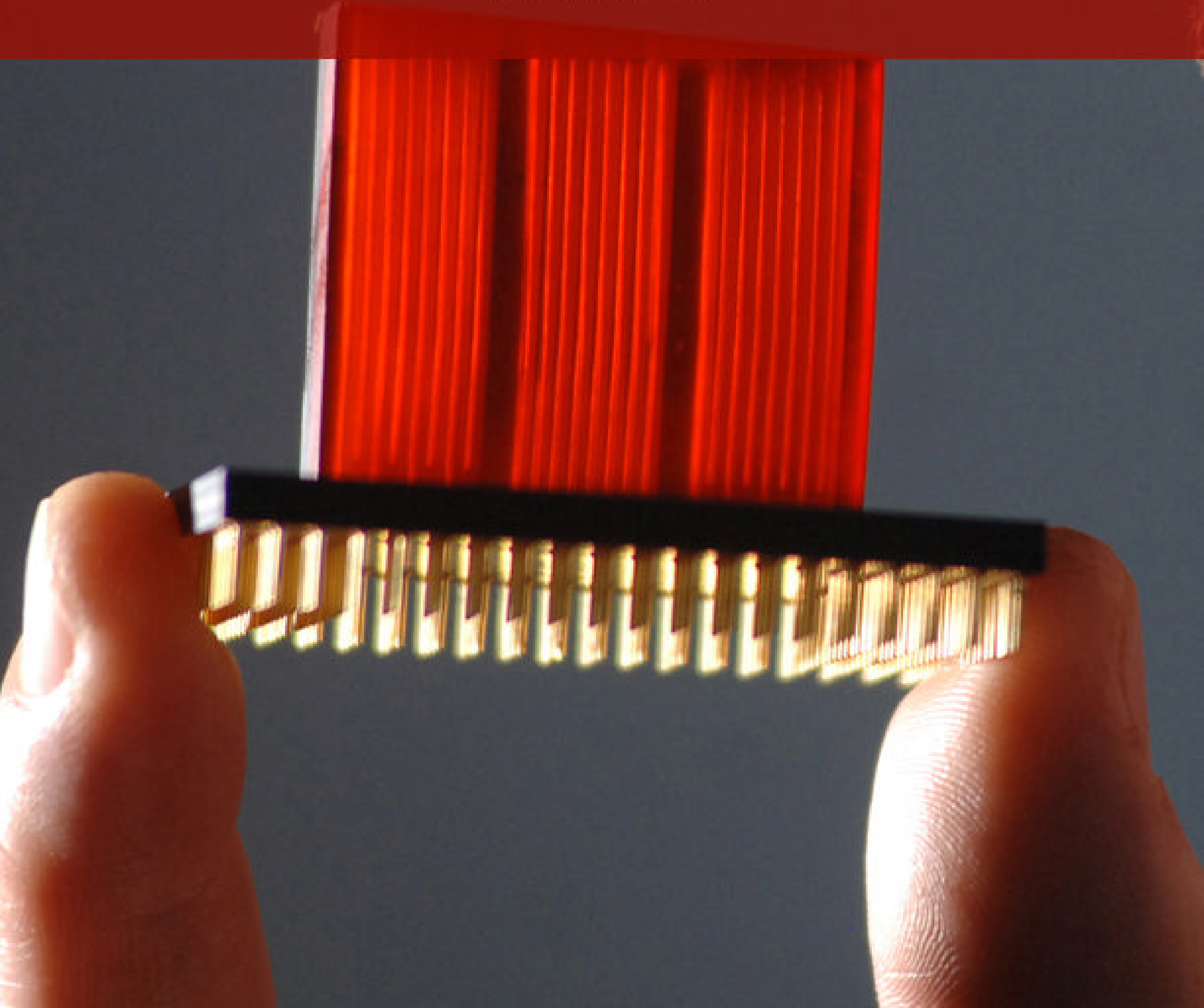
Electronic Engineering

Julia Aldred



First Edition, 2009

ISBN 978 81 907188 9 9

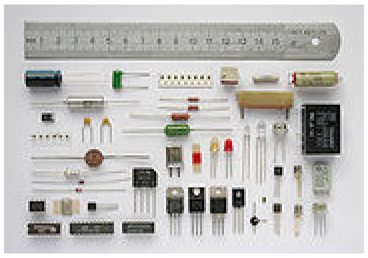
Published by:

Global Media
1819, Bhagirath Palace,
Chandni Chowk, Delhi-110 006
Email: globalmedia@dkpd.com

Table of Contents

Introduction

Electronic engineering



Electronic components

Electronics Engineering, also referred to as **Electronic Engineering**, is an engineering discipline which uses the scientific knowledge of the behavior and effects of electrons to develop components, devices, systems, or equipment (as in electron tubes, transistors, integrated circuits, and printed circuit boards) that uses electrictity as part of its driving force. Both terms denote a broad engineering field that encompasses many subfields including those that deal with power, instrumentation engineering, telecommunications, and semiconductor circuit design amongst many others.

The term also covers a large part of electrical engineering degree courses as studied at most European universities. In the U.S., however, electrical engineering implies all the wide electrical disciplines including electronics. The Institute of Electrical and Electronics Engineers is one of the most important and influential organizations for electronic engineers. Turkish universities have departments of **Electronic and Electrical Engineering**).

Terminology

The name electrical engineering is still used to cover electronic engineering amongst some of the older (notably American) universities and graduates there are called electrical engineers. The distinction between electronic and electrical engineers is becoming more and more distinct. While electrical engineers utilize voltage and current to deliver power, electronic engineers utilize voltage and current to deliver information through information technology.

Some people believe the term "electrical engineer" should be reserved for those having specialized in power and heavy current or high voltage engineering, while others believe that power is just one subset of electrical engineering (and indeed the term "power engineering" is used in that industry) as well as "electrical distribution engineering". Again, in recent years there has been a growth of new separate-entry degree courses such as "information engineering" and "communication systems engineering", often followed by academic departments of similar name.

Most European universities now refer to electrical engineering as power engineers and make a distinction between Electrical and Electronics Engineering. Beginning in the 1980s, the term computer engineer was often used to refer to electronic or information engineers. However, Computer Engineering is now considered a subset of Electronics Engineering and the term is now becoming archaic.

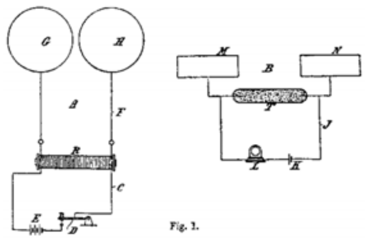
Chapter-1 History of Electronic Engineering

Electronic engineering as a profession sprang from technological improvements in the telegraph industry in the late 1800s and the radio and the telephone industries in the early 1900s. People were attracted to radio by the technical fascination it inspired, first in receiving and then in transmitting. Many who went into broadcasting in the 1920s were only "amateurs" in the period before World War I.

The modern discipline of electronic engineering was to a large extent born out of telephone, radio, and television equipment development and the large amount of electronic systems development during World War II of radar, sonar, communication systems, and advanced munitions and weapon systems. In the interwar years, the subject was known as radio engineering and it was only in the late 1950s that the term **electronic engineering** started to emerge.

The electronic laboratories (Bell Labs in the United States for instance) created and subsidized by large corporations in the industries of radio, television, and telephone equipment began churning out a series of electronic advances. In 1948, came the transistor and in 1960, the IC to revolutionize the electronic industry. In the UK, the subject of electronic engineering became distinct from electrical engineering as a university degree subject around 1960. Before this time, students of electronics and related subjects like radio and telecommunications had to enroll in the electrical engineering department of the university as no university had departments of electronics. Electrical engineering was the nearest subject with which electronic engineering could be aligned, although the similarities in subjects covered (except mathematics and electromagnetism) lasted only for the first year of the three-year course.

Early electronics



1896 Marconi patent

In 1893, Nikola Tesla made the first public demonstration of radio communication. Addressing the Franklin Institute in Philadelphia and the National Electric Light Association, he described and demonstrated in detail the principles of radio communication. In 1896, Guglielmo Marconi went on to develop a practical and widely used radio system. In 1904, John Ambrose Fleming, the first professor of electrical Engineering at University College London, invented the first radio tube, the diode. One year later, in 1906, Robert von Lieben and Lee De Forest independently developed the amplifier tube, called the triode.

Electronics is often considered to have begun when Lee De Forest invented the vacuum tube in 1907 . Within 10 years, his device was used in radio transmitters and receivers as well as systems for long distance telephone calls. In 1912, Edwin H. Armstrong invented the regenerative feedback amplifier and oscillator; he also invented the superheterodyne radio receiver and could be considered the "Father of Modern Radio". Vacuum tubes remained the preferred amplifying device for 40 years, until researchers working for William Shockley at Bell Labs invented the transistor in 1947 . In the following years, transistors made small portable radios, or transistor radios, possible as well as allowing more powerful mainframe computers to be built. Transistors were smaller and required lower voltages than vacuum tubes to work.In the interwar years the subject of electronics was dominated by the worldwide interest in *radio* and to some extent telephone and telegraph communications. The terms 'wireless' and 'radio' were then used to refer to anything electronic. There were indeed few non-military applications of electronics beyond radio at that time until the advent of television. The subject was not even offered as a separate university degree subject until about 1960.

Prior to the second world war, the subject was commonly known as 'radio engineering' and basically was restricted to aspects of communications and RADAR, commercial radio and early television. At this time, study of radio engineering at universities could only be undertaken as part of a physics degree. Later, in post war years, as consumer devices began to be developed, the field broadened to include modern TV, audio systems, Hi-Fi and latterly computers and microprocessors. In the mid to late 1950s, the term radio engineering gradually gave way to the name electronic engineering, which then became a stand alone university degree subject, usually taught alongside electrical engineering with which it had become associated due to some similarities.

Before the invention of the integrated circuit in 1959, electronic circuits were constructed from discrete components that could be manipulated by hand. These non-integrated circuits consumed much space and power, were prone to failure and were *limited in speed although* they are still common in simple applications. By contrast, integrated circuits packed a large number — often millions — of tiny electrical components, mainly transistors, into a small chip around the size of a coin.

Tubes or valves

The vacuum tube detector

The invention of the triode amplifier, generator, and detector made audio communication by radio practical. (Reginald Fessenden's 1906 transmissions used an electro-mechanical alternator.) The first known radio news program was broadcast 31 August 1920 by station 8MK, the unlicensed predecessor of WWJ (AM) in Detroit, Michigan. Regular wireless broadcasts for entertainment commenced in 1922 from the Marconi Research Centre at Writtle near Chelmsford, England.

While some early radios used some type of amplification through electric current or battery, through the mid 1920s the most common type of receiver was the crystal set. In the 1920s, amplifying vacuum tubes revolutionized both radio receivers and transmitters.

Phonographs and radiogrammes

This is the early name for record players or combined radios and record player

Television

In 1928 Philo Farnsworth made the first public demonstration of purely electronic television. During the 1930s several countries began broadcasting, and after World War II it spread to millions of receivers, eventually worldwide.

Ever since then, electronics have been fully present in television devices. Nowadays, electronics in television have evolved to be the basics of almost every component inside TV’s.

One of the latest and most advance technologies in TV screens/displays has to do entirely with electronics principles, and it’s the OLED (organic light emitting diode) displays, and it’s most likely to replace LCD and Plasma technologies.

Radar and radio location

During World War II many efforts were expended in the electronic location of enemy targets and aircraft. These included radio beam guidance of bombers, electronic counter measures, early radar systems etc. During this time very little if any effort was expended on consumer electronics developments.

Computers

History of computing hardware

History of computing hardware

The **history of computing hardware** encompasses the hardware, its architecture, and its impact on software. The elements of computing hardware have undergone significant improvement over their history. This improvement has triggered worldwide use of the technology, performance has improved and the price has declined. Computers are accessible to ever-increasing sectors of the world's population. Computing hardware has become a platform for uses other than computation, such as automation, communication, control, entertainment, and education. Each field in turn has imposed its own requirements on the hardware, which has evolved in response to those requirements.

The von Neumann architecture unifies current computing hardware implementations. Since digital computers rely on digital storage, and tend to be limited by the size and speed of memory, the history of computer data storage is tied to the development of computers. The major elements of computing hardware implement abstractions: input, output, memory, and processor. A processor is composed of control and datapath. In the von Neumann architecture, control of the datapath is stored in memory. This allowed control to become an automatic process; the datapath could be under software control, perhaps in response to events. Beginning with mechanical datapaths such as the abacus and astrolabe, the hardware first started using analogs for a computation, including water and even air as the analog quantities: analog computers have used lengths, pressures, voltages, and currents to represent the results of calculations. Eventually the voltages or currents were standardized, and then digitized. Digital computing elements have ranged from mechanical gears, to electromechanical relays, to vacuum tubes, to transistors, and to integrated circuits, all of which are currently implementing the von Neumann architecture.

History of computing
Hardware before 1960
Hardware 1960s to present
Hardware in Soviet Bloc countries
Artificial intelligence
Computer science
Operating systems
Programming languages
Software engineering

Graphical user interface

Internet

Personal computers

Laptops

Video games

World Wide Web

Timeline of computing

- Timeline of computing 2400 BC–1949
- 1950–1979
- 1980–1989
- 1990-1999
- 2000—
- More timelines...

Before computer hardware

The first use of the word "computer" was recorded in 1613, referring to a person who carried out calculations, or computations, and the word continued to be used in that sense until the middle of the 20th century. From the end of the 19th century onwards though, the word began to take on its more familiar meaning, describing a machine that carries out computations.

Earliest hardware

Calculator

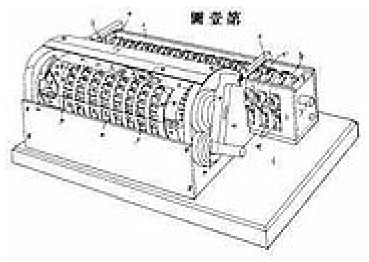
Devices have been used to aid computation for thousands of years, using one-to-one correspondence with our fingers. The earliest counting device was probably a form of tally stick. Later record keeping aids throughout the Fertile Crescent included calculi (clay spheres, cones, etc.) which represented counts of items, probably livestock or grains, sealed in containers. Counting rods is one example.

The abacus was used for arithmetic tasks. The Roman abacus was used in Babylonia as early as 2400 BC. Since then, many other forms of reckoning boards or tables have been invented. In a medieval counting house, a checkered cloth would be placed on a table, and markers moved around on it according to certain rules, as an aid to calculating sums of money (this is the origin of "Exchequer" as a term for a nation's treasury).

A number of analog computers were constructed in ancient and medieval times to perform astronomical calculations. These include the Antikythera mechanism and the astrolabe from ancient Greece (c. 150–100 BC), which are generally regarded as the first mechanical analog computers. Other early versions of mechanical devices used to perform some type of calculations include the planisphere and other mechanical computing devices invented by Abū Rayhān al-Bīrūnī (c. AD 1000); the equatorium and universal latitude-independent astrolabe by Abū Ishāq Ibrāhīm al-Zarqālī (c. AD 1015); the astronomical analog computers of other medieval Muslim astronomers and engineers; and the astronomical clock tower of Su Song (c. AD 1090) during the Song Dynasty.

The "castle clock", an astronomical clock invented by Al-Jazari in 1206, is considered to be the earliest programmable analog computer. It displayed the zodiac, the solar and lunar orbits, a crescent moon-shaped pointer traveling across a gateway causing automatic doors to open every hour, and five robotic musicians who play music when struck by levers operated by a camshaft attached to a water wheel. The length of day and night could be re-programmed every day in order to account for the changing lengths of day and night throughout the year.

Scottish mathematician and physicist John Napier noted multiplication and division of numbers could be performed by addition and subtraction, respectively, of logarithms of those numbers. While producing the first logarithmic tables Napier needed to perform many multiplications, and it was at this point that he designed Napier's bones, an abacus-like device used for multiplication and division. Since real numbers can be represented as distances or intervals on a line, the slide rule was invented in the 1620s to allow multiplication and division operations to be carried out significantly faster than was previously possible. Slide rules were used by generations of engineers and other mathematically inclined professional workers, until the invention of the pocket calculator. The engineers in the Apollo program to send a man to the moon made many of their calculations on slide rules, which were accurate to three or four significant figures.



Yazu Arithmometer. Patented in Japan in 1903. Note the lever for turning the gears of the calculator.

German polymath Wilhelm Schickard built the first digital mechanical calculator in 1623, and thus became the father of the computing era. Since his calculator used techniques such as cogs and gears first developed for clocks, it was also called a 'calculating clock'. It was put to practical use by his friend Johannes Kepler, who revolutionized astronomy when he condensed decades of astronomical observations into algebraic expressions. An original calculator by Pascal (1640) is preserved in the Zwinger Museum. Machines by Blaise Pascal (the Pascaline, 1642) and Gottfried Wilhelm von Leibniz (the Stepped Reckoner, c. 1672) followed. Leibniz once said "It is unworthy of excellent men to lose hours like slaves in the labour of calculation which could safely be relegated to anyone else if machines were used."

Around 1820, Charles Xavier Thomas created the first successful, mass-produced mechanical calculator, the Thomas Arithmometer, that could add, subtract, multiply, and divide. It was mainly based on Leibniz' work. Mechanical calculators, like the base-ten addiator, the comptometer, the Monroe, the Curta and the Addo-X remained in use until the 1970s. Leibniz also described the binary numeral system, a central ingredient of all modern computers. However, up to the 1940s, many subsequent designs (including Charles Babbage's machines of the 1800s and even ENIAC of 1945) were based on the decimal system; ENIAC's ring counters emulated the operation of the digit wheels of a mechanical adding machine.

In Japan, Ryoichi Yazu patented a mechanical calculator called the Yazu Arithmometer in 1903. It consisted of a single cylinder and 22 gears, and employed the mixed base-2 and base-5 number system familiar to users to the soroban (Japanese abacus). Carry and end of calculation were determined automatically. More than 200 units were sold, mainly to government agencies such as the Ministry of War and agricultural experiment stations. Yazu invested the profits in a factory to build what would have been Japan's first propeller-driven airplane, but that project was abandoned after his untimely death at the age of 31.

1801: punched card technology

analytical engine

Punched card system of a music machine, also referred to as Book music, a one-stop European medium for organs

As early as 1725 Basile Bouchon used a perforated paper loop in a loom to establish the pattern to be reproduced on cloth, and in 1726 his co-worker Jean-Baptiste Falcon improved on his design by using perforated paper cards attached to one another for efficiency in adapting and changing the program. The Bouchon-Falcon loom was semi-automatic and required manual feed of the program. In 1801, Joseph-Marie Jacquard developed a loom in which the pattern being woven was controlled by punched cards. The series of cards could be changed without changing the mechanical design of the loom. This was a landmark point in programmability.

In 1833, Charles Babbage moved on from developing his difference engine to developing a more complete design, the analytical engine, which would draw directly on Jacquard's punched cards for its programming. In 1835, Babbage described his analytical engine. It was the plan of a general-purpose programmable computer, employing punch cards for input and a steam engine for power. One crucial invention was to use gears for the function served by the beads of an abacus. In a real sense, computers all contain automatic abacuses (the *datapath*, arithmetic logic unit, or floating-point unit). His initial idea was to use punch-cards to control a machine that could calculate and print logarithmic tables with huge precision (a specific purpose machine). Babbage's idea soon developed into a general-purpose programmable computer, his analytical engine. While his design was sound and the plans were probably correct, or at least debuggable, the project was slowed by various problems. Babbage was a difficult man to work with and argued with anyone who didn't respect his ideas. All the parts for his machine had to be made by hand. Small errors in each item can sometimes sum up to large discrepancies in a machine with thousands of parts, which required these parts to be much better than the usual tolerances needed at the time. The project dissolved in disputes with the artisan who built parts and was ended with the depletion of government funding. Ada Lovelace, Lord Byron's daughter, translated and added notes to the "*Sketch of the Analytical Engine*" by Federico Luigi, Conte Menabrea.

IBM 407 tabulating machine, (1961)

A reconstruction of the Difference Engine II, an earlier, more limited design, has been operational since 1991 at the London Science Museum. With a few trivial changes, it works as Babbage designed it and shows that Babbage was right in theory. The museum used computer-operated machine tools to construct the necessary parts, following tolerances which a machinist of the period would have been able to achieve. The failure of Babbage to complete the engine can be chiefly attributed to difficulties not only related to politics and financing, but also to his desire to develop an increasingly sophisticated computer. Following in the footsteps of Babbage, although unaware of his earlier work, was Percy Ludgate, an accountant from Dublin, Ireland. He independently designed a programmable mechanical computer, which he described in a work that was published in 1909.

In the late 1880s, the American Herman Hollerith invented the recording of data on a medium that could then be read by a machine. Prior uses of machine readable media had been for control (automatons, piano rolls, looms, ...), not data. "After some initial trials with paper tape, he settled on punched cards..." (Hollerith came to use punched cards after observing how railroad conductors encoded personal characteristics of each passenger with punches on their tickets.) To process these punched cards, first known as "Hollerith cards" he invented the tabulator, and the key punch machines. These three inventions were the foundation of the modern information processing industry. His machines used mechanical relays (and solenoids) to increment mechanical counters. Hollerith's method was used in the 1890 United States Census and the completed results were "... finished months ahead of schedule and far under budget". Hollerith's company eventually became the core of IBM. IBM developed punch card technology into a powerful tool for business data-processing and produced an extensive line of unit record equipment. By 1950, the IBM card had become ubiquitous in industry and government. The warning printed on most cards intended for circulation as documents (checks, for example), "Do not fold, spindle or mutilate," became a motto for the post-World War II era.

Punched card with the extended alphabet

Leslie Comrie's articles on punched card methods and W.J. Eckert's publication of *Punched Card Methods in Scientific Computation* in 1940, described techniques which were sufficiently advanced to solve differential equations or perform multiplication and division using floating point representations, all on punched cards and unit record machines. In the image of the tabulator (see left), note the patch panel, which is visible on the right side of the tabulator. A row of toggle switches is above the patch panel. The Thomas J. Watson Astronomical Computing Bureau, Columbia University performed astronomical calculations representing the state of the art in computing.

Computer programming in the punch card era revolved around the computer center. The computer users, for example, science and engineering students at universities, would submit their programming assignments to their local computer center in the form of a stack of cards, one card per program line. They then had to wait for the program to be queued for processing, compiled, and executed. In due course a printout of any results, marked with the submitter's identification, would be placed in an output tray outside the computer center. In many cases these results would comprise solely a printout of error messages regarding program syntax *etc.*, necessitating another edit-compile-run cycle. Punched cards are still used and manufactured to this day, and their distinctive dimensions[41] (and 80-column capacity) can still be recognized in forms, records, and programs around the world.

1930s–1960s: desktop calculators

Post–Turing machine

The Curta calculator can also do multiplication and division

By the 1900s, earlier mechanical calculators, cash registers, accounting machines, and so on were redesigned to use electric motors, with gear position as the representation for the state of a variable. The word "computer" was a job title assigned to people who used these calculators to perform mathematical calculations. By the 1920s Lewis Fry Richardson's interest in weather prediction led him to propose human computers and numerical analysis to model the weather; to this day, the most powerful computers on Earth are needed to adequately model its weather using the Navier-Stokes equations.[42]

Companies like Friden, Marchant Calculator and Monroe made desktop mechanical calculators from the 1930s that could add, subtract, multiply and divide. During the Manhattan project, future Nobel laureate Richard Feynman was the supervisor of the roomful of human computers, many of them women mathematicians, who understood the differential equations which were being solved for the war effort. Even the renowned Stanisław Ulam was pressed into service to translate the mathematics into computable approximations for the hydrogen bomb,[43] after the war.

In 1948, the Curta was introduced. This was a small, portable, mechanical calculator that was about the size of a pepper grinder. Over time, during the 1950s and 1960s a variety of different brands of mechanical calculator appeared on the market. The first all-electronic desktop calculator was the British ANITA Mk.VII, which used a Nixie tube display and 177 subminiature thyratron tubes. In June 1963, Friden introduced the four-function EC-130. It had an all-transistor design, 13-digit capacity on a 5-inch (130 mm) CRT, and introduced reverse Polish notation (RPN) to the calculator market at a price of $2200. The model EC-132 added square root and reciprocal functions. In 1965, Wang Laboratories produced the LOCI-2, a 10-digit transistorized desktop calculator that used a Nixie tube display and could compute logarithms.

Advanced analog computers

analog computer

Cambridge differential analyzer, 1938

Before World War II, mechanical and electrical analog computers were considered the "state of the art", and many thought they were the future of computing. Analog computers take advantage of the strong similarities between the mathematics of small-scale properties—the position and motion of wheels or the voltage and current of electronic components—and the mathematics of other physical phenomena,[44] for example, ballistic trajectories, inertia, resonance, energy transfer, momentum, and so forth. They model physical phenomena with electrical voltages and currents[45][46] as the analog quantities.

Centrally, these analog systems work by creating electrical *analogs* of other systems, allowing users to predict behavior of the systems of interest by observing the electrical analogs.[47] The most useful of the analogies was the way the small-scale behavior could be represented with integral and differential equations, and could be thus used to solve those equations. An ingenious example of such a machine, using water as the analog quantity, was the water integrator built in 1928; an electrical example is the Mallock machine built in 1941. A planimeter is a device which does integrals, using distance as the analog quantity. Until the 1980s, HVAC systems used air both as the analog quantity and the controlling element. Unlike modern digital computers, analog computers are not very flexible, and need to be reconfigured (i.e., reprogrammed) manually to switch them from working on one problem to another. Analog computers had an advantage over early digital computers in that they could be used to solve complex problems using behavioral analogues while the earliest attempts at digital computers were quite limited.

Since computers were rare in this era, the solutions were often *hard-coded* into paper forms such as nomograms,[48] which could then produce analog solutions to these problems, such as the distribution of pressures and temperatures in a heating system. Some of the most widely deployed analog computers included devices for aiming weapons, such as the Norden bombsight[49] and the fire-control systems,[50] such as Arthur Pollen's Argo system for naval vessels. Some stayed in use for decades after WWII; the Mark I Fire Control Computer was deployed by the United States Navy on a variety of ships from destroyers to battleships. Other analog computers included the Heathkit EC-1, and the hydraulic MONIAC Computer which modeled econometric flows.[51]

The art of analog computing reached its zenith with the differential analyzer,[52] invented in 1876 by James Thomson and built by H. W. Nieman and Vannevar Bush at MIT starting in 1927. Fewer than a dozen of these devices were ever built; the most powerful was constructed at the University of Pennsylvania's Moore School of Electrical Engineering, where the ENIAC was built. Digital electronic computers like the ENIAC spelled the end for most analog computing machines, but hybrid analog computers, controlled by digital electronics, remained in substantial use into the 1950s and 1960s, and later in some specialized applications. But like all digital devices, the decimal precision of a digital device is a limitation,[53] as compared to an analog device, in which the accuracy is a limitation.[54] As electronics progressed during the twentieth century, its problems of operation at low voltages while maintaining high signal-to-noise ratios[55] were steadily addressed, as shown below, for a digital circuit is a specialized form of analog circuit, intended to operate at standardized settings (continuing in the same vein, logic gates can be realized as forms of digital circuits). But as digital computers have become faster and use larger memory (for example, RAM or internal storage), they have almost entirely displaced analog computers. Computer programming, or coding, has arisen as another human profession.

Digital computation

Punched tape programs would be much longer than the short fragment of yellow paper tape shown.

The era of modern computing began with a flurry of development before and during World War II, as electronic circuit elements replaced mechanical equivalents and digital calculations replaced analog calculations. Machines such as the Z3, the Atanasoff–Berry Computer, the Colossus computers, and the ENIAC were built by hand using circuits containing relays or valves (vacuum tubes), and often used punched cards or punched paper tape for input and as the main (non-volatile) storage medium.

In this era, a number of different machines were produced with steadily advancing capabilities. At the beginning of this period, nothing remotely resembling a modern computer existed, except in the long-lost plans of Charles Babbage and the mathematical ideas of Alan Turing. At the end of the era, devices like the Colossus computers and the EDSAC had been built, and are agreed to be electronic digital computers. Defining a single point in the series as the "first computer" misses many subtleties (see the table "Defining characteristics of some early digital computers of the 1940s" below).

Alan Turing's 1936 paper proved enormously influential in computing and computer science in two ways. Its main purpose was to prove that there were problems (namely the halting problem) that could not be solved by any sequential process. In doing so, Turing provided a definition of a universal computer which executes a program stored on tape. This construct came to be called a Turing machine; it replaces Kurt Gödel's more cumbersome universal language based on arithmetics. Except for the limitations imposed by their finite memory stores, modern computers are said to be Turing-complete, which is to say, they have algorithm execution capability equivalent to a universal Turing machine.

Nine-track magnetic tape

For a computing machine to be a practical general-purpose computer, there must be some convenient read-write mechanism, punched tape, for example. With a knowledge of Alan Turing's theoretical 'universal computing machine' John von Neumann defined an architecture which uses the same memory both to store programs and data: virtually all contemporary computers use this architecture (or some variant). While it is theoretically possible to implement a full computer entirely mechanically (as Babbage's design showed), electronics made possible the speed and later the miniaturization that characterize modern computers.

There were three parallel streams of computer development in the World War II era; the first stream largely ignored, and the second stream deliberately kept secret. The first was the German work of Konrad Zuse. The second was the secret development of the Colossus computers in the UK. Neither of these had much influence on the various computing projects in the United States. The third stream of computer development, Eckert and Mauchly's ENIAC and EDVAC, was widely publicized.

George Stibitz is internationally recognized as one of the fathers of the modern digital computer. While working at Bell Labs in November 1937, Stibitz invented and built a relay-based calculator that he dubbed the "Model K" (for "kitchen table", on which he had assembled it), which was the first to calculate using binary form.

Zuse

Konrad Zuse

A reproduction of Zuse's Z1 computer

Working in isolation in Germany, Konrad Zuse started construction in 1936 of his first Z-series calculators featuring memory and (initially limited) programmability. Zuse's purely mechanical, but already binary Z1, finished in 1938, never worked reliably due to problems with the precision of parts.

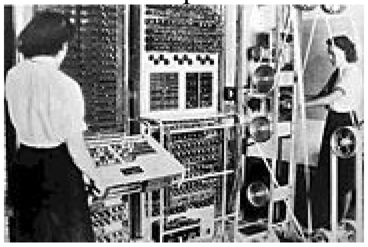
Zuse's later machine, the Z3, was finished in 1941. It was based on telephone relays and did work satisfactorily. The Z3 thus became the first functional program-controlled, all-purpose, digital computer. In many ways it was quite similar to modern machines, pioneering numerous advances, such as floating point numbers. Replacement of the hard-to-implement decimal system (used in Charles Babbage's earlier design) by the simpler binary system meant that Zuse's machines were easier to build and potentially more reliable, given the technologies available at that time.

Programs were fed into Z3 on punched films. Conditional jumps were missing, but since the 1990s it has been proved theoretically that Z3 was still a universal computer (ignoring its physical storage size limitations). In two 1936 patent applications, Konrad Zuse also anticipated that machine instructions could be stored in the same storage used for data—the key insight of what became known as the von Neumann architecture, first implemented in the British SSEM of 1948. Zuse also claimed to have designed the first higher-level programming language, (Plankalkül), in 1945 (published in 1948) although it was implemented for the first time in 2000 by a team around Raúl Rojas at the Free University of Berlin—five years after Zuse died.

Zuse suffered setbacks during World War II when some of his machines were destroyed in the course of Allied bombing campaigns. Apparently his work remained largely unknown to engineers in the UK and US until much later, although at least IBM was aware of it as it financed his post-war startup company in 1946 in return for an option on Zuse's patents.

Colossus

Colossus computer



Colossus was used to break German ciphers during World War II.

During World War II, the British at Bletchley Park (40 miles north of London) achieved a number of successes at breaking encrypted German military communications. The German encryption machine, Enigma, was attacked with the help of electro-mechanical machines called *bombes*. The bombe, designed by Alan Turing and Gordon Welchman, after the Polish cryptographic *bomba* by Marian Rejewski (1938), came into use in 1941.

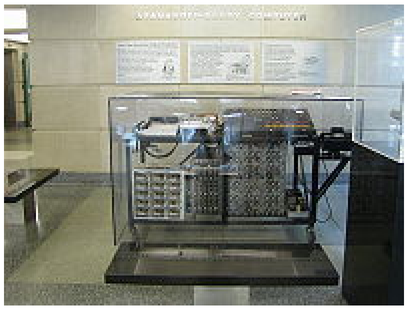
They ruled out possible Enigma settings by performing chains of logical deductions implemented electrically. Most possibilities led to a contradiction, and the few remaining could be tested by hand.

The Germans also developed a series of teleprinter encryption systems, quite different from Enigma. The Lorenz SZ 40/42 machine was used for high-level Army communications, termed "Tunny" by the British. The first intercepts of Lorenz messages began in 1941. As part of an attack on Tunny, Professor Max Newman and his colleagues helped specify the Colossus. The Mk I Colossus was built between March and December 1943 by Tommy Flowers and his colleagues at the Post Office Research Station at Dollis Hill in London and then shipped to Bletchley Park in January 1944.

Colossus was the first totally *electronic* computing device. The Colossus used a large number of valves (vacuum tubes). It had paper-tape input and was capable of being configured to perform a variety of boolean logical operations on its data, but it was not Turing-complete. Nine Mk II Colossi were built (The Mk I was converted to a Mk II making ten machines in total). Details of their existence, design, and use were kept secret well into the 1970s. Winston Churchill personally issued an order for their destruction into pieces no larger than a man's hand. Due to this secrecy the Colossi were not included in many histories of computing. A reconstructed copy of one of the Colossus machines is now on display at Bletchley Park.

American developments

In 1937, Claude Shannon showed there is a one-to-one correspondence between the concepts of Boolean logic and certain electrical circuits, now called logic gates, which are now ubiquitous in digital computers. In his master's thesis at MIT, for the first time in history, Shannon showed that electronic relays and switches can realize the expressions of Boolean algebra. Entitled *A Symbolic Analysis of Relay and Switching Circuits*, Shannon's thesis essentially founded practical digital circuit design. George Stibitz completed a relay-based computer he dubbed the "Model K" at Bell Labs in November 1937. Bell Labs authorized a full research program in late 1938 with Stibitz at the helm. Their *Complex Number Calculator*, completed January 8, 1940, was able to calculate complex numbers. In a demonstration to the American Mathematical Society conference at Dartmouth College on September 11, 1940, Stibitz was able to send the Complex Number Calculator remote commands over telephone lines by a teletype. It was the first computing machine ever used remotely, in this case over a phone line. Some participants in the conference who witnessed the demonstration were John von Neumann, John Mauchly, and Norbert Wiener, who wrote about it in their memoirs.



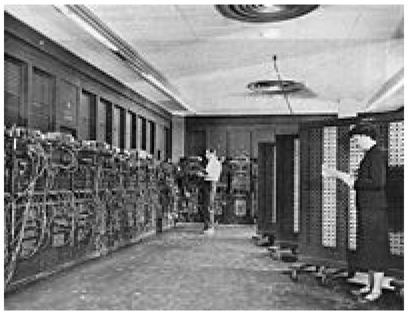
Atanasoff–Berry Computer replica at 1st floor of Durham Center, Iowa State University

In 1939, John Vincent Atanasoff and Clifford E. Berry of Iowa State University developed the Atanasoff–Berry Computer (ABC), The Atanasoff-Berry Computer was the world's first electronic digital computer. The design used over 300 vacuum tubes and employed capacitors fixed in a mechanically rotating drum for memory. Though the ABC machine was not programmable, it was the first to use electronic tubes in an adder. ENIAC co-inventor John Mauchly examined the ABC in June 1941, and its influence on the design of the later ENIAC machine is a matter of contention among computer historians. The ABC was largely forgotten until it became the focus of the lawsuit *Honeywell v. Sperry Rand*, the ruling of which invalidated the ENIAC patent (and several others) as, among many reasons, having been anticipated by Atanasoff's work.

In 1939, development began at IBM's Endicott laboratories on the Harvard Mark I. Known officially as the Automatic Sequence Controlled Calculator, the Mark I was a general purpose electro-mechanical computer built with IBM financing and with assistance from IBM personnel, under the direction of Harvard mathematician Howard Aiken. Its design was influenced by Babbage's Analytical Engine, using decimal arithmetic and storage wheels and rotary switches in addition to electromagnetic relays. It was programmable via punched paper tape, and contained several calculation units working in parallel. Later versions contained several paper tape readers and the machine could switch between readers based on a condition. Nevertheless, the machine was not quite Turing-complete. The Mark I was moved to Harvard University and began operation in May 1944.

ENIAC

ENIAC



ENIAC performed ballistics trajectory calculations with 160 kW of power.

The US-built ENIAC (Electronic Numerical Integrator and Computer) was the first electronic general-purpose computer. It combined, for the first time, the high speed of electronics with the ability to be programmed for many complex problems. It could add or subtract 5000 times a second, a thousand times faster than any other machine. (Colossus couldn't add.) It also had modules to multiply, divide, and square root. High speed memory was limited to 20 words (about 80 bytes.) Built under the direction of John Mauchly and J. Presper Eckert at the University of Pennsylvania, ENIAC's development and construction lasted from 1943 to full operation at the end of 1945. The machine was huge, weighing 30 tons, and contained over 18,000 valves. One of the major engineering feats was to minimize valve burnout, which was a common problem at that time. The machine was in almost constant use for the next ten years.

ENIAC was unambiguously a Turing-complete device. It could compute any problem (that would fit in memory). A "program" on the ENIAC, however, was defined by the states of its patch cables and switches, a far cry from the stored program electronic machines that evolved from it. Once a program was written, it had to be mechanically set into the machine. Six women did most of the programming of ENIAC. (Improvements completed in 1948 made it possible to execute stored programs set in function table memory, which made programming less a "one-off" effort, and more systematic.)

Defining characteristics of some early digital computers of the 1940s (In the history of computing hardware)

Name	First operational	Numeral system	Computing mechanism	Programming	Turing complete
Zuse Z3 (Germany)	May 1941	Binary	Electro-mechanical	Program-controlled by punched film stock (but no conditional branch)	Yes (1998)
Atanasoff–Berry Computer (US)	1942	Binary	Electronic	Not programmable—single purpose	No
Colossus Mark 1 (UK)	February 1944	Binary	Electronic	Program-controlled by patch cables and switches	No

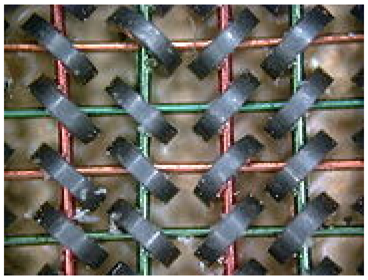
Harvard Mark I – IBM ASCC (US)	May 1944	Decimal	Electro-mechanical	Program-controlled by 24-channel punched paper tape (but no conditional branch)	No
Colossus Mark 2 (UK)	June 1944	Binary	Electronic	Program-controlled by patch cables and switches	No
ENIAC (US)	July 1946	Decimal	Electronic	Program-controlled by patch cables and switches	Yes
Manchester Small-Scale Experimental Machine (UK)	June 1948	Binary	Electronic	Stored-program in Williams cathode ray tube memory	Yes
Modified ENIAC (US)	September 1948	Decimal	Electronic	Program-controlled by patch cables and switches plus a primitive read-only stored programming mechanism using the Function Tables as program ROM	Yes
EDSAC (UK)	May 1949	Binary	Electronic	Stored-program in mercury delay line memory	Yes
Manchester Mark 1 (UK)	October 1949	Binary	Electronic	Stored-program in Williams cathode ray tube memory and magnetic drum memory	Yes

CSIRAC (Australia)	November 1949	Binary	Electronic	Stored-program in mercury delay line memory	Yes

First-generation von Neumann machines

Design of the von Neumann architecture (1947)

Even before the ENIAC was finished, Eckert and Mauchly recognized its limitations and started the design of a stored-program computer, EDVAC. John von Neumann was credited with a widely circulated report describing the EDVAC design in which both the programs and working data were stored in a single, unified store. This basic design, denoted the von Neumann architecture, would serve as the foundation for the worldwide development of ENIAC's successors. In this generation of equipment, temporary or working storage was provided by acoustic delay lines, which used the propagation time of sound through a medium such as liquid mercury (or through a wire) to briefly store data. A series of acoustic pulses is sent along a tube; after a time, as the pulse reached the end of the tube, the circuitry detected whether the pulse represented a 1 or 0 and caused the oscillator to re-send the pulse. Others used Williams tubes, which use the ability of a television picture tube to store and retrieve data. By 1954, magnetic core memory was rapidly displacing most other forms of temporary storage, and dominated the field through the mid-1970s.



Magnetic core memory. Each core is one bit.

EDVAC was the first stored-program computer designed; however it was not the first to run. Eckert and Mauchly left the project and its construction floundered. The first working von Neumann machine was the Manchester "Baby" or Small-Scale Experimental Machine, developed by Frederic C. Williams and Tom Kilburn at University of Manchester in 1948; it was followed in 1949 by the Manchester Mark 1 computer, a complete system, using Williams tube and magnetic drum memory, and introducing index registers. The other contender for the title "first digital stored program computer" had been EDSAC, designed and constructed at the University of Cambridge. Operational less than one year after the Manchester "Baby", it was also capable of tackling real problems. EDSAC was actually inspired by plans for EDVAC (Electronic Discrete Variable Automatic Computer), the successor to ENIAC; these plans were already in place by the time ENIAC was successfully operational. Unlike ENIAC, which used parallel processing, EDVAC used a single processing unit. This design was simpler and was the first to be implemented in each succeeding wave of miniaturization, and increased reliability. Some view Manchester Mark 1 / EDSAC / EDVAC as the "Eves" from which nearly all current computers derive their architecture. Manchester University's machine became the prototype for the Ferranti Mark 1. The first Ferranti Mark 1 machine was delivered to the University in February, 1951 and at least nine others were sold between 1951 and 1957.

The first universal programmable computer in the Soviet Union was created by a team of scientists under direction of Sergei Alekseyevich Lebedev from Kiev Institute of Electrotechnology, Soviet Union (now Ukraine). The computer MESM (*МЭСМ*, *Small Electronic Calculating Machine*) became operational in 1950. It had about 6,000 vacuum tubes and consumed 25 kW of power. It could perform approximately 3,000 operations per second. Another early machine was CSIRAC, an Australian design that ran its first test program in 1949. CSIRAC is the oldest computer still in existence and the first to have been used to play digital music.

Commercial computers

In October 1947, the directors of J. Lyons & Company, a British catering company famous for its teashops but with strong interests in new office management techniques, decided to take an active role in promoting the commercial development of computers. By 1951 the LEO I computer was operational and ran the world's first regular routine office computer job. On 17 November 1951, the J. Lyons company began weekly operation of a bakery valuations job on the LEO (Lyons Electronic Office). This was the first business application to go live on a stored program computer.

In June 1951, the UNIVAC I (Universal Automatic Computer) was delivered to the U.S. Census Bureau. Remington Rand eventually sold 46 machines at more than $1 million each ($8.2 million as of 2009). UNIVAC was the first "mass produced" computer; all predecessors had been "one-off" units. It used 5,200 vacuum tubes and consumed 125 kW of power. It used a mercury delay line capable of storing 1,000 words of 11 decimal digits plus sign (72-bit words) for memory. A key feature of the UNIVAC system was a newly invented type of metal magnetic tape, and a high-speed tape unit, for non-volatile storage. Magnetic media is still used in almost all computers.

In 1952, IBM publicly announced the IBM 701 Electronic Data Processing Machine, the first in its successful 700/7000 series and its first IBM mainframe computer. The IBM 704, introduced in 1954, used magnetic core memory, which became the standard for large machines. The first implemented high-level general purpose programming language, Fortran, was also being developed at IBM for the 704 during 1955 and 1956 and released in early 1957. (Konrad Zuse's 1945 design of the high-level language

Plankalkül was not implemented at that time.) A volunteer user group, which exists to this day, was founded in 1955 to share their software and experiences with the IBM 701.

IBM 650 front panel

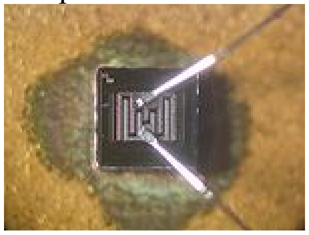
IBM introduced a smaller, more affordable computer in 1954 that proved very popular. The IBM 650 weighed over 900 kg, the attached power supply weighed around 1350 kg and both were held in separate cabinets of roughly 1.5 meters by 0.9 meters by 1.8 meters. It cost $500,000 ($3.96 million as of 2009) or could be leased for $3,500 a month ($30 thousand as of 2009). Its drum memory was originally 2,000 ten-digit words, later expanded to 4,000 words. Memory limitations such as this were to dominate programming for decades afterward. Efficient execution using drum memory was provided by a combination of hardware architecture: the instruction format included the address of the next instruction; and software: the Symbolic Optimal Assembly Program, SOAP, assigned instructions to optimal address (to the extent possible by static analysis of the source program). Thus many instructions were, when needed, located in the next row of the drum to be read and additional wait time for drum rotation was not required.

In 1955, Maurice Wilkes invented microprogramming, which allows the base instruction set to be defined or extended by built-in programs (now called firmware or microcode). It was widely used in the CPUs and floating-point units of mainframe and other computers, such as the IBM 360 series.

IBM introduced its first magnetic disk system, RAMAC (Random Access Method of Accounting and Control) in 1956. Using fifty 24-inch (610 mm) metal disks, with 100 tracks per side, it was able to store 5 megabytes of data at a cost of $10,000 per megabyte ($80 thousand as of 2009). (As of 2008, magnetic storage, in the form of hard disks, costs less than two cents per megabyte).

Second generation: transistors

computer architecture and von Neumann architecture



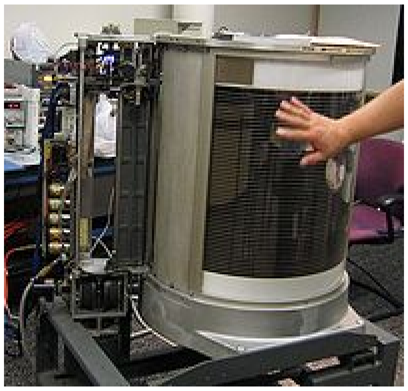
A bipolar junction transistor

By the early 1950s transistors started to become available, offering the possibility of building cheaper and faster computers. Initially the only devices available were germanium point-contact transistors, which although less reliable than the vacuum tubes they replaced had the advantage of consuming far less power. The first transistorised computer was built at the University of Manchester and was operational by 1953; a second version was completed there in April 1955. The later machine used 200 transistors and 1,300 solid-state diodes and had a power consumption of 150 watts. However, it still required valves to generate the clock waveforms at 125 kHz and to read and write on the magnetic drum memory, whereas the Harwell CADET operated without any valves by using a lower clock frequency, of 58 kHz when it became operational in February 1955. Problems with the reliability of early batches of point contact and alloyed junction transistors meant that the machine's mean time between failures was about 90 minutes, but this improved once the more reliable bipolar junction transistors became available.

The bipolar junction transistor (BJT) was invented in 1947. If no electrical current flows through the base-emitter path of a bipolar transistor, the transistor's collector-emitter path blocks electrical current (and the transistor is said to "turn full off"). If sufficient current flows through the base-emitter path of a transistor, that transistor's collector-emitter path also passes current (and the transistor is said to "turn full on"). Current flow or current blockage represent binary 1 (true) or 0 (false), respectively. From 1955 onwards bipolar junction transistors replaced vacuum tubes in computer designs, giving rise to the "second generation" of computers. Compared to vacuum tubes, transistors have many advantages: they are less expensive to manufacture and are much faster, switching from the condition 1 to 0 in millionths or billionths of a second. Transistor volume is measured in cubic millimeters compared to vacuum tubes' cubic centimeters. Transistors' lower operating temperature increased their reliability, compared to vacuum tubes. Transistorized computers could contain tens of thousands of binary logic circuits in a relatively compact space.

Initially, it was believed that very few computers would ever be produced or used. This was due in part to their size, cost, and the skill required to operate or interpret their results. Transistors greatly reduced computers' size, initial cost, and operating cost.

Typically, second-generation computers were composed of large numbers of printed circuit boards such as the IBM Standard Modular System each carrying one to four logic gates or flip-flops. A second generation computer, the IBM 1401, captured about one third of the world market. IBM installed more than one hundred thousand 1401s between 1960 and 1964— This period saw the only Italian attempt: the Olivetti ELEA, produced in 110 units.



This RAMAC DASD is being restored at the Computer History Museum

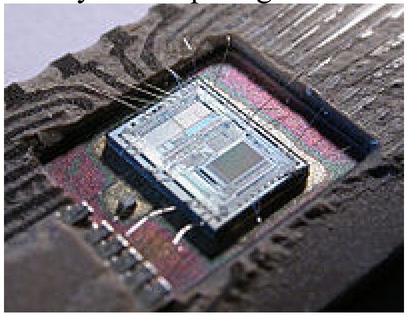
Transistorized electronics improved not only the CPU (Central Processing Unit), but also the peripheral devices. The IBM 350 RAMAC was introduced in 1956 and was the world's first disk drive. The second generation disk data storage units were able to store tens of millions of letters and digits. Multiple Peripherals can be connected to the CPU, increasing the total memory capacity to hundreds of millions of characters. Next to the fixed disk storage units, connected to the CPU via high-speed data transmission, were removable disk data storage units. A removable disk stack can be easily exchanged with another stack in a few seconds. Even if the removable disks' capacity is smaller than fixed disks,' their interchangeability guarantees a nearly unlimited quantity of data close at hand. But magnetic tape provided archival capability for this data, at a lower cost than disk.

Many second generation CPUs delegated peripheral device communications to a secondary processor. For example, while the communication processor controlled card reading and punching, the main CPU executed calculations and binary branch instructions. One databus would bear data between the main CPU and core memory at the CPU's fetch-execute cycle rate, and other databusses would typically serve the peripheral devices. On the PDP-1, the core memory's cycle time was 5 microseconds; consequently most arithmetic instructions took 10 microseconds (100,000 operations per second) because most operations took at least two memory cycles; one for the instruction, one for the operand data fetch.

During the second generation remote terminal units (often in the form of teletype machines like a Friden Flexowriter) saw greatly increased use. Telephone connections provided sufficient speed for early remote terminals and allowed hundreds of kilometers separation between remote-terminals and the computing center. Eventually these standalone computer networks would be generalized into an interconnected *network of networks*—the Internet.

Post-1960: third generation and beyond

history of computing hardware (1960s–present) and history of general purpose CPUs



Intel 8742 eight-bit microcontroller IC

The explosion in the use of computers began with "third-generation" computers, making use of Jack St. Clair Kilby's and Robert Noyce's independent invention of the integrated circuit (or microchip), which later led to the invention of the microprocessor, by Ted Hoff, Federico Faggin, and Stanley Mazor at Intel. The integrated circuit in the image on the right, for example, an Intel 8742, is 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.

During the 1960s there was considerable overlap between second and third generation technologies. IBM implemented its IBM Solid Logic Technology modules in hybrid circuits for the IBM System/360 in 1964. As late as 1975, Sperry Univac continued the manufacture of second-generation machines such as the UNIVAC 494. The Burroughs large systems such as the B5000 were stack machines, which allowed for simpler programming. These pushdown automatons were also implemented in minicomputers and microprocessors later, which influenced programming language design. Minicomputers served as low-cost computer centers for industry, business and universities. It became possible to simulate analog circuits with the *simulation program with integrated circuit emphasis*, or SPICE (1971) on minicomputers, one of the programs for electronic design automation (EDA). The microprocessor led to the development of the microcomputer, small, low-cost computers that could be owned by individuals and small businesses. Microcomputers, the first of which appeared in the 1970s, became ubiquitous in the 1980s and beyond. Steve Wozniak, co-founder of Apple Computer, is credited with developing the first mass-market home computers. However, his first computer, the Apple I, came out some time after the MOS Technology KIM-1 and Altair 8800, and the first Apple computer with graphic and sound capabilities came out well after the Commodore PET. Computing has evolved with microcomputer architectures, with features added from their larger brethren, now dominant in most market segments.

Systems as complicated as computers require very high reliability. ENIAC remained on, in continuous operation from 1947 to 1955, for eight years before being shut down. Although a vacuum tube might fail, it would be replaced without bringing down the system. By the simple strategy of never shutting down ENIAC, the failures were dramatically reduced. Hot-pluggable hard disks, like the hot-pluggable vacuum tubes of yesteryear, continue the tradition of repair during continuous operation. Semiconductor memories routinely have no errors when they operate, although operating systems like Unix have employed memory tests on start-up to detect failing hardware. Today, the requirement of reliable performance is made even more stringent when server farms are the delivery platform. Google has managed this by using fault-tolerant software to recover from hardware failures, and is even working on the concept of replacing entire server farms on-the-fly, during a service event.

In the twenty-first century, multi-core CPUs became commercially available. Content-addressable memory (CAM) has become inexpensive enough to be used in networking, although no computer system has yet implemented hardware CAMs for use in programming languages. Currently, CAMs (or associative arrays) in software are programming-language-specific. Semiconductor memory cell arrays are very regular structures, and manufacturers prove their processes on them; this allows price reductions on memory products. When the CMOS field effect transistor-based logic gates supplanted bipolar transistors, computer power consumption could decrease dramatically (A CMOS Field-effect transistor only draws significant current during the 'transition' between logic states, unlike the substantially higher (and continuous) bias current draw of a BJT). This has allowed computing to become a commodity which is now ubiquitous, embedded in many forms, from greeting cards and telephones to satellites. Computing hardware and its software have even become a metaphor for the operation of the universe.

An indication of the rapidity of development of this field can be inferred by the history of the seminal article. By the time that anyone had time to write anything down, it was obsolete. After 1945, others read John von Neumann's *First Draft of a Report on the EDVAC*, and immediately started implementing their own systems. To this day, the pace of development has continued, worldwide.

In 1941, Konrad Zuse presented the Z3, the world's first functional computer. After the Colossus computer in 1943, the ENIAC (Electronic Numerical Integrator and Computer) of John Presper Eckert and John Mauchly followed in 1946, beginning the computing era. The arithmetic performance of these machines allowed engineers to develop completely new technologies and achieve new objectives. Early examples include the Apollo missions and the NASA moon landing.

Transistors

The invention of the transistor in 1947 by William B. Shockley, John Bardeen and Walter Brattain opened the door for more compact devices and led to the development of the integrated circuit in 1959 by Jack Kilby.

Microprocessors

In 1969, Ted Hoff conceived the commercial microprocessor at Intel and thus ignited the development of the personal computer. Hoff's invention was part of an order by a Japanese company for a desktop programmable electronic calculator, which Hoff wanted to build as cheaply as possible. The first realization of the microprocessor was the Intel 4004, a 4-bit processor, in 1969, but only in 1973 did the Intel 8080, an 8-bit processor, make the building of the first personal computer, the MITS Altair 8800, possible. The first PC was announced to the general public on the cover of the January 1975 issue of Popular Electronics. Mechatronics would have a good fortune in the near future.

Electronics

In the field of electronic engineering, engineers design and test circuits that use the electromagnetic properties of electrical components such as resistors, capacitors, inductors, diodes and transistors to achieve a particular functionality. The tuner circuit, which allows the user of a radio to filter out all but a single station, is just one example of such a circuit.

In designing an integrated circuit, electronics engineers first construct circuit schematics that specify the electrical components and describe the interconnections between them. When completed, VLSI engineers convert the schematics into actual layouts, which map the layers of various conductor and semiconductor materials needed to construct the circuit. The conversion from schematics to layouts can be done by software (see electronic design automation) but very often requires human fine-tuning to decrease space and power consumption. Once the layout is complete, it can be sent to a fabrication plant for manufacturing.

Integrated circuits and other electrical components can then be assembled on printed circuit boards to form more complicated circuits. Today, printed circuit boards are found in most electronic devices including televisions, computers and audio players.

Chapter-2

Electromagnetism & Photoelectric Effect

Electromagnetism

Electromagnetism is the physics of the electromagnetic field, a field that exerts a force on particles with the property of electric charge and is reciprocally affected by the presence and motion of such particles.

A changing magnetic field produces an electric field (this is the phenomenon of electromagnetic induction, the basis of operation for electrical generators, induction motors, and transformers). Similarly, a changing electric field generates a magnetic field.

The magnetic field is produced by the motion of electric charges, i.e., electric current. The magnetic field causes the magnetic force associated with magnets.

The theoretical implications of electromagnetism led to the development of special relativity by Albert Einstein in 1905; and from this it was shown that magnetic fields and electric fields are convertible with relative motion as a four vector and this led to their unification as *electromagnetism*.

History

While preparing for an evening lecture on 21 April 1820, Hans Christian Ørsted developed an experiment that provided surprising evidence. As he was setting up his materials, he noticed a compass needle deflected from magnetic north when the electric current from the battery he was using was switched on and off. This deflection convinced him that magnetic fields radiate from all sides off of a wire carrying an electric current, just as light and heat do, and that it confirmed a direct relationship between electricity and magnetism.

At the time of discovery, Ørsted did not suggest any satisfactory explanation of the phenomenon, nor did he try to represent the phenomenon in a mathematical framework.

However, three months later he began more intensive investigations. Soon thereafter he published his findings, proving that an electric current produces a magnetic field as it flows through a wire. The CGS unit of magnetic induction (oersted) is named in honor of his contributions to the field of electromagnetism.

His findings resulted in intensive research throughout the scientific community in electrodynamics. They influenced French physicist André-Marie Ampère's developments of a single mathematical form to represent the magnetic forces between current-carrying conductors. Ørsted's discovery also represented a major step toward a unified concept of energy.

This unification, which was observed by Michael Faraday, extended by James Clerk Maxwell, and partially reformulated by Oliver Heaviside and Heinrich Hertz, is one of the accomplishments of 19th century Mathematical Physics. It had far-reaching consequences, one of which was the understanding of the nature of light. Light and other electromagnetic waves take the form of quantized, self-propagating oscillatory electromagnetic field disturbances called photons. Different frequencies of oscillation give rise to the different forms of electromagnetic radiation, from radio waves at the lowest frequencies, to visible light at intermediate frequencies, to gamma rays at the highest frequencies.

Ørsted was not the only person to examine the relation between electricity and magnetism. In 1802 Gian Domenico Romagnosi, an Italian legal scholar, deflected a magnetic needle by electrostatic charges. Actually, no galvanic current existed in the setup and hence no electromagnetism was present. An account of the discovery was published in 1802 in an Italian newspaper, but it was largely overlooked by the contemporary scientific community.

The electromagnetic force

Electromagnetic force

The force that the electromagnetic field exerts on electrically charged particles, called the **electromagnetic force**, is one of the fundamental forces. The other fundamental forces are strong nuclear force (which holds atomic nuclei together), the weak nuclear force and the gravitational force. All other forces are ultimately derived from these fundamental forces.

The electromagnetic force is the one responsible for practically all the phenomena encountered in daily life, with the exception of gravity. All the forces involved in interactions between atoms can be traced to the electromagnetic force acting on the electrically charged protons and electrons inside the atoms. This includes the forces we experience in "pushing" or "pulling" ordinary material objects, which come from the intermolecular forces between the individual molecules in our bodies and those in the objects. It also includes all forms of chemical phenomena, which arise from interactions between electron orbitals.

Classical electromagnetism

Classical electromagnetism (or **classical electrodynamics**) is a branch of theoretical physics that studies consequences of the electromagnetic forces between electric charges and currents. It provides an excellent description of electromagnetic phenomena whenever the relevant length scales and field strengths are large enough that quantum mechanical effects are negligible (see quantum electrodynamics). Fundamental physical aspects of classical electrodynamics are presented e.g. by Feynman, Leighton and Sands, Panofsky and Phillips, and Jackson.

The theory of electromagnetism was developed over the course of the 19th century, most prominently by James Clerk Maxwell. For a detailed historical account, consult Pauli, Whittaker, and Pais.

Ribarič and Šušteršič considered a dozen open questions in the current understanding of classical electrodynamics; to this end they studied and cited about 240 references from 1903 to 1989. The outstanding problem with classical electrodynamics, as stated by Jackson, is that we are able to obtain and study relevant solutions of its basic equations only in two limiting cases: »... one in which the sources of charges and currents are specified and the resulting electromagnetic fields are calculated, and the other in which external electromagnetic fields are specified and the motion of charged particles or currents is calculated... Occasionally, ..., the two problems are combined. But the treatment is a stepwise one -- first the motion of the charged particle in the external field is determined, neglecting the emission of radiation; then the radiation is calculated from the trajectory as a given source distribution. It is evident that this manner of handling problems in electrodynamics can be of only approximative validity.« As a consequence, we do not yet have physical understanding of those electromechanical systems where we cannot neglect the mutual interaction between electric charges and currents, and the electromagnetic field emitted by them. Despite of a century long effort, there is as yet no generally accepted classical equation of motion for charged particles, as well as no pertinent experimental data, cf.

Lorentz force

Lorentz force

The electromagnetic field exerts the following force (often called the Lorentz force) on charged particles:

where all boldfaced quantities are vectors: **F** is the force that a charge *q* experiences, **E** is the electric field at the location of the charge, **v** is the velocity of the charge, **B** is the magnetic field at the location of the charge.

The electric field E

Electric field

The electric field **E** is defined such that, on a stationary charge:

where q_0 is what is known as a test charge. The size of the charge doesn't really matter, as long as it is small enough as to not influence the electric field by its mere presence. What is plain from this definition, though, is that the unit of **E** is N/C, or newtons per coulomb. This unit is equal to V/m (volts per meter), see below.

The above definition seems a little bit circular but, in electrostatics, where charges are not moving, Coulomb's law works fine. So what we end up with is:

where n is the number of charges, q_i is the amount of charge associated with the ith charge, $\mathbf{r}_i$ is the position of the ith charge, $\mathbf{r}$ is the position where the electric field is being determined, and ε_0 is a universal constant called the permittivity of free space.

Note: the above is just Coulomb's law, divided by q_1, adding up multiple charges.

Changing the summation to an integral yields the following:

where ρ is the charge density as a function of position, $\mathbf{r}_{\text{unit}}$ is the unit vector pointing from dV to the point in space **E** is being calculated at, and r is the distance from the point **E** is being calculated at to the point charge.

Both of the above equations are cumbersome, especially if one wants to calculate **E** as a function of position. There is, however, a scalar function called the electrical potential that can help. Electric potential, also called voltage (the units for which are the volt), which is defined thus:

where $\varphi_{\mathbf{E}}$ is the electric potential, and s is the path over which the integral is being taken.

Unfortunately, this definition has a caveat. From Maxwell's equations, it is clear that is not always zero, and hence the scalar potential alone is insufficient to define the electric field exactly. As a result, one must resort to adding a correction factor, which is generally done by subtracting the time derivative of the **A** vector potential described below. Whenever the charges are quasistatic, however, this condition will be essentially met, so there will be few problems.

From the definition of charge, one can easily show that the electric potential of a point charge as a function of position is:

where q is the point charge's charge, $\mathbf{r}$ is the position, and $\mathbf{r}_q$ is the position of the point charge. The potential for a general distribution of charge ends up being:

where ρ is the charge density as a function of position, and r is the distance from the volume element dV.

Note well that φ is a scalar, which means that it will add to other potential fields as a scalar. This makes it relatively easy to break complex problems down in to simple parts and add their potentials. Taking the definition of φ backwards, we see that the electric field is just the negative gradient (the del operator) of the potential. Or:

From this formula it is clear that **E** can be expressed in V/m (volts per meter).

Electromagnetic waves

Electromagnetic waves

A changing electromagnetic field propagates away from its origin in the form of a wave. These waves travel in vacuum at the speed of light and exist in a wide spectrum of wavelengths. Examples of the dynamic fields of electromagnetic radiation (in order of increasing frequency): radio waves, microwaves, light (infrared, visible light and ultraviolet), x-rays and gamma rays. In the field of particle physics this electromagnetic radiation is the manifestation of the electromagnetic interaction between charged particles.

General field equations

Jefimenko's equations and Liénard-Wiechert Potentials

As simple and satisfying as Coulomb's equation may be, it is not entirely correct in the context of classical electromagnetism. Problems arise because changes in charge distributions require a non-zero amount of time to be "felt" elsewhere (required by special relativity). Disturbances of the electric field due to a charge propagate at the speed of light.

For the fields of general charge distributions, the retarded potentials can be computed and differentiated accordingly to yield Jefimenko's Equations.

Retarded potentials can also be derived for point charges, and the equations are known as the Liénard-Wiechert potentials. The scalar potential is:

where q is the point charge's charge and $\mathbf{r}$ is the position. $\mathbf{r}_q$ and $\mathbf{v}$ are the position and velocity of the charge, respectively, as a function of retarded time. The vector potential is similar:

These can then be differentiated accordingly to obtain the complete field equations for a moving point particle.

The photoelectric effect

Photoelectric effect

The **photoelectric effect** is a phenomenon in which electrons are emitted from matter (metals and non-metallic solids, liquids, or gases) after the absorption of energy from electromagnetic radiation such as X-rays or visible light. The emitted electrons can be referred to as **photoelectrons** in this context. The effect is also termed the **Hertz Effect**, due to its discovery by Heinrich Rudolf Hertz, although the term has generally fallen out of use. Hertz observed and then showed that electrodes illuminated with ultraviolet light create electric sparks more easily.

The photoelectric effect takes place with photons with energies from about a few electronvolts to, in some cases, over 1 MeV. At the high photon energies comparable to the electron rest energy of 511 keV, Compton scattering, another process, may take place, and above twice this (1.022 MeV) pair production may take place.

Study of the photoelectric effect led to important steps in understanding the quantum nature of light and electrons and influenced the formation of the concept of wave–particle duality.

The term may also, but incorrectly, refer to related phenomena such as the photoconductive effect (also known as photoconductivity or photoresistivitity), the photovoltaic effect, or the photoelectrochemical effect which are, in fact, distinctly different.

Introduction and early historical view

When a surface is exposed to electromagnetic radiation above a certain threshold frequency (typically visible light for alkali metals, near ultraviolet for other metals, and vacuum or extreme ultraviolet for non-metals), the light is absorbed and electrons are emitted. In 1902, Philipp Eduard Anton von Lenard observed that the energy of individual emitted electrons increased with the frequency, or color, of the light. This appeared to be at odds with James Clerk Maxwell's wave theory of light, which was thought to predict that the electron energy would be proportional to the intensity of the radiation. In 1905, Einstein solved this apparent paradox by describing light as composed of discrete quanta, now called photons, rather than continuous waves. Based upon Max Planck's theory of black-body radiation, Einstein theorized that the energy in each quantum of light was equal to the frequency multiplied by a constant, later called Planck's constant. A photon above a threshold frequency has the required energy to eject a single electron, creating the observed effect. This discovery led to the quantum revolution in physics and earned Einstein the Nobel Prize in 1921.

Modern view

It has been shown that it is not necessary for light to be "quantized" to explain the photoelectric effect. The most common way physicists calculate the probability of ejecting an electron uses what is known as Fermi's golden rule. Although based upon quantum mechanics, the method treats a light quantum as an electromagnetic wave that causes an atom and its constituent electrons to transition from one energy state ("eigenstate") to another. It is also important to note that the particle nature of light cannot explain the dependence on polarization with regard to the direction electrons are emitted, a phenomenon that has been considered useful in gathering polarization data from black holes and neutron stars.. Nonetheless, the notion that the photoelectric effect demonstrates the particle nature of light persists in many introductory textbooks.

Traditional explanation

The photons of a light beam have a characteristic energy determined by the frequency of the light. In the photoemission process, if an electron within some material absorbs the energy of one photon and thus has more energy than the work function (the electron binding energy) of the material, it is ejected. If the photon energy is too low, the electron is unable to escape the material. Increasing the intensity of the light beam increases the number of photons in the light beam, and thus increases the number of electrons emitted, but does not increase the energy that each electron possesses. Thus the energy of the emitted electrons does not depend on the intensity of the incoming light, but only on the energy of the individual photons. (This is true as long as the intensity is low enough for non-linear effects caused by multiphoton absorption to be insignificant. This was a given in the age of Einstein, well before lasers had been invented.)

Electrons can absorb energy from photons when irradiated, but they usually follow an "all or nothing" principle. All of the energy from one photon must be absorbed and used to liberate one electron from atomic binding, or the energy is re-emitted. If the photon energy is absorbed, some of the energy liberates the electron from the atom, and the rest contributes to the electron's kinetic energy as a free particle.

Experimental results of the photoelectric emission

1. For a given metal and frequency of incident radiation, the rate at which photoelectrons are ejected is directly proportional to the intensity of the incident light.
2. For a given metal, there exists a certain minimum frequency of incident radiation below which no photoelectrons can be emitted. This frequency is called the threshold frequency.
3. For a given metal of particular work function, increase in frequency of incident beam increases the intensity of the photoelectric current.
4. Above the threshold frequency, the maximum kinetic energy of the emitted photoelectron is independent of the intensity of the incident light but depends on the frequency of the incident light.
5. The time lag between the incidence of radiation and the emission of a photoelectron is very small, less than 10^{-9} second.

6. The direction distribution of emitted electrons peaks in the direction of polarization (the direction of the electric field) of the incident light, if it is linearly polarized.

Equations

In effect quantitatively using Einstein's method, the following equivalent equations are used (valid for visible and ultraviolet radiation):

Energy of photon = Energy needed to remove an electron + Kinetic energy of the emitted electron

Algebraically:

where

- h is Planck's constant,
- f is the frequency of the incident photon,
- $\varphi = hf_0$ is the work function (sometimes denoted W instead), the minimum energy required to remove a delocalised electron from the surface of any given metal,
- is the maximum kinetic energy of ejected electrons,
- f_0 is the threshold frequency for the photoelectric effect to occur,
- m is the rest mass of the ejected electron, and
- v_m is the speed of the ejected electron.

Since an emitted electron cannot have negative kinetic energy, the equation implies that if the photon's energy (hf) is less than the work function (φ), no electron will be emitted.

According to Einstein's special theory of relativity the relation between energy (E) and momentum (p) of a particle is , where m is the rest mass of the particle and c is the velocity of light in a vacuum.

Three-step model

In the X-ray regime, the photoelectric effect in crystalline material is often decomposed into three steps:

1. Inner photoelectric effect (see photodiode below). The hole left behind can give rise to auger effect, which is visible even when the electron does not leave the material. In molecular solids phonons are excited in this step and may be visible as lines in the final electron energy. The inner photoeffect has to be dipole allowed. The transition rules for atoms translate via the tight-binding model onto

the crystal. They are similar in geometry to plasma oscillations in that they have to be transversal.
2. Ballistic transport of half of the electrons to the surface. Some electrons are scattered.
3. Electrons escape from the material at the surface.

In the three-step model, an electron can take multiple paths through these three steps. All paths can interfere in the sense of the path integral formulation. For surface states and molecules the three-step model does still make some sense as even most atoms have multiple electrons which can scatter the one electron leaving.

History

Early observations

In 1839, Alexandre Edmond Becquerel observed the photoelectric effect via an electrode in a conductive solution exposed to light. In 1873, Willoughby Smith found that selenium is photoconductive.

Hertz's spark gaps

In 1887, Heinrich Hertz observed the photoelectric effect and the production and reception of electromagnetic (EM) waves. He published these observations in the journal Annalen der Physik. His receiver consisted of a coil with a spark gap, where a spark would be seen upon detection of EM waves. He placed the apparatus in a darkened box to see the spark better. However, he noticed that the maximum spark length was reduced when in the box. A glass panel placed between the source of EM waves and the receiver absorbed ultraviolet radiation that assisted the electrons in jumping across the gap. When removed, the spark length would increase. He observed no decrease in spark length when he substituted quartz for glass, as quartz does not absorb UV radiation. Hertz concluded his months of investigation and reported the results obtained. He did not further pursue investigation of this effect, nor did he make any attempt at explaining how this phenomenon was brought about.

Stoletov: the first law of photoeffect

In the period from February 1888 and until 1891, a detailed analysis of photoeffect was performed by Aleksandr Stoletov with results published in 6 works; four of them in *Comptes Rendus*, one review in *Physikalische Revue* (translated from Russian), and the last work in *Journal de Physique*. First, in these works Stoletov invented a new experimental setup which was more suitable for a quantitative analysis of photoeffect. Using this setup, he discovered the direct proportionality between the intensity of light and the induced photo electric current (the first law of photoeffect or Stoletov's law). One of his other findings resulted from measurements of the dependence of the intensity of the electric photo current on the gas pressure, where he found the existence of an optimal gas pressure P_m corresponding to a maximum photocurrent; this property was used for a creation of solar cells.

JJ Thomson: electrons

In 1899, J. J. Thomson investigated ultraviolet light in Crookes tubes. Influenced by the work of James Clerk Maxwell, Thomson deduced that cathode rays consisted of negatively charged particles, later called electrons, which he called "corpuscles". In the research, Thomson enclosed a metal plate (a cathode) in a vacuum tube, and exposed it to high frequency radiation. It was thought that the oscillating electromagnetic fields caused the atoms' field to resonate and, after reaching a certain amplitude, caused a subatomic "corpuscle" to be emitted, and current to be detected. The amount of this current varied with the intensity and colour of the radiation. Larger radiation intensity or frequency would produce more current.

Radiant energy

Rays falling on insulated conductor connected to a capacitor: the capacitor charges electrically.

Nikola Tesla described the photoelectric effect in 1901. He described such radiation as vibrations of aether of small wavelengths which ionized the atmosphere. On November 5, 1901, he received the patent US685957, *Apparatus for the Utilization of Radiant Energy*, that describes radiation charging and discharging conductors. This was done by using a metal plate or piece of mica exposed to "*radiant energy*". Tesla used this effect to charge a capacitor with energy by means of a conductive plate, making a solar cell precursor. The radiant energy threw off with great velocity minute particles (i.e., electrons) which were strongly electrified. The patent specified that the radiation (or radiant energy) included many different forms. These devices have been referred to as "*Photoelectric alternating current stepping motors*".

In practice, a polished insulated metal plate or other conducting-body in radiant energy (e.g. sunlight) will gain a positive charge as electrons are emitted by the plate. As the plate charges positively, electrons form an electrostatic force on the plate (because of surface emissions of the photoelectrons), and "*drain*" any negatively charged capacitors. As the rays or radiation fall on the insulated conductor (which is connected to a capacitor), the condenser will indefinitely charge electrically.

Von Lenard's observations

In 1902, Philipp Lenard observed the variation in electron energy with light frequency. He used a powerful electric arc lamp which enabled him to investigate large changes in intensity, and had sufficient power to enable him to investigate the variation of potential with light frequency. His experiment directly measured potentials, not electron kinetic energy: he found the electron energy by relating it to the maximum stopping potential (voltage) in a phototube. He found that the calculated maximum electron kinetic energy is determined by the frequency of the light. For example, an increase in frequency results in an increase in the maximum kinetic energy calculated for an electron upon liberation - ultraviolet radiation would require a higher applied stopping potential to stop current in a phototube than blue light. However Lenard's results were qualitative rather than quantitative because of the difficulty in performing the experiments: the experiments needed to be done on freshly cut metal so that the pure metal was observed, but it oxidised in a matter of minutes even in the partial vacuums he used. The current emitted by the surface was determined by the light's intensity, or brightness: doubling the intensity of the light doubled the number of electrons emitted from the surface. Lenard did not know of photons.

Einstein: light quanta

Albert Einstein's mathematical description in 1905 of how the photoelectric effect was caused by absorption of quanta of light (now called photons), was in the paper named "*On a Heuristic Viewpoint Concerning the Production and Transformation of Light*". This paper proposed the simple description of "light quanta", or photons, and showed how they explained such phenomena as the photoelectric effect. His simple explanation in terms of absorption of discrete quanta of light explained the features of the phenomenon and the characteristic frequency. Einstein's explanation of the photoelectric effect won him the Nobel Prize in Physics in 1921.

The idea of light quanta began with Max Planck's published law of black-body radiation ("*On the Law of Distribution of Energy in the Normal Spectrum*". Annalen der Physik 4 (1901)) by assuming that Hertzian oscillators could only exist at energies E proportional to the frequency f of the oscillator by $E = hf$, where h is Planck's constant. By assuming that light actually consisted of discrete energy packets, Einstein wrote an equation for the photoelectric effect that fitted experiments. It explained why the energy of photoelectrons were dependent only on the *frequency* of the incident light and not on its *intensity*: a low-intensity, high-frequency source could supply a few high energy photons, whereas a high-intensity, low-frequency source would supply no photons of sufficient individual energy to dislodge any electrons. This was an enormous theoretical leap, but the concept was strongly resisted at first because it contradicted the wave theory of light that followed naturally from James Clerk Maxwell's equations for electromagnetic behaviour, and more generally, the assumption of infinite divisibility of energy in physical systems. Even after experiments showed that Einstein's equations for the photoelectric effect were accurate, resistance to the idea of photons continued, since it appeared to contradict Maxwell's equations, which were well-understood and verified.

Einstein's work predicted that the energy of individual ejected electrons increases linearly with the frequency of the light. Perhaps surprisingly, the precise relationship had not at that time been tested. By 1905 it was known that the energy of photoelectrons increases with increasing *frequency* of incident light and is independent of the *intensity* of the light. However, the manner of the increase was not experimentally determined until 1915 when Robert Andrews Millikan showed that Einstein's prediction was correct.

Effect on wave–particle question

The photoelectric effect helped propel the then-emerging concept of the dualistic nature of light, that light simultaneously possesses the characteristics of both waves and particles, each being manifested according to the circumstances. The effect was, at the time, thought to be impossible to understand in terms of the classical wave description of light, as the energy of the emitted electrons did not depend on the intensity of the incident radiation. Classical theory was believed to predict that the electrons could 'gather up' energy over a period of time, and then be emitted. For such a classical theory to work a pre-loaded state would need to persist in matter. The idea of the pre-loaded state was discussed in Millikan's book *Electrons (+ & –)* and in Compton and Allison's book *X-Rays in Theory and Experiment*.

Uses and effects

Photodiodes and phototransistors

Solar cells (used in solar power) and light-sensitive diodes use a variant of the photoelectric effect, but not ejecting electrons out of the material. In semiconductors, light of even relatively low energy, such as visible photons, can kick electrons out of the valence band and into the higher-energy conduction band, where they can be harnessed, creating electric current at a voltage related to the bandgap energy.

Photomultipliers

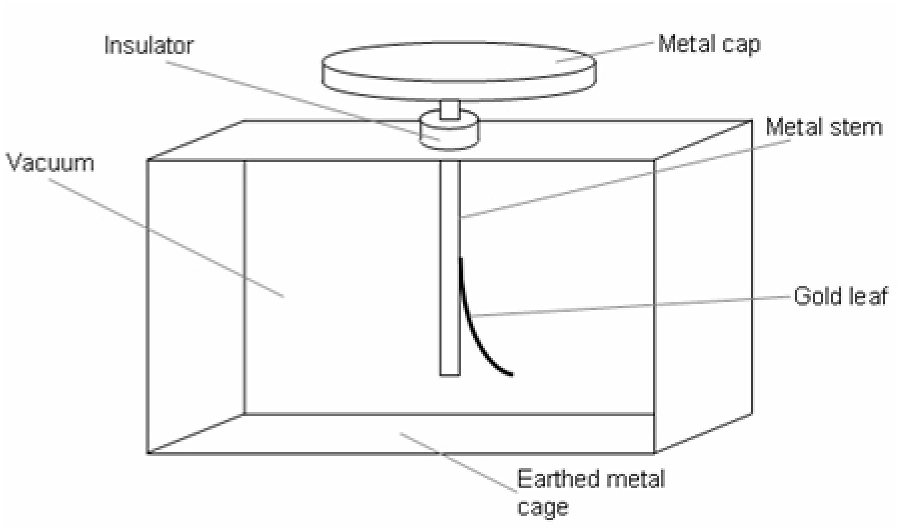
These are extremely light-sensitive vacuum tubes with a photocathode coated onto part (an end or side) of the inside of the envelope. The photocathode contains combinations of materials such as caesium, rubidium and antimony specially selected to provide a low work function, so when illuminated even by very low levels of light, the photocathode readily releases electrons. By means of a series of electrodes (dynodes) at ever-higher potentials, these electrons are accelerated and substantially increased in number through secondary emission to provide a readily-detectable output current. Photomultipliers are still commonly used wherever low levels of light must be detected.

Image sensors

Video camera tubes in the early days of television used the photoelectric effect; newer variants used photoconductive rather than photoemissive materials.

Silicon image sensors, such as charge-coupled devices, widely used for photographic imaging, are based on a variant of the photoelectric effect, in which photons knock electrons out of the valence band of energy states in a semiconductor, but not out of the solid itself.

The gold-leaf electroscope



The gold leaf electroscope.

Gold-leaf electroscopes are designed to detect static electricity. Charge placed on the metal cap spreads to the stem and the gold leaf of the electroscope. Because they then have the same charge, the stem and leaf repel each other. This will cause the leaf to bend away from the stem. The electroscope is an important tool in illustrating the photoelectric effect. Let us say that the scope is negatively charged throughout. There is an excess of electrons and the leaf is separated from the stem. But if we then shine high-frequency light onto the cap, the scope discharges and the leaf will fall limp. This is because the frequency of the light shining on the cap is above the cap's threshold frequency. The photons in the light have enough energy to liberate electrons from the cap, reducing its negative charge. This will discharge a negatively charged electroscope and further charge a positive electroscope.

However, if the EM radiation hitting the metal cap does not have a high enough frequency, (its frequency is below the threshold value for the cap) then the leaf will never discharge, no matter how long one shines the low-frequency light at the cap.

Photoelectron spectroscopy

Since the energy of the photoelectrons emitted is exactly the energy of the incident photon minus the material's work function or binding energy, the work function of a sample can be determined by bombarding it with a monochromatic X-ray source or UV source (typically a helium discharge lamp), and measuring the kinetic energy distribution of the electrons emitted.

Photoelectron spectroscopy is done in a high-vacuum environment, since the electrons would be scattered by significant numbers of gas atoms present (e.g. even in low-pressure air).

The concentric hemispherical analyser (CHA) is a typical electron energy analyzer, and uses an electric field to divert electrons different amounts depending on their kinetic energies. For every element and core (atomic orbital) there will be a different binding energy. The many electrons created from each of these combinations will show up as spikes in the analyzer output, and these can be used to determine the elemental composition of the sample.

Spacecraft

The photoelectric effect will cause spacecraft exposed to sunlight to develop a positive charge. This can get up to the tens of volts. This can be a major problem, as other parts of the spacecraft in shadow develop a negative charge (up to several kilovolts) from nearby plasma, and the imbalance can discharge through delicate electrical components. The static charge created by the photoelectric effect is self-limiting, though, because a more highly-charged object gives up its electrons less easily.

Moon dust

Light from the sun hitting lunar dust causes it to become charged through the photoelectric effect. The charged dust then repels itself and lifts off the surface of the Moon by electrostatic levitation. This manifests itself almost like an "atmosphere of dust", visible as a thin haze and blurring of distant features, and visible as a dim glow after the sun has set. This was first photographed by the Surveyor program probes in the 1960s. It is thought that the smallest particles are repelled up to kilometers high, and that the particles move in "fountains" as they charge and discharge.

Night vision devices

Photons hitting a gallium arsenide plate in night vision devices cause the ejection of photoelectrons due to the photoelectric effect. These are then amplified into a cascade of electrons that light up a phosphor screen.

Cross section

The photoelectric effect is simply an interaction mechanism conducted between photons and atoms. However, this mechanism does not have exclusivity in interactions of this nature and is one of 12 theoretically possible interactions . As noted in the prologue; Compton scattering and pair production are an example of two other competing mechanisms. Indeed, even if the photoelectric effect is the favoured reaction for a particular single-photon bound-electron interaction, the result is also subject to statistical processes and is not guaranteed, albeit the photon has certainly disappeared and a bound electron has been excited (usually K or L shell electrons at nuclear (gamma ray) energies). The probability of the photoelectric effect occurring is measured by the cross section of interaction, σ. This has been found to be a function of the atomic number of the target atom and photon energy. A crude approximation, for photon energies above the highest atomic binding energy, is given by :

Where n is a number which varies between 4 and 5. (At lower photon energies a characteristic structure with edges appears, K edge, L edges, M edges, etc.) The obvious interpretation follows that the photoelectric effect rapidly decreases in significance, in the gamma ray region of the spectrum, with increasing photon energy, and that photoelectric effect is directly proportional to atomic number. The corollary is that high-Z materials make good gamma-ray shields, which is the principal reason that lead (Z = 82) is a preferred and ubiquitous gamma radiation shield [Kno99].

Units

Electromagnetic units are part of a system of electrical units based primarily upon the magnetic properties of electric currents, the fundamental SI unit being the ampere. The units are:

- ampere (current)
- coulomb (charge)
- farad (capacitance)
- henry (inductance)
- ohm (resistance)
- volt (electric potential)
- watt (power)
- tesla (magnetic field)

In the electromagnetic cgs system, electrical current is a fundamental quantity defined via Ampère's law and takes the permeability as a dimensionless quantity (relative permeability) whose value in a vacuum is unity. As a consequence, the square of the speed of light appears explicitly in some of the equations interrelating quantities in this system.

SI electromagnetism units

v • d • e

Symbol	Name of Quantity	Derived Units	Unit	Base Units
I	Electric current	ampere (SI base unit)	A	A (= W/V = C/s)
Q	Electric charge	coulomb	C	A·s

U, ΔV, $\Delta\varphi$; E	Potential difference; Electromotive force	volt	V	J/C = kg·m^2·s^{-3}·A^{-1}
R; Z; X	Electric resistance; Impedance; Reactance	ohm	Ω	V/A = kg·m^2·s^{-3}·A^{-2}
ρ	Resistivity	ohm metre	Ω·m	kg·m^3·s^{-3}·A^{-2}
P	Electric power	watt	W	V·A = kg·m^2·s^{-3}
C	Capacitance	farad	F	C/V = kg^{-1}·m^{-2}·A^2·s^4
$\boldsymbol{E}$	Electric field strength	volt per metre	V/m	N/C = kg·m·A^{-1}·s^{-3}
$\boldsymbol{D}$	Electric displacement field	Coulomb per square metre	C/m^2	A·s·m^{-2}
ε	Permittivity	farad per metre	F/m	kg^{-1}·m^{-3}·A^2·s^4
χ_e	Electric susceptibility	(dimensionless)	-	-
G; Y; B	Conductance; Admittance; Susceptance	siemens	S	Ω^{-1} = kg^{-1}·m^{-2}·s^3·A^2
κ, γ, σ	Conductivity	siemens per metre	S/m	kg^{-1}·m^{-3}·s^3·A^2
$\boldsymbol{B}$	Magnetic flux density, Magnetic induction	tesla	T	Wb/m^2 = kg·s^{-2}·A^{-1} = N·A^{-1}·m^{-1}
Φ	Magnetic flux	weber	Wb	V·s = kg·m^2·s^{-2}·A^{-1}

H	Magnetic field strength	ampere per metre	A/m	$A \cdot m^{-1}$
L, *M*	Inductance	henry	H	$Wb/A = V \cdot s/A = kg \cdot m^2 \cdot s^{-2} \cdot A^{-2}$
μ	Permeability	henry per metre	H/m	$kg \cdot m \cdot s^{-2} \cdot A^{-2}$
χ	Magnetic susceptibility	(dimensionless)	-	-

Electromagnetic phenomena

With the exception of gravitation, electromagnetic phenomena as described by quantum electrodynamics account for almost all physical phenomena observable to the unaided human senses, including light and other electromagnetic radiation, all of chemistry, most of mechanics (excepting gravitation), and of course magnetism and electricity.

Network analysis

Network graphs: matrices associated with graphs; incidence, fundamental cut set and fundamental circuit matrices. Solution methods: nodal and mesh analysis. Network theorems: superposition, Thevenin and Norton's maximum power transfer, Wye-Delta transformation. Steady state sinusoidal analysis using phasors. Linear constant coefficient differential equations; time domain analysis of simple RLC circuits, Solution of network equations using Laplace transform: frequency domain analysis of RLC circuits. 2-port network parameters: driving point and transfer functions. State equatioons for networks.

Electronic devices and circuits

Electronic devices: Energy bands in silicon, intrinsic and extrinsic silicon. Carrier transport in silicon: diffusion current, drift current, mobility, resistivity. Generation and recombination of carriers. p-n junction diode, Zener diode, tunnel diode, BJT, JFET, MOS capacitor, MOSFET, LED, p-i-n and avalanche photo diode, LASERs. Device technology: integrated circuit fabrication process, oxidation, diffusion, ion implantation, photolithography, n-tub, p-tub and twin-tub CMOS process.

Analog circuits: Equivalent circuits (large and small-signal) of diodes, BJTs, JFETs, and MOSFETs. Simple diode circuits, clipping, clamping, rectifier. Biasing and bias stability of transistor and FET amplifiers. Amplifiers: single-and multi-stage, differential, operational, feedback and power. Analysis of amplifiers; frequency response of amplifiers. Simple op-amp circuits. Filters. Sinusoidal oscillators; criterion for oscillation; single-transistor and op-amp configurations. Function generators and wave-shaping circuits, Power supplies.

Digital circuits: of Boolean functions; logic gates digital IC families (DTL, TTL, ECL, MOS, CMOS). Combinational circuits: arithmetic circuits, code converters, multiplexers and decoders. Sequential circuits: latches and flip-flops, counters and shift-registers. Sample and hold circuits, ADCs, DACs. Semiconductor memories. Microprocessor 8086: architecture, programming, memory and I/O interfacing.

Signals and systems

Definitions and properties of Laplace transform, continuous-time and discrete-time Fourier series, continuous-time and discrete-time Fourier Transform, z-transform. Sampling theorems. Linear Time-Invariant (LTI) Systems: definitions and properties; causality, stability, impulse response, convolution, poles and zeros frequency response, group delay, phase delay. Signal transmission through LTI systems. Random signals and noise: probability, random variables, probability density function, autocorrelation, power spectral density, function analogy between vectors & functions.

Control systems

Basic control system components; block diagrammatic description, reduction of block diagrams - Mason's rule. Open loop and closed loop (negative unity feedback) systems and stability analysis of these systems. Signal flow graphs and their use in determining transfer functions of systems; transient and steady state analysis of LTI control systems and frequency response. Analysis of steady-state disturbance rejection and noise sensitivity.

Tools and techniques for LTI control system analysis and design: root loci, Routh-Hurwitz stability criterion, Bode and Nyquist plots. Control system compensators: elements of lead and lag compensation, elements of Proportional-Integral-Derivative controller (PID). Discretization of continuous time systems using Zero-order hold (ZOH) and ADC's for digital controller implementation. Limitations of digital controllers: aliasing. State variable representation and solution of state equation of LTI control systems. Linearization of Nonlinear dynamical systems with state-space realizations in both frequency and time domains. Fundamental concepts of controllability and observability for MIMO LTI systems. State space realizations: observable and controllable canonical form. Ackerman's function for state-feedback pole placement. Design of full order and reduced order estimators.

Communications

Analog communication systems: amplitude and angle modulation and demodulation systems, spectral analysis of these operations, superheterodyne noise conditions.

Digital communication systems: pulse code modulation (PCM), [[Differential Pulse Code Modulation (DPCM), Delta modulation (DM), digital modulation schemes-amplitude, phase and frequency shift keying schemes (ASK, PSK, FSK), matched filter receivers, bandwidth consideration and probability of error calculations for these schemes, GSM, TDMA.

Education and training

Electronics engineers typically possess an academic degree with a major in electronic engineering. The length of study for such a degree is usually three or four years and the completed degree may be designated as a Bachelor of Engineering, Bachelor of Science, Bachelor of Applied Science, or Bachelor of Technology depending upon the university. Many UK universities also offer Master of Engineering (MEng) degrees at undergraduate level.

The degree generally includes units covering physics,chemistry,mathematics, project management and specific topics in electrical engineering. Initially such topics cover most, if not all, of the subfields of electronic engineering. Students then choose to specialize in one or more subfields towards the end of the degree.

Some electronics engineers also choose to pursue a postgraduate degree such as a Master of Science (MSc), Doctor of Philosophy in Engineering (PhD), or an Engineering Doctorate (EngD). The Master degree is being introduced in some European and American Universities as a first degree and the differentiation of an engineer with graduate and postgraduate studies is often difficult. In these cases, experience is taken into account. The Master's degree may consist of either research, coursework or a mixture of the two. The Doctor of Philosophy consists of a significant research component and is often viewed as the entry point to academia.

In most countries, a Bachelor's degree in engineering represents the first step towards certification and the degree program itself is certified by a professional body. After completing a certified degree program the engineer must satisfy a range of requirements (including work experience requirements) before being certified. Once certified the engineer is designated the title of Professional Engineer (in the United States, Canada and South Africa), Chartered Engineer or Incorporated Engineer (in the United Kingdom, Ireland, India and Zimbabwe), Chartered Professional Engineer (in Australia) or European Engineer (in much of the European Union).

Fundamental to the discipline are the sciences of physics and mathematics as these help to obtain both a qualitative and quantitative description of how such systems will work. Today most engineering work involves the use of computers and it is commonplace to use computer-aided design programs when designing electronic systems. Although most electronic engineers will understand basic circuit theory, the theories employed by engineers generally depend upon the work they do. For example, quantum mechanics and solid state physics might be relevant to an engineer working on VLSI but are largely irrelevant to engineers working with macroscopic electrical systems.

Licensure, certification, and regulation

Some locations require a license for one to legally be called an electronics engineer, or an engineer in general. For example, in the United States and Canada "only a licensed engineer may seal engineering work for public and private clients". This requirement is enforced by state and provincial legislation such as Quebec's Engineers Act. In other countries, such as Australia, no such legislation exists. Practically all certifying bodies maintain a code of ethics that they expect all members to abide by or risk expulsion. In this way these organizations play an important role in maintaining ethical standards for the profession. Even in jurisdictions where licenses are not required, engineers are subject to the law. For example, much engineering work is done by contract and is therefore covered by contract law. In cases where an engineer's work fails he or she may be subject to the tort of negligence and, in extreme cases, the charge of criminal negligence. An engineer's work must also comply with numerous other rules and regulations such as building codes and legislation pertaining to environmental law.

In locations where licenses are not required, professional certification may be advantageous.

Professional bodies

Professional bodies of note for electrical engineers include the Institute of Electrical and Electronics Engineers (IEEE) and the Institution of Electrical Engineers (IEE),now the Institution of Engineering and Technology(IET). The IEEE claims to produce 30 percent of the world's literature in electrical/electronic engineering, has over 370,000 members, and holds more than 450 IEEE sponsored or cosponsored conferences worldwide each year.

Modern electronic engineering

Electronic engineering in Europe is a very broad field that encompasses many subfields including those that deal with, electronic devices and circuit design, control systems, electronics and telecommunications, computer systems, embedded software etc. Many European universities now have departments of electronics that are completely separate from their respective departments of electrical engineering.

Chapter-3

Signal processing, Telecommunications Engineering & Control engineering

Electronics engineering has many subfields. This section describes some of the most popular subfields in electronic engineering. Although there are engineers who focus exclusively on one subfield, there are also many who focus on a combination of subfields.

Signal processing

It deals with the analysis and manipulation of signals. Signals can be either analog, in which case the signal varies continuously according to the information, or digital, in which case the signal varies according to a series of discrete values representing the information

Signal processing is an area of applied mathematics that deals with operations on or analysis of signals, in either discrete or continuous time to perform useful operations on those signals. Depending upon the application, a useful operation could be control, data compression, data transmission, denoising, prediction, filtering, smoothing, deblurring, tomographic reconstruction, identification, classification, or a variety of other operations.

Signals of interest can include sound, images, time-varying measurement values and sensor data, for example biological data such as electrocardiograms, control system signals, telecommunication transmission signals such as radio signals, and many others.

History

According to Alan V. Oppenheim and Ronald W. Schafer, the principles of signal processing can be found in the classical numerical analysis techniques of the 17th century. They further state that the "digitalization" or digital refinement of these techniques can be found in the digital control systems of the 1940s and 1950s.

Mathematical topics embraced by signal processing

- Linear signals and systems, and transform theory
- Probability and stochastic processes
- Programming

- Calculus and analysis
- Vector spaces and linear algebra
- Numerical methods
- Functional analysis
- Optimization
- Statistical decision theory
- Iterative methods

Categories of signal processing

- Analog signal processing — for signals that have not been digitized, as in classical radio, telephone, radar, and television systems. This involves linear electronic circuits such as passive filters, active filters, additive mixers, integrators and delay lines. It also involves non-linear circuits such as compandors, multiplicators (frequency mixers and voltage-controlled amplifiers), voltage-controlled filters, voltage-controlled oscillators and phase-locked loops.
- Discrete time signal processing — for sampled signals that are considered as defined only at discrete points in time, and as such are quantized in time, but not in magnitude. *Analog discrete-time signal processing* is a technology based on electronic devices such as sample and hold circuits, analog time-division multiplexers, analog delay lines and analog feedback shift registers. This technology was a predecessor of digital signal processing, see below, and is still used in advanced processing of gigahertz signals. The concept of discrete-time signal processing also refers to a theoretical discipline that establishes a mathematical basis for digital signal processing, without taking quantization error into consideration.
- Digital signal processing — for signals that have been digitized. Processing is done by general-purpose computers or by digital circuits such as ASICs, field-programmable gate arrays or specialized digital signal processors (DSP chips). Typical arithmetical operations include fixed-point and floating-point, real-valued and complex-valued, multiplication and addition. Other typical operations supported by the hardware are circular buffers and look-up tables. Examples of algorithms are the Fast Fourier transform (FFT), finite impulse response (FIR) filter, Infinite impulse response (IIR) filter, Wiener filter and Kalman filter.

Fields of signal processing

- Statistical signal processing — analyzing and extracting information from signals based on their statistical properties
- Audio signal processing — for electrical signals representing sound, such as speech or music
- Speech signal processing — for processing and interpreting spoken words
- Image processing — in digital cameras, computers, and various imaging systems
- Video processing — for interpreting moving pictures
- Array processing — for processing signals from arrays of sensors

For analog signals, signal processing may involve the amplification and filtering of audio signals for audio equipment or the modulation and demodulation of signals for telecommunications. For digital signals, signal processing may involve the compression, error checking and error detection of digital signals.

Telecommunications engineering

It deals with the transmission of information across a channel such as a co-axial cable, optical fiber or free space.

Transmissions across free space require information to be encoded in a carrier wave in order to shift the information to a carrier frequency suitable for transmission, this is known as modulation. Popular analog modulation techniques include amplitude modulation and frequency modulation. The choice of modulation affects the cost and performance of a system and these two factors must be balanced carefully by the engineer.

Once the transmission characteristics of a system are determined, telecommunication engineers design the transmitters and receivers needed for such systems. These two are sometimes combined to form a two-way communication device known as a transceiver. A key consideration in the design of transmitters is their power consumption as this is closely related to their signal strength. If the signal strength of a transmitter is insufficient the signal's information will be corrupted by noise

Telecommunications engineering or **telecom engineering** is a major field within electronic engineering. Telecom engineers come in a variety of different types from basic circuit designers to strategic mass developments. A telecom engineer is responsible for designing and overseeing the installation of telecommunications equipment and facilities, such as complex electronic switching systems to copper telephone facilities and fiber optics. Telecom engineering also overlaps heavily with broadcast engineering.

Telecommunications is a diverse field of engineering including electronics, civil, structural, and electrical engineering as well as being a political and social ambassador, a little bit of accounting and a lot of project management. Ultimately, telecom engineers are responsible for providing the method that customers can get telephone and high speed data services.

Telecom engineers use a variety of different equipment and transport media available from a multitude of manufacturers to design the telecom network infrastructure. The most common media, often referred to as plant in the telecom industry, used by telecommunications companies today are copper, coaxial cable, fiber, and radio.

Telecom engineers are often expected, as most engineers are, to provide the best solution possible for the lowest cost to the company. This often leads to creative solutions to problems that often would have been designed differently without the budget constraints dictated by modern society. In the earlier days of the telecom industry massive amounts of cable were placed that were never used or have been replaced by modern technology such as fiber optic cable and digital multiplexing techniques.

Telecom engineers are also responsible for keeping the records of the companies' equipment and facilities and assigning appropriate accounting codes for purposes of taxes and maintenance. As telecom engineers responsible for budgeting and overseeing projects and keeping records of equipment, facilities and plant the telecom engineer is not only an engineer but an accounting assistant or bookkeeper (if not an accountant) and a project manager as well.

Telecom equipment engineer

A telecom equipment engineer is an electronics engineer that designs equipment such as routers, switches, multiplexers, and other specialized computer/electronics equipment designed to be used in the telecommunication network infrastructure.

Central-office engineer

A Central-office engineer is responsible for designing and overseeing the implementation of telecommunications equipment in a central office (CO for short), also referred to as a wire center or telephone exchange. A CO engineer is responsible for integrating new technology into the existing network, assigning the equipments location in the wire center and providing power, clocking (for digital equipment) and alarm monitoring facilities for the new equipment. The CO engineer is also responsible for providing more power, clocking, and alarm monitoring facilities if there isn't currently enough available to support the new equipment being installed. Finally, the CO Engineer is responsible for designing how the massive amounts of cable will be distributed to various equipment and wiring frames throughout the wire center and overseeing the installation and turn up of all new equipment.

As structural engineers, CO engineers are responsible for the structural design and placement of racking and bays for the equipment to be installed in as well as for the plant to be placed on.

As electrical engineers, CO engineers are responsible for the resistance, capacitance, and inductance (RCL) design of all new plant to ensure telephone service is clear and crisp and data service is clean as well as reliable. Attenuation and loop loss calculations are required to determine cable length and size required to provide the service called for. In addition power requirements have to be calculated and provided for to power any electronic equipment being placed in the wire center.

Outside-plant engineer

Outside plant (OSP) engineers are also often called Field Engineers as they often spend a great deal of time in the field taking notes about the civil environment, aerial, above ground, and below ground. OSP Engineers are responsible for taking plant (copper, fiber, etc.) from a wire center to a distribution point or destination point directly. If a distribution point design is used then a cross connect box is placed in a strategic location to feed a determined distribution area.

The cross-connect box, also known as a service area interface is then installed to allow connections to be made more easily from the wire center to the destination point and ties up fewer facilities by not having dedication facilities from the wire center to every destination point. The plant is then taken directly to its destination point or to another small closure called a pedestal where access can also be gained to the plant if necessary. These access points are preferred as they allow faster repair times for customers and save telephone operating companies large amounts of money.

The plant facilities can be delivered via underground facilities, either direct buried or through conduit or in some cases laid under water, via aerial facilities such as telephone or power poles, or via microwave radio signals for long distances where either of the other two methods is too costly.

As structural engineers, OSP egineers are responsible for the structural design and placement of cellular towers and telephone poles as well as calculating pole capabilities of existing telephone or power poles new plant is being added onto. Structural calculations are required when boring under heavy traffic areas such as highways or when attaching to other structures such as bridges. Shoring also has to be taken into consideration for larger trenches or pits. Conduit structures often include encasements of slurry that needs to be designed to support the structure and withstand the environment around it (soil type, high traffic areas, etc.).

As electrical engineers, OSP engineers are responsible for the resistance, capacitance, and inductance (RCL) design of all new plant to ensure telephone service is clear and crisp and data service is clean as well as reliable. Attenuation and loop loss calculations are required to determine cable length and size required to provide the service called for. In addition power requirements have to be calculated and provided for to power any electronic equipment being placed in the field. Ground potential has to be taken into consideration when placing equipment, facilities, and plant in the field to account for lightning strikes, high voltage intercept from improperly grounded or broken power company facilities, and from various sources of electromagnetic interference.

As civil engineers, OSP egineers are responsible for drawing up plans, either by hand or using Computer Aided Drafting (CAD) software, for how telecom plant facilities will be placed. Often when working with municipalities trenching or boring permits are required and drawings must be made for these. Often these drawings include about 70% or so of the detailed information required to pave a road or add a turn lane to an existing street. Structural calculations are required when boring under heavy traffic areas such as highways or when attaching to other structures such as bridges. As Civil Engineers

Telecom Engineers provide the modern communications backbone for all technological communications distributed throughout civilizations today.

Unique to Telecom Engineering is the use of air core cable which requires an extensive network of air handling equipment such as compressors, manifolds, regulators and hundreds of miles of air pipe per system that connects to pressurized splice cases all designed to pressurize this special form of copper cable to keep moisture out and provide a clean signal to the customer.

As Political and Social Ambassador, the OSP Engineer is the telephone operating companies' face and voice to the local authorities and other utilities. OSP Engineers often meet with municipalities, construction companies and other utility companies to address their concerns and educate them about how the telephone utility works and operates. Additionally, the OSP Engineer has to secure real estate to place outside facilities on such as an easement to place a cross connect box on.

Control engineering

Control systems play a critical role in space flight

Control engineering is the engineering discipline that applies control theory to design systems with predictable behaviors. The engineering activities focus on the mathematical modeling of systems of a diverse nature.

Overview

Modern day control engineering (also called control systems engineering) is a relatively new field of study that gained a significant attention during twentieth century with the advancement in technology. It can be broadly defined as practical application of control theory. Control engineering has an essential role in a wide range of control systems from a simple household washing machine to a complex high performance F-16 fighter aircraft. It allows one to understand a physical system in terms of its inputs, outputs and various components with different behaviors using mathematical modeling, control it in a desired manner with the controllers designed using control systems design tools, and implement the controller on the physical system employing available technology. A system can be mechanical, electrical, fluid, chemical, financial and even biological, and the mathematical modeling, analysis and controller design shall be done using control theory in one or many of the time, frequency and complex-s domains depending on the nature of the control system design problem.

Before it emerged as a unique discipline, control engineering was practiced as a part of mechanical engineering and control theory was studied as a part of electrical engineering, since electrical circuits can often be easily described using control theory techniques. In the very first control relationships, a current output was represented with a voltage control input. However, not having proper technology to implement electrical control systems, designers left with the option of less efficient and slow responding mechanical systems. A very effective mechanical controller that is still widely used in some hydro plants is the governor. Later on, previous to modern power electronics, process control systems for industrial applications were devised by mechanical engineers using pneumatic and hydraulic control devices, many of which are still in use today.

There are two major divisions in control theory, namely, classical and modern, which have direct implications over the control engineering applications. The scope of classical control theory is limited to single-input and single-output (SISO) system design. The system analysis is carried out in time domain using differential equations, in complex-s domain with Laplace transform or in frequency domain by transforming from complex-s domain. All the systems are assumed to be second order, single variable, and the higher order system responses and multivariable effects are ignored. A controller designed using classical theory usually requires on-site tuning due to design approximations. Yet, due to the easiness in physical implementation of the controller designs over the controllers designed using modern control theory, these controllers are preferred in most of the industrial applications. Most popular controllers that come under classical control engineering are PID controller. In contrast, modern control theory is strictly carried out in complex-s domain or in frequency domain, and can deal with multi-input and multi-output (MIMO) systems. This overcomes the limitations in classical control theory to be used in sophisticate control systems design problems such as fighter aircraft control. In modern controls a system is represented in terms of a set of first order differential equations defined using state variables. Nonlinear, multivariable, adaptive and robust control theories come under this division. Being fairly new, modern control theory has many areas yet to be explored. Scholars like Rudolf E. Kalman and Aleksandr Lyapunov are well known among the people who have shaped modern control theory.

Originally control engineering was all about continuous systems. Development of computer control tools, posed a requirement of discrete control system engineering because the communications between the computer-based digital controller and the physical system are governed by a computer clock. The equivalent to Laplace transform in the discrete domain is the z-transform. Today many of the control systems are computer controlled and they consist of both digital and analogue components. Therefore, at the design stage either digital components are mapped into the continuous domain and the design is carried out in the continuous domain, or analogue components are mapped in to discrete domain and design is carried out there. The first of these two methods is more commonly encountered in practice because many industrial systems have many continuous systems components, including mechanical, fluid, biological and analogue electrical components, with a few digital controllers.

At many universities, control engineering courses are taught in electrical and electronic engineering, mechanical engineering, and aerospace engineering; in others it is connected to computer science, as most control techniques today are implemented through computers, often as embedded systems (as in the automotive field). The field of control within chemical engineering is often known as process control. It deals primarily with the control of variables in a chemical process in a plant. It is taught as part of the undergraduate curriculum of any chemical engineering program, and employs many of the same principles in control engineering. Other engineering disciplines also overlap with control engineering, as it can be applied to any system for which a suitable model can be derived.

Control engineering has diversified applications that include science, finance management, and even human behavior. Students of control engineering may start with a linear control system course dealing with the time and complex-s domain, which requires a thorough background in elementary mathematics and Laplace transform (called classical control theory). In linear control, the student does frequency and time domain analysis. Digital control and nonlinear control courses require z transformation and algebra respectively, and could be said to complete a basic control education. From here onwards there are several sub branches.

Control systems

Control engineering is the engineering discipline that focuses on the modeling of a diverse range of dynamic systems (e.g. mechanical systems) and the design of controllers that will cause these systems to behave in the desired manner. Although such controllers need not be electrical many are and hence control engineering is often viewed as a subfield of electrical engineering. However, the falling price of microprocessors is making the actual implementation of a control system essentially trivial. As a result, focus is shifting back to the mechanical engineering discipline, as intimate knowledge of the physical system being controlled is often desired.

Electrical circuits, digital signal processors and microcontrollers can all be used to implement Control systems. Control engineering has a wide range of applications from the flight and propulsion systems of commercial airliners to the cruise control present in many modern automobiles.

In most of the cases, control engineers utilize feedback when designing control systems. This is often accomplished using a PID controller system. For example, in an automobile with cruise control the vehicle's speed is continuously monitored and fed back to the system which adjusts the motor's torque accordingly. Where there is regular feedback, control theory can be used to determine how the system responds to such feedback. In practically all such systems stability is important and control theory can help ensure stability is achieved.

Although feedback is an important aspect of control engineering, control engineers may also work on the control of systems without feedback. This is known as open loop control. A classic example of open loop control is a washing machine that runs through a pre-determined cycle without the use of sensors.

Chapter-4

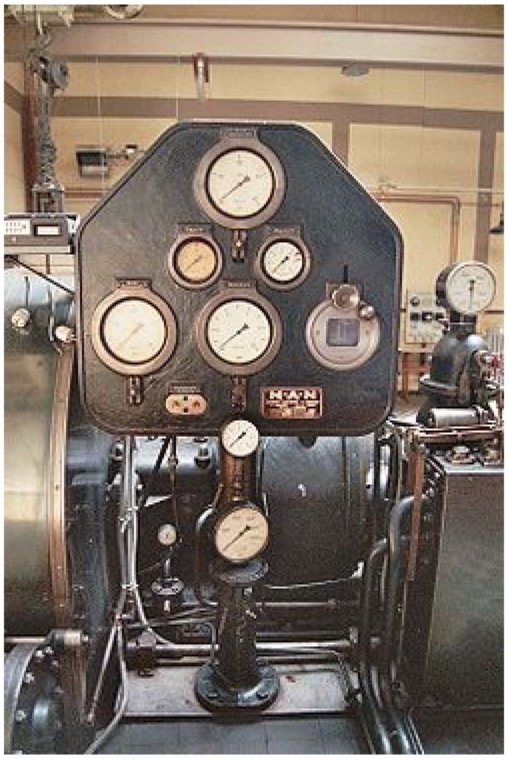
Instrumentation Engineering & Computer Engineering

Instrumentation engineering deals with the design of devices to measure physical quantities such as pressure, flow and temperature. These devices are known as instrumentation.

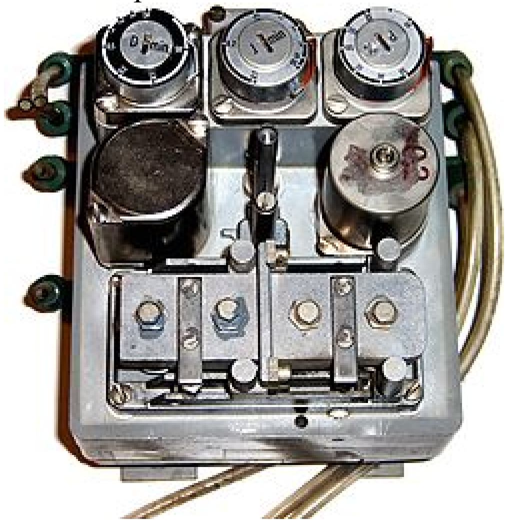
The design of such instrumentation requires a good understanding of physics that often extends beyond electromagnetic theory. For example, radar guns use the Doppler effect to measure the speed of oncoming vehicles. Similarly, thermocouples use the Peltier-Seebeck effect to measure the temperature difference between two points.

Often instrumentation is not used by itself, but instead as the sensors of larger electrical systems. For example, a thermocouple might be used to help ensure a furnace's temperature remains constant. For this reason, instrumentation engineering is often viewed as the counterpart of control engineering.

Instrumentation



A control post of a steam turbine.



Pneumatic PID controller.

Instrumentation is the branch of engineering that deals with measurement and control.

An instrument is a device that measures or manipulates variables such as flow, temperature, level, or pressure. Instruments include many varied contrivances which can be as simple as valves and transmitters, and as complex as analyzers. Instruments often comprise control systems of varied processes. The control of processes is one of the main branches of applied instrumentation.

Control instrumentation includes devices such as solenoids, valves, circuit breakers, and relays. These devices are able to change a field parameter, and provide remote or automated control capabilities.

Transmitters are devices which produce an analog signal, usually in the form of a 4–20 mA electrical current signal, although many other options using voltage, frequency, or pressure are possible. This signal can be used to control other instruments directly, or it can be sent to a PLC, DCS, SCADA system, or other type of computerized controller, where it can be interpreted into readable values and used to control other devices and processes in the system.

Instrumentation plays a significant role in both gathering information from the field and changing the field parameters, and as such are a key part of control loops.

Measurement

Instrumentation can be used to measure certain field parameters (physical values):

These measured values include:

- pressure, either differential or static
- flow
- temperature - Temperature_measurement
- level - Level Measurement
- density
- viscosity
- radiation
- current
- voltage
- inductance
- capacitance
- frequency
- resistivity
- conductivity
- chemical composition
- chemical properties
- various physical properties

Control

Control valve.

In addition to measuring field parameters, instrumentation is also responsible for providing the ability to modify some field parameters.

Some examples include:

Device	Field Parameter(s)
Valve	Flow, Pressure
Relay	Voltage, Current
Solenoid	Physical Location, Level
Circuit breaker	Voltage, Current

Instrumentation engineering

Instrumentation engineering is the engineering specialization focused on the principle and operation of measuring instruments which are used in design and configuration of automated systems in electrical, pneumatic domains etc. They typically work for industries with automated processes, such as chemical or manufacturing plants, with the goal of improving system productivity, reliability, safety, optimization and stability. To control the parameters in a process or in a particular system Microprocessors , Micro controllers ,PLCs etc are used. But their ultimate aim is to control the parameters of a system.

Instrumentation technologists and mechanics

Instrumentation technologists, technicians and mechanics specialize in troubleshooting and repairing and maintenance of instruments and instrumentation systems. This trade is so intertwined with electricians, pipefitters, power engineers, and engineering companies, that one can find him/herself in extremely diverse working situations. An over-arching term, "Instrument Fitter" is often used to describe people in this field, regardless of any specialization.

Computer Engineering

Computer engineering deals with the design of computers and computer systems. This may involve the design of new hardware, the design of PDA's or the use of computers to control an industrial plant. Computer engineers may also work on a system's software. However, the design of complex software systems is often the domain of software engineering, which is usually considered a separate discipline.

Desktop computers represent a tiny fraction of the devices a computer engineer might work on, as computer-like architectures are now found in a range of devices including video game consoles and DVD players.

Computer Engineering (also called **Electronic and Computer Engineering** , or **Computer Systems Engineering**) is a discipline that combines both Electrical Engineering and Computer Science. Computer engineers usually have training in electrical engineering, software design and hardware-software integration instead of only software engineering or electrical engineering. Computer engineers are involved in many aspects of computing, from the design of individual microprocessors, personal computers, and supercomputers, to circuit design. This field of engineering not only focuses on how computer systems themselves work, but also how they integrate into the larger picture.

Usual tasks involving computer engineers include writing software and firmware for embedded microcontrollers, designing VLSI chips, designing analog sensors, designing mixed signal circuit boards, and designing operating systems. Computer engineers are also suited for robotics research, which relies heavily on using digital systems to control and monitor electrical systems like motors, communications, and sensors.

Computer engineering as an academic discipline

The first accredited computer engineering degree program in the United States was established at Case Western Reserve University in 1971; as of October 2004 there were 170 ABET-accredited computer engineering programs in the US.

Due to increasing job requirements for engineers, who can design and manage all forms of computer systems used in industry, some tertiary institutions around the world offer a bachelor's degree generally called computer engineering. Both computer engineering and electronic engineering programs include analog and digital circuit design in their curricula. As with most engineering disciplines, having a sound knowledge of mathematics and sciences is necessary for computer engineers.

In many institutions, computer engineering students are allowed to choose areas of in-depth study in their junior and senior year, as the full breadth of knowledge used in the design and application of computers is well beyond the scope of an undergraduate degree. The joint IEEE/ACM *Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering* defines the core knowledge areas of computer engineering as

- Algorithms
- Computer architecture and organization
- Computer systems engineering
- Circuits and signals
- Database systems
- Digital logic
- Digital signal processing
- Electronics
- Embedded systems
- Human-computer interaction
- Interactive Systems Engineering
- Operating systems
- Programming fundamentals
- Social and Professional issues
- Software engineering
- VLSI design and fabrication

The breadth of disciplines studied in computer engineering is not limited to the above subjects but can include any subject found in engineering.

Algorithm

This is an algorithm that tries to figure out why the lamp doesn't turn on and tries to fix it using the steps. Flowcharts are often used to graphically represent algorithms.

In mathematics, computing, linguistics, and related subjects, an **algorithm** is a finite sequence of instructions, logic, an explicit, step-by-step procedure for solving a problem, often used for calculation and data processing and many other fields. It is formally a type of effective method in which a list of well-defined instructions for completing a task, will when given an initial state, proceed through a well-defined series of successive states, eventually terminating in an end-state. The transition from one state to the next is not necessarily deterministic; some algorithms, known as probabilistic algorithms, incorporate randomness.

A partial formalization of the concept began with attempts to solve the Entscheidungsproblem (the "decision problem") posed by David Hilbert in 1928. Subsequent formalizations were framed as attempts to define "effective calculability" (Kleene 1943:274) or "effective method" (Rosser 1939:225); those formalizations included the Gödel-Herbrand-Kleene recursive functions of 1930, 1934 and 1935, Alonzo Church's lambda calculus of 1936, Emil Post's "Formulation 1" of 1936, and Alan Turing's Turing machines of 1936–7 and 1939.

Etymology

Al-Khwārizmī, Persian astronomer and mathematician, wrote a treatise in 825 AD, *On Calculation with Arabic Numerals*. (See algorism). It was translated into Latin in the 12th century as *Algoritmi de numero Indorum* (al-Daffa 1977), whose title was likely intended to mean "Algoritmi on the numbers of the Indians", where "Algoritmi" was the translator's rendition of the author's name; but people misunderstanding the title treated *Algoritmi* as a Latin plural and this led to the word "algorithm" (Latin *algorismus*) coming to mean "calculation method". The intrusive "th" is most likely due to a false cognate with the Greek ἀριθμός (*arithmos*) meaning "number".

Why algorithms are necessary: an informal definition

While there is no generally accepted *formal* definition of "algorithm", an informal definition could be "a process that performs some sequence of operations." For some people, a program is only an algorithm if it stops eventually. For others, a program is only an algorithm if it stops before a given number of calculation steps.

A prototypical example of an "algorithm" is Euclid's algorithm to determine the maximum common divisor of two integers (X and Y) which are greater than one: We follow a series of steps: In step i, we divide X by Y and find the remainder, which we call R_1. Then we move to step i + 1, where we divide Y by R_1, and find the remainder, which we call R_2. If R_2=0, we stop and say that R_1 is the greatest common divisor of X and Y. If not, we continue, until R_n=0. Then R_{n-1} is the max common division of X and Y. This procedure is known to stop always and the number of subtractions needed is always smaller than the larger of the two numbers.

We can derive clues to the issues involved and an informal meaning of the word from the following quotation from Boolos & Jeffrey (1974, 1999) (boldface added):

No human being can write fast enough or long enough or small enough to list all members of an enumerably infinite set by writing out their names, one after another, in some notation. But humans can do something equally useful, in the case of certain enumerably infinite sets: They can give **explicit instructions for determining the nth member of the set**, for arbitrary finite n. Such instructions are to be given quite explicitly, in a form in which **they could be followed by a computing machine**, or by a **human who is capable of carrying out only very elementary operations on symbols** (Boolos & Jeffrey 1974, 1999, p. 19)

The words "enumerably infinite" mean "countable using integers perhaps extending to infinity." Thus Boolos and Jeffrey are saying that an algorithm *implies* instructions for a process that "creates" output integers from an *arbitrary* "input" integer or integers that, in theory, can be chosen from 0 to infinity. Thus we might expect an algorithm to be an algebraic equation such as **y = m + n** — two arbitrary "input variables" **m** and **n** that produce an output **y**. As we see in Algorithm characterizations — the word algorithm implies much more than this, something on the order of (for our addition example):

> Precise instructions (in language understood by "the computer") for a "fast, efficient, good" *process* that specifies the "moves" of "the computer" (machine or human, equipped with the necessary internally-contained information and capabilities) to find, decode, and then munch arbitrary input integers/symbols **m** and **n**, symbols + and = ... and (reliably, correctly, "effectively") produce, in a "reasonable" time, output-integer **y** at a specified place and in a specified format.

The concept of *algorithm* is also used to define the notion of decidability. That notion is central for explaining how formal systems come into being starting from a small set of axioms and rules. In logic, the time that an algorithm requires to complete cannot be measured, as it is not apparently related with our customary physical dimension. From such uncertainties, that characterize ongoing work, stems the unavailability of a definition of *algorithm* that suits both concrete (in some sense) and abstract usage of the term.

Formalization

Algorithms are essential to the way computers process information. Many computer programs contain algorithms that specify the specific instructions a computer should perform (in a specific order) to carry out a specified task, such as calculating employees' paychecks or printing students' report cards. Thus, an algorithm can be considered to be any sequence of operations that can be simulated by a Turing-complete system. Authors who assert this thesis include Savage (1987) and Gurevich (2000):

...Turing's informal argument in favor of his thesis justifies a stronger thesis: every algorithm can be simulated by a Turing machine (Gurevich 2000:1)...according to Savage [1987], an algorithm is a computational process defined by a Turing machine. (Gurevich 2000:3)

Typically, when an algorithm is associated with processing information, data is read from an input source, written to an output device, and/or stored for further processing. Stored data is regarded as part of the internal state of the entity performing the algorithm. In practice, the state is stored in one or more data structures.

For any such computational process, the algorithm must be rigorously defined: specified in the way it applies in all possible circumstances that could arise. That is, any conditional steps must be systematically dealt with, case-by-case; the criteria for each case must be clear (and computable).

Because an algorithm is a precise list of precise steps, the order of computation will always be critical to the functioning of the algorithm. Instructions are usually assumed to be listed explicitly, and are described as starting "from the top" and going "down to the bottom", an idea that is described more formally by *flow of control*.

So far, this discussion of the formalization of an algorithm has assumed the premises of imperative programming. This is the most common conception, and it attempts to describe a task in discrete, "mechanical" means. Unique to this conception of formalized algorithms is the assignment operation, setting the value of a variable. It derives from the intuition of "memory" as a scratchpad. There is an example below of such an assignment.

Termination

Some writers restrict the definition of *algorithm* to procedures that eventually finish. In such a category Kleene places the "*decision procedure* or *decision method* or *algorithm* for the question" (Kleene 1952:136). Others, including Kleene, include procedures that could run forever without stopping; such a procedure has been called a "computational method" (Knuth 1997:5) or "*calculation procedure* or *algorithm*" (Kleene 1952:137); however, Kleene notes that such a method must eventually exhibit "some object" (Kleene 1952:137).

Minsky makes the pertinent observation, in regards to determining whether an algorithm will eventually terminate (from a particular starting state):

But if the length of the process is not known in advance, then "trying" it may not be decisive, because if the process does go on forever — then at no time will we ever be sure of the answer (Minsky 1967:105).

As it happens, no other method can do any better, as was shown by Alan Turing with his celebrated result on the undecidability of the so-called halting problem. There is no algorithmic procedure for determining of arbitrary algorithms whether or not they terminate from given starting states. The analysis of algorithms for their likelihood of termination is called termination analysis.

See the examples of (im-)"proper" subtraction at partial function for more about what can happen when an algorithm fails for certain of its input numbers — e.g., (i) nontermination, (ii) production of "junk" (output in the wrong format to be considered a number) or no number(s) at all (halt ends the computation with no output), (iii) wrong number(s), or (iv) a combination of these. Kleene proposed that the production of "junk" or failure to produce a number is solved by having the algorithm detect these instances and produce e.g., an error message (he suggested "0"), or preferably, force the algorithm into an endless loop (Kleene 1952:322). Davis does this to his subtraction algorithm — he fixes his algorithm in a second example so that it is proper subtraction (Davis 1958:12-15). Along with the logical outcomes "true" and "false" Kleene also proposes the use of a third logical symbol "u" — undecided (Kleene 1952:326) — thus an algorithm will always produce *something* when confronted with a "proposition". The problem of wrong answers must be solved with an independent "proof" of the algorithm e.g., using induction:

We normally require auxiliary evidence for this (that the algorithm correctly defines a mu recursive function), e.g., in the form of an inductive proof that, for each argument value, the computation terminates with a unique value (Minsky 1967:186).

Expressing algorithms

Algorithms can be expressed in many kinds of notation, including natural languages, pseudocode, flowcharts, and programming languages. Natural language expressions of algorithms tend to be verbose and ambiguous, and are rarely used for complex or technical algorithms. Pseudocode and flowcharts are structured ways to express algorithms that avoid many of the ambiguities common in natural language statements, while remaining independent of a particular implementation language. Programming languages are primarily intended for expressing algorithms in a form that can be executed by a computer, but are often used as a way to define or document algorithms.

There is a wide variety of representations possible and one can express a given Turing machine program as a sequence of machine tables (see more at finite state machine and state transition table), as flowcharts (see more at state diagram), or as a form of rudimentary machine code or assembly code called "sets of quadruples" (see more at Turing machine).

Sometimes it is helpful in the description of an algorithm to supplement small "flow charts" (state diagrams) with natural-language and/or arithmetic expressions written inside "block diagrams" to summarize what the "flow charts" are accomplishing.

Representations of algorithms are generally classed into three accepted levels of Turing machine description (Sipser 2006:157):

- **1 High-level description**:

  "...prose to describe an algorithm, ignoring the implementation details. At this level we do not need to mention how the machine manages its tape or head"

- **2 Implementation description**:

  "...prose used to define the way the Turing machine uses its head and the way that it stores data on its tape. At this level we do not give details of states or transition function"

- **3 Formal description**:

  Most detailed, "lowest level", gives the Turing machine's "state table".
  For an example of the simple algorithm "Add m+n" described in all three levels.

Computer algorithms

In computer systems, an algorithm is basically an instance of logic written in software by software developers to be effective for the intended "target" computer(s), in order for the software on the target machines to *do something*. For instance, if a person is writing software that is supposed to print out a PDF document located at the operating system folder "/My Documents" at computer drive "D:" every Friday at 10PM, they will write an algorithm that specifies the following actions: "If today's date (computer time) is 'Friday,' open the document at 'D:/My Documents' and call the 'print' function". While this simple algorithm does not look into whether the printer has enough paper or whether the document has been moved into a different location, one can make this algorithm more robust and anticipate these problems by rewriting it as a formal CASE statement or as a (carefully crafted) sequence of IF-THEN-ELSE statements. For example the CASE statement might appear as follows (there are other possibilities):

> CASE 1: IF today's date is NOT Friday THEN *exit this CASE instruction* ELSE
> CASE 2: IF today's date is Friday AND the document is located at 'D:/My Documents' AND there is paper in the printer THEN print the document (and *exit this CASE instruction*) ELSE
> CASE 3: IF today's date is Friday AND the document is NOT located at 'D:/My Documents' THEN display 'document not found' error message (and *exit this CASE instruction*) ELSE
> CASE 4: IF today's date is Friday AND the document is located at 'D:/My Documents' AND there is NO paper in the printer THEN (i) display 'out of paper' error message and (ii) *exit*.

Note that CASE 3 includes two possibilities: (i) the document is NOT located at 'D:/My Documents' AND there's paper in the printer OR (ii) the document is NOT located at 'D:/My Documents' AND there's paper in the printer.

The sequence of IF-THEN-ELSE tests might look like this:

> TEST 1: IF today's date is NOT Friday THEN *done* ELSE TEST 2:
> TEST 2: IF the document is located at 'D:/My Documents' THEN display 'document not found' error message ELSE TEST 3:

TEST 3: IF there is NO paper in the printer THEN display 'out of paper' error message ELSE print the document.

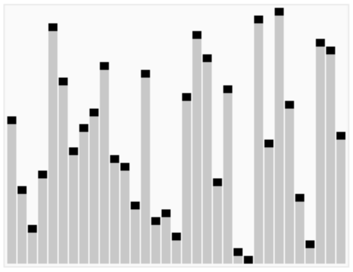
These examples' logic grants precedence to the instance of "NO document at 'D:/My Documents' ". Also observe that in a well-crafted CASE statement or sequence of IF-THEN-ELSE statements the number of distinct actions -- 4 in these examples: do nothing, print the document, display 'document not found', display 'out of paper' -- equals the number of cases.

Because a computational machine equipped with unbounded memory and the ability to execute CASE statements or a sequence of IF-THEN-ELSE statements together with just a few other instructions is Turing complete, anything that is computable will be computable by this machine. Thus this form of algorithm is fundamental to computer programming in all its forms (see more at McCarthy formalism).

Implementation

Most algorithms are intended to be implemented as computer programs. However, algorithms are also implemented by other means, such as in a biological neural network (for example, the human brain implementing arithmetic or an insect looking for food), in an electrical circuit, or in a mechanical device.

Example



An animation of the quicksort algorithm sorting an array of randomized values. The red bars mark the pivot element; at the start of the animation, the element farthest to the right hand side is chosen as the pivot.

One of the simplest algorithms is to find the largest number in an (unsorted) list of numbers. The solution necessarily requires looking at every number in the list, but only once at each. From this follows a simple algorithm, which can be stated in a high-level description English prose, as:

High-level description:

1. Assume the first item is largest.

2. Look at each of the remaining items in the list and if it is larger than the largest item so far, make a note of it.
3. The last noted item is the largest in the list when the process is complete.

(Quasi-)formal description: Written in prose but much closer to the high-level language of a computer program, the following is the more formal coding of the algorithm in pseudocode or pidgin code:

```
Algorithm LargestNumber
  Input: A non-empty list of numbers L.
  Output: The largest number in the list L.

  largest ← $L_0$
  for each item in the list $L_{\geq 1}$, do
    if the item > largest, then
      largest ← the item
  return largest
```

- "←" is a loose shorthand for "changes to". For instance, "*largest* ← *item*" means that the value of *largest* changes to the value of *item*.
- "**return**" terminates the algorithm and outputs the value that follows.

For a more complex example of an algorithm, see Euclid's algorithm for the greatest common divisor, one of the earliest algorithms known.

Algorithmic analysis

It is frequently important to know how much of a particular resource (such as time or storage) is required for a given algorithm. Methods have been developed for the analysis of algorithms to obtain such quantitative answers; for example, the algorithm above has a time requirement of O(*n*), using the big O notation with *n* as the length of the list. At all times the algorithm only needs to remember two values: the largest number found so far, and its current position in the input list. Therefore it is said to have a space requirement of *O(1)*, if the space required to store the input numbers is not counted, or O(*n*) if it is counted.

Different algorithms may complete the same task with a different set of instructions in less or more time, space, or 'effort' than others. For example, a binary search algorithm will usually outperform a brute force sequential search when used for table lookups on sorted lists.

Abstract versus empirical

The analysis and study of algorithms is a discipline of computer science, and is often practiced abstractly without the use of a specific programming language or implementation. In this sense, algorithm analysis resembles other mathematical disciplines in that it focuses on the underlying properties of the algorithm and not on the specifics of any particular implementation. Usually pseudocode is used for analysis as it is the simplest and most general representation. However, ultimately, most algorithms are usually implemented on particular hardware / software platforms and their algorithmic efficiency is eventually put to the test using real code.

Empirical testing is useful because it may uncover unexpected interactions that affect performance. For instance an algorithm that has no locality of reference may have much poorer performance than predicted because it thrashes the cache.

Classification

There are various ways to classify algorithms, each with its own merits.

By implementation

One way to classify algorithms is by implementation means.

- **Recursion** or **iteration**: A recursive algorithm is one that invokes (makes reference to) itself repeatedly until a certain condition matches, which is a method common to functional programming. Iterative algorithms use repetitive constructs like loops and sometimes additional data structures like stacks to solve the given problems. Some problems are naturally suited for one implementation or the other. For example, towers of Hanoi is well understood in recursive implementation. Every recursive version has an equivalent (but possibly more or less complex) iterative version, and vice versa.
- **Logical**: An algorithm may be viewed as controlled logical deduction. This notion may be expressed as: **Algorithm = logic + control** (Kowalski 1979). The logic component expresses the axioms that may be used in the computation and the control component determines the way in which deduction is applied to the axioms. This is the basis for the logic programming paradigm. In pure logic programming languages the control component is fixed and algorithms are specified by supplying only the logic component. The appeal of this approach is the elegant semantics: a change in the axioms has a well defined change in the algorithm.
- **Serial** or **parallel** or **distributed**: Algorithms are usually discussed with the assumption that computers execute one instruction of an algorithm at a time. Those computers are sometimes called serial computers. An algorithm designed for such an environment is called a serial algorithm, as opposed to parallel algorithms or distributed algorithms. Parallel algorithms take advantage of computer architectures where several processors can work on a problem at the same time, whereas distributed algorithms utilize multiple machines connected with a network. Parallel or distributed algorithms divide the problem into more symmetrical or asymmetrical subproblems and collect the results back together. The resource consumption in such algorithms is not only processor cycles on each processor but also the communication overhead between the processors. Sorting algorithms can be parallelized efficiently, but their communication overhead is

expensive. Iterative algorithms are generally parallelizable. Some problems have no parallel algorithms, and are called inherently serial problems.

- **Deterministic** or **non-deterministic**: Deterministic algorithms solve the problem with exact decision at every step of the algorithm whereas non-deterministic algorithms solve problems via guessing although typical guesses are made more accurate through the use of heuristics.
- **Exact** or **approximate**: While many algorithms reach an exact solution, approximation algorithms seek an approximation that is close to the true solution. Approximation may use either a deterministic or a random strategy. Such algorithms have practical value for many hard problems.

By design paradigm

Another way of classifying algorithms is by their design methodology or paradigm. There is a certain number of paradigms, each different from the other. Furthermore, each of these categories will include many different types of algorithms. Some commonly found paradigms include:

- **Divide and conquer**. A divide and conquer algorithm repeatedly reduces an instance of a problem to one or more smaller instances of the same problem (usually recursively) until the instances are small enough to solve easily. One such example of divide and conquer is merge sorting. Sorting can be done on each segment of data after dividing data into segments and sorting of entire data can be obtained in the conquer phase by merging the segments. A simpler variant of divide and conquer is called a **decrease and conquer algorithm**, that solves an identical subproblem and uses the solution of this subproblem to solve the bigger problem. Divide and conquer divides the problem into multiple subproblems and so the conquer stage will be more complex than decrease and conquer algorithms. An example of decrease and conquer algorithm is the binary search algorithm.
- **Dynamic programming**. When a problem shows optimal substructure, meaning the optimal solution to a problem can be constructed from optimal solutions to subproblems, and overlapping subproblems, meaning the same subproblems are used to solve many different problem instances, a quicker approach called *dynamic programming* avoids recomputing solutions that have already been computed. For example, the shortest path to a goal from a vertex in a weighted graph can be found by using the shortest path to the goal from all adjacent vertices. Dynamic programming and memoization go together. The main difference between dynamic programming and divide and conquer is that subproblems are more or less independent in divide and conquer, whereas subproblems overlap in dynamic programming. The difference between dynamic programming and straightforward recursion is in caching or memoization of recursive calls. When subproblems are independent and there is no repetition, memoization does not help; hence dynamic programming is not a solution for all complex problems. By using memoization or maintaining a table of subproblems already solved, dynamic programming reduces the exponential nature of many problems to polynomial complexity.

- **The greedy method**. A greedy algorithm is similar to a dynamic programming algorithm, but the difference is that solutions to the subproblems do not have to be known at each stage; instead a "greedy" choice can be made of what looks best for the moment. The greedy method extends the solution with the best possible decision (not all feasible decisions) at an algorithmic stage based on the current local optimum and the best decision (not all possible decisions) made in a previous stage. It is not exhaustive, and does not give accurate answer to many problems. But when it works, it will be the fastest method. The most popular greedy algorithm is finding the minimal spanning tree as given by Kruskal.
- **Linear programming**. When solving a problem using linear programming, specific inequalities involving the inputs are found and then an attempt is made to maximize (or minimize) some linear function of the inputs. Many problems (such as the maximum flow for directed graphs) can be stated in a linear programming way, and then be solved by a 'generic' algorithm such as the simplex algorithm. A more complex variant of linear programming is called integer programming, where the solution space is restricted to the integers.
- **Reduction**. This technique involves solving a difficult problem by transforming it into a better known problem for which we have (hopefully) asymptotically optimal algorithms. The goal is to find a reducing algorithm whose complexity is not dominated by the resulting reduced algorithm's. For example, one selection algorithm for finding the median in an unsorted list involves first sorting the list (the expensive portion) and then pulling out the middle element in the sorted list (the cheap portion). This technique is also known as *transform and conquer*.
- **Search and enumeration**. Many problems (such as playing chess) can be modeled as problems on graphs. A graph exploration algorithm specifies rules for moving around a graph and is useful for such problems. This category also includes search algorithms, branch and bound enumeration and backtracking.
- **The probabilistic and heuristic paradigm**. Algorithms belonging to this class fit the definition of an algorithm more loosely.

1. Probabilistic algorithms are those that make some choices randomly (or pseudo-randomly); for some problems, it can in fact be proven that the fastest solutions must involve some randomness.
2. Genetic algorithms attempt to find solutions to problems by mimicking biological evolutionary processes, with a cycle of random mutations yielding successive generations of "solutions". Thus, they emulate reproduction and "survival of the fittest". In genetic programming, this approach is extended to algorithms, by regarding the algorithm itself as a "solution" to a problem.
3. Heuristic algorithms, whose general purpose is not to find an optimal solution, but an approximate solution where the time or resources are limited. They are not practical to find perfect solutions. An example of this would be local search, tabu search, or simulated annealing algorithms, a class of heuristic probabilistic algorithms that vary the solution of a problem by a random amount. The name "simulated annealing" alludes to the metallurgic term meaning the heating and cooling of metal to achieve freedom from defects. The purpose of the random variance is to find close to globally optimal solutions rather than simply locally

optimal ones, the idea being that the random element will be decreased as the algorithm settles down to a solution.

By field of study

Every field of science has its own problems and needs efficient algorithms. Related problems in one field are often studied together. Some example classes are search algorithms, sorting algorithms, merge algorithms, numerical algorithms, graph algorithms, string algorithms, computational geometric algorithms, combinatorial algorithms, machine learning, cryptography, data compression algorithms and parsing techniques.

Fields tend to overlap with each other, and algorithm advances in one field may improve those of other, sometimes completely unrelated, fields. For example, dynamic programming was originally invented for optimization of resource consumption in industry, but is now used in solving a broad range of problems in many fields.

By complexity

Algorithms can be classified by the amount of time they need to complete compared to their input size. There is a wide variety: some algorithms complete in linear time relative to input size, some do so in an exponential amount of time or even worse, and some never halt. Additionally, some problems may have multiple algorithms of differing complexity, while other problems might have no algorithms or no known efficient algorithms. There are also mappings from some problems to other problems. Owing to this, it was found to be more suitable to classify the problems themselves instead of the algorithms into equivalence classes based on the complexity of the best possible algorithms for them.

By computing power

Another way to classify algorithms is by computing power. This is typically done by considering some collection (class) of algorithms. A recursive class of algorithms is one that includes algorithms for all Turing computable functions. Looking at classes of algorithms allows for the possibility of restricting the available computational resources (time and memory) used in a computation. A subrecursive class of algorithms is one in which not all Turing computable functions can be obtained. For example, the algorithms that run in polynomial time suffice for many important types of computation but do not exhaust all Turing computable functions. The class of algorithms implemented by primitive recursive functions is another subrecursive class.

Burgin (2005, p. 24) uses a generalized definition of algorithms that relaxes the common requirement that the output of the algorithm that computes a function must be determined after a finite number of steps. He defines a super-recursive class of algorithms as "a class of algorithms in which it is possible to compute functions not computable by any Turing machine" (Burgin 2005, p. 107). This is closely related to the study of methods of hypercomputation.

Legal issues

Algorithms, by themselves, are not usually patentable. In the United States, a claim consisting solely of simple manipulations of abstract concepts, numbers, or signals does not constitute "processes" (USPTO 2006), and hence algorithms are not patentable (as in Gottschalk v. Benson). However, practical applications of algorithms are sometimes patentable. For example, in Diamond v. Diehr, the application of a simple feedback algorithm to aid in the curing of synthetic rubber was deemed patentable. The patenting of software is highly controversial, and there are highly criticized patents involving algorithms, especially data compression algorithms, such as Unisys' LZW patent.

History: Development of the notion of "algorithm"

Origin of the word

The word *algorithm* comes from the name of the 9th century Persian mathematician Abu Abdullah Muhammad ibn Musa al-Khwarizmi whose works introduced Indian numerals and algebraic concepts. He worked in Baghdad at the time when it was the centre of scientific studies and trade. The word *algorism* originally referred only to the rules of performing arithmetic using Arabic numerals but evolved via European Latin translation of al-Khwarizmi's name into *algorithm* by the 18th century. The word evolved to include all definite procedures for solving problems or performing tasks.

Discrete and distinguishable symbols

Tally-marks: To keep track of their flocks, their sacks of grain and their money the ancients used tallying: accumulating stones or marks scratched on sticks, or making discrete symbols in clay. Through the Babylonian and Egyptian use of marks and symbols, eventually Roman numerals and the abacus evolved (Dilson, p.16–41). Tally marks appear prominently in unary numeral system arithmetic used in Turing machine and Post-Turing machine computations.

Manipulation of symbols as "place holders" for numbers: algebra

The work of the ancient Greek geometers, Persian mathematician Al-Khwarizmi (often considered the "father of algebra" and from whose name the terms "algorism" and "algorithm" are derived), and Western European mathematicians culminated in Leibniz's notion of the calculus ratiocinator (ca 1680):

> "A good century and a half ahead of his time, Leibniz proposed an algebra of logic, an algebra that would specify the rules for manipulating logical concepts in

> the manner that ordinary algebra specifies the rules for manipulating numbers" (Davis 2000:1)

Mechanical contrivances with discrete states

The clock: Bolter credits the invention of the weight-driven clock as “The key invention [of Europe in the Middle Ages]", in particular the verge escapement (Bolter 1984:24) that provides us with the tick and tock of a mechanical clock. “The accurate automatic machine” (Bolter 1984:26) led immediately to "mechanical automata" beginning in the thirteenth century and finally to “computational machines" – the difference engine and analytical engines of Charles Babbage and Countess Ada Lovelace (Bolter p.33–34, p.204–206).

Jacquard loom, Hollerith punch cards, telegraphy and telephony — the electromechanical relay: Bell and Newell (1971) indicate that the Jacquard loom (1801), precursor to Hollerith cards (punch cards, 1887), and “telephone switching technologies” were the roots of a tree leading to the development of the first computers (Bell and Newell diagram p. 39, cf. Davis 2000). By the mid-1800s the telegraph, the precursor of the telephone, was in use throughout the world, its discrete and distinguishable encoding of letters as “dots and dashes” a common sound. By the late 1800s the ticker tape (ca 1870s) was in use, as was the use of Hollerith cards in the 1890 U.S. census. Then came the Teletype (ca. 1910) with its punched-paper use of Baudot code on tape.

Telephone-switching networks of electromechanical relays (invented 1835) was behind the work of George Stibitz (1937), the inventor of the digital adding device. As he worked in Bell Laboratories, he observed the “burdensome’ use of mechanical calculators with gears. "He went home one evening in 1937 intending to test his idea... When the tinkering was over, Stibitz had constructed a binary adding device". (Valley News, p. 13).

Davis (2000) observes the particular importance of the electromechanical relay (with its two "binary states" *open* and *closed*):

> It was only with the development, beginning in the 1930s, of electromechanical calculators using electrical relays, that machines were built having the scope Babbage had envisioned." (Davis, p. 14).

Mathematics during the 1800s up to the mid-1900s

Symbols and rules: In rapid succession the mathematics of George Boole (1847, 1854), Gottlob Frege (1879), and Giuseppe Peano (1888–1889) reduced arithmetic to a sequence of symbols manipulated by rules. Peano's *The principles of arithmetic, presented by a new method* (1888) was "the first attempt at an axiomatization of mathematics in a symbolic language" (van Heijenoort:81ff).

But Heijenoort gives Frege (1879) this kudos: Frege’s is "perhaps the most important single work ever written in logic. ... in which we see a " 'formula language', that is a *lingua characterica*, a language written with special symbols, "for pure thought", that is, free from rhetorical embellishments ... constructed from specific symbols that are manipulated according to definite rules" (van Heijenoort:1). The work of Frege was further simplified and amplified by Alfred North Whitehead and Bertrand Russell in their Principia Mathematica (1910–1913).

The paradoxes: At the same time a number of disturbing paradoxes appeared in the literature, in particular the Burali-Forti paradox (1897), the Russell paradox (1902–03), and the Richard Paradox (Dixon 1906, cf. Kleene 1952:36–40). The resultant considerations led to Kurt Gödel's paper (1931) — he specifically cites the paradox of the liar — that completely reduces rules of recursion to numbers.

Effective calculability: In an effort to solve the Entscheidungsproblem defined precisely by Hilbert in 1928, mathematicians first set about to define what was meant by an "effective method" or "effective calculation" or "effective calculability" (i.e., a calculation that would succeed). In rapid succession the following appeared: Alonzo Church, Stephen Kleene and J.B. Rosser's λ-calculus, (cf. footnote in Alonzo Church 1936a:90, 1936b:110) a finely-honed definition of "general recursion" from the work of Gödel acting on suggestions of Jacques Herbrand (cf. Gödel's Princeton lectures of 1934) and subsequent simplifications by Kleene (1935-6:237ff, 1943:255ff). Church's proof (1936:88ff) that the Entscheidungsproblem was unsolvable, Emil Post's definition of effective calculability as a worker mindlessly following a list of instructions to move left or right through a sequence of rooms and while there either mark or erase a paper or observe the paper and make a yes-no decision about the next instruction (cf. "Formulation I", Post 1936:289-290). Alan Turing's proof of that the Entscheidungsproblem was unsolvable by use of his "a- [automatic-] machine"(Turing 1936-7:116ff) -- in effect almost identical to Post's "formulation", J. Barkley Rosser's definition of "effective method" in terms of "a machine" (Rosser 1939:226). S. C. Kleene's proposal of a precursor to "Church thesis" that he called "Thesis I" (Kleene 1943:273–274), and a few years later Kleene's renaming his Thesis "Church's Thesis" (Kleene 1952:300, 317) and proposing "Turing's Thesis" (Kleene 1952:376).

Emil Post (1936) and Alan Turing (1936-7, 1939)

Here is a remarkable coincidence of two men not knowing each other but describing a process of men-as-computers working on computations — and they yield virtually identical definitions.

Emil Post (1936) described the actions of a "computer" (human being) as follows:

> "...two concepts are involved: that of a *symbol space* in which the work leading from problem to answer is to be carried out, and a fixed unalterable *set of directions*.

His symbol space would be

"a two way infinite sequence of spaces or boxes... The problem solver or worker is to move and work in this symbol space, being capable of being in, and operating in but one box at a time.... a box is to admit of but two possible conditions, i.e., being empty or unmarked, and having a single mark in it, say a vertical stroke.
"One box is to be singled out and called the starting point. ...a specific problem is to be given in symbolic form by a finite number of boxes [i.e., INPUT] being marked with a stroke. Likewise the answer [i.e., OUTPUT] is to be given in symbolic form by such a configuration of marked boxes....
"A set of directions applicable to a general problem sets up a deterministic process when applied to each specific problem. This process will terminate only when it comes to the direction of type (C) [i.e., STOP]." (U p. 289–290) See more at Post-Turing machine

Alan Turing's work (1936, 1939:160) preceded that of Stibitz (1937); it is unknown whether Stibitz knew of the work of Turing. Turing's biographer believed that Turing's use of a typewriter-like model derived from a youthful interest: "Alan had dreamt of inventing typewriters as a boy; Mrs. Turing had a typewriter; and he could well have begun by asking himself what was meant by calling a typewriter 'mechanical'" (Hodges, p. 96). Given the prevalence of Morse code and telegraphy, ticker tape machines, and Teletypes we might conjecture that all were influences.

Turing — his model of computation is now called a Turing machine — begins, as did Post, with an analysis of a human computer that he whittles down to a simple set of basic motions and "states of mind". But he continues a step further and creates a machine as a model of computation of numbers (Turing 1936-7:116).

"Computing is normally done by writing certain symbols on paper. We may suppose this paper is divided into squares like a child's arithmetic book....I assume then that the computation is carried out on one-dimensional paper, i.e., on a tape divided into squares. I shall also suppose that the number of symbols which may be printed is finite....
"The behavior of the computer at any moment is determined by the symbols which he is observing, and his "state of mind" at that moment. We may suppose that there is a bound B to the number of symbols or squares which the computer can observe at one moment. If he wishes to observe more, he must use successive observations. We will also suppose that the number of states of mind which need be taken into account is finite...
"Let us imagine that the operations performed by the computer to be split up into 'simple operations' which are so elementary that it is not easy to imagine them further divided" (Turing 1936-7:136).

Turing's reduction yields the following:

"The simple operations must therefore include:
"(a) Changes of the symbol on one of the observed squares

"(b) Changes of one of the squares observed to another square within L squares of one of the previously observed squares.

"It may be that some of these change necessarily invoke a change of state of mind. The most general single operation must therefore be taken to be one of the following:

"(A) A possible change (a) of symbol together with a possible change of state of mind.
"(B) A possible change (b) of observed squares, together with a possible change of state of mind"
"We may now construct a machine to do the work of this computer." (Turing 1936-7:136)

A few years later, Turing expanded his analysis (thesis, definition) with this forceful expression of it:

"A function is said to be "effectively calculable" if its values can be found by some purely mechanical process. Although it is fairly easy to get an intuitive grasp of this idea, it is nevertheless desirable to have some more definite, mathematical expressible definition . . . [he discusses the history of the definition pretty much as presented above with respect to Gödel, Herbrand, Kleene, Church, Turing and Post] . . . We may take this statement literally, understanding by a purely mechanical process one which could be carried out by a machine. It is possible to give a mathematical description, in a certain normal form, of the structures of these machines. The development of these ideas leads to the author's definition of a computable function, and to an identification of computability † with effective calculability . . . .
"† We shall use the expression "computable function" to mean a function calculable by a machine, and we let "effectively calculable" refer to the intuitive idea without particular identification with any one of these definitions."(Turing 1939:160)

J. B. Rosser (1939) and S. C. Kleene (1943)

J. Barkley Rosser boldly defined an 'effective [mathematical] method' in the following manner (boldface added):

"'Effective method' is used here in the rather special sense of a method each step of which is precisely determined and which is certain to produce the answer in a finite number of steps. With this special meaning, three different precise definitions have been given to date. [his footnote #5; see discussion immediately below]. The simplest of these to state (due to Post and Turing) says essentially that **an effective method of solving certain sets of problems exists if one can build a machine which will then solve any problem of the set with no human intervention beyond inserting the question and (later) reading the answer**. All three definitions are equivalent, so it doesn't matter which one is used.

> Moreover, the fact that all three are equivalent is a very strong argument for the correctness of any one." (Rosser 1939:225–6)

Rosser's footnote #5 references the work of (1) Church and Kleene and their definition of λ-definability, in particular Church's use of it in his *An Unsolvable Problem of Elementary Number Theory* (1936); (2) Herbrand and Gödel and their use of recursion in particular Gödel's use in his famous paper *On Formally Undecidable Propositions of Principia Mathematica and Related Systems I* (1931); and (3) Post (1936) and Turing (1936-7) in their mechanism-models of computation.

Stephen C. Kleene defined as his now-famous "Thesis I" known as the Church-Turing thesis. But he did this in the following context (boldface in original):

> "12. **Algorithmic theories**... In setting up a complete algorithmic theory, what we do is to describe a procedure, performable for each set of values of the independent variables, which procedure necessarily terminates and in such manner that from the outcome we can read a definite answer, "yes" or "no," to the question, "is the predicate value true?”" (Kleene 1943:273)

History after 1950

A number of efforts have been directed toward further refinement of the definition of "algorithm", and activity is on-going because of issues surrounding, in particular, foundations of mathematics (especially the Church-Turing Thesis) and philosophy of mind (especially arguments around artificial intelligence). For more, see Algorithm characterizations.

Electrical network

An **electrical network** is an interconnection of electrical elements such as resistors, inductors, capacitors, transmission lines, voltage sources, current sources, and switches.

An **electrical circuit** is a network that has a closed loop, giving a return path for the current. A network is a connection of two or more components, and may not necessarily be a circuit.

Electrical networks that consist only of sources (voltage or current), linear lumped elements (resistors, capacitors, inductors), and linear distributed elements (transmission lines) can be analyzed by algebraic and transform methods to determine DC response, AC response, and transient response.

A network that also contains active electronic components is known as an **electronic circuit**. Such networks are generally nonlinear and require more complex design and analysis tools.

Design methods

To design any electrical circuit, either analog or digital, electrical engineers need to be able to predict the voltages and currents at all places within the circuit. Linear circuits, that is, circuits with the same input and output frequency, can be analyzed by hand using complex number theory. Other circuits can only be analyzed with specialized software programs or estimation techniques.

Circuit simulation software, such as VHDL and HSPICE, allows engineers to design circuits without the time, cost and risk of error involved in building circuit prototypes.

Electrical laws

A number of electrical laws apply to all electrical networks. These include

- Kirchhoff's current law: The sum of all currents entering a node is equal to the sum of all currents leaving the node.
- Kirchhoff's voltage law: The directed sum of the electrical potential differences around a loop must be zero.
- Ohm's law: The voltage across a resistor is equal to the product of the resistance and the current flowing through it (at constant temperature).
- Norton's theorem: Any network of voltage and/or current sources and resistors is electrically equivalent to an ideal current source in parallel with a single resistor.
- Thévenin's theorem: Any network of voltage and/or current sources and resistors is electrically equivalent to a single voltage source in series with a single resistor.

Other more complex laws may be needed if the network contains nonlinear or reactive components. Non-linear self-regenerative heterodyning systems can be approximated. Applying these laws results in a set of simultaneous equations that can be solved either by hand or by a computer.

Network simulation software

More complex circuits can be analyzed numerically with software such as SPICE or symbolically using software such as SapWin.

Linearization around operating point

When faced with a new circuit, the software first tries to find a steady state solution, that is, one where all nodes conform to Kirchhoff's Current Law *and* the voltages across and through each element of the circuit conform to the voltage/current equations governing that element.

Once the steady state solution is found, the **operating points** of each element in the circuit are known. For a small signal analysis, every non-linear element can be linearized around its operation point to obtain the small-signal estimate of the voltages and currents. This is an application of Ohm's Law. The resulting linear circuit matrix can be solved with Gaussian elimination.

Piecewise-linear approximation

Software such as the PLECS interface to Simulink uses piecewise-linear approximation of the equations governing the elements of a circuit. The circuit is treated as a completely linear network of ideal diodes. Every time a diode switches from on to off or vice versa, the configuration of the linear network changes. Adding more detail to the approximation of equations increases the accuracy of the simulation, but also increases its running time.

Database

A **database** is an integrated collection of logically related records or files which consolidates records into a common pool of data records that provides data for many applications. A database is a collection of information that is organized so that it can easily be accessed, managed, and updated.

In one view, databases can be classified according to types of content: bibliographic, full-text, numeric, and images.

The data in a database is organized the data according to a database model. The model that is most commonly used today is the relational model. Other models such as the hierarchical model and the network model use a more explicit representation of relationships.

Database topics

Architecture

Depending on the intended use, there are a number of database architectures in use. Many databases use a combination of strategies. On-line Transaction Processing systems (OLTP) often use a **row-oriented** datastore architecture, while data-warehouse and other retrieval-focused applications like Google's BigTable, or bibliographic database (library catalogue) systems may use a Column-oriented DBMS architecture.

Document-Oriented, XML, knowledgebases, as well as frame databases and RDF-stores (aka triple-stores), may also use a combination of these architectures in their implementation.

Finally, it should be noted that not all databases have or need a database schema (so called schema-less databases).

Over many years the database industry has been dominated by General Purpose database systems, which offer a wide range of functions that are applicable to many, if not most circumstances in modern data processing. These have been enhanced with extensible datatypes, pioneered in the PostgreSQL project, to allow a very wide range of applications to be developed.

There are also other types of database which cannot be classified as relational databases.

Database management systems

A **Database Management System (DBMS)** is a set of computer programs that controls the creation, maintenance, and the use of the database of an organization and its end users. It allows organizations to place control of organizationwide database development in the hands of Database Administrators (DBAs) and other specialist. DBMSes may use any of a variety of database models, such as the network model or relational model. In large systems, a DBMS allows users and other software to store and retrieve data in a structured way.

A computer database relies on software to organize the storage of data. This software is known as a database management system (DBMS). Database management systems are categorized according to the database model that they support. The model tends to determine the query languages that are available to access the database, the most common of which is SQL. A great deal of the internal engineering of a DBMS, however, is independent of the data model, and is concerned with managing factors such as performance, concurrency, integrity, and recovery from hardware failures. In these areas there are large differences between products.

A Relational Database Management System (RDBMS) implements the features of the relational model outlined above. In this context, Date's "Information Principle" states: "the entire information content of the database is represented in one and only one way. Namely as explicit values in column positions (attributes) and rows in relations (tuples). Therefore, there are no explicit pointers between related tables."

Five Components of DBMS

- **DBMS Engine** accepts logical request from the various other DBMS subsystems, converts them into physical equivalent, and actually accesses the database and data dictionary as they exist on a storage device.
- **Data Definition Subsystem** helps user to create and maintain the data dictionary and define the structure of the files in a database.
- **Data Manipulation Subsystem** helps user to add, change, and delete information in a database and query it for valuable information. Software tools within the data manipulation subsystem are most often the primary interface between user and the

information contained in a database. It allows user to specify its logical information requirements.

- **Application Generation Subsystem** contains facilities to help users to develop transactions-intensive applications. It usually requires that user perform a detailed series of tasks to process a transaction. It facilities easy-to-use data entry screens, programming languages, and interfaces.
- **Data Administration Subsystem** helps users to manage the overall database environment by providing facilities for backup and recovery, security management, query optimization, concurrency control, and change management.

Primary Tasks of DBMS Packages

- **Database Development.** It is used to define and organize the content, relationships, and structure of the data needed to build a database.
- **Database Interrogation.** It can access the data in a database for information retrieval and report generation. End users can selectively retrieve and display information and produce printed reports and documents.
- **Database Maintenance.** It is used to add, delete, update, correct, and protect the data in a database.
- **Application Development.** It is used to develop prototypes of data entry screens, queries, forms, reports, tables, and labels for a prototyped application. Or use 4GL or 4th Generation Language or application generator to develop program codes.

Types of Databases

Operational Database

Operational Databases. These databases store detailed data needed to support the operations of the entire organization. They are also called Subject Area Databases (SADB), Transaction Databases, and Production Databases. These are all examples:

- Customer Databases
- Personal Databases
- Inventory Databases

Analytical Database

Analytical Databases. These databases stores data and information extracted from selected operational and external databases. They consist of summarized data and information most needed by an organizations manager and other end user. They may also be called multidimensional database, Management database, and Information database.

Data Warehouse

Data Warehouse Databases. It stores data from current and previous years that has been extracted from the various operational databases of an organization. It is the central source of data that has been screened, edited, standardized and integrated so that it can be used by managers and other end user professionals throughout an organization

Distributed Database

Distributed Databases. These are databases of local work groups and departments at regional offices, branch offices, manufacturing plants and other work sites. These databases can include segments of both common operational and common user databases, as well as data generated and used only at a user's own site.

End-User Database

End-User Databases. These databases consist of a variety of data files developed by end-users at their workstations. Examples of these are collection of documents in spreadsheets, word processing and even downloaded files.

External Database

External Databases. These databases where access to external, privately owned online databases or data banks is available for a fee to end users and organizations from commercial services. Access to a wealth of information from external database is available for a fee from commercial online services and with or without charge from many sources in the internet.

Hypermedia Databases on the Web

Hypermedia Databases. These are set of interconnected multimedia pages at a web-site. It consists of home page and other hyperlinked pages of multimedia or mixed media such as text, graphic, photographic images, video clips, audio etc.

Navigational database

Navigational databases. Type of database characterized by the fact that objects in it are found primarily by following references from other objects. Traditionally navigational interfaces are procedural, though one could characterize some modern systems like XPath as being simultaneously navigational and declarative.

In-Memory databases

In-Memory databases. It is a database management system that primarily relies on main memory for computer data storage. It is contrasted with database management systems which employ a disk storage mechanism. Main memory databases are faster than disk-optimized databases since the internal optimization algorithms are simpler and execute fewer CPU instructions. Accessing data in memory provides faster and more predictable performance than disk. In applications where response time is critical, such as telecommunications network equipment that operates 9-1-1 emergency systems, main memory databases are often used.

Document-oriented databases

Document-oriented Databases. It is a computer program designed for document-oriented applications. These systems may be implemented as a layer above a relational database or an object database. As opposed to relational databases, document-based databases do not store data in tables with uniform sized fields for each record. Instead, each record is stored as a document that has certain characteristics. Any number of fields of any length can be added to a document. Fields can also contain multiple pieces of data.

Real-time databases

Real-time databases. It is a processing system designed to handle workloads whose state is constantly changing. This differs from traditional databases containing persistent data, mostly unaffected by time. For example, a stock market changes very rapidly and is dynamic. Real-time processing means that a transaction is processed fast enough for the result to come back and be acted on right away. Real-time databases are useful for accounting, banking, law, medical records, multi-media, process control, reservation systems, and scientific data analysis. As computers increase in power and can store more data, they are integrating themselves into our society and are employed in many applications.

Database models

Database model

Post-relational database models

Products offering a more general data model than the relational model are sometimes classified as post-relational. The data model in such products incorporates relations but is not constrained by the Information Principle, which requires that all information is represented by data values in relations.

Some of these extensions to the relational model actually integrate concepts from technologies that pre-date the relational model. For example, they allow representation of a directed graph with trees on the nodes.

Some products implementing such models have been built by extending relational database systems with non-relational features. Others, however, have arrived in much the same place by adding relational features to pre-relational systems. Paradoxically, this allows products that are historically pre-relational, such as PICK and MUMPS, to make a plausible claim to be post-relational in their current architecture.

Object database models

In recent years, the object-oriented paradigm has been applied to database technology, creating a various kinds of new programming model known as object databases. These databases attempt to bring the database world and the application programming world closer together, in particular by ensuring that the database uses the same type system as the application program. This aims to avoid the overhead (sometimes referred to as the *impedance mismatch*) of converting information between its representation in the database (for example as rows in tables) and its representation in the application program (typically as objects). At the same time, object databases attempt to introduce the key ideas of object programming, such as encapsulation and polymorphism, into the world of databases.

A variety of these ways have been tried for storing objects in a database. Some products have approached the problem from the application programming end, by making the objects manipulated by the program persistent. This also typically requires the addition of some kind of query language, since conventional programming languages do not have the ability to find objects based on their information content. Others have attacked the problem from the database end, by defining an object-oriented data model for the database, and defining a database programming language that allows full programming capabilities as well as traditional query facilities.

Database storage structures

Database storage structures

Relational database tables/indexes are typically stored in memory or on hard disk in one of many forms, ordered/unordered flat files, ISAM, heaps, hash buckets or B+ trees. These have various advantages and disadvantages discussed further in the main article on this topic. The most commonly used are B+ trees and ISAM.

Object databases use a range of storage mechanisms. Some use virtual memory mapped files to make the native language (C++, Java etc.) objects persistent. This can be highly efficient but it can make multi-language access more difficult. Others break the objects down into fixed and varying length components that are then clustered tightly together in fixed sized blocks on disk and reassembled into the appropriate format either for the client or in the client address space. Another popular technique is to store the objects in tuples, much like a relational database, which the database server then reassembles for the client.

Other important design choices relate to the clustering of data by category (such as grouping data by month, or location), creating pre-computed views known as materialized views, partitioning data by range or hash. Memory management and storage topology can be important design choices for database designers as well. Just as normalization is used to reduce storage requirements and improve the extensibility of the database, conversely denormalization is often used to reduce join complexity and reduce execution time for queries.

Indexing

All of these databases can take advantage of indexing to increase their speed. This technology has advanced tremendously since its early uses in the 1960s and 1970s. The most common kind of index is a sorted list of the contents of some particular table column, with pointers to the row associated with the value. An index allows a set of table rows matching some criterion to be located quickly. Typically, indexes are also stored in the various forms of data-structure mentioned above (such as B-trees, hashes, and linked lists). Usually, a specific technique is chosen by the database designer to increase efficiency in the particular case of the type of index required.

Most relational DBMS's and some object DBMSs have the advantage that indexes can be created or dropped without changing existing applications making use of it. The database chooses between many different strategies based on which one it estimates will run the fastest. In other words, indexes are transparent to the application or end-user querying the database; while they affect performance, any SQL command will run with or without index to compute the result of an SQL statement. The RDBMS will produce a plan of how to execute the query, which is generated by analyzing the run times of the different algorithms and selecting the quickest. Some of the key algorithms that deal with joins are nested loop join, sort-merge join and hash join. Which of these is chosen depends on whether an index exists, what type it is, and its cardinality.

An index speeds up access to data, but it has disadvantages as well. First, every index increases the amount of storage on the hard drive necessary for the database file, and second, the index must be updated each time the data are altered, and this costs time. (Thus an index saves time in the reading of data, but it costs time in entering and altering data. It thus depends on the use to which the data are to be put whether an index is on the whole a net plus or minus in the quest for efficiency.)

A special case of an index is a primary index, or primary key, which is distinguished in that the primary index must ensure a unique reference to a record. Often, for this purpose one simply uses a running index number (ID number). Primary indexes play a significant role in relational databases, and they can speed up access to data considerably.

Transactions and concurrency

In addition to their data model, most practical databases ("transactional databases") attempt to enforce a database transaction. Ideally, the database software should enforce the ACID rules, summarized here:

- Atomicity: Either all the tasks in a transaction must be done, or none of them. The transaction must be completed, or else it must be undone (rolled back).

- Consistency: Every transaction must preserve the integrity constraints — the declared consistency rules — of the database. It cannot place the data in a contradictory state.
- Isolation: Two simultaneous transactions cannot interfere with one another. Intermediate results within a transaction are not visible to other transactions.
- Durability: Completed transactions cannot be aborted later or their results discarded. They must persist through (for instance) restarts of the DBMS after crashes

In practice, many DBMSs allow most of these rules to be selectively relaxed for better performance.

Concurrency control is a method used to ensure that transactions are executed in a safe manner and follow the ACID rules. The DBMS must be able to ensure that only serializable, recoverable schedules are allowed, and that no actions of committed transactions are lost while undoing aborted transactions.

Replication

Replication of databases is closely related to transactions. If a database can log its individual actions, it is possible to create a duplicate of the data in real time. The duplicate can be used to improve performance or availability of the whole database system. Common replication concepts include:

- Master/Slave Replication: All write requests are performed on the master and then replicated to the slaves
- Quorum: The result of Read and Write requests are calculated by querying a "majority" of replicas.
- Multimaster: Two or more replicas sync each other via a transaction identifier.

Parallel synchronous replication of databases enables transactions to be replicated on multiple servers simultaneously, which provides a method for backup and security as well as data availability.

Security

Database security denotes the system, processes, and procedures that protect a database from unintended activity.

Security is usually enforced through **access control**, **auditing**, and **encryption**.

- Access control ensures and restricts who can connect and what can be done to the database.
- Auditing logs what action or change has been performed, when and by whom.
- Encryption: Since security has become a major issue in recent years, many commercial database vendors provide built-in encryption mechanisms. Data is

encoded natively into the tables and deciphered "on the fly" when a query comes in. Connections can also be secured and encrypted if required using DSA, MD5, SSL or legacy encryption standard.

Enforcing security is one of the major tasks of the DBA.

In the United Kingdom, legislation protecting the public from unauthorized disclosure of personal information held on databases falls under the Office of the Information Commissioner. United Kingdom based organizations holding personal data in electronic format (databases for example) are required to register with the Data Commissioner.

Locking

Locking is how the database handles multiple concurrent operations. This is how concurrency and some form of basic integrity is managed within the database system. Such locks can be applied on a row level, or on other levels like page (a basic data block), extent (multiple array of pages) or even an entire table. This helps maintain the integrity of the data by ensuring that only one process at a time can modify the **same** data.

In basic filesystem files or folders, only one lock at a time can be set, restricting the usage to one process only. Databases, on the other hand, can set and hold mutiple locks at the same time on the different level of the physical data structure. How locks are set, last is determined by the database engine locking scheme based on the submitted SQL or transactions by the users. Generally speaking, no activity on the database should be translated by no or very light locking.

For most DBMS systems existing on the market, locks are generally **shared** or **exclusive**. Exclusive locks mean that no other lock can acquire the current data object as long as the exclusive lock lasts. Exclusive locks are usually set while the database needs to change data, like during an UPDATE or DELETE operation.

Shared locks can take ownership one from the other of the current data structure. Shared locks are usually used while the database is reading data, during a SELECT operation. The number, nature of locks and time the lock holds a data block can have a huge impact on the database performances. Bad locking can lead to disastrous performance response (usually the result of poor SQL requests, or inadequate database physical structure)

Default locking behavior is enforced by the **isolation level** of the data server. Changing the isolation level will affect how shared or exclusive locks must be set on the data for the entire database system. Default isolation is generally 1, where data can not be read while it is modified, forbidding to return "ghost data" to end user.

At some point intensive or inappropriate exclusive locking, can lead to the "dead lock" situation between two locks. Where none of the locks can be released because they try to acquire resources mutually from each other. The Database has a fail safe mechanism and will automatically "sacrifice" one of the locks releasing the resource. Doing so processes or transactions involved in the "dead lock" will be rolled back.

Databases can also be locked for other reasons, like access restrictions for given levels of user. Some databases are also locked for routine database maintenance, which prevents changes being made during the maintenance. See "Locking tables and databases" (section in some documentation / explanation from IBM) for more detail.) However, many modern databases don't lock the database during routine maintenance. e.g. "Routine Database Maintenance" for PostgreSQL.

Applications of databases

Databases are used in many applications, spanning virtually the entire range of computer software. Databases are the preferred method of storage for large multiuser applications, where coordination between many users is needed. Even individual users find them convenient, and many electronic mail programs and personal organizers are based on standard database technology. Software database drivers are available for most database platforms so that application software can use a common Application Programming Interface to retrieve the information stored in a database. Two commonly used database APIs are JDBC and ODBC.

Examples of use

The largest statistical database maintained by the central authority of statistics in Denmark is called StatBank. The very large database in English is available free-of-charge for all users on the internet. It is updated every day 9.30 am (CET) and contains all new statistics in a very detailed form. The statistics can be presented as cross-tables, diagrams or maps. There are about 2 million hits every year (2006). The output can be transferred to other programs for further compilation.

Digital electronics

Digital electronics are electronics systems that use digital signals. Digital electronics are representations of Boolean algebra (also see truth tables) and are used in computers, mobile phones, and other consumer products. In a digital circuit, a signal is represented in discrete states or logic levels. The advantages of digital techniques stem from the fact 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, traditionally only two states, '1' and '0' are used though digital systems are not limited to this.

Digital electronics or any **digital circuit** are usually made from large assemblies of logic gates, simple electronic representations of Boolean logic functions.

To most electronic engineers, the terms "digital circuit", "digital system" and "logic" are interchangeable in the context of digital circuits.

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 charactersitics 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.

The sensed world is analog, and signals from this world are analog quantities. For example, light, temperature, sound, electrical conductivity, electric and magnetic fields are analog. 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.

Embedded software designs that employ Immunity Aware Programming, such as the practice of filling unused program memory with interrupt instructions that point to an error recovery routine. This helps guard against failures that corrupt the microcontroller's instruction pointer which could otherwise cause random code to be executed.

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.

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. Each logic gate represents a function of boolean logic. A logic gate is an arrangement of electrically controlled switches.

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 (See below for more information).

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 (See below for more information).

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 "combinatorial systems" and "sequential systems." A combinatorial 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 combinatorial 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 divide it into a piece of combinatorial 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 combinatorial logic, and feeds it back as an unchanging input to the combinatorial part of the state machine. The fastest rate of the clock is set by the most time-consuming logic calculation in the combinatorial 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 combinatorial 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, 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 simply runs at the maximum speed permitted by the propagation rates of the logic gates from which it is constructed. Building an asynchronous circuit using faster parts implicitly makes the circuit "go" faster.

More generally, 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 combinatorial 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 combinatorial 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 relatively independent 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 combinatorial part 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 (see next section).

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

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 of nearly a trillion (1×10^{12}) 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. 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.

Another contender was emitter coupled logic. This is very fast but uses a lot of power. It's now used mostly in radio-frequency circuits.

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

Non-electronic logic

unconventional computing

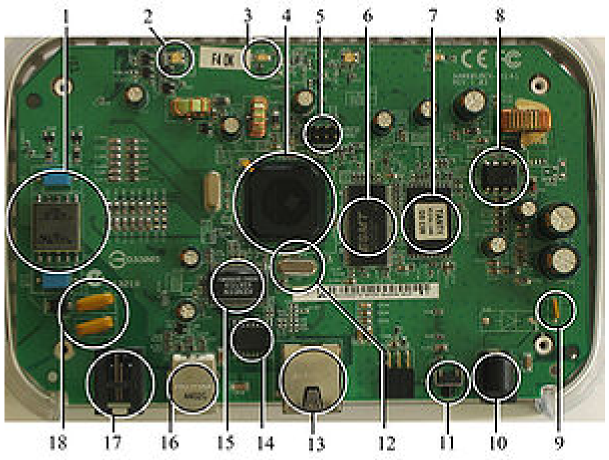
It is possible to construct non-electronic digital mechanisms. In principle, any technology capable of representing discrete states and representing logic operations could be used to build mechanical logic. MIT students Erlyne Gee, Edward Hardebeck, Danny Hillis (co-author of The Connection Machine), 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 supposed to be in the Houston Museum of Natural 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.

Embedded system



Picture of the internals of a Netgear ADSL modem/router. A modern example of an embedded system. Labelled parts include a microprocessor (4), RAM (6), and flash memory (7).

An **embedded system** is a computer system designed to perform one or a few dedicated functions, often with real-time computing constraints. It is usually *embedded* as part of a complete device including hardware and mechanical parts. In contrast, a general-purpose computer, such as a personal computer, can do many different tasks depending on programming. Embedded systems control many of the common devices in use today.

Since the embedded system is dedicated to specific tasks, design engineers can optimize it, reducing the size and cost of the product, or increasing the reliability and performance. Some embedded systems are mass-produced, benefiting from economies of scale.

Physically, embedded systems range from portable devices such as digital watches and *MP3* players, to large stationary installations like traffic lights, factory controllers, or the systems controlling nuclear power plants. Complexity varies from low, with a single microcontroller chip, to very high with multiple units, peripherals and networks mounted inside a large chassis or enclosure.

In general, "embedded system" is not an exactly defined term, as many systems have some element of programmability. For example, Handheld computers share some elements with embedded systems — such as the operating systems and microprocessors which power them — but are not truly embedded systems, because they allow different applications to be loaded and peripherals to be connected.

Examples of embedded systems

PC Engines' ALIX.1C Mini-ITX embedded board with an x86 AMD Geode LX 800 together with Compact Flash, miniPCI and PCI slots, 22-pin IDE interface, audio, USB and 256MB RAM

An embedded RouterBoard 112 with U.FL-RSMA pigtail and R52 miniPCI Wi-Fi card widely used by wireless Internet service providers (WISPs) in the Czech Republic.

Embedded systems span all aspects of modern life and there are many examples of their use.

Telecommunications systems employ numerous embedded systems from telephone switches for the network to mobile phones at the end-user. Computer networking uses dedicated routers and network bridges to route data.

Consumer electronics include personal digital assistants (PDAs), mp3 players, mobile phones, videogame consoles, digital cameras, DVD players, GPS receivers, and printers. Many household appliances, such as microwave ovens, washing machines and dishwashers, are including embedded systems to provide flexibility, efficiency and features. Advanced HVAC systems use networked thermostats to more accurately and efficiently control temperature that can change by time of day and season. Home automation uses wired- and wireless-networking that can be used to control lights, climate, security, audio/visual, surveillance, etc., all of which use embedded devices for sensing and controlling.

Transportation systems from flight to automobiles increasingly use embedded systems. New airplanes contain advanced avionics such as inertial guidance systems and GPS receivers that also have considerable safety requirements. Various electric motors — brushless DC motors, induction motors and DC motors — are using electric/electronic motor controllers. Automobiles, electric vehicles, and hybrid vehicles are increasingly using embedded systems to maximize efficiency and reduce pollution. Other automotive safety systems such as anti-lock braking system (ABS), Electronic Stability Control (ESC/ESP), traction control (TCS) and automatic four-wheel drive.

Medical equipment is continuing to advance with more embedded systems for vital signs monitoring, electronic stethoscopes for amplifying sounds, and various medical imaging (PET, SPECT, CT, MRI) for non-invasive internal inspections.

In addition to commonly described embedded systems based on small computers, a new class of miniature wireless devices called motes are quickly gaining popularity as the field of wireless sensor networking rises. Wireless sensor networking, WSN, makes use of miniaturization made possible by advanced IC design to couple full wireless subsystems to sophisticated sensor, enabling people and companies to measure a myriad of things in the physical world and act on this information through IT monitoring and control systems. These motes are completely self contained, and will typically run off a battery source for many years before the batteries need to be changed or charged.

History

In the earliest years of computers in the 1930–40s, computers were sometimes dedicated to a single task, but were far too large and expensive for most kinds of tasks performed by embedded computers of today. Over time however, the concept of programmable controllers evolved from traditional electromechanical sequencers, via solid state devices, to the use of computer technology.

One of the first recognizably modern embedded systems was the Apollo Guidance Computer, developed by Charles Stark Draper at the MIT Instrumentation Laboratory. At the project's inception, the Apollo guidance computer was considered the riskiest item in the Apollo project as it employed the then newly developed monolithic integrated circuits to reduce the size and weight. An early mass-produced embedded system was the Autonetics D-17 guidance computer for the Minuteman missile, released in 1961. It was built from transistor logic and had a hard disk for main memory. When the Minuteman II went into production in 1966, the D-17 was replaced with a new computer that was the first high-volume use of integrated circuits. This program alone reduced prices on quad nand gate ICs from $1000/each to $3/each, permitting their use in commercial products.

Since these early applications in the 1960s, embedded systems have come down in price and there has been a dramatic rise in processing power and functionality. The first microprocessor for example, the Intel 4004, was designed for calculators and other small systems but still required many external memory and support chips. In 1978 National Engineering Manufacturers Association released a "standard" for programmable microcontrollers, including almost any computer-based controllers, such as single board computers, numerical, and event-based controllers.

As the cost of microprocessors and microcontrollers fell it became feasible to replace expensive knob-based analog components such as potentiometers and variable capacitors with up/down buttons or knobs read out by a microprocessor even in some consumer products. By the mid-1980s, most of the common previously external system components had been integrated into the same chip as the processor and this modern form of the microcontroller allowed an even more widespread use, which by the end of the decade were the norm rather than the exception for almost all electronics devices.

The integration of microcontrollers has further increased the applications for which embedded systems are used into areas where traditionally a computer would not have been considered. A general purpose and comparatively low-cost microcontroller may often be programmed to fulfill the same role as a large number of separate components. Although in this context an embedded system is usually more complex than a traditional solution, most of the complexity is contained within the microcontroller itself. Very few additional components may be needed and most of the design effort is in the software. The intangible nature of software makes it much easier to prototype and test new revisions compared with the design and construction of a new circuit not using an embedded processor.

Characteristics

Soekris net4801, an embedded system targeted at network applications.

1. Embedded systems are designed to do some specific task, rather than be a general-purpose computer for multiple tasks. Some also have real-time performance constraints that must be met, for reasons such as safety and usability; others may have low or no performance requirements, allowing the system hardware to be simplified to reduce costs.
2. Embedded systems are not always standalone devices. Many embedded systems consist of small, computerized parts within a larger device that serves a more general purpose. For example, the Gibson Robot Guitar features an embedded system for tuning the strings, but the overall purpose of the Robot Guitar is, of course, to play music. Similarly, an embedded system in an automobile provides a specific function as a subsystem of the car itself.

3. The program instructions written for embedded systems are referred to as firmware, and are stored in read-only memory or Flash memory chips. They run with limited computer hardware resources: little memory, small or non-existent keyboard and/or screen.

User interfaces

Embedded system text user interface using MicroVGA

Embedded systems range from no user interface at all — dedicated only to one task — to complex graphical user interfaces that resemble modern computer desktop operating systems.

Simple systems

Simple embedded devices use buttons, LEDs, and small character or digit-only displays, often with a simple menu system.

In more complex systems

A full graphical screen, with touch sensing or screen-edge buttons provides flexibility while minimising space used: the meaning of the buttons can change with the screen, and selection involves the natural behavior of pointing at what's desired.

Handheld systems often have a screen with a "joystick button" for a pointing device.

Many systems have "maintenance" or "test" interfaces that provide a menu or command system via an RS-232 interface. This avoids the cost of a display, but gives a lot of control. Most consumers cannot assemble the required cables, however.

The rise of the World Wide Web has given embedded designers another quite different option: providing a web page interface over a network connection. This avoids the cost of a sophisticated display, yet provides complex input and display capabilities when needed, on another computer. This is successful for remote, permanently installed equipment such as Pan-Tilt-Zoom cameras and network routers.

CPU platforms

Embedded processors can be broken into two broad categories: ordinary microprocessors (μP) and microcontrollers (μC), which have many more peripherals on chip, reducing cost and size. Contrasting to the personal computer and server markets, a fairly large number of basic CPU architectures are used; there are Von Neumann as well as various degrees of Harvard architectures, RISC as well as non-RISC and VLIW; word lengths vary from 4-bit to 64-bits and beyond (mainly in DSP processors) although the most typical remain 8/16-bit. Most architectures come in a large number of different variants and shapes, many of which are also manufactured by several different companies.

Ready made computer boards

PC/104 and PC/104+ are examples of available *ready made* computer boards intended for small, low-volume embedded and ruggedized systems. These often use DOS, Linux, NetBSD, or an embedded real-time operating system such as MicroC/OS-II, QNX or VxWorks.

In certain applications, where small size is not a primary concern, the components used may be compatible with those used in general purpose computers. Boards such as the VIA EPIA range help to bridge the gap by being PC-compatible but highly integrated, physically smaller or have other attributes making them attractive to embedded engineers. The advantage of this approach is that low-cost commodity components may be used along with the same software development tools used for general software development. Systems built in this way are still regarded as embedded since they are integrated into larger devices and fulfill a single role. Examples of devices that may adopt this approach are ATMs and arcade machines.and which contain code specific to the application.

ASIC and FPGA solutions

A common configuration for very-high-volume embedded systems is the system on a chip (SoC) which contains a complete system consisting of multiple processors, multipliers, caches and interfaces on a single chip. SoCs can be implemented as an application-specific integrated circuit (ASIC) or using a field-programmable gate array (FPGA).

Peripherals

Embedded Systems talk with the outside world via peripherals, such as:

- Serial Communication Interfaces (SCI): RS-232, RS-422, RS-485 etc
- Synchronous Serial Communication Interface: I2C, SPI, SSC and ESSI (Enhanced Synchronous Serial Interface)
- Universal Serial Bus (USB)
- Multi Media Cards (SD Cards, Compact Flash etc)
- Networks: Ethernet, Controller Area Network, LonWorks, etc
- Timers: PLL(s), Capture/Compare and Time Processing Units
- Discrete IO: aka General Purpose Input/Output (GPIO)

- Analog to Digital/Digital to Analog (ADC/DAC)
- Debugging: JTAG, ISP, ICSP, BDM Port, ...

Tools

As for other software, embedded system designers use compilers, assemblers, and debuggers to develop embedded system software. However, they may also use some more specific tools:

- In circuit debuggers or emulators (see next section).
- Utilities to add a checksum or CRC to a program, so the embedded system can check if the program is valid.
- For systems using digital signal processing, developers may use a math workbench such as Scilab / Scicos, MATLAB / Simulink, EICASLAB, MathCad, or Mathematica to simulate the mathematics. They might also use libraries for both the host and target which eliminates developing DSP routines as done in DSPnano RTOS and Unison Operating System.
- Custom compilers and linkers may be used to improve optimisation for the particular hardware.
- An embedded system may have its own special language or design tool, or add enhancements to an existing language such as Forth or Basic.
- Another alternative is to add a Real-time operating system or Embedded operating system, which may have DSP capabilities like DSPnano RTOS.

Software tools can come from several sources:

- Software companies that specialize in the embedded market
- Ported from the GNU software development tools
- Sometimes, development tools for a personal computer can be used if the embedded processor is a close relative to a common PC processor

As the complexity of embedded systems grows, higher level tools and operating systems are migrating into machinery where it makes sense. For example, cellphones, personal digital assistants and other consumer computers often need significant software that is purchased or provided by a person other than the manufacturer of the electronics. In these systems, an open programming environment such as Linux, NetBSD, OSGi or Embedded Java is required so that the third-party software provider can sell to a large market.

Debugging

Embedded Debugging may be performed at different levels, depending on the facilities available. From simplest to most sophisticated they can be roughly grouped into the following areas:

- Interactive resident debugging, using the simple shell provided by the embedded operating system (e.g. Forth and Basic)

- External debugging using logging or serial port output to trace operation using either a monitor in flash or using a debug server like the Remedy Debugger which even works for heterogeneous multicore systems.
- An in-circuit debugger (ICD), a hardware device that connects to the microprocessor via a JTAG or NEXUS interface. This allows the operation of the microprocessor to be controlled externally, but is typically restricted to specific debugging capabilities in the processor.
- An in-circuit emulator replaces the microprocessor with a simulated equivalent, providing full control over all aspects of the microprocessor.
- A complete emulator provides a simulation of all aspects of the hardware, allowing all of it to be controlled and modified, and allowing debugging on a normal PC.

Unless restricted to external debugging, the programmer can typically load and run software through the tools, view the code running in the processor, and start or stop its operation. The view of the code may be as assembly code or source-code.

Because an embedded system is often composed of a wide variety of elements, the debugging strategy may vary. For instance, debugging a software- (and microprocessor-) centric embedded system is different from debugging an embedded system where most of the processing is performed by peripherals (DSP, FPGA, co-processor). An increasing number of embedded systems today use more than one single processor core. A common problem with multi-core development is the proper synchronization of software execution. In such a case, the embedded system design may wish to check the data traffic on the busses between the processor cores, which requires very low-level debugging, at signal/bus level, with a logic analyzer, for instance.

Reliability

Embedded systems often reside in machines that are expected to run continuously for years without errors, and in some cases recover by themselves if an error occurs. Therefore the software is usually developed and tested more carefully than that for personal computers, and unreliable mechanical moving parts such as disk drives, switches or buttons are avoided.

Specific reliability issues may include:

1. The system cannot safely be shut down for repair, or it is too inaccessible to repair. Examples include space systems, undersea cables, navigational beacons, bore-hole systems, and automobiles.
2. The system must be kept running for safety reasons. "Limp modes" are less tolerable. Often backups are selected by an operator. Examples include aircraft navigation, reactor control systems, safety-critical chemical factory controls, train signals, engines on single-engine aircraft.

3. The system will lose large amounts of money when shut down: Telephone switches, factory controls, bridge and elevator controls, funds transfer and market making, automated sales and service.

A variety of techniques are used, sometimes in combination, to recover from errors—both software bugs such as memory leaks, and also soft errors in the hardware:

- watchdog timer that resets the computer unless the software periodically notifies the watchdog
- subsystems with redundant spares that can be switched over to
- software "limp modes" that provide partial function
- Designing with a Trusted Computing Base (TCB) architecture ensures a highly secure & reliable system environment
- An Embedded Hypervisor is able to provide secure encapsulation for any subsystem component, so that a compromised software component cannot interfere with other subsystems, or privileged-level system software. This encapsulation keeps faults from propagating from one subsystem to another, improving reliability. This may also allow a subsystem to be automatically shut down and restarted on fault detection.
- Immunity Aware Programming

High vs Low Volume

For high volume systems such as portable music players or mobile phones, minimizing cost is usually the primary design consideration. Engineers typically select hardware that is just “good enough” to implement the necessary functions.

For low-volume or prototype embedded systems, general purpose computers may be adapted by limiting the programs or by replacing the operating system with a real-time operating system.

Embedded software architectures

There are several different types of software architecture in common use.

Simple control loop

In this design, the software simply has a loop. The loop calls subroutines, each of which manages a part of the hardware or software.

Interrupt controlled system

Some embedded systems are predominantly interrupt controlled. This means that tasks performed by the system are triggered by different kinds of events. An interrupt could be generated for example by a timer in a predefined frequency, or by a serial port controller receiving a byte.

These kinds of systems are used if event handlers need low latency and the event handlers are short and simple.

Usually these kinds of systems run a simple task in a main loop also, but this task is not very sensitive to unexpected delays.

Sometimes the interrupt handler will add longer tasks to a queue structure. Later, after the interrupt handler has finished, these tasks are executed by the main loop. This method brings the system close to a multitasking kernel with discrete processes.

Cooperative multitasking

A nonpreemptive multitasking system is very similar to the simple control loop scheme, except that the loop is hidden in an API. The programmer defines a series of tasks, and each task gets its own environment to “run” in. When a task is idle, it calls an idle routine, usually called “pause”, “wait”, “yield”, “nop” (stands for *no operation*), etc.

The advantages and disadvantages are very similar to the control loop, except that adding new software is easier, by simply writing a new task, or adding to the queue-interpreter.

Preemptive multitasking or multi-threading

In this type of system, a low-level piece of code switches between tasks or threads based on a timer (connected to an interrupt). This is the level at which the system is generally considered to have an "operating system" kernel. Depending on how much functionality is required, it introduces more or less of the complexities of managing multiple tasks running conceptually in parallel.

As any code can potentially damage the data of another task (except in larger systems using an MMU) programs must be carefully designed and tested, and access to shared data must be controlled by some synchronization strategy, such as message queues, semaphores or a non-blocking synchronization scheme.

Because of these complexities, it is common for organizations to buy a real-time operating system, allowing the application programmers to concentrate on device functionality rather than operating system services, at least for large systems; smaller systems often cannot afford the overhead associated with a *generic* real time system, due to limitations regarding memory size, performance, and/or battery life.

Microkernels and exokernels

A microkernel is a logical step up from a real-time OS. The usual arrangement is that the operating system kernel allocates memory and switches the CPU to different threads of execution. User mode processes implement major functions such as file systems, network interfaces, etc.

In general, microkernels succeed when the task switching and intertask communication is fast, and fail when they are slow.

Exokernels communicate efficiently by normal subroutine calls. The hardware, and all the software in the system are available to, and extensible by application programmers.

Monolithic kernels

In this case, a relatively large kernel with sophisticated capabilities is adapted to suit an embedded environment. This gives programmers an environment similar to a desktop operating system like Linux or Microsoft Windows, and is therefore very productive for development; on the downside, it requires considerably more hardware resources, is often more expensive, and because of the complexity of these kernels can be less predictable and reliable.

Common examples of embedded monolithic kernels are Embedded Linux and Windows CE.

Despite the increased cost in hardware, this type of embedded system is increasing in popularity, especially on the more powerful embedded devices such as Wireless Routers and GPS Navigation Systems. Here are some of the reasons:

- Ports to common embedded chip sets are available.
- They permit re-use of publicly available code for Device Drivers, Web Servers, Firewalls, and other code.
- Development systems can start out with broad feature-sets, and then the distribution can be configured to exclude unneeded functionality, and save the expense of the memory that it would consume.
- Many engineers believe that running application code in user mode is more reliable, easier to debug and that therefore the development process is easier and the code more portable.
- Many embedded systems lack the tight real time requirements of a control system. A system such as Embedded Linux has fast enough response for many applications.
- Features requiring faster response than can be guaranteed can often be placed in hardware.
- Many RTOS systems have a per-unit cost. When used on a product that is or will become a commodity, that cost is significant.

Exotic custom operating systems

A small fraction of embedded systems require safe, timely, reliable or efficient behavior unobtainable with the one of the above architectures. In this case an organization builds a system to suit. In some cases, the system may be partitioned into a "mechanism controller" using special techniques, and a "display controller" with a conventional operating system. A communication system passes data between the two.

Additional software components

In addition to the core operating system, many embedded systems have additional upper-layer software components. These components consist of networking protocol stacks like CAN, TCP/IP, FTP, HTTP, and HTTPS, and also included storage capabilities like FAT and Flash memory management systems. If the embedded devices has audio and video capabilities, then the appropriate drivers and codecs will be present in the system. In the case of the monolithic kernels, many of these software layers are included. In the RTOS category, the availability of the additional software components depends upon the commercial offering.