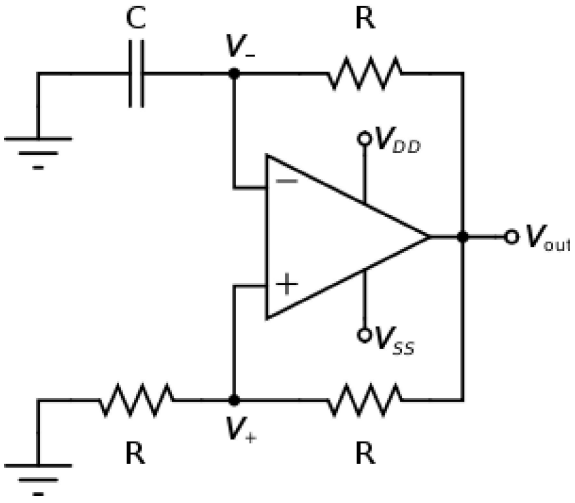
For example, in Fairchild Semiconductor's QSE15x family of infrared photosensors, an amplified infrared photodiode generates an electric signal that switches frequently between its absolute lowest value and its absolute highest value. This signal is then low-pass filtered to form a smooth signal that rises and falls corresponding to the relative amount of time the switching signal is on and off. That filtered output passes to the input of a Schmitt trigger. The net effect is that the output of the Schmitt trigger only passes from low to high after a received infrared signal excites the photodiode for longer than some known delay, and once the Schmitt trigger is high, it only moves low after the infrared signal ceases to excite the photodiode for longer than a similar known delay. Whereas the photodiode is prone to spurious switching due to noise from the environment, the delay added by the filter and Schmitt trigger ensures that the output only switches when there is certainly an input stimulating the device.

As discussed in the example above, the Fairchild Semiconductor QSE15x family of photosensors use a Schmitt trigger internally for noise immunity. Schmitt triggers are common in many switching circuits for similar reasons (e.g., for switch debouncing). The extended content below contains a list of IC including input Schmitt triggers.

Use as an oscillator



Output and capacitor waveforms for comparator-based relaxation oscillator



A comparator-based implementation of a relaxation oscillator

A Schmitt trigger is a bistable multivibrator, and it can be used to implement another type of multivibrator, the relaxation oscillator. This is achieved by connecting a single RC integrating circuit between the output and the input of an inverting Schmitt trigger. The output will be a continuous square wave whose frequency depends on the values of R and C, and the threshold points of the Schmitt trigger. Since multiple Schmitt trigger circuits can be provided by a single integrated circuit (e.g. the 4000 series CMOS device type 40106 contains 6 of them), a spare section of the IC can be quickly pressed into service as a simple and reliable oscillator with only two external components.

Here, a comparator-based Schmitt trigger is used in its inverting configuration. Additionally, slow negative feedback is added with an integrating RC network. The result, which is shown on the right, is that the output automatically oscillates from V_{SS} to V_{DD} as the capacitor charges from one Schmitt trigger threshold to the other.

Chapter- 11

Logic Family

In computer engineering, a **logic family** may refer to one of two related concepts. A logic family of monolithic digital integrated circuit devices is a group of electronic logic gates constructed using one of several different designs, usually with compatible logic levels and power supply characteristics within a family. Many logic families were produced as individual components, each containing one or a few related basic logical functions, which could be used as "building-blocks" to create systems or as so-called "glue" to interconnect more complex integrated circuits.

A "logic family" may also refer to a set of techniques used to implement logic within VLSI integrated circuits such as central processors, memories, or other complex functions. Some such logic families use static techniques to minimize design complexity. Other such logic families, such as domino logic, use clocked dynamic techniques to minimize size, power consumption, and delay.

Before the widespread use of integrated circuits, various solid-state and vacuum-tube logic systems were used but these were never as standardized and interoperable as the integrated-circuit devices.

Technologies

The list of packaged building-block logic families can be divided into categories, listed here in rough chronological order of introduction along with their usual abbreviations:

- Resistor–transistor logic (RTL)
  - Direct-coupled transistor logic (DCTL)
  - Resistor–transistor logic (RCTL)
- Diode–transistor logic (DTL)
  - Complemented transistor diode logic (CTDL)
  - High-threshold logic (HTL)
- Emitter-coupled logic (ECL)
  - Positive emitter-coupled logic (PECL)
  - Low-voltage positive emitter-coupled logic (LVPECL)

- Gunning transceiver logic (GTL)
- Transistor–transistor logic (TTL)
- P-type metal–oxide–semiconductor logic (PMOS)
- N-type metal–oxide–semiconductor logic (NMOS)
    - Depletion-load NMOS logic
- Complementary metal–oxide–semiconductor logic (CMOS)
- Bipolar complementary metal–oxide–semiconductor logic (BiCMOS)
- Integrated injection logic (I^2L)

The families (RTL, DTL, and ECL) were derived from the logic circuits used in early computers, originally implemented using discrete components. One example is the Philips NORbits family of logic building blocks.

The PMOS and I^2L logic families were used for relatively short periods, mostly in special purpose custom LSI circuits devices and are generally considered obsolete. For example, early digital clocks or electronic calculators may have used one or more PMOS devices to provide most of the logic for the finished product. The F14 CADC, Intel 4004, Intel 4040, and Intel 8008 microprocessors and their support chips were PMOS.

Of these families, only ECL, TTL, NMOS, CMOS, and BiCMOS are currently still in widespread use.

RTL

The Atanasoff–Berry Computer used vacuum tube logic circuits similar to RTL. Several early transistorized computers (e.g., IBM 1620, 1959) used RTL, where it was implemented using discrete components.

A family of simple resistor–transistor logic integrated circuits was developed at Fairchild Semiconductor for the Apollo Guidance Computer in 1962.

Texas Instruments soon introduced its own family of RTL.

A variant with integrated capacitors, RCTL, had increased speed, but lower immunity to noise than RTL. This was made by Texas Instruments as their "51XX" series.

DTL

Diode logic goes back as far as ENIAC and was used in many early vacuum tube computers. Several early transistorized computers (e.g., IBM 1401) used DTL, where it was implemented using discrete components.

The first diode–transistor logic family of integrated circuits was introduced by Signetics in 1962. DTL was also made by Fairchild and Westinghouse.

A family of diode logic and diode-transistor logic integrated circuits was developed by Texas Instruments for the D-37C Minuteman II Guidance Computer in 1962, but these devices were not available to the public.

A variant of DTL called "high-threshold logic" incorporated Zener diodes to create a large offset between logic 1 and logic 0 voltage levels. These devices usually ran off a 15 volt power supply and were found in industrial control, where the high differential was intended to minimize the effect of noise.

ECL

The ECL family, ECL is also known as current-mode logic (CML), was invented by IBM as current steering logic for use in the transistorized IBM 7030 Stretch computer, where it was implemented using discrete components.

The first ECL logic family to be available in integrated circuits was introduced by Motorola as *MECL* in 1962.

TTL

The first transistor-transistor logic family of integrated circuits was introduced by Sylvania as *Sylvania Universal High–Level Logic* (SUHL) in 1963. Texas Instruments introduced 5400 Series TTL family in 1964.

Transistor–transistor logic uses bipolar transistors to form its integrated circuits. TTL has changed significantly over the years, with newer versions replacing the older types.

Since the transistors of a standard TTL gate are saturated switches, minority carrier storage time in each junction limits the switching speed of the device. Variations on the basic TTL design are intended to reduce these effects and improve speed, power consumption, or both.

The German physicist Walter H. Schottky formulated a theory predicting the **Schottky effect**, which led to the Schottky diode and later Schottky transistors. Schottky transistors have a much higher switching speed than conventional transistors because the Schottky junction does not promote charge storage, leading to faster switching gates. Gates built with Schottky transistors use more power than normal TTL and switch faster. With **Low-power Schottky** (LS), internal resistance values were increased to reduce power consumption and increase switching speed over the original version. The introduction of **Advanced Low-power Schottky** (ALS) further increased speed and reduced power consumption. A faster logic family called **Fast** (Schottky) (F) was also introduced that was faster than normal Schottky TTL.

CMOS

CMOS logic gates use complementary arrangements of NMOS and PMOS FETs. Since the initial devices used oxide-isolated metal gates, they were called **CMOS** (complementary metal–oxide–semiconductor logic).

In contrast to TTL, CMOS uses almost no power in the static state (that is, when inputs are not changing). A CMOS gate draws no current other than leakage when in a steady 1 or 0 state. When the gate switches states, current is drawn from the power supply to charge the capacitance at the output of the gate. This means that the current draw of CMOS devices increases with switching rate (controlled by clock speed, typically).

The first CMOS family of logic integrated circuits was introduced by RCA as *CD4000 COS/MOS*, the 4000 series, in 1968. Initially CMOS logic was slower than LS-TTL. However, because the logic thresholds of CMOS were proportional to the power supply voltage, CMOS devices were well-adapted to battery-operated systems with simple power supplies. CMOS gates can also tolerate much wider voltage ranges than TTL gates because the logic thresholds are (approximately) proportional to power supply voltage, and not the fixed levels required by bipolar circuits.

With this technology the required silicon area for implementing such digital CMOS functions has rapidly shrunk. VLSI technology incorporating millions of basic logic operations onto one chip, almost exclusively uses CMOS. The extremely small capacitance of the on-chip wiring, caused an increase in performance by several orders of magnitude. On-chip clock rates as high as 4 GHz have become common, approximately 1000 times faster than the technology by 1970.

Lowering the power supply voltage

One very important feature of CMOS chips is that they work with a broader range of power supply voltages. While TTL ICs all require a power supply voltage of 5V (+/- 0.5V), CMOS works with a wider range of power supply voltage – usually anywhere from 3 to 15V. Lowering the supply voltage reduces the current required to charge stray capacitance, and so reduces the current drawn by complex microprocessors. This in turn reduces the heat dissipation of the processor. By lowering the power supply from 5V to 3.3V, switching power was reduced by almost 60 percent (power dissipation is proportional to the square of the supply voltage). Newer CPUs have lowered their power supply voltages further.

HC logic

Because of the incompatibility of the CD4000 series of chips with the previous TTL family, a new standard emerged which combined the best of the TTL family with the advantages of the CD4000 family. It was known as the 74HC (High-performance silicon gate) family of devices and used the pinout of the 74LS family with an improved version of CMOS technology inside the chip. It could be used both with logic devices which used

3.3V power supplies (and thus 3.3V logic levels), and with devices that used 5V power supplies and TTL logic levels.

The CMOS–TTL logic level problem

Interconnecting any two logic families often required special techniques such as additional pull-up resistors, or purpose-built interface circuits, since the logic families may use different voltage levels to represent 1 and 0 states, and may have other interface requirements only met within the logic family.

TTL logic levels are different from those of CMOS – generally a TTL output does not rise high enough to be reliably recognized as a logic 1 by a CMOS input. This problem was solved by the invention of the 74HCT family of devices that uses CMOS technology but TTL input logic levels. These devices only work with a 5V power supply. They form a replacement for TTL, although HCT is slower than original TTL (HC logic has about the same speed as original TTL).

Other CMOS families

Other CMOS circuit families within integrated circuits include cascode voltage switch logic (CVSL) and pass transistor logic (PTL) of various sorts. These are generally used "on-chip" and are not delivered as building-block medium-scale or small-scale integrated circuits.

BiCMOS

One major improvement was to combine CMOS inputs and TTL drivers to form of a new type of logic devices called BiCMOS logic, of which the LVT and ALVT logic families are the most important. The BiCMOS family has many members, including ABT logic, ALB logic, ALVT logic, BCT logic and LVT logic.

Improved versions

With HC and HCT logic and LS-TTL logic competing in the market it became clear that further improvements were needed to create the *ideal* logic device that combined high speed, with low power dissipation and compatibility with older logic families. A whole range of newer families has emerged that use CMOS technology. A short list of the most important family designators of these newer devices includes:

- LV logic (lower supply voltage)
- LVT logic (lower supply voltage while retaining TTL logic levels)
- ALVT logic (an 'advanced' version of LVT logic)

There are many others including AC/ACT logic, AHC/AHCT logic, ALVC logic, AUC logic, AVC logic, CBT logic, CBTLV logic, FCT logic and LVC logic.

Monolithic integrated circuit logic families table

The following logic families would either have been used to build up systems from functional blocks such as flip-flops, counters, and gates, or else would be used as "glue" logic to interconnect very-large scale integration devices such as memory and processors. Not shown are some early obscure logic families from the early 1960's such as DCTL (direct-coupled transistor logic), which did not become widely available.

Propagation delay is the time taken for a two-input NAND gate to produce a result after a change of state at its inputs. *Toggle speed* represents the fastest speed at which a J-K flip flop could operate. *Power per gate* is for an individual 2-input NAND gate; usually there would be more than one gate per IC package. Values are very typical and would vary slightly depending on application conditions, manufacturer, temperature, and particular type of logic circuit. *Introduction year* is when at least some of the devices of the family were available in volume for civilian uses. Some military applications pre-dated civilian use.

Family	Description	Propagation delay (ns)	Toggle speed (MHz)	Power per gate @1 MHz (mW)	Typical supply voltage V (range)	Introduction year	Remarks
RTL	Resistor–transistor logic	—	4	10	3.3	1963	the first CPU built from integrated circuits (the Apollo Guidance Computer) used RTL.
DTL	Diode–transistor logic	—	—	10	5	1962	Introduced by Signetics, Fairchild 930 line became industry standard in 1964
CMOS	AC/ACT	3	125	0.5	3.3 or 5 (2-6 or 4.5-5.5)	1985	ACT has TTL Compatible levels
CMOS	HC/HCT	9	30	0.5	5 (2-6 or 4.5-5.5)	1982	HCT has TTL compatible levels
CMOS	4000B/74C	30	5	1.2	10V (3-18)	1970	Approximately half speed and power at 5 volts
TTL	Original series	10	25	10	5 (4.75-5.25)	1964	Several manufacturers
TTL	L	33	3	1	5 (4.75-5.25)	1964	Low power
TTL	H	6	43	22	5 (4.75-5.25)	1964	High speed
TTL	S	3	110	19	5 (4.75-5.25)	1969	Schottky high speed
TTL	LS	10	33	2	5 (4.75-5.25)	1976	Low power Schottky high speed

TTL	ALS	4	34	1.3	5 (4.5-5.5)	1976	Advanced Low power Schottky
TTL	F	3.5	100	5.4	5 (4.75-5.25)	1979	Fast
TTL	AS	2	105	8	5 (4.5-5.5)	1980	Advanced Schottky
TTL	G	1.5	1125 (1.125 GHz)		1.65 - 3.6	2004	First GHz 7400 series logic
ECL	ECL III	1	500	60	-5.2(-5.19 - -5.21)	1968	Improved ECL
ECL	MECL I	8		31	-5.2	1962	first integrated logic circuit commercially produced
ECL	ECL 10K	2	125	25	-5.2(-5.19 - -5.21)	1971	Motorola
ECL	ECL 100K	.75	350	40	-4.5(-4.2 - -5.2)	1981	
ECL	ECL 100KH	1	250	25	-5.2(-4.9 - -5.5)	1981	