WIKIPEDIA

Embedded system

An *embedded system* on a plug-in card with processor, memory, power supply, and external interfaces

An **embedded system** is a computer system—a combination of a computer processor, computer memory, and input/output peripheral devices—that has a dedicated function within a larger mechanical or electronic system.[1][2] It is *embedded* as part of a complete device often including electrical or electronic hardware and mechanical parts. Because an embedded system typically controls physical operations of the machine that it is embedded within, it often has real-time computing constraints. Embedded systems control many devices in common use today.[3] In 2009 it was estimated that ninety-eight percent of all microprocessors manufactured were used in embedded systems.[4]

Modern embedded systems are often based on microcontrollers (i.e. microprocessors with integrated memory and peripheral interfaces), but ordinary microprocessors (using external chips for memory and peripheral interface circuits) are also common, especially in more complex systems. In either case, the processor(s) used may be types ranging from general purpose to those specialized in a certain class of computations, or even custom designed for the application at hand. A common standard class of dedicated processors is the digital signal processor (DSP).

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

Embedded systems range in size from portable personal devices such as digital watches and MP3 players to bigger machines like home appliances, industrial assembly lines, robots, transport vehicles, traffic light controllers, and medical imaging systems. Often they constitute subsystems of other machines like avionics in aircraft and spacecraft. Large installations like factories, pipelines and electrical grids rely on multiple embedded systems networked together. Generalized through software customization, embedded systems such as programmable logic controllers frequently comprise their functional units.

Embedded systems range from those low in complexity, with a single microcontroller chip, to very high with multiple units, peripherals and networks, which may reside in equipment racks or across large geographical areas connected via long-distance communications lines.

Contents

History

Background

The origins of the microprocessor and the microcontroller can be traced back to the MOS integrated circuit, which is an integrated circuit chip fabricated from MOSFETs (metal-oxide-semiconductor field-effect transistors) and was developed in the early 1960s. By 1964, MOS chips had reached higher transistor density and lower manufacturing costs than bipolar chips. MOS chips further increased in complexity at a rate predicted by Moore's law, leading to large-scale integration (LSI) with hundreds of transistors on a single MOS chip by the late 1960s. The application of MOS LSI chips to computing was the basis for the first microprocessors, as engineers began recognizing that a complete computer processor system could be contained on several MOS LSI chips.[5]

The first multi-chip microprocessors, the Four-Phase Systems AL1 in 1969 and the Garrett AiResearch MP944 in 1970, were developed with multiple MOS LSI chips. The first single-chip microprocessor was the Intel 4004, released in 1971. It was developed by Federico Faggin, using his silicon-gate MOS technology, along with Intel engineers Marcian Hoff and Stan Mazor, and Busicom engineer Masatoshi Shima.[6]

Development

One of the first recognizably modern embedded systems was the Apollo Guidance Computer, developed ca. 1965 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 computer's size and weight.

An early mass-produced embedded system was the Autonetics D-17 guidance computer for the Minuteman missile, released in 1961. When the Minuteman II went into production in 1966, the D-17 was replaced with a new computer that represented the first high-volume use of integrated circuits.

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. An early microprocessor, the Intel 4004 (released in 1971), was designed for calculators and other small systems but still required external memory and support chips. By the early 1980s, memory, input and output system components had been integrated into the same chip as the processor forming a microcontroller. Microcontrollers find applications where a general-purpose computer would be too costly. As the cost of microprocessors and microcontrollers fell the prevalence of embedded systems increased.

Today, a comparatively low-cost microcontroller may be programmed to fulfill the same role as a large number of separate components. With microcontrollers, it became feasible to replace, even in consumer products, expensive knob-based analog components such as potentiometers and variable capacitors with up/down buttons or knobs read out by a microprocessor. 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. Software prototype and test can be quicker compared with the design and construction of a new circuit not using an embedded processor.

Applications

Embedded systems are commonly found in consumer, industrial, automotive, home appliances, medical, telecommunication, commercial, and aerospace and military applications.

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

Consumer electronics include MP3 players, television sets, mobile phones, video game consoles, digital cameras, GPS receivers, and printers. Household appliances, such as microwave ovens, washing machines and dishwashers, include 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.

Embedded Computer Sub-Assembly for Accupoll Electronic Voting Machine[7]

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. Spacecraft rely on avionics systems for trajectory correction. Various electric motors — brushless DC motors, induction motors and DC motors — use electronic motor controllers. Automobiles, electric vehicles, and hybrid vehicles increasingly use embedded systems to maximize efficiency and reduce pollution. Other automotive safety systems using embedded systems include anti-lock braking system (ABS), Electronic Stability Control (ESC/ESP), traction control (TCS) and automatic four-wheel drive.

Medical equipment uses embedded systems for monitoring, and various medical imaging (PET, Single-photon emission computed tomography (SPECT), CT, and MRI) for non-invasive internal inspections. Embedded systems within medical equipment are often powered by industrial computers.[8]

Embedded systems are used for safety-critical systems in aerospace and defense industries. Unless connected to wired or wireless networks via on-chip 3G cellular or other methods for IoT monitoring and control purposes, these systems can be isolated from hacking and thus be more secure. For fire safety, the systems can be designed to have a greater ability to handle higher temperatures and continue to operate. In dealing with security, the embedded systems can be self-sufficient and be able to deal with cut electrical and communication systems.

Miniature wireless devices called motes are networked wireless sensors. Wireless sensor networking makes use of miniaturization made possible by advanced IC design to couple full wireless subsystems to sophisticated sensors, enabling people and companies to measure a myriad of things in the physical world and act on this information through monitoring and control systems. These motes are completely self-contained and will typically run off a battery source for years before the batteries need to be changed or charged.

Characteristics

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.

Embedded systems are not always standalone devices. Many embedded systems consist of small 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.[9] Similarly, an embedded system in an automobile provides a specific function as a subsystem of the car itself.

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 or screen.

e-con Systems eSOM270 & eSOM300 Computer on Modules

User interfaces

Embedded systems range from no user interface at all, in systems dedicated only to one task, to complex graphical user interfaces that resemble modern computer desktop operating systems. Simple embedded devices use buttons, LEDs, graphic or character LCDs (HD44780 LCD for example) with a simple menu system. More sophisticated devices that use a graphical screen with touch sensing or screen-edge soft keys provide flexibility while minimizing space used: the meaning of the buttons can change with the screen, and selection involves the natural behavior of pointing at what is desired.

Embedded system text user interface using MicroVGA[nb 1]

Some systems provide user interface remotely with the help of a serial (e.g. RS-232) or network (e.g. Ethernet) connection. This approach extends the capabilities of the embedded system, avoids the cost of a display, simplifies BSP and allows designers to build a rich user interface on the PC. A good example of this is the combination of an Embedded HTTP server running on an embedded device (such as an IP camera or a network router). The user interface is displayed in a web browser on a PC connected to the device.

Processors in embedded systems

Examples of properties of typical embedded computers, when compared with general-purpose counterparts, are low power consumption, small size, rugged operating ranges, and low per-unit cost. This comes at the price of limited processing resources.

Numerous microcontrollers have been developed for embedded systems use. General-purpose microprocessors are also used in embedded systems, but generally, require more support circuitry than microcontrollers.

Ready-made computer boards

PC/104 and PC/104+ are examples of standards for *ready-made* computer boards intended for small, low-volume embedded and ruggedized systems. These are mostly x86-based and often physically small compared to a standard PC, although still quite large compared to most simple (8/16-bit) embedded systems. They may use DOS, Linux, NetBSD, or an embedded real-time operating system (RTOS) such as MicroC/OS-II, QNX or VxWorks.

In certain applications, where small size or power efficiency are not primary concerns, the components used may be compatible with those used in general-purpose x86 personal 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, which contain code specific to the application.

However, most ready-made embedded systems boards are not PC-centered and do not use the ISA or PCI busses. When a system-on-a-chip processor is involved, there may be little benefit to having a standardized bus connecting discrete components, and the environment for both hardware and software tools may be very different.

One common design style uses a small system module, perhaps the size of a business card, holding high density BGA chips such as an ARM-based system-on-a-chip processor and peripherals, external flash memory for storage, and DRAM for runtime memory. The module vendor will usually provide boot software and make sure there is a selection of operating systems, usually including Linux and some real-time choices. These modules can be manufactured in high volume, by organizations familiar with their specialized testing issues, and combined with much lower volume custom mainboards with application-specific external peripherals. Prominent examples of this approach include Arduino and Raspberry Pi.

ASIC and FPGA SoC solutions

A system on a chip (SoC) contains a complete system - consisting of multiple processors, multipliers, caches, even different types of memory and commonly various peripherals like interfaces for wired or wireless communication on a single chip. Often graphics processing units (GPU) and DSPs are included such chips. SoCs can be implemented as an application-specific integrated circuit (ASIC) or using a field-programmable gate array (FPGA) which typically can be reconfigured.

ASIC implementations are common for very-high-volume embedded systems like mobile phones and smartphones. ASIC or FPGA implementations may be used for not-so-high-volume embedded systems with special needs in kind of signal processing performance, interfaces and reliability, like in avionics.

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 Interface: I2C, SPI, SSC and ESSI (Enhanced Synchronous Serial Interface)
- Universal Serial Bus (USB)
- Media cards (SD cards, CompactFlash, etc.)
- Network interface controller: Ethernet, WiFi, etc.
- Fieldbuses: CAN bus, LIN-Bus, PROFIBUS, etc.

- Timers: Phase-locked loops, programmable interval timers
- General Purpose Input/Output (GPIO)
- Analog-to-digital and digital-to-analog converters)
- Debugging: JTAG, In-system programming, background debug mode interface port, BITP, and DB9 ports.

A close-up of the SMSC LAN91C110 (SMSC 91x) chip, an embedded Ethernet chip

Tools

As with other software, embedded system designers use compilers, assemblers, and debuggers to develop embedded system software. However, they may also use 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 computational notebook to simulate the mathematics.
- System-level modeling and simulation tools help designers to construct simulation models of a system with hardware components such as processors, memories, DMA, interfaces, buses and software behavior flow as a state diagram or flow diagram using configurable library blocks. Simulation is conducted to select the right components by performing power vs. performance trade-offs, reliability analysis and bottleneck analysis. Typical reports that help a designer to make architecture decisions include application latency, device throughput, device utilization, power consumption of the full system as well as device-level power consumption.
- A model-based development tool creates and simulates graphical data flow and UML state chart diagrams of components like digital filters, motor controllers, communication protocol decoding and multi-rate tasks.
- Custom compilers and linkers may be used to optimize specialized 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 RTOS or embedded operating system
- Modeling and code generating tools often based on state machines

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. Considerations include: does it slow down the main application, how close is the debugged system or application to the actual system or application, how expressive are the triggers that can be set for debugging (e.g., inspecting the memory when a particular program counter value is reached), and what can be inspected in the debugging process (such as, only memory, or memory and registers, etc.).

From simplest to most sophisticated debugging techniques and systems 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)
- Software-only debuggers have the benefit that they do not need any hardware modification but have to carefully control what they record in order to conserve time and storage space.[10]
- 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 that 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.[11] 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 (ICE) 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. The downsides are expense and slow operation, in some cases up to 100 times slower than the final system.
- For SoC designs, the typical approach is to verify and debug the design on an FPGA prototype board. Tools such as Certus[12] are used to insert probes in the FPGA implementation that make signals available for observation. This is used to debug hardware, firmware and software interactions across multiple FPGAs in an implementation with capabilities similar to a logic analyzer.

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 High-level programming language, assembly code or mixture of both.

Tracing

Real-time operating systems often supports tracing of operating system events. A graphical view is presented by a host PC tool, based on a recording of the system behavior. The trace recording can be performed in software, by the RTOS, or by special tracing hardware. RTOS tracing allows developers to understand timing and performance issues of the software system and gives a good understanding of the high-level system behaviors.

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:

- 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.
- 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.
- 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[13] ensures a highly secure & reliable system environment
- A hypervisor designed for embedded systems 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.[14] This encapsulation keeps faults from propagating from one subsystem to another, thereby improving reliability. This may also allow a subsystem to be automatically shut down and restarted on fault detection.
- Immunity-aware programming can help to produce more reliable embedded systems code, and a variety of guidelines and industry standards such as MISRA C/C++ are available to assist developers.[15][16] These guidelines and coding rules aim to assist developers produce reliable, portable firmware in a number of different ways: typically by advising or mandating against coding practices which may lead to run-time errors (memory leaks, invalid pointer uses), use of run-time checks and exception handling (range/sanity checks, divide-by-zero and buffer index validity checks, default cases in logic checks), loop bounding, production of human-readable, well commented and well structured code, and avoiding language ambiguities which may lead to compiler-induced inconsistencies or side-effects (expression evaluation ordering, recursion, certain types of macro). These rules can often be used in conjunction with code static checkers and/or bounded model checking for functional verification purposes, and also assist in determination of code timing properties.[15]

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 RTOS.

Embedded software architectures

In 1978 National Electrical Manufacturers Association released a standard for programmable microcontrollers, including almost any computer-based controllers, such as single board computers, numerical, and event-based controllers.

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

Simple control loop or Programmed Input-output

In this design, the software simply has a loop which monitors the input device and executes the corresponding subroutine(s) only if there is a new action on the input device. The loop calls subroutines, each of which manages a part of the hardware or software. Hence it is called a simple control loop or programmed Input-output.

Interrupt-controlled system

Some embedded systems are predominantly controlled by interrupts. 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 non-preemptive multitasking system is very similar to the simple control loop scheme, except that the loop is hidden in an API.[3][1] 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 similar to that of the control loop, except that adding new software is easier, by simply writing a new task, or adding to the queue.

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 use a RTOS, 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, or battery life. The choice that an RTOS is required brings in its own issues, however, as the selection must be made prior to starting to the application development process. This timing forces developers to choose the embedded operating system for their device based upon current requirements and so restricts future options to a large extent.[17] The restriction of future options becomes more of an issue as product life decreases. Additionally, the level of complexity is continuously growing as devices are required to manage variables such as serial, USB, TCP/IP, Bluetooth, Wireless LAN, trunk radio, multiple channels, data and voice, enhanced graphics, multiple states, multiple threads, numerous wait states and so on. These trends are leading to the uptake of embedded middleware in addition to a RTOS.

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 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, VXWorks 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 and easier to debug, thus making the development process easier and the code more portable.
- Features requiring faster response than can be guaranteed can often be placed in hardware.

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 device 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.

Domain-specific architectures

In the automotive sector, AUTOSAR is a standard architecture for embedded software.

See also

- Communications server
- Cyber-physical system
- Electronic control unit
- Hypervisor
- Information appliance
- Integrated development environment
- Photonically Optimized Embedded Microprocessors
- Silicon compiler
- Software engineering
- System on module
- Ubiquitous computing

Notes

1. For more details of MicroVGA see this PDF (http://www.microvga.com/pdf/uvga-text-ds.pdf).

References

1. Michael Barr. "Embedded Systems Glossary" (http://www.netrino.com/Embedded-Systems/Glossary). *Neutrino Technical Library*. Retrieved 2007-04-21.
2. Heath, Steve (2003). *Embedded systems design* (https://archive.org/details/embeddedsystemsd0000heat). EDN series for design engineers (2 ed.). Newnes. p. 2 (https://archive.org/details/embeddedsystemsd0000heat/page/2). ISBN 978-0-7506-5546-0. "An embedded system is a microprocessor based system that is built to control a function or a range of functions."
3. Michael Barr; Anthony J. Massa (2006). "Introduction" (https://books.google.com/books?id=nPZaPJrw_L0C&pg=PA1). *Programming embedded systems: with C and GNU development tools*. O'Reilly. pp. 1–2. ISBN 978-0-596-00983-0.
4. Barr, Michael (1 August 2009). "Real men program in C" (https://www.embedded.com/electronics-blogs/barr-code/4027479/Real-men-program-in-C). *Embedded Systems Design*. TechInsights (United Business Media). p. 2. Retrieved 2009-12-23.
5. Shirriff, Ken (30 August 2016). "The Surprising Story of the First Microprocessors" (https://spectrum.ieee.org/tech-history/silicon-revolution/the-surprising-story-of-the-first-microprocessors). *IEEE Spectrum*. Institute of Electrical and Electronics Engineers. **53** (9): 48–54. doi:10.1109/MSPEC.2016.7551353 (https://doi.org/10.1109%2FMSPEC.2016.7551353). S2CID 32003640 (https://api.semanticscholar.org/CorpusID:32003640). Retrieved 13 October 2019.
6. "1971: Microprocessor Integrates CPU Function onto a Single Chip" (https://www.computerhistory.org/siliconengine/microprocessor-integrates-cpu-function-onto-a-single-chip/). *The Silicon Engine*. Computer History Museum. Retrieved 22 July 2019.
7. "Electronic Frontier Foundation" (https://www.eff.org/). *Electronic Frontier Foundation*.
8. Embedded Systems Dell OEM Solutions | Dell (http://content.dell.com/us/en/enterprise/oem-industry-solutions-build-your-product-with-dell). Content.dell.com (2011-01-04). Retrieved on 2013-02-06.
9. David Carey (2008-04-22). "Under the Hood: Robot Guitar embeds autotuning" (https://web.archive.org/web/20080708195311/http://embedded.com/underthehood/207401418). *Embedded Systems Design*. Archived from the original (https://www.embedded.com/underthehood/207401418) on 2008-07-08.
10. Tancreti, Matthew; Sundaram, Vinaitheerthan; Bagchi, Saurabh; Eugster, Patrick (2015). "TARDIS: Software-only System-level Record and Replay in Wireless Sensor Networks". *Proceedings of the 14th International Conference on Information Processing in Sensor Networks*. IPSN '15. New York, NY, USA: ACM: 286–297. doi:10.1145/2737095.2737096 (https://doi.org/10.1145%2F2737095.2737096). ISBN 9781450334754. S2CID 10120929 (https://api.semanticscholar.org/CorpusID:10120929).
11. Tancreti, Matthew; Hossain, Mohammad Sajjad; Bagchi, Saurabh; Raghunathan, Vijay (2011). "Aveksha: A Hardware-software Approach for Non-intrusive Tracing and Profiling of Wireless Embedded Systems". *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*. SenSys '11. New York, NY, USA: ACM: 288–301. doi:10.1145/2070942.2070972 (https://doi.org/10.1145%2F2070942.2070972). ISBN 9781450307185. S2CID 14769602 (https://api.semanticscholar.org/CorpusID:14769602).
12. "Tektronix Shakes Up Prototyping, Embedded Instrumentation Boosts Boards to Emulator Status" (http://www.eejournal.com/archives/articles/20121030-tektronix/). Electronic Engineering Journal. 2012-10-30. Retrieved 2012-10-30.

13. Heiser, Gernot (December 2007). "Your System is secure? Prove it!" (http://c59951.r51.cf2.rackcdn.com/5557-528-heiser.pdf) (PDF). *;login:*. **2** (6): 35–8.
14. Moratelli, C; Johann, S; Neves, M; Hessel, F (2016). "Embedded virtualization for the design of secure IoT applications" (https://ieeexplore.ieee.org/document/7909116). *2016 International Symposium on Rapid System Prototyping (RSP)*: 2–6. doi:10.1145/2990299.2990301 (https://doi.org/10.1145%2F2990299.2990301). ISBN 9781450345354. S2CID 17466572 (https://api.semanticscholar.org/CorpusID:17466572). Retrieved 2 February 2018.
15. Short, Michael (March 2008). "Development guidelines for dependable real-time embedded systems" (https://ieeexplore.ieee.org/document/4493674). *2008 IEEE/ACS International Conference on Computer Systems and Applications*: 1032–1039. doi:10.1109/AICCSA.2008.4493674 (https://doi.org/10.1109%2FAICCSA.2008.4493674). ISBN 978-1-4244-1967-8. S2CID 14163138 (https://api.semanticscholar.org/CorpusID:14163138).
16. Motor Industry Software Reliability Association. "MISRA C:2012 Third Edition, First Revision" (https://www.misra.org.uk/product/misra-c2012-third-edition-first-revision/). Retrieved 2022-02-03.
17. "Working across Multiple Embedded Platforms" (http://www.clarinox.com/docs/whitepapers/Whitepaper_06_CrossPlatformDiscussion.pdf) (PDF). clarinox. Retrieved 2010-08-17.

Further reading

- John Catsoulis (May 2005). *Designing Embedded Hardware, 2nd Edition*. O'Reilly. ISBN 0-596-00755-8.
- James M. Conrad; Alexander G. Dean (September 2011). *Embedded Systems, An Introduction Using the Renesas RX62N Microcontroller*. Micrium. ISBN 978-1935-7729-96.
- Klaus Elk (August 2016). *Embedded Software Development for the Internet Of Things, The Basics, The Technologies and Best Practices*. ISBN 978-1534602533.

External links

- Embedded Systems course with mbed (https://www.youtube.com/watch?v=H-OKGOMoCSI&list=PLo7bVbJhQ6qwlDa-R6pz7tA7kPzn1s5Ae) YouTube, ongoing from 2015
- Trends in Cyber Security and Embedded Systems (http://geer.tinho.net/geer.nro.6xi13.txt) Dan Geer, November 2013
- Modern Embedded Systems Programming Video Course (https://www.youtube.com/playlist?list=PLPW8O6W-1chwyTzI3BHwBLbGQoPFxPAPM) YouTube, ongoing from 2013
- Embedded Systems Week (ESWEEK) (http://www.esweek.org/) yearly event with conferences, workshops and tutorials covering all aspects of embedded systems and software
- Workshop on Embedded and Cyber-Physical Systems Education (http://www.emsig.net/conf/2015/wese/), workshop covering educational aspects of embedded systems