Debug Extension

to the

Real Time Executive Interface Definition

DRAFT 2.0

Prepared by:

MOTOROLA Microcomputer Division

Abstract:

This specification defines a basic set of functions that constitute the Debug Extension to the Real Time Executive Interface Definition. Draft 2.0 is for public review. MOTOROLA retains the right to modify this definition as appropriate during implementation. Draft 2.0 will be submitted to the VITA technical committee no later than 01/25/88.

 PRELIMINARY

DISCLAIMER

This Debug Extension to the Real Time Executive Interface Definition specification is being proposed to be used as the basis for formal standardization by the VME International Trade Association (VITA). However, since the standardization process has just begun, any standard resulting from this document might be different from this document. Any Product designed to this document might not be compatible with the final standard. No responsibility is assumed for such incompatibilities and no liability is assumed for any product built to conform to this document.

While considerable effort has been expended to make this document comprehensive, reliable, and unambiguous, it is still being published in preliminary form for public study and comment.

This document is prepared by Motorola Inc., Microcomputer Division. Interest in the Debug Extension to the RTEID is welcome and encouraged. Any technical questions, suggestions or comments may be directed to:

Motorola, Inc.

Microcomputer Division

Dept.: RTEID

2900 South Diablo Way

Tempe, Arizona 85282

Tel: (602)438-3500

Fax: (602)438-3581

Tlx: 4998071 (MOTPHE)

 TABLE OF CONTENTS

 1. DEBUG EXTENSIONS...................................................................................

 1.1 Debugging Tasks.......................................................................................

 1.1.1 Controlling Tasks............................................................................

 1.1.2 Read/Write Memory.......................................................................

 1.1.3 Read/Write Registers.....................................................................

 1.1.4 Exceptions in Tasks........................................................................

 1.1.5 The debug\_msg message queue......................................................

 1.1.6 Trace and Breakpoìnt......................................................................

 1.1.6 Trace and Breakpoint............................................................

1.1.6.1 Trace..................................................................................

1.1.6.2 Breakpoints........................................................................

 1.1.7 Directives........................................................................................

1.1.8 DB\_CONTROL..............................................................................

 NAME 5

 SYNOPSIS 5

 DESCRIPTION 5

 RETURN VALUE 5

 ERROR CONDITIONS 5

 NOTES 6

1.1.9 DB\_REMOTE.................................................................................

 NAME 7

 SYNOPSIS 7

 DESCRIPTION 7

 RETURN VALUE 7

 ERROR CONDITIONS 7

 NOTES 8

1.1.10 DB\_BLOCK.................................................................................

 NAME 9

 SYNOPSIS 9

 DESCRIPTION 9

 RETURN VALUE 9

 ERROR CONDITIONS 9

 NOTES 9

1.1.11 DB\_UNLOCK..............................................................................

 NAME 10

 SYNOPSIS 10

 DESCRIPTION 10

 RETURN VALUE 10

 ERROR CONDITIONS 10

 NOTES 10

1.1.12 DB\_GETMEM....................................................................................

 NAME 11

 SYNOPSIS 11

 DESCRIPTION 11

LIST OF FIGURES

 FIGURE 1. General Info Block.............................................................................................................30

 FIGURE 2. Task Info Block..................................................................................................................30

 FIGURE 3. Message Queue Info Block................................................................................................30

 FIGURE 4. Message Info Block...........................................................................................................30

 FIGURE 5. Semaphore Info Block.......................................................................................................31

FIGURE 6. Region Info Block.............................................................................................................31

FIGURE 7. Segment Info Block...........................................................................................................31

FIGURE 8. Partition Info Block...........................................................................................................31

FIGURE 9. Buffer Info Block..............................................................................................................32

|  |  |  |
| --- | --- | --- |
|  | REVISION RECORD |  |
| Issue | Revision Description | Date |
| 1 | Initial version. Internal Only. | 06/01/87 |
| 2 | Draft 2.0, limited distribution. | 01/25/88 |
| 3 |  |  |
|  |  |  |

1. DEBUG EXTENSIONS

The debug extensions to the RTEID support several features targeted for use in debugging tasks and interrupt service routines (ISR’s). Since debugging is inherently non-real time, systems running under debug control may not exhibit true real time performance.

1.1 Debugging Tasks

Most debugging can be performed by debugging a task or a collection of tasks. In this type of debugging, the actual debug task can reside on the local cpu, or it can be remote if the appropriate GLOBAL flags are set.

1.1.1 Controlling Tasks

The relationship between the debug task and the task being debugged is established using the db\_control directive in the “set” mode. The task issuing the db\_control directive in the set mode must provide a message queue. This message queue is used to communicate between the executive and the task that issued the db\_control directive. After completion of the db\_control directive, the task being debugged becomes controlled, and cannot compete for processor time unless directed to execute by the debug task using the db\_unblock directive. The db\_block directive is used to block execution of the controlled task. The db\_control directive in the "clear" mode is used to terminate the relationship between the debug task and the controlled task.

1.1.2 Read/Write Memory

To read and write memory belonging to the controlled task the pair of directives db\_getmem and db\_setmem are provided. Db\_getmem reads memory from an address of the controlled task and copies it to a buffer provided by the debug task for a length specified by the debug task. Db\_setmem writes memory to an address of the controlled task copying it from a buffer provided by the debug task for a length specified by the debug task.

1.1.3 Read/Write Registers

To read and write the processor registers belonging to the controlled task the pair of directives db\_getreg and db\_setreg are provided. Db\_getreg reads a register belonging to the controlled task and copies it to a buffer provided by the debug task. Db\_setreg writes to a register belonging to the controlled task by copying it from a buffer provided by the debug task.

1.1.4 Exceptions in Tasks

When a controlled task issues an exception, such as a bus error, the executive will prevent further execution by placing the controlled task in a blocked state. The executive will also format a message containing information about the exception and place it on the message queue identified by the debug task in the db\_control directive.

1.1.5 The debug\_msg message queue

The executive requires the ability to inform the debug task about abnormal activity that occurs when a controlled task executes. This is done by using a message queue specified by the debug task when the db\_control directive is issued. This message queue is used to pass information from the executive to the debug task. When a controlled task is running and suffers an exception, the

executive will block further execution of the task, and inform the debug task of the exception by posting a message on the debug\_msg queue. The format of the message is:

|  |  |
| --- | --- |
| Bytes | Meaning |
| 0..3 | Task id of task causing exception. |
| 4..7 | Exceptions vector offset. |
| 8..11 | Address of the Exception Stack Frame |
| 12..15 | Program counter at the point of the exception |

1.1.6 Trace and Breakpoint

A fundamental feature in debugging a task or ISR is the ability to control its execution. This is typically done either by causing the controlled task to single step one instruction, or by having the controlled task execute up to a particular breakpoint. With the debug extensions to the RTEID, a debugger can provide these features.

1.1.6.1 Trace

In order to single step, or trace, a controlled task, the debugger must manipulate the status register of the controlled task, cause it to resume execution, and then process the resulting exception.

Tracing can be accomplished by the following steps:

1. The debug task prevents further execution of the controlled task by issuing a db\_block directive.

2. The controlled task’s status register is read using the db\_getreg directive.

3. The debug task sets the trace bit in the status register, and writes it back using the db\_setreg directive.

4. The debug task then permits execution of the controlled task by issuing the db\_unblock directive.

5. Since the trace bit is set, when the controlled task executes it will take a trace exception.

6. When the trace exception occurs, the executive will block further execution of the controlled task and send a message to the debug task using the debug\_msg message queue specified in the db\_control directive.

7. The debug task can then receive the message, process it, and continue debugging the task.

1.1.6.2 Breakpoints

Breakpoints are accomplished in a similar fashion.

1. Execution of the controlled task is stopped using the db\_block directive.

2. The instruction at the breakpoint locations is read and saved using the db\_getmem directive.

3. The instruction is replaced with the breakpoint code using the directive.

4. The debug task then executes the controlled task with the db\_unblock directive.

5. The controlled task will execute until it reaches the breakpoint code. At this point it will take an exception.

6. The executive will block further execution of the debug task and post a message to the debug\_msg message queue specified in the db\_control directive.

7. The debugger will receive the message and perform the appropriate action.

1.1.7 Directives

The directives provided by the debug manager are:

|  |  |
| --- | --- |
| Directive | Function |
| db\_control | Control a task |
| db\_remote | Perform directive on remote cpu |
| db\_block | Prevent a task from running |
| db\_unblock | Run a task under control |
| db\_getmem | Get a task’s memory |
| db\_setmem | Set a task’s memory |
| db\_getreg | Get a task’s register |
| db\_setreg | Set a task’s register |

1.1.8 DB\_CONTROL

NAME

db\_control - “Control a Task During Debug"

SYNOPSIS

unit db\_control ( tid, mode, qid)

 uint tid; /\* task id as returned from t\_create or t\_ident \*/

 uint mode; /\* new mode \*/

 uint qid; /\* debug\_msg qid \*/

DESCRIPTION

Db\_control is used to establish or remove debug control over a task.

The tid parameter specifies the task to be controlled. This task may exist on the local processor, or any remote processor in the multiprocessing configuration if the task was created with the GLOBAL Bag set (see t\_create).

The mode specifies what type of action is to be performed when an exception occurs.

 DB\_TASK\_CONTROL set to establish control over task

 clear to remove control over task

These values are mutually exclusive.

The message queue identified by the qid parameter is used by the executive to report exceptions to the debug task. This queue must exist and if debugging is to be done on multiple cpu’s, then this queue must have been created with the GLOBAL flag set.

RETURN VALUE

If db\_control successfully completes, 0 is returned.

If the call was not successful, an error code is returned.

ERROR CONDITIONS

Invalid tid.

Task already under debug control.

NOTES

Not callable from ISR.

Asserting control over a task will place it in the blocked state.

Removing debug control from a task will unblock the task if it was blocked.

Will not cause a preempt when mode is set.

May cause a preempt when mode is clear by unblocking a higher priority task.