



## eT-Kernel : Product Line Platform Management

### T-Kernel

Reducing development time is increasingly important in today's world of intense time-to-market pressure. The best software development practices advocate reusable code. Embedded systems developers rarely achieve this goal because code is so dependent on the development environment: hardware, system, and OS. When the OS and hardware must change to meet new product requirements, the system and even application code must often change as well.

A standardized embedded platform addresses these issues by allowing for fast porting, and for code reuse from project to project. In this article, I discuss two such standards:  $\mu$ ITRON and T-Engine. The use of  $\mu$ ITRON in Japanese and Asian products in the last decade has proved its success. T-Engine is an evolution of  $\mu$ ITRON that meets evolving requirements for today's embedded applications: software reuse, loadable modules, process-based operating system, and more.

Standardized operating systems allow companies to reduce development costs by reducing overhead associated with recreating software from scratch for each project. Software portability means that it is easy to port the entire system code to another hardware platform. Reusing proven software modules on multiple platforms reduces development cost and time. Developers in the enterprise world have a choice of many software standards, but standards have been slow to spread in the embedded world. The embedded standards in wide use today are Linux and  $\mu$ ITRON. T-Engine wants to be the next generation standard. This article will not discuss embedded Linux because the topic is well-covered elsewhere. This article provides an overview of the  $\mu$ ITRON and T-Engine specifications, and discusses the open source version of T-Kernel as well as an optimized solution called eT-Kernel from eSOL.

*μITRON: OS standardization for embedded devices*

### What is μITRON?

μITRON is a real time OS standard for microcontrollers. μITRON is part of the Real Time Nucleus (TRON) initiative launched in Japan in 1984. While the ubiquitous computing goals of TRON remain, μITRON has had great success. About 80% of Japanese products that use a real time OS use a μITRON OS. Virtually every Japanese chip vendor and OS company provides a μITRON-based OS. While the use of μITRON started in Japan, its use has spread to Asia and beyond.

The first ITRON specification was released in 1984. The μITRON version 2.0 specification for 8, 16, and 32-bit microcontrollers came out in 1987. The μITRON specification evolved to meet changing customer requirements, culminating in the μITRON version 4.0 specification released in 1999. The wide adoption of μITRON in Japan has reduced development costs in part by flattening the learning curve for engineers. A firmware engineer can move from project to project, or even among companies, and still maintain expertise in developing applications with a μITRON OS.

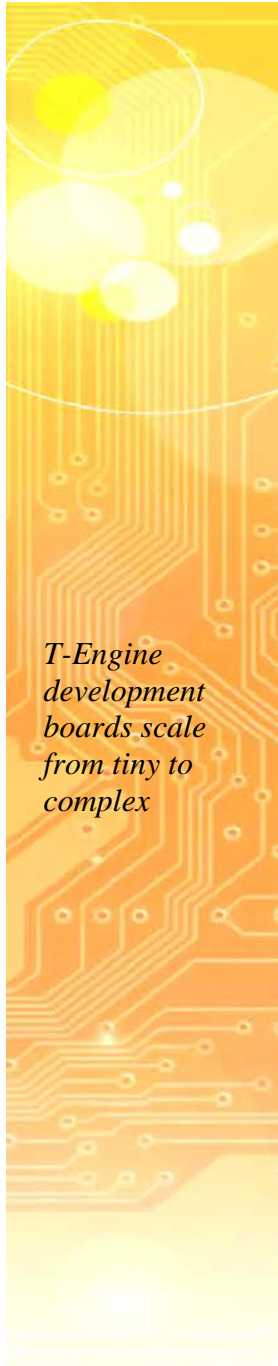
The μITRON specification is “weak standardization.” It specifies the kernel architecture and API. The standard does not specify the driver interface, middleware interface, or the hardware itself. Although μITRON met some goals for reducing development time, the lack of stronger standardization means that it is still difficult to move an entire application from one μITRON-based platform to another. These considerations led the industry to see a need for a more complete standard. T-Engine is the evolution of μITRON to a more complete standard.

### What is T-Engine?

T-Engine is an open specification for embedded systems development that provides strong standardization and allows scaling. The T-Engine standard specifies both hardware and software. The goals of T-Engine are to provide a platform for CPU-independent code that is easy to migrate from one CPU to another, on an open environment. The T-Engine standard includes specifications for the following areas:

- T-Engine development boards

*T-Engine – Open development platform that scales for wide range of applications*



*T-Engine  
development  
boards scale  
from tiny to  
complex*

- T-Engine Software
  - T-Kernel OS
  - Middleware framework
  - Driver framework
  - Debugging framework

The T-Engine forum is an industry consortium that defines and develops the T-Engine specification. The Forum promotes T-Engine internationally, and has over 450 members from countries all over the world. Forum members develop and distribute reusable middleware to encourage a large T-Engine community. The following sections discuss technical details about the T-Engine development platform, the T-Kernel OS, and the associated software frameworks.

### ***T-Engine Development Boards***

The T-Engine development board specification aims to make application development easy and CPU-independent. The development environment is standardized and supported to the point that switching CPUs should involve minor effort. The user is still free to design custom hardware in any form – the development board specification serves to provide a quick and simple platform to start product development. T-Engine hardware scales from complex to tiny with four different development board specifications:

- T-Engine  
For complex devices with a user interface, such as PDAs, smartphones, set top boxes, car mounted information systems, etc.
- $\mu$ T-Engine  
For control devices without a sophisticated user interface, such as controllers for home information appliances, network devices, factory automation, etc.
- n-T-Engine (nano-T-Engine)  
Network node in control systems such as building control, lighting, etc. Multiple nodes make up a real time network.
- p-T-Engine (pico-T-Engine)  
Low power node used to build up sensor network

The standard T-Engine and  $\mu$ T-Engine boards are available today. Both boards have standardized dimensions, peripherals, peripheral placement, connectors, etc. The pictures and table below show the differences between a T-Engine and  $\mu$ T-Engine board.



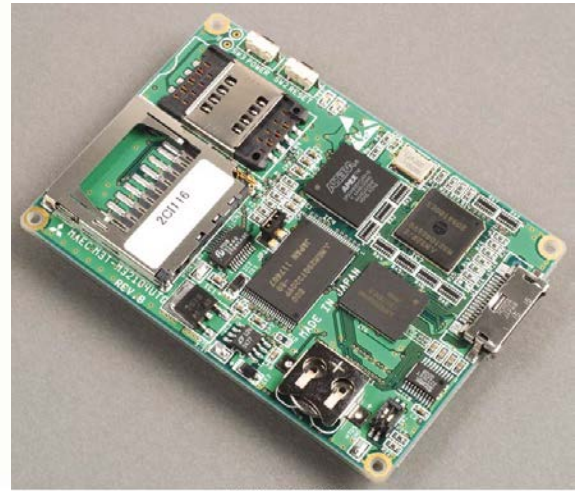
| Feature            | T-Engine       | μT-Engine         |
|--------------------|----------------|-------------------|
| Board size         | 70 mm x 120 mm | 60 mm x 85 mm     |
| MMU                | Mandatory      | Optional          |
| Sound CODEC        | Mandatory      | Optional          |
| LCD                | Mandatory      | Optional          |
| Touch panel        | Mandatory      | Optional          |
| Card interface     | PCMCIA         | CF card, MMC card |
| USB Host interface | Mandatory      | Optional          |

**Table 1: T-Engine and μT-Engine Board Features**



Picture 1. Standard T-Engine/SH7727 (equipped with carrying case)

**Figure 1: T-Engine Development Board**



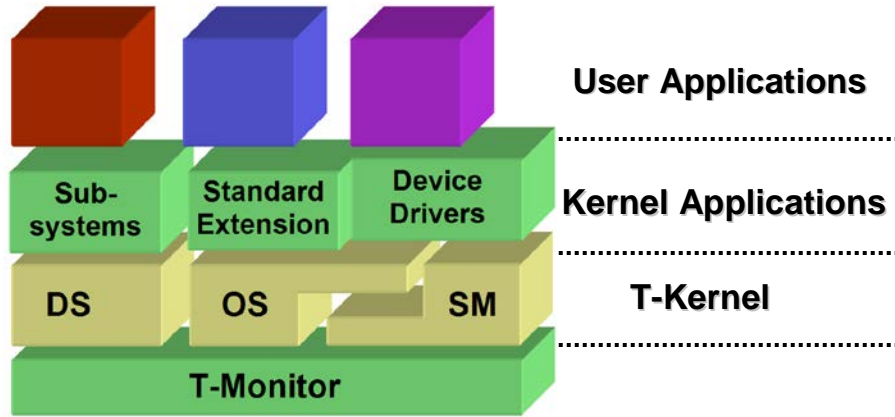
Picture 2. μT-EngineM32104

**Figure 2: μT-Engine Development Board**

T-Engine provides a layered, scalable architecture. The software is designed from the ground up for memory protection and a process-based OS, but can also scale down to a non-MMU system. OS and middleware vendors must provide products that implement the full specification. Users can then scale back by choosing only the parts they need to use. The T-Engine specification is open, and an open-source version of T-Kernel is available from the T-Engine forum. The T-Engine software architecture consists of the components shown in Figure 3.



*T-Engine software architecture: framework for full system*



**Figure 3: T-Engine Software Architecture**

*T-Monitor*

T-Monitor is a CPU and board dependent module that provides hardware initialization, system startup code, and debugging functions. T-Monitor starts the T-Kernel OS. T-Monitor code is not provided with the open source T-Kernel. The user needs either to write the code themselves, or get T-Monitor code from the board vendor.

*T-Kernel OS*

The T-Kernel OS is a specification for a real-time OS with system management and debugging support. The OS has all the task management, interprocess communication, memory management, and system control functions present in any standard real-time OS. The T-Kernel OS contains the OS functions shown in Table 2.

| <b>Function Area</b>              | <b>Explanation</b>  |
|-----------------------------------|---|
| Task management                   | Manipulate or reference task states                       |
| Task-dependent synchronization    | Synchronize tasks by changing task states                 |
| Task exception handling           | Handle task exceptions                                    |
| Synchronization and communication | Inter-task communication and synchronization              |
| Memory pool management            | Software management of memory pools and memory allocation |
| Time management                   | System time management, cyclic and alarm handlers         |
| Interrupt management              | Define and manipulate interrupt handlers                  |
| System management                 | Change and reference system status                        |
| Subsystem management              | Define, use, and get status for subsystem handlers        |

**Table 2: T-Kernel OS Functions**



*A full Instruction Set Simulator accelerates early development*

The T-Kernel OS is similar to  $\mu$ ITRON with the addition of a mechanism to manage subsystems and support for a memory protected, process-based OS. The kernel in T-Kernel differs from  $\mu$ ITRON in a couple of significant ways: memory allocation and the model to call kernel APIs. T-Kernel has dynamic memory management for kernel objects. All kernel object IDs (such as task, queue, semaphore, etc) are assigned dynamically, and the memory for the kernel object is dynamically allocated from a memory pool. The T-Kernel OS does not implement direct function calls. It instead implements a software exception model for system calls. This model is the basis for process model support.

*T-Kernel System Manager*

T-Kernel provides a System Manager (SM) for system services. The SM manages T-Kernel OS resources. Using these system services make it easier to port middleware between different T-Engine systems.  $\mu$ ITRON did not have the system services as shown in Table 1.

|  |
|--|
| System memory management                           |
| Address space management                           |
| Device management                                  |
| Interrupt management                               |
| I/O port access                                    |
| Power management                                   |
| System configuration and information management    |
| Entry processing for device drivers and subsystems |

**Table 1: T-Kernel System Manager Services**

*T-Kernel Debugger Support*

The T-Kernel Debugger Support (DS) standardizes interaction with debuggers. The DS interface is internal to T-Kernel, and provides debugging information such as the current state of kernel objects, and information for tracing program execution. Applications do not call the DS directly. The term “T-Kernel OS” sometimes refers to the sum of the OS, SM, and DS.

*T-Kernel Device Drivers*

T-Engine also contains a device driver specification for both the



driver interface, and actual drivers. Device drivers are required for all standard T-Engine board peripherals.

*T-Engine Subsystems*

T-Engine subsystems extend T-Kernel by providing a standardized interface for middleware. A file system or TCP/IP stack, for example, are provided as T-Kernel subsystems. A subsystem is a shared library with a specified interface. T-Kernel calls this interface an “extended SVC,” where SVC refers to “service calls.” Applications call into the extended SVC, and do not generally call T-Kernel APIs directly. Subsystems are dynamically loadable, and are the basis for the reusable software concept in T-Engine.

*Standard Extension*

The T-Kernel Standard Extension specifies services for process-based applications. When using the Standard Extension, application processes must call its interface instead of calling the T-Kernel OS/SM APIs directly. The Standard Extension includes a program loader to load processes, subsystems, and device drivers. The Standard Extension also specifies a file system subsystem. The SE provides the services listed in Table 2.

| Service                                | Description   |
|--|---|
| Memory management                      | Local process, shared, and system memory                  |
| Process/task management                | Local process services (i.e. APIs for task in process)    |
| Interprocess messaging                 | Signal-like messaging. Can register message handlers      |
| Global name management                 | Name service for interprocess data sharing                |
| Task synchronization and communication | Intra and interprocess synchronization and communication  |
| Event management                       | Device event management (for user interface)              |
| Device management                      | Device driver interface                                   |
| Time management                        | System time access, conversion for calendar time          |
| System management                      | Functions to load drivers/subsystems; OS status functions |

**Table 2: T-Kernel Standard**



*T-Kernel scales:  
use the same  
code in a variety  
of applications*

*eT-Kernel:  
optimized, fully  
supported T-  
Kernel  
implementation*

### ***T-Engine Scaling and Programming***

T-Kernel supports using an MMU for memory protection. T-Kernel applications can scale from using flat memory space with no memory protection to full memory protected processes. Non-MMU applications use linear memory and a physical address space. Applications following this model are very similar to  $\mu$ ITRON applications because they call the kernel API directly. T-Kernel applications may also use the MMU, but still call the kernel APIs. In this configuration, the application remains relatively simple, but uses the MMU to protect resources. In the full process-based case, applications can run as processes that are dynamically loaded to the system. This scaling is an advantage because applications can use the same basic OS in a variety of products.

### ***Software Reuse***

Subsystems are the T-Kernel model for reusing software. Applications can link with subsystems statically, or dynamically load them using the program loader. Subsystems developed for a flat memory model can be reused without modification with the T-Kernel Standard Extension. Programmers can add new features available within the SE to subsystems, but this is not required.

This subsystem model means that any existing kernel application can be modified into a subsystem and run under T-Kernel. T-Engine's modular design means that subsystems are independent of CPU and board. As a result, middleware moves easily from platform to platform as needed.

### **eT-Kernel**

The T-Engine standards fully define the T-Engine board specification and the T-Kernel OS specification. The T-Engine forum provides an open-source version of the T-Kernel OS. eSOL has optimized the T-Kernel OS and provides a scalable solution with eT-Kernel/Compact, eT-Kernel/Standard, and eT-Kernel/Extended. eSOL's eT-Kernel family provides a highly robust, widely applicable development platform, with capabilities ranging from small and fast  $\mu$ ITRON-like configuration to a full-fledged process based system similar to Linux. This section provides an overview of eT-Kernel.



### *Configuration*

The T-Kernel specification requires all features are available in the RTOS. eT-Kernel provides configuration switches to allow the user to easily choose features suitable for the target platform. eT-Kernel still satisfies the T-Engine requirement that the OS implements all features, but additionally allows the user to easily choose whether to use each feature. Using the System Manager to manage I/O is another important example of the importance of configuration. If a customer is ready to transition to T-Kernel, but is not yet ready to use the I/O manger, eT-Kernel makes it possible to swap out the SM. Removing the SM reduces the memory footprint, eliminates the SM initialization, and allows customers to use their own I/O model, if desired.

### *Migration from $\mu$ ITRON*

eT-Kernel makes migration from  $\mu$ ITRON easy by providing configuration switches to make eT-Kernel  $\mu$ ITRON-compatible. Developers can start the migration process by turning on all the compatibility switches. As the application requirements grow more complex, developers can start using T-Kernel features through configuring eT-Kernel.

### *T-Monitor*

eT-Kernel implements the following parts of T-Monitor:

- Board initialization
- CPU initialization
- Exception vector handling
- Debug utilities: command line interface, memory dumps

eBinder takes the place of start up functions in T-Monitor. eSOL has optimized T-Monitor for speed and size.

### *MMU Handling*

The T-Kernel Standard Extension specification is only available in beta form to T-Engine “A+” members. Therefore, the open source T-Kernel does not yet implement the process model because the standard for it is not yet public. eSOL has implemented the standard extension, and a full process-model version of eT-Kernel is available.



***eT-Kernel: Management Platform for Entire Product Line***

The T-Kernel solution provided by eSOL not only improves performance, but also provides transparent scaling. eSOL offers three versions of eT-Kernel. Each version satisfies requirements for a specific class of applications. Among the three versions, one will provide a solution for almost any type of application. eSOL's eT-Kernel comes as eT-Kernel/Compact, eT-Kernel/Standard, and eT-Kernel/Extended.

***eT-Kernel Compact***

eT-Kernel/Compact contains the eT-Kernel OS, System Manager (SM), and Debugger Services (DS). eT-Kernel/Compact is the OS most comparable to  $\mu$ ITRON OSes such as PrKERNELv4 because it uses a flat memory space, and does not use the MMU. This version suits users who are satisfied with the features in  $\mu$ ITRON version 4, but who see the need for more features and scalability in the future. eT-Kernel uses a software exception interface to kernel functions. eT-Kernel/Compact has a configuration switch to make it completely compatible with PrKERNELv4. This configuration switch extends even to using the PrKERNELv4 direct function call interface instead of the software exception interface. Additionally, when the kernel core assumes support of the process model, additional code must exist to handle the MMU. eT-Kernel/Compact removes the need for MMU support at the kernel level when the Standard Extension is not used. This optimization improves kernel performance and reduces code size.

***eT-Kernel/Standard***

The T-Engine specification requires the kernel to implement a standard set of device drivers. eT-Kernel/Standard contains eT-Kernel/Compact and the T-Engine device drivers. Real reference drivers are provided with eT-Kernel/Standard so developers can get started quickly. eT-Kernel/Standard is otherwise equivalent to eT-Kernel/Compact.

***eT-Kernel/Extended***

eT-Kernel/Extended is a single user, multi-process operating system based on the T-Kernel specification and the Standard Extension. eT-Kernel/Extended uses the MMU to separate the overall application into a kernel layer and a process application layer. The process application layer follows the UNIX process model where each process has its own memory space. Processes cannot call eT-Kernel APIs directly, but must do so through a Standard Extension interface. eT-Kernel incorporates a POSIX-



Standard Extension interface. eT-Kernel incorporates a POSIX-compliant file system. eT-Kernel/Extended can load processes and system programs.

eT-Kernel is a scalable platform that can be a base for managing an entire product line of embedded devices.

- The same kernel can be used for systems from small to large – the drivers and middleware are reusable.
- Applications developed for eT-Kernel/Compact can be reused in the process model under eT-Kernel/Extended.
- eT-Kernel supports all major embedded CPU architectures.

### ***eBinder IDE***

The open source T-Kernel requires using a gcc/gdb development environment. The GNU tools run on a Linux workstation or on Windows, using a GNU utility, such as Cygwin. While gdb debugging may be sufficient for simple applications, it does not meet the requirements for debugging high-end, complex applications. Currently, gdb is the only choice for debugging the open-source T-Kernel. eSOL integrates eT-Kernel with the eBinder IDE to provide a complete development package. The eBinder IDE combines an integrated development environment, multi-threaded debugging tools, and real-time debugging capability with integrated support for eT-Kernel operating systems. Using eBinder, developers can debug individual tasks or individual processes. The main application and system can run while stopping only the task or process that needs debugging. eBinder also has powerful analysis tools to examine the state of the system and analyze performance.

### **Conclusion**

The T-Engine forum created a new standard that extends the successful  $\mu$ ITRON specification into the future. New customer requirements and the evolving embedded industry drove the need for a new standard. The T-Engine and T-Kernel standards were crafted to specifically meet those needs. eSOL is one of the first vendors to provide a full T-Kernel operating system, and is the first to provide a full Integrated Development Environment for T-Kernel. The complex applications made possible with the T-Kernel standard need a complete development and debugging tool like eBinder to fully realize T-Kernel's potential.



Founded in 1975 in Tokyo, Japan, eSOL is a leading embedded software developer with core technologies in real time operating systems. We develop, market and sell proven RTOS suites, along with a rich set of vertical oriented middleware libraries. Our rugged software development tools provide optimal reliability in backing up the highly complicated development process for RTOS-based applications. We know that a reliable RTOS and development tools make a significant difference to the quality and timeliness of our customers' products in a continuously growing and competitive world market. Today, our customers - global OEMs and ODMs ranging from consumer electronics to automotive applications - ship millions of products with technologies pioneered by eSOL.

For more information, please visit <http://www.esol.com/>.

**eSOL Co., Ltd.**  
**Embedded Products Division**

Harmony Tower, 1-32-2 Honcho  
Nakano-ku, Tokyo 164-9721, Japan  
Tel: +81 3-5302-1360 Fax: +81 3-5302-1361  
[ep-info@esol.co.jp](mailto:ep-info@esol.co.jp)