Introduction

This document describes the changes between the AMCC PowerPC 440GP and 440GX from a software perspective. The 440GX User's Manual and related documents will discuss the changes in detail. The purpose of this document is to highlight the changes and identify the areas where software updates may be necessary. The reader should be familiar with the details of the 440GP before reading this document.

The 440GX is largely backwards compatible with the 440GP. While the 440GX contains new components, software systems which have been written for the 440GP and do not attempt to use the new features will run largely unchanged. In this way existing software environments can exploit the increased performance of the 440GX without a large software migration effort.

Alternatively, software can be upgraded to take advantage of the new features of the 440GX. When using these new features some of the backwards compatibility to the 440GP is not available.

Overview

The 440GX incorporates the following new or changed features compared to the 440GP:

- Updated PowerPC 440 core with parity
- Configuration register changes
- 2x 10/100/gigabit ethernets(EMAC4)
- 2x 10/100 ethernets upgraded to EMAC4
- 2x TCP/IP acceleration hardware
- Additional interrupt controllers
- Messaging Unit
- PCI-X upgrade, PCI 2.3 compliant
- 256Kbytespacket/code store SRAM
- Level 2 cache
- General Purpose Timers changes
- DDR-SDRAM additions in timing register

New and Updated Features

440 embedded processor core

The updated PowerPC 440 embedded processor core adds the following new features:

- Parity on I-cache and D-cache
- Parity on UTLB
- New Machine Check Handling
- New User-mode instruction: **isel**
- D-cache full-line flush capability
- Timer Clock changes

Parity

As technology continues to shrink transistors to smaller and smaller dimensions and use lower voltages, there is an increasing chance of data corruption from a variety of sources, such as naturally occurring background radiation. These are often referred to as soft errors (SER). One method for protecting data against unnoticed corruption is the use of parity bits on the memory areas of microprocessors.

On the 440GX, parity bits are associated with the I-cache, D-cache and unified translation look-aside buffer (UTLB). There are many parity bits for each line of data. For example, each data cache line has 39 parity bits (one for each byte, two for the tag, one for the U-field and one for each of the four dirty bits). Checking for parity errors is always enabled.

Parity errors are reflected by asserting an asynchronous Machine Check exception. Details of the parity error are provided in the new Machine Check Status Register (MCSR).

Whether a parity error is recoverable or not depends on the type of error. I-cache parity errors are always recoverable. The recovery procedure involves invalidating the entire I-cache and restoring any locked lines back into the cache.

In order for D-cache errors to be fully recoverable, certain conditions must be followed: only using a write-through strategy on all cachable pages and setting a configuration bit to defer all load confirmations to a later stage in the processor pipeline -CCR0[PRE] (Parity Recoverability Enable). These conditions may contribute to reduced performance, but they ensure that a D-cache parity error is fully recoverable by invalidating the D-cache (and restoring any locked lines). Alternatively, if those restrictions are not observed then a D-cache parity error is only semi-recoverable. Recovery would involve the operating system terminating the process which had the error. Some systems may choose to reset the processor completely.

Parity errors in the TLB are mostly recoverable, as long asCCR0[PRE] is set. The machine check handler can use the **tlbre** instruction to read all TLB entries to find the one with bad parity and restore it from memory. The handler must first clear the MCSR[MCS, TLBE] bits, then issue a **tlbre** for each TLB entry. The act of reading a TLB entry which has a parity error will itself cause a parity exception. However, as machine checks are disabled (MSR[ME]=0) in the machine check handler this will not cause another machine check to occur, but the status bits MCSR[MCS, TLBE, IMCE] will be set. After each **tlbre**, the handler must check the MCSR[TLBE] bit to see whether the **tlbre** found the TLB entry with bad parity. The corrupted TLB entry can be restored from a good copy in main memory. The handler must clear the MSCR[MCS, TLBE, IMCE] bits before it exits.

Note that if there is a parity error on the TLB entry which maps the machine check exception handler then the machine check handler may not be called, so no recovery is possible in this one case. One technique which may be used to reduce the likelihood of this occurring is as follows. Dedicate one TLB entry solely for mapping the machine check handler. Refresh the contents of this TLB entry periodically. The more often it is refreshed, the less likely it is that the TLB entry will be corrupted between the time of the refresh and the time that it is used when a machine check occurs.

Parity bits are returned by the three instructions which explicitly read the cache lines or TLB entries: **dcread**, **icread** and **tlbre**. This affects the contents of these registers updated by these instructions: DCDBTRL, ICDBTRH and the GPR which is the RT target of **tlbre**. The parity bits are returned in bits which were previously reserved in the target registers. For compatibility with the 440GP, these new bits are not returned by default, and must be enabled by setting the CCR0[CRPE] (Cache Read Parity Enable) bit. Details of the changes can be found in the *Change Details* section of this document.

There is no write equivalent of the **icread** and **dcread** instructions, but TLB entries can be written with **tlbwe**. When writing a TLB entry with the **tlbwe** instruction, the parity bits in the register being written are ignored and do not contribute to the parity calculated and stored in the TLB. This means that existing code which writes TLB entries can still run with no modification.

Note that when initializing TLB entries, even by just clearing the Valid bit, it is necessary to initialize the whole TLB, by using 3 **tlbwe** instructions with WS=0,1,2. This ensures the parity in the whole TLB entry is set correctly. On 440GP it was possible to just clear the Valid bit with one **tlbwe** instruction, but on 440GX all 3 instructions must be used to initialize the entire TLB entry. Failure to do this may result in TLB parity error machine check exceptions.

A mechanism is provided to force parity errors to occur. This can be useful when testing software designed to recover from or report parity errors. This is implemented by writing to bit sin the CCR1 register.

Machine Check Handling

Machine check handling is changed in the 440GX. A third class of interrupt is introduced to join the existing “Interrupt” and “Critical Interrupt”. This is the “Machine Check Interrupt”. The Machine Check Interrupt handler has its interrupted context saved into two new registers: MCSRR0 which contains the return address and MCSRR1 which contains the saved contents of the MSR. The Machine Check interrupt handler should exit with the new instruction rfmc**i** -“return from machine check interrupt”, instead of the rfc**i** (return from critical interrupt) used previously. The machine check interrupt is enabled and disabled the same way as it was previously, by using the Machine Check Enable (ME) bit in the MSR.

A new SPR is added: MCSR, Machine Check Status Register. This contains status bits which detail the type of asynchronous machine check error which has occurred, which includes parity errors. A machine check handler can query this register and take the appropriate action to recover from the machine check, if possible.

As well as bits indicating the area of the parity error, the MCSR also contains a summary bit MCSR[MCS] which is set when an asynchronous machine check occurs. This single bit can be checked to see if any of the asynchronous exception bits has been set. Also included in the MCSR is the Imprecise Machine Check Exception bit, MCSR[IMCE]. This bit is set when an asynchronous machine check occurs but machine checks are disabled (MSR[ME]=0). If MCSR[MCS] is set when machine checks are enabled by setting MSR[ME], then a machine check exception will occur. Software should ensure that after handling all machine checks, MCSR[MCS] is cleared before exiting the machine check handler.

The reason for the new class of machine check interrupt is that it can occur during execution of a critical interrupt handler without corrupting its registersCSRR0 and CSRR1. Previously, a critical interrupt handler could not be interrupted unless it had saved away its own copies of CSRR0, CSRR1 and re-enabled critical interrupts. Even then, the first and last few instructions of the handler would not be interruptible while the registers were saved away and subsequently restored. Usually this was not a significant problem. The entry and exit portions of a critical interrupt handler are relatively simple to code, and could be written in such a way to reduce the likelihood of machine check interrupts occurring there. And if a machine check did occur during the execution of a critical interrupt handler, it is likely that it would not be recoverable anyway. The addition of a parity check as the cause of a machine check interrupt changes this. The entry and exit portions of the critical interrupt handler are no longer immune from having machine checks (for parity errors) occur while they are executing. Some of the parity errors are fully recoverable, so this flavor of machine check is no longer as catastrophic as other kinds. It is essential, however, that it be serviced as soon as it is detected. If it were turned off (as would be required during critical interrupt handler entry and exit) the parity error would not be detected until a later time when the interrupt was enabled, and this would be too late for a recovery. The solution is the new level of machine check interrupt, which can always be enabled, even during the prolog and epilog of critical interrupt handlers. This ensures that all parts of an operating system and applications are protected from recoverable parity errors. The one exception to this is the machine check handler itself of course -there is no way to recover from a machine check in the machine check handler, but this is no different from previous implementations.

New User-mode Instruction

A new non-privileged instruction is added in the 440GX. This is **isel** (integer select), which provides a mechanism for conditional assignment. This improves performance by reducing conditional branching. It has the following syntax:

isel RT,RA,RB,CRb

where RT, RA, RB are GPRs and CRb specifies a bit in the Condition Register, CR. If CR[CRb] is 1, then RT is assigned the value of RA|0, otherwise RT is assigned the value of RB.

isel provides a way to speed up the common operation of comparing two values then deciding which of them to assign based on the comparison. An example is an in-line macro for the “min” function, which takes two values and assigns the smaller to a third variable. In C code this may appear as:

z=min(x,y);

If the variables x, y and z are assigned to registers RX, RY and RZ respectively, this statement could be compiled to produce the following assembly code:

```
                cmpw    RX,RY
                blt     label1
                ori     RZ,RY,0
                b       next

label1:         ori     RZ,RX,0
next:
```

By using the **isel** instructions this code can be rewritten as:

```
cmpw RX,RY
isel RZ,RX,RY,CR0_LT
```

Note: The final argument of **isel** in this example is the constant CR0_LT, which represents the bit position of the LT (less than) bit in Condition Register field 0 (CR0). CR0_LT has the value 0.

D-cache full-line flush capability

An additional performance option is available to change the way that the data cache flushes lines back to memory.

It is possible that only part of a data cache line has been updated and is marked “dirty”. In the 440GP, when the line is flushed back to memory, if the dirty bytes are only located in one half of the cache line, only that half line is flushed back to memory. A new option is introduced in 440GX which allows the whole line to be flushed instead of just the dirty half line. This may have performance advantages in some circumstances.

Timer Clock Changes

A new bit is added to select whether the timers are based off the CPU clock or an external clock. The external clock is limited to a maximum frequency of half the CPU speed. The selection is made using CCR1[TCS] (Timer Clock Select); 0=CPU clock, 1=external clock.

Configuration Register Changes

The registers which control system clocking and configuration have changed significantly. In the 440GP there were a series of individually addressable DCRs called CPC0_xxx. For the 440GX, these have been split into two groups of registers: the Clocking and Power-On Reset (CPR) registers and the System DCR Registers (SDR). Each set of registers is indirectly addressed through an address and data pair of registers. Power management registers(CPC0_SR/ER/FR) remain as directly addressable DCRs, but with altered DCR names and addresses.

In the 440GP, many different and often unrelated bit fields were placed into the same configuration register. The new scheme implemented in the 440GX allows a more logical partitioning. For example, there are now separate SDR registers to hold the UART0 and UART1 parameters, rather than have them mixed with unrelated fields in the old CPC0_CR0. Information which was specific to the 440 CPU is now placed in its own SDR, rather than being scattered in CPC0_SYS0, CPC0_CR0 and CPC0_CR1.

New Ethernet Functions

The 440GX incorporates an updated ethernet core (EMAC4). One of the most notable features of this upgraded core is the removal of the need for two transmit channels for optimum performance. On the 440GP, a packet sent on one channel had to be completely transmitted onto the ethernet media before its final status could be returned and a second packet could be sent on the same channel. The new ethernet core has the ability to simultaneously transmit one packet on the ethernet media while accepting a second transmit packet on the same channel from the processor (via the MAL). It can then return the final transmit status for the first packet after the second packet has been accepted. Any 440GP ethernet code which uses two transmit channels will need to be rewritten to only use one. Also, device drivers will need to handle transmit status being returned for a previously-transmitted packet.

The 440GX supports jumbo packets (packets larger than 1518 bytes). However, the MAL only supports a maximum buffer size of 4K bytes. This means that jumbo packets larger than 4K will need to be split into multiple buffers. The operating system's communication stack must be able to handle these large packets which are split across multiple buffers. The 440GP did not support jumbo packets, so this is new functionality being added -existing software which does not support jumbo packets will continue to work as it did on the 440GP.

The EMAC4 in the 440GX adds two additional interrupts, which will require software support. These are:

- Parity Error. The FIFOs in the EMAC have parity bits associated with them, for similar reasons that parity has been added to the memory in the 440 core. A failure in the parity check in the FIFO will result in this interrupt. The interrupt status register will show both a CRC error and a parity error at the same time. The parity error can be handled the same way that a CRC error is currently handled.
- FIFO Limit Alert. When the transmit FIFO is about to be empty, or the receive FIFO is about to be full, this interrupt is generated. This allows the driver to change priority on the problematic channel to avoid an underrun/overrun condition. The limits where the interrupt are generated are programmable.

In addition to the two 10/100 ethernet controller sin 440GP, the 440GX adds two additional 10/100/gigabit ethernet controllers.

The ethernet controllers have a variety of interfaces available to communicate with a PHY. On the 440GP, the options for the two ethernets were either 1 MII (media independent interface), 2 RMII (reduced MII) or 2 SMII (serial MII)[1].On440GX these same options exist for the two 10/100 ether-nets. The 10/100/gigabit ethernets have a different set of options. When they are being operated in 10/100 mode, they can each use SMII. For gigabit (1000Mbps) speeds one controller may use the full GMII (gigabit MII) or TBI (ten-bit interface), or each controller can use RGMII (reduced GMII) or RTBI (reduced TBI).

1.SMII not available on all versions of 440GP due to errata.

The signal pins for the new ethernet controllers are multiplexed with existing signals, so there are some restrictions on the combination of features which can be accessed simultaneously. The other interfaces affected are external DMA channels numbers2 and 3, GPIO snumbers27 to 31, and the trace signals. The trace signals can be critical to debugging, so they are optionally provided in a second location, multiplexed with EBMI signals. The trace signals are always available at this location, provided that the trace/EBMI selection is set to trace, independently of which ethernet combination is selected. The possible combinations of ethernet interfaces are listed in *Ethernet combinations* table on page 6. The desired combination is selected by programming the ethernet group field in the new SDR0_PFC1 (Pin Function Control 1) register. From this table it can be seen that DMA channels 2 and 3 share pins with EMAC2 in gigabit mode, so that neither of those DMA channels are available for external use when ethernet 2 is configured for any speed other than 10/100 Mbps in SMII mode. Note that DMA channels 2 and 3 are still available for use within the processor by internal components or for memory-to-memory transfers. Also, the GPIO[27:31] and CPU trace pins are shared with the gigabit modes for ethernet 3, and so cannot be used simultaneously unless ethernet 3 is only configured for 10/100 Mbps SMII mode. System designers should bear these restrictions in mind when designing for the 440GX.

Table 1: Ethernet Combinations

Option #	EMAC0	EMAC1	EMAC2	EMAC3	External DMA2/3	Trace/GPIO	Trace on EBMI
0	MII				Y	Y	Y
1	RMII	RMII			Y	Y	Y
2	SMII	SMII	SMII	SMII	Y	Y	Y
3	RMII		RGMII/RTBI			Y	Y
4*	SMII	SMII	RGMII/RTBI	RGMII/RTBI			Y
			GMII/TBI				
5	SMII	SMII	SMII	RGMII/RTBI	Y		Y
6	SMII	SMII	RGMII/RTBI			Y	Y

Note: * Option 4 has two versions, either two RGMII/RTBI ethernets or one GMII/TBI. The choice is made in the RGMII/RTBI core

The gigabit interfaces are controlled by the RGMII/RTBI core, which has two MMIO registers associated with it:

RGMII0_FER (RGMII Function Enable Register)

RGMII0_SSR (RGMII Speed Select Register)

These allow the selection between GMII and TBI modes, and the choice of either the full versions of these modes (GMII/TBI) or the reduced versions (RGMII/RTBI).

The current EMAC to PHY Bridge Registers (ZMII0_*) have new fields added to them to support SMII mode for the new ethernet controllers2 and 3.

TCP/IP Acceleration

Each of the 10/100/gigabit ethernet controllers has TCP/IP acceleration hardware (TAH) associated with it. This acceleration hardware performs common functions which are normally done in a software TCP/IP stack, thus relieving software and the 440 core of the burden and increasing performance.

The TCP/IP acceleration hardware checks the checksum on incoming TCP/IP packets, and generates a checksum for outgoing TCP/IP packets. It can also segment large outgoing packets into smaller ones.

For transmitted packets, the decision on whether to have the acceleration hardware generate checksums and segment packets is made on a packet-by-packet basis. This is achieved by setting the appropriate hardware acceleration flags in the MAL transmit control quadword. The size that packets should be segmented into is controlled by one of six Segment Size Registers, TAHx_SSR0 -TAHx_SSR5. Which register is used is determined on a packet-by-packet basis, again using the MAL transmit control quadword. Transmit status is returned in the Transmit Status Register, TAHx_TSR.

For received packets, hardware checksum verification is turned on globally by a bit in the acceleration mode register, TAHx_MR.

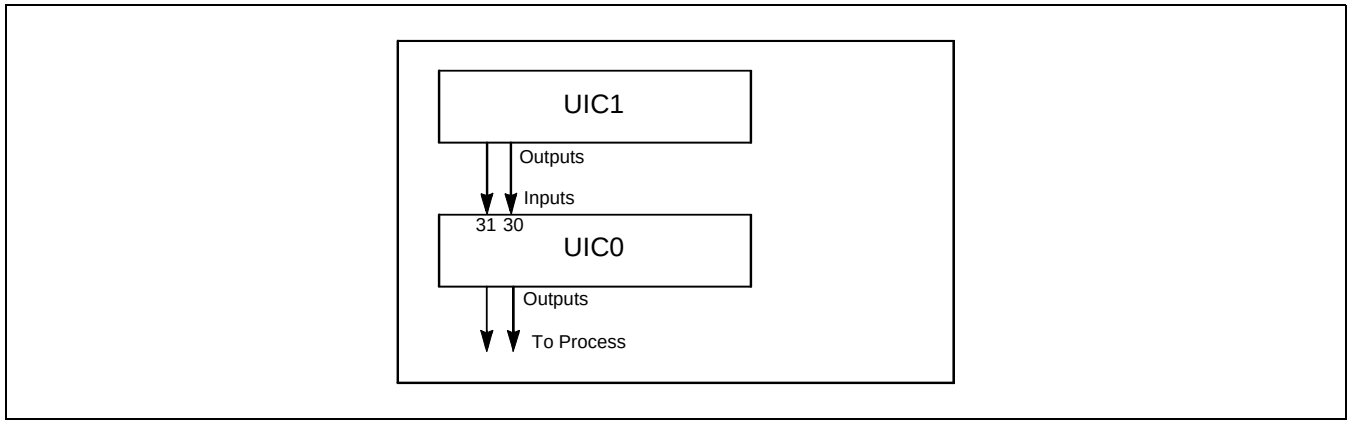
Typically, activities involving the TCP/IP acceleration hardware are performed in an operating system's TCP/IP stack, rather than in application code or a device driver.

Use of this feature is optional -it is possible to use the 10/100/gigabit controllers either with or without the TCP/IP acceleration. This allows legacy TCP/IP stacks or other protocol stacks to still work while providing increased performance for TCP/IP stacks which choose to exploit the acceleration.

Interrupt Controllers

On the 440GP there are two universal interrupt controllers (UICs), known as UIC0 and UIC1. Each has32 inputs which come from either internal chip components or external devices. Each UIC has2 outputs: one critical (high priority) and one non-critical. The outputs from UIC1 are cascaded into two of the inputs of UIC0 (30 and 31), then UIC0's outputs are connected to the processor, see Figure 1.

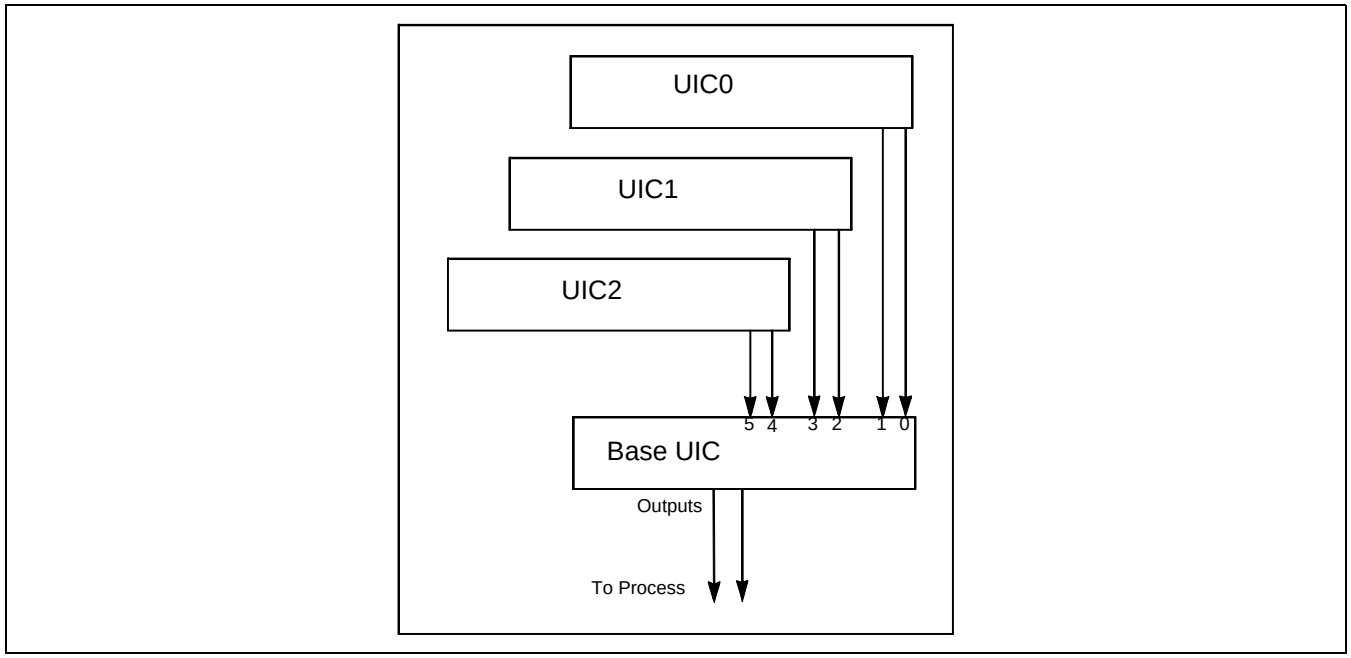
Figure 1: Interrupts in 440GP-compatible mode



On the 440GX there are additional cores present, which have their own interrupts. In order to provide backwards compatibility with 440GP, a switch is provided which will allow the original two interrupt controllers to operate in 440GP-compatible mode. In this mode, all of the interrupts which were present in the 440GP are present in the same locations and no modification to software is needed. However, none of the interrupts from the new 440GX components are accessible in this mode (see "General Purpose Timers Changes" on page 12 for an exception to this).

The switch is implemented as a bit in the new SDR0_MFR (Miscellaneous Function Register). In order to take advantage of the new components present in the 440GX, the switch must be set to “440GX-mode”. In this mode, a new UIC (UIC2) is present. The inputs to UIC2 are from the new interrupts available in the 440GX. The way that the UICs are connected is also changed, see Figure 2. Instead of UIC1 cascading its outputs into UIC0, the three UICs become peers, and each of them feeds its outputs into a “base UIC”. Thus the critical and non-critical interrupts from UIC0 are fed as inputs to the base UIC pins0 and 1. Similarly, the outputs from UIC1 go to input pins2 and 3 of the base UIC, and those from UIC2 go to pins4 and 5. All other input pins on the base UIC are reserved. The outputs of the base UIC drive the processor.

Figure 2: Interrupts in 440GXmode



The interrupt handling code for this new 440GX configuration must take into account the new structure. Note that in this mode, UIC0 inputs30 and 31 are not used, and so must be masked off (these used to be the cascade inputs from UIC1 in 440GP compatible mode). Whenever an interrupt is received, the interrupt dispatch code should first check the base UIC to determine which of UIC0, UIC1 or UIC2 caused the interrupt, then check that UIC to determine the exact interrupt signal to be serviced. When clearing an interrupt, an interrupt handler should first clear the interrupt in the UIC to which it is attached (UIC0, UIC1 or UIC2), and then clear the corresponding cascade bit in the base UIC.

Messaging Unit

The Messaging Unit (IMU), designed to I2O specification, allows messages to be transferred between the 440 core and a PCI(X) device.

The IMU supports three messaging methods:

- Message Registers
- Doorbell Registers
- Circular Queues

These functions are implemented by a set of memory-mapped registers. There are inbound and outbound status registers to indicate register status, and several interrupts associated with the IMU. Interrupts may be masked by the use of mask registers. In the following descriptions, the phrase “may generate an interrupt” means that an interrupt will be generated unless it has been masked off.

Message Registers

There are 2 inbound message registers. These can be written to by a PCI device. This will set a bit in the inbound status register and may generate an interrupt to the CPU, which can then read the registers.

Similarly, the two outbound message registers can be written by the CPU, which sets a bit in the outbound status registers and may generate an interrupt to the PCI device.

Doorbell Registers

The inbound doorbell register can be written to by a PCI device. When any bit has a ‘1’ written to it, it will set a bit in the inbound status register, and may generate a CPU interrupt. Two distinct interrupts are supported -the IRQ Doorbell interrupt is associated with one bit in the register; the Inbound Doorbell Interrupt is associated with the other 31 bits.

Similarly, the outbound doorbell register can be written by the CPU, and sets a status bit in the outbound status register when a ‘1’ is written to any bit. It may generate an interrupt to PCI. Only one outbound interrupt is provided, the Outbound Doorbell Interrupt, associated with all 32 bits in the register.

Circular Queues

Two inbound and two outbound circular queues are provided. There is one ‘Free’ and one ‘Post’ queue for each direction. Each queue consists of a head and a tail pointer register.

Each queue element is1 word long. The number of queue elements in each queue may be selected, from 4k to 64k in power-of-2 increments. The queue elements all reside in a programmable 1MB region.

The PCI device places pointers to free data areas in its Free queue, which may be retrieved by the CPU. The CPU places data in the data area, and put sit on the Post queue, which automatically sets a status bit and may generate an interrupt to allow the PCI device to retrieve the data.

A similar series of events allows the CPU to place free data area son its Free queue and have the PCI device retrieve them, update them, and place them on the Post queue to be received by the CPU.

PCI-X Changes

The PCI core is PCI 2.3 compliant. Some PCI-X registers have changed bit fields in them and additional registers have been added.

Inbound PIM/BAR windows can be larger than the previous 4GB, which has resulted in additional size/attribute high registers. The new registers are PCIX0_PIM0SAH, and PCIX0_PIM2SAH (PCI-X PIM Size/Attribute High registers), consisting of a single 32-bit field which extends the available PIM size to 2^{64} bytes. These are logical extensions of the existing SIZE field in the PCIX0_PIMxSA registers.

Four VPD (Vital Product Data) registers are added:

- PCI-X VPD Capability Identifier Register
- PCI-X VPD Next Item Pointer register
- PCI-X VPD Address Register
- •PCI-X VPD Data Register

An additional PCI-X VPD interrupt is added, and appears in UIC2.

Details of the use of these registers and interrupt are outside the scope of this document -consult the PCI 2.3 specifications for more details.

Support is added for unexpected split completion errors which may optionally be enabled by setting the new bit, USEE (Unexpected Split completion Error Enable) in the PCIX0_ERREN (PCI-X Error Enable) register. The error is reflected in PCIX0_PCIXSTS[USC] (PCI-X Error Status-Unexpected Split Completion), which was previously documented to always return 0.

Additionally, six outbound split discard timers are added. Split discard timers are enabled by setting PCIX0_ERREN[SDTE] (outbound Split Discard Timer Enable). When a split response is received a timer is started. If a partial split completion is not received before the timer expires, a new bit, SDTE (split discard timer expired) is set in the PCIX0_ERRSTS (PCI-X Error status) register.

Packet/Code Store SRAM

The on-chip static RAM (SRAM) is increased from 8K on the 440GP to 256K bytes on the 440GX. The SRAM is divided into four banks of 64KB each, so four bank configuration registers (SRAM0_SBxCR) are used for access. On the 440GP, only one or two banks were available. Any 440GP code wishing to make use of the extra memory should be modified appropriately (e.g. update TLB size).

Note that operating systems may choose to take advantage of this increased size. One use may be for ethernet descriptors (hence the term “packet store”), or to place frequently-used code such as exception handlers here.

The SRAM has parity protection, the same as the SRAM on the 440GP had.

The memory used as SRAM may be configured as an L2 cache instead of SRAM -see the following section.

Level 2 Cache

The level 2 cache (L2) controller uses the 256K of SRAM present on the chip. The SRAM may either be wholly used as conventional SRAM, as described in the section *Packet/Code Store SRAM*, or may be wholly used as an L2 cache. Its usage cannot be shared between the two.

The L2 cache is 256Kbytes in size, 4 way set-associative, each way being 64K. It is physically indexed and tagged. The line size is 32 bytes, resulting in 2048 sets each having 4 ways. It is a unified instruction and data cache, although it may optionally be used completely as an instruction cache or completely as a data cache. The data cache operates in write-through mode only. The L2 maintains coherency to main memory by snooping the PLB. It incorporates optional parity protection.

The processor cache instructions (**iccci**, **dcbi** etc.) do not affect the L2 cache, so separate commands are provided through a set of DCRs. However, as a result of coherency to memory and use of only physical addresses, many of the operations performed on the L1 cache do not need to be performed on the L2. The following paragraphs discuss the operations which may be performed on the L1 cache, and contrast their potential uses on the L2 cache.

Initialization

The L2 needs to be initialized by invalidating and clearing all lines. A simple software command sequence to perform this task should be executed during chip initialization. This initialization also allows the selection between SRAM-only mode and L2 cache, using the L20_CFG[L2M] (L2 cache configuration register, L2 Mode) bit.

I-cache flash invalidate

Although the L1 I-cache may be flash invalidated, using the **iccci** instruction, there is no need to provide this function for the L2. In looking at the circumstances where the L1 is completely invalidated, it can be shown that these circumstances either do not apply to the L2, or alternate methods may be used. The L1 I-cache may be flash invalidated under three circumstances:

The first is in chip initialization, which for the L2 is covered above.

The second is for parity errors, which is done automatically by the L2, as described below.

The third is in a multi-tasking operating system when a task exits. Due to the virtual tagging of the L1 I-cache, all lines associated with the terminated task must be invalidated so that they are not available to a new task which has the same PID as the terminating one. The quickest way to achieve this is to invalidate the entire cache. However, because the L2 is only addressed by physical addresses, and protection is enforced within the processor MMU, there is no need to invalidate the instructions in the L2 cache after a task has terminated.

Line invalidate

An L1 cache line may be invalidated using the icb**i** or dcb**i** instructions. This is typically done to enforce coherency with main memory, for example due to self-modifying code or memory altered by something other than the processor (for example DMA). As the L2 has hardware coherency with main memory, there is no need to invalidate a line using software.

Data Line flush.

The **dcbf** or **dcbst** instructions are used to write dirty data from the L1 D-cache (in write-back mode) to main memory. The L2 always operates in write-through mode, so there is never any dirty data in memory, and so no L2 equivalent action is needed.

Data Line zero

The **dcbz** instruction is used to write zeros to a line in the L1 cache. When this line is eventually written to main memory it will proceed directly to memory (due to the L2'swrite-through policy). If a previous stale copy of the data was in the L2 it will be replaced by the current data. Therefore there is no need for an L2 equivalent of the L1 **dcbz** instruction.

D-cache flash invalidate

The only practical uses of invalidating the entire L1 D-cache, with the **dccci** instruction, is for initialization or a parity error. The L2 can be initialized as described above, and parity errors are handled automatically as described below.

Parity protection

The L2 data lines and tags have optional parity protection. When a parity error is detected, an interrupt may be generated, and a mechanism is available via the L2 DCRs to determine the address or cache line which caused the error. However, the L2 cache controller will automatically handle parity errors, as described below, so in normal use no action is required and the interrupt need not be taken. Some environments may use this interface to log parity errors, in which case when the interrupt is generated the interrupt handler may log the affected addresses and should reset the error status bits before exiting.

For cache data, parity is checked when data is requested by the CPU. If a parity error is detected, the line in the cache is immediately invalidated and the processor is informed that an error occurred when reading the data. Unlike the L1 cache parity, which is not checked until the instruction is inside the processor pipeline, the L2 parity is checked as the data is read from the L2, which allows the transaction to be aborted before data with bad parity is accepted by the processor. This allows the processor to re-request the data. Since the line with bad parity has been invalidated in the L2, the request will miss there and proceed directly to main memory.

For the tags, the parity bit is part of the tag compare. Therefore a parity error in the tag will always result in a tag miscompare. This will be treated as a miss and the line will be fetched from memory and placed in the cache with a correct tag. Eventually the way with bad tag parity will be flushed from the cache as a consequence of the least recently used (LRU) algorithm.

Executing L2 commands

Even though not needed in typical usage, the L2 cache contains several DCRs which can be used to perform actions on the L2, for example to invalidate a line or a way, or to clear the trap address registers which are used to record the location of parity errors. The sequence needed to execute a command to the L2 is as follows. Any parameters for the command are loaded into the appropriate register. For example, to invalidate a cache line for a particular address, the address is loaded into the L2C0_ADDR (L2 address) register. Then the appropriate bit in the L2C0_CMD (L2 command) register is set, which starts the command executing. In order to determine when the command has completed, software must poll the L2C0_STAT[CC] (L2 status register Command Complete) bit until it is set to ‘1’. At this point the command is complete.

General Purpose Timers Changes

The General Purpose Timers (GPT) core has increased the number of compare timers from 5 in the 440GP to 7 in the 440GX. This results in two additional registers for compare timers5 and 6 (GPT0_COMP5, GPT0_COMP6) and associated Compare Mask registers (GPT0_MASK5, GPT0_MASK6). The associated interrupts have also been added to the UIC0 set of registers, in bit positions which were previously reserved.

DDR-SDRAM Changes

In the DDR-SDRAM timing register 0 (SDRAM0_TR0), some previously reserved bits now have meaning. Bit 24, TWTR, configures the write-to-read command. Bits25:26, RFTA, the refresh to activate field adder, supplement the values in the adjacent SDRA field.

Compatibility Mode

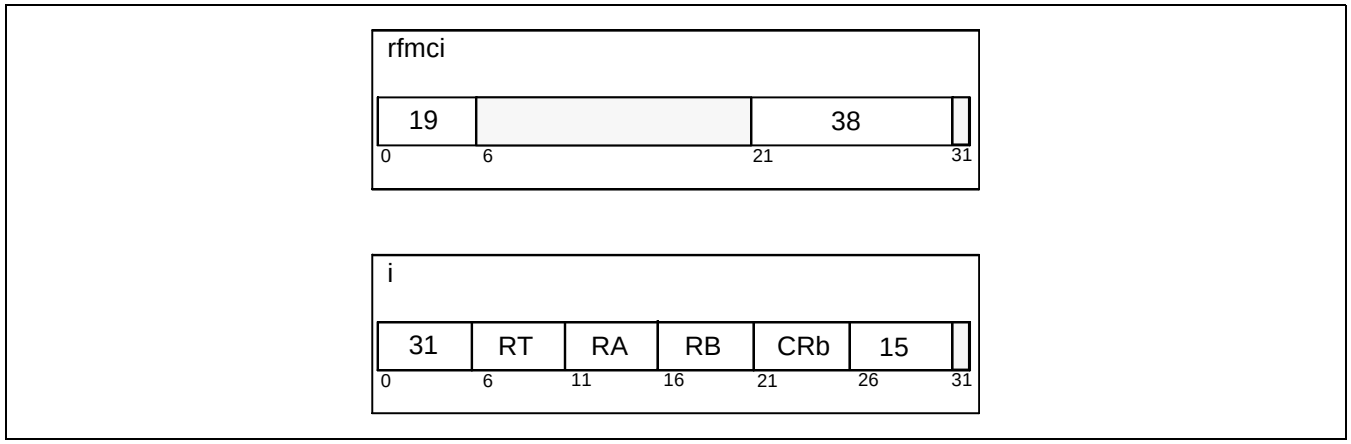
Although much backward compatibility to the 440GP is retained when running the 440GX in “compatibility mode”, that is, not exploiting any of the new features on the chip, there are nonetheless some changes which will be needed to be made to software to allow it to run on the 440GX. These features are mainly contained in the areas of boot code and operating system services, and so should have minimal to no effect on applications.

The minimal areas which are affected include the following:

- Machine check handlers - should be rewritten to use the new MCSRR0 and MCSRR1 registers and exit with the **rfmci** instruction (previously was the CSRR0, CSRR1 registers and **rfci** instruction).
- Boot code - changes are necessary to accommodate the updated clocking and chip initialization register structure.
- Ethernet device drivers - review changes for EMAC4 to see which changes are applicable. This will depend on specific details of each driver, for example whether it implemented support for two transmit channels.

Change Details

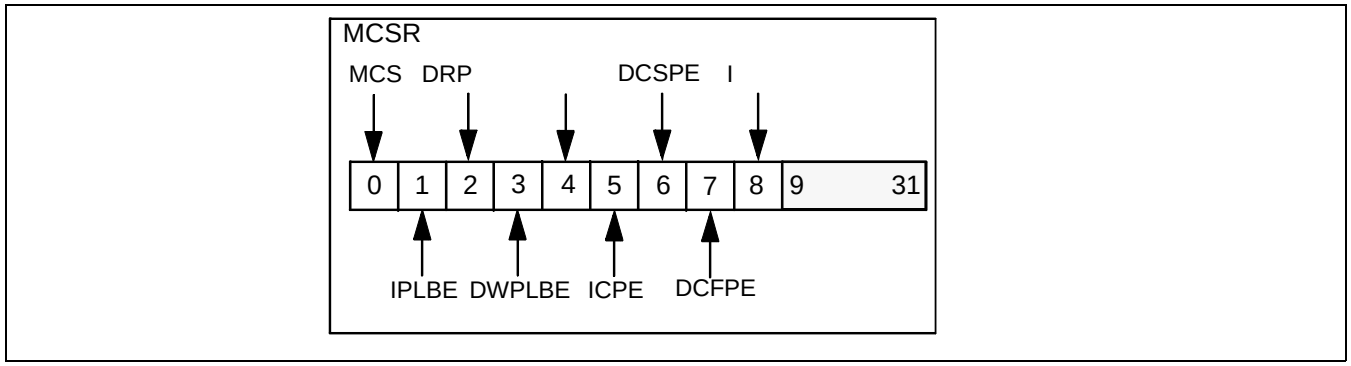
New Instructions



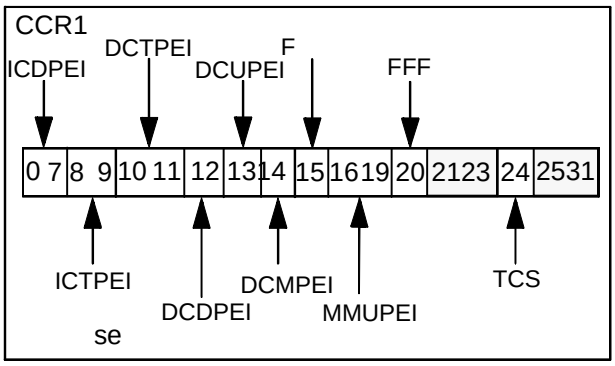
New Registers

SPRs

```
MCSRR0      (SPRN=0x23a)
MCSRR1      (SPRN=0x23b)
MCSR        (SPRN=0x23c)
```



```
MCS         Machine Check Summary
IPLBE       Instruction PLB Error
DRPLBE      Data Read PLB Error
DWPLBE      Data Write PLB Error
TLBE        TLB Error
ICPE        Instruction Cache Parity Error
DCSPE       Data Cache Search Parity Error
DCFPE       Data Cache Flush Parity Error
IMCE        Imprecise Machine Check Exception
CCR1 (SPRN=0x378)
```




```
ICDPEI      I-cache Data Parity Error Insert
ICTPEI      I-cache Tag Parity Error Insert
DCTPEI      D-cache Tag Parity Error Insert
DCDPEI      D-cache Data Parity Error Insert
DCUPEI      D-cache U-bit Parity Error Insert
DCMPEI      D-cache Modified bit Parity Error Insert
FCOM        Force Cache Operation Miss
MMUPEI      MMU Parity Error Insert
FFF         Force Full-line Flush
TCS         TimerClock Select
```

DCRs

The following DCRsare used to access the indirectly-addressed CPR and SDR sets of registers.

```
CPR0_CFGADDR      (DCRN=0x00C)
CPR0_CFGDATA      (DCRN=0x00D)
SDR0_CFGADDR      (DCRN=0x00E)
SDR0_CFGDATA      (DCRN=0x00F)
```

The indirectly addressed CPR registers:

```
CPR0_CLKUPD       (Offset=0x0020)
CPR0_PLLC         (Offset=0x0040)
CPR0_PLLD         (Offset=0x0060)
CPR0_PRIMAD       (Offset=0x0080)
CPR0_PRIMBD       (Offset=0x00A0)
CPR0_OPBD         (Offset=0x00C0)
CPR0_PERD         (Offset=0x00E0)
CPR0_MALD         (Offset=0x0100)
CPR0_ICFG         (Offset=0x0140)
```

The indirectly addressed SDR registers:

```
SDR0_SDSTP0       (Offset=0x0020)
SDR0_SDSTP1       (Offset=0x0021)
SDR0_PINSTP       (Offset=0x0040)
SDR0_SDCS         (Offset=0x0060)
SDR0_ECID0        (Offset=0x0080)
SDR0_ECID1        (Offset=0x0081)
SDR0_ECID2        (Offset=0x0082)
SDR0_JTAG         (Offset=0x00C0)
SDR0_DDRDL        (Offset=0x00E0)
SDR0_EBC          (Offset=0x0100)
SDR0_UART0        (Offset=0x0120)
SDR0_UART1        (Offset=0x0121)
SDR0_CP440        (Offset=0x0180)
SDR0_XCR          (Offset=0x01C0)
SDR0_XPLLC        (Offset=0x01C1)
SDR0_XPLLD        (Offset=0x01C2)
SDR0_SRST         (Offset=0x0200)
SDR0_SLPIPE       (Offset=0x0220)
SDR0_AMP          (Offset=0x0240)
SDR0_MIRQ0        (Offset=0x0260)
SDR0_MIRQ1        (Offset=0x0261)
SDR0_MALTBL       (Offset=0x0280)
SDR0_MALRBL       (Offset=0x02A0)
SDR0_MALTBS       (Offset=0x02C0)
SDR0_MALRBS       (Offset=0x02E0)
SDR0_CUST0        (Offset=0x4000)
SDR0_SDSTP2       (Offset=0x4001)
SDR0_CUST1        (Offset=0x4002)
SDR0_SDSTP3       (Offset=0x4003)
SDR0_PFC0         (Offset=0x4100)
SDR0_PFC1         (Offset=0x4101)
SDR0_PLBTR        (Offset=0x4200)
```

```
SDR0_MFR          (Offset=0x4300)
```

DCRs removed: all CPC0 registers, replaced by the above-mentioned CPR and SDR sets, with the exception of the power management registers which have new DCR names and addresses:

```
CPM0_ER           (DCRN=0x0B0)
CPM0_FR           (DCRN=0x0B1)
CPM0_SR           (DCRN=0x0B2)
```

UIC register sets added for UIC2 and Base UIC (UICB0). These registers have similar functionality to the UIC0 and UIC1 sets.

```
UICB0_SR   (DCRN=0x200)
UICB0_ER   (DCRN=0x202)
UICB0_CR   (DCRN=0x203)
UICB0_PR   (DCRN=0x204)
UICB0_TR   (DCRN=0x205)
UICB0_MSR  (DCRN=0x206)
UICB0_VR   (DCRN=0x207)
UICB0_VCR  (DCRN=0x208)
UIC2_SR    (DCRN=0x210)
UIC2_ER    (DCRN=0x212)
UIC2_CR    (DCRN=0x213)
UIC2_PR    (DCRN=0x214)
UIC2_TR    (DCRN=0x215)
UIC2_MSR   (DCRN=0x216)
UIC2_VR    (DCRN=0x217)
UIC2_VCR   (DCRN=0x218)
```

Level 2 cache registers:

```
L2C0_CFG   (DCRN=0x030)
L2C0_CMD   (DCRN=0x031)
L2C0_ADDR  (DCRN=0x032)
L2C0_DATA  (DCRN=0x033)
L2C0_STAT  (DCRN=0x034)
L2C0_CVER  (DCRN=0x035)
L2C0_SNP0  (DCRN=0x036)
L2C0_SNP1  (DCRN=0x037)
```

MMIO

The following memory-mapped I/O registers are added:

An additional set of ethernet registers for the two 10/100/gigabit ethernets:

```
EMAC2_* at 0x1 4000 0C00
EMAC3_* at 0x1 4000 0E00
```

An additional ethernet register is added for each ethernet core:

EMACx_IPCSCFG, offset 0x70, EMAC Internal PCS Configuration Register

The two sets of TCP/IP acceleration hardware registers are added:

```
TAH0_MR           at 0x1 4000 0B60
TAH0_SSR0         at 0x1 4000 0B64
TAH0_SSR1         at 0x1 4000 0B68
TAH0_SSR2         at 0x1 4000 0B6C
TAH0_SSR3         at 0x1 4000 0B70
TAH0_SSR4         at 0x1 4000 0B74
TAH0_SSR5         at 0x1 4000 0B78
TAH0_TSR          at 0x1 4000 0B7C
TAH1_MR           at 0x1 4000 0D60
TAH1_SSR0         at 0x1 4000 0D64
TAH1_SSR1         at 0x1 4000 0D68
```

```
TAH1_SSR2          at 0x1 4000 0D6C
TAH1_SSR3          at 0x1 4000 0D70
TAH1_SSR4          at 0x1 4000 0D74
TAH1_SSR5          at 0x1 4000 0D78
TAH1_TSR           at 0x1 4000 0D7C
```

RGMII registers are added:

```
RGMII0_FER         at 0x1 4000 0790
RGMII0_SSR         at 0x1 4000 0794
```

PCI-X registers are added:

```
PCIX0_PIM0SAH      at 0x2 0EC8 00F8
PCIX0_PIM2SAH      at 0x2 0EC8 00FC
```

IMU registers are added.

Many IMU registers can be accessed at two addresses. One is for access from the local CPU, and one is for access from the remote PCI device.

Mnemonic	Local Access	Remote Access
IMU0_IMR0	0 FFFF 0090	0 FFFF 0010
IMU0_IMR1	0 FFFF 0094	0 FFFF 0014
IMU0_OMR0	0 FFFF 0098	0 FFFF 0018
IMU0_OMR1	0 FFFF 009C	0 FFFF 001C
IMU0_IDR	0 FFFF 00A0	0 FFFF 0020
IMU0_IISR	0 FFFF 00A4	0 FFFF 0024
IMU0_IIMR	0 FFFF 00A8	0 FFFF 0028
IMU0_ODR	0 FFFF 00AC	0 FFFF 002C
IMU0_OISR	0 FFFF 00B0	0 FFFF 0030
IMU0_OIMR	0 FFFF 00B4	0 FFFF 0034
IMU0_IQPR		0 FFFF 0040
IMU0_OQPR		0 FFFF 0044
IMU0_CFG	0 FFFF 00D0	0 FFFF 0050
IMU0_QBAHR	0 FFFF 00D4	0 FFFF 0054
IMU0_QBALR	0 FFFF 00D8	0 FFFF 0058
IMU0_IFHPR	0 FFFF 00E0	0 FFFF 0060
IMU0_IFTPR	0 FFFF 00E4	0 FFFF 0064
IMU0_IPHPR	0 FFFF 00E8	0 FFFF 0068
IMU0_IPTPR	0 FFFF 00EC	0 FFFF 006C
IMU0_OFHPR	0 FFFF 00F0	0 FFFF 0070
IMU0_OFTPR	0 FFFF 00F4	0 FFFF 0074
IMU0_OPHPR	0 FFFF 00F8	0 FFFF 0078

Other IMU registers have only a single address:

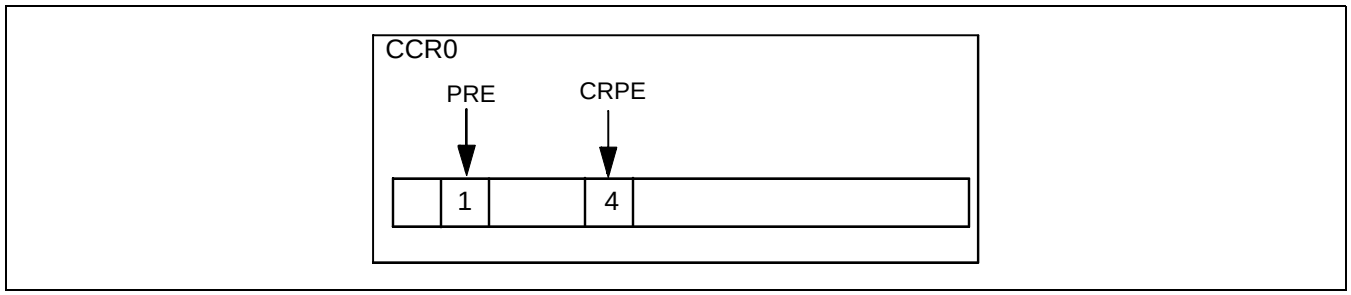
```
IMU0_BESR          at 0x0 FFFF 0100
IMU0_BEMR          at 0x0 FFFF 0104
IMU0_BEAR          at 0x0 FFFF 0108
IMU0_MIRQ          at 0x0 FFFF 010C
IMU0_PLB           at 0x0 FFFF 0110
IMU0_SLP           at 0x0 FFFF 0130
IMU0_REVID         at 0x0 FFFF 0134
IMU0_SR            at 0x0 FFFF 0138
```

New GPT registers are added:

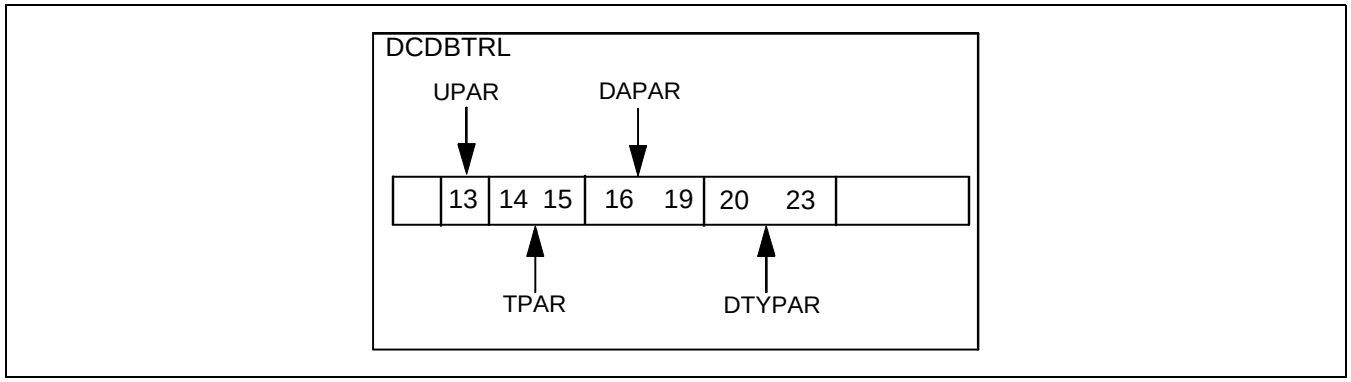
```
GPT0_COMP5         at 0x1 4000 0A94
GPT0_COMP6         at 0x1 4000 0A98
GPT0_MASK5         at 0x1 4000 0AD4
GPT0_MASK6         at 0x1 4000 0AD8
```

New Register Fields

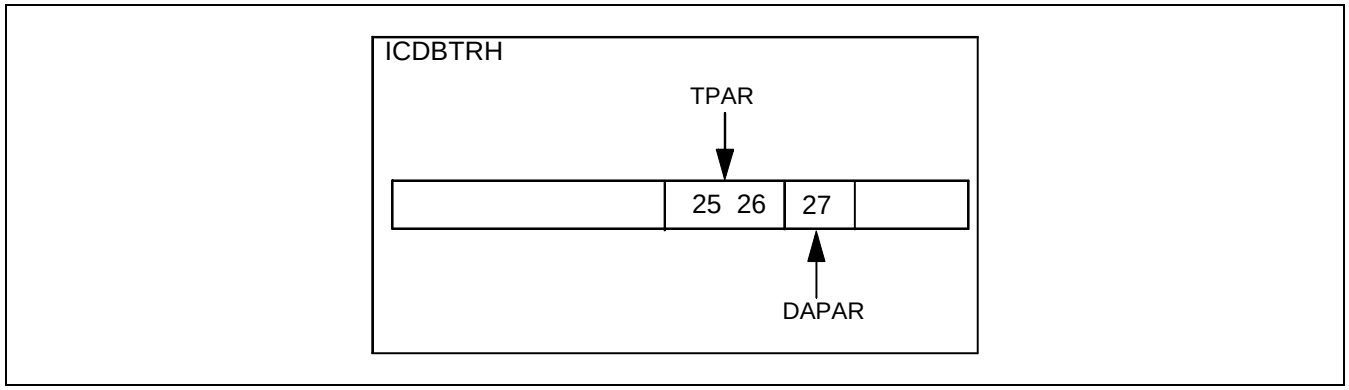
The following registers have new fields added to them. Existing fields remain as in 440GP and are not shown here.



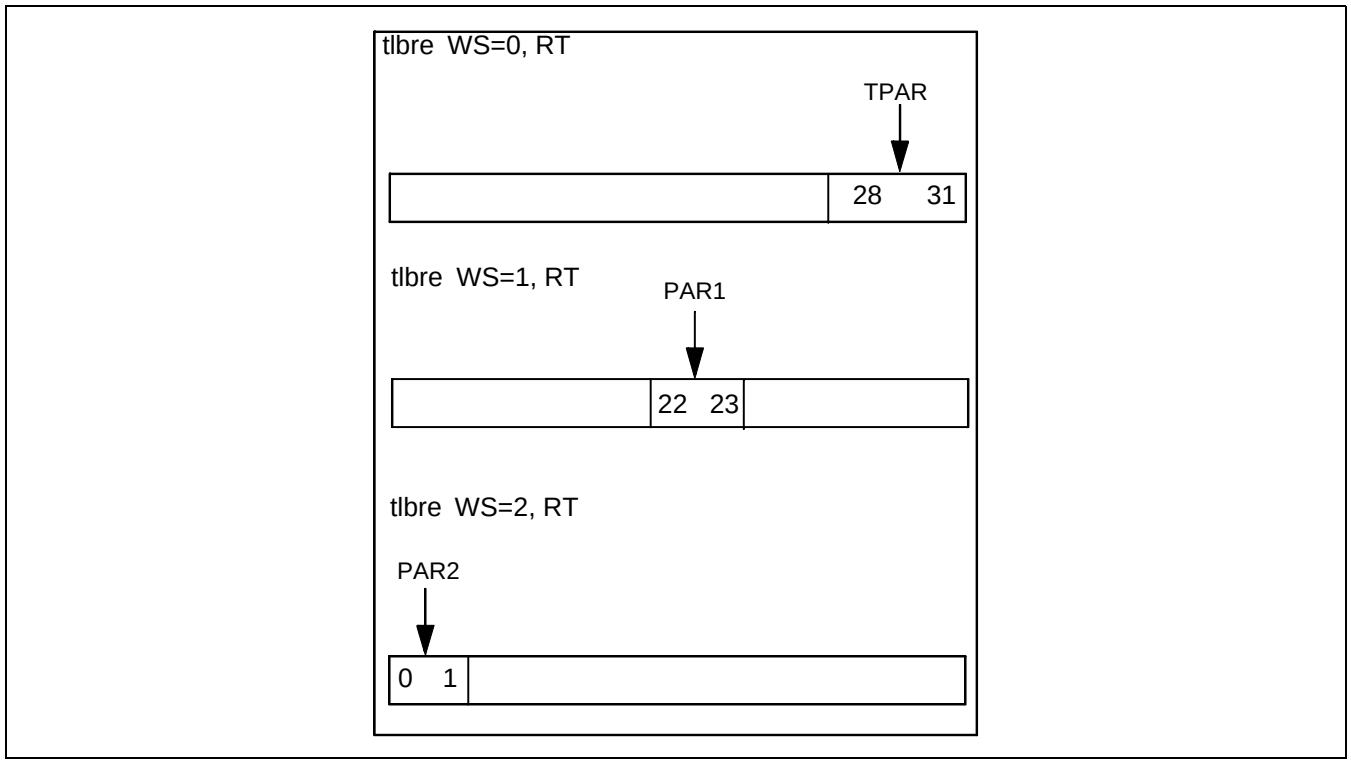

```
PRE Parity Recoverability Enable (for D-cache)
CRPE Cache Read Parity Enable
```



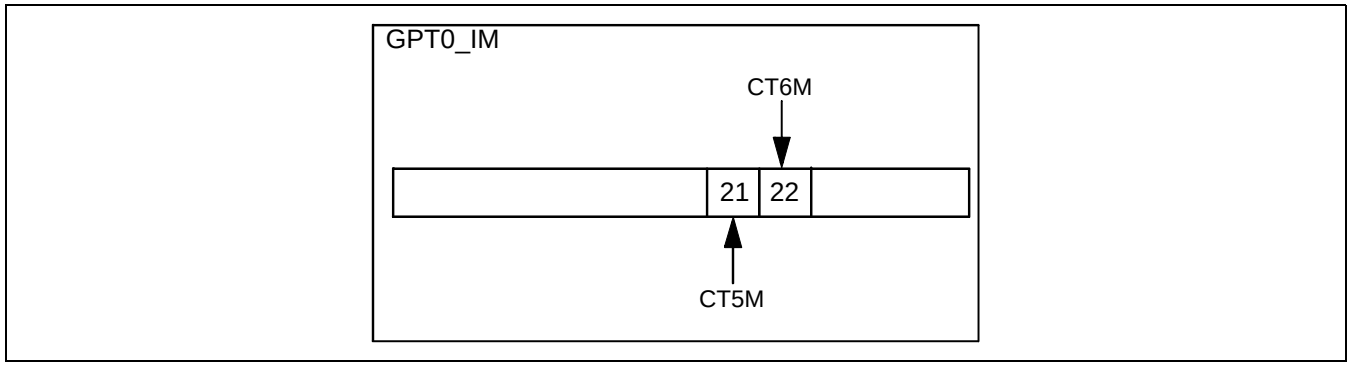

```
UPAR       U-field parity
TPAR       Tag parity
DAPAR      Data Parity
DTYPAR     Dirty bits parity
```



TPAR Tag Parity
DAPAR Data Parity

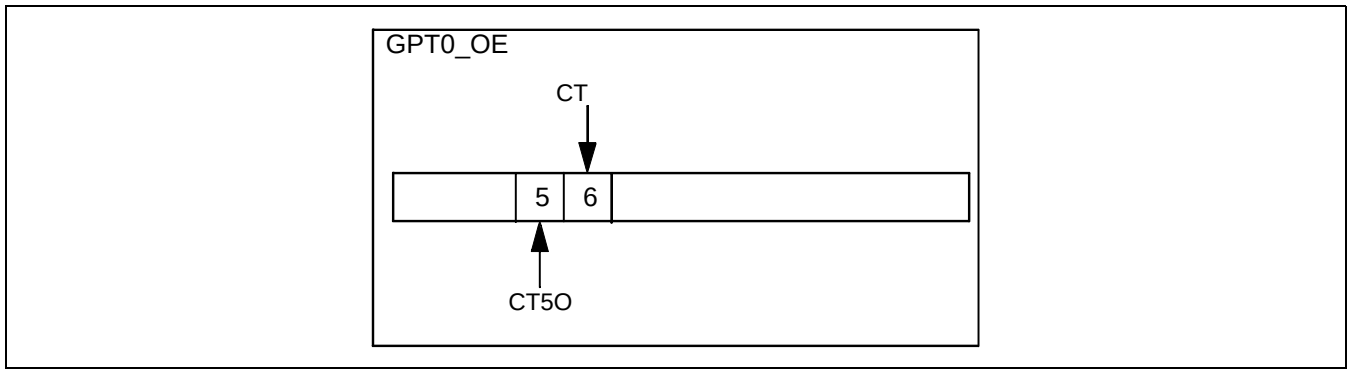


TPAR Tag Parity
PAR1 Parity for word 1
PAR2 Parity for word 2

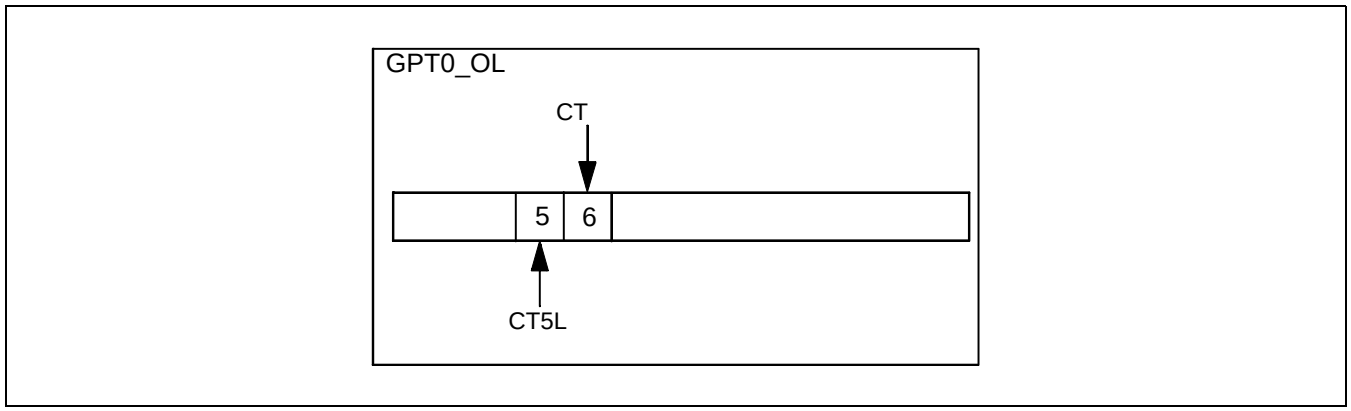


CT5M Compare Timer 5 Interrupt Mask

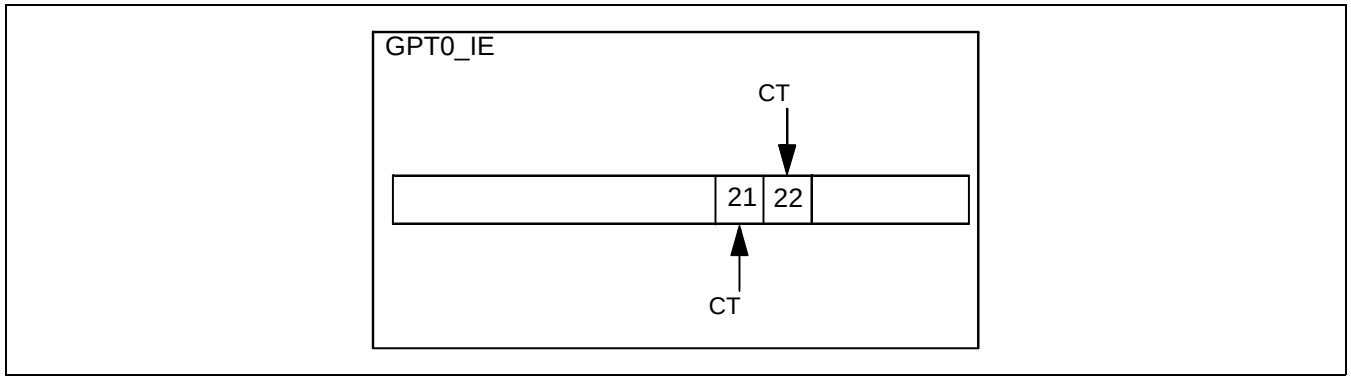
CT6M Compare Timer 6 Interrupt Mask



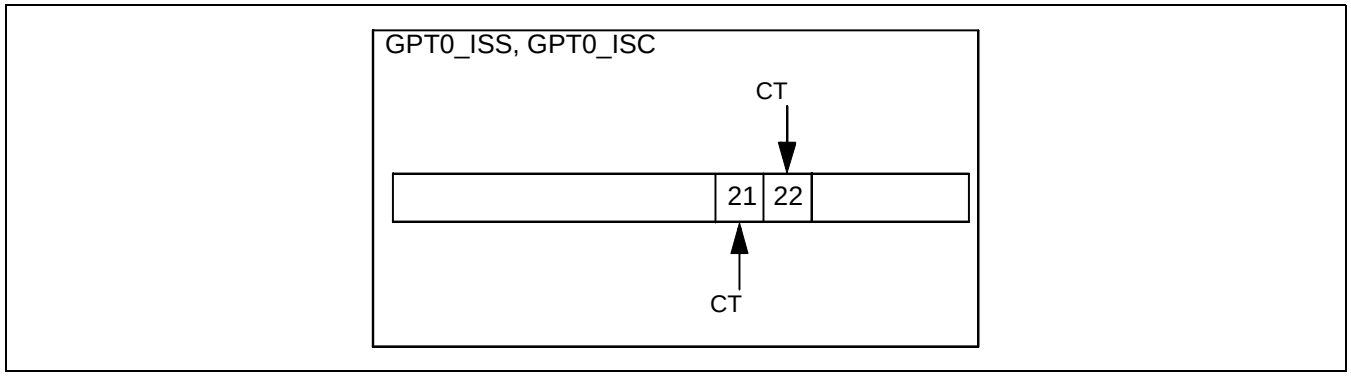
CT5O Compare Timer 5 Output
CT6O Compare Timer 6 Output



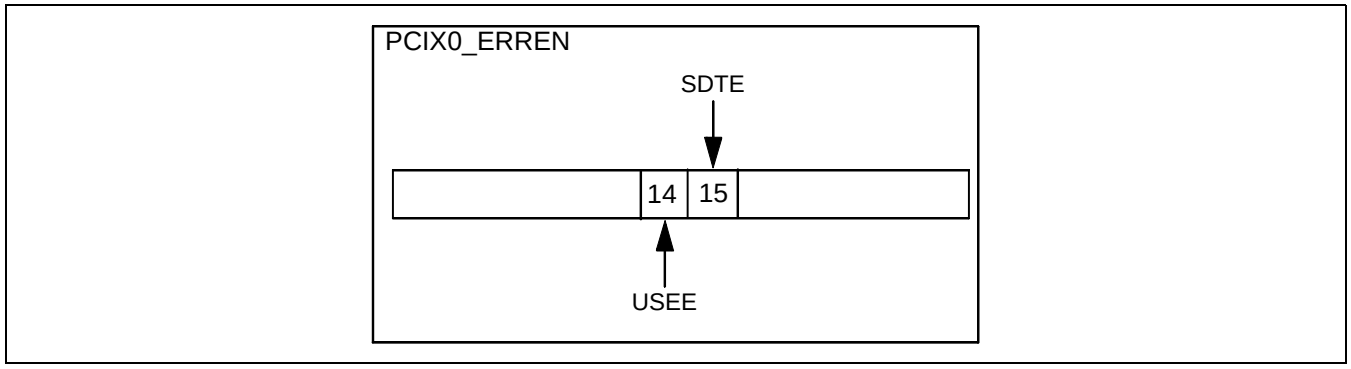
CT5L Compare Timer 5 Output Level
CT6L Compare Timer 6 Output Level



CT5I Compare Timer 5 Interrupt Enable
CT6I Compare Timer 6 Interrupt Enable

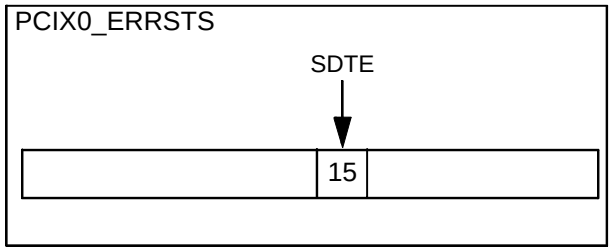


CT5S Compare Timer 5 Interrupt Status
CT6S Compare Timer 6 Interrupt Status

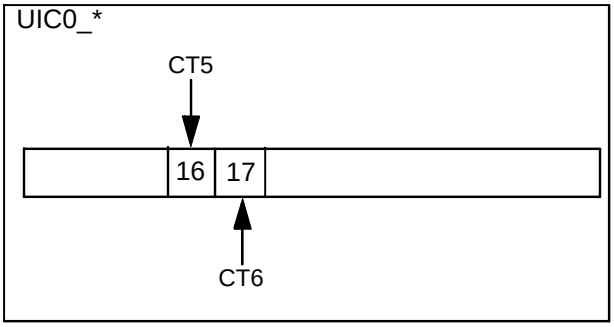


USEE Unexpected Split Completion Error Enable

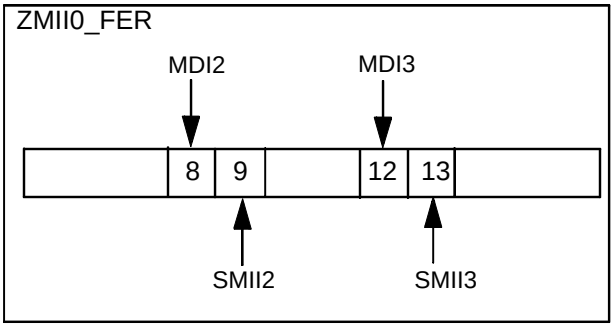
SDTE Outbound Split Discard Timer Enable



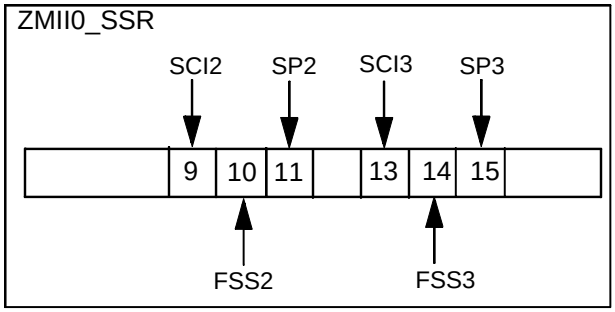
SDTE Split Discard Timer Expired



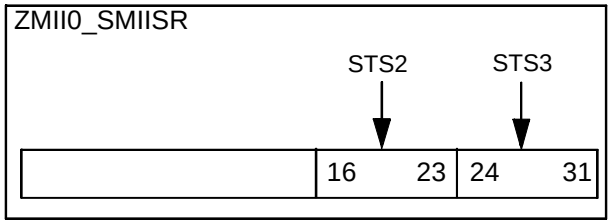
CT5 GPT Compare Timer 5 Interrupt
CT6 GPT Compare Timer 6 Interrupt



MDI2 EMAC2 MDI Enable
SMII2 EMAC2 SMII Enable
MDI3 EMAC3 MDI Enable
SMII3 EMAC3 SMII Enable

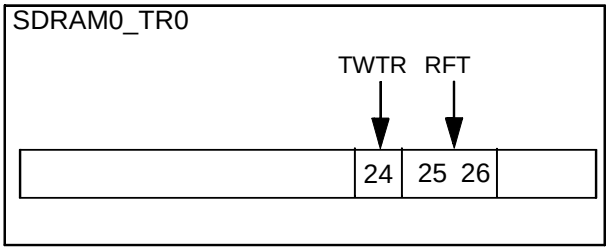


SCI2 EMAC2 Suppress Collision Indication
FSS2 EMAC2 Force Speed Selection
SP2 EMAC2 Speed Selection
SCI3 EMAC3 Suppress Collision Indication
FSS3 EMAC3 Force Speed Selection
SP3 EMAC3 Speed Selection



STS2 EMAC2 SMII Status bits
STS3 EMAC3 SMII Status bits

These status bits correspond to the status bits for EMAC0 and EMAC1 in the rest of the register.



TWTR Write to Read Command
RFTA Refresh to Activate Command

Document Revision History

Revision	Date	Description
v1.01	1/22/08	Converted layout to AMCC format.

Applied Micro Circuits Corporation
6310 Sequence Dr., San Diego, CA 92121
Main Phone: (858) 450-9333 — Technical Support Phone: (858) 535-6517 — (800) 840-6055
http://www.amcc.com (support@amcc.com)