Embedding it better...

µTasker Document

µTasker – RAM Test

Table of Contents

1. Introduction

It is sometimes required that the integrity of the system RAM (Random Access Memory) be tested. This often takes place when the processor has just been reset but can also take place at regular intervals during operation.

This document details the implementation of the RAM test in the µTasker project, which allows the test of a complete RAM area, including all areas occupied by system variables, heap and stack. The test code supports execution during system operation; *the only exception being when peripheral DMA access requires RAM access during the test, in which case the peripheral's operation would need to be temporarily suspended.*

2. Activating the RAM Test and Basic Test Properties

The RAM test is incorporated in the file `application.c` and can be enabled by activating `#define RAM_TEST` in that file.

The demo project calls the RAM task from the function `fnUserHWInit()`, which is executed very early on in the initialisation sequence. This ensures that no peripheral DMA operation takes place which may cause corruption during the test but it is performed with the system already configured for it maximum operating speed.

It performs a complete test of all of the specified RAM area in one go, whereby the routines used are also designed to allow the test to be performed in small blocks, allowing blocks to be tested at regular intervals rather than all together.

During the test of each block of RAM interrupts are disabled to avoid interrupt routines using the area that are presently being tested (in the demo project's test there are no interrupt actually enabled during the test period).

Since a backup is made of each block to be tested, the test doesn't corrupt any system memory. The test is also designed to allow testing of all RAM areas, including those occupied by system stack and the test routine itself

The test call looks like this:

```
unsigned long *fnRAM_test(int iBlockNumber, int iBlockCount);

if (fnRAM_test(0, (SIZE_OF_RAM/RAM_BLOCK_SIZE)) !=
    (unsigned long *)0xffffffff) { // test code of a complete RAM area
    // The return address was the address in RAM that failed
    //
    while (1) {}             // serious error found in RAM - stop here
}
```

The complete RAM (`SIZE_OF_RAM`) is tested in blocks of `RAM_BLOCK_SIZE` (eg. 128 bytes). It is expected that the test will usually be successful, in which case the value `0xffffffff` is returned. Should a memory location fail, the returned pointer will point to this long word location (`0xffffffff` was chosen since no known internal RAM occupies this location).

3. Test Details with Handling of Stack Space Test

The RAM test of a single RAM block (example 128 bytes as defined by `RAM_BLOCK_SIZE`) is performed by first making a backup of the content of the block to be tested on the system stack.

Then the RAM test is performed on each long word location in the block – each location is written with `0x55555555`, tested for correctness, written with `0xaaaaaaaa` and checked again.

If an error is detected during the test the error location is remembered so that it can be returned as result. The test is otherwise interrupted since an error has already been identified.

Once the block test has completed the backup of the original content is returned.

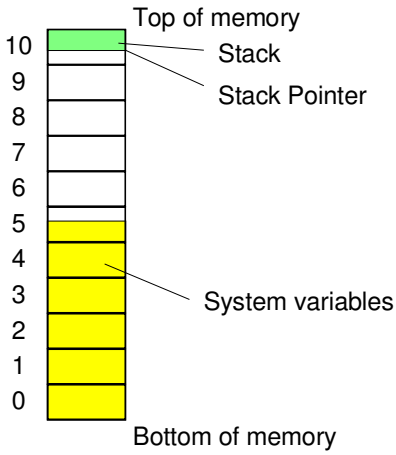
This means that the RAM block's content is not actually changed by the test; so that the content cannot be used by other processes during the brief period where is has test content, system interrupts are temporarily blocked.

Note that any peripheral DMA that could access the block would need to be specifically stopped if the test were being performed at a time when it could otherwise result in a possible corruption.

In order to test the complete RAM area the test is repeated for all blocks of RAM in it. For example, a 96kByte internal SRAM would require the test of 768 blocks of each 128 bytes in size.

This test is very simple as long as the test routine itself is not actually using the RAM which is to be tested. This complication is overcome by always checking whether this is the case before each block is tested and, in case there is a conflict, performing a recursive call to the test routine in order to move the test routine's stack space to another location in RAM. The test routine requires a stack space of slightly more that the RAM block size (due to the fact that it creates a backup buffer of that size on its stack), meaning that one or maximum two recursive calls will be necessary to avoid the collision. Therefore it can be concluded that the stack space required for the test is approximately maximum three times the `RAM_BLOCK_SIZE`, which is also useful to be aware of in case the `RAM_BLOCK_SIZE` is to be re-dimensioned. Larger block sizes require more stack space to be available; smaller block sizes allow individual blocks to be tested faster, remembering that each block test will also block interrupts during its execution period.

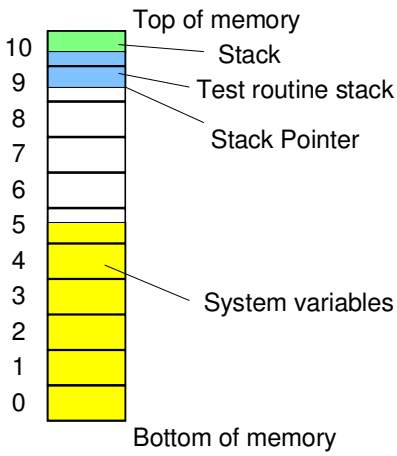
The operation, especially the recursive call of the routine to avoid stack collision and thus allow the test of the entire RAM area during system operation, is represented in diagrams below:



RAM area divided
in to test blocks (0..10)

The first diagram shows an area of RAM to be tested. It is divided into a number of test blocks (numbered 0..10). Test block 0 is at the bottom of memory and test block 10 at the top.

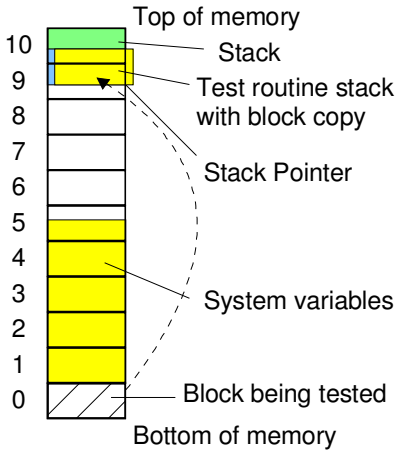
Typically the lower portion of RAM will contain system variables (and heap). The top of RAM will usually be used as system stack, whereby the stack grows downwards. The initial stack use and the location of the stack pointer is shown before the RAM test is called.



RAM area divided
in to test blocks (0..10)

The second diagram shows the RAM utilisation when the RAM test function has been called. The function has created some space on the stack for its working variables and a buffer large enough to hold a backup of the block that will be tested.

The stack has thus grown down.



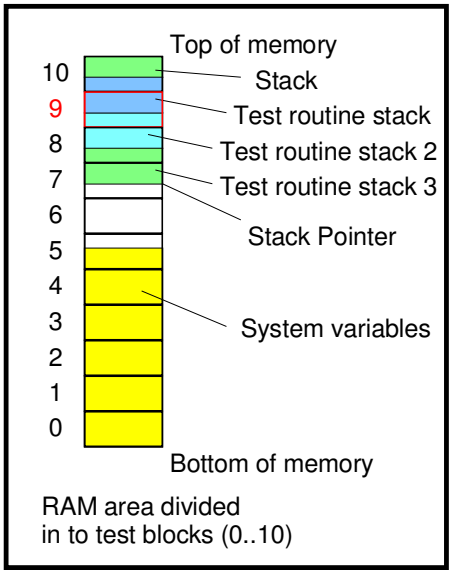
RAM area divided
in to test blocks (0..10)

Before the first block (test block 0) is tested a backup of its original content is made on the RAM test routine's stack.

The test block 0 can then be temporarily modified by the test routine to verify that all of its locations write and read correctly.

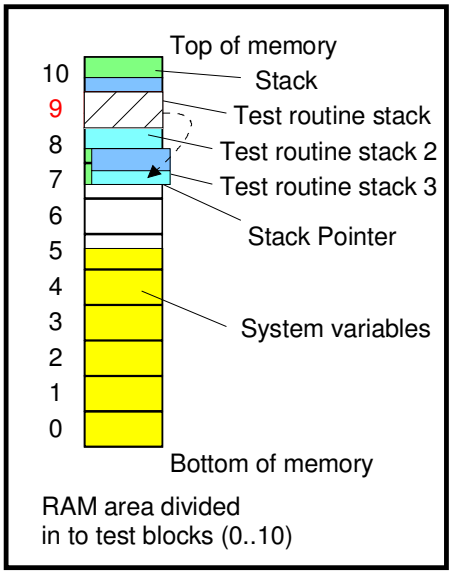
After the block test has been completed the backup can be returned so that the original content is achieved again.

This can be repeated without difficulty for test blocks 1, 2, 3 etc. up to 8.



When test block 9 is to be tested there is a complication since this block is being used by the RAM test routine. Since there is a collision it is not possible to make a backup and test the block without causing memory corruption to take place and likely cause the processor to crash.

Therefore the RAM test routine is called a second time (recursively) and creates a second stack for its use (stack 2). In this case it is seen that the second stack is lower in RAM but there is still some overlap. A second recursive call results in stack 3 being created which no longer overlaps with the test block 9.



The second recursive instance of the RAM test routine now performs the test of test block 9 by first making a backup to its stack space (without overlap).

Once test block 9 has been verified, its original content (in this case containing the stacks of the original test routine and the first recursive instance of the routine) is returned to test block 9.

The result is that the complete RAM content after the test is fully intact.

4. Conclusion

This document has detailed the operation of the RAM test included in the µTasker demo project. The test is designed to test all locations in a RAM area even when the test program is operating from stack within that area.

The test program is also designed to be able to be executed continuously during the operation of a system in case this is a requirement by periodically calling the test of each block within the RAM area.

It is to be noted that the stack should have adequate space to create three times the RAM_BLOCK size during the test (when its stack is also in the area). Since interrupts are blocked during each block test it tends to be beneficial to use small block sizes (128 bytes is default).

In the case of peripheral DMA which may use the same RAM area during the test, the peripheral should also be disabled during the test.

Modifications:

V1.00 30.07.2011 - Initial version