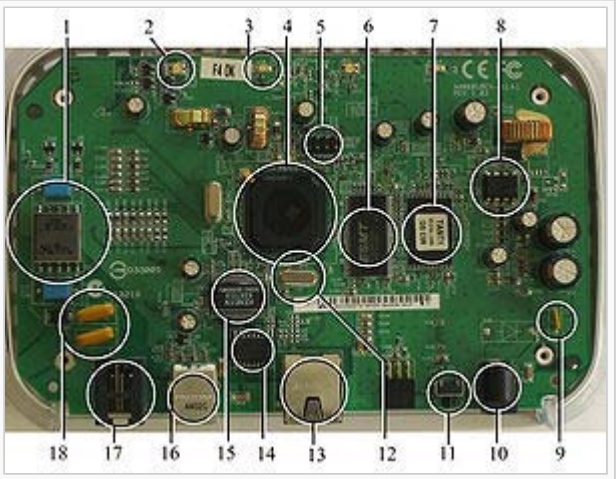
Embedded system

From Wikipedia, the free encyclopedia

An **embedded system** is a special-purpose computer system designed to perform one or a few dedicated functions,[1] 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.



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).

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.

Contents

Examples of embedded systems

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

PC Engines' ALIX.1C Mini-ITX embedded board with AMD Geode LX 800 together with Compact Flash, miniPCI and PCI slots, 44-pin IDE interface 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.

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, 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 miniturization 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.[2] 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 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.

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.

A long but still not exhaustive list of common architectures are: 65816, 65C02, 68HC08, 68HC11, 68k, 8051, ARM, AVR, AVR32, Blackfin, C167, Coldfire, COP8, eZ8, eZ80, FR-V, H8, HT48, M16C, M32C, MIPS, MSP430, PIC, PowerPC, R8C, SHARC, ST6, SuperH, TLCS-47, TLCS-870, TLCS-900, Tricore, V850, x86, XE8000, Z80, etc.

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.

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
- Universal Serial Bus (USB)
- 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, 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
- 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. Then, 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 consists of networking protocol stacks like 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.

See also

- Communications server
- DSP
- Electronic Control Unit

- Embedded operating systems
- Embedded software
- System on module
- System on a chip
- Firmware
- Information appliance
- Microprocessor
- Programming languages
- Real-time operating system
- Software engineering
- Ubiquitous computing
- Cyber-physical system

References

1. ^ Michael Barr. "Embedded Systems Glossary (http://www.netrino.com/Embedded-Systems/Glossary)". *Netrino Technical Library*. Retrieved on 2007-04-21.
2. ^ Embedded.com - Under the Hood: Robot Guitar embeds autotuning (http://www.embedded.com/underthehood/207401418)

Further reading

- John Catsoulis, Designing Embedded Hardware (http://www.oreilly.com/catalog/dbhardware2/), O'Reilly, May 2005, ISBN 0-596-00755-8.
- Anoop MS, Security needs in embedded systems (http://www.infosecwriters.com/text_resources/pdf/Anoopms_Embedded_Systems.pdf), Tata Elxsi, India, May 2008.

External links

- Embedded System News (http://embeddedsystemnews.com/)