OS Awareness Manual MTOS-UX

OS Awareness Manual MTOS-UX

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 MTOS-UX 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 Versions

Currently MTOS-UX is supported for the version MTOS-UX/68k V 3.6 on the Freescale Semiconductor 68332/376.

Manual Configuration

Format:	**TASK.CONFIG mtos** *<magic_address> <args>*

<magic_address> Specifies a memory location that contains the current running task. This address can be found at the label "LCLBAS".

<args> The configuration requires one additional argument that specifies an MTOS-UX internal table. Give the label "GBLBAS".

This command configures the OS Awareness for MTOS-UX with manual setup.

The **TASK.CONFIG** command loads an extension definition file called "mtos.t32" (directory "~~/demo/m68k/kernel/mtosux"). It contains all necessary extensions.

```
; manual configuration for MTOS-UX support
TASK.CONFIG mtos LCLBAS GBLBAS
```

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. See also **TASK.MAP**.

See also the example "~~/demo/m68k/kernel/mtosux/mtos.cmm"

Format:	**TASK.CONFIG mtos**

This command configures the OS Awareness for MTOS-UX with automatic setup.

The **TASK.CONFIG** command loads an extension definition file called “mtos.t32” (directory “~~/demo/m68k/kernel/mtosux”). It contains all necessary extensions.

This configuration tries to locate the MTOS-UX internals automatically. For this purpose the symbols ’LCLBAS’ and ’GBLBAS’ must be loaded and accessible at any time, the OS Awareness is used.

Each TASK.CONFIG argument can be substituted by ’0’, which means, that this argument will be searched and configured automatically. For a full automatic configuration omit all arguments to the command:

```
; full automatic configuration for MTOS-UX support
TASK.CONFIG mtos
```

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.

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. See also **TASK.MAP**.

See also the example “~~/demo/m68k/kernel/mtosux/mtos.cmm”

Hooks & Internals of MTOS-UX

The local ram data table (LCL) is used for determination of the current running task. The global ram data table (GBL) and its referenced tables are used for detecting all MTS-UX system resources.

Features

The OS Awareness for MTOS_UX supports the following features.

SYSC Terminal Emulation

The Terminal Emulation window can be used to communicate with the target system console, called 'SYSC'.

The communication is done via two memory cells, requiring no external hardware interface. See the **TERM** command for a description of the terminal emulation.

On request LAUTERBACH can provide you with the source code for the target interface routine.

The example ("~~/demo/m68k/kernel/mtosux/mtos.cmm") contains this interface and the terminal emulation for output messages. The example also contains the MTOS-UX dynamic debugger, which uses the system console, too. Invoke the MTOS-UX dynamic debugger by pressing CTRL-D inside the system console window.

Display of Kernel Resources

The extension defines new PRACTICE commands to display various kernel resources. The following information can be displayed:

tasks	(DispTask)
time & TOD	(DispTIme)
global event flag groups	(DispEvent)
common memory pools	(DispPool)
fixed block memory pools	(DispFixed)
message buffers	(DispmBuff)
mailboxes	(DispMbx)
semaphores	(DispSem)
controlled shared variables	(DispsVar)
physical I/O units	(DispUnit)

For a detailed description of each command refer to the chapter “MTOS-UX PRACTICE 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 MTOS-UX holds its tables.

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

Task Stack Coverage

For stack usage coverage of the tasks, you can use the **TASK.STacK** command. Without any parameter, this command will open a window displaying with all active tasks. If you specify only a task 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** or **TASK.STacK.ReMove** commands with the task magic number as the parameter, or omit the parameter and select the task from the **TASK.STacK.*** 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.

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)".

MTOS-UX specific Menu

The menu file "mtos.men" contains a menu with MTOS-UX specific menu items. Load this menu with the **MENU.ReProgram** command.

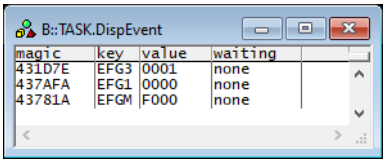
You will find a new menu called **MTOS-UX**.

- The item "SYSC Terminal" brigs up a terminal emulation window, which communicates with the preconfigured MTOS-UX system console. The "Break to .SYSD" item initiates a <CTRL-D> key to the system console and thus starts the MTOS-UX dynamic debugger.
- The "Display" Topics launch the kernel resource display windows.
- The "Stack Coverage" submenu starts and resets the MTOS-UX specific stack coverage, and provides an easy way to add or remove tasks from the stack coverage window.
- The Analyzer->List pull-down menu is changed. You can additionally choose for an analyzer list window showing only task switches (if any) or task switches and 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.DispEvent — Display global event flags

Format: **TASK.DispEvent**

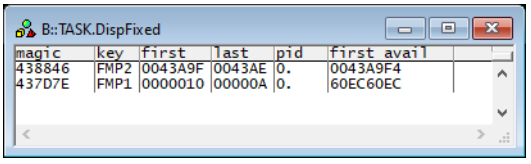
Displays a table with all created global event flag groups.



TASK.DispFixed — Display fixed pools

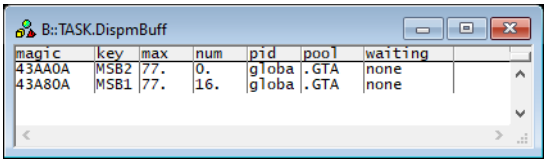
Format: **TASK.DispFixed**

Displays a table with all created fixed block memory pools.



Format: **TASK.DispmBuff**

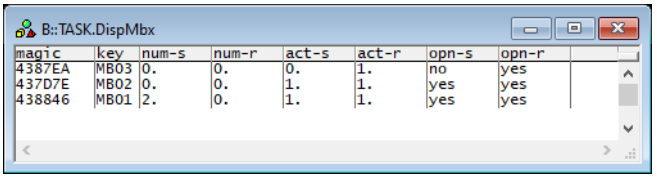
Displays a table with all created message buffers.



TASK.DispMbx Display mailboxes

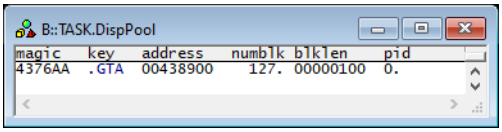
Format: **TASK.DispMbx**

Displays a table with all created mailboxes



Format: **TASK.DispPool**

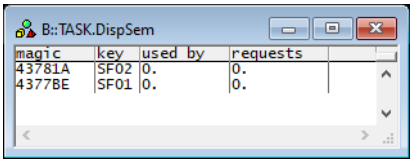
Displays a table with all created common memory pools.



TASK.DispSem — Display semaphores

Format: **TASK.DispSem**

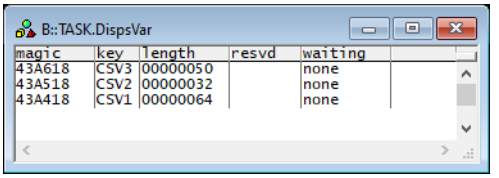
Displays a table with all created semaphores.



TASK.DispsVar — Display shared variables

Format: **TASK.DispsVar**

Displays a table with all created controlled shared variables

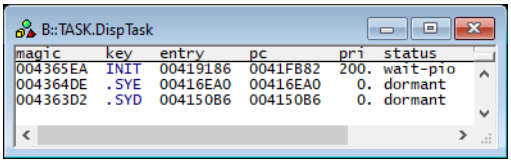


Format: **TASK.DispTask** [<*task*>]

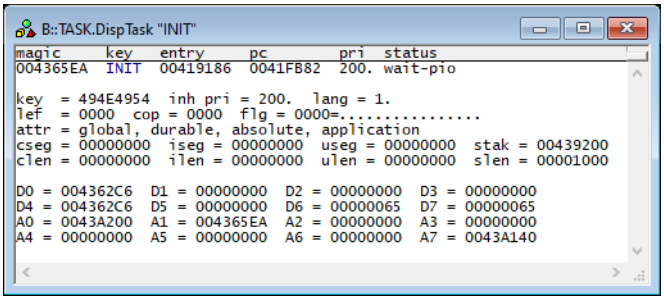
<*task*>: <*magic_number*> | <*task_key*>

Displays a table with all MTOS-UX tasks or one task in detail.

Without any parameters, a summary table of all created tasks is shown.

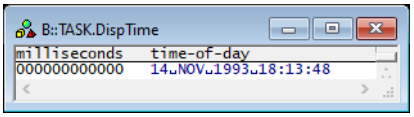


The magic number is a unique ID to the OS Awareness to specify a specific task. It is equal to the MTOS-UX task ID. A double click on the magic number or on the key opens the detailed task window. A double click on the entry address or on the pc address opens a **Data.List** window on this address.
If you specify a task magic number or a task key as parameter, this task is shown in detailed. An given task magic number is not checked for validation.



Format: **TASK.DispTIme**

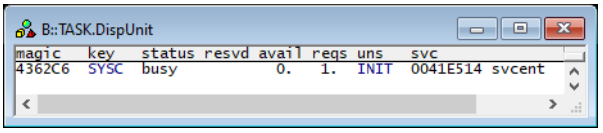
Displays a window with the milliseconds since MTOS-UX start and with the time-of-day string, reported by MTOS-UX.



TASK.DispUnit Display peripheral units

Format: **TASK.DispUnit**

Displays a table with all created peripheral I/O units.



TASK.MAP Mapping suggestion

Format: **TASK.MAP**

Display a mapping suggestion to have dual port access.

To display the MTOS-UX resources in real-time (“On-The-Fly”), emulation memory must be mapped to the MTOS-UX internal tables. If you do not know, where these tables are located (magics!), this command gives you a suggestion, which memory areas to map.

While leaving the window open, and creating more objects, the memory usage will be accumulated to the addresses used while running.

It is recommended to map more memory than suggested, this increases the chance, that new created object will be in the mapped memory area.

Format: **TASK.TASKState**

This command sets Alpha breakpoints on all task status words.

The statistic evaluation of task states (see **Task State Analysis**) requires recording of the accesses to the task state words. By setting Alpha breakpoints to this words, and selectively recording Alpha's, you can do a selective recording of task 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 tasks currently created. It does NOT set breakpoints to tasks, that terminated or haven't yet been created.

MTOS-UX PRACTICE Functions

There are special definitions for MTOS-UX 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