This application note describes a design method for an embedded system that easily allows loading the contents of a (potentially blank) flash from the PCI bus. A secondary benefit is that when the switches are reset, different boot images can be selected.

This application note covers the following topics:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Section 1, “Overview”</td>
<td>2</td>
</tr>
<tr>
<td>Section 2, “Hardware Implementation”</td>
<td>2</td>
</tr>
<tr>
<td>Section 3, “Local Program Software”</td>
<td>3</td>
</tr>
<tr>
<td>Section 4, “MPC8240 Restrictions”</td>
<td>4</td>
</tr>
<tr>
<td>Section 5, “Restrictions on the Tsi106 Host Bridge”</td>
<td>4</td>
</tr>
<tr>
<td>Section 6, “Other Restrictions”</td>
<td>5</td>
</tr>
<tr>
<td>Section 7, “Conclusion”</td>
<td>5</td>
</tr>
<tr>
<td>Section 8, “Revision History”</td>
<td>6</td>
</tr>
</tbody>
</table>

The Freescale PowerPC™ integrated processor/host bridge devices, the MPC8240, MPC8241 and MPC8245, as well as the Tundra Tsi106™ and Tsi107™ PowerPC host bridge devices, can load start-up code from a flash device called the boot flash. This flash device is typically located on the local memory bus, but can be optionally redirected to the PCI bus. To enhance performance or system architecture, the boot flash should be on the local bus.

When an embedded system is manufactured, the boot flash is often completely blank and requires initialization from a master image. If a boot-sector flash (pre-loaded with appropriate software) is not appropriate or not available, the local memory bus flash can be programmed in place only with special tools such as in-circuit programmers or JTAG tools.

All of the PowerPC memory controller devices listed above share a common architecture, and the same restrictions that prevent the most obvious solutions from working affect them all. These restrictions include the following limitations:

- 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 that RCS0 controls.
• When the controller is configured to boot from local ROM, it loses access to the PCI boot ROM space that a PCI-to-ISA bridge usually provides.

The following sections provide a solution that works for the Tsi106 host bridge, Tsi107 host bridge, and MPC8240 and requires only a small amount of hardware and software support.

1 Overview

The method outlined in this application note uses one of the additional chip-select lines (RCS1, RCS2 and RCS3) that the MPC824X and the Tsi106 host bridge and Tsi107 host bridge provide. (RCS2 and RCS3 are available only on the MPC8245 and the Tsi107 host bridge.) 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 requires that the embedded controller has the following facilities:

• Access to a PCI-hosted local ROM
• Hardware to re-route RCS0 and one of RCS1, RCS2 or RCS3

A suitable PCI-based flash is available on the Freescale Sandpoint reference platform. This system can serve as a suitable basis for PCI access because it uses the Winbond W83C553, which translates PCI memory accesses to 0xFFF0_0100 into ROM accesses. If Sandpoint is not used, a similar facility is needed on the target system.

2 Hardware Implementation

The hardware requirement is minimal, and can be implemented with a three-position jumper. It can fit in a PAL or tiny fraction of an ASIC, or can be implemented with discrete gates. Two different implementation methods, which are shown in Figure 1 and Figure 2, consist of a few gates that are inserted between the RCS0 signal and the chip select (CS) of the flash or a simple three-position jumper.
Whether logic, jumpers, or switches are used depends largely upon reliability and the designer’s accessibility requirements. Any of these methods can serve for the purposes of this application note. The following VHDL code is representative of code that a PAL or ASIC implementation uses:

```vhdl
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 `PROGMODE` is high, the `RCS0_OUT` and `RCS1_OUT` pins follow their respective `RCSx_IN` inputs and act as normal flash, ROM, or I/O chip selects. In program mode, when `PROGMODE` is asserted low, `RCS1_OUT` is deactivated and `RCS0_OUT` is asserted whenever `RCS1_IN` is asserted.

Note that `RCS0` appears to be asserted while booting from PCI. The memory controller does not drive the line, and the external pull-down apparently makes it low. The logic must account for this situation, and the board must not fail to operate when a flash device is present and always enabled. (Usually, when PCI boot is enabled `RCS0` is not used.) The logic shown above handles this situation.

Note also that the logic is required even if `RCS1` is never used. Without `PROGMODE`, the flash would be permanently enabled when PCI boot is selected. Therefore, while the logic can be simplified, it cannot be reduced to an ‘OR’ of the two signals (`RCS0` and `RCS1`). `PROGMODE` must be involved.

### 3 Local Program Software

Hardware and a few software issues must be addressed. A custom controller program for the PCI master, which transfers the desired program to the device to program, must be created. 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. Because the MPC824X and Tsi106 and Tsi107 devices do not treat the memory spaces differently, no programming effort is required for the application. Instead, the code can be manually instructed to copy itself to local ROM or can automatically detect it running on PCI and initiate the transfer automatically. Motorola’s DINK debugger, which has an “fupdate” command to transfer itself from PCI to local flash on PPCMC cards which have local flash attached, uses the former approach.

The general programming steps are as follows:

1. Set board to boot from PCI space with hardware jumper or switch. Use a switch enabling a pull-down resistor on `RCS0`. This step must be done in hardware.
2. Enable 'program mode' using a hardware jumper if needed.
3. Apply power and reset.
4. The board fetches instructions from PCI.
5. Initialization software sets the PICR2[CF_FF0_LOCAL] bit to re-enable local access to the RCST flash space.
6. Startup code either automatically enters program mode or waits for a command from the user.
7. 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 RCST space are redirected to the RCS0 flash space.
8. Remove program mode and PCI boot options.
9. Apply reset. The board boots from newly-programmed local flash.

Note that the software routine must use the proper alignment of stores when writing to the RCST space: 32- or 64-bits. The MPC8241, MPC8245 and the Tsi107 host bridge do not require that the write operation is the same size of the write, but the store address must be properly adjusted. The following copy sequence is used:

```c
//=====================================================================
// Enter unlock sequence for flash if needed, by doing dummy writes to
// special addresses.
  lis r3,0x0010 // Set R3 = 1M
  mtcotr3 // 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
  addir3,1 // Next byte-aligned byte
  addir4,4 // 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 requirement is not shown in the code above, but is available in DINK V11.0.2 or later, which is available at the Freescale web site.

4 MPC8240 Restrictions

The MPC8240 does not have the additional RCS2 and RCS3 chip selects of the Tsi107 host bridge, but it exactly implements the methods that this application note describes. The MPC8240 is limited to programming 1 Mbyte without additional hardware because it supplies only 20 address bits when accessing the RCST address space.

5 Restrictions on the Tsi106 Host Bridge

The Tsi106 host bridge implements the methods that this application note describes, but has a serious limitation. The Tsi106 host bridge rejects writes to the RCST 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 Tsi106 host bridge is used.
For others, the only difference shows up in writing to the 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 r3,0x0FF0 // R3 is now PCI flash/ROM address
lis r4,0xFF00 // R4 is now local aliases flash address
lis r6,HI(itof)
or 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 that unlock and enable programming for the flash device must be done with the 64-bit FPR registers.

6 Other Restrictions

Because the limited number of address lines that the Tsi106 host bridge and MPC8240 supply, flash ROMs larger than one Mbyte cannot be directly programmed. Software might be able to do bank selection to control the high-order address pin (SDMA0/SDBA1/AR0) directly, which the Tsi106 host bridge and the MPC8240 do not drive when writing to the RCS1 space. This functionality is beyond the scope of this application note, but is relatively straightforward.

The 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. Because the change from local to PCI occurs only at reset, software should be able to configure I/O addresses then.

7 Conclusion

With some teamwork between the software and hardware, a blank, unsocketed flash program can be soldered directly to an embedded controller or computer system.
# 8 Revision History

Table 1 shows the revision history of this document.

<table>
<thead>
<tr>
<th>Revision Number</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Initial release</td>
</tr>
<tr>
<td>1</td>
<td>Updates and nontechnical reformatting</td>
</tr>
<tr>
<td>1.1</td>
<td>Nontechnical reformatting</td>
</tr>
<tr>
<td>1.2</td>
<td>Nontechnical reformatting</td>
</tr>
</tbody>
</table>
Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document. Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor 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 which may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including “Typicals” must be validated for each customer application by customer’s technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor 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 Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor 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 Freescale Semiconductor was negligent regarding the design or manufacture of the part.