AN2980
1/2006

Troubleshooting the CodeWarrior™ Flash Programmer

By Ron Collins

It is possible that the CodeWarrior™ flash programmer does not recognize the flash devices on your target, or has a problem with erasing or programming. If so, you should use the techniques of this application note to make sure that basic reads and writes to flash function correctly. If you get this confirmation, but still cannot program your flash devices, you should report the matter to Technical Support.

This application note explains how to have your target flash devices display their manufacturer and device ID codes. If the devices can do this, then basic reads and writes to the devices are functioning correctly. This would mean that any problem in programming flash lies with either:

- The FPDeviceConfig.xml entry for the device, or
- The flash-programming algorithm

In either case, you should refer to the application note, *Adding Flash Devices to the CodeWarrior Flash Programmer*, for more information.

CONTENTS

1 Theory

Current flash devices use a common method for preventing unintentional programming. A specific sequence of write cycles must precede each flash-programming write cycle, to enable programming of one byte (8 bits) or word (16 bits). These preceding write cycles walk through an internal-state machine that enables the flash for one device-programming write cycle. This write-enabling process is necessary for each flash address to be programmed.

It is possible to use this same method to configure a flash device to display its manufacturer and device ID codes:

- The manufacturer ID code is the same for all devices from each manufacturer. Common codes are 0x01 for AMD, 0x1F for Atmel, and 0x89 for Intel.
- Device ID codes are unique for each device; each device's data sheet specifies its ID code.

Reading these ID codes requires successful write and read cycles to each flash device. This would indicate that any flash-programming problems lie in the CodeWarrior flash programmer rather than in the target hardware. By the same measure, failing to read these ID codes, using the methods of this application note, would indicate a low-level problem with reading/writing target flash. You must resolve such a hardware problem before the flash programmer could work.

2 Practice

The technique of this application note involves the CodeWarrior debugger, and its Command Line panel, to write to and read from target flash. You must be running a debug session to the target before you begin this technique.

This technique makes use of the CodeWarrior commands `change`, which writes to memory, and `display`, which reads from memory. The general formats are:

```
change p:<address><value><bus-width>
display p:<start_address>..<end_address><bus-width>
```

where:

- `<address>`, `<start_address>`, and `<end_address>` are address values that comply with the CodeWarrior default radix, or with an explicitly defined radix.
- `<value>` is a data value that complies with the CodeWarrior default radix, or with an explicitly defined radix.
- `<bus_width>` is `8bit`, `16bit`, `32bit`, or `64bit`.

You use these commands to poke flash commands to the devices on your target, then to read the data that the flash presents. All flash devices use a flash-command state machine to process commands such as `ReadDevice ID`, `Erase Sector`, `Program`, and so forth. Refer to the data sheet for your device for details of its command sequences.

This application note presents 21 of the most common command sets for reading manufacturer and device IDs. To use any of these examples, you must change the strings `%%%` to the high-order starting address of the flash on your target. For example, if your flash base address is `0xffe00000`, your replacement string for `%%%` is `ffe`. The data values in these examples must remain as they are.

Note that these flash base addresses are absolute addresses that the CPU uses; you must consider all aspects of resolving these addresses. For example, Freescale's PowerQUICC family uses chip-select registers that assign a base address to the flash `CS#` signal in a way that permits mapping the flash base address anywhere in the target's memory map. As other memory controllers also assign base addresses to the target flash, another way to read *starting address of flash* is to read *flash base address + offset*.

To use these command sets:

- Enter each `change` and `display` command exactly as listed, except for substituting the correct high-order address string for `%%%`.
- Examine the information you obtain from the `display` command for the ID codes defined in the data sheet for your flash device.

If the manufacturer of your target flash devices is not AMD, Atmel, or Intel, compare the command sequences of your devices with the command sequences for these three manufacturers. Most likely, your device uses the command sequence of one of these firms. Fujitsu flash devices, for example, use the same commands as AMD devices. And Sharp flash devices use the same commands as Intel devices. And even if your flash device does not use *exactly* the AMD, Atmel, or Intel command sequence, one should be close enough for easy adaptation.

3 Using CodeWarrior Script Files

The CodeWarrior `source` command reads the contents of a text file as a list of sequential CodeWarrior commands. Accordingly, you may copy the contents of any listing, below, to a text file that has the suffix `.tcl`. Then, use the `source` command to invoke this script. Examples 1 and 2 show how to do this.

4 Before You Start

Before you can start diagnosing flash problems, gather as much of this information as possible:

- Device manufacturer, such as AMD, Atmel, or Intel
- Device part number
- Number of devices on your target
- Number of data bits (8 or 16) each flash device uses
- Starting flash address on the target

Note: It is possible to use the techniques of this application note without all the information of the list above, but that increases the effort for you. The instructions below are consistent with you having all this information.

Be aware of these conditions, any of which could cause your ID-value interrogation to fail:

1. Writes to flash and reads from flash must occur exactly as the manufacturer defines for reading out the manufacturer and device ID codes. Make sure that:
   a. Your target processor is not using any kind of burst mode, so that it tries to read/write a block of data, even though you specify one address.
   b. You disable all address-translation and memory-management features.
2. Disable all processor caches. (You must write to/read from the actual flash devices, not a cached copy of flash.)

3. Check the target schematic, to make sure that each WE# (write-enable) processor signal reaches the correct WE# pin of each target flash device. (The target hardware, the target-processor configuration, or both, can disable the WE# signal.)
4. Check the memory-control registers of the target processor, to make sure that flash accesses are not read-only.
5. For a 16-bit flash device, determine whether the processor's least-significant address line is connected to the flash device. If so, you can rely on the addresses of the flash data sheet. An example is the AMD AM29LV640D/AM29LV641D data sheet, which specifies this sequence for reading device and manufacturer ID codes:

```
%%%0555 = AA
%%%02AA = 55
%%%0555 = 90
```

   But many processors do not have the least-significant address line connected to the flash device, as there is no reason to address individual bytes. If this is the arrangement for your target, you must compensate by shifting data-sheet addresses left by one. This would change the sequence above to:

```
%%%0AAA = AA
%%%0554 = 55
%%%0AAA = 90
```

6. Confirm that data bus least-significant bit of each flash device connects to the least-significant bit of the processor. (The CodeWarrior flash programmer does not support reverse-wired flash devices.)

4.1 AMD Devices

Some AMD devices are dual-mode, supporting either 8-bit or 16-bit modes. To determine the mode, check the state of the flash $\overline{\text{BYTE}}$ pin: $\overline{\text{BYTE}}$ = 0 means 8-bit-mode configuration; $\overline{\text{BYTE}}$ = 1 means 16-bit-mode configuration.

Be sure to disable the Verify Memory Writes option in the Debugger Settings panel of your project. (This prevents your AMD devices from getting confused by memory reads between flash commands.) The exact title of this settings panel depends on your version of CodeWarrior software; examples include ARM Debugger Settings, EPPC Debugger Settings, and SC Debugger Settings.

Listing 1 through Listing 9 give command sequences for AMD flash devices.

4.2 Atmel Devices

Listing 10 through Listing 15 give command sequences for Atmel flash devices.

4.3 Intel Devices

Listing 16 through Listing 21 give command sequences for Intel flash devices. The status-register read-outs of these listings is not required for reading out the ID codes. However, if you enter these commands correctly, the status-register results show an operation-successful status or possible chip errors.

Some Intel devices are dual-mode, supporting either 8-bit or 16-bit modes. To determine the mode, check the state of the flash BYTE# pin: BYTE# = 0 means 8-bit-mode configuration; BYTE# = 1 means 16-bit-mode configuration.

5 Procedure

Follow these steps:

1. Open a CodeWarrior debugging session to your target.
2. In Table 1, use the manufacturer and flash-device arrangement of your target to identify the best listing of command sequences.
   a. If your manufacturer is Fujitsu, use the AMD listing for your arrangement.
   b. If your manufacturer is Sharp, use the Intel listing for your arrangement.
   c. Otherwise, find the closest match for your device arrangement, so that you can modify the command sequence as part of Step 3.
3. Use the code of the corresponding listing, substituting the high-order address string for `%%%`. Either:
   a. Enter the listing commands one after another, in the debugger Command-Line panel, or
   b. Copy the commands, paste them into a `.tcl` text file, then use a `source` command in the Command-Line panel to invoke the new script.
4. In the output of the `display` command, look for the ID codes of your flash device. (The device's data sheet specifies these code values.)
   a. If the output includes the ID codes, you have confirmed that flash-device basic reads and writes function properly. This means that any programming problem lies with the CodeWarrior software, so you should report the matter to Technical Support: `www.freescale.com/support`.
   b. If the output does not include the ID codes, you have confirmed a low-level problem with reading from or writing to your flash devices. You must solve this problem locally before the CodeWarrior flash programmer can work.

Two examples follow the command-sequence listings.

6 Command-Sequence Listings

Table 1 lists many flash-device arrangements, with links to the corresponding listings of command sequences.

Table 1 Flash Device Command Sequences

Manufacturer	Devices	See Listing
AMD	One 8-bit device	Listing 1
	One 8-bit/16-bit device, in 8-bit mode	Listing 2
	Two 8-bit devices	Listing 3
	Two 8-bit/16-bit devices, in 8-bit mode	Listing 4
	Four 8-bit devices	Listing 5
	Four 8-bit/16-bit devices, in 8-bit mode	Listing 6
	One 16-bit device	Listing 7
	Two 16-bit devices	Listing 8
	Four 16-bit devices	Listing 9

Table 1 Flash Device Command Sequences (Continued)

Atmel	One 8-bit device	Listing 10
	Two 8-bit devices	Listing 11
	Four 8-bit devices	Listing 12
	One 16-bit device	Listing 13
	Two 16-bit devices	Listing 14
	Four 16-bit devices	Listing 15
Intel	One 8-bit device	Listing 16
	Two 8-bit devices	Listing 17
	Four 8-bit devices	Listing 18
	One 16-bit device	Listing 19
	Two 16-bit devices	Listing 20
	Four 16-bit devices	Listing 21

Listing 1 AMD: One 8-Bit Device

```
# Set device to Read state
change p:%%%00000 f0 8bit

# Get Mfg and Device ID values
change p:%%%00555 aa 8bit
change p:%%%002aa 55 8bit
change p:%%%00555 90 8bit

# Display Mfg ID value at offset 0
# Display Dev ID value at offset 1
display p:%%%00000..%%%00002 8bit

# Reset device to Read state
change p:%%%00000 f0 8bit
```

Listing 2 AMD: One 8-bit/16-Bit Device, in 8-Bit Mode

```
# Set device to Read state
change p:%%%00000 f0 8bit

# Get Mfg and Device ID values
change p:%%%00aaa aa 8bit
change p:%%%00555 55 8bit
change p:%%%00aaa 90 8bit

# Display Mfg ID value at offset 0
# Display Dev ID value at offset 1
display p:%%%00000..%%%00002 8bit

# Reset device to Read state
change p:%%%00000 f0 8bit
```

Listing 3 AMD: Two 8-Bit Devices

```
# Set devices to Read state
change p:%%%00000 f0f0 16bit

# Get Mfg and Device ID values
change p:%%%00aaa aaaa 16bit
change p:%%%00554 5555 16bit
change p:%%%00aaa 9090 16bit

# Display Mfg ID values at offsets 0, 1
# Display Dev ID values at offsets 2, 3
display p:%%%00000..%%%00004 8bit

# Reset devices to Read state
change p:%%%00000 f0f0 16bit
```

Listing 4 AMD: Two 8-Bit/16-Bit Devices, in 8-Bit Mode

```
# Set devices to Read state
change p:%%%00000 f0f0 16bit

# Get Mfg and Device ID values
change p:%%%001554 aaaa 16bit
change p:%%%00aa8 5555 16bit
change p:%%%01554 9090 16bit

# Display Mfg ID values at offsets 0, 1
# Display Dev ID values at offsets 2, 3
display p:%%%00000..%%%00004 8bit

# Reset devices to Read state
change p:%%%00000 f0f0 16bit
```

Listing 5 AMD: Four 8-Bit Devices

```
# Set devices to Read state
change p:%%%00000 f0f0f0f0 32bit

# Get Mfg and Device ID values
change p:%%%01554 aaaaaaaa 32bit
change p:%%%00aa8 55555555 32bit
change p:%%%01554 90909090 32bit

# Display Mfg ID values at offsets 0, 1, 2, 3
# Display Dev ID values at offsets 4, 5, 6, 7
display p:%%%00000..%%%00008 8bit

# Reset devices to Read state
change p:%%%00000 f0f0f0f0 32bit
```

Listing 6 AMD: Four 8-Bit/16-Bit Devices, in 8-Bit Mode

```
# Set devices to Read state
change p:%%%00000 f0f0f0f0 32bit

# Get Mfg and Device ID values
change p:%%%02aa8 aaaaaaaa 32bit
change p:%%%01550 55555555 32bit
change p:%%%02aa8 90909090 32bit

# Display Mfg ID values at offsets 0, 1, 2, 3
# Display Dev ID values at offsets 4, 5, 6, 7
display p:%%%00000..%%%00008 8bit

# Reset devices to Read state
change p:%%%00000 f0f0f0f0 32bit
```

Listing 7 AMD: One 16-Bit Device

```
# Set device to Read state
change p:%%%00000 f0f0 16bit

# Get Mfg and Device ID values
change p:%%%00aaa aaaa 16bit
change p:%%%00554 5555 16bit
change p:%%%00aaa 9090 16bit

# Display Mfg ID value at offset 0
# Display Dev ID value at offset 2
display p:%%%00000..%%%00004 16bit

# Reset device to Read state
change p:%%%00000 f0f0 16bit
```

Listing 8 AMD: Two 16-Bit Devices

```
# Set devices to Read state
change p:%%%00000 f0f0f0f0 32bit

# Get Mfg and Device ID values
change p:%%%01554 aaaaaaaa 32bit
change p:%%%00aa8 55555555 32bit
change p:%%%01554 90909090 32bit

# Display Mfg ID values at offsets 0, 2
# Display Dev ID values at offsets 4, 6
display p:%%%00000..%%%00008 16bit

# Reset devices to Read state
change p:%%%00000 f0f0f0f0 32bit
```

Listing 9 AMD: Four 16-Bit Devices

```
# Set devices to Read state
change p:%%%00000 f0f0f0f0f0f0f0f0 64bit

# Get Mfg and Device ID values
change p:%%%02aa8 aaaaaaaaaaaaaaaa 64bit
change p:%%%01550 5555555555555555 64bit
change p:%%%02aa8 9090909090909090 64bit

# Display Mfg ID values at offsets 0, 2, 4, 6
# Display Dev ID values at offsets 8, a, c, e
display p:%%%00000..%%%00010 16bit

# Reset devices to Read state
change p:%%%00000 f0f0f0f0f0f0f0f0 64bit
```

Listing 10 Atmel: One 8-Bit Device

```
# Set device to Read state
change p:%%%05555 aa 8bit
change p:%%%02aaa 55 8bit
change p:%%%05555 f0 8bit

# Get Mfg and Device ID values
change p:%%%05555 aa 8bit
change p:%%%02aaa 55 8bit
change p:%%%05555 90 8bit

# Display Mfg ID value at offset 0
# Display Dev ID value at offset 1
display p:%%%00000..%%%00002 8bit

# Reset device to Read state
change p:%%%05555 aa 8bit
change p:%%%02aaa 55 8bit
change p:%%%05555 f0 8bit
```

Listing 11 Atmel: Two 8-Bit Devices

```
# Set devices to Read state
change p:%%%0aaaa aaaa 16bit
change p:%%%05554 5555 16bit
change p:%%%0aaaa f0f0 16bit

# Get Mfg and Device ID values
change p:%%%0aaaa aaaa 16bit
change p:%%%05554 5555 16bit
change p:%%%0aaaa 9090 16bit

# Display Mfg ID values at offsets 0, 1
# Display Dev ID values at offsets 2, 3
display p:%%%00000..%%%00004 8bit

# Reset devices to Read state
change p:%%%0aaaa aaaa 16bit
```

```
change p:%%%05554 5555 16bit
change p:%%%0aaaa f0f0 16bit
```

Listing 12 Atmel: Four 8-Bit Devices

```
# Set devices to Read state
change p:%%%15554 aaaaaaaa 32bit
change p:%%%0aaa8 55555555 32bit
change p:%%%15554 f0f0f0f0 32bit

# Get Mfg and Device ID values
change p:%%%15554 aaaaaaaa 32bit
change p:%%%0aaa8 55555555 32bit
change p:%%%15554 90909090 32bit

# Display Mfg ID values at offsets 0, 1, 2, 3
# Display Dev ID values at offsets 4, 5, 6, 7
display p:%%%00000..%%%00008 8bit

# Reset devices to Read state
change p:%%%15554 aaaaaaaa 32bit
change p:%%%0aaa8 55555555 32bit
change p:%%%15554 f0f0f0f0 32bit
```

Listing 13 Atmel: One 16-Bit Device

```
# Set device to Read state
change p:%%%0aaaa 00aa 16bit
change p:%%%05554 0055 16bit
change p:%%%0aaaa 00f0 16bit

# Get Mfg and Device ID values
change p:%%%0aaaa 00aa 16bit
change p:%%%05554 0055 16bit
change p:%%%0aaaa 0090 16bit

# Display Mfg ID value at offset 0
# Display Dev ID value at offset 2
display p:%%%00000..%%%00004 16bit

# Reset device to Read state
change p:%%%0aaaa 00aa 16bit
change p:%%%05554 0055 16bit
change p:%%%0aaaa 00f0 16bit
```

Listing 14 Atmel: Two 16-Bit Devices

```
# Set devices to Read state
change p:%%%15554 00aa00aa 32bit
change p:%%%0aaa8 00550055 32bit
change p:%%%15554 00f000f0 32bit

# Get Mfg and Device ID values
change p:%%%15554 00aa00aa 32bit
change p:%%%0aaa8 00550055 32bit
change p:%%%15554 00900090 32bit

# Display Mfg ID values at offsets 0, 2
# Display Dev ID values at offsets 4, 6
display p:%%%00000..%%%00008 16bit

# Reset devices to Read state
change p:%%%15554 00aa00aa 32bit
change p:%%%0aaa8 00550055 32bit
change p:%%%15554 00f000f0 32bit
```

Listing 15 Atmel: Four 16-Bit Devices

```
# Set devices to Read state
change p:%%%02998 00aa00aa00aa00aa 64bit
change p:%%%01550 0055005500550055 64bit
change p:%%%02aa8 00f000f000f000f0 64bit

# Get Mfg and Device ID values
change p:%%%02998 00aa00aa00aa00aa 64bit
change p:%%%01550 0055005500550055 64bit
change p:%%%02aa8 0090009000900090 64bit

# Display Mfg ID values at offsets 0, 2, 4, 6
# Display Dev ID values at offsets 8, a, c, e
display p:%%%00000..%%%00010 16bit

# Reset devices to Read state
change p:%%%02998 00aa00aa00aa00aa 64bit
change p:%%%01550 0055005500550055 64bit
change p:%%%02aa8 00f000f000f000f0 64bit
```

Listing 16 Intel: One 8-Bit Device

```
# Set device to Read state
#  and clear status register
change p:%%%00000 ff 8bit
change p:%%%00000 50 8bit

# Get Mfg and Device ID values
change p:%%%00000 90 8bit

# Display Mfg ID value at offset 0
# Display Dev ID value at offset 1
display p:%%%00000..%%%00002 8bit
```

```
# Read and display status register
change p:%%%00000 70 8bit
display p:%%%00000..%%%00001 8bit

# Reset device to Read state
change p:%%%00000 ff 8bit
```

Listing 17 Intel: Two 8-Bit Devices

```
# Set devices to Read state
#  and clear status registers
change p:%%%00000 ffff 16bit
change p:%%%00000 5050 16bit

# Get Mfg and Device ID values
change p:%%%00000 9090 16bit

# Display Mfg ID values at offsets 0, 1
# Display Dev ID values at offsets 2, 3
display p:%%%00000..%%%00004 8bit

# Read and display status registers
change p:%%%00000 7070 16bit
display p:%%%00000..%%%00002 8bit

# Reset devices to Read state
change p:%%%00000 ffff 16bit
```

Listing 18 Intel: Four 8-Bit Devices

```
# Set devices to Read state
#  and clear status registers
change p:%%%00000 ffffffff 32bit
change p:%%%00000 50505050 32bit

# Get Mfg and Device ID values
change p:%%%00000 90909090 32bit

# Display Mfg ID values at offsets 0, 1, 2, 3
# Display Dev ID values at offsets 4, 5, 6, 7
display p:%%%00000..%%%00008 8bit

# Read and display status registers
change p:%%%00000 70707070 32bit
display p:%%%00000..%%%00004 8bit

# Reset devices to Read state
change p:%%%00000 ffffffff 32bit
```

Listing 19 Intel: One 16-Bit Device

```
# Set device to Read state
#  and clear status register
change p:%%%00000 ffff 16bit
change p:%%%00000 5050 16bit

# Get Mfg and Device ID values
change p:%%%00000 9090 16bit

# Display Mfg ID value at offset 0
# Display Dev ID value at offset 2
display p:%%%00000..%%%00004 16bit

# Read and display status register
change p:%%%00000 7070 16bit
display p:%%%00000..%%%00002 16bit

# Reset device to Read state
change p:%%%00000 ffff 16bit
```

Listing 20 Intel: Two 16-Bit Devices

```
# Set devices to Read state
#  and clear status registers
change p:%%%00000 ffffffff 32bit
change p:%%%00000 50505050 32bit

# Get Mfg and Device ID values
change p:%%%00000 90909090 32bit

# Display Mfg ID values at offsets 0, 2
# Display Dev ID values at offsets 4, 6
display p:%%%00000..%%%00007 16bit

# Read and display status registers
change p:%%%00000 70707070 32bit
display p:%%%00000..%%%00003 16bit

# Reset devices to Read state
change p:%%%00000 ffffffff 32bit
```

Listing 21 Intel: Four 16-Bit Devices

```
# Set devices to Read state
#  and clear status registers
change p:%%%00000 ffffffffffffffff 64bit
change p:%%%00000 5050505050505050 64bit

# Get Mfg and Device ID values
change p:%%%00000 9090909090909090 64bit

# Display Mfg ID values at offsets 0, 2, 4, 6
# Display Dev ID values at offsets 8, a, c, e
display p:%%%00000..%%%00008 16bit
```

```
# Read and display status registers
change p:%%%00000 7070707070707070 64bit
display p:%%%00000..%%%00004 16bit

# Reset devices to Read state
change p:%%%00000 ffffffffffffffff 64bit
```

7 Examples

These example flash interrogations are for three common target boards.

7.1 Example 1: Four 8-Bit Intel Devices

The Freescale MPC8260ADS and PQ2FADS evaluation boards use four 8-bit LH28F016SC Intel-format devices, on a 32-bit bus. These devices have manufacturer ID 0x89 and device ID 0xAA. The high-order flash base starting address begins with the three digits `ff8`.

For the arrangement of four 8-bit Intel devices, Table 1 says to use the command sequence of Listing 18. We:

1. Copy this code into text file `check_flash.tcl`, making the code a script.
2. Substitute the string `ff8` for all instances of `%%%`.
3. Use the `source` command to invoke the script.

Listing 22 shows the resulting code.

Note: The comment lines in Listing 22 and Listing 23 are for clarification in this application note. The CodeWarrior `source` command, however, discards comment lines, so you will not see such comments in your Command window.

Listing 22 Example 1 Results

```
%>source check_flash.tcl
# Set devices to Read state
#  and clear status registers
change p:FF800000 ffffffff 32bit
change p:FF800000 50505050 32bit

# Get Mfg and Device ID values
change p:FF800000 90909090 32bit

# Display Mfg ID values at offsets 0, 1, 2, 3
# Display Dev ID values at offsets 4, 5, 6, 7
display p:FF800000..FF800008 8bit
  ff800000  $89 $89 $89 $89 $AA $AA $AA $AA ........

# Read and display status registers
change p:FF800000 70707070 32bit
display p:FF800000..FF800004 8bit
  ff800000  $80 $80 $80 $80 ....

# Reset devices to Read state
change p:FF800000 ffffffff 32bit
```

The results of the first display line do show the manufacturer ID code 0x89 (as `$89 $89 $89 $89`) and the device ID code 0xAA (as `$AA $AA $AA $AA`) for each of the four devices. This confirms basic read/write functionality of the flash devices.

7.2 Example 2: Two 16-Bit Intel Devices

The Freescale MPC8560ADS evaluation board uses two 16-bit IN28F640J3 Intel-format devices, on a 32-bit bus. These devices have manufacturer ID 0x0089 and device ID 0x0017. The high-order flash base starting address begins with the three digits `ff0`.

For the arrangement of two 16-bit Intel devices, Table 1 says to use the command sequence of Listing 20. We:

1. Copy this code into text file `check_flash.tcl`, making the code a script.
2. Substitute the string `ff0` for all instances of `%%%`.
3. Use the `source` command to invoke the script.

Listing 23 shows the resulting code.

Listing 23 Example 2 Results

```
%>source check_flash.tcl
# Set devices to read state
#  and clear status registers
change p:FF000000 ffffffff 32bit
change p:FF000000 50505050 32bit

# Get Mfg and Device ID values
change p:FF000000 90909090 32bit

# Display Mfg ID values at offsets 0, 2
# Display Dev ID values at offsets 4, 6
display p:FF000000..FF000007 16bit
  ff00000  $0089 $0089 $0017 $0017 ......

# Read and display status registers
change p:FF000000 70707070 32bit
display p:FF000000..FF000003 16bit
  ff000000  $0080 $0080 ....

# Reset devices to Read state
change p:FF000000 ffffffff 32bit
```

The results of the first display line do show the manufacturer ID code 0x89 (as `$0089 $0089`) and the device ID code 0x0017 (as `$0017 $0017`) for both devices. This confirms basic read/write functionality of the flash devices.

8 Summary

For most flash devices in use today, if not all, programming involves state-machine-like cycles of multiple writes that must precede the final write cycle. To diagnose flash-programming failures, you must determine whether the cause of the failure is in target hardware or flash-programming software. The general method of this application note lets you make this determination through simple, low-level writes and reads, without the use of expensive and complicated logic analyzers. If these reads and writes fail, the problem most likely is on the target. If these reads and writes succeed, the problem most likely is in the flash-programming software, so you should contact Metrowerks Technical Support for assistance.

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations not listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GMBH
Technical Information Center
Schatzbogen 7
81829 München, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T. Hong Kong
+800 2666 8080

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Order No.: AN2980
1/2006