*ITRON Internatinal Meeting '99*

# Recent Activities of the ITRON Project

***29th Sep. 1999***

**Hiroaki Takada**

ITRON Committee, TRON Association
Toyohashi Univ. of Technology

ITRON Project Home Page
http://www.itron.gr.jp/

# ITRON Project – *2nd Phase*

- a project to standardize RTOS and related specifications for embedded systems

*1st Phase* (1984– )

- focused on *real-time kernel* specifications

*2nd Phase* (1996– )

- broaden the scope of the standardization effort to related aspects
    - *software components* (software IP)
    - *development environments*
    - *application-specific standards*

↓

*Several standardization activities are in progress.*

# 2nd Phase Activities

## Preconditions for software components

- µITRON4.0 Specification *released in June. 1999*
- Conformance Testing Method *near future*
- Application Design Guidelines

## Standard API for software components

- ITRON TCP/IP API Specification *released in May 1998*
- JTRON2.0 Specification *released in Oct. 1998*
- Device Driver Design Guidelines *current*

## Development environments

- µITRON4.0 Debugging Interface Specification *current*
- C++/EC++ Language Binding *near future*

## Application-specific standards

*reflected to µITRON4.0*

- RTOS for Automotive Control Application

# µITRON4.0 – What and Why?

- µITRON4.0 is the next generation µITRON real-time kernel specification.

## Why it is necessary?

- raising software portability
  - Our “*loose standardization*” policy often condradicts with software portability.
- functions for independently-developped software components
- incorporating the results of our recent investigations
  - hard real-time system supports
  - requirements for *automotive control applications*
- following the advancement of microprocessor technology

# Portability *vs.* Adaptability

- Software portability can be raised if we define the kernel functions more strictly.
- Adaptability (*incl.* scalability) is the most important advantage of μITRON, and we should keep it.

*standard profile*

- the set of kernel functions *strictly* defined for raising software portability

μITRON4.0 — loose standardization
standard profile — strict standardiation

- *Subsetting* is still acceptable for small systems.
- *Extended functions* are also defined.

# Standard Profile – Overview

## Target System

- target processor: high-end 16bit to 32bit
- kernel size: 10KB to 20KB with all functions
- The whole software is linked to one module.
- Kernel objects are statically defined.

## Function Overview

- including almost all level S functions of μITRON3.0
- incorporating some level E functions of μITRON3.0
- some newly introduced functions
- some modifications and more strict definitions based on μITRON3.0

## Standard Profile – Function Overview (1)

### Level S of μITRON3.0

- basic task management and synchronizations
- semaphore, eventflag, mailbox
- interrupt management, basic time management

### From Level E of μITRON3.0

- fixed-sized memory pool, cyclic handler
- service calls with timeout

### Major Modifications / More Strict Definitions

- `act_tsk` with queueing instead of `sta_tsk`
- some terminologies and service call names
- how to write an interrupt handler in C language
- service calls used in an interrupt handler

# Standard Profile – Function Overview (2)

## Newly Introduced Functions

- data queue (queue for one word messages)
- exception handling mechanism

  task exception routine, CPU exception handler
- system state reference
- `can_act`, `isig_tim`

## Static API

- Standard description (in a system configuration file) for defining kernel objects statically.

  cre_tsk(…) — service call for creating a task

  CRE_TSK(…) — *static API* for creating a task

  *common parameters*

# Broader Scalability

## New Functions not Included in μITRON3.0

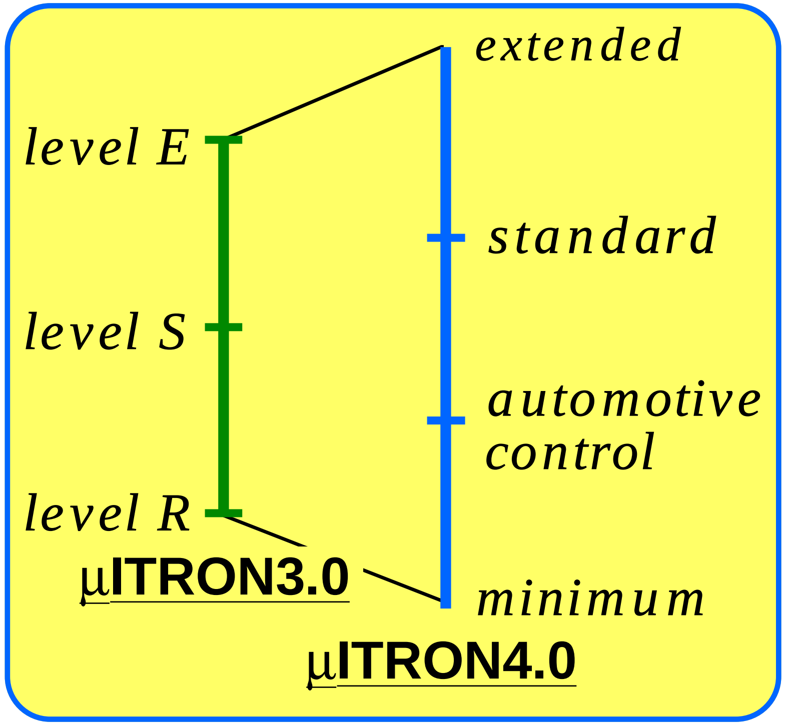
- data queue
- task exception handling
- system state reference
- interrupt service routine
- hard real-time support
- automatic ID assignment

## Automotive Control Profile

- a smaller profile definition especiall suitable for automotive control applications

## Minimum Requirements

- Dormant state instead of waiting state is mandatory.



# Functions Supported in μITRON4.0 Specification

- task management
- task-dependent synchronization
- task exception management
- basic synchronization and communication
  ( semaphore, eventflag, data queue, mailbox )
- extended synchronization and communication
  ( mutex, message buffer, rendezvous )
- memory pool management
  ( fixed-sized, variable-sized )
- time management
  ( cyclic handler, alarm handler, overrun handler )
- system state management
- interrupt management
- service call management
- system configuration management

# Debugging Interface Specification

! Currently, each debugging tool (debugger, ICE, etc.) must support each µITRON-specification kernel separately.

⬇

- With standardized interface between µITRON-specification kernels and debugging tools, µITRON support becomes easy.

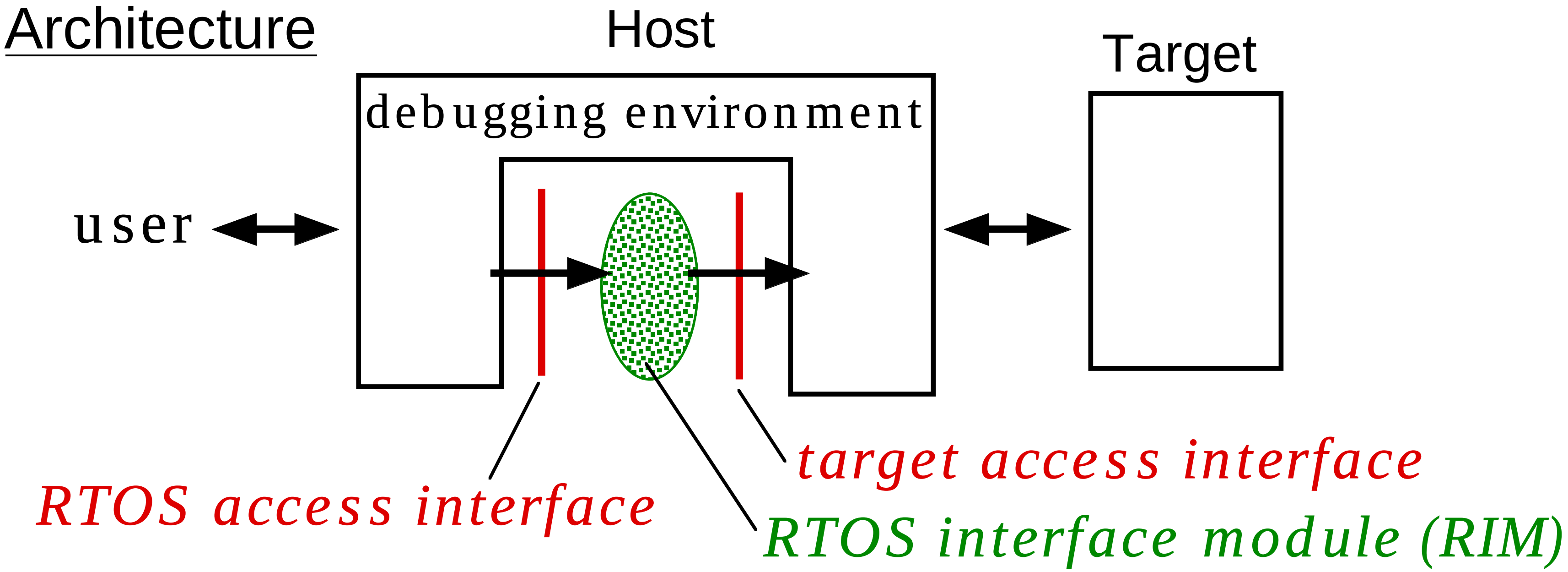
## Scope of the Specification

- the interace between µITRON-specification kernels and the *RTOS-support functions* of debugging tools
  - kernel object state reference
  - task-aware breakpoint and stepping
  - kernel trace *etc.*

## Goal

- Run-time overhead should be minimal.
- Most part of the specification should be common to different kind of debugging tools (debugger, ICE).
- The basic concept/architecture should be applicable to other RTOS and software components.

## Architecture



# Schedule

- µITRON4.0 Specification
  - Ver. 4.00.00 released in June 1999
  - now translating into English (takes about a half year)
- µITRON4.0 Debugging Interface Specification
  - Prelimimary “request for comment” version will be released soon. The validation stage will follow.
  - proposing the joint work with the OSEK/VDX Project
- ITRON TCP/IP API Specification
  - Ver. 1.00.01 released in May 1998
  - Preliminary English version is now ready.
- JTRON2.0 Specification
  - Ver. 2.00.00 released in Oct. 1998
  - English version has been just released.

# ITRON TCP/IP API Specification

- TCP/IP protocol stack is one of the most important software components, today.
- The socket interface is *not suitable* for (*esp*. small-scale) embedded systems.
  - necessity of dynamic memory management within the protocol stack
    - → *Errors occurred within the protocol stack is not notified to the application.*
  - difference between UNIX process model and ITRON (*or* real-time kernel) task model

↓

- Standard TCP/IP API suitable for embedded system is required.

ITRON TCP/IP API Specification

*approach:*

- based on the socket interface
- The socket interface can be implemented as a library on the new API.

*differences with the socket interface:*

- TCP API and UDP API are separately defined.
- "*End point*" abstraction is adopted instead of "*socket*" abstraction. TCP end point for waiting for connection requests and TCP connection end point are handled as different objects.
- TCP APIs for reducing data copies are also defined.
- Non-blocking calls and callbacks are supported.
- The callback routine is used for receiving UDP packets.

*etc.*

# JTRON Specification

- a practical approach for applying Java technology to embedded real-time systems
  - implementing Java runtime environment on real-time OS
  - taking the advantages of both environment

modules requiring real-time property

… *implemented on real-time OS*

downloadable module, GUI

… *implemented on Java runtime environment*

↓

- Communication interface between real-time tasks and Java applications should be standardized.

JTRON Specification

## *three types of communication interfaces:*

Type 1: *attach classes*

- Java applications can access real-time OS resources through attach classes.

Type 2: *shared object*

- Real-time tasks can access shared objects exported from the Java application.
- explicit locking/unlocking mechanism
- Java application must explicitly call the unshare method on the object.

Type 3: *stream interface*

- Real-time tasks and Java applications can communicate through stream interface.

# Other Recent Activities

- standardization activities targeted for automotive control applications

*finished*

↓ *the first application-specific activity*

Requiremens on real-time kernel in automotive control applications have been clarified.

→ *reflected to μITRON4.0*

- device driver design guidelines
  - hierarchical model of DIC (device interface component)
  - design guidelines for the portability of DIC
- debugging interface of real-time kernel
  - standard interface between ITRON-specification kernel and debugging tools, including software debuggers, ICE, and logic analyzer