**National Research Journal of Information Technology & Information Science**
**Volume-1, Issue-1, Year-2014 PP: 32-38**

# A Study on Embedded Software

***Manisha Sharma***
***Department of Computer Science and Engineering, Shri Siddhi Vinayak Group of Institutions, Bilaspur, India.***

**Abstract**

*Embedded software are playing a very important role in our day to day life . They are used in our life in many forms. It makes our life easy and we are not in a state to think without it. Examples of embedded software include pacemakers, cellphones, home appliances, energy generation and distribution, satellites, and automotive components such as antilock brakes. Embedded software creates both huge value and unprecedented risks. This paper give us a brief study of an embedded software with the help of which we can make the embedded system. In this paper I give a brief description about the embedded system like operating system, architecture, tools and challenges*

## Introduction

Embedded software refers to a computer program that plays a vital role in the electronics it is supplied with. This software is written on computer chips and has widely been applied in real time systems like teller machines, missiles and airplanes.

Embedded software is computer software, written to control machines or devices that are not typically thought of as computers. It is typically specialized for the particular hardware that it runs on and has time and memory constraints.[1] This term is sometimes used interchangeably with firmware, although firmware can also be applied to ROM-based code on a computer, on top of which the OS runs, whereas embedded software is typically the only software on the device in question.

As we know that not all functions of embedded software are controlled via a human interface, but through machine-interfaces instead. This is a precise and stable characteristic feature of an embedded system.

Manufacturers 'build in' embedded software in the electronics in cars, telephones, modems, robots, appliances, toys, security systems, pacemakers, televisions and set-top boxes, and digital watches, for example. This software can be very simple, such as lighting controls running on an 8-bit microprocessor and a few kilobytes of memory, or can become very sophisticated in applications such as airplanes, missiles, and process control systems.

## 1. Operating system

Standard computers use a small number of operating systems e.g. Windows and Linux. Embedded software comes with a variety of operating systems specially real time operating system

An embedded operating system is typically quite limited in terms of function – depending on the device in question, the system may only run a single application. However, that single application is crucial to the device’s operation, so an embedded OS must be reliable and able to run with constraints on memory, size and processing power.

RTOS, RTEMS,INTEGRITY, UC/OS, QNX and OSE. Code is typically written in C or C++. Ada is used in some military and aviation projects.

An operating system (OS) is an optional part of an embedded device's system software stack, meaning that not all embedded systems have one. OSs can be used on any processor (Instruction Set Architecture (ISA)) to which the OS has been ported. As shown in Figure 9-1, an OS either sits over the hardware, over the device driver layer, or over a BSP (Board Support Package, which will be discussed in Section 9.7).

Click for larger image

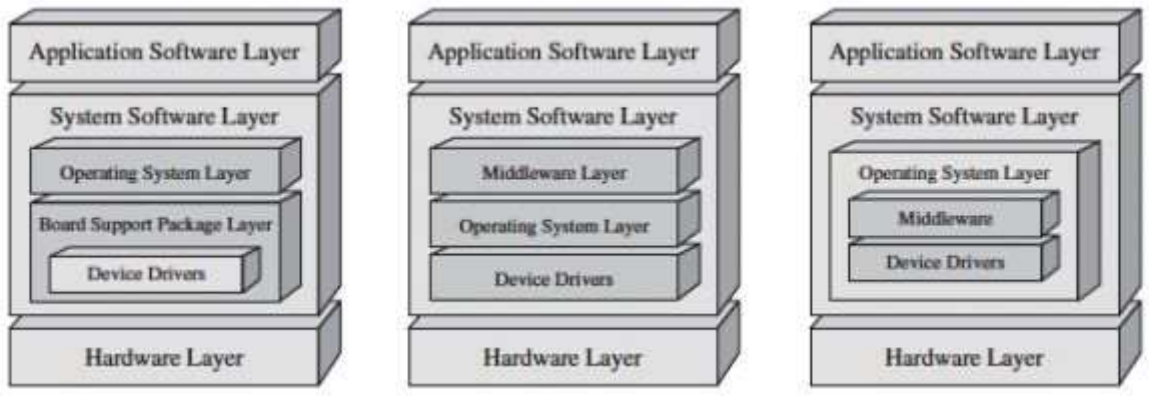

Figure 9-1. OSs and the Embedded Systems Model.

The OS is a set of software libraries that serves two main purposes in an embedded system: providing an abstraction layer for software on top of the OS to be less dependent on hardware, making the development of middleware and applications that sit on top of the OS easier, and managing the various system hardware and software resources to ensure the entire system operates efficiently and reliably. While embedded OSs vary in what components they possess, all OSs have a kernel at the very least. The kernel is a component that contains the main functionality of the OS, specifically all or some combination of features and their interdependencies

## 2. Difference between application software and embedded software

Most consumers are familiar with application software. Application software is all the computer software that causes a computer to perform useful tasks beyond the running of the computer itself. A specific instance of such software is called a software application, application program, application or app. that provide functionality on a computer.

Embedded software however is often less visible, but no less complicated. Embedded software has fixed hardware requirements and capabilities, addition of third-party hardware or software is strictly controlled.

Embedded software needs all needed device drivers at manufacturing time, and the device drivers are written for the specific hardware. This software is highly dependent on the CPU and specific chips chosen. Most embedded software engineers have at least a passing knowledge of reading schematics, and reading data sheets for components to determine usage of registers and communication system.

Web applications are rarely used, although XML files and other output may be passed to a computer for display. File systems with folders are typically absent as are SQL databases.

Software development requires use of a cross compiler, which runs on a computer but produces executable code for the target device. Debugging requires use of an in-circuit emulator (An in-circuit emulator (ICE) is a hardware device used to debug the software of an embedded system.)

**3. Communication Protocols**

Communications between processors and between one processor and other components are essential. Besides direct memory addressing, common protocols include I²C, SPI, serial ports, and USB.

Communications protocols designed for use in embedded systems are available as closed source from companies including InterNiche Technologies and CMX Systems. Open-source protocols stem from uIP, lwip, and others.

**4. Embedded Software Architecture**

There are several type of architecture is used for an embedded software which are explained below:

**5.1 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. This is one of the examples of simple control loop.

**5.2 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.

**5.3 Cooperative multitasking[edit]**

A nonpreemptive multitasking system is very similar to the simple control loop scheme, except that the loop is hidden in anAPI. 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.

**5.4 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. 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..

Because of some complexities like that any code can damage the data of another task so programs must be designed and tested and access to data must be controlled by some strategy, it is common for organizations to use a real-time operating system (RTOS), allowing the application programmers to concentrate on device functionality, 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 done prior to starting to the application development process.

**5.5 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 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.

**5.6 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.

**5.7 Exotic custom operating systems**
A small fraction of embedded systems require safe, timely, reliable, or efficient behavior unobtainable with any 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.

**5.8 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.

**6.Tools**
As with 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
- 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, Mathematica,or FlowStone DSP to simulate the mathematics. They might also use libraries for both the host and target
- A model based development tool like VisSim lets you create and simulate graphical data flow and UML State chart diagrams of components like digital filters, motor controllers, communication protocol decoding and multi-rate tasks.Interrupt handlers can also be created graphically.
- 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 real-time operating system or embedded operating system, which may have DSP capabilities like DSPnano RTOS.
- 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.

**7 The Challenges of Embedded Software**

Most microprocessors are in systems for cars, mobile communication, washing machines, aircraft, robots, traffic management, cameras, and audio equipment.

This trend toward embedded software in practically all systems is accelerating. The world market for embedded systems is approximately 160 billion euros, Cars today have 100 Mbytes of software running, with a complexity growing more quickly than that of IT systems such as those of SAP, Oracle, and Microsoft. The same holds for pacemakers and satellites, starting from a smaller size, of course. To highlight embedded software's ubiquity, Figure 1 shows the size and annual distribution volume of selected embedded software.

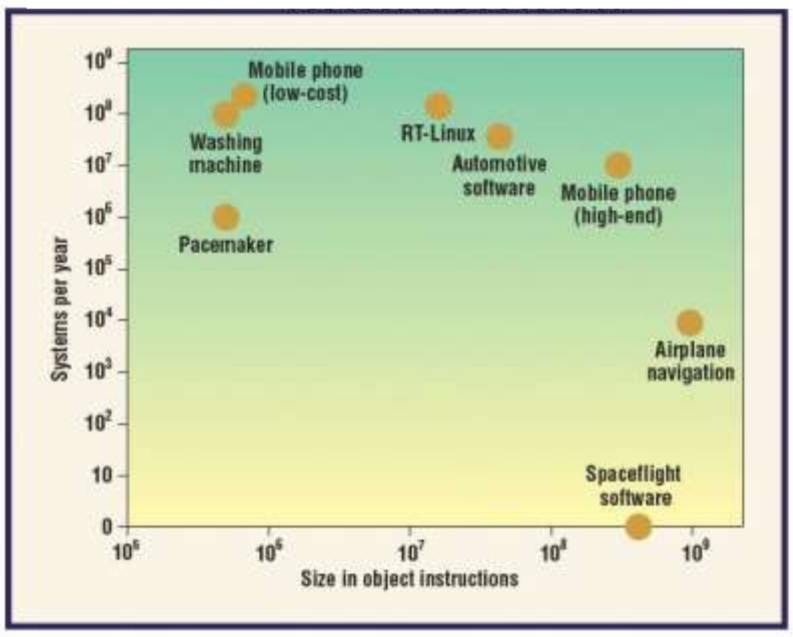

Figure 1. Embedded software size and deployment.

We divide the challenges into six parts:

### 7.1 Real Time

Embedded software is by definition part of a larger system, such as a technical process, the human body, or an industrial automation system. These systems all pose external constraints that must be addressed in real time. The embedded system's timing must provide the expected action within a maximum specified time under all circumstances.

### 7.2 Reliability

Unexpected behavior from an embedded system might seriously damage its environment. Often, such software must operate for decades without service; there's no possibility for upgrades and service patches. End users demand deterministic long-term behaviors from these systems.

### 7.3 Safety

Safety has become a key requirement, specifically as mechanical backups are for cost reasons The entire life cycle thus is governed by standards that demand systematic processes, state-of-the-practice technologies, and continually educated engineers.

### 7.4 Security

In many embedded systems, security directly means safety. A car contains approximately 30 to 70 embedded systems that communicate with each other across a variety of standardized bus systems. Embedded security is crucial to avoid life-threatening situations.

### 7.5 Limited Resources

Embedded software is constrained by small memory space and limited data-processing capabilities, low-cost microcontrollers, stringent regulations with respect to "green" behaviors, and low power consumption. For instance, implanted medical devices must operate for years without battery replacement, and mobile phones must work for hours on subwatt batteries.

### 7.6 Heterogeneity

Embedded software's long lifetime, it must conform to a wide range of changes in its environment. Processors, sensors, and hardware parts change over time, whereas the software remains almost the same.

## Conclusion

a) Embedded Software
b) Operating system and communication protocols
c) Difference b/w application software and embedded software
d) Types of Architecture of embedded software
e) Tools used for the embedded software
f) Challenges in embedded software

## REFERENCES

1. Edward A. Lee, "Embedded Software", Advances in Computers (M. Zelkowitz, editor) 56, Academic Press, London, 2002.
    1. "Stroustrup on C++ for embedded (bottom p.2)". Retrieved 9 December 2012.
    2. "I.C.S. on embedded software". Retrieved 22 July 2013.
    3. ^"Embedded Systems Methods and Technologies". Retrieved 9 December 2012.
    4. http://ptolemy.eecs.berkeley.edu/publications/papers/02/embsoft/embsoftwre.pdf
    5. "Stroustrup on embedded software". Retrieved 9 December 2012.
    6. "Example of embedded CPU". Retrieved 9 December 2012.
2. Heath, Steve (2003). Embedded systems design. EDN series for design engineers (2 ed.). Newnes. p. 2.ISBN 978-0-7506-5546-0. "An embedded system is amicroprocessor based system that is built to control a function or a range of functions."

1. Michael Barr; Anthony J. Massa (2006). "Introduction".Programming embedded systems: with C and GNU development tools. O'Reilly. pp. 1–2. ISBN 978-0-596-00983-0.
2. Giovino, Bill. "Micro controller.com – Embedded Systems supersite".

3. "Tektronix Shakes Up Prototyping, Embedded Instrumentation Boosts Boards to Emulator Status". Electronic Engineering Journal. 2012-10-30. Retrieved 2012-10-30.

1. Heiser, Gernot (December 2007). "Your System is secure? Prove it!". ;login: 2 (6): 35–8.

4. "Working across Multiple Embedded Platforms". clarinox. Retrieved 2010-08-17.