OS Awareness Manual OSE Classic

OS Awareness Manual OSE Classic

Version 18-Nov-2020

History

28-Aug-18 The title of the manual was changed from “RTOS Debugger for *<x>*” to “OS Awareness Manual *<x>*”.

Overview

The OS Awareness for OSE Classic contains special extensions to the TRACE32 Debugger. This chapter describes the additional features, such as additional commands and statistic evaluations.

Brief Overview of Documents for New Users

Architecture-independent information:

- **“Debugger Basics - Training”** (training_debugger.pdf): Get familiar with the basic features of a TRACE32 debugger.
- **“T32Start”** (app_t32start.pdf): T32Start assists you in starting TRACE32 PowerView instances for different configurations of the debugger. T32Start is only available for Windows.
- **“General Commands”** (general_ref_*<x>*.pdf): Alphabetic list of debug commands.

Architecture-specific information:

- **“Processor Architecture Manuals”**: These manuals describe commands that are specific for the processor architecture supported by your debug cable. To access the manual for your processor architecture, proceed as follows:

  - Choose **Help** menu > **Processor Architecture Manual**.

- **“OS Awareness Manuals”** (rtos_*<os>*.pdf): TRACE32 PowerView can be extended for operating system-aware debugging. The appropriate OS Awareness manual informs you how to enable the OS-aware debugging.

Supported Version

Currently OSE Classic is supported for the following version:

- OS68 on Freescale Semiconductor 68k.

Configuration

The **TASK.CONFIG** command loads an extension definition file called “osec.t32” (directory “~~/demo/<*processor*>/kernel/osec”). It contains all necessary extensions.

Automatic configuration tries to locate the OSE Classic internals automatically. For this purpose all symbol tables must be loaded and accessible at any time the OS Awareness is used.

If a system symbol is not available or if another address should be used for a specific system variable then the corresponding argument must be set manually with the appropriate address. This can be done by manual configuration which can require some additional arguments, too.

If you want to have dual port access for the display functions (display “On The Fly”), you have to map emulation memory to the address space of all used system tables.

Manual configuration of the OS Awareness for OSE Classic can be used to explicitly define some memory locations.

Format:	**TASK.CONFIG osec** *<magic_address> <sleep> <pool_start> <signal_desc> <pool_end>*

<magic_address> Specifies a memory location that contains the current running task. This address can be found at "ZZ_CUR_PCB", or "P_ZZ_CUR_PCB".

<sleep> The argument for *<sleep>* is currently not used. Specify "0".

<pool_start> This argument must be the address of the OS68 Memory Pool Start. Give the label "ZZPOOL_START".

<system_call> This argument passes the address of the system call patch routine. A '0' indicates, that no such routine is present, i.e. no patch was done.

<signal_desc> The argument *<signal_desc>* specifies the internal signal description table, marked by the label "ose_sigdesc_tab". Specify '0' if this table is not available.

<pool_end> This argument is the address of the OS68 Memory Pool End (the label "ZZPOOL_END") and is needed for recognizing dynamic processes. If you do not need the display of dynamic processes, or if you do not use dynamic processes at all, then specify a '0'. This will speed up the table display by showing only static processes.

If the task selective debugging features are not used, the patching of the kernel is not required. The PRACTICE script 'posec.cmm' can make the required patches to OS68 and configure the display command:

```
do posec251 nopatch             ; configures only display functions
                                ; no patches are made

do posec251                     ; patched XEC 68 for full debug support
```

When patching is required the patch area in the PRACTICE script file must be modified to point to an unused memory area.

The demo script 'osec.cmm' in the '~~/demo/m68k/kernel/oseclass' directory can be started with the same parameters.

NOTE: In this version no user processes are supported. The configuration file for OS68 (os68.con) must contain the line "%USER_PROCESSES NO".

Automatic Configuration

Automatic Configuration is not supported for OSE Classic.

Quick Configuration Guide

To access all features of the OS Awareness you should follow the following roadmap:

1. Run the demo script (~~/demo/m68k/kernel/oseclass/osec.cmm) without any patching. Start the demo with 'do osec nopatch'. The result should be a list of tasks, which change continuously their state.
2. Run the demo script with patching.
3. Try the analyzer demo programs (tasksc, taskstat and taskfunc).
4. Make a copy of the 'posec.cmm' PRACTICE file. Modify the file according to your application. This can be choosing a different memory area for the patches. Use the PRACTICE file which corresponds to your OS68 version!
5. Run the modified version in your application **without patching** (with 'nopatch' argument). This should allow you to display the kernel resources.
6. Run your application with patching.

Hooks & Internals in OSE Classic

To determine the entry of a task, the patching of OS68 is required. All returns to the task context (by return of a system call) are patched to pass control to the multitask monitor. The patch writes the current executing tcb address to the magic-word of the OS Awareness.

The entries to OS68 are patched in the routines of the system calls. The patches write the value 1 to the magic-word.

Features

The OS Awareness for OSE Classic supports the following features.

Display of Kernel Resources

The extension defines new PRACTICE commands to display various kernel resources. The commands can either give an overview about one resource type or display a single resource in detail. The resource can be defined by its ID, magic or name. The following information can be displayed:

- processes	(DP, DPS)
- signal queues	(DQ)
- signal names	DS (if available)

For a detailed description of each command refer to the chapter “OSE Classic Commands”.

When working with emulation memory or shadow memory, these resources can be displayed “On The Fly”, i.e. while the target application is running, without any intrusion to the application. If using this dual port memory feature, be sure that emulation memory is mapped to all places, where OSE Classic holds its tables.

When working only with target memory, the information will only be displayed, if the target application is stopped.

Task Runtime Statistics

NOTE:	This feature is *only* available, if your debug environment is able to trace task switches (program flow trace is not sufficient). It requires either an on-chip trace logic that is able to generate task information (eg. data trace), or a software instrumentation feeding one of TRACE32 software based traces (e.g. **FDX** or **Logger**). For details, refer to **“OS-aware Tracing”** (glossary.pdf).

Based on the recordings made by the **Trace** (if available), the debugger is able to evaluate the time spent in a task and display it statistically and graphically.

To evaluate the contents of the trace buffer, use these commands:

Trace.List List.TASK DEFault	Display trace buffer and task switches
Trace.STATistic.TASK	Display task runtime statistic evaluation
Trace.Chart.TASK	Display task runtime timechart

Trace.PROfileSTATistic.TASK	Display task runtime within fixed time intervals statistically
Trace.PROfileChart.TASK	Display task runtime within fixed time intervals as colored graph
Trace.FindAll Address TASK.CONFIG(magic)	Display all data access records to the “magic” location
Trace.FindAll CYcle owner OR CYcle context	Display all context ID records

The start of the recording time, when the calculation doesn’t know which task is running, is calculated as “(unknown)”.

Task State Analysis

NOTE:	This feature is *only* available, if your debug environment is able to trace task switches and data accesses (program flow trace is not sufficient). It requires either an on-chip trace logic that is able to generate a data trace, or a software instrumentation feeding one of TRACE32 software based traces (e.g. **FDX** or **Logger**). For details, refer to **“OS-aware Tracing”** (glossary.pdf).

The time different tasks are in a certain state (running, ready, suspended or waiting) can be evaluated statistically or displayed graphically.

This feature requires that the following data accesses are recorded:

- All accesses to the status words of all tasks
- Accesses to the current task variable (= magic address)

Adjust your trace logic to record all data write accesses, or limit the recorded data to the area where all TCBs are located (plus the current task pointer).

Example: This script assumes that the TCBs are located in an array named TCB_array and consequently limits the tracing to data write accesses on the TCBs and the task switch.

```
Break.Set Var.RANGE(TCB_array) /Write /TraceData
Break.Set TASK.CONFIG(magic) /Write /TraceData
```

To evaluate the contents of the trace buffer, use these commands:

Trace.STATistic.TASKState	Display task state statistic
Trace.Chart.TASKState	Display task state timechart

The start of the recording time, when the calculation doesn't know which task is running, is calculated as "(unknown)".

Function Runtime Statistics

NOTE:	This feature is *only* available, if your debug environment is able to trace task switches (program flow trace is not sufficient). It requires either an on-chip trace logic that is able to generate task information (eg. data trace), or a software instrumentation feeding one of TRACE32 software based traces (e.g. **FDX** or **Logger**). For details, refer to **"OS-aware Tracing"** (glossary.pdf).

All function-related statistic and time chart evaluations can be used with task-specific information. The function timings will be calculated dependent on the task that called this function. To do this, in addition to the function entries and exits, the task switches must be recorded.

To do a selective recording on task-related function runtimes based on the data accesses, use the following command:

```
; Enable flow trace and accesses to the magic location
Break.Set TASK.CONFIG(magic) /TraceData
```

To do a selective recording on task-related function runtimes, based on the Arm Context ID, use the following command:

```
; Enable flow trace with Arm Context ID (e.g. 32bit)
ETM.ContextID 32
```

To evaluate the contents of the trace buffer, use these commands:

Trace.ListNesting	Display function nesting
Trace.STATistic.Func	Display function runtime statistic
Trace.STATistic.TREE	Display functions as call tree
Trace.STATistic.sYmbol /SplitTASK	Display flat runtime analysis
Trace.Chart.Func	Display function timechart
Trace.Chart.sYmbol /SplitTASK	Display flat runtime timechart

The start of the recording time, when the calculation doesn't know which task is running, is calculated as "(unknown)".

System Calls

System calls with parameters can be traced and used as trigger criteria by the state analyzer. This feature requires patching the kernel. The example script 'tasksc.cmm' makes a trace of system calls and task switches in the demo application.

Analyzer.List	List.TASK	Display system calls

OSE Classic specific Menu

The file 'osec.men' contains an alternate menu with OSE Classic specific topics. Load this menu with the **MENU.ReProgram** command.

You will find a new pull-down menu called 'OSE Classic'.

The 'Display' topics launch the kernel resource display windows.

The 'Analyzer->List' pull-down menu is changed. You can additionally choose for a analyzer list window showing only task switches (if any) or task switches with defaults.

The 'Perf' menu contains the additional submenus for task runtime statistics, task related function runtime statistics and statistics on task states. For the function runtime statistics, a prepare command file called "men_ptfp.cmm" is used. This command file must be adapted to your application.

TASK.DProc — Process table

Format: **TASK.DProc** [*<process>*]

Displays the process-table of OS68.
It is similar to the 'dp' command of the OS68 debugger without any argument. With a process name or a process ID as an argument it displays one process in detail, like the 'dp *<name>*,f' command of OS68 debugger.

NOTE:

- The pc location is not always computed right - use it as an approximation.
- The process state 'running' is only recognized if running with patch.

Without patching a running task is displayed as 'ready'.

- If there are dashes ('--'), then the information is not available.
- The number of signals in queue is limited to 99, the number of signals waiting for is limited to 5. More signals as displayed are indicated by a leading '>'.
- The signal information is always right when the system is inside a task. When the system is inside the kernel and changing currently the signal descriptors, the signal queues could be displayed wrong (especially while running in real time).
- The signals are displayed by name, if the signal description is available. Otherwise they are displayed in hex.

TASK.DPS — Process statistics table

Format: **TASK.DPS**

Displays the statistics table of OS68.

This table is designed like the command 'dp *,s' of the OS68 debugger.

NOTE:

- The percentages are currently dummies and not computed.
- For the signals in queue see the note at TASK.DP.

Format: **TASK.DQ** *<process>*

Displays the signal queue of the specified process.

NOTE: See the note at TASK.DP

TASK.DS Signal names

Format: **TASK.DS**

Displays the signal names as defined according their ids.
This command is only available, if the internal signal description table is found.

Format: **TASK.SysCall** *<function> <parameters>*

<function>: **ADDRessee | Current_P | ERRor | Get_Fsem | Get_Pri**
Get_Ticks | ReCeiVe | Send | Send_W_S | SENDER
Set_Fsem | Set_Pri | SIG_Fsem | SIGSize | START | STOP

Executes a system call.
The function can only be executed, when the emulation is stopped in a regular task. The function runs under the current task ID. Take care that a preempted task switch (e.g. in waiting conditions) can hang the emulation. So no task switch condition should be given in a system call. Patching is required to use this function.

```
task.sc current_p

task.sc get_pri 3F20C
```

TASK.TASKState Mark task state words

Format: **TASK.TASKState**

This command sets Alpha breakpoints on all process status words.

The statistic evaluation of process states (see Task State Analysis) requires recording of the accesses to the process state words. By setting Alpha breakpoints to this words, and selectively recording Alpha's, you can do a selective recording of process state transitions.

Because setting the Alpha breakpoints by hand is very hard to do, this utility command sets automatically the Alpha's to the status words of all processes currently created. It does NOT set breakpoints to processes, that terminated or haven't yet been created.

OSE Classic PRACTICE Functions

There are special definitions for OSE Classic specific PRACTICE functions.

TASK.CONFIG() OS Awareness configuration information

Syntax: **TASK.CONFIG(magic | magicsize)**

Parameter and Description:

magic	**Parameter Type**: String (***without*** quotation marks). Returns the magic address, which is the location that contains the currently running task (i.e. its task magic number).
magicsize	**Parameter Type**: String (***without*** quotation marks). Returns the size of the magic number (1, 2 or 4).

Return Value Type: Hex value.

Frequently-Asked Questions

No information available