

μITRON: A Standard Real-Time Kernel Specification for Small-Scale Embedded Systems

The μITRON is a standard real-time kernel specification for use in small-scale embedded control systems. This specification has been implemented for a large number of 8-bit, 16-bit, and 32-bit microcontroller units (MCUs). Counting only commercial products, about 40 implementations have been reported. Moreover, ITRON-specification kernels have been adopted in countless embedded systems, giving the specification the stature of an industry standard in this field in Japan.

CHARACTERISTICS OF SMALL-SCALE EMBEDDED SYSTEMS

Advances in microcomputer technologies have continued to expand the application fields for embedded systems. What came into the world originally for industrial uses such as production line control in factories later expanded to communication equipment, office automation and other business uses. More recently, the world of embedded systems has spread out to include automotive systems, electronic musical instruments, cellular phones, TV's, audio gear, washing machines, air-conditioners and many more consumer applications.

As a general rule, small-scale embedded applications tend to be manufactured in much greater quantities than large-scale systems, and are priced more cheaply. For these reasons manufacturers of large-scale embedded systems try to emphasize reducing development costs, while makers of small-scale embedded systems work hardest at lowering the cost of the final product. Another feature of small-scale embedded systems is that once a product is put on sale, the software is rarely modified, therefore the system development life cycle is very short.

In the field of small-scale embedded systems, it is common practice to use a single-chip microcontroller unit (MCU), integrating a processor core with ROM, RAM, general I/O devices and application-specific peripheral modules. Looking at worldwide semiconductor trade statistics for 1995 (Table 1), about 2.9 billion MCUs were shipped, accounting for 89.5 percent of processor shipments compared to a mere 6.5 percent for MPUs.

A particular problem in developing software for an MCU is the limited hardware resources, due to the need for cost reduction. Memory size limits are especially severe, with a typical 8-bit MCU having around 32 KB of ROM and 1 KB of RAM. Even larger-scale chips have no more than 128 KB of ROM and 4 KB of

RAM. Another point worth noting is that the need for high cost performance in an MCU-based system means the design is frequently optimized to the application, resulting in a large number of different processor cores.

Even in this field of small-scale embedded systems, raising software productivity is an important issue. Use of C or other high-level programming languages is one solution; another approach being adopted with increasing frequency is the use of a real-time OS like those implementing the μITRON specification.

Manufacturers of large-scale embedded systems try to emphasize reducing development costs, while makers of small-scale embedded systems work hardest at lowering the cost of the final product.

REQUIREMENTS ON A STANDARD REAL-TIME OS FOR EMBEDDED SYSTEMS

Requirements on a standard real-time OS for small-scale embedded systems are summarized as follows.

Being able to derive maximum performance from hardware

Given the severe hardware resource limitations of a typical MCU-based system, the ability to derive maximum performance from the available hardware is a prerequisite for real-time OS adoption.

Kinds	Q'ty	% in pcs
8-bit MPU	48.2	1.5%
16-bit MPU	58.8	1.8%
32-bit or greater MPU	104.6	3.2%
4-bit MCU	1,066.0	32.5%
8-bit MCU	1,667.4	50.9%
16-bit or greater MCU	199.2	6.1%
DSP	134.2	4.1%
Total	3,278.4	

Table 1. Shipment of MPU and MCU
(1995, worldwide, unit of million)
(from World Semiconductor Trade Statistics)

In the case of small-scale embedded systems, maximizing hardware performance is considered more important than full source code compatibility. This is because porting to a different processor and its OS normally becomes necessary only when there are changes also in the equipment being controlled, so there is hardly ever a situation where an application program is ported to another OS without modification. One more reason for reducing the importance of complete source code compatibility is that when the processor itself is changed, the software fine-tuning process is usually carried out again in order to lower the cost of the final product.

Helping to improve software productivity

Especially important is standardisation from a training standpoint, such as adopting consistent concepts and terminology, and standardizing design methods.

Being uniformly applicable to various processor scales and types

The hardware used in a small-scale embedded system is normally designed optimally for its application. The processor scale, moreover, may vary widely from 8-bit to 32-bit processors depending on the kind of equipment to be controlled.

In addition to the above requirements, another very important issue is whether the specifications are truly open. This means not only that the specification documents can be obtained, but also that everyone is free to implement and sell products based on those specifications.

PROBLEMS WITH CONVENTIONAL OSS

Having examined above the requirements on a real-time OS for small-scale embedded systems, we now look at some problems with a conventional OS.

The first problem that can be raised is the lowering of runtime performance caused by virtualization of the processor architecture. When an OS is designed to run on a number of different processors, the usual approach is to assume a virtual architecture common to those processors and to build the OS on that architecture. With this approach, however, the differences between the virtual architecture and that of the actual processor are a cause of runtime overhead, lowering performance.

A second problem is the tendency of an OS to be equipped with advanced functions many small-scale embedded system do not need, for the sake of software productivity. A feature-heavy OS can adversely affect performance, and make it difficult to run the OS on limited hardware resources.

One more major problem is the lack of an open standard specification for a real-time OS that can be used with many different types of processors. Semiconductor manufacturers provide an OS only for their own

processors, while OS suppliers are unable to support the wide range of processors in use today. The lack of standard specifications also means the same word has different semantics with each OS, not to mention the confusion resulting from the many different ways in which similar functions are implemented in each OS.

DESIGN PRINCIPLES OF THE μITRON SPECIFICATION

Responding to the need for standardisation discussed above, around ten years ago we began studying specifications for a standard real-time kernel for embedded systems, leading to the development and release of a series of ITRON specifications. The reason for centering these studies on a kernel specification is that in most deeply embedded systems, only the kernel functions are used.

The following design principles were established for the μITRON specification, in order to satisfy the requirements noted above.

Avoid excessive hardware virtualization

Too much hardware virtualization must be avoided in order to derive the maximum performance from hardware and achieve high real-time performance.

A feature-heavy OS can adversely affect performance, and make it difficult to run the OS on limited hardware resources.

Although the μITRON specification is a kernel specification intended for implementing on a variety of different processors, the basic policy is to design each implementation independently for each processor.

To this end, the μITRON specification is divided clearly into aspects that are standard across all processors and implementation-dependent aspects. Standardized items include task scheduling rules, system call names and functions, parameter names, sequence and meanings, and error code names and meanings. On the other hand, those aspects that need to be decided separately for each implementation, based on runtime performance considerations, are not precisely standardized.

Allow for optimization to application

Optimizing to the application means modifying the kernel specification and internal implementation method based on the requirements of a particular application. In the case of embedded systems, the kernel object code is generated for each application, making this optimization especially effective.

Specifically, the specification was designed so as to make the kernel functions independent of each other to the extent possible, so that each application can use just the functions it needs. In fact, most μITRON-specification kernels are provided in the form of libraries, and are designed so that only the necessary modules are loaded when the kernel is linked to the application. Also, each system call provides a single function, making it easy to select out the necessary functions for an application.

Allow for optimization to hardware

Modifying the kernel specification and internal implementation method, based on the characteristics of the hardware and its performance, also increases system performance. For example, the method of invoking interrupt handlers is left unspecified in the ITRON specifications. In fact, it is a usual approach to invoke a user-defined interrupt handler when an external interrupt occurs without going through the operating system. The overhead required here is practically zero. The user must, instead, save the registers used in the interrupt handler.

Emphasize ease of software engineer training

As noted before, software compatibility and portability tend to be considered as of relatively minor importance with small-scale embedded systems. The significance of standardizing kernel specifications relates more to ease of training software engineers, and the improved communication among software personnel resulting from consistent terminology and concepts.

A primary aim of standardisation in the μITRON specification is to facilitate learning by and training of software engineers, so that once they learn something they will be able to apply that knowledge broadly. Likewise, the use of terminology in the specification, and things like the way system calls are named, are made as consistent as possible.

Create a specification series and/or divide into levels

Specifications are developed in series to make them applicable to a wide variety of hardware. Moreover, each specification divides functions into different levels based on their degree of necessity.

In addition, work on creating standard specifications for network-linked distributed systems and specifications for multiprocessor systems is being carried out within the ITRON specification series. Of these, the specifications for distributed systems are included in the latest specification.

Make available a full range of functions

Rather than limiting the number of primitives provided by the kernel, the approach is taken of making available a wide variety of primitives with different functions. The idea is to enable implementors to raise the runtime performance and improve ease of programming by using primitives geared to the particular hardware and application.

A concept common to many of these design principles is that of loose standardisation. This means setting uniform standards only to the extent that performance will not suffer, rather than trying to force all systems into one rigid mold, and leaving room to decide matters dependent on the processor or application.

CURRENT STATUS OF THE μITRON SPECIFICATION

As noted above, we have been devising and releasing a series of ITRON specifications since about ten years ago. The first ITRON specification was released in 1987 as ITRON1. Thereafter studies were carried out on a reduced-function specification called μITRON (Ver 2.0) for smaller-scale 8-bit and 16-bit MCUs, and on the ITRON2 specification for larger-scale systems with 32-bit MPUs. Both of these were released in 1989.

Of these, the μITRON specification offered very realistic performance even on an MCU with only very limited processing and memory resources, and has therefore been implemented on many different

Company	Processor
FUJITSU LIMITED	F2MC-16
	F2MC-8L
	FR20 Series
	SPARClite Series
Hitachi, Ltd.	H8/300
	H8/500
	H8/300
	SH
Mitsubishi Electric Semiconductor Softw Corp	M32 Family
	7700 Family
	M16 Family
	38000 Series
Miyazaki System Design Co	M16C/60 Series
	8086
	H8/300H
	Z80
CPU32 Morson Japan	SH-1, SH-2
	H8/500
	68000, 68020,
	MC68020
NEC	MC68000
	8086 Family
	78K/III Series
	78K/II Series
System Algo Co., Ltd. Sony Corp.	78K/0 Series
	78K/IV Series
	i960 Series
	SPC900
Three Ace Computer Corp.	8086 Family
	TOSHIBA CORP.
Toshiba Information Systems	TLCS-90
	TLCS-900
Toshiba Information Systems	8086 Series
	68000 Series

Table 2: μITRON-specification kernel implementations (Products registered with the TRON Association as of Feb. 10, 1997)

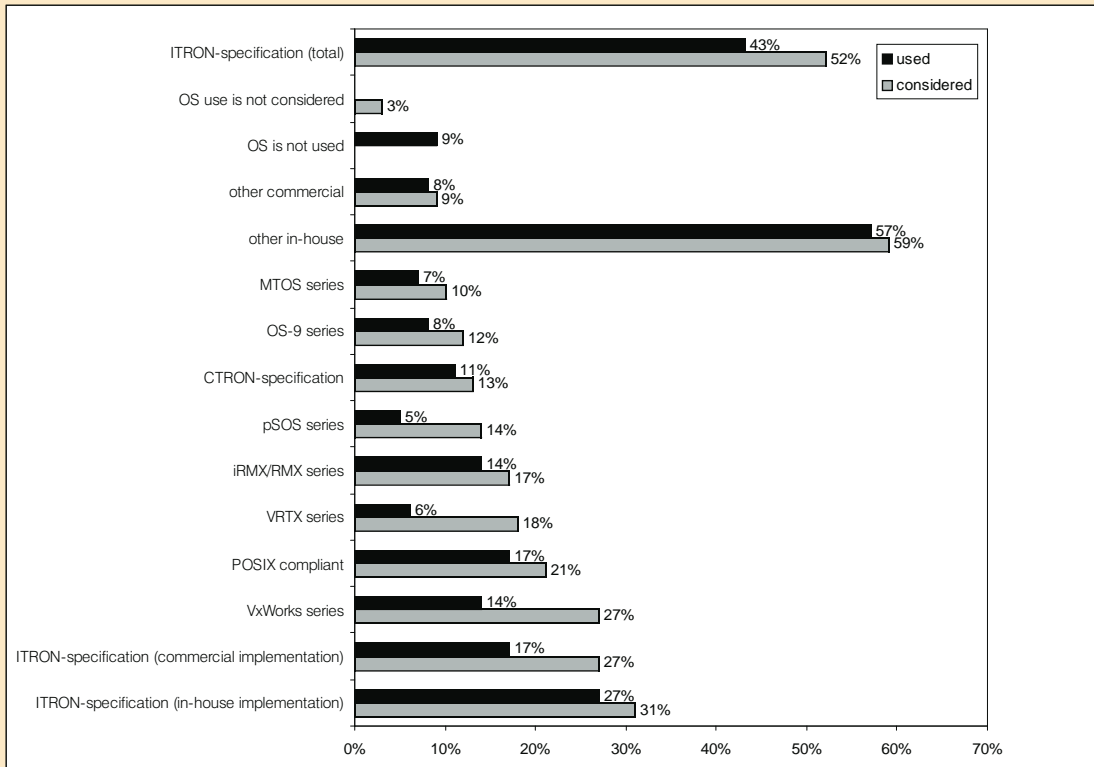


Figure 1: Real-time OSs used (or considered for use) in embedded systems

MCUs. In fact, it would not be going too far to say that μITRON-specification kernels have been developed for the 8-bit MCUs of nearly all Japanese manufacturers. Its application has even widened to various 16-bit MCUs as well as 32-bit MCUs and MPUs. Just counting the μITRON-specification products that have been registered officially with the TRON Association, there are more than 30 implementations for more than 20 processors (Table 2). In addition to them, the μITRON-specification kernel, with its small size and relative ease of implementation, has been used in numerous developments exclusively for in-house systems. There are also several μITRON-specification kernels that have been made available as free software.

It goes without saying that the reason for this large number of μITRON-specification kernel implementations is the wide range of application fields and numerous application examples. Table 3 lists some of the applications in which μITRON-specification kernels are used.

As is clear from a survey taken in Japan by the TRON Association in 1995 (Figure 1), the μITRON specification is widely used in Japan and can be said to be a defacto industry standard. Considering also the many cases where this specification is used for in-house systems, clearly the μITRON specification has become established as a truly open standard specification.

As the μITRON-specification kernel has come to be applied to a wide range of fields, a clearer picture has emerged as to the necessity of each function and the performance demands. Also, as noted above, the μITRON-specification kernel has in some instances been implemented for 32-bit processors, something we did not originally anticipate. It was therefore deci-

Consumer Applications

TVs, VCRs, digital cameras, audio components, settop box, air-conditioners, washing machines, micro-wave ovens, game gear, rice cookers, lighting

Office Applications

printers, copiers, image scanners, word processors, FAX, optical filing systems

Communications

answer phones, ISDN telephones, cellular phones, broadcasting equipment, wireless systems, antenna controllers, satellite controllers, ATM switches

Other Applications

PDA's, automobiles, vending machines, electronic musical instruments, FA computers, industrial robots

Table 3. Typical ITRON-specification kernel applications

GreenSpring Ad

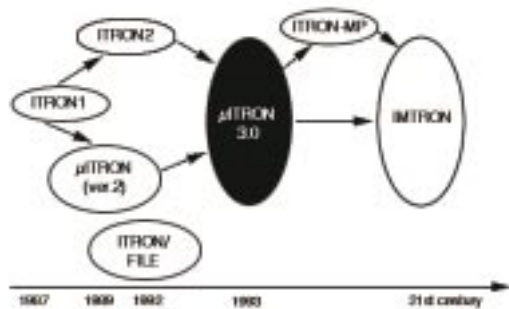


Figure 2. Development of the ITRON specification

ded to rethink the existing ITRON specifications, resulting in the release in 1993 of the third-generation ITRON specification, called μITRON 3.0. The main functions in the μITRON 3.0-specification kernel are listed in Table 4. The μITRON 3.0 specification is available on the Internet at "ftp://tron.um.u-tokyo.ac.jp/TRON/ITRON/SPEC/mitron3.txt.Z"

1. Task management

- Direct manipulation and referencing of task status.

2. Task-dependent synchronisation

- Task synchronisation function in the task itself.

3. Synchronisation and communication

- Three synchronisation and communication functions independent of tasks, namely, semaphore, eventflag and mailbox functions.

4. Extended synchronisation and communication

- Two advanced task-independent synchronisation and communication functions, namely, message buffer and rendezvous.

5. Interrupt management

- Function for defining a handler for external interrupts.
- Function for disabling and enabling external interrupts.

6. Memory pool management

- Functions for software management of memory pools and memory block allocation.

7. Time management

- Functions for system clock setting and reference.
- Task delay function.
- Timer handler functions, for time-triggered starting.

8. System management

- Functions for setting and referencing the system environment as a whole.

9. Network management

- Management and support functions for a loosely coupled network.

Table 4. Main functions supported in μITRON3.0-specification kernel

NETWORK SUPPORT AND FUTURE OUTLOOK

With low-cost MCUs becoming available, multiple MCU's are increasingly being used to control one piece of equipment. A major reason is to increase ease of maintenance and reliability by configuring products of component units. Specific examples include copiers, fax machines and automobiles.

In line with this trend, the μITRON 3.0 specification incorporates additional functions, called connection functions, that support distributed systems consisting of ITRON-specification kernel-based nodes interconnected on a network. With the connection functions, objects such as tasks and semaphores on other nodes can be directly manipulated using ordinary system calls.

The connection functions in the μITRON 3.0 specification at this time are intended only for controlling one piece of equipment or system, and it is necessary to decide the network configuration statically at the time the system is built. As the next step it will be important to enable interconnection among separately designed systems. It will further be necessary to deal with dynamic changes in network configuration. The specification at that stage is called IMTRON, which is positioned as the next-generation μITRON specification (Figure 2).

In building a distributed system, the communication interface between different equipment can be simplified if all are using the same kernel specification,

WHAT IS THE TRON PROJECT?

TRON (The Real-time Operating system Nucleus) is a project started by Dr. Ken Sakamura of University of Tokyo in 1984. The project aims to create an ideal computer architecture with a view toward the future computerized society. In the computerized society, all the objects making up our living environment will come to have microcomputers embedded in them. These objects, moreover, will be capable of interacting with each other via world-wide computer networks. The long-term goal of the TRON Project is to develop highly functionally distributed system (HFDS), in which each of these computerized objects is able to work in cooperation with other objects. These computer-embedded networked objects are called intelligent objects.

The project is now going ahead on various subprojects including the ITRON subproject. The BTRON subproject aims to design an OS specification for personal computers and workstations. CTRON is an OS interface specifications for communication and information processing. MTRON is an attached OS architecture for connecting various systems in HFDS. CHIP subproject aims to design a VLSI microprocessor architecture for use in these OSs. HMI subproject designs standard human-machine interface guidelines. Application subprojects, including the TRON-concept Computer Augmented Building subproject, are proceeded to find problems in actual applications of HFDS.

which also realizes improved runtime performance and greater development efficiency. In that sense the existence of μITRON as a standard OS specification will have even greater significance in a distributed environment.

Other important development projects are the ITRON-MP specification studies, with the aim of extending ITRON to shared-memory multiprocessor systems, and the on-going work to realize a standard environment supporting software development on the ITRON-specification kernel.

Hiroaki Takada is a research associate at the Dept. of Information Science, Univ. of Tokyo. He has received Ph.D. in information science in 1996. His current research topics include real-time operating system, real-time scheduling theory, and computer networks. He is also taking an important role in designing the ITRON specification, a standard real-time kernel specification for small-scale embedded systems. He has developed a free real-time kernel based on the specification as a reference implementation.