OS Monitor V6.2
User's Guide

DRIVING | EMBEDDED EXCELLENCE

Copyright

Document EC012402 V6.2 R01 EN - 05.2013

Contents

1 Safety Advice

Please adhere to the Product Liability Disclaimer (ETAS Safety Advice) and to the following safety instructions to avoid injury to yourself and others as well as damage to the device.

1.1 Correct Use

ETAS GmbH cannot be made liable for damage which is caused by incorrect use and not adhering to the safety instructions.

1.2 Labeling of Safety Instructions

The safety instructions contained in this manual are shown with the standard danger symbol shown below:

⚠

The following safety instructions are used. They provide extremely important information. Read this information carefully.

⚠ **WARNING!**

Indicates a possible medium-risk danger which could lead to serious or even fatal injuries if not avoided.

⚠ **CAUTION!**

Indicates a low-risk danger which could result in minor or less serious injury or damage if not avoided.

NOTICE

Indicates behavior which could result in damage to property.

1.3 Demands on the Technical State of the Product

The following special requirements are made to ensure safe operation:

- Take all information on environmental conditions into consideration before setup and operation (see the documentation of your computer, hardware, etc.).

WARNING!

Wrongly initialized NVRAM variables can lead to unpredictable behavior of a vehicle or a test bench, and thus to safety-critical situations.

*ASCET projects that use the NVRAM possibilities of ASCET-RP targets expect a **user-defined** INIT process that checks whether all NV variables are valid for the current project, both individually and in combination with other NV variables. If this is not the case, all NV variables have to be initialized with their (reasonable) default values.*

*Due to the NVRAM saving concept, this is **absolutely necessary** when projects are used in environments where any harm to people and equipment can happen when unsuitable initialization values are used (e.g. in-vehicle-use or at test benches).*

Further safety advice is given in the ASCET V6.2 safety manual (`ASCET Safety Manual.pdf`) available on your installation disk, in the `ETASManuals\ASCET V6.2` folder on your computer or in the download center of the ETAS web site.

2 Operating System Monitor

This chapter mainly addresses ES1000 users who integrate their own VME bus cards in the ES1000 system with an ES113x simulation controller (e.g., ES1130 or ES1135). For this purpose, familiarity with the ES113x hardware is required.

2.1 Introduction

The use of the PowerPC experimental targets under ASCET in practice has shown that system crashes may occur under certain circumstances. Causes of these crashes can be the occurrence of exceptions (e.g. accessing an invalid address), the triggering of a watchdog reset or a timeout during the host-target communication.

To investigate the cause of a system crash, the operating system monitor and thus the possibility to analyze the crash cause later as well as to localize the error location within the program code has been implemented for usage with ERCOS^{EK} V4.1. For this purpose, the task and process sequences are monitored during the run-time by instrumenting the ERCOS^{EK} code. When the OS execution has stopped, this information can be analyzed.

This document is not relevant for projects using ERCOS^{EK} V4.3. In this case, OS monitoring is always on.

2.2 Implementation

In order to implement the operating system monitor, the ERCOS^{EK} code is modified such that task acitvations, process starts, process ends, task terminations and idle task procedures are continuously recorded and stored in a trace structure.

The trace calls are contained in the system library `a_system_gnu344.lib` delivered. However, they are encapsulated using the flag `osMonActiveFlag`. The operating system monitor is activated by setting the **`osMonActiveFlag`** parameter to **`1`** in the files for online experiment from the RAM or for standalone code in the flash. The names of these files are listed in Tab. 2-1

File Name	Experiment	Target	Directory
`a_es1130ra_gnu.loc`[a]	online experiment	ES1130	`legacy\ES1130`
`a_es1130ro_gnu.loc`[a]	standalone	ES1130	`legacy\ES1130`
`a_es1135ra_gnu.loc`[a]	online experiment	ES1135	`legacy\ES1135`
`a_es1135ro_gnu.loc`[a]	standalone	ES1135	`legacy\ES1135`

a. for GNU-Compiler

Tab. 2-1 Files for online experiments and standalone code

The directories are located beneath the ASCET target directory. The files are copied from there once to the ASCET subdirectory `CGEN` when the ASCET project is opened.

Note

It is recommended to make the modifications in the files listed in Tab. 2-1 only in the ASCET subdirectory CGEN unless a permanent activation of the OS monitor is desired.

2.2.1 Trace Structure

If, during the runtime of an online experiment, a exception occurs, the PowerPC computer node sends a specific L1 error message to the host (ASCET PC) via the L1 protocol[1]. Then, the error cause can be signaled in a debug window in plain text by ASCET.

In the process, the user needs to terminate the ASCET experiment. The target program itself performs an endless loop within the exception handler during which the watchdog is served cyclically. The user can now restart the online experiment. Before he clicks on the **Start OS** button in the "Physical Experiment" environment, he has to open the Target Debug window (**Tools** → **Target Debugger**).

Before the operating system is actually started, the trace information stored in the RAM is prepared and displayed in the Target Debug window. This display provides the following information:

- Board version.
- Cause of the trace display.
- Boot mode prior to the system crash/OP mode change.
- Display of the runtime and the idle time since the last OP mode change.
- List of the task activations (successful, failed, maximum number allowed).
- List of the last task/process runs.

2.2.2 System Utilization

The trace information is available not only after a system crash but also after ERCOS^{EK} has been stopped. This way, an estimate regarding the system utilization may be obtained, for example.

2.2.3 Runtime

Operating system monitoring requires additional runtime, of course, as can be seen from the folllowing overview.

- Recording a SW task activation: 4.0 µs
- Recording a HW task activation: 3.2 µs
- Recording a process start/end: 3.3 µs

The above values have been determined using a sample project. Since the task ID sometimes needs to be determined using a reference table during recording, the values can slightly vary depending on the specific project. In general, however, it is advisable to activate operating system monitoring only for troubleshooting purposes.

2.3 Application

Any working ASCET project must be bound and loaded into the flash after the `osMonActiveFlag` in the file `a_es113x_ro_gnu.loc` has been set to 1. This is required so that the trace information is saved and is thus available at the next experiment. Also, any existing flash code of older ASCET-RP versions might destroy the trace information.

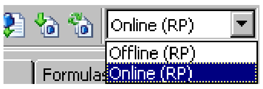
[1]. L1 = Protocol between host (PC) and target (PowerPC computer node)

To program the ES113x flash memory

- Open the ASCET project.
- In the ASCET subdirectory `CGEN`, open the appropriate `a_es113x_ro_gnu.loc` file.
- In the file, set the parameter `osMonActiveFlag` to 1.
- In the Project Editor, select **Build** → **Touch** → **Flat** to bind the model.
- In the Project Editor, select **Build** → **Flash Target** to store the model in the flash.

To troubleshoot with operating system monitoring

- Open the ASCET project.
- In the ASCET subdirectory `CGEN`, open the appropriate `a_es113x_ra_gnu.loc` file.
- In the file, set the parameter `osMonActiveFlag` to 1.
- In the Project Editor, select **Build** → **Touch** → **Flat** to bind the model.

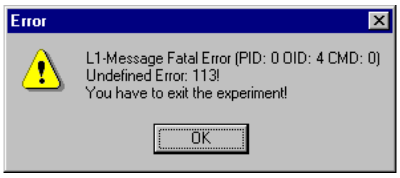


- Select `Online (RP)` from the "Experiment Target" combo box.

  `Offline (RP)` is intended for offline experiments on the Target.
- Select **Build** → **Experiment** to start the online experiment.
- In the "Physical Experiment" window, select **Tools** → **Target Debugger** to open the Target Debug window.

  When the operating system stops (**Stop OS** button), the last piece of trace information is displayed in this window.
- In the "Physical Experiment" window, select **Experiment** → **Start OS**.

  In case of an error, the message "L1 Message Fatal Error" is displayed which also includes information on the error cause. Furthermore, you are prompted to exit the experiment.

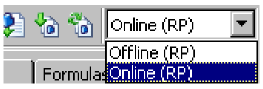


Error

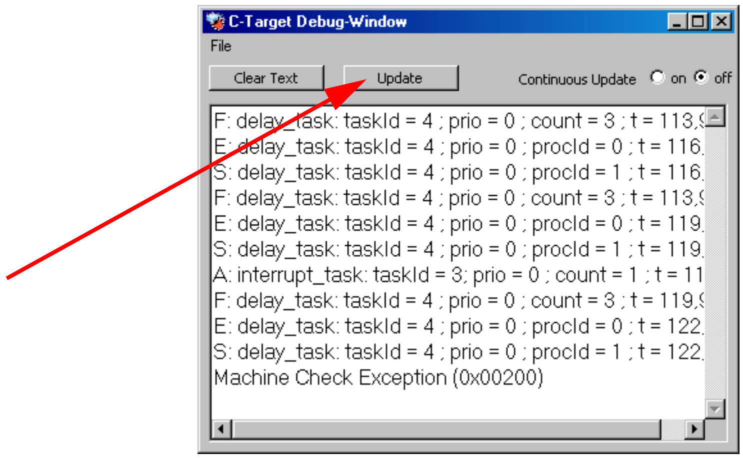
L1-Message Fatal Error (PID: 0 OID: 4 CMD: 0)
Undefined Error: 113!
You have to exit the experiment!

- In the "Physical Experiment" window, select **File** → **Exit** to exit the experiment.

  To evaluate the error cause, the experiment must be restarted.



- Select `Online (RP)` from the "Experiment Target" combo box.
- Select **Build** → **Experiment** to start the online experiment.
- In the "Physical Experiment" window, select **Tools** → **Target Debugger** to open the Target Debug window.



```
F: delay_task: taskId = 4 ; prio = 0 ; count = 3 ; t = 113,9
E: delay_task: taskId = 4 ; prio = 0 ; procId = 0 ; t = 116.
S: delay_task: taskId = 4 ; prio = 0 ; procId = 1 ; t = 116.
F: delay_task: taskId = 4 ; prio = 0 ; count = 3 ; t = 113,9
E: delay_task: taskId = 4 ; prio = 0 ; procId = 0 ; t = 119.
S: delay_task: taskId = 4 ; prio = 0 ; procId = 1 ; t = 119.
A: interrupt_task: taskId = 3; prio = 0 ; count = 1 ; t = 11
F: delay_task: taskId = 4 ; prio = 0 ; count = 3 ; t = 119,9
E: delay_task: taskId = 4 ; prio = 0 ; procId = 0 ; t = 122.
S: delay_task: taskId = 4 ; prio = 0 ; procId = 1 ; t = 122.
Machine Check Exception (0x00200)
```

- Click on the **Update** button.

  The last pieces of trace information are displayed in the Target Debug window.

2.4 Trace Information

The trace information indicates the cause of the system crash. In the case of an exception, the relevant register contents (e.g. SRR0) of the PowerPC are displayed. Using the PPC manual as a reference, you can then pinpoint the cause and the address at which the error occurred. Using a map file, you can then establish a relationship to the project code. With the GNU compiler, this file is always created.

When using the ES1130 with DIAB compiler, the `-m2` option must be inserted in the `comptool.ini` file under `LinkCal1` or `ROMLinkCal1`, so that the map file is created.

Example:

```
LinkCall1=redir dcc -m2 -lm -L %bp% -Wsp_st_ram.o -
o%fn%.out %fn%.lnk -Wmp_ram.loc
```

The `comptool.ini` file is located in the ASCET subdirectory `target\ES1130`.

If monitoring has not been activated in the OS Editor, the user processes are counted starting with 0. If monitoring is activated, counting starts with 1. In any case, more processes are displayed than exist in the OS Editor. This results from an implicit process insertion made by ASCET.

The task priorities are displayed in absolute terms. This means that, for example, with 20 cooperative and 20 preemptive levels, the cooperative tasks have the priorities 0-19 and the preemptive tasks have the priorities 20-39.

Tasks defined explicitly within the trace information are only displayed correctly in the OS Editor since only these tasks are contained in the reference table generated by ASCET. If other tasks generated by ASCET are run, these tasks are marked as unknown tasks in the trace information.

Usually, the trace information is not displayed for the OP mode 0 (**Stop OS** button pressed) since generally, no user task is run there except for an exit task which may have been defined. Should a system crash occur in OP mode 0, however, the associated trace information is displayed when the system is restarted the next time.

2.5 Example of a Trace Information Output

In the first portion, the board versions, the cause of the system crash, any relevant register contents (SRR0: address at which the crash occurred), the boot mode and runtime information are displayed.

```
********** Tracing Information **********

Operating Mode: 1
Reason for Trace-Info: Operating Mode Change
Mode: Boot from RAM
RunTime: 5.126512s  IdleTime: 5.116872s  SysLoad: 0.2%
```

Following this, the activations of the individual user tasks are displayed. They provide information on the number of successful and failed activations. Also, the maximum number of simultaneous activations of the individual tasks allowed is displayed.

```
Task Activation Logging:
SW: Init: Id = 1, Prio = 0 ; SucA = 0 ; FailA = 0 ; ↵
     MaxA = 1
SW: Exit: Id = 2, Prio = 0 ; SucA = 0 ; FailA = 0 ; ↵
     MaxA = 1
SW: Config: Id = 3, Prio = 0 ; SucA = 42 ;       ↵
     FailA = 0 ; MaxA = 1
SW: Measure: Id = 4, Prio = 0 ; SucA = 42 ;      ↵
     FailA = 0 ; MaxA = 1
HW: Handler: Id = 5, Prio = 20 ; SucA = 0 ;      ↵
     FailA = 0 ; MaxA = 1
```

Following this is a section containing explanations of the abbreviations used later.

```
Task Monitoring (oldest Entries first):
A: Successful Activation
F: Failed Activation
S: Start of Process
E: End of Process
T: Task Termination; End of Process
I: Idle-Task
H: RTIO-Interrupt Hook
```

Following this is the actual tracing information. In this example, a ring buffer overflow occurred. As a consequence, all activations and processes from the start up to the crash are listed.

```
Trace Buffer Overflow ...
E:  Config: taskId = 3 ; prio = 0 ; procId = 1 ; ↵
     t = 0.000000 ms
S:  Config: taskId = 3 ; prio = 0 ; procId = 2 ; ↵
     t = 0.003360 ms
E:  Config: taskId = 3 ; prio = 0 ; procId = 2 ; ↵
     t = 0.007080 ms
S:  Config: taskId = 3 ; prio = 0 ; procId = 3 ; ↵
     t = 0.018360 ms
T:  Config: taskId = 3 ; prio = 0 ; procId = 3 ; ↵
     t = 0.022020 ms
S:  Measure: taskId = 4 ; prio = 0 ; procId = 0 ; ↵
     t = 0.033720 ms
E:  Measure: taskId = 4 ; prio = 0 ; procId = 0 ; ↵
     t = 0.037500 ms
S:  Measure: taskId = 4 ; prio = 0 ; procId = 1 ; ↵
     t = 0.040860 ms
E:  Measure: taskId = 4 ; prio = 0 ; procId = 1 ; ↵
     t = 0.127260 ms
S:  Measure: taskId = 4 ; prio = 0 ; procId = 2 ; ↵
     t = 0.130620 ms
E:  Measure: taskId = 4 ; prio = 0 ; procId = 2 ; ↵
     t = 0.134340 ms
S:  Measure: taskId = 4 ; prio = 0 ; procId = 3 ; ↵
     t = 0.145680 ms
T:  Measure: taskId = 4 ; prio = 0 ; procId = 3 ; ↵
     t = 0.149340 ms
I:  Idle-Task: t = 0.153900 ms
```

```
A:  L1-Task: prio = 0 ; count = 1 ; t = 59.188080 ms
S:  L1-Task: prio = 0 ; procId = 0 ; t = 59.197680 ms
E:  L1-Task: prio = 0 ; procId = 0 ; t = 59.204400 ms
S:  L1-Task: prio = 0 ; procId = 1 ; t = 59.213520 ms
T:  L1-Task: prio = 0 ; procId = 1 ; t = 59.217360 ms
I:  Idle-Task: t = 59.222040 ms
A:  Config: taskId = 3 ; prio = 0 ; count = 1 ; ↵
     t = 99.973264 ms
A:  Measure: taskId = 4 ; prio = 0 ; count = 1 ; ↵
     t = 99.976736 ms
S:  Config: taskId = 3 ; prio = 0 ; procId = 0 ; ↵
     t = 99.988976 ms
E:  Config: taskId = 3 ; prio = 0 ; procId = 0 ; ↵
     t = 99.993056 ms

[...]

E:  Measure: taskId = 4 ; prio = 0 ; procId = 2 ; t =
200.134384 ms
S:  Measure: taskId = 4 ; prio = 0 ; procId = 3 ; t =
200.145664 ms
T:  Measure: taskId = 4 ; prio = 0 ; procId = 3 ; t =
200.149312 ms
I:  Idle-Task: t = 200.153888 ms
A:  L1-Task: prio = 0 ; count = 1 ; t = 224.329984 ms
S:  L1-Task: prio = 0 ; procId = 0 ; t = 224.338320 ms
E:  L1-Task: prio = 0 ; procId = 0 ; t = 224.345408 ms
S:  L1-Task: prio = 0 ; procId = 1 ; t = 224.354464 ms
T:  L1-Task: prio = 0 ; procId = 1 ; t = 224.358240 ms
I:  Idle-Task: t = 224.362976 ms
Reason for Trace-Info: Operating Mode Change
```

Since the individual processes in the OS Editor are not assigned any IDs, this assignment has to be performed manually using the OS Editor.

A distinction must be made as to whether monitoring is activated for the respective task in the OS Editor. If monitoring is deactivated, the processes are counted starting with 0. If monitoring is activated, the processes are counted starting with 1 since in this case, code generation implicitly inserts a monitor process at the very beginning which does not appear in the OS Editor.

In general, each task has one or several processes at the end which do not appear in the OS Editor. In the case of the Init task, these are specific Init processes, an optional monitor process and a terminate task process.

The `t` time stamps of the events in the task monitoring are relative to the oldest event still existing in the ring buffer.

The abbreviations at the beginning of each line indicate what kind of event has resulted in the respective monitor entry. This is followed by the task name, the task ID (except in the case of the L1 and the Init tasks), the task priority (except for Init and Idle), the respective process ID and the time stamp.

Activations can either be successful (abbreviation A) or they can fail (abbreviation F). The cause of an unsuccessful activation may either be the exceeding of the maximum number of simultaneous activations of the respective task allowed or a limited FIFO capacity on this priority level. In the case of the Idle task, only the abbreviation I and the time are recorded when the Idle task is executed for the first time after any other task. All Idle task activations following immediately are no longer recorded. The Idle phase, however, is terminated when the next normal task begins to be processed.

2.6 Causes of L1 Errors

The most common causes of L1 errors are:

- L1 timeout while the Init/Exit task is being run.
- Watchdog reset while the Init/Exit task is being run.
- L1 error due to unmatched priority of the L1 task.
- Watchdog reset during runtime due to interrupt lock.
- Machine check exception due to unreliable memory accesses.

To avoid these errors, the following rules must be observed:

- The Init task is executed under a global interrupt lock. This means that within an Init task process neiter L1 requests nor the operating system watchdog are being served. At the process borders, only the watchdog is served. In the case of process runtimes of more than 200 ms, it is required, therefore, to serve the watchdog using the `triggerWD()` function within the process. In the case of Init task runtimes of more than 1 sec, L1 requests need to be served additionally using the `serveL1Request()` function. The `triggerWD()` and `serveL1Request()` functions are contained in the system library (`a_system.lib`) provided.
- In the case of very long process runtimes or a very high system load, the L1 task activated by the host may not run due to its priority. If required, the L1 priority (by default, priority 0; cooperative) may be increased. For this purpose, the value of `__L1_Prio` in the `pvmecomp.inv` or `pcio-comp.inv` target file must be increased. For example, with 20 cooperative and 20 preemptive levels, a value of `__L1_Prio=26` means that the L1 task has the preemptive priority 6. Since this priority increase may affect the real-time behavior of the model code, this step must be carefully considered.
- If a task subject to an interrupt lock waits for a long period of time (e.g. 200 ms; e.g. polling a status register in an interrupt handler), the watchdog is not served by the operating system during this period of time. Here, too, `triggerWD()` function calls need to be inserted accordingly.
- Any access to inadmissible memory areas or IO addresses (e.g. IO cards not plugged in) results in a machine check exception. Therefore, this may not occur in any case.

Index