ITRON Newsletter No.11

ITRON Technical Committee, TRON Association
Katsuta Building 5F, 3-39, Mita 1-chome, Minato-ku, Tokyo 108, JAPAN
TEL: (03) 3454-3191 FAX: (03) 3454-3224

ITRON-related Publications

Listed in another page are the publications prepared and issued by the ITRON Technical Committee as of October 1, 1994. The ITRON-μITRON Standard Handbook is a one-volume compilation of μITRON (Ver 2.0) and ITRON2 specifications. Each of the publications below can be obtained directly from the sources indicated.

The latest version of μITRON3.0 is now Ver 3.02.00. Changes made since the μITRON3.0 Standard Handbook was released (Ver 3.00.00) are noted in Newsletter No.5 and here below.

The ITRON Standard Guidebook '92-'93 still applies to users of μITRON (Ver 2.0) and ITRON2 specifications, even though the dates in its title are now past. A new edition of the ITRON Standard Guidebook, focusing primarily on the μITRON3.0 specification, is now being prepared.

Revisions to μITRON3.0 Specification Issued

Revisions have been made to the μITRON3.0 specification to correct problems discovered since the issuing of the μITRON3.0 Standard Handbook. The changes are summarized in the table below. The new version number is 3.02.00.

Free Software Available

The software introduced below is not sold commercially, and is therefore not registered with the TRON Association. Instead it is available free by downloading from NIFTY-Serve or the Internet, as explained below.

ASURA-I

Mineyuki Kimoto

ASURA-I is a μITRON3.0-specification OS that runs as a resident program in MS-DOS. Since it is loaded as a device driver, μITRON system calls can be used not only by ordinary applications, but also by TSR or other device drivers. The present version does not implement timer-related functions, since it is intended for general-purpose use with DOS; but there are plans to add this support in the future.

One more feature of ASURA-I is its adoption of a BTRON real/virtual object network in its source code. The real/virtual object source list is not contained in the DOS archive, but it is found in the provisional documentation “Inside of ASURA-I.”

Applications are bundled with this software for displaying the states of each object, and for simple CLI work area testing. Include files for an assembler are also provided.

ASURA-I can be downloaded from the Personal Media Forum (FPMC) in the NIFTY-Serve on-line service, or from Internet ftp servers at the University of Tokyo and Waseda University. The ftp addresses are: utsun.s.u-tokyo.ac.jp [133.11.11.11] and ftp.waseda.ac.jp [133.9.1.4].

Recent Works on ITRON

The Information Processing Society of Japan devotes the October 1994 issue of its monthly journal (Vol.35, No.10) to a series of articles on the present status and future outlook of the TRON Project. The ITRON subproject is covered in an article entitled, “The Present and Future of the ITRON Subproject – Kernel Specifications and their Implementation,” written by the University of Tokyo's Hiroaki Takada, Toshiba's Kiichiro Tamaru, and other members of the ITRON Technical Committee.

The article starts by explaining the present situation in the field of small-scale embedded systems, and outlines the design policy adopted for the ITRON specifications. It then reports on implementations and applications of ITRON-specification OSs, and discusses the benefits gained from standardization. Next, the authors present a technical introduction to the real-time kernel implementation technologies de-

† This newsletter is reprinted from TRONWARE vol.30 and TRON PROJECT BIMONTHLY No.35.

ITRON-related Publications

Name	Type	Price	Publisher	ISBN No.
ITRON-μITRON Standard Handbook	Specification (Japanese)	4,800Yen	Personal Media Co.	4-89362-079-7
μITRON3.0 Standard Handbook	Specification (Japanese)	4,000Yen	Personal Media Co.	4-89362-106-8
ITRON/FILE Standard Handbook	Specification (Japanese)	3,000Yen	Personal Media Co.	4-89362-092-4
ITRON Standard Guidebook '92-'93	Textbook (Japanese)	3,500Yen	Personal Media Co.	4-89362-197-6
μITRON Specification Ver 2.01.00.00	Specification (English)	12,000Yen	TRON Association	–
ITRON2 Specification Ver 2.02.00.10	Specification (English)	15,000Yen	TRON Association	–
μITRON3.0 Specification Ver 3.00.00	Specification (English)	–	TRON Association	–

NOTES:
- Prices do not include consumption tax.
- The documents issued by the TRON Association are available to Association members at a special discount rate.
- English-language specifications are distributed free of charge on the Internet as explained in Newsletter No.8.

Revisions to the μITRON3.0 specification (Ver 3.01.00 $\rightarrow$ Ver 3.02.00)

1. In the System Call Index on p.VIII, the page number for `cre_sem` is corrected as follows (p.VIII).

Wrong:	`cre_sem`	[EN]	Create semaphore	11
Correct:	`cre_sem`	[EN]	Create semaphore	111

2. The following clarification is added to Fig.3: ITRON State Transition Diagram (p.27).

   * This state transition diagram is only intended to illustrate typical state transitions. Depending on the implementation, state transitions not shown here may occur as well.

3. The following correction is made on p.27, the second line from the bottom.

   Wrong: $\cdots$ the execution of interrupt A, parts ②–③ $\cdots$
   Correct: $\cdots$ the execution of interrupt X, parts ②–③ $\cdots$

veloped through the ITRON subproject. They explain the μITRON3.0 specification with its added support for distributed systems, and conclude by looking ahead to the future of the project. The article as a whole does not presuppose knowledge of the ITRON subproject and is therefore a useful introduction for those reading about the project for the first time, or those with little detailed prior knowledge.

Revisions to the μITRON3.0 specification (Ver 3.01.00 → Ver 3.02.00) (Cont.)

4. The following section is added to 1.6 Basic ITRON Concepts, following the section on the "Task-Independent and Quasi-Task Portions."

   Task States During a Dispatch Delay
   Dispatching of tasks in ITRON may be temporarily disabled, because of the principle of delayed dispatch, or when a system call is issued that prohibits dispatching. During that time, task states are as explained below.
   During the time dispatching is disabled, even if a situation occurs that would normally call for a running task to be preempted, the task that would preempt it is not dispatched. The dispatching of the latter task is delayed until the dispatch disabled state is cleared. While the dispatch is delayed, the running task remains in RUN state, and the task to be executed after the dispatch disabled state is cleared is treated as in READY state. As for the sequence of tasks in RUN state in the ready queue during a dispatch delay, this is an implementation-dependent matter.
   If, while dispatching is disabled, a `sus_tsk` system call is issued for the running task to put it in SUSPEND state, or if `ter_tsk` is issued to put it in DORMANT state, the task transition is delayed until the dispatch disabled state is cleared. During this time, the running task is considered to be in a transient state, with the specific handling of this status an implementation-dependent matter. Here too, the task to be executed after the dispatch disabled state is cleared is treated as in READY state.

5. In the explanation of `dis_dsp`, the following explanation replaces the earlier one for the first point on operation during dispatch disabled state (p.74).

   - Even in a situation where normally a task executing `dis_dsp` should be preempted by a system call issued by an interrupt handler or by another task executing `dis_dsp`, the task newly going to executable state is not dispatched. Instead, dispatching of this task is delayed until the dispatch disabled state is cleared by `ena_dsp`.

6. In the explanation of `dis_dsp`, the following explanation is added to the second point on operation during dispatch disabled state (p.74).

   - If an interrupt handler invoked during dispatch disabled state issues `sus_tsk` for a running task (one that executed `dis_dsp`) to put it in SUSPEND state, or `ter_tsk` to put it in DORMANT state, the task transition is delayed until the dispatch disabled state is cleared.

7. The following explanation replaces the second paragraph in the explanation of `rot_rdq` (p.81).

   When `rot_rdq` is issued by the task portion, the ready queue with the priority of the calling task is rotated by means of `tskpri`=`TPRI_RUN`=0.

8. The last paragraph in the explanation of `rot_rdq` is made a supplementary explanation. Also, the following is added as a supplementary explanation for `rot_rdq` (p.82).

   Depending on the implementation, it may be possible to issue `rot_rdq`(`tskpri`=`TPRI_RUN`) from a task-independent portion, such as a cyclic start handler. In this case the ready queue containing the running task, or the ready queue containing the highest priority task, is rotated. Normally these two are the same, but not always, as when the task dispatch is delayed. In that case it is implementation dependent whether to rotate the ready queue containing the running task or the ready queue containing the highest priority task. Note that this is an extended function [Level X], for which compatibility and connectivity are not guaranteed; so it cannot always be used.

Revisions to the μITRON3.0 specification (Ver 3.01.00 → Ver 3.02.00) (Cont.)

9. The following replacement is made for the first sentence in paragraph three giving the reasons for specification decisions for `rsm_tsk` (p.100).

   From a specification standpoint, even if a given task is in SUSPEND state, preferably there should be no influence on the scheduling sequence, as also with preemption.

   The following replacement is made for the second sentence in paragraph four giving the reasons for specification decisions for `rsm_tsk` (p.100).

   Hereafter "ready queue" means a ready queue in the implementation sense, as distinct from a ready queue as defined in the specifications, in the sense of including only tasks in RUN state and tasks in READY state.

10. The following replacement is made for paragraph two of the explanation of `can_wup` (p.105).

    The calling task can be designated by means of `tskid`=`TSK_SELF`=0. An `E_ID` error will result, however, if `tskid`=`TSK_SELF`=0 is designated with a system call issued from a task-independent portion.

11. The following supplementary explanation is added at `del_sem` (p.115).

    [Supplementary Explanation]
    When a semphore being waited for by more than one task is deleted, the sequence of tasks in the ready queue after the wait state is cleared is implementation dependent in the case of tasks having the same priority.

    A similar explanation is added at `del_flg`, `del_mbx`, `del_mbf`, `del_por`, `del_mpl`, `del_mpf`.

12. The following addition is made to the fourth sentence in the last paragraph of the explanation of `set_flg` (p.132).

    In this case, the sequence of tasks in the ready queue after the wait state is cleared is guaranteed for tasks having the same priority.

13. The explanation of `E_PAR` error under the `snd_msg` error codes is corrected as follows (p.149).

    `E_PAR` parameter error (value unusable with `pk_msg`, or illegal parameter (`msgpri`, etc.) in message header)

14. The following replacement is made for sentence one in paragraph six of the `def_int` explanation (p.212).

    Even when the issuing of a system call in an interrupt handler results in a situation where a task in RUN state should go to another state and another task should go to RUN state, no dispatch (switching of the running task) takes place while the interrupt handler is running.

    A similar revision is made in the explanations at `def_cyc` and `def_alm`.

15. The following replacement is made in the `loc_cpu` explanation for the second point on operations during an interrupt and dispatch disabled state (p.220).

    - Even when a task that executed `loc_cpu` is in a situation where it should be preempted because of a system call issued by a task that executed `loc_cpu`, the task newly going to executable state is not dispatched. Instead, dispatching of this task is delayed until the dispatch disabled state is cleared by `unl_cpu`.

Revisions to the μITRON3.0 specification (Ver 3.01.00 → Ver 3.02.00) (Cont.)

16. The following addition is made to the supplementary explanation for the `def_cyc` system call (p.275).

    In an implementation allowing `def_cyc` to be called by a timer handler, it is possible to redefine a cyclic start handler with the same number in the handler.

17. In the addition to the supplementary explanation of the `def_alm` system call that was added as of Ver 3.01.00 (see Newsletter No.5), the following replacement is made (p.285).

    At the time an alarm handler is started, the definition of that handler is considered as already canceled. Thus if `ref_alm` is used to refer to information on the started handler in a handler, a `E_NOEXS` error will result. Also, in an implementation allowing `def_alm` to be called by a timer handler, it is possible to redefine an alarm handler with the same number in a handler.

18. In the explanation of `get_ver`, the following revision is made in the third line from the top of p.295.

    Wrong: ··· is common to ITRON and BTRON.
    Correct: ··· is common to ITRON, μITRON, and BTRON.

19. In the explanation of `ref_sys`, the following revision is made in the first line on p.298.

    Wrong: ··· execution state of the CPU and μITRON3.0 ···
    Correct: ··· execution state of the CPU and OS ···

20. In the explanation of `def_svc`, the following revision is made in the eighth line from the top of p.303.

    Wrong: - the indivisibility of system calls
    Correct: - the indivisibility of an extended SVC handler

21. The following addition is made at the end of the section on high-level-language-support routines, in section 5.2 (p.356).

    Note that if routines written in C-language functions are called using the same interface as assembly routines, use of high-level-language-support routines is not necessarily required even when `TA_HLNG` is designated.

22. The following replacement is made in 6.4 Data Types, in the section on "Universal data types" (p.376).

B	Signed 8-bit integer
H	Signed 16-bit integer
W	Signed 32-bit integer
UB	Unsigned 8-bit integer
UH	Unsigned 16-bit integer
UW	Unsigned 32-bit integer
VB	Data of indeterminate type (8-bit size)
VH	Data of indeterminate type (16-bit size)
VW	Data of indeterminate type (32-bit size)
VP	Pointer to data of indeterminate type
FP	Program start address (in general)

23. The section title on p.384 is revised as follows.

    Wrong: 6.5 List of C-language Interfaces
    Correct: 6.6 Common Constants and Packet Format of Structures