PowerPC™

IBM27-82660 Errata Summary Revision 1.3

MPR660ESU-05

This document details differences between the operation of the 660 Bridge chipset as described in the IBM27–82660 PowerPC to PCI Bridge User's Manual*, IBM document number MPR660UMU–01, and the actual operation of the Revision 1.1 chipset (IBM27–82663 Rev. 1.0 and IBM27–82664 Rev. 1.1). Where applicable, workarounds are described. The information in this summary also applies to Rev. 2.0 of the IBM27-82663.*

January 1996

IBM ®

Document Revision History

Version 1.0 (April 1995)	(MPR660ESU–00) Errata for the 1.0 revision of the chipset.
Version 1.1 (Unpublished)	(MPR660ESU–02) Updated errata for the 1.0 revision of the chipset.
Version 1.2 (August 1995)	(MPR660ESU–03) Errata for the 1.1 revision of the chipset.
Version 1.3 (January 1996)	(MPR660ESU–05) Updated errata.
Product Update 2.0 (June 5, 1995)	(MPR660ESU–01) Changes from IBM27-82663 rev 1.0 to rev. 2.0.
Product Update 2.1 (October 30, 1995)	(MPR660ESU–04) Updated changes.

Part Identification, IBM27-82660 Chipset

IBM27-82663 (663 Buffer)
Chipset Rev. 1.0 contains IBM27-82663 Rev. 1.0. The package is marked 94G0235.

Chipset Rev. 1.1 contains IBM27-82663 Rev. 1.0. The package is marked 94G0235.

Some errata apply to IBM27-82663 Rev. 2.0. The package is marked 94G0178.

IBM27-82664 (664 Controller)
Chipset Rev. 1.0 contains IBM27-82664 Rev. 1.0. The package is marked 94G0232.

Chipset Rev. 1.1 contains IBM27-82664 Rev. 1.1. The package is marked 94G0176.

IBM27-82660 Errata Summary 1.3

For the IBM27-82660 Rev. 1.1 (IBM27-82663 Rev. 1.0 and IBM27-82664 Rev. 1.1)

January 1996

All references to the 660 Bridge chipset, the 663 Buffer, and the 664 Controller refer to the above revisions of these devices. All references to the *660 Bridge User's Manual* refer to the *IBM27–82660 PowerPC to PCI Bridge User's Manual*, IBM document number MPR660UMU–01, which is a beta level document. More information is contained in the revision 2.0 and 2.1 Product Updates.

Unless otherwise defined, the errata listed in this document detail differences between the operation of the 660 Bridge chipset as described in the 660 Bridge User's Manual and the actual operation of the revision 1.1 chipset.

Section 1 lists the individual errata.

Section 2 contains the logic design files and a suggested implementation drawing for a set of two 22V10 PALs that can be used to implement the external logic that is required to correct various errata.

Section 3 contains the logic design files and a suggested implementation drawing for a set of three 16V8 PALs that can be used to implement the external logic that is required to correct various errata.

Revision bars (like the one to the left of this paragraph) indicate areas of this document which have been changed or added since the previous revision. The PAL equations have not changed.

1 Individual Errata

The following sections contain a detailed breakdown of individual 660 Bridge Errata. All equations are shown in PALASM™ format: INVERT is /, OR is +, and AND is *. The term *performance* refers to the number of clock cycles required to complete an operation.

Advisory 1 . . . NCR 53C810 SCSI Controller

Date 6/21/95

Functional . . . . . . On certain PCI configuration writes to the SCSI controller, the 664 will wait several (>3) PCI clocks before asserting IRDY#. Under these conditions the SCSI controller appears to latch in the write data when it asserts TRDY# even if IRDY# is not yet asserted. The manufacturer states that this problem exists on NCR part number 609–0391399, but that it has been corrected on later parts, starting with part number 609–1391635.

Work Around . . . Use part number 609–1391635 or later.

Impact . . . . None.

Errata 1 . . . . . . MIO Test

Date 6/09/95

Functional . . . . . . The 664 MIO test function is incorrect.

Work Around . . . None.

Impact . . . . None. This errata only affects chip-level testing.

Errata 2 . . . . . . PCI Busmaster Timing

Date 6/19/95

Functional . . . . . . When a PCI busmaster accesses system memory, the 664 CPU bus arbiter samples FRAME#, AD[31:30], and C/BE[3:0] on CPU clock edges rather than on PCI clock edges. If these signals are switching at the time of sampling, then the PCI bus may hang (the 664 may not assert TRDY#).

Work Around . . .

Option 1: Set the CPU bus frequency below 50Mhz to ensure that the signals are not sampled while switching.

Option 2: Cut FRAME# between the 664 and the rest of the PCI bus. Connect the net on each side of the cut to a PAL. When a PCI busmaster runs a cycle, hold FRAME# (as an input to the 664) deasserted until the second (low) half of the PCI clock, so that the 664 samples it active on the rising edge of PCI_CLK.

Impact . . . . Option 1 affects maximum operating frequency. Option 2 has no effect on performance, but may affect the maximum operating frequency of the PCI bus.

Errata 3 . . . . . . TA# Not Tristated for CPU Bus Targets

Date 6/09/95

Functional . . . . . . The 664 fails to tristate TA# when a cycle is claimed (using CPU_BUS_CLAIM#) by a CPU bus target and the CPU bus is not pipelined and the L2 is enabled.

Work Around . . . Disable the L2 when using a CPU bus target.

Details . . . . To properly disable the 660 L2:
1) Do not enable the L2 cache. This is controlled by System Control 81C BCR (addr 8000081Ch) bit 6, which is set to 0 during reset. Do not set this bit to 1.
2) Disable the internal cache controller. This is controlled by the Cache Status Register (index B1h) bit 1, which is set to 1 during reset. Set this bit to 0.

Errata 4 . . . . . . ECIWX, ECOWX and TBST#

Date 6/09/95

Functional . . . . . . If an ECIWX or ECOWX bus operation is run with TBST# active, then the 664 incorrectly treats the cycle as a burst by providing four TA# assertions. However, when an ECIWX or ECOWX bus operation is run, the TBST# and TSIZE do not actually define the tranfer size, the size is always 4 bytes.

Work Around . . . Do not execute ECIWX or ECOWX instructions.

Impact . . . . This workaround has no effect on performance or maximum frequency.

Errata 5 ...... PCI Busmaster Disconnect on Refresh

Date 6/19/95

Functional ...... The 664 asserts spurious TA#s when a PCI busmaster read from memory is target–disconnected due to a memory refresh request while a CPU to PCI address tenure is outstanding.

Work Around ... Cut BG1# (and BG2# if MP) between the 664 and the CPU. Connect the net on each side of the cut to a PAL. When a PCI busmaster access to memory is target disconnected, deassert BG1# (and BG2#) for six PCI CLKs.

Impact .... This workaround has negligible impact on performance but may affect the maximum operating frequency of the CPU bus.

Errata 6 ...... PCI Busmaster Hang

Date 6/19/95

Functional ...... PCI to memory reads hang when the PCI busmaster initiates the transaction while a CPU to memory data tenure is outstanding and a CPU to PCI address tenure is outstanding.

These PCI busmaster accesses to memory hang the CPU and PCI busses if they occur while A0 is high and a CPU–to–memory cycle (only) is pending. A0 can go high during a CPU–to–memory cycle if a pullup resister is put on A0, such that A0 goes high after AACK# of the CPU–to–memory cycle, but before the data tenure ends.

A0 can also go high during a CPU–to–memory cycle when a CPU–to–memory data tenure is outstanding and a CPU–to–PCI address tenure is outstanding, and then a PCI busmaster initiates a PCI to memory transaction. When the PCI busmaster transaction is detected, the CPU–to–PCI cycle is ARTRYed (since it cannot complete), leaving A0 high. If the CPU–to–memory data tenure is still outstanding at this time, then the busses hang.

Work Around ... Drive CPU_ADDR[0] low (PAL fix) on the CPU bus clock following the assertion of AACK#. Also, do not place a pullup resistor on A0.

Impact .... This workaround has no effect on performance or maximum frequency.

Errata 7 ...... Bus Hang During L2 Read Hit + CPU to Memory + PCI to Memory

Date 7/11/95

Functional ...... If a PCI busmaster begins a system memory access while a CPU to memory address tenure is pending and an L2 read hit data tenure is pending, then the CPU and PCI busses will hang (TA# and TRDY# are never asserted).

Work Around ... Cut BG1# (and BG2# if MP) between the 664 and the CPU. Connect the net on each side of the cut to a PAL. Deassert BG1# (and BG2#) on the clock following the assertion of AACK#. If asynchronous SRAM is used in the L2, then also deassert BG1# (and BG2# if MP) while DBB# is active, from the time PCI_GNT# goes inactive until ABB# has asserted twice.

Impact .... This workaround has negligible impact on performance but may affect the maximum operating frequency of the CPU bus.

Errata 8 . . . . . . Back-To-Back PCI Busmaster Write

Date 6/22/95

Functional . . . . . . The CPU and PCI busses hang if a PCI busmaster write to memory begins while the CPU is asserting BR# and the CPU address bus is becoming free from the broadcast snoop caused by a previous PCI busmaster write to memory.

Work Around . . . Cut BG1# (and BG2# if MP) between the 664 and the CPU. Connect the net on each side of the cut to a PAL. Deassert BR1# (and BR2#) while PCI_CLK is high and a PCI busmaster is accessing memory.

Impact . . . . This workaround has negligible impact on performance but may affect the maximum operating frequency of the CPU bus.

Errata 9 . . . . . . 603 Snoop on PCI Busmaster Write

Date 6/27/95

Functional . . . . . . The snoop operation that the 664 broadcasts on the CPU address bus when a PCI busmaster begins a write to memory is Write–with–Kill (00110b). For a 603, the snoop is supposed to be Write–with–Flush (00010b).

Work Around . . . In 603/603e designs, cut TT[2] between the 664 and the CPU bus. Pull both signals (664_TT[2] and CPU_TT[2]) down with 500 ohm resistors. This workaround is not required in 604 designs.

Impact . . . . This workaround has no effect on performance or maximum operating frequency.

Errata 10 . . . . CPU to PCI Bus Clocks at 3:1

Date 7/21/95

Functional . . . . . . If the CPU:PCI clocks are running at a frequency ratio of 3:1, and a CPU to memory data tenure is pending, then PCI busmaster to memory accesses hang.

Work Around . . . Do not operate the CPU:PCI bus clocks at 3:1.

Impact . . . . This workaround has no effect on performance or maximum frequency.

Errata 11 . . . . . PCI Bus Hang On Pipelined CPU to PCI Master Abort

Date 8/08/95

Functional . . . . . . Normally, when a CPU initiates a PCI bus access, if the 660 bridge initiates the PCI transaction before the CPU data tenure begins (the CPU bus is pipelined), then FRAME# is held asserted (more than the usual 2 PCI_CLKs) until the CPU data tenure begins.

If the 660 bridge detects a PCI master abort condition (due to no DEVSEL#) before the CPU data tenure begins, then the 660 bridge will hang the PCI bus. The CPU bus transfer will complete normally.

If the PCI target watchdog timer is enabled then the busses will not hang, since the hung PCI data phase will cause the PCI cycle to terminate after 2000 PCI clocks. However, if the CPU attempts to again access the PCI bus before this point, then the CPU cycle will be lost (terminated on the CPU bus without executing on the PCI bus).

Work Around . . . Cut PCI_GNT# between the 664 and the PCI arbiter. Connect the net on each side of the cut to a PAL. Deassert PCI_GNT# (to the 664) from the time the CPU starts a cycle to the PCI until the CPU is no longer pipelined (DBB# goes inactive).

Impact . . . . This workaround has negligible impact on performance but may affect the maximum operating frequency of the CPU bus.

Errata 12 .... 663 Memory Read

Date 7/24/95, 9/18/95

Functional ...... Under certain conditions CPU_RDL_OPEN goes low before MEM_RD_SMPL goes low during a CPU to memory read transfer, causing the 663 to supply corrupted data to the CPU bus on CPU_DATA[1, 3, 5, ...].

Work Around ... For proper operation, MEM_RD_SMPL must switch a minimum interval before CPU_RDL_OPEN switches. This is achieved by adding a resistor to the CPU_RDL_OPEN net between the 664 and the 663.

Column 3 of Table 1 shows the minimum time interval that the 663 requires between the switching of MEM_RD_SMPL and the switching of CPU_RDL_OPEN. Column 4 shows the minimum interval supplied by the 664:

Table 1. MEM_RD_SMP to CPU_RDL_OPEN Timing

Case	Conditions	663 Requires (Min)	664 Provides (Min)
1	Worst Case Process, T_J and V_{DD}	1.8ns	1.7ns
2	Best Case Process, Worst Case T_J and V_{DD}	0.2ns	0.5ns
3	Best Case Process, T_J and V_{DD}	0.1ns	0.3ns

The worst reasonable combination of conditions is when the 664 is in Case 2 (provides 0.5ns difference) but the 663 is in Case 1 (requires 1.8ns difference). These conditions require at minimum an additional 1.3ns of delay to be added to CPU_RDL_OPEN.

Add a 200 ohm series resister to the CPU_RDL_OPEN net. (Delay = RC = 200 ohm * 12pf = 2.4ns). This assumes the CPU_RDL_OPEN and MEM_RD_SMPL nets are both about three inches long and the resister is close to the 664. A different resister value may be required if net lengths or resister placement differ significantly.

Impact .... This workaround has no effect on performance but may affect the maximum operating frequency of the CPU bus.

Errata 13 .... Extraneous CPU to PCI Cycle

Date 8/8/95

Functional ...... Normally, if a CPU to PCI memory write is pending, and a PCI busmaster initiates a system memory access, then the CPU cycle is retried. Under certain conditions however, the CPU to PCI transaction is also posted inside the 660, causing the CPU to PCI transaction to be executed once by the 660 and once by the CPU. In addition, the 660 executes the posted transaction with random levels on the C/BE# lines.

The incorrect posting occurs when the address tenure of the CPU–to–PCI transfer begins more than 4 clocks before its data tenure begins (posting logic is waiting for the data tenure), and the PCI busmaster transaction is detected before the data tenure of the CPU–to–PCI cycle begins.

Work Around ... Cut BG1# (and BG2# if MP) between the 664 and the CPU. Connect the net on each side of the cut to a PAL. Deassert BG1# (and BG2#) from the beginning of each CPU bus address tenure until 4 CPU_CLKs before the associated data tenure ends. This can be done by waiting until the 3rd TA# is received for memory cycles, or until one clock after AACK# on L2 cache hits.

Impact .... This workaround has a less than 2% impact on performance, and may affect the maximum operating frequency of the CPU bus.

Errata 14 .... TT[0:4] Float to ECIWX/ECOWX

Date 8/8/95

Functional ...... If the CPU TT[0:4] lines float to the value 1x10xb while a CPU data tenure is pending, then the 664 will malfunction, usually by exhibiting incorrect behavior on the TA# line.

Details .... During a CPU bus transfer, TT[0:4] are driven to a correct value, and then allowed to float for up to 16 CPU clocks in certain situations. The 664 incorrectly samples their value during the above conditions. As long as there are no current sources operating on the TT lines during the float, these lines will maintain their value due to bus capacitance.

Work Around ... Do not put pullup or pulldown resistors, TTL–type inputs, or any other current sources on TT[0, 2, 3]. For 603 or 603e designs only, workaround 9 overrides this workaround; pull down TT[2]_664 and TT[2]_CPU as shown in errata 9.

Impact .... This workaround has no effect on performance or maximum frequency.

Errata 15 .... Spurious Set of the Flash Write Lockout Bit

Date 8/1/95

Functional ...... While the 660 is configured for direct-connect ROM (PCI-based), and a ROM read cycle is pipelined over a single-beat CPU to memory write to an odd memory location, then the flash write lockout bit (located at register address FFFF FFF1h) may be inadvertantly set (without a write to the aforementioned address). Once written, this bit can only be reset via power on reset. Once the bit is set, the PCI-based flash ROM can not be updated.

Work Around ... A hardware workaround is TBD.
A possible software workaround is to prevent the above scenario, possibly by adding sync cycles before the ROM reads.

Impact .... This workaround has no effect on performance or maximum frequency.

Errata 16 .... PCI Lock in 603 1:1 Mode

Date 8/1/95

Functional ...... While the 660 bridge is configured for 603 1:1 mode (which ensures a wait state between TS# and AACK#) the assertion of PCI_LOCK# may cause a system hang.

Details .... This bug has only been predicted by system simulation, and has not been observed in the lab.

Work Around ... Do not use PCI_LOCK# while in 603 1:1 mode.

Impact .... This workaround has no effect on performance or maximum frequency.

Errata 17 .... 663 Parity Error Detection

Date 8/4/95

Functional ...... Parity errors on the CPU and memory data busses are only detected by the 663 if there are an odd number of bytes containing parity errors. All single bit errors (in the number of bytes transferred during that beat) are reported, but double bit errors may not be reported, even if the two errors are in different bytes.

Work Around ... TBD.

Impact .... This errata has no effect on performance or maximum frequency.

Errata 18 .... CPU Bus Target Hang

Date 9/12/95

Functional ...... When the 660 bridge L2 controller is disabled, and an ARTRYed memory accesses is pipelined over the data tenure of a memory access, and the next cycle is claimed by a CPU bus target, then the CPU bus hangs.

Work Around ... Included in the workaround for errata #13.

Errata 19 .... Error Handling of EIEIO

Date 10/13/95

Functional ...... The error handling logic incorrectly detects EIEIO as a CPU transfer type error.

Work Around ... Always mask CPU transfer type errors (Error Enable 1 bit 0). Note that since CPU transfer type errors are enabled following reset, a CPU transfer type error may already have been detected before it can be masked. Therefore, the Error Status Register 1 should be written with value 02 (to clear any errors that have already been detected) after the mask is enabled.

Errata 20 .... Error Simulation Registers

Date 10/13/95

Functional ...... Error Simulation Register 1 and Error Simulation Register 2 are not (and will not be) implemented. Writing to these BCRs has no effect.

Work Around ... Do not use these BCRs.

Errata 21 .... 663 Simultaneous Switching

Date 12/11/95

Functional ...... The 663 and the 664 are designed to work together such that the 663 never switches more than 32 CPU data bus outputs on the clock before the CPU samples data.

However, if a PCI busmaster read from memory is followed by a CPU read from memory such that the CPU memory (read) data is available four CPU bus clocks after the last data for the PCI is available, then the CPU_DATA_OE# signal from the 664 to the 663 is asserted such that the CPU data bus drivers are enabled on the clock before the CPU samples data. Depending on the previous and the current data pattern, this can result in up to 64 CPU data bus outputs switching on the clock before the CPU samples data.

If this occurs, simultaneous switching noise within the 663 can result in the CPU sampling incorrect data.

The time between the last PCI data available and the first CPU data available is four clocks when the CPU cycle is a page hit and Tasc+Tcpw=4 and Tcpw+Tcp=4. Tasc is determined by Memory Timing Register 2 bit 7. Tcpw is determined by Memory Timing Register 2 bits 3:2. Tcp is determined by Memory Timing Register 2 bits 6:5.

Work Around ... Either set Tasc+Tcpw to a value greater than four, or set Tcpw+Tcp to a value greater than four. Either option will add CPU bus cycle(s) such that CPU_DATA_OE# will not be asserted during the cycle before the CPU samples the read data.

Changing Tasc may require changes to other parameters. See the 660 User's Manual for more information.

Errata 22 .... PCI Busmaster Long Burst Read From Slow Memory

Date 1/5/96

Functional ...... PCI BM burst reads from (system) memory may hang if the memory controller is configured with memory timings Tcp + Tcpw = 5 or 6 clocks. The reset value of Memory Timing Register 2 sets Tcp + Tcpw = 6 clocks.

Work Around ... Set up the memory controller timings such that Tcp + Tcpw <= 4 clocks (these values are controlled by Memory Timing Register 2, index A2h). Note that Tcp + Tcpw > 4 clocks is only useful for DRAM memory that is slower than 80ns.

2 PAL Design Files – Two Large PALs

The following PAL equations describe a set of two 22V10 PALs that can be used to implement the workarounds in the Errata section. These are PALASM design files.

2.1 CPUPAL MC3 Design Files

```
;------------------------- Declaration Segment -------------------
TITLE    MC.pds
PATTERN  none
REVISION 1.0
COMPANY  IBM
DATE     08/10/95

CHIP  _MC  PALCE22V10

;--------------------------- PIN Declarations --------------------
;
;PIN  1   CPU_CLK                ; CLOCK                    (PIN 2)
;PIN 12   GND                    ; GROUND                   (PIN 14)
;PIN 24   VCC                    ; VCC                      (PIN 28)

;  Inputs

PIN  2    BG_IN_                  ; INPUT                    (PIN 3)
PIN  3    ABB_                    ; INPUT                    (PIN 4)
;PIN  4                           ; INPUT                    (PIN 5)
PIN  5    MASK_BG2                ; INPUT                    (PIN 6)
PIN  6    MASK_BG1                ; INPUT                    (PIN 7)
;PIN  7                           ; INPUT                    (PIN 9)
PIN  8    DBB_                   ; INPUT                    (PIN 10)
PIN  9    PCI_GNT_IN_            ; INPUT                    (PIN 11)
PIN 10    SRAM_OE_               ; INPUT                    (PIN 12)
PIN 11    TA_                    ; INPUT                    (PIN 13)
;PIN 13                          ; INPUT                    (PIN 16)

;  outputs

PIN 14    CPU_A0          COMB   ; COMB                     (PIN 17)
PIN 15    BG_OUT_         COMB   ; COMB                     (PIN 18)


;  Internal Registers

;PIN 16                   REG    ; REG                      (PIN 19)
PIN 17    DEL_ABB_        REG    ; REG                      (PIN 20)
PIN 18    MASK_BG3        REG    ; REG                      (PIN 21)
PIN 19    MASK_7A         REG    ; REG                      (PIN 23)
PIN 20    MASK_7B         REG    ; REG                      (PIN 24)
PIN 21    TA_CNT0         REG    ; REG                      (PIN 25)
PIN 22    TA_CNT1         REG    ; REG                      (PIN 26)
PIN 23    MASK_PCI_GNT    REG    ; REG                      (PIN 27)
```

```
;------------------- Boolean Equation Segment ----------------
;
;   Output enables for Comb logic

CPU_A0.TRST = ABB_ * /DEL_ABB_          ;for Errata #6

;   Equations for internal registers

DEL_ABB_ = ABB_

MASK_PCI_GNT = /ABB_ * DEL_ABB_ * CPU_A0 * /DBB_  ;for Errata #11
             + MASK_PCI_GNT * /DBB_           ; Delay granting PCI to
                       ; the CPU when a CPU to PCI cycle is pipelined
                       ; over the data tenure of the previous cycle

BG_OUT_  = BG_IN_
         + MASK_BG1               ;for Errata #5
         + MASK_BG2               ;for Errata #5
         + MASK_BG3               ;for Errata #7

CPU_A0 = GND                      ;for Errata #6

;*************************************************************
;
;  Below is fix for Errata #7B, 11, 13,
TA_CNT0 = /TA_CNT0 * /TA_
        + TA_CNT0 * /DBB_ * TA_

TA_CNT1 = TA_CNT0 * /TA_
        + TA_CNT1 * /DBB_

MASK_BG3 = (/TA_CNT0 + /TA_CNT1) * SRAM_OE_ * /DBB_  ;ERR 13
         + /ABB_ * /SRAM_OE_ * /DBB_                 ;ERR 7B
         + PCI_GNT_IN_ * /DBB_                       ;ERR 11
         + MASK_7A * /ABB_ * DEL_ABB_* /DBB_         ;ERR 7A
         + MASK_7B * /(/ABB_ * DEL_ABB_) * /DBB_     ;ERR 7A

;*****************************************************************
;ERRATA 7 (fix for Async caches)
;MASK_7B feeds wires to CPUPAL MASK_BG3 input to be included
;        in BG equation shown below:

MASK_7A = PCI_GNT_IN_
        + MASK_7A * /(/ABB_ * DEL_ABB_)

MASK_7B = MASK_7A * /ABB_ * DEL_ABB_
        + MASK_7B * /(/ABB_ * DEL_ABB_)

;------------------------ END OF FILE --------------------------
```

2.2 PCIPAL MPP1 Design Files

```
;-------------------------- Declaration Segment -------------------
TITLE    MPP.pds
PATTERN  none
REVISION 1.0
COMPANY  IBM
DATE     08/10/95

CHIP  _MPP  PALCE22V10

;---------------------------- PIN Declarations --------------------
;
;PIN 12   GND                        ; GROUND                  (PIN 14)
;PIN 24   VCC                        ; VCC                     (PIN 28)

;
;  Inputs
;
PIN  1   PCI_CLK                     ; CLOCK                   (PIN 2)

PIN  2   IRDY_                       ; INPUT                   (PIN 3)
PIN  3   TRDY_                       ; INPUT                   (PIN 4)
PIN  4   STOP_                       ; INPUT                   (PIN 5)
PIN  5   PCI_GNT_IN_                 ; INPUT                   (PIN 6)
PIN  6   MASK_PCI_GNT                ; INPUT                   (PIN 7)
PIN  7   BR_IN_                      ; INPUT                   (PIN 9)
;PIN  8                              ; INPUT                   (PIN 10)
;PIN  9                              ; INPUT                   (PIN 11)
;PIN 10                              ; INPUT                   (PIN 12)
;PIN 11                              ; INPUT                   (PIN 13)
;PIN 13                              ; INPUT                   (PIN 16)

;  outputs
;
PIN 14    PCI_FRAME_      COMB       ;                         (PIN 17)
PIN 15    664_FRAME_      COMB       ;                         (PIN 18)
PIN 21    PCI_GNT_OUT_    COMB       ;                         (PIN 25)
PIN 22    BR_OUT_         COMB       ;                         (PIN 26)
PIN 23    DEL_PCI_CLK     COMB       ;                         (PIN 27)

;  Internal Registers
;
PIN 16    664_OWN_PCI     REG        ;                         (PIN 19)
PIN 17    MASK_BG2        REG        ;                         (PIN 20)
PIN 18    MASK_BG1        REG        ;                         (PIN 21)
PIN 19    MASK_BG0        REG        ;                         (PIN 23)
PIN 20    DEL_PCI_FRAME_  REG        ;                         (PIN 24)
```

```
;---------------------- Boolean Equation Segment ------------------
;   Output enables for Comb logic (Reg OE's controllered by pin 11)
;

PCI_FRAME_.TRST  = 664_OWN_PCI
664_FRAME_.TRST  = /664_OWN_PCI

;   Combinational and registered equations
;

DEL_PCI_FRAME_ = PCI_FRAME_

664_OWN_PCI = /PCI_GNT_IN_ * PCI_FRAME_ * IRDY_
            + 664_OWN_PCI * /(PCI_FRAME_ * IRDY_)

PCI_FRAME_ = 664_FRAME_                  ; Errata #2

664_FRAME_ = PCI_FRAME_
           + PCI_CLK * DEL_PCI_FRAME_    ; Errata #2

;*****************************************************************

MASK_BG2 = /PCI_FRAME_ * /IRDY_ * /TRDY_ * /STOP_     ; Errata #5
         + MASK_BG2 * MASK_BG1
         + MASK_BG2 * MASK_BG0

MASK_BG1 = /PCI_FRAME_ * /IRDY_ * /TRDY_ * /STOP_      ; Errata #5
         + MASK_BG2 * /MASK_BG1 * /MASK_BG0
         + MASK_BG1 * MASK_BG0

MASK_BG0 = /PCI_FRAME_ * /IRDY_ * /TRDY_ * /STOP_      ; Errata #5
         + MASK_BG2 * /MASK_BG0
         + MASK_BG1 * /MASK_BG0

;*****************************************************************

DEL_PCI_CLK = PCI_CLK

BR_OUT_ =  BR_IN_
        + /DEL_PCI_CLK * /PCI_FRAME_ * /664_OWN_PCI  ;for Errata #8

;*****************************************************************

PCI_GNT_OUT_ = PCI_GNT_IN_
             + MASK_PCI_GNT

;------------------------ END OF FILE -----------------------
```

2.3 Workaround PAL Installation (Two PAL Solution)

Figure 1 shows a typical installation of the workaround PALs in a generic 603e/604 uniprocessor system.

PAL connections shown without an asterisk do not require any nets to be cut. Merely connect the PAL pin to the an existing net (eg: TA#) or, in the case of the clocks, to another clock driver output.

* Cut these nets between the 664 and everything else. Then connect the PALs to each side of the cut as shown.

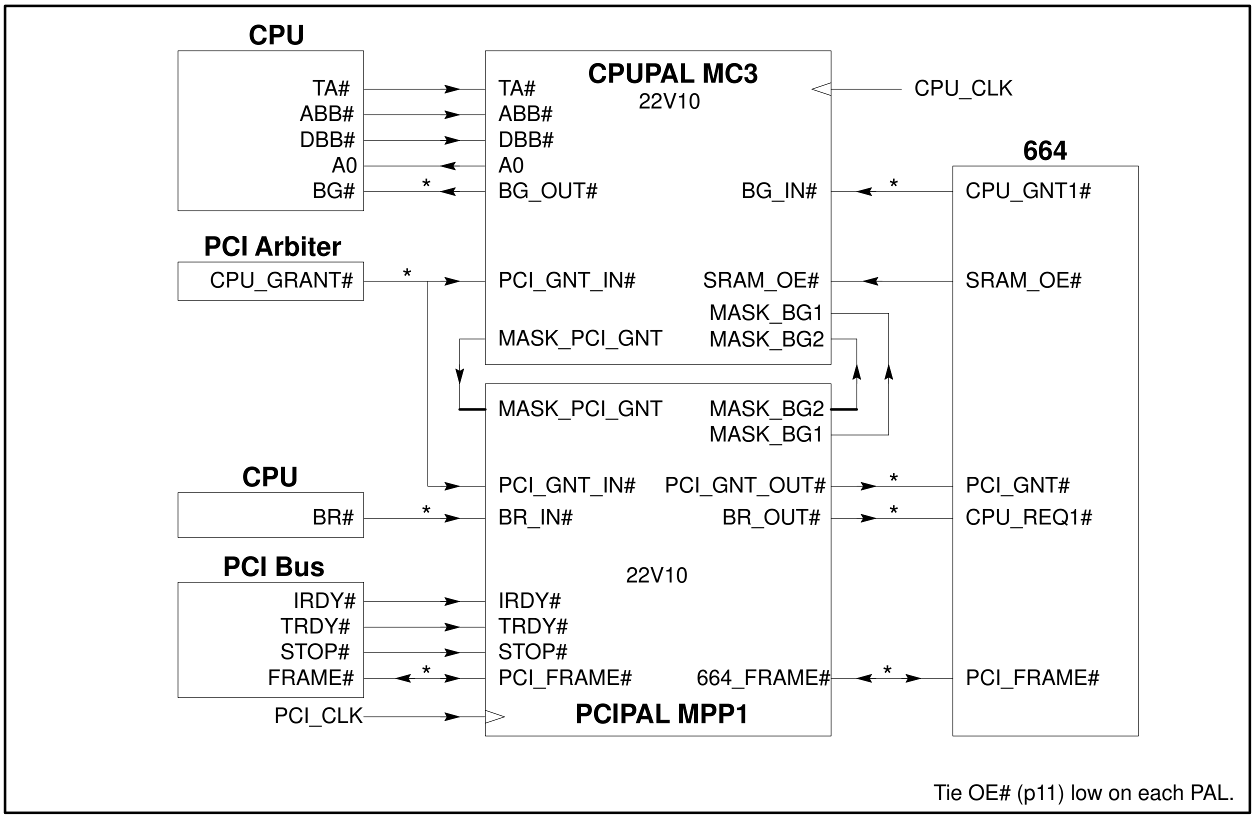

Figure 1. Generic Implementation PAL Installation

3 PAL Design Files – Three Small PALs

The following PAL equations describe a set of three 16V8 PALs that can be used to implement the workarounds in the Errata section. These are PALASM design files.

3.1 HPP1 Design File

```
TITLE    HPP.pds
PATTERN  none
REVISION 1.0
COMPANY  IBM
DATE     06/29/95
CHIP  _FPP  PALCE16V8

;--------------------------- PIN Declarations --------------------
;
;  Predefined
;
;PIN  1   PCI_CLK                   ; CLOCK
;PIN 10   GND                       ; GROUND
;PIN 11    REG_OE#                      ;  OUTPUT ENABLE FOR REGISTERED
OUTPUTS
;PIN 20   VCC                       ; VCC

;  Inputs
;
PIN  2    IRDY_                     ; INPUT from PCI bus
PIN  3    TRDY_                     ; INPUT from PCI bus
PIN  4    PCI_CLK                   ; INPUT
PIN  5    STOP_                     ; INPUT from PCI bus
PIN  6    PCI_GNT_                  ; INPUT from PCI bus arbiter
;PIN  7                              ; INPUT
;PIN  8                              ; INPUT
;PIN  9                              ; INPUT

;  outputs
;
PIN 12     PCI_FRAME_       COMB      ; CONBINATIONAL  goes to/from PCI
bus
PIN 13      664_FRAME_        COMB       ; COMBINATIONAL  goes to/from
Kauai
PIN 19    MASK_BR          COMB     ; COMBINATIONAL  goes to H603C PAL
;
;  Internal Registers
;
PIN 14    664_OWN_PCI      REG      ; REGISTERED
PIN 15    DEL_PCI_FRAME_   REG      ; REGISTERED
PIN 16    MASK_BG0         REG      ; REGISTERED     goes to H603C PAL
PIN 17    MASK_BG2         REG      ; REGISTERED     goes to H603C PAL
PIN 18    MASK_BG1         REG      ; REGISTERED
```

```
;--------------------- Boolean Equation Segment -------------------
;
EQUATIONS
;
;   Output enables for Comb logic (Reg OE's controllered by pin 11)
;

PCI_FRAME_.TRST  = 664_OWN_PCI
664_FRAME_.TRST  = /664_OWN_PCI

;
;   Combinational and registered equations
;

DEL_PCI_FRAME_ = PCI_FRAME_

664_OWN_PCI = /PCI_GNT_ * PCI_FRAME_ * IRDY_
            + 664_OWN_PCI * /(PCI_FRAME_ * IRDY_)

PCI_FRAME_ = 664_FRAME_                  ; Errata #2

664_FRAME_ = PCI_FRAME_
           + PCI_CLK * DEL_PCI_FRAME_    ; Errata #2

;**************************************************************

MASK_BG2 = /PCI_FRAME_ * /IRDY_ * /TRDY_ * /STOP_     ; Errata #5
         + MASK_BG2 * MASK_BG1
         + MASK_BG2 * MASK_BG0

MASK_BG1 = /PCI_FRAME_ * /IRDY_ * /TRDY_ * /STOP_      ; Errata #5
         + MASK_BG2 * /MASK_BG1 * /MASK_BG0
         + MASK_BG1 * MASK_BG0

MASK_BG0 = /PCI_FRAME_ * /IRDY_ * /TRDY_ * /STOP_      ; Errata #5
         + MASK_BG2 * /MASK_BG0
         + MASK_BG1 * /MASK_BG0

;**************************************************************

MASK_BR = /PCI_CLK * /PCI_FRAME_ * /664_OWN_PCI    ;for Errata #8

;------------------------ END OF FILE -----------------------
```

3.2 C3 PAL Design File

```
TITLE    C.pds
PATTERN  none
REVISION 1.0
COMPANY  IBM
DATE     07/31/95

CHIP  _C2  PALCE16V8

;---------------------------- PIN Declarations --------------------
;  Predefined
;
;PIN  1   CPU_CLK                 ; CLOCK
;PIN 10   GND                     ; GROUND
;PIN 11    REG_OE#                     ; OUTPUT ENABLE FOR REGISTERED
OUTPUTS
;PIN 20   VCC                     ; VCC

;
;  Inputs
;

PIN  2    BG_IN_                  ; INPUT
PIN  3    ABB_                    ; INPUT
PIN  4    BR_IN_                  ; INPUT
PIN  5    MASK_BG2                ; INPUT
PIN  6    MASK_BG1                ; INPUT
PIN  7    MASK_BR                 ; INPUT
PIN  8    DBB_                    ; INPUT
PIN  9    PCI_GNT_IN_             ; INPUT
PIN 17    MASK_BG3                ; INPUT


;  outputs
;
PIN 12    CPU_A0        COMB      ; COMBINATIONAL
PIN 13    BR_OUT_       COMB      ; COMBINATIONAL
PIN 18    PCI_GNT_OUT_  COMB      ; COMBINATIONAL
PIN 19    BG_OUT_       COMB      ; COMBINATIONAL
;
;  Internal Registers
;
PIN 14    MASK_PCI_GNT  REG       ; REG
PIN 15    DEL_ABB_      REG       ; REGISTERED
PIN 16    DEL_BG_       REG       ; REGISTERED
```

```
;---------------------- Boolean Equation Segment -------------------
;
EQUATIONS
;
; Output enables for Comb logic (Reg OE's controllered by pin 11)
;

CPU_A0.TRST = ABB_ * /DEL_ABB_          ;for Errata #6

;
;   Equations for internal registers
;

DEL_BG_  = BG_IN_

DEL_ABB_ = ABB_

MASK_PCI_GNT = /ABB_ * DEL_ABB_ * CPU_A0 * /DBB_  ;for Errata #11
             + MASK_PCI_GNT * /DBB_               ; Delay granting PCI
to
                                                  ; the CPU when a CPU
to
PCI_GNT_OUT_  =  PCI_GNT_IN_                           ; PCI cycle is
pipelined
             + MASK_PCI_GNT                     ; over the data tenure
of
                                                ; the previous cycle

BG_OUT_  = BG_IN_
         + MASK_BG1              ;for Errata #5
         + MASK_BG2              ;for Errata #5
         + MASK_BG3              ;for Errata #7

CPU_A0 = GND                     ;for Errata #6

BR_OUT_ = BR_IN_
         + MASK_BR               ;for Errata #8
;------------------------- END OF FILE --------------------------
```

3.3 Broom PAL Design File

```
;-------------------------- Declaration Segment -------------------
TITLE    BROOM1.pds
PATTERN  none
REVISION 1.0
COMPANY  IBM
DATE     08/02/95
CHIP  _BROOM1  PALCE16V8
;--------------------------- PIN Declarations --------------------
;
;  Predefined
;
;PIN  1   CPU_CLK                 ; CLOCK
;PIN 10   GND                     ; GROUND
;PIN 11   REG_OE#             ; OUTPUT ENABLE FOR REGISTERED OUTPUTS
;PIN 20   VCC                     ; VCC
;
;  Inputs
;
PIN  2    DBB_                    ; INPUT
PIN  3    PCI_GNT_IN_             ; INPUT
PIN  4    ABB_                    ; INPUT
PIN  5    TA_                     ; INPUT
PIN  6    SRAM_OE_                ; INPUT
;PIN  7                            ; INPUT
;PIN  8                            ; INPUT
;PIN  9                            ; INPUT
;
;  Outputs
;
;PIN 13                  COMB      ; COMBINATIONAL
;PIN 14                  COMB      ; COMBINATIONAL
;
;  Internal Registers
;
PIN 12    DEL_ABB_       REG      ; REG
PIN 15    MASK_BG3       REG      ; REG
PIN 16    MASK_7A        REG      ; REGISTERED
PIN 17    MASK_7B        REG      ; REGISTERED
PIN 18    TA_CNT0        REG      ;
PIN 19    TA_CNT1        REG      ;
;
;
```

```
;-------------------- Boolean Equation Segment ----------------
;
EQUATIONS
;
;   Output enables for Comb logic (Reg OE's controllered by pin 11)
;   Equations for internal registers
;
;*******************************************************************

DEL_ABB_  = ABB_

;*******************************************************************
;
;  Below is fix for Errata #7B, 11, 13,
;
;******************************************************************
TA_CNT0 = /TA_CNT0 * /TA_
        + TA_CNT0 * /DBB_ * TA_

TA_CNT1 = TA_CNT0 * /TA_
        + TA_CNT1 * /DBB_

MASK_BG3 = (/TA_CNT0 + /TA_CNT1) * SRAM_OE_ * /DBB_  ;ERR 13
         + /ABB_                                     ;ERR 7B
         + PCI_GNT_IN_ * /DBB_                       ;ERR 11
         + MASK_7A * /ABB_ * DEL_ABB_* /DBB_         ;ERR 7A
         + MASK_7B * /(/ABB_ * DEL_ABB_) * /DBB_     ;ERR 7A

;*****************************************************************
;ERRATA 7 (fix for Async caches)
;MASK_7B feeds wires to CPUPAL MASK_BG3 input to be included in BG
equation
;   shown below:
;
;BG_OUT_  = BG_IN_
;         + MASK_BG2              ;for Errata #5
;         + MASK_BG1              ;for Errata #5
;         + ABB_ * /DEL_ABB_      ;for Errata #7
;         + MASK_BG3 * /DBB_      ;for Errata #7 (Asych cache fix)
;******************************************************************

MASK_7A = PCI_GNT_IN_
        + MASK_7A * /(/ABB_ * DEL_ABB_)

MASK_7B = MASK_7A * /ABB_ * DEL_ABB_
        + MASK_7B * /(/ABB_ * DEL_ABB_)

;------------------------ END OF FILE --------------------------
```

3.4 Workaround PAL Installation (Three PAL Solution)

Figure 2 shows a typical installation of the workaround PALs in a generic 603e/604 uniprocessor system.

PAL connections shown without an asterisk do not require any nets to be cut. Merely connect the PAL pin to the an existing net (eg: TA#) or, in the case of the clocks, to another clock driver output.

* Cut these nets between the 664 and everything else. Then connect the PALs to each side of the cut as shown. For example, cut the PCI_FRAME# net at the 664, connect only the 664 PCI_FRAME# pin to the 664_FRAME# pin of the HPP1 PAL, and connect all other devices on the FRAME# net to the PCI_FRAME# pin of the HPP1 PAL.

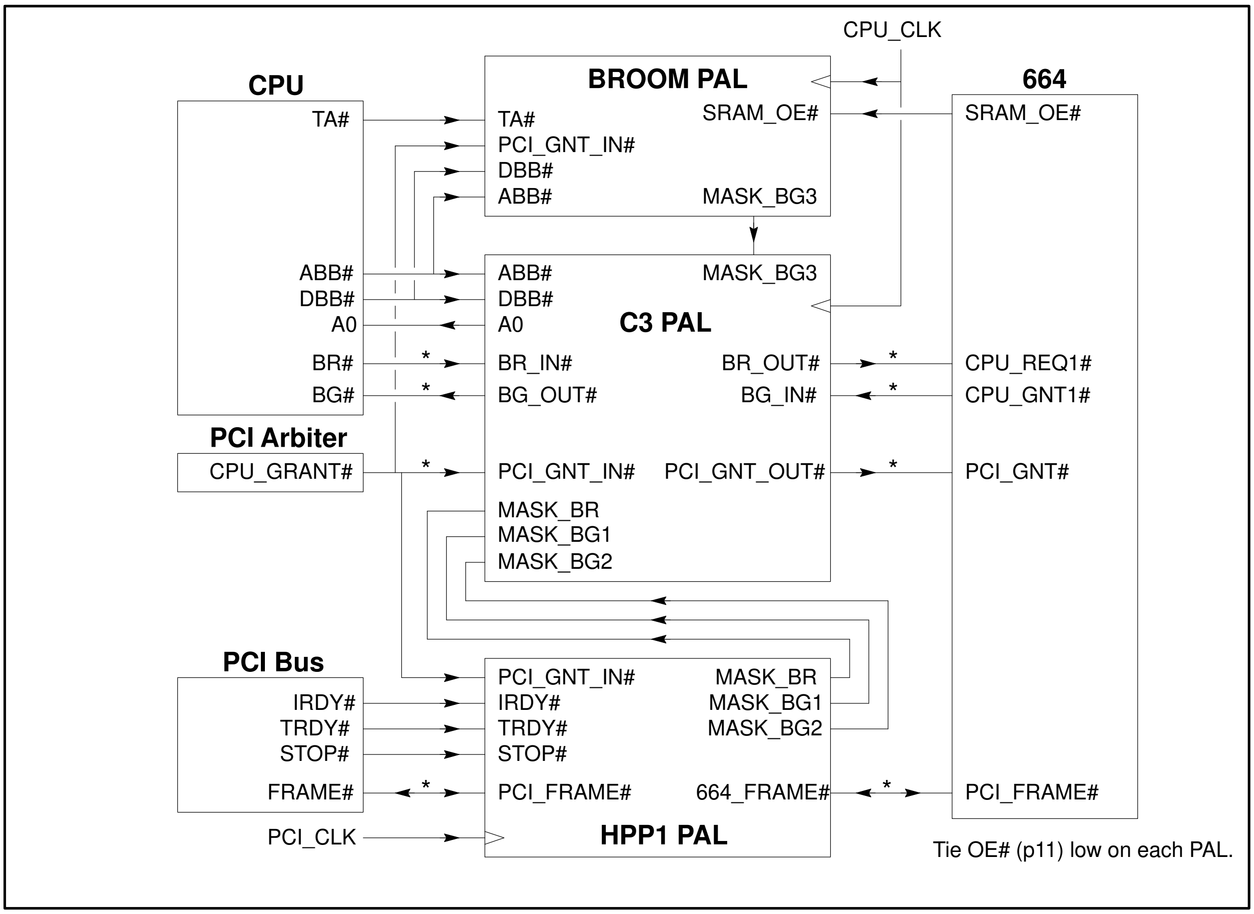

Figure 2. Generic Implementation PAL Installation

PowerPC

USA and Canada:
IBM Microelectronics Division
1580 Route 52, Bldg. 504
Hopewell Junction, NY 12533-6531
Tel: (800) PowerPC
Fax: (800) PowerFax
http://www.chips.ibm.com
http://www.ibm.com
ftp://ftp.austin.ibm.com/pub/PPC_support

Document Number: MPR660ESU–05