AN1806/D
(Motorola Order Number)

7/1999
REV. 0

Application Note

Initializing Blank Flash Devices on the PowerPC™ System Bus

by Gary Milliorn
Motorola RISC Applications
risc10@email.sps.mot.com

This document describes the steps required for an embedded controller or computer system to initialize a blank Flash memory device attached to the PowerPC system bus. This document contains the following topics:

The PowerPC memory controller family (the MPC106, MPC107, and the integrated MPC8240—which includes an MPC107-like core) all support the ability to load start-up code from a Flash device (called the "boot ROM"). This Flash device may be located on

This document contains information on a new product under development by Motorola. Motorola reserves the right to change or discontinue this product without notice.

© Motorola, Inc., 1999, 2000. All rights reserved.

the local PowerPC data bus or on the PCI bus. For performance or system architecture reasons, it is usually desirable to have the boot ROM on the local bus.

When an embedded system is manufactured with one of the PowerPC memory controllers, the boot ROM may be blank or may need to be updated from a master source. If a boot-sector Flash (with appropriate software) is not appropriate or not available, the Flash cannot be programmed in-place.

All of the PowerPC memory controller devices listed above share a common architecture, and so all share the same restrictions that prevent the most obvious solutions from working:

- External PCI masters cannot write to the Flash/ROM addresses.
- External PCI masters cannot configure the target system sufficiently to force it to boot into a configured and downloaded DRAM (the target must initialize its own memory, which requires it to run a program, which in turn requires a non-blank Flash).
- When the controller is configured to boot from a PCI-hosted ROM, it loses the access to the local boot ROM space (controlled by $\overline{\text{RCS0}}$).
- When the controller is configured to boot from local ROM, it loses access to the PCI boot ROM space (usually provided by a PCI-to-ISA bridge).

The following sections provide a solution to this issue that works for the MPC106, MPC107, and MPC8240 and that requires only a small amount of hardware and software support.

To locate any published errata or updates for this document, refer to the website at http://www.mot.com/PowerPC/.

1.1 Overview of the Solution

The method outlined in this application note uses one of the additional $\overline{\text{RCS}}$[1-3] lines provided by the MPC106, MPC107, and MPC8240 ($\overline{\text{RCS}}$[2-3] are only available on the MPC107). With a small amount of hardware, the system can recover access to the local ROM when booting from PCI by relocating the local boot ROM to a different chip select. The system, therefore, requires that the embedded controller have the following facilities available:

- Access to a PCI-hosted local ROM
- Hardware to re-route $\overline{\text{RCS0}}$ and one of $\overline{\text{RCS1}}$, $\overline{\text{RCS2}}$ or $\overline{\text{RCS3}}$

A PCI-based ROM is available on the Motorola Yellowknife-X4 and Sandpoint reference platforms. These systems can serve as a suitable base for PCI access since they use the Winbond W83C553, which translates PCI memory accesses to 0xFFF0_0100 into ROM accesses. If Yellowknife-X4 or Sandpoint is not used, a similar facility is needed on the target system.

1.2 Hardware Implementation

The hardware requirement is minimal; it can be implemented with a three-position jumper, or will fit in a PAL or the corner of an ASIC, or can be implemented with discrete gates. The logic is shown in , and consists of either a jumper or a few gates inserted between the $\overline{\text{RCS0}}$ signal and the chip select of the Flash, $\overline{\text{CS}}$. Whether logic, jumpers, or switches are used depends largely upon reliability and accessibility requirements of the designer. Any one may be used for the purposes of this application note.

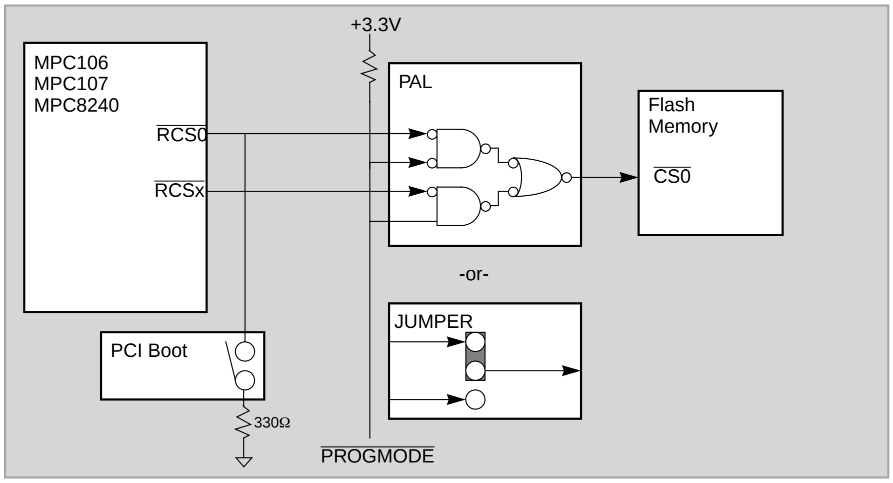

Figure 1. RCS Routing Logic

The following VHDL code is representative of code used in a PAL or ASIC implementation:

```
rcs0o_L <= ‘0’ WHEN ((rcs0i_L = '0' AND progmode_L = '1')
               OR   (rcs1i_L = '0' AND progmode_L = '0'))
               ELSE '1';

rcs1o_L <= '0’ WHEN (rcs1i_L = '0' AND progmode_L = '0')
               ELSE '1';
```

The essence of the logic is that in normal mode, when $\overline{\text{PROGMODE}}$ is high, the $\overline{\text{RCS0\_OUT}}$ and $\overline{\text{RCS1\_OUT}}$ pins follow their respective $\overline{\text{RCSx\_IN}}$ inputs and act as normal Flash, ROM, or I/O chip selects. In program mode, when $\overline{\text{PROGMODE}}$ is asserted low, $\overline{\text{RCS1\_OUT}}$ is deactivated and $\overline{\text{RCS0\_OUT}}$ is asserted whenever $\overline{\text{RCS1\_IN}}$ is asserted.

Note that $\overline{\text{RCS0}}$ will appear to be asserted while booting from PCI. The memory controller does not drive the line and the external pull-down will make it appear to be low. The logic must account for this, and the board must not fail to operate when a Flash device is present and always enabled (usually, when PCI boot is enabled $\overline{\text{RCS0}}$ is not used). The logic shown above handles this situation.

Note also that the logic is required even if $\overline{\text{RCS1}}$ is never used. Without $\overline{\text{PROGMODE}}$, the Flash would be permanently enabled when PCI boot is selected. Therefore, while the logic can be simplified, it cannot be reduced to simply an “OR” of the two signals ($\overline{\text{RCS0}}$ and $\overline{\text{RCS1}}$)—$\overline{\text{PROGMODE}}$ must be involved.

1.3 Local Program Software

In addition to the hardware, there are a few software issues to address. The first is to create a custom controller program for the PCI master, which transfers the desired program to the device to program. This image could be embedded in the PCI boot ROM, or transferred to PCI memory from disk, depending on the complexity of the master program.

An alternate approach is to have the same code run in both the PCI and local ROM spaces. The MPC106, MPC107, and MPC8240 do not treat the memory spaces any differently, so this requires no programming effort for the application. Instead, the code can be manually instructed to copy itself to local ROM or can automatically detect it is running on PCI and initiate the transfer automatically. The former approach is used by Motorola’s DINK debugger, which has an “fupdate” command to transfer itself from PCI to local Flash on PPMC cards which have local Flash attached.

The general programming steps are as follows:

- Set board to boot from PCI space with hardware jumper or switch. This is typically done with a switch enabling a pull-down resistor on $\overline{\text{RCS0}}$. This must be done in hardware.
- Enable “program mode” using a hardware jumper if needed.
- Apply power and reset.
- The board fetches instructions from PCI.
- Initialization software sets the PICR2[CF_FF0_LOCAL] bit to re-enable local access to the $\overline{\text{RCS1}}$ ROM space.
- Startup code either automatically enters program mode or waits for a command from the user.
- Software copies itself from PCI ROM (at 0xFF800000–0xFFFFFFFF or as size indicates) to local ROM (at 0xFF000000–0xFF7FFFFF) using normal Flash write algorithms. With the logic above writes to the $\overline{\text{RCS1}}$ space are redirected to the $\overline{\text{RCS0}}$ Flash ROM space.
- Remove program mode and PCI boot options.
- Apply reset. Board should boot from newly-programmed local Flash.

Note that the software routine must use the proper alignment of stores when writing to the $\overline{\text{RCS1}}$ space: 32- or 64-bits. The MPC8240 and MPC107 do not require that the write operation be the same size of the write, but the store address must be properly adjusted. The copy sequence is:

```
//!=====================================================================

//! Enter “unlock” sequence for Flash if needed, by doing dummy writes to
//! special addresses.

        lis         r3,0x0010       //! Set R3 = 1M
        mtctr       r3              //! store in counter register
        lis         r3,0xFFF0       //! R3 is now PCI Flash/ROM address
        lis         r4,0xFF00       //! R4 is now local aliases Flash address
loop:   lbz         r5,0(r3)

//! Do program enable sequence for Flash, if needed.

        stb         r5,0(r4)        //! Write new byte
        addi        r3,1            //! Next byte-aligned byte
        addi        r4,4            //! Next word-aligned byte (see text).
        bdnz        loop            //! Until 1M done
```

Many Flash devices, such as the AMD Am29LV800, require write sequences to dummy addresses before write operations can occur. This is not shown in the code above but is available in DINK V11.0.2, available at the Motorola website.

1.4 MPC107 Restrictions

The MPC107 implements the methods described in this application note exactly. In fact, it is somewhat easier since it has two additional chip selects, $\overline{\text{RCS2}}$ and $\overline{\text{RCS3}}$, which can be used in place of $\overline{\text{RCS1}}$. MPC107 users will probably prefer to use one of these chip-selects, since those spaces support Flash/ROMs larger than 1 Mbyte and can be programmed to be byte-accessible (this makes the software easier). The addressing is slightly different (0x7800_0000 and up), but is easily accommodated.

1.5 MPC8240 Restrictions

The MPC8240 does not have the additional $\overline{\text{RCS2}}$ and $\overline{\text{RCS3}}$ chip selects of the MPC107, but it otherwise it implements the methods described in this application note exactly. It is limited to programming 1 Mbyte without additional hardware since it only supplies 20 address bits when accessing the $\overline{\text{RCS1}}$ address space.

1.6 MPC106 Restrictions

The MPC106 implements the methods described in this application note with a serious limitation—it rejects writes to the $\overline{\text{RCS1}}$ space unless they are 64-bit single-beat writes. The only way to generate such a cycle is to use the floating-point unit to do a store. Since MPE60x processors do not have floating point, they cannot implement this method at all if the MPC106 is used.

For others, the only difference shows up in writing to the $\overline{\text{RCS1}}$ address (which is actually the boot ROM). Since the write must occur in the floating-point unit, the code changes to:

```
        align      8               //! itof must be aligned
itof:   bss        8               //! 8 byte to hold GPR->FPR transfer

        //! Enter “unlock” sequence for Flash if needed, by doing dummy writes to
        //! special addresses.

        lis        r3,0x0010       //! Set R3 = 1M
        mtctr      r3              //! store in counter register
        lis        r3,0xFFF0       //! R3 is now PCI Flash/ROM address
        lis        r4,0xFF00       //! R4 is now local aliases Flash address
        lis        r6,HI(itof)
        ori        r6,r6,LO(itof)  //! R6 points to “itof” buffer
loop:   lbz        r5,0(r3)

        //! Move data from GPR5 to FPR5
        stb        r5,0(r6)
        lfd        f5,0(r6)

        //! Do program enable sequence for Flash, if needed.

        stfd       f5,0(r4)        //! Write new byte
        addi       r3,1            //! Next byte-aligned byte
        addi       r4,4            //! Next word-aligned byte (see text).
        bdnz       loop            //! Until 1M done
```

Note that the writes which unlock and enable programming for the Flash device must be done with the 64-bit FPR registers as well.

1.7 Other Restrictions

As noted, due to the limited number of address lines supplied by the MPC106 and MPC8240, Flash ROMs larger than 1 Mbyte cannot be directly programmed. It may be possible for software to do bank selection to directly control the high-order address pin (SDMA0/SDBA1/AR0) which is not driven by the MPC106/ MPC8240 when writing to the $\overline{\text{RCS1}}$ space. This is beyond the scope of this application note but is relatively straightforward.

The $\overline{\text{RCS0}}$ space is often subdivided into spaces for ROM and I/O on embedded controllers. If so, this application note is still valid as long as the software can handle the fact that the I/O addresses change when in program mode. Since the change from local to PCI only occurs upon reset, software should be able to configure I/O addresses at that time.

1.8 Conclusion

With a little bit of software and hardware working together, it is possible to program a blank, unsocketed Flash soldered directly to an embedded controller or computer system.

Mfax is a trademark of Motorola, Inc.

The PowerPC name and the PowerPC logotype are trademarks of International Business Machines Corporation used by Motorola under license from International Business Machines Corporation.

Information in this document is provided solely to enable system and software implementers to use PowerPC microprocessors. There are no express or implied copyright licenses granted hereunder to design or fabricate PowerPC integrated circuits or integrated circuits based on the information in this document.

Motorola reserves the right to make changes without further notice to any products herein. Motorola makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Motorola assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters can and do vary in different applications. All operating parameters, including “Typicals” must be validated for each customer application by customer’s technical experts. Motorola does not convey any license under its patent rights nor the rights of others. Motorola products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the part.

Motorola and Ⓜ are registered trademarks of Motorola, Inc. Motorola, Inc. is an Equal Opportunity/Affirmative Action Employer.

Motorola Literature Distribution Centers:

USA/EUROPE: Motorola Literature Distribution; P.O. Box 5405; Denver, Colorado 80217; Tel.: 1-800-441-2447 or 1-303-675-2140;
World Wide Web Address: http://ldc.nmd.com/

JAPAN: Nippon Motorola Ltd SPD, Strategic Planning Office 4-32-1, Nishi-Gotanda Shinagawa-ku, Tokyo 141, Japan Tel.: 81-3-5487-8488

ASIA/PACIFIC: Motorola Semiconductors H.K. Ltd Silicon Harbour Centre 2, Dai King Street Tai Po Industrial Estate Tai Po, New Territories, Hong Kong

Mfax™: RMFAX0@email.sps.mot.com; TOUCHTONE 1-602-244-6609; US & Canada ONLY (800) 774-1848;
World Wide Web Address: http://sps.motorola.com/mfax

INTERNET: http://motorola.com/sps

Technical Information: Motorola Inc. SPS Customer Support Center 1-800-521-6274; electronic mail address: crc@wmkmail.sps.mot.com.

Document Comments: FAX (512) 895-2638, Attn: RISC Applications Engineering.

World Wide Web Addresses: http://www.motorola.com/PowerPC
http://www.motorola.com/netcomm
http://www.motorola.com/HPESD

MOTOROLA