NS9360 Internal RAM Test

APPLICATION NOTE

Contents

NS9360 Internal RAM Test

This application note describes how to use the RAM test software.

Overview

The NS9360 processor has several internal RAMs. Because of an issue in the silicon manufacturing, some of the processors have shipped with faulty internal RAM.

The processors have a Built-In Self Test (BIST) that tests the internal RAMs; the BIST is run at the factory, and defective parts are discarded. Some early versions of the NS9360, however, had faulty test logic, which caused all the parts to pass the BIST — even those with bad RAM. Some parts were shipped before Digi became aware of this problem and have already been built into customer hardware. Digi has developed software to test the internal RAMs and uses it to test all the affected NS9360 parts

This application note describes how to integrate the test software into customer code. If you are using NET+OS, use the example applications and step-by-step instructions for customizing the test software. If you are not using NET+OS, you can follow a general procedure for integrating the test code into your application.

Distribution

The test code is supplied in a zip file with three directories:

Directory	Contents
ns9360test	The test code
examples/ns9360test	A simple example program that runs the tests. After the tests complete, the program continuously blinks the LEDs in a pattern that indicates the test results.
examples/naftpapp_9360test	An example program that runs the tests and then executes the NET+OS naftpapp example application if all tests pass

Hardware requirements

The LCD FIFO test requires a 3.6864MHz clock input on GPIO pin 15. You must configure the software to skip this test if the clock is not available. You do not need to test the LCD FIFO if your application does not use the LCD controller.

Modifications to the NET+OS BSP

Modifications to customize.lx or customize.ldr

The test code uses the upper 3 MB RAM, which you must reserve.

To reserve the upper 3 MB RAM:

1. In your `platform` directory, edit `customize.lx` (Green Hills) or `customize.ldr` (GNU Tools) by changing the value of `RAM_SIZE` to 3 MB less than the amount of RAM on your platform. (For example, if your hardware platform has 16 MB RAM, set `RAM_SIZE` to 13 MB.)
2. Rebuild the BSP.

Use this version of the BSP when you build an application with the NS9360 internal RAM tests. Use the standard value for `RAM_SIZE` when you build applications that do not include the NS9360 internal RAM tests.

Make sure that nothing uses the upper 3 MB RAM until after the test code completes.

Modifications to environment.h

To set up the configuration parameter values, edit `environment.h` as shown here:

Set this constant	To
ENV_RAM_SIZE	The amount of RAM on your hardware platform. For example, if your platform has 16 MB RAM, `set ENV_RAM_SIZE` to `16*ENV_MEGABYTE`.
`ENV_CONTINUE_TO_APPLICATION_IF_SUCCESS`	`1` if you want the application to continue running if all tests complete `0` if you want the application to blink the LEDs forever when the tests complete
`ENV_TEST_LCD_FIFO`	`1` if you want to test the LCD FIFO: `1. (`Requires the additional hardware described above.) `0` if your hardware does not support this test

Modifications to report.c

The `report.c` file contains the code that reports the test results. The `reportFailure` function is called if any test fails, and the `reportAllTestsPassed` function is called if all tests pass. The `reportFailure` function is passed a module code and a subcode that indicate the test that failed. The default implementation blinks LEDs on the board in different patterns to indicate these codes:

If	The module code	And the subcode
Two or more LEDs are available	Is blinked on LED 1	Is blinked on LED 2
Only one LED is available	Is blinked with slow blinks	Is blinked with fast blinks

If your hardware does not have LEDs, you must modify `report.c` to report the test results in a different way.

These codes are defined:

Module code	Subcode	Error condition
1	1	Failure in USB FIFO
1	2	Failure in BBUS DMA context RAM
1	3	Failure in BBUS cache RAM
2	1	Failure in high vector RAM
2	2	Unexpected exception occurred during the test. One possible cause is executing the LCD FIFO test without a clock input on GPIO pin 15. Otherwise, the problem is probably with ICache.
2	3	Failure in MMU lookaside RAM
3	1	Failure in Ethernet transmitter buffer descriptor RAM
3	2	Failure in Ethernet receive FIFO
3	3	Failure in Ethernet data FIFO
4	1	Failure in ICache valid bit RAM
4	2	Failure in ICache data RAM
4	3	Failure in ICache tag RAM
5	1	Failure in DCache valid bit RAM
5	2	Failure in DCache data RAM
5	3	Failure in DCache tag RAM
5	4	Failure in DCache dirty bit RAM
6	1	Failure in LCD FIFO
6	2	Failure in LCD Palette RAM

The default implementation blinks both LEDs in unison if all test pass.

Some of the tests use the serial ports in loop back mode. These tests may fail if a device is connected to any of the serial ports while the test is running.

Ns9360test application

The code in `src/examples/ns9360test` shows how to make a simple NET+OS application that brings up the hardware and runs the internal RAM tests.

To build `ns9360test`:

1. In your `platform` directory, edit `customize.lx` (Green Hills) or `customize.ldr` (GNU Tools) as described in "Modifications to the NET+OS BSP".
2 Rebuild the BSP.

3. To set the RAM size and control whether the LCD FFO test is executed, edit the environment.h file for ns9360test as described in "Modifications to environment.h."
4. Build the application as you normally do.

Naftpapp_9360test application

The code in src/examples/naftpapp_9360test shows how to integrate the test code into a standard NET+OS application. In this case, the test code has been integrated into the naftpapp application. The result is an application that tests the NS9360 chip, and if it passes, loads an FTP server that can be used to reprogram flash on the board.

To build naftpapp_9360test:

1 In your platform directory, edit customize.lx (Green Hills) or customize.ldr (GNU Tools) as described in "Modifications to the NET+OS BSP."

2 Rebuild the BSP.

3 To set the RAM size and control whether the LCD FIFO test is executed, edit the environment.h file for naftpapp_9360test as described in "Modifications to environment.h."

4 Build the application as you normally do.

To use this application as an example of how to include the tests in another application, use this procedure:

1. Copy the main.c file from src\bsp\common in your NET+OS distribution directory to your application directory. Edit the copy in your application directory, and rename the main function realMain().
2. Copy the report.c and environment.h files from naftpapp_9360test to your application directory.
3. In your platform directory, edit customize.lx (Green Hills) or customize.ldr (GNU Tools) as described in "Modifications to the NET+OS BSP."
4. Rebuild the BSP.

   To set the RAM size and control whether the LCD FIFO test is executed, edit the environment.h file for this application as described in "Modifications to the NET+OS BSP."
5. Using the Makefile or build file in naftpapp_9360test as an example, edit the Makefile (GNU Tools) or build file (Green Hills) for your application so that the files in src/ns9360 are included in the build. Make sure source and include paths are set up to src/ns9360test.
6. Build your application as you normally do.

Libraries

Do not create a separate library for the files in src/ns9360test.

Some of the objects generated from these files override objects in the BSP library. To ensure the linker uses the objects from src/ns9360test instead of the ones in the BSP

library, you must specify the ones from src/ns9360test in the command line as separate objects. This is already done for you in the GNU Makefile and Green Hills gpj file supplied with the two example applications.

To specify additional objects, either add more source file names to the APP_C_FILES and APP_ASM_FILES variables in the Makefiles (GNU Tools) or add more source files to the project.gpj files (Green Hills).

Building a non-NET+OS test application

If you do not use NET+OS, you can still use the code in src/ns9360test and link it into an application as described above.

To create your test application:

1. Adjust your linker scripts so that the upper 3 MB RAM are not used until after the processor tests have completed.
2. Have your application execute the low level hardware initialization code as normal, and call the C-Library initialization routines that initialize the C runtime environment.
3. The file ns9360main.c in src/ns9360test contains a replacement for the main() routine called by the C-Library after it has initialized the C runtime environment. The test code calls realMain(), an internal routine, if all tests pass. To integrate the test code, rename the original main() routine to realMain() so the C-Library calls the processor test code and your application's main() routine is called if all tests pass.
4. Make sure the MMU/Cache initialization code is not executed until after the processor tests complete.

   (The upper 3 MB RAM can be used after the processor tests complete.)
5. Update your application Makefile so that the code in src/ns9360test is compiled and linked into your application.
6. Implement the reportTestsStarting, reportFailure, and reportAllTestsPassed routines.

   (Example implementations are provided in the report.c files in src/ns9360test and src/naftpapp_9360test.)
7. Copy a version of environment.h from one of the example applications to a directory in the include path. Then edit this file as described in "Modifications to environment.h."
8. Some ARM assembler files in src/ns9360test have the .arm extension. The Green Hills and GNU tool sets preprocess .arm files with the C preprocessor. If your toolset does not do this, you must create rules in your Makefile that use the C preprocessor to process these files before the assembler is invoked.