OS Awareness Manual VDK

TRACE32 Online Help

TRACE32 Directory

TRACE32 Index

TRACE32 Documents ....

OS Awareness Manuals ....

OS Awareness Manual VDK 1

History 2

Overview 2

Terminology 2

Brief Overview of Documents for New Users 2

Supported Versions 3

Configuration 4

Quick Configuration Guide 4

Hooks & Internals in VDK 5

Features 6

Display of Kernel Resources 6

Task Stack Coverage 6

Task-Related Breakpoints 7

Dynamic Task Performance Measurement 8

VDK Specific Menu 8

VDK Commands 9

TASK.DevFlag Display device flags 9

TASK.MemPool Display memory pools 9

TASK.Semaphore Display semaphores 10

TASK.Thread Display threads 10

VDK PRACTICE Functions 11

TASK.CONFIG() OS Awareness configuration information 11

Frequently-Asked Questions 11

©1989-2020 Lauterbach GmbH

OS Awareness Manual VDK

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 VDK contains special extensions to the TRACE32 Debugger. This manual describes the additional features, such as additional commands and statistic evaluations.

Terminology

VDK uses the term “threads”. If not otherwise specified, the TRACE32 term “task” corresponds to VDK threads.

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.

©1989-2020 Lauterbach GmbH

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 Versions

Currently VDK is supported for the following versions:

- VDK from VisualDSP 4.5 for Blackfin

©1989-2020 Lauterbach GmbH

Configuration

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

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

If you want to display the OS objects “On The Fly” while the target is running, you need to have access to memory while the target is running. In case of ICD, you have to enable **SYStem.MemAccess** or **SYStem.CpuAccess** (CPU dependent). In case of a ICE or FIRE, you have to map emulation or shadow memory to the address space of all used system tables.

For system resource display and trace functionality, you can do an automatic configuration of the OS Awareness. For this purpose it is necessary that all system internal symbols are loaded and accessible at any time, the OS Awareness is used. Each of the **TASK.CONFIG** arguments can be substituted by '0', which means that this argument will be searched and configured automatically. For a fully automatic configuration omit all arguments:

Format:	**TASK.CONFIG vdk**

See also “**Hooks & Internals**” for details on the used symbols.

Quick Configuration Guide

To get a quick access to the features of the OS Awareness for VDK with your application, follow the following roadmap:

1. Copy the files “`vdk.t32`” and “`vdk.men`” to your project directory (from TRACE32 directory “~~/demo/*<processor>*/kernel/vdk”).
2. Start the TRACE32 Debugger.
3. Load your application as normal.
4. Execute the command “`TASK.CONFIG vdk`” (See “**Configuration**”).
5. Execute the command “`MENU.ReProgram vdk`” (See “**VDK Specific Menu**”).
6. Start your application.

Now you can access the VDK extensions through the menu.

In case of any problems, please carefully read the previous Configuration chapter.

©1989-2020 Lauterbach GmbH

Hooks & Internals in VDK

No hooks are used in the kernel.

For retrieving the kernel data and structures, the OS Awareness uses the global kernel symbols and structure definitions. Ensure that access to those structures is possible every time when features of the OS Awareness are used.

Be sure that your application is compiled and linked with debugging symbols switched on.

©1989-2020 Lauterbach GmbH

Features

The OS Awareness for VDK supports the following features.

Display of Kernel Resources

The extension defines new commands to display various kernel resources. Information on the following VDK components can be displayed:

TASK.Thread	Threads
TASK.Semaphore	Semaphores
TASK.DevFlag	Device flags
TASK.MemPool	Memory pools

For a description of the commands, refer to chapter “**VDK Commands**”.

If your hardware allows memory access while the target is running, these resources can be displayed “On The Fly”, i.e. while the application is running, without any intrusion to the application.

Without this capability, the information will only be displayed if the target application is stopped.

Task Stack Coverage

For stack usage coverage of VDK tasks, you can use the **TASK.STacK** command. Without any parameter, this command will set up a window with all active VDK tasks. If you specify only a magic number as parameter, the stack area of this task will be automatically calculated.

To use the calculation of the maximum stack usage, flag memory must be mapped to the task stack areas, when working with the emulation memory. When working with the target memory, a stack pattern must be defined with the command **TASK.STacK.PATtern** (default value is zero).

To add/remove one task to/from the task stack coverage, you can either call the **TASK.STacK.ADD** rsp. **TASK.STacK.ReMove** commands with the task magic number as parameter, or omit the parameter and select from the task list window.

It is recommended to display only the tasks you are interested in, because the evaluation of the used stack space is very time consuming and slows down the debugger display.

©1989-2020 Lauterbach GmbH

Task-Related Breakpoints

Any breakpoint set in the debugger can be restricted to fire only if a specific task hits that breakpoint. This is especially useful when debugging code which is shared between several tasks. To set a task-related breakpoint, use the command:

Break.Set *<address>*|*<range>* [**/***<option>*] **/TASK** *<task>*	Set task-related breakpoint.

- Use a magic number, task ID, or task name for *<task>*. For information about the parameters, see **"What to know about the Task Parameters"** (general_ref_t.pdf).
- For a general description of the **Break.Set** command, please see its documentation.

By default, the task-related breakpoint will be implemented by a conditional breakpoint inside the debugger. This means that the target will *always* halt at that breakpoint, but the debugger immediately resumes execution if the current running task is not equal to the specified task.

NOTE:	Task-related breakpoints impact the real-time behavior of the application.

On some architectures, however, it is possible to set a task-related breakpoint with *on-chip* debug logic that is less intrusive. To do this, include the option **/Onchip** in the **Break.Set** command. The debugger then uses the on-chip resources to reduce the number of breaks to the minimum by pre-filtering the tasks.

For example, on ARM architectures: *If* the RTOS serves the Context ID register at task switches, and *if* the debug logic provides the Context ID comparison, you may use Context ID register for less intrusive task-related breakpoints:

Break.CONFIG.UseContextID ON	Enables the comparison to the whole Context ID register.
Break.CONFIG.MatchASID ON	Enables the comparison to the ASID part only.
TASK.List.tasks	If **TASK.List.tasks** provides a trace ID (**traceid** column), the debugger will use this ID for comparison. Without the trace ID, it uses the magic number (**magic** column) for comparison.

When single stepping, the debugger halts at the next instruction, regardless of which task hits this breakpoint. When debugging shared code, stepping over an OS function may cause a task switch and coming back to the same place - but with a different task. If you want to restrict debugging to the current task, you can set up the debugger with **SETUP.StepWithinTask ON** to use task-related breakpoints for single stepping. In this case, single stepping will always stay within the current task. Other tasks using the same code will not be halted on these breakpoints.

If you want to halt program execution as soon as a specific task is scheduled to run by the OS, you can use the **Break.SetTask** command.

©1989-2020 Lauterbach GmbH

Dynamic Task Performance Measurement

The debugger can execute a dynamic performance measurement by evaluating the current running task in changing time intervals. Start the measurement with the commands **PERF.Mode TASK** and **PERF.Arm**, and view the contents with **PERF.ListTASK**. The evaluation is done by reading the 'magic' location (= current running task) in memory. This memory read may be non-intrusive or intrusive, depending on the **PERF.METHOD** used.

If **PERF** collects the PC for function profiling of processes in MMU-based operating systems (**SYStem.Option MMUSPACES ON**), then you need to set **PERF.MMUSPACES**, too.

For a general description of the **PERF** command group, refer to **"General Commands Reference Guide P"** (general_ref_p.pdf).

All kernel activities up to the task switch are added to the calling task.

VDK Specific Menu

The menu file "vdk.men" contains a menu with VDK specific menu items. Load this menu with the **MENU.ReProgram** command.

You will find a new menu called **VDK**.

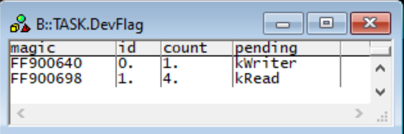
- The **Display** menu items launch the kernel resource display windows.
- The **Stack Coverage** submenu starts and resets the VDK specific stack coverage and provides an easy way to add or remove tasks from the stack coverage window.

©1989-2020 Lauterbach GmbH

TASK.DevFlag Display device flags

Format:	**TASK.DevFlag**

Displays the device flag table of VDK.

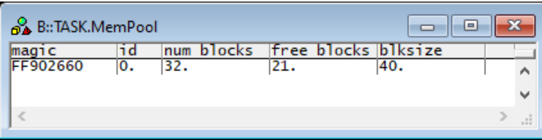


“magic” is a unique ID, used by the OS Awareness to identify a specific device flag (address of the DeviceFlag object).

TASK.MemPool Display memory pools

Format:	**TASK.MemPool**

Displays the memory pool table of VDK.



“magic” is a unique ID, used by the OS Awareness to identify a specific memory pool (address of the MemoryPool object).

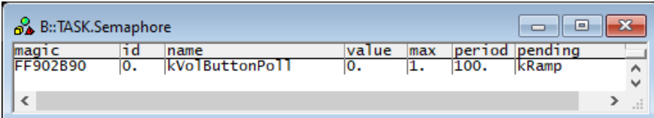
©1989-2020 Lauterbach GmbH

TASK.Semaphore

Display semaphores

Format:	**TASK.Semaphore**

Displays the semaphore table of VDK.



B::TASK.Semaphore

magic	id	name	value	max	period	pending
FF902B90	0.	kVolButtonPoll	0.	1.	100.	kRamp

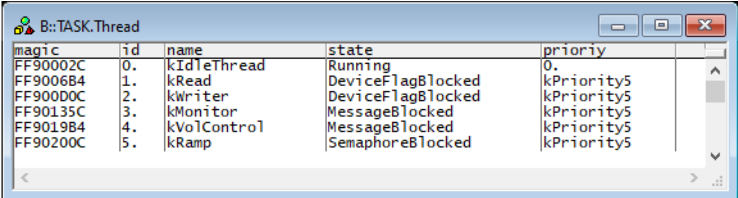
“magic” is a unique ID, used by the OS Awareness to identify a specific semaphore (address of the Semaphore object).

TASK.Thread

Display threads

Format:	**TASK.Thread**

Displays the thread table of VDK.



B::TASK.Thread

magic	id	name	state	prioriy
FF90002C	0.	kIdleThread	Running	0.
FF9006B4	1.	kRead	DeviceFlagBlocked	kPriority5
FF900D0C	2.	kWriter	DeviceFlagBlocked	kPriority5
FF90135C	3.	kMonitor	MessageBlocked	kPriority5
FF9019B4	4.	kVolControl	MessageBlocked	kPriority5
FF90200C	5.	kRamp	SemaphoreBlocked	kPriority5

“magic” is a unique ID, used by the OS Awareness to identify a specific thread (address of the TMK_Thread object).

©1989-2020 Lauterbach GmbH

VDK PRACTICE Functions

There are special definitions for VDK 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 task magic number (1, 2 or 4).

Return Value Type: Hex value.

Frequently-Asked Questions

No information available

©1989-2020 Lauterbach GmbH