*AN2489/D*
*Rev. 0.1, 6/2003*

# *Upgrading Software from MPC7451 to MPC7455 or MPC7457*

*Maurie Ommerman*
*CPD Applications*
*risc10@*
*email.mot.com*

This document describes the process of modifying source code, software changes, for upgrading from a MPC7450/MPC7451 to an MPC7455 or MPC7457, using the DINK32 source code as an example. The MPC7450, MPC7451, MPC7455, and MPC7457 are PowerPC™ microprocessors. The MPC7451 has the same functionality as the MPC7450 and any differences in data regarding bus timing, signal behavior, and AC, DC, and thermal characteristics are detailed in the hardware specifications. Note that because the MPC7450 and MPC7451 have the same functionality they are used interchangeably through-out the documentation and code.

The following topics are addressed:

# 1 Introduction

This document describes the changes required in software to upgrade from an MPC7450/MPC7451 to an MPC7455 or MPC7457. A general description of the changes is given followed by some code snippets from DINK32, see Section 4.1, "CPU Information," to illustrate the differences. Some familiarity with DINK32 may be required to understand all the code snippets.

- "Step 1: Add Processor Version Register (PVR) Definitions," shows the PVR setting.
- "Step 2: Set DINK CPU Information and the PLL Table," discusses the PLL settings.

- “Step 3: Set the L3 Cache Bits,” discusses the cache bit settings.
- “Step 4: Setup and Perform Software Flush,” discusses the settings that do not change from the MPC7450, MPC7455, to MPC7457, but may need to be detected.
- “Step 5: Block Address Translation Registers (BATs) for the MPC7455 and MPC7457,” discusses enabling the additional four BATs.
- “Step 6: Setting Special Purpose Registers (SPRs),” discusses the L3 cache input timing control register (L3ITCR0) setting.

## 1.1 Terminology

The following terms are used in this document:

- DINK32—A small operating system (OS) debugger for the Sandpoint evaluation board.The executable, user’s manual, and some source code are available on the Motorola web site. Search for “DINK32” on the Motorola home page:www.mot.com/semiconductors.
- BAT—Block Address Translation register, used to define large blocks of memory.
- DDR—Dual Data Rate SDRAM
- PLL—Phased-Lock Loop. A set of pins on the processor determine the divider to use for various frequency settings. The PMC has a set of switches that can manipulate the pins on the processor to allow various frequency settings to be configured at start up time.
- PMC—PCI Mezzanine Card
- RAM—Random Access Memory
- SPR—Special Purpose Register
- SDRAM—Synchronous Dynamic RAM
- SRAM—Static RAM

# 2 What are the Steps?

To upgrade support from the MPC7450 to the MPC7455 and then the MPC7457, DINK32 source code was changed as follows:

- Added Processor Version Register (PVR) definitions
- Defined the L3 cache settings
- Performed a software flush
- Encoded the phased-lock loop (PLL) settings
- Established the hardware implementation-dependent register (HID0) settings.

In addition the AltiVec settings and the L3 cache input timing control register (L3ITCR0), a special purpose register (SPR 984), are all discussed and the code is shown.

# 3 Step 1: Add Processor Version Register (PVR) Definitions

Each Motorola processor has a unique read only register known as the PVR. Software that needs to differentiate between the various processors must test for this PVR value. DINK32 is general purpose software that determines which processor the software is running on via the PVR. DINK32 has a definition for all the PVRs for the processors. DINK32 supports the PVR for the MPC7450, MPC7455, and MPC7457.

The PVR value for the:

- MPC7450 is 0x8000
- MPC7455 is 0x8001
- MPC7457 is 0x8002.

The MPC7457's PVR is defined in DINK32's config.h file and is defined as:

```
#define PVR_7457        -32766      // 0x8002
```

> The gcc compiler does not accept any 16 bit value where bit 0 is a 1 (0x8002 = 0b1000_0000_0000_0010). So, for any instruction that expects a 16 bit value, such as immediate instructions, a value of 0x8002 will get an error. However, gcc does accept the negative decimal equivalent, which in this case is -32766 (-32766 = 0x8002 = 0b1000_0000_0000_0010).

| Processor | PVR |
|---|---|
| MPC7450 | 0x8000 |
| MPC7455 | 0x8001 |
| MPC7457 | 0x8002 |

**Figure 1. PVR Designators**

# 4 Step 2: Set DINK CPU Information and the PLL Table

In order for the software to behave properly with each processor, DINK32 must understand the attributes of each device it supports. These attributes include, the name that is printed out on the splash screen, the PVR, cache size, BAT attributes, floating point availability, AltiVec availability, and others. In addition, DINK32 must understand the hardware characteristics of each processor as described in the hardware specifications for that device. DINK32 must know the core frequency for each processor that it supports. It uses an algorithm that computes the core frequency from the known bus frequency and the PLL table information.The following section provides details on how DINK32 sets the attributes for each specific processor.

## 4.1 CPU Information

DINK32 uses a cpuinfo table to define the various attributes of each device. This table existed before the MPC745x, and is used to define all known processors. The entry for the MPC7455 and MPC7457 is new in this table, the MPC7450/MPC7451 is shown for reference. The complete table of all the processors is not shown here, only the portions relating to the MPC745x.

Table 1 shows the differences between the MPC7451, MPC7455, and the MPC7457 microprocessors.

**Table 1. MPC7451, MPC7455, and MPC7457CPU Information**

| Microprocessor Information | MPC7450/MPC7451 | MPC7455 | MPC7457 |
|---|---|---|---|
| PVR | 0x8000 | 0x8001 | 0x8002 |
| Revision | 0x0000 | | |
| Name | 7450 | 7455 | 7457 |
| Generation | 4+ | | |
| Exception type | PPCVGER | PPCAPOLLO | PPCAPOLLO |
| L1I_size | 32 | | |
| L1D_size | 32 | | |
| L2_size | 256 | | 512 |
| L2_extsize (external size) | 0 | | |
| L3_size | 0 | | |
| L3_extsize (external size) | 2048 | | |
| Has 8 BATs [1] | 0 | 1 | |
| number of Bats | 4 | 8 | |
| number TLBs | 128 | | |
| Floating point avail [1] | 1 | | |
| Altivec (classic) avail [1] | 1 | | |
| Altivec (e500) avail [1] | 0 | | |
| PCI avail [1] | 0 | | |
| RapidIO available [1] | 0 | | |
| PCI interface available [1] | 0 | | |
| Ethernet interface available [1] | 0 | | |

**Note:** This table is defined in dink.h as shown in Section A.1, “CPUINFO.”

[1] A value 0 in the table indicates “no” and the value 1 indicates “yes.”

## 4.2 PLL Table Settings

DINK32 uses an algorithm to compute the core frequency from the processor’s bus frequency and the PLL table information.The PMC dip switches determine the PLLBITS that can be read by software to give a value from 0x00 to 0x1F, that is the index into this PLLTable. Thus PLLTable[PLLBITS] returns the multiplier for determining the core frequency from the bus frequency.

For PLL encoding there is a table of 16 or 32 shorts for each PLL encoding in existence (most are relatively common). All entries are fixed-point, scaled by 10 to avoid floating point operations; thus, 11.5 is represented as ‘115.’ Table 2 shows the PLL settings for the MPC7451, MPC7455, and MPC7457.

Table 2. MPC745x PLL Settings

<table>
<tr><th>Multiplier for the Core’s Frequency in the MPC7450/MPC7451 and MPC7455 V3.2 and Earlier</th><th>Multiplier for the Core’s Frequency in the MPC7455 V3.3 and Later and the MPC7457</th><th>PMC Dip Switch Setting (Index into the PLLTable)</th><th>Summary of Changes to the Multiplier Value in the PLLTable from the MPC7450/MPC7451 to the MPC7455/MPC7457</th></tr>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td>5</td><td>115</td><td>0x00</td><td>Multiplier value changes</td></tr>
<tr><td>0</td><td>170</td><td>0x01</td><td>New multiplier value added</td></tr>
<tr><td colspan="2">75</td><td>0x02</td><td>Same multiplier value</td></tr>
<tr><td colspan="2">150</td><td>0x03</td><td>Same multiplier value</td></tr>
<tr><td colspan="2">70</td><td>0c04</td><td>Same multiplier value</td></tr>
<tr><td>0</td><td>180</td><td>0x05</td><td>New Multiplier Value Added</td></tr>
<tr><td colspan="2">10</td><td>0x06 bypass</td><td>Same multiplier value</td></tr>
<tr><td>0</td><td>200</td><td>0x07</td><td>New multiplier value added</td></tr>
<tr><td colspan="2">20</td><td>0x08</td><td>Same multiplier value</td></tr>
<tr><td>0</td><td>210</td><td>0x09</td><td>New multiplier value added</td></tr>
<tr><td colspan="2">65</td><td>0c0a</td><td>Same multiplier value</td></tr>
<tr><td colspan="2">130</td><td>0x0b</td><td>Same multiplier value</td></tr>
<tr><td>25</td><td>85</td><td>0x0c</td><td>Multiplier value changes</td></tr>
<tr><td>0</td><td>240</td><td>0x0d</td><td>New multiplier value added</td></tr>
<tr><td>0</td><td>95</td><td>0x0e</td><td>New multiplier value added</td></tr>
<tr><td colspan="2">90</td><td>0x0f</td><td>Same multiplier value</td></tr>
<tr><td colspan="2">30</td><td>0x10</td><td>Same multiplier value</td></tr>
<tr><td>0</td><td>105</td><td>0x11</td><td>New multiplier value added</td></tr>
<tr><td colspan="2">55</td><td>0x12</td><td>Same multiplier value</td></tr>
<tr><td colspan="2">110</td><td>0x13</td><td>Same multiplier value</td></tr>
<tr><td colspan="2">40</td><td>0x14</td><td>Same multiplier value</td></tr>
<tr><td colspan="2">100</td><td>0x15</td><td>Same multiplier value</td></tr>
<tr><td colspan="2">50</td><td>0X16</td><td>Same multiplier value</td></tr>
<tr><td colspan="2">120</td><td>0X17</td><td>Same multiplier value</td></tr>
<tr><td colspan="2">80</td><td>0X18</td><td>Same multiplier value</td></tr>
<tr><td colspan="2">140</td><td>0x19</td><td>Same multiplier value</td></tr>
<tr><td colspan="2">60</td><td>0x1a</td><td>Same multiplier value</td></tr>
<tr><td colspan="2">160</td><td>0x1b</td><td>Same multiplier value</td></tr>
</table>

Table 2. MPC745x PLL Settings (continued)

| Multiplier for the Core’s Frequency in the MPC7450/MPC7451 and MPC7455 V3.2 and Earlier | Multiplier for the Core’s Frequency in the MPC7455 V3.3 and Later and the MPC7457 | PMC Dip Switch Setting (Index into the PLLTable) | Summary of Changes to the Multiplier Value in the PLLTable from the MPC7450/MPC7451 to the MPC7455/MPC7457 |
|---|---|---|---|
| | | | |
| | | | |
| 35 | 135 | 0x1c | Multiplier value changes |
| 0 | 280 | 0x1d | New multiplier value added |
| 0 | | 0x1e off | Same multiplier value |
| 0 | 125 | 0x1f | New multiplier value added |

These settings are defined in pmc.c and are shown in Section A.2, “PLL Table Settings.”

# 5 Step 3: Set the L3 Cache Bits

Each processor has a special purpose register that is used to specify cache parameters. These registers are specific to the processor and the specific memory used for off chip cache memory. In addition to cache memory, the off chip cache can be set up as private memory. The MPC7455 introduced two new bits in two existing registers, the existing registers are the L3CR (SPR 1018) and the L2CR (SPR 1017). The new bits are the L3CR[L3OH1] output hold bit and the L2CR[L3OH0] output hold bit, these bits are specific to the MPC7455 and are reserved in all other processors. The MPC7450/MPC7451 introduced the L3ITCR0 (SPR 984) special purpose register that allows one to set the AC timings. The code and an explanation on setting the SPR is shown in “Step 6: Setting Special Purpose Registers (SPRs).” The MPC7457 introduced the L3ITCR1, L3ITCR2, and LCITCR3 registers, however, these settings are not changed in DINK.

L3CR[L3OH1] and the L2CR[L3OH0] bits apply only to the MPC7455. The MPC7457 has an entire register, L3OHCR (SPR 1000) to control output AC timing. For processors other than the MPC7455, the L3CR[L3OH1] and the L2CR[L3OH0] bits are reserved and should not change.

| Register | MPC7450/MPC7451 | MPC7455 | MPC7457 |
|---|---|---|---|
| L3CR[12] | Not implemented | New L3CR[L3OH1] bit | Not implemented |
| L2CR[12] | Not implemented | New L2CR[L3OH0] bit | Not implemented |
| L3OHCR[0-31] | Not implemented | | New L3OHCR register |
| L3ITCR0[0-31] | Changes | | Does not change |
| L3ITCR1[0-31] | Does not change | | |
| L3ITCR2[0-31] | Does not change | | |
| L3ITCR3[0-31] | Does not change | | |

Figure 2. New SPRs and Bits for the MPC745x

All memory specific SPR settings need to be determined before the registers can be set. So the first thing the user needs to do is look at the specifications for the memory and then determine the settings.

For example in our board, DINK32 uses the mask 0x73EF_F3C5 that is specific to our board to control which bits are allowed to be set in the L3CR (SPR 1018), and the comments in the code indicate each particular bit setting that is set in DINK32. The function in DINK32 does all this in the cache_inval_enable_L3 function that is shown below.

Code snippets illustrating the specific code for the MPC7450/MPC7451, MPC7455, and MPC7457 from the cache_inval_enable_L3 function are shown here. The entire function is shown in Appendix A, "Complete Code Examples," that includes specific code for dealing with the MPC7457 Chip Errata 20 and 21. Note that the code text in italics is new for the MPC7457.

```
//-------------------------------------------------------------------------
//  cache_inval_enable_L3 - Invalidate and enable the L3
        .global cache_inval_enable_L3
cache_inval_enable_L3:

        mfspr   r9, 287                 // read PVR
        rlwinm  r9,r9,16,16,31  // shift down 16 bits
        cmpli   0,0,r9,0x8000   // Vger pvr
        beq     cache_inval_enable_L3_MPC7450
        cmpli   0,0,r9,0x8001   // Apollo pvr
        beq     cache_inval_enable_L3_MPC7450
        cmpli   0,0,r9,0x8002   // Apollo7 pvr MPC7457
        beq     cache_inval_enable_L3_MPC7450
        xor     r3,r3,r3// All others: There is no L3 to enable.
        b       cache_inval_enable_L3_exit// just return zero

........

#endif

//      Do not allow user to set L3CR[L3E/0]; we'll do that below.

        lis     r4,0x73ef

                //  L3E/0           = 0   (don't allow enabling yet)
                //  L3PE/1          = x   (parity)
                //  L3APE/2         = x   (address parity)
                //  L3SIZ/3         = x   (0 - 1MB; 1 - 2MB)
                //  L3CLKEN/4       = 0   (L3 clock enable -
                //                         see MPC7450_errata20)
                //  L3CR/5          = 0   (Must be set by software -
                //                         named L3DX?)
                //  L3CLK[6:8]      = xxx (L3 divider)
                //  IONLY/9          = x    (Instructions only)
                //  L3CR[10]         = x    (Extended L3CLK on MPC7457)
                //  L3CR[11]         = 0    (reserved)MPC7457 L3CKSPEXT
                //  L3CR[12]         = x    (L3OH1)MPC7455 only
                //  L3SPO/13         = x    (sample point overide)
                //  L3CKSP[14:15]    = xx   (clock sample point)

cmpli   0,0,r9,0x8002        // Apollo7 pvr
        beq     MPC7457_L3CR_mask    //need different mask for non-7457 parts
        ori     r4,r4,0xf3c5         //bit 30 is reserved for non-7457
        b       no_MPC7457_L3CR_mask
MPC7457_L3CR_mask:
        ori     r4,r4,0xf3c7
no_MPC7457_L3CR_mask:

                //  L3PSP[16:18]     = xxx (processor sample point)
                //  L3REP/19         = x    (Replacement algorithm - 0 default)
                //  L3HWF/20         = 0    (L3 Hardware Flush - not here)
                //  L3I/21           = 0    (global invalidate - not set here)
                //  L3RT[22:23]      = xx   (SRAM type - MSUG2 DDR SRAMs)
                //  L3CYA/24         = x    (L3 SRAM clock control bit)
                //  DONLY/25         = x    (data only)
                //  L3CR[26:28]      = 000 (reserved)
                //  PMEN/29          = x    (Private memory enable)
                //  L3CR/30          = 0    (reserved on non-MPC7457)
                //  PMSIZ/30         = x    (L3PM 4MB on MPC7457)
                //  PMSIZ/31         = x    (0 - 1MB; 1 - 2MB)
        and     r4,r3,r4// Mask off non-user bits, preserve r3
        oris    r4,r4,0x0400// Enforce L3DX/5 in L3CR pattern (not optional)

//      dssall         // Cancel any pending data stream operations with dssall
        .long 0x7e00066c // Not all assemblers generate
                                // the dssall instruction yet
        sync             // Finish any pending store ops in LSU (UM p3-65)

.........

// Look at SRAM type; if PB2, do not play with L3ITCR0 register below.
```

See “Step 6: Setting Special Purpose Registers (SPRs)” for the L3ITCR0 description and code.

# 6 Modify Code to Test for the MPC7457 Case

All these steps are the same for the MPC7451, MPC7455, and MPC7457. DINK32 has separate tests for each processor, so if your code has detection software for various processors, you will also need to add processor tests (detection) for the MPC7457. If your code is limited to only a single MPC745x processor, it is not necessary to add these tests.

## 6.1 Step 4: Setup and Perform Software Flush

The MPC7451, MPC7455, and MPC7457 do not support an L1 hardware flush, thus a software flush algorithm must be used. This has not changed from the MPC7450/MPC7451 to the MPC7455/MPC7457. However, it is necessary to detect the MPC7457 and perform a software flush. Text in italics is new code for the MPC7457.

The following code fragment from cache.S shows how DINK32 determines the type of processor flash and performs the correct flushing mechanism.

```
// Which part are we dealing with?
        mfspr   r9,287          //Only use upper half of PVR
//Be careful! srawi r9,r9,16 will sign extend the MPC7450 and MPC7455
        rlwinm  r9,r9,16,16,31
//Be careful! cmpi r9,0x8nnn will sign extend and not be equal on 7450/7455
        cmpli   0,0,r9,0x0008   //Is this MPC750/MPC755?
        beq     HW_flush_750_DCache //Yes? Use HW flush assist.
        cmpli   0,0,r9,0x000c   //Is this MPC7400?
        beq     HW_flush_7400_DCache    //Do hardware assisted flush
        cmpli   0,0,r9,0x800C   //Is this MPC7410?
        beq             HW_flush_7400_DCache    //Do hardware assisted flush
        cmpli   0,0,r9,0x8000   //Is this MPC7450?
        beq             SW_flush_7450_DCache    //Do software flush
        cmpli   0,0,r9,0x8001   //Is this MPC7455?
        beq             SW_flush_7450_DCache    //Do software flush
cmpli   0,0,r9,0x8002   //Is this MPC7457?
        beq             SW_flush_7450_DCache    //Do software flush
```

## 6.2 Setting the HID0 Register Appropriately

The HID0 (SPR 1008) register defines the L1 cache characteristics, time base, branch table, and many other characteristics of each processor. The *MPC7450 RISC Microprocessor Family User's Manual* describes these bits in detail. In all processors other than the MPC745x, HID0[0] was used to enable the MCP and this feature has since been moved to HID1[0]. In the MPC745x, HID0[0] is reserved and is hard coded to 1, and the user must not change this bit. Simply, the *MPC7450 RISC Microprocessor Family User's Manual* indicates that bits HID0[0–4] = '0b1000_0'. This was true for the MPC7451. To support the MPC7457 in DINK32 we need to add a similar test to maintain this bit in the same manner as for the MPC7451.

HID0 is set up initially in except2.S, however, when the L1 cache is enabled and/or disabled, the cache routines need to turn on/off the bits associated with L1 cache, HID0[16-21]. In addition, the code must ensure that HID0[0] is not changed.

The MPC7451 preferred bit setting for HID0 is 0x0410C0BC.

The MPC7455 and MPC7457 HID0 preferred bit setting is 0x0410c1bc and is defined in DINK32, config.h as:

#define DEFAULT_HID0_7457 0x0410C1BC

As shown in the table below the only bit that is different is HID0[23], the XBSEN bit, which is the Extended BAT Block Size Enable allowing BAT block sizes up to 4 Gigabytes.

| Processor | HID0 bit settings |
|---|---|
| MPC7451 | 0x0410C0BC |
| MPC7455 | 0x0410C1BC |
| MPC7457 | 0x0410C1BC |

**Figure 3. HID0 bit settings**

This code snippet in cache.S performs a test for the MPC7451/MPC7455/MPC7457 and does not set the bit HID0[0]. The complete code is in Appendix A, “Complete Code Examples”. Text in italics in the code is new for the MPC7457.

```
.global cache_inval_enable_L1
cache_inval_enable_L1:
        mfspr   r8,1008                 // Get current HID0=1008
// If this is MPC7450 ignore HID0/0 in following comparison.  It will always
// read as a one.
        mfspr   r9,287                  //Only use upper half of PVR
        rlwinm  r9,r9,16,16,31  //Shift down 16 bits
        cmpli   0,0,r9,0x8000   // MPC7450 pvr
        beq             ignore_HID0_0
        cmpli   0,0,r9,0x8001   // MPC7455 pvr
        beq             ignore_HID0_0
        cmpli   0,0,r9,0x8002   // MPC7457 pvr
        beq             ignore_HID0_0
        b               compare_desired_w_current
ignore_HID0_0:
        rlwinm  r8,r8,0,1,31
compare_desired_w_current:
```

The code then continues setting up HID0.

## 6.3 Various Settings at Initialization

Initialization settings have not changed from the MPC7450/MPC7451 to the MPC7455 and MPC7457. However, since each processor is detected individually, it is necessary to add the code to detect the MPC7457 and perform the same functions as for the other MPC745x processors.

During initialization DINK32 will determine the processor from the PVR and then perform initialization of specific SPRs, memory, and other attributes of the processor and the board environments. Most of this initialization is done in except2.S. The specific settings for the MPC7455/MPC7457 are set in except2.S as shown below.

> Enable Altivec, set MPC7450/MPC7451 modes, initialize HID0 with the default from config.h, initialize L2, set MPC7450/MPC7451 Java mode, and set the BATs for the MPC7455. All these actions were true for the MPC7451 as well, we just added code to ensure the same actions for the MPC7455 and MPC7457. The code snippets below show the additional code detecting the MPC7457 (PVR 0x8002).

**NOTE**

> In most cases, the code will compare this processor's PVR to known PVRs and perform actions associated with the correct processor. This code compares the PVR, then branch on equal to the function to perform the activity. However, the added tests for MPC7457 will branch not equal around the same MPC7451/55 functions. Thus, in effect, performing the same MPC7451/55 functions, that is **bne** versus **beq**.

Only the code snippet for enabling AltiVec is shown here. All the other code is similar and the complete code is in Appendix A, "Complete Code Examples."

1. Enable Altivec

```
// For MPC7xxx parts, enable AltiVec so we can save/restore the
// contents.
cmpli   0,0,r20,0x000C          // MPC7400
        beq     enable_altivec
        cmpli   0,0,r20,0x8000          // MPC7450
        beq     enable_altivec
        cmpli   0,0,r20,0x8001          // MPC745x
        beq     enable_altivec
        cmpli   0,0,r20,0x8002          // MPC7457
        beq     enable_altivec
        cmpli   0,0,r20,0x800C          // MPC7410
        bne     no_enable_altivec
```

2. Do set v'ger modes
3. Initialize HID0 with the default from config.h
4. Do initialize L2 (This is not required, the L2 can be initialized at any point in the code)
5. Do set v'ger Java mode

# 7 Step 5: Block Address Translation Registers (BATs) for the MPC7455 and MPC7457

The MPC7451 has four pairs of IBATs and DBATs. The MPC7455 and MPC7457 has eight pairs of IBATs and DBATs. The code snippets below shows a method to set the four extra pairs of IBATs and DBATs.

## 7.1 Set the BATs for the MPC7455 and MPC7457

This code snippet shows the additional 4 IBATs and DBATs. Code in italics is specific to the MPC7457

```
ldb_standard:
        cmplwi  r20,0x8001              // MPC7455
        beq     ldb_extended
        cmplwi  r20,0x8002              // MPC7457
        beq     ldb_extended
        lis     r5,BATS_standard@h      // Standard BATs
        ori     r5,r5,BATS_standard@l
        b       load_bat_array
ldb_extended:
        lis     r5,BATS_apollo@h        // Extended BATs
        ori     r5,r5,BATS_apollo@l
        b       load_bat_array

// Extended BATS (Apollo)

BATS_apollo:

        // Flash space + EUMBBAR

        .long   b_BEPI( 0xFC000000 ) + b_64M    + b_VS + b_VP   // IBAT0
        .long   b_BPRN( 0xFC000000 ) + b_WIMG(0,0,0,0) + b_RW
        .long   b_BEPI( 0xFC000000 ) + b_64M    + b_VS + b_VP   // DBAT0
        .long   b_BPRN( 0xFC000000 ) + b_WIMG(0,1,0,0) + b_RW

        // SDRAM space

        .long   b_BEPI( 0x00000000 ) + b_1G     + b_VS + b_VP   // IBAT1
        .long   b_BPRN( 0x00000000 ) + b_WIMG(0,0,0,0) + b_RW
        .long   b_BEPI( 0x00000000 ) + b_1G     + b_VS + b_VP   // DBAT1
        .long   b_BPRN( 0x00000000 ) + b_WIMG(0,0,0,0) + b_RW

        // PCI memory space

        .long   b_BEPI( 0x80000000 ) + b_512M   + b_VS + b_VP   // IBAT2
        .long   b_BPRN( 0x80000000 ) + b_WIMG(0,1,0,0) + b_RW
        .long   b_BEPI( 0x80000000 ) + b_512M   + b_VS + b_VP   // DBAT2
        .long   b_BPRN( 0x80000000 ) + b_WIMG(0,1,0,1) + b_RW

        .long   b_BEPI( 0x78000000 ) + b_128M   + b_VS + b_VP   // IBAT3
        .long   b_BPRN( 0x78000000 ) + b_WIMG(0,0,0,0) + b_RW
        .long   b_BEPI( 0x78000000 ) + b_128M   + b_VS + b_VP   // DBAT3
        .long   b_BPRN( 0x78000000 ) + b_WIMG(0,1,0,0) + b_RW
```

# 8 Step 6: Setting Special Purpose Registers (SPRs)

The following section provides a general discussion on how SPRs are set. For further details on setting SPRs see the *MPC7450 RISC Microprocessor Family User's Manual*.

## 8.1 SPRs

All the SPRs specific to the MPC7451, MPC7455, and MPC7457 must be saved and restored. Code performing the save and restore must be aware of the additional SPRs introduced by these processors.

## 8.2 Setting the SPRs

As discussed previously, in order to use the L3 cache, the special purpose register L3ITCR0 (SPR 984) must receive special attention. The MPC7451, MPC7455, and MPC7457 determines its own settings for the L3ITCR0 during hardware startup. It is recommended that the default settings for the L3ITCR0 are used for the MPC7457.

For the MPC7455, the setting can be adjusted as shown below to increase system margin. The MPC7455 L3ITCR0 register causes the L3 input clock phases to be shifted in a way similar, but not identical, to the way L2CR[L3OH0] and L3CR[(L3OH1]) registers work for the output timings on the MPC7455. The operation of this register is rather complex and improper settings can cause erratic behavior on the L3 and is only applicable for DDR.

Dink adjusts the setting for MPC7451 and MPC7455 class parts in cache.S by manipulating the L3ITCR0 register, as illustrated below.

**NOTE:**

> This information for the L3ITCR0 special purpose registers is only applicable to the MPC7455. Do not make these changes for the MPC7457.

The L3ITCR0 value (SPR 984) adds/subtracts some extra taps to the clock DLL beyond what it determined via its own feedback mechanism. In actuality, each sequential “1” bit in bits 0-22 indicates an internal delay to align the clock edge to the center of the data. The algorithm implements via a bit setting specific number of taps. According to the design team this table shows the association of settings to frequency.

**Table 3. Frequency versus Adjustment Value for the MPC7451 and MPC7457**

| Frequency | Additional Taps |
|---|---|
| 175 | +4 |
| 200 | +3 |
| 250 | +1 |

These values hold for both the MPC7451 and the MPC7457.

The frequency values shown are the SRAM frequencies (for example, 175 = 350Mt/s DDR). Mt/s is mega transfers per second.

One can extrapolate the above table for frequencies not shown.

**NOTE**

> Always read the register L3ITCR0 first, then shift bits into it, shift in 1s if shifting right, 0s if shifting left. DINK32 uses the **rlwinm** instruction for this. Also, one must set L3ITCR0[24] to force the processor to use the shifted value. See the code example below.

The code in DINK32 adds an extra +4 to the tap. To experiment with other adjustments modify this code.

```
// Look at SRAM type; if PB2, do not play with L3ITCR0 register below.

        li      r5,0x0300               // 3 = PB2
        and     r5,r3,r5                // Mask off non-user bits
        cmpi    0,0,r5,0x0300
        beq     skip_L3ITCR0
        cmpli   0,0,r9,0x8002           // Apollo7 pvr
        beq     skip_L3ITCR0            // Leave default L3ITCR0 setting for
                                        // MPC7457

// The MPC7457 designers recommend the following adjustments to L3ITCR0 for the
7450 & 7455, no not change this register for the MPC7457.

mfspr   r5,984          // L3ITCR0: adjust SRAM Clock edges by shifting
        rlwinm  r5,r5,28,4,31   // for data being read from bus by adding
        oris    r5,r5,0xf000    // four additional taps on a pulse shaper
        ori     r5,r5,0x00ff
        xori    r5,r5,0x007f // these two instructions set bit 24 on
        mtspr   984,r5
        isync

skip_L3ITCR0:
```

## 8.3 Processor designator

We assigned a specific designator to all our processors and the code must check for the MPC7457 designator. These are arbitrary since we chose these values. This value is stored in the global variable process_type, and each value is #defined in config.h. Once the processor is determined via its PVR, the process_type is set and used for further comparisons for each processor.

These designators are used in the code snippets below.

**Table 4. Processor Designators**

| Processor | config.h name | process_type value |
|---|---|---|
| MPC7451 | PPCVGER | 38 |
| MPC7455 | PPCAPOLLO | 3b |
| MPC7457 | PPCAPOLLO7 | 3d |

## 8.4 Code Changes

The following section summarizes code changes that were made to DINK32. All the following code snippets are from reg_swap.S.

### 8.4.1 Test for MPC7457

The code tests specifically for MPC7451 and MPC7455, thus new tests for the MPC7457 must be built and those SPRs specific to the MPC7457 must be saved and restored. Full code, including testing for the

MPC7451 and MPC7455, is shown in Appendix A, “Complete Code Examples. The following is a code snippet of how the code changes for the MPC7457.

Save MPC7457 specific SPRs.

```
//  check for the MPC7457
        lis     r4,process_type@h
        ori     r4,r4,process_type@l
        lbz     r4,0(r4)
        cmpi    0,0,r4,PPCAPOLLO7  // PPCAPOLLO7 MPC7457 from config.h
        bne     cr0,noMPC7457
// save MPC7457 special purpose registers.
        mfspr   r4,l3ohcr                               // L3OHCR= 1000
        stw     r4,L3OHCR_LOC*spr_inc(r3)
        mfspr   r4,l3itcr0                              // L3ITCR0= 984
        stw     r4,L3ITCR0_LOC*spr_inc(r3)
        mfspr   r4,l3itcr1                              // L3ITCR1= 1001
        stw     r4,L3ITCR1_LOC*spr_inc(r3)
        mfspr   r4,l3itcr2                              // L3ITCR2= 1002
        stw     r4,L3ITCR2_LOC*spr_inc(r3)
        mfspr   r4,l3itcr3                              // L3ITCR3= 1003
        stw     r4,L3ITCR3_LOC*spr_inc(r3)

noMPC7457:
.....

Restore MPC7457 specific SPRs
//  check for the MPC7457
        lis     r4,process_type@h
        ori     r4,r4,process_type@l
        lbz     r4,0(r4)
        cmpi    0,0,r4,PPCAPOLLO7  // PPCAPOLLO7 MPC7457 from config.h
        bne     cr0,noStoreMPC7457
// save MPC7457 special purpose registers.
        lwz     r4,L3OHCR_LOC*spr_inc(r3)
        mtspr   l3ohcr,r4                               // L3OHCR= 1000
        lwz     r4,L3ITCR0_LOC*spr_inc(r3)
        mtspr   l3itcr0,r4                              // L3ITCR0= 984
        lwz     r4,L3ITCR1_LOC*spr_inc(r3)
        mtspr   l3itcr1,r4                              // L3ITCR1= 1001
        lwz     r4,L3ITCR2_LOC*spr_inc(r3)
        mtspr   l3itcr2,r4                              // L3ITCR2= 1002
        lwz     r4,L3ITCR3_LOC*spr_inc(r3)
        mtspr   l3itcr3,r4                              // L3ITCR3= 1003
noStoreMPC7457:
```

### 8.4.2 Checking for Processor Specific Designator

This code determines via the special designator if we have a MPC7451/MPC7455/MPC7457 mode processor and, if not, then we ignore all the settings. It is required for saving registers and restoring registers.

```
lis     r4,process_type@h
        ori     r4,r4,process_type@l
        lbz     r4,0(r4)
... other code
set_vger:

        cmpi    0,0,r6,0x38
        beq     set_vger_apollo
        cmpi    0,0,r6,0x3B
        beq     set_vger_apollo
        cmpi    0,0,r6,0x3D
        bne     end_setting
......
set_vger_regs:
        cmpi    0,0,r6,0x38
        beq     set_vger_apollo_regs
        cmpi    0,0,r6,0x3B
        beq     set_vger_apollo
        cmpi    0,0,r6,0x3D
        bne     end_set_regs
```

# 9 Conclusion

Armed with this information, you can either modify DINK32 or your own code to accommodate for the MPC7455 or the MPC7457 processor.

There are several pieces of code that need to be added to MPC7450/MPC7451 and MPC7455 aware code to also manage the MPC7457. One needs to add code to understand the MPC7457 PVR (0x8002), add the PLL setting values and CPU information, set the L3 cache bits, set the HID0[XBSEN/23] bit if desired (it is not required), recognize that HID0[0] bit is reserved, enable various features of the MPC7457, and set bits in the L2CR, L3CR, and L3ITCR0.

# 10 Revision History

Table 5 lists the revisions to this document.

Table 5. Revision History

| Revision Number | Changes |
|---|---|
| 0.0 | Initial release |
| 0.1 | Nontechnical reformatting |

# Appendix A Complete Code Examples

## A.1 CPUINFO

```
//-------------------------------------------------------------------------
//      CPUInfo -- a database of CPU information
//-------------------------------------------------------------------------
typedef struct st_cpuinfo {
        USHORT  pvr;                            // Upper PVR
        USHORT  rev_min;                        // REV part of PVR must be >=
this to qualify.
    char        Name[ 10 ];                     // Official name
    char        NickName[ 12 ];         // Nickname
        char    Gen[4];                         // Generation.
        char    type;                           // exception handler type
        USHORT  L1I_size;                       // in K.
        USHORT  L1D_size;                       // in K.
        USHORT  L2_size;                        // in K.
        USHORT  L2_extsize;                     // in K.
        USHORT  L3_size;                        // in K.
        USHORT  L3_extsize;                     // in K.
    char        HasXBSEN;                       // Has larger BAT size support
        USHORT  no_BATs;                        // I+D
        USHORT  no_TLBs;                        // # of TLBs
    char        HasFP;                          // Has floating point
    char        HasAV;                          // Has AltiVec (classic)
    char        HasAVe;                         // Has AltiVec (e500)
    char        PCI;                            // Has PCI interface
    char        HasRapidIO;                     // Has RapidIO interface
    char        HasPCI;                         // Has PCI interface
    char        HasENet;                        // Has Ethernet interface(s)
} CPUINFO;
```

6. The setting for the 7450/55/57 is initilaized in main.c as:

```
//-------------------------------------------------------------------------
//      cdb -- various collected data on this CPU.
//-------------------------------------------------------------------------

CPUINFO cdb[] = {
// PVR    REV     NAME      NICKNAME     GEN     CLASS
  // L1I  L1D  L2    L2X  L3    L3X    XBSEN
  // #BATS  #TLBS FP AV AVe PCI RIO ENET

// MPC7451
```

```
  { 0x8000, 0x0000, "7450",   "V'ger",     "4+",  PPCVGER,
    32,  32,  256,    0, 0,  2048, 0,
    4,    128,   1, 1, 0,  0,  0,  0 },

  // MPC7455
  { 0x8001, 0x0000, "7455",   "Apollo",    "4+",  PPCAPOLLO,
    32,  32,  256,    0, 0,  2048, 1,
    8,    128,   1, 1, 0,  0,  0,  0 },

  // MPC7457
  { 0x8002, 0x0000, "7457",   "Apollo7",   "4+",  PPCAPOLLO,
    32,  32,  512,    0, 0,  2048, 1,
    8,    128,   1, 1, 0,  0,  0,  0 },
```

## A.2 PLL Table Settings

The M7455_V3_PLLTable is used for the MPC7457 as well.

```
//---------------------------------------------------------------------------
// PLL Encoding. There is a table of 16 or 32 shorts for each pll
// encoding in existence (most are relatively common).
//
// All entries are fixed-point, scaled by 10 to avoid floating
// point operations; thus, 6.5X is represented as "65".
// M745X_PLLTable: For MPC7451, MPC7451, MPC7455 <= 2.x

// M745X_PLLTable: For MPC7451, MPC7451, MPC7455 <= 2.x

short M745X_PLLTable[] = { // MPC7451 and MPC7455 V3.2 and earlier
                                // PLL
          5,            // 0x00
          0,            // 0x01
         75,            // 0x02
        150,            // 0x03
         70,            // 0x04
          0,            // 0x05
         10,            // 0x06 bypass
          0,            // 0x07
         20,            // 0x08
          0,            // 0x09
         65,            // 0x0A
        130,            // 0x0B
         25,            // 0x0C
          0,            // 0x0D
          0,            // 0x0E
         90,            // 0x0F
         30,            // 0x10
          0,            // 0x11
         55,            // 0x12
        110,            // 0x13
         40,            // 0x14
        100,            // 0x15
         50,            // 0x16
        120,            // 0x17
         80,            // 0x18
        140,            // 0x19
         60,            // 0x1A
        160,            // 0x1B
         35,            // 0x1C
          0,            // 0x1D
          0,            // 0x1E off
          0             // 0x1F
};

// M7455_PLLTable: For MPC7455 V3.3 or later and MPC7457

short M7455_V3_PLLTable[] = {
                                // PLL            MPC7451 comparison
        115,            // 0x00                 (was 0.5x)
        170,            // 0x01                 NEW
         75,            // 0x02                 (same)
        150,            // 0x03                 (same)
         70,            // 0x04                 (same)
        180,            // 0x05                 NEW
         10,            // 0x06 bypass  (same)
        200,            // 0x07                 NEW
         20,            // 0x08                 (same)
        210,            // 0x09                 NEW
         65,            // 0x0A                 (same)
        130,            // 0x0B                 (same)
         85,            // 0x0C                 (was 2.5x)
        240,            // 0x0D                 NEW
         95,            // 0x0E                 NEW
         90,            // 0x0F                 (same)
         30,            // 0x10                 (same)
        105,            // 0x11                 NEW
         55,            // 0x12                 (same)
        110,            // 0x13                 (same)
         40,            // 0x14                 (same)
        100,            // 0x15                 (same)
         50,            // 0x16                 (same)
        120,            // 0x17                 (same)
         80,            // 0x18                 (same)
        140,            // 0x19                 (same)
         60,            // 0x1A                 (same)
        160,            // 0x1B                 (same)
        135,            // 0x1C                 (was 3.5x)
        280,            // 0x1D                 NEW
          0,            // 0x1E off             (same)
        125             // 0x1F                 NEW
```

## A.3 Step 3: Set the Cache L3 Bits

```
//-----------------------------------------------------------------------
//  cache_inval_enable_L3 - Invalidate and enable the L3
//
// If this routine is entered, we will use as much of arg0 (in r3)
//   as makes sense to configure the L3.  User should pass exactly the final
//   desired L3CR setting, including L3CR[L3E].
// In fact, if the user doesn't pass in L3CR with L3E set (e.g, like the
//   user mistakenly gives us all zeroes) let's just exit without enabling.
// If this routine is entered with the L3 enabledthe user is asking for
//   the same settings we will just return.
// If this routine is entered with the L3 enabled and the user is asking for
//   different settings we will flush and disable the L3 before invalidating
//   and reenabling.
//
// Register usage:
//      r3 - enter with desired L3CR value and return with actual L3CR
//      r4 - temporary mask(ed) values for L3CR
//      r5 - target for L3CR reads for polling
//      r6 - save MSSCR0 during MPC7451_errata21 workaround and restore
//      r9 - current PVR shifted right 16 bits
//      r12 - saved link register (r12 will not be used by routines called)
//-----------------------------------------------------------------------

        .global cache_inval_enable_L3
cache_inval_enable_L3:

        mfspr   r9, 287                 // read PVR
        rlwinm  r9,r9,16,16,31  // shift down 16 bits
        cmpli   0,0,r9,0x8000   // Vger pvr
        beq     cache_inval_enable_L3_MPC7450
        cmpli   0,0,r9,0x8001   // Apollo pvr
        beq     cache_inval_enable_L3_MPC7450
        cmpli   0,0,r9,0x8002   // Apollo7 pvr
        beq     cache_inval_enable_L3_MPC7450
        xor     r3,r3,r3// All others: There is no L3 to enable.
        b       cache_inval_enable_L3_exit// just return zero

// Cache initialization, invalidation, and enabling for MPC7450, and 7455
cache_inval_enable_L3_MPC7450:
// Check that L3 is not enabled already with User's
// desired value; return if it is
        mfspr   r5,1018         // Get L3CR
        cmpw    r3,r5           // Is L3 already configured as user desires?
        beq     cache_inval_enable_L3_exit// Yes, exit without changes
// Check if L3 was already enabled with other values and needs flushing.
        andis.  r4,r5,0x8000// Is L3E/0 already set?
        beq     check_L3E_MPC7450// No (result = 0), continue configuring
        mflr    r12                     // Yes, save link register
        bl      cache_flush_disable_L3// Flush and disable
        mtlr    r12                     // Restore link register
        mfspr   r5,1018                 // Refresh r5 with L3CR
check_L3E_MPC7450:
// Allow init of L3CR without the L3E bit set. This allows initializing
// the L3 interface while using the entire space for Private Memory.
// tpeters - 111602
// Don't enable if user didn't set L3E
//      andis.  r4,r3,0x8000    // Did the user ask to set L3E?
//      bne     init_L3_Cache_MPC7450// Yes (result !=0), then go initialize.
//      mr      r3,r5                   // No, return the current value of L3CR
//      b       cache_inval_enable_L3_exit// and exit without changes

init_L3_Cache_MPC7450:

#define MPC7450_errata21  //This is supposed to be perf impact only
#ifdef MPC7450_errata21  // D fetches may allocate in IONLY L3 or
                                // I fetches may allocate in DONLY L3
        mfspr   r6,1014         // Save current value of MSSCR0 = spr 1014
        lis     r4,0x0040// IONLY/9
        ori     r4,r4,0x0040// DONLY/25
        and.    r4,r3,r4// Are we doing an IONLY or DONLY init
        beq     init_L3_wo_errata21 // No, then  MPC7450_errata21 doesn't apply
        ori     r4,r6,0x0003
        xori    r4,r4,0x0003// Turn off HW prefetching MSSCR0[30:31]
        sync
        mtspr   1014,r4
init_L3_wo_errata21:

#endif

//      Do not allow user to set L3CR[L3E/0]; we'll do that below.

        lis     r4,0x73ef

                //  L3E/0            = 0   (don't allow enabling yet)
                //  L3PE/1           = x   (parity)
                //  L3APE/2          = x   (address parity)
                //  L3SIZ/3          = x   (0 - 1MB; 1 - 2MB)
                //  L3CLKEN/4        = 0   (L3 clock enable -
                //                          see MPC7450_errata20)
                //  L3CR/5           = 0   (Must be set by software -
                //                          named L3DX?)
                //  L3CLK[6:8]       = xxx (L3 divider)
                //  IONLY/9          = x   (Instructions only)
                //  L3CR[10]         = x   (Extended L3CLK on MPC7457)
                //  L3CR[11]         = 0   (reserved)MPC7457 L3CKSPEXT
                //  L3CR[12]         = x   (L3OH1)MPC7455 only
                //  L3SPO/13         = x   (sample point overide)
                //  L3CKSP[14:15]    = xx  (clock sample point)

cmpli   0,0,r9,0x8002        // Apollo7 pvr
        beq     MPC7457_L3CR_mask    //need different mask for non-7457 parts
        ori     r4,r4,0xf3c5         //bit 30 is reserved for non-7457
        b       no_MPC7457_L3CR_mask
MPC7457_L3CR_mask:
        ori     r4,r4,0xf3c7
no_MPC7457_L3CR_mask:


                //  L3PSP[16:18]     = xxx (processor sample point)
                //  L3REP/19         = x   (Replacement algorithm - 0 default)
                //  L3HWF/20         = 0   (L3 Hardware Flush - not here)
                //  L3I/21           = 0   (global invalidate - not set here)
                //  L3RT[22:23]      = xx  (SRAM type - MSUG2 DDR SRAMs)
                //  L3CYA/24         = x   (L3 SRAM clock control bit)
                //  DONLY/25         = x   (data only)
                //  L3CR[26:28]      = 000 (reserved)
                //  PMEN/29          = x   (Private memory enable)
                //  L3CR/30          = 0   (reserved on non-MPC7457)
                //  PMSIZ/30         = x   (L3PM 4MB on MPC7457)
                //  PMSIZ/31         = x   (0 - 1MB; 1 - 2MB)
        and     r4,r3,r4// Mask off non-user bits, preserve r3
        oris    r4,r4,0x0400// Enforce L3DX/5 in L3CR pattern (not optional)

//      dssall        // Cancel any pending data stream operations with dssall
        .long 0x7e00066c // Not all assemblers generate
                                // the dssall instruction yet
        sync           // Finish any pending store ops in LSU (UM p3-65)

#define MPC7450_errata20
#ifdef MPC7450_errata20
// Set all L3CR bits to their desired values except L3CLKEN, L3E, L3PE, and L3I
//      lis     r5,0xbffff//this mask is incorrect - tpeters 111602
//      ori     r5,r5,0xffff
//      andis.  r4,r3,0xbfff
        lis     r5,0x37ff//correct mask to make sure L3CLKEN, L3E
        ori     r5,r5,0xfbff//L3PE, and L3I are not enabled - tpeters
        and     r4,r4,r5//Set all L3CR bits to their
                                //desired values except
        mtspr   1018,r4         //L3CLKEN, L3E, L3PE, and L3I
#endif

        oris    r4,r4,0x0800// Set L3CR[L3CLKEN/4]
        mtspr   1018,r4
        li      r5,128          // Wait for minimum 100 processor clocks
inval_L3_MPC7450_wait:// after changing L3CLKEN
        addi    r5,r5,-1
        cmpi    0,0,r5,0// Same as cmpwi r5,0  CHECK ELSEWHERE
        bne     inval_L3_MPC7450_wait// Perform a delay loop
        ori     r4,r4,0x0400// Set L3CR[L3I/21] to start invalidate routine
        mtspr   1018,r4
inval_L3_MPC7450_poll:
        mfspr   r5,1018
        mtcrf   0xff,r5    // poll for invalidate to complete, L3CR[L3I/21] = 0
        bc      0xC,21,inval_L3_MPC7450_poll
```

```
// Look at SRAM type; if PB2, do not play with L3ITCR0 register below.
// Same for late-write SRAM don't change the L3ITCR0

        li      r5,0x0300       // 3 = PB2
        and     r5,r3,r5        // Mask off non-user bits
        cmpi    0,0,r5,0x0300
        beq     skip_L3ITCR0
        cmpli   0,0,r9,0x8002   // Apollo7 pvr
        beq     skip_L3ITCR0    // Leave default L3ITCR0 setting for
                                        // MPC7457

// Somerset recommends the following adjustments to L3ITCR0 for the 7450 & 7455
// for Samsumg 4Mb DDR.

        mfspr   r5,984          // L3ITCR0: Supposed to adjust SRAM Clock edges
        rlwinm  r5,r5,28,4,31// for data being read from bus by adding
        oris    r5,r5,0xf000// four additional taps on a pulse shaper
        ori     r5,r5,0x00ff
        xori    r5,r5,0x007f
        mtspr   984,r5
        isync

skip_L3ITCR0:

#ifdef MPC7450_errata20
        xoris   r4,r4,0x0800// L3CR[L3CLKEN/4]
        mtspr   1018,r4         // Set L3CLKEN off now
        sync                    // UM says put a sync here (7450UM p3-66)
        li      r5,128          // Wait for minimum 100 processor clocks
inval_L3_MPC7450_wait2:// after changing L3CLKEN
        addi    r5,r5,-1
        cmpi    0,0,r5,0// Same as cmpwi r5,0
        bne     inval_L3_MPC7450_wait2// Perform a delay loop
        oris    r4,r4,0x0800    // Set L3CLKEN
//check if L3PE was set in r3, if it was set it now - tpeters
        andis.  r5,r3,0x4000//is L3PE set by user?
        or      r4,r4,r5//set L3PE if it was set in r3
#endif
        xor     r5,r5,r5// Clear r5 for store to MSSSR0 - tpeters
        mtspr   1014,r5         // Clear the sticky L3TAG parity status error
                                //  which  might  have  been  set  during
invalidation.
        oris    r4,r4,0x0800// Set L3CLKEN
```

```
// Now we check if the user wanted the L3E bit set.
// r3 should still have the user L3CR setting preserved.
        andis.  r5,r3,0x8000//Did the user ask to set L3E?
        or      r4,r4,r5//set L3E if it was set in r3

        mtspr   1018,r4
        sync                    // Somerset puts this here

#ifdef MPC7450_errata20
        addi    r5,r0,128// Wait for another 100 processor clocks
inval_L3_MPC7450_wait3:// because we changed CLKEN for the errata
        addi    r5,r5,-1
        cmpi    0,0,r5,0
        bne     inval_L3_MPC7450_wait3// Perform a delay loop
#endif

#ifdef MPC7450_errata21
        mfspr   r5,1014         // If we changed the value of MSSCR0,
        or      r5,r5,r6// Restore HW prefetching MSSCR0[30:31]
        mtspr   1014,r5         // to whatever it was.
        sync                    // sync per 7450UM p3-66
#endif

        mfspr   r3,1018         // Return whatever got set

cache_inval_enable_L3_exit:
        blr
```

## A.4 Step 3: Setup and Perform Software Flush

1. The following code fragment from cache.S shows how DINK32 determines the type of processor flash and performs the correct flushing mechanism .

```
// Need to disable interrupts to avoid interference with flushing
       mfmsr   r10                           // Get current MSR value and save
        ori             r4,r10,0x8000   // Clear MSR[EE/16]
        xori    r4,r4,0x8000
        mtmsr   r4                              // Disable interrupts
// Which part are we dealing with?
        mfspr   r9,287          //Only use upper half of PVR
//Be careful! srawi r9,r9,16 will sign extend the MPC7450 and MPC7455
        rlwinm  r9,r9,16,16,31
//Be careful! cmpi r9,0x8nnn will sign extend and not be equal on 7450/7455
        cmpli   0,0,r9,0x0008   //Is this MPC750/MPC755?
        beq     HW_flush_750_DCache //Yes? Use HW flush assist.
        cmpli   0,0,r9,0x000c   //Is this MPC7400?
        beq     HW_flush_7400_DCache    //Do hardware assisted flush
        cmpli   0,0,r9,0x800C   //Is this MPC7410?
        beq             HW_flush_7400_DCache    //Do hardware assisted flush
        cmpli   0,0,r9,0x8000   //Is this MPC7450?
        beq             SW_flush_7450_DCache    //Do software flush
        cmpli   0,0,r9,0x8001   //Is this MPC7455?
        beq             SW_flush_7450_DCache    //Do software flush
cmpli   0,0,r9,0x8002   //Is this MPC7457?
        beq             SW_flush_7450_DCache    //Do software flush
```

2. Notice that the MPC7451 also uses a software flush mechanism, therefor the code fragment shown below is not new for the MPC7455/57, only the determination of the processor type of MPC7455/57 is new. The software flush algorithm for the MPC7451/55/57 is shown below for those users who may not have L1 software flushing code.

```
// Software flush of L1 Dcache on MPC7450/7455
SW_flush_7450_DCache:

// tpeters - 042203 - changed the software flush algorithm to how the 745x UM
// recommends.
        li      r7,8    //counter for 8 ways
        li      r6,0*128*32
        lis     r5,0x40 //arbitrary flush start address
flush_7450_L1_top:
        li      r4,128  // Prepare to do a unique load to each way of each
        mtctr   r4      // of the 138 sets.
flush_7450_L1:
        lwzx    r4,r6,r5        // load way X from known address
        dcbf    r6,r5           // flush way X if modified in cache
        addi    r6,r6,0x20      // increment to next cache line
        bdnz    flush_7450_L1   // repeat
        addic.  r7,r7,-1        // decrement way counter
        bne     flush_7450_L1_top //flush next way of each set

flush_L1D_comp:
// Re-enable interrupts if they were on
        mtmsr   r10                             // Write back saved MSR value
flush_L1D_exit:
        blr
```

## A.5 Various Settings at Initialization

1. Enable AltiVec

```
// For MPC7xxx parts, enable AltiVec so we can save/restore the
// contents.

        cmpli   0,0,r20,0x000C          // MPC7400
        beq     enable_altivec
        cmpli   0,0,r20,0x8000          // MPC7450
        beq     enable_altivec
        cmpli   0,0,r20,0x8001          // MPC745x
        beq     enable_altivec
        cmpli   0,0,r20,0x8002          // MPC7457
        beq     enable_altivec
        cmpli   0,0,r20,0x800C          // MPC7410
        bne     no_enable_altivec
```

2. Do set V’ger modes

```
// Init PID/PIR from MSSCR0

        cmplwi  r20,0x8000              // MPC7450
        beq     set_vger_modes
        cmplwi  r20,0x8001              // MPC7455
        beq     set_vger_modes
        cmplwi  r20,0x8002              // MPC7457
        bne     not_vger

set_vger_modes:
        mtspr   pir,r0                  // PID(PIR) = 0.
        sync

        mfspr   r3,1008                 // Current HID0
        oris    r3,r3,0x0400            // Set TBEN
        oris    r3,r3,0x0010            // Set DPM
```

3. Initialize HID0 with the default from config.h

```
        lis     r4,DEFAULT_HID0_7457@h
        ori     r4,r4,DEFAULT_HID0_7457@l
        cmpli   0,0,r20,0x8002        // An MPC7457? (i.e. PVR = 0x8002_nnnn?)
        beq     init_HID0
```

4. Do initialize L2 (This is not necessary, the L2 can be initialized at any point in the code)

```
// For V'ger and Apollo, enable the L2 cache now, always

        cmplwi  r20,0x8000              // MPC7450
        beq     init_l2_now
        cmplwi  r20,0x8001              // MPC7455
        beq     init_l2_now
        cmplwi  r20,0x8002              // MPC7457
        bne     no_init_l2

init_l2_now:
//      lis     r3,0x8000               // L2EN
        lis     r3,0xC000               // L2EN + L2PAR
        bl      cache_inval_enable_L2
no_init_l2:
```

5. Do set v’ger Java mode

```
//----------------------------------------------------------------------------
// V'ger powers up with the AltiVec FP in Java-mode.  Change it to
// precise, like all the other CPUs.  Note that this has to be done
// after L1D cache is enabled.

// Currently still takes an exception -- load is WT*

#ifdef CHANGE_VGER_JAVA
        .global vger_java
vger_java:
        cmplwi  r20,0x8000              // MPC7450
        beq     set_vger_java
        cmplwi  r20,0x8001              // MPC7455
        beq     set_vger_java
        cmplwi  r20,0x8002              // MPC7457
        bne     not_vger_java

set_vger_java:
```

## A.6 Test for MPC7457

Save the MPC7451/55/57 specific SPRs.

```
//  check for the MPC7450 and MPC7455
        lis     r4,process_type@h
        ori     r4,r4,process_type@l
        lbz     r4,0(r4)
        cmpi    0,0,r4,PPCAPOLLO  // PPCAPOLLO MPC7455 from config.h
        bne     cr0,noMPC7455
        mfspr   r4,l3itcr0                              // L3ITCR0= 984
        stw     r4,L3ITCR0_LOC*spr_inc(r3)
noMPC7455:
        lis     r4,process_type@h
        ori     r4,r4,process_type@l
        lbz     r4,0(r4)
        cmpi    0,0,r4,PPCVGER  // PPCVGER MPC7450 from config.h
        bne     cr0,noMPC7450
        mfspr   r4,l3itcr0                              // L3ITCR0= 984
        stw     r4,L3ITCR0_LOC*spr_inc(r3)
noMPC7450:

//  check for the MPC7457
        lis     r4,process_type@h
        ori     r4,r4,process_type@l
        lbz     r4,0(r4)
cmpi    0,0,r4,PPCAPOLLO7  // PPCAPOLLO7 MPC7457 from config.h
        bne     cr0,noMPC7457
// save MPC7457 special purpose registers.
        mfspr   r4,l3ohcr                               // L3OHCR= 1000
        stw     r4,L3OHCR_LOC*spr_inc(r3)
        mfspr   r4,l3itcr0                              // L3ITCR0= 984
        stw     r4,L3ITCR0_LOC*spr_inc(r3)
        mfspr   r4,l3itcr1                              // L3ITCR1= 1001
        stw     r4,L3ITCR1_LOC*spr_inc(r3)
        mfspr   r4,l3itcr2                              // L3ITCR2= 1002
        stw     r4,L3ITCR2_LOC*spr_inc(r3)
        mfspr   r4,l3itcr3                              // L3ITCR3= 1003
        stw     r4,L3ITCR3_LOC*spr_inc(r3)

noMPC7457:
```

Restore the MPC7451/55/57 specific SPRs

```
//  check for the MPC7450 and MPC7455
        lis     r4,process_type@h
        ori     r4,r4,process_type@l
        lbz     r4,0(r4)
        cmpi    0,0,r4,PPCAPOLLO  // PPCAPOLLO MPC7455 from config.h
        bne     cr0,noStoreMPC7455
        lwz     r4,L3ITCR0_LOC*spr_inc(r3)
        mtspr   l3itcr0,r4                              // L3ITCR0= 984
noStoreMPC7455:

        lis     r4,process_type@h
        ori     r4,r4,process_type@l
        lbz     r4,0(r4)
        cmpi    0,0,r4,PPCVGER  // PPCVGER MPC7450 from config.h
        bne     cr0,noStoreMPC7450
        lwz     r4,L3ITCR0_LOC*spr_inc(r3)
        mtspr   l3itcr0,r4      // L3ITCR0= 984
noStoreMPC7450:

//  check for the MPC7457
        lis     r4,process_type@h
        ori     r4,r4,process_type@l
lbz     r4,0(r4)
        cmpi    0,0,r4,PPCAPOLLO7  // PPCAPOLLO7 MPC7457 from config.h
        bne     cr0,noStoreMPC7457
// save MPC7457 special purpose registers.
        lwz     r4,L3OHCR_LOC*spr_inc(r3)
        mtspr   l3ohcr,r4                               // L3OHCR= 1000
        lwz     r4,L3ITCR0_LOC*spr_inc(r3)
        mtspr   l3itcr0,r4                              // L3ITCR0= 984
        lwz     r4,L3ITCR1_LOC*spr_inc(r3)
        mtspr   l3itcr1,r4                              // L3ITCR1= 1001
        lwz     r4,L3ITCR2_LOC*spr_inc(r3)
        mtspr   l3itcr2,r4                              // L3ITCR2= 1002
        lwz     r4,L3ITCR3_LOC*spr_inc(r3)
        mtspr   l3itcr3,r4                              // L3ITCR3= 1003
noStoreMPC7457:
```

***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 Muenchen, 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
support.asia@freescale.com

***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

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 the 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.

freescale™
semiconductor