By Frank Büchner,
SW Product Manager,
Hitex Development Tools.

OSEK/VDX Debugging - Racing on to Becoming a Standard

Standards relating to the OS, COM, NM and OIL areas of the OSEK/VDX project have already been accepted and partly subject to several revisions. However, no standard currently exists for the testing and debugging of OSEK applications. In response to urgent requests from OSEK users, OSEK operating system manufacturers and the manufacturers of test tools for embedded applications have agreed on a method to be used for the testing and debugging of OSEK applications.

A TEST METHOD FOR OSEK APPLICATIONS

This method is based on a file known as the OSEK Run-Time Interface (ORTI) file, which contains the OSEK specific debug information of an OSEK application. The file is generated together with the OSEK application and it can then be used by software test tools, providing them with all the test, debug and analysis features necessary for the particular OSEK application being tested and debugged.

The ORTI File Method has the advantage of being portable, since the ORTI file is simply a standard text file that can be used on every host operating system (Windows, Unix). Furthermore, this text file can be used as a simple and easy to examine interface between the OS manufacturer, the creator of the ORTI file and the test tool manufacturer who will eventually use the file. Any interface problems that do occur are then easy to detect and analyze.

In the meantime, the ORTI File Method has gained the support of several manufacturers of OSEK operating systems and several test tool manufacturers and it has established itself as the accepted de-facto standard.

THE CONTENT OF AN ORTI FILE

The ORTI file contains the address of an identifier in the target system's memory that always indicates which task is currently running (i.e. which task is in "Running state"). This address is normally not specified as an absolute value, but symbolically. E.g.

```
RUNNINGTASK = "RunningTaskId";
```

Thus, the test tool is required to convert "RunningTaskId" into an absolute address such as 0x1000 using symbol data from the application's debug information, originating from the compiler and linker.

In principle, every symbol found in the ORTI file, must be clearly capable of being converted into an address value using this debug information. This is not as easy as it may seem. The application must be created together with the debug information, and this debug information is to be preserved through all stages of generation. Even if the necessary debug information is assumed to be available, the use of symbols in the ORTI file should be considered carefully. As an example, the addresses of stack symbols are only valid for limited amounts of time and their evaluation at any

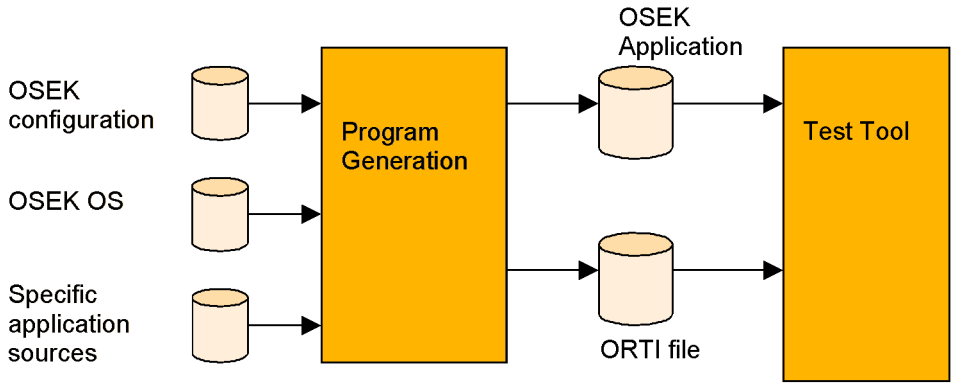

Figure 1. The Test Tool makes use of both the application and the ORTI file.

point in time would most likely result in false values. Thus, the use of stack symbols in ORTI files is to be avoided. Furthermore, symbols sharing the same name can sometimes show up in different modules. If such a "multi-occurring" symbol were used in the ORTI file, it would present the test tool with an unsolvable problem. Hence, the use of names occurring more than once in an application is also to be avoided in an ORTI file.

A test tool is unaware of the plain text name of the currently running task, even if it is aware of the address of the location containing the task's identifier. This is solved by informing the test tool which identifier (i.e. number) has been allocated to which name. This allocation information is also stored in the ORTI file.

```
E.g.     ENUM [ "NO_TASK" = 0, "MY_TASK" = 1,
         "YOUR_TASK" = 2] RUNNINGTASK ,
         "Name of the Running Task";
```

In the above example, there are only three values that can be used for identifying the running task. The ORTI file has already been created during generation of the OSEK application. As a pre-requisite, the number of tasks in a particular OSEK application is required to be known at the time of generation. Defining the number of tasks at this stage is made possible by the fact that OSEK is a static and not a dynamic operating system. If OSEK was not a static operating system; it would be impossible to use the ORTI File Method of testing.

The semantics of the three values used in the above example have not yet been specified in the current de-facto standard. However, if no task can be shown as the running task, it is convention to write a value of 0 to the relevant memory location and to name this non-task as NO_TASK.

WHAT TYPE OF ANALYSES DOES THE ORTI-FILE FACILITATE?

To ensure that the correct running task identifier is held in the relevant memory location at all times, the operating system writes the new identifier to this location with each task change. This results in a write cycle on the bus of the microcontroller on which the OSEK application is running.

Use of In-circuit Emulator Recordings to Analyze Operating System Events

If an in-circuit emulator is used to test and debug an OSEK application, the write cycle described above can be selectively recorded in the emulator's trace memory. The emulator will normally also record a timestamp with the write cycle. The trace memory will then contain a series of recordings, comprising of the value written (i.e. the identification of the task in running state) and the point in time at which this occurred. The data stored in the emulator's trace memory could then be used to provide answers to questions such as:

- In which sequence did tasks enter running state?
- How long (minimum time, maximum time and total time) did a task remain in running state?
- How often did a task enter running state?
- Which task has proportionally used up the most processor time?
- What is the resource usage of the complete system?

Task changes are only one example of operating system events that can be recorded in this way. Other examples of such events are calls to the operating system (i.e. system services) by a task, or the occurrence of interrupts.

The recording of calls to the operating system and subsequent returns can provide answers to questions such as:

- What operating system functions did a task use?
- What was the execution time of operating system functions?
- Which operating system functions were used most frequently?
- Which calls to the operating system used up the most processor time in general?

An In-circuit-Emulator Provides Zero Resource Usage

The in-circuit emulator recording described above uses no operating system resources in terms of memory and execution time. Since nearly all operating systems save the identification of a currently running task somewhere and perform the necessary write access anyway, the ability to record task changes normally comes at no extra cost. If the pre-requisites for recording a particular event in the operating system are not given initially, the additional consumption of resources is kept to a minimum, since the memory required amounts to one or two bytes and the additional write accesses are negligible.

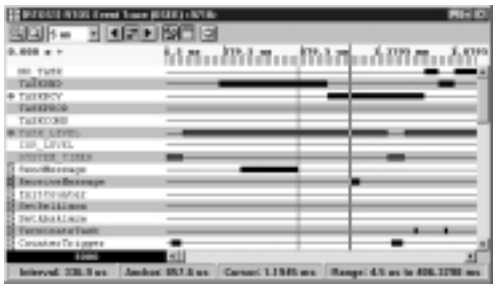

Figure 2. A depiction of operating system events in HiTOP's Event Trace.

The test tool is responsible for determining how data recorded in trace memory is evaluated and finally presented to the user as useful information. The Event Trace depiction in HiTOP, the user interface for Hitex in-circuit emulators, is an excellent example of a graphical representation of a trace recording. It provides a first class overview of operating system events.

WORKING GROUP PURSUES STANDARDIZATION

In order to bring OSEK debugging to a binding stan-

ITRON

The ITRON Project

ITRON is an abbreviation for, "Industrial TRON," where TRON stands for, "The Real-time Operating system Nucleus."

ITRON's Objective

The objective of the ITRON project is to set standards to specify real-time operating systems for the embedded world. In Japan, a great deal of real-time operating systems have already been realized according to the ITRON standard which appears to have developed into the de-facto standard for operating systems aimed at small embedded systems. The ITRON committee has thus expanded its activities to include related areas such as embedded systems design and the testing of embedded operating system applications.

Committee

Oki, Fujitsu, Hitachi, Matushita, Mitsubishi, NEC and Toshiba make up the ITRON committee. Participating research institutes include the University of Tokyo and the University of Toyohashi. Japanese companies dominate the group of "Observer Members," however a few American companies are also to be found here.

History

The ITRON-Project was started back in 1984. 1987 saw the emergence of ITRON1, the first specification. ITRON2 appeared in 1989 and was intended for 32-bit processors, while another specification, μITRON, was released in parallel for both 8-bit and 16-bit processors. In 1993, both specifications were merged into μITRON3.0, intended for all processors from 8-bit to 32-bit. μITRON4.0 was released in 1999, its main objective being to improve the portability of applications.

Scope of Use

A great deal of microcontrollers originating from Japanese manufacturers can be used with real-time operating systems whose kernels comply with the μITRON specification. Implementations for non-Japanese microcontrollers are limited to well-known architectures such as the 68000 and the 8086

dard, the, "OSEK/VDX Debug Interface Working Group," made up of members of the OSEK technical committee, has been working on a specification since October 2000.

OSEK and ITRON

Although the original intention was to involve the Japanese ITRON project in the OSEK standard, this has been put on hold for the time being. The main reason for this is that ITRON is based on a dynamic operating system (i.e. where the number of operating system objects is not defined at the time of generating the operating system) thus prohibiting use of the ORTI File Method. The ITRON project follows its own basic approach for the testing of ITRON applications, where an active program component is necessary because of the dynamic operating system. The operating system manufacturer supplies this active component, and it is integrated into the tool using defined interfaces so that interaction between the two is possible. Since an operating system and test tool do not usually originate from the same manufacturer, despite specification, interface problems are pre-programmed and their investigation can prove to be difficult. Furthermore, for program components written in C or C++, the problem of portability between different host operating systems arises (Windows, Unix). These issues have motivated the Debug Interface Working Group to continue pursuing the simple and portable ORTI File Method. If however the ORTI method reaches its limits in future, ITRON is definitely a replacement candidate.

Other reasons for initially not involving ITRON with OSEK are based on the current disjunctive state of affairs between them. ITRON is primary distributed in Japan and is rarely supplied to Europe. This also applies vice versa to OSEK. Furthermore, an ITRON OS or test tool manufacturer is usually unlikely to manufacture an OSEK OS, and this also applies vice versa.

Objective of the OSEK Working Group

The OSEK Debug Interface Working Group is currently pursuing the objective of transforming the de-facto standard into an accurate specification, whereby the functionality already available is not to be extended for the time being. One emphasis of current activity is the definition of semantics for the ORTI file content, which is inadequately specified in the de-facto standard and currently regulated by "silent agreement" between manufacturers of OSEK operating systems and OSEK test tools. An important near term objective is to create a reliable base for all parties involved in OSEK (OSEK manufacturer, OSEK user and test tool manufacturer).

OSEK's Outlook

As soon as the standard is accepted, the working group will most likely find itself faced with the task of enhancing the standard with additional functionality required. The identification of user requirements will then become a key aspect of the project ■

Frank Büchner studied Information Science at the Technical University of Karlsruhe. He has been working in the area of embedded systems for the past twelve years. He is presently working as a product manager for development tool software at Hitex, in Karlsruhe Germany and he is also a member of the OSEK/VDX Debug Interface Working Group.