X86 Multiprocessing Basics

Application Note

by Sherry L. Silver

Revision Summary: This is the initial release of this Application Note.

Introduction

This purpose of this paper is to introduce the reader to multiprocessing systems. The basic features of a multiprocessing system are defined, design issues discussed, and various system configurations evaluated. This paper provides the reader with an understanding of X86 multiprocessing system solutions but not with detailed design information.

General Concepts

The two fundamental elements of a multiprocessing system are its hardware configuration and its controlling software. The basic hardware requirement for a multiprocessing configuration is a system containing two or more microprocessors and some logic to control communication with and between the processors. For the controlling software, the Basic Input/Output System (BIOS) and the operating system chosen must have the ability to support multiple microprocessors.

Multiprocessing systems are either symmetric or asymmetric, where the symmetry is defined with respect to memory, input/output (I/O), and interrupts. In a fully symmetric multiprocessing system (SMP), the microprocessors are functionally identical, have equal status, and can communicate with every other processor. The operating system may use this symmetry to dynamically assign program tasks to each processor based on the current needs of the application. Essentially, any code segment can run on any microprocessor at any time after system initialization, or boot-up.

The one exception to this symmetry is during boot-up when either the hardware alone or the hardware along with the BIOS identifies one of the processors as the boot processor. The boot processor then works to enable the other processors. After the system is initialized, the operating system switches to symmetric mode, and the boot processor functions in the same manner as the other processors.

In an asymmetric system, the microprocessors may be different and program tasks are strictly divided by type among processors. Dynamic load balancing is not performed. If the specific task that one processor handles is not needed, the processor will be idle. The processors may be limited to communication with neighboring processors only and may be configured in a hierarchical fashion. A typical asymmetric software scenario would consist of one processor running the kernel with various tasks being performed on the other processors in a master-slave type relationship. An example of an asymmetric system would a 386 microprocessor paired with a 387 floating point coprocessor.

Also, software applications must be multithreaded to benefit from symmetrical multiprocessing so that threads, or tasks, can be assigned to different processors. If an application is not multithreaded, it will run on only one of the processors. Examples of operating systems which support multiprocessing are: Novell**, Windows NT**, Unixware**, and OS/2* SMP (version of 2.1).

Multiprocessing Design Issues

In all multiprocessing systems, there are unique design issues that arise due to the flow of information among the various bus masters (multiple microprocessors and I/O subsystems) and memory (internal caches, external caches and main memory). The following are the three main issues:

1. Cache Coherency
2. Arbitration
3. Interrupt Handling

Cache Coherency. By definition, maintaining cache coherency in a cache/memory system will guarantee that a request for data from memory will always retrieve the most up-to-date data value, regardless of which cache or memory location currently holds that value. This is particularly important for write-back caches where data may not be transferred to main memory immediately. In fact, with write-back caches, the data is not written back to main memory until the last possible instant - when another processor or I/O device needs to access the data or when its location in the cache is needed for other data.

In a uniprocessor system, coherency is required between the processor's internal level 1 (L1) cache, the external level 2 (L2) cache, main memory, and I/O devices. In a multiprocessing system, the need for coherency is extended to include the multiple caches associated with each microprocessor on the bus. For example, if Processor A updates a memory location in its write-back L2 cache and another processor, Processor B, requests a read from the same memory address, Processor B must always get the most up-to-date data which, in this case, would reside in Processor A's L2 cache rather than main memory.

The responsibility for maintaining cache coherency lies in hardware. Depending on the system topology, the cache coherency may be controlled by the core logic, the L2 caches, or the processors. Independent of what the controlling hardware is, the process used is the same. The

main elements of this process are snooping the bus, interrogating the caches and, if necessary, updating main memory.

When another bus master has control of the bus, the cache controlling hardware snoops, or monitors, the system bus in order to determine which address in main memory the bus master is accessing. After learning the desired memory address, each L2 cache controller interrogates its L2 cache and its associated processor's internal L1 cache to determine, by use of the MESI model defined below, if either cache has modified data for that memory location. If the result is yes, indicating that main memory has stale data, then the cache controller seizes control of the bus from the bus master and updates the location in main memory. After the data is written back to main memory, the cache controller releases the bus, and the bus master may complete its read or write cycle.

The MESI cache coherency model is used to determine the state of the data stored in cache memory. The model keeps track of the coherency of data for each cache line -- usually 16 or 32 bytes each. The four defined MESI states are listed below.

- **M**odified — The cache line has been updated, or modified, by the processor.
- **E**xclusive — The cache line is unaware of any other cache containing this data.
- **S**hared — An exact duplicate of the cache line exists in at least one other cache.
- **I**nvalid — The cache line does not contain valid data. This is the initial state of each cache line after reset.

Arbitration. One requirement of symmetric multiprocessing systems is that all processors must be able to communicate with each other and with the I/O subsystems. In order to provide interference-free and orderly communication in multiprocessing systems, both hardware and software arbitration mechanisms must be implemented.

Hardware arbitration ensures that only one bus master, a single processor or I/O device, has control of the bus at any time. In a design with hardware arbitration, a bus master must issue a bus request to the arbitrator. When the arbitrator determines that the previous bus master has disconnected from the bus and that it is the requesting bus master's turn to use the bus, it grants the bus to the bus master. Hardware arbitration schemes are dependent on the chosen topology and may be controlled by the core logic, the L2 caches, or the processors.

Software arbitration is necessary to ensure that in certain situations only one processor or I/O device can access a specific address or run a particular piece of code at a time. Locked cycles are often used to control software arbitration. For example, if one processor is performing a critical memory operation, such as updating a semaphore using a read-modify-write operation, it will lock out any of the other bus masters from accessing the locked address until it has completed the locked cycle operation.

Interrupt Handling. The third challenge in designing a multiprocessing system is the channeling and processing of interrupts. This is accomplished by the interrupt controller, which is typically a part of the core logic. In a multiprocessing system, the interrupt controller receives interrupt

requests from I/O devices and/or processors, determines which processor(s) shall service each interrupt, and routes the interrupts to the correct microprocessors.

In a symmetric system, any interrupt from any source can be routed to any processor and handled there. Two solutions available today for interrupt handling in SMP systems are AMD** and Cyrix's** OpenPIC** specification and Intel's** APIC** specification. A more detailed discussion on interrupt handling will be provided following an overview of multiprocessing system configurations.

Multiprocessing System Configurations

Multiprocessing is used both in servers and high-end desktop systems. As with any design, the trade-off between desired performance and cost as well as technical limitations helps to determine the type of system configuration and the number of processors used. For example, a typical server configuration today might have four microprocessors, whereas multiprocessing desktop systems usually have only two. Note, however, that the actual number of microprocessors can be much higher. In fact, massively parallel systems such as IBM's POWERparallel* systems or Intel's supercomputers can have upwards of 512 or 1024 microprocessors. Given these variables, there are various strategies a designer may use in developing a multiprocessing system. The following three are evaluated in this paper.

1. Private Cache Symmetric Multiprocessing (PC SMP)
2. Shared Look-Through Cache Symmetric Multiprocessing (SLTC SMP)
3. Shared Look-Aside Cache Symmetric Multiprocessing (SLAC SMP)

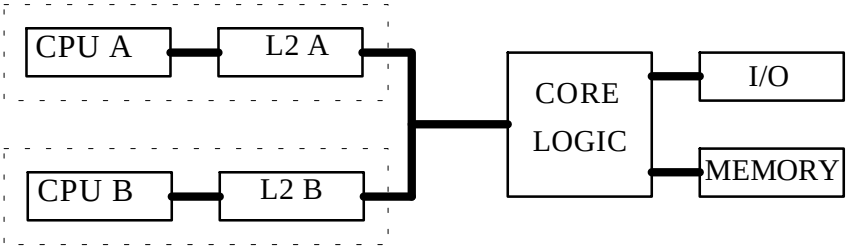

Diagram 1: Private Cache Symmetric Multiprocessing (PC SMP)

(Dotted lines show potential integration)

Private Cache Symmetric Multiprocessing (PC SMP). In a PC SMP topology, each microprocessor has its own unique look-through L2 cache. A typical PC SMP configuration is shown in Diagram 1. Arbitration and cache coherency in PC SMP systems are usually controlled by the core logic; however, a few processors, such as the Intel Pentium** processor are designed to handle arbitration and cache coherency directly over a private bus between the processors.

The individual L2 caches provide a number of performance enhancements. The first performance improvement is due to a smaller amount of electrical loading on the microprocessors. Since each processor is connected only to its L2 cache and not the other processors', its electrical loading is reduced. This allows for faster bus speeds between the processor and its L2 cache, assuming the L2 can support the higher frequency. Even better performance can be achieved by integrating the L2 cache with the processor as one chip.

The PC SMP system also benefits from the look-through nature of the individual caches. A look-through cache provides processor bus - system bus isolation, which allows for increased processor performance and bus concurrent operations.

Each of the look-through L2 caches provides a buffer between the system bus traffic and its processor, allowing only relevant requests through to the processor. For example, since a look-through L2 cache always has a copy of every line that is resident in its processor's internal L1 cache, the processor does not need to be interrupted and asked to check its internal L1 cache if the memory location is not stored in its L2 cache. With fewer interruptions, the processor's performance is improved.

Due to the isolation between the processor bus and the system bus, multiple operations can be performed simultaneously to improve system performance. This is referred to as bus concurrent operations. An example would be one processor accessing its external L2 cache memory while another bus master accesses memory or I/O over the system bus at the same time.

Another advantage of this topology is a reduction in system bus traffic. Since each processor has frequently used data stored in its own L2 cache, the number of main memory accesses occurring on the system bus to fulfill processor requests is greatly reduced. This benefits both the processors and the system bus in the PC SMP system. The processors incur fewer wait states when retrieving data from main memory, and the system bus utilization by each processor is decreased.

Since the use of individual look-through caches reduces the percentage of time each processor ties up the system bus, more processors may be implemented in the system. However, if the percentage of system bus time required by all bus master devices adds up to over 100%, the bus bandwidth is exceeded, and individual processor performance decreases due to added wait states for bus accesses. For example, a 64-bit data bus in a system running at 33 MHz yields a bandwidth equal to 264 MB/sec. Assuming the bus is 30% utilized by each processor (.30 * 264 = 79 MB/sec) and that there are four of these microprocessors in the system (4 * 79 = 316 MB/sec), the bus would be saturated!

The technical drawback to PC SMP designs is the demand placed on the system bus due to L2-L2 cache snooping and electrical loading. In order to maintain cache coherency with multiple caches on the bus, L2-L2 snoop traffic is now required. The additional L2-L2 snoop traffic puts a high demand on an already crowded system bus. Furthermore, the high number of electrical loads on the bus demands that the system bus be run at a lower speed than the processor bus. One-half of the processor bus speed is typical.

An estimate of the overall system performance gain for a dual-processor PC SMP system over a uniprocessor system is 1.6X - 1.8X. For a quad-processor system, the gain increases to around 2.8X - 3.4X. The tradeoff for the high performance of these systems is cost. In order to provide the individual look-through caching capability, multiple expensive dual-ported L2 caches must be used. Due to the higher costs, this configuration is used exclusively for servers.

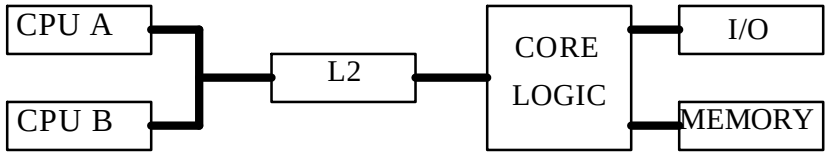

Diagram 2: Shared Look-Through Cache Symmetric Multiprocessing (SLTC SMP)

Shared Look-Through Cache Symmetric Multiprocessing (SLTC SMP). In an SLTC SMP configuration, a look-through cache is shared between the processors, as shown in Diagram 2, in order to reduce system costs. However, sharing the cache also reduces the per processor caching capability, limiting the number of processors that can be used effectively to only two.

While each processor has only half as much caching capability as in the PC SMP system (assuming the same size L2 cache) and must share the processor bus, the performance benefit due to the processor bus - system bus isolation as discussed for the PC SMP configuration still exists. The net performance gain for this type of system is approximately 1.4X - 1.6X over a single processor configuration.

In the SLTC SMP configuration, the arbitration and cache coherency is usually controlled by either the L2 cache or the processors themselves. In this scenario, the core logic has no knowledge of the multiple processors behind it, and standard desktop core logic may be used. However, if an L2 cache that is not multiprocessing-capable is used in an SLTC SMP scheme, then extra "glue" logic must be included on the system board, or possibly in the core logic. This is less desirable as it would either increase the design complexity for the system board designer or raise the cost of the core logic.

Overall, the cost savings over the private cache configuration permits the use of the SLTC SMP topology in high-end desktop systems.

Shared Look-Aside Cache Symmetric Multiprocessing (SLAC SMP). In a SLAC SMP topology, the multiple processors share one look-aside L2 cache that resides on the system bus as shown in Diagram 3. This is very similar to a typical uniprocessor desktop configuration, except with an additional processor.

In this scheme, various degrees of core logic support are necessary. If the processor controls the arbitration and cache coherency directly, standard desktop core logic may be used.

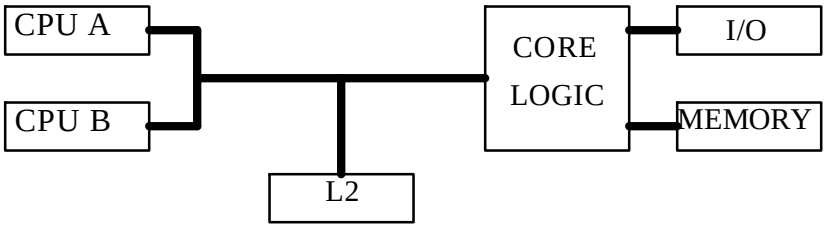

Diagram 3: Shared Look-Aside Cache Symmetric Multiprocessing (SLAC SMP)

Otherwise, special core logic would need to be incorporated to provide the multiprocessing control.

The biggest advantage of this topology is cost. Of the three multiprocessing solutions discussed in this paper, the SLAC SMP is the least costly to implement. Look-aside caches are less expensive than look-through caches since they have only one port and have a simpler design.

Also, since the processors are sharing an L2 cache, the only unique expense incurred is for the multiprocessing-capable core logic (if required).

The disadvantage to this type of configuration is the high amount of traffic between the processors, the L2 cache, and the core logic on the system bus. In a look-aside configuration, there is no bus isolation, or filtering, of irrelevant information (i.e., a snoop must go to all processors and the L2 cache). All traffic occurs on the system bus. To avoid saturating the bus, this configuration is typically used with only two microprocessors. Due to the overhead on the system bus, the performance gain is usually around 1.3X - 1.5X.

Although it has the lowest performance of the three multiprocessing configurations, SLAC SMP systems are the most commonly used. The price/performance ratio delivers a solution well suited for the high-end desktop market.

Configuration Summary. Table 1 summarizes the performance gain and relative costs associated with each of the three multiprocessing configurations which have been discussed. As expected, performance gain is directly proportional to cost.

Configuration	Performance Gain	Relative Cost
PC SMP	1.6 - 1.8 X	$$$$
SLTC SMP	1.4 - 1.6 X	$$
SLAC SMP	1.3 - 1.5 X	$

Table 1: Multiprocessing System Configuration Summary. Performance gain estimates of three dual-processor SMP system configurations versus a uniprocessor system and the relative costs of each configuration.

Uniprocessor Interrupt Handling

As stated earlier, the channeling and processing of interrupts is handled by the interrupt controller. In a symmetric multiprocessing system, any interrupt from any source can be routed to any processor and handled there. Before discussing how the interrupt controller works to control interrupts in such an SMP system, a general overview of the industry standard 8259 interrupt process for uniprocessor systems is provided.

The Interrupt Controller. In an x86 uniprocessor system, there are many sources for interrupts (the serial interface, parallel interface, floppy/hard disk controller, network adapter, keyboard, mouse, etc.) but only one interrupt pin on the single microprocessor to receive them. It is the responsibility of the programmable interrupt controller (PIC) to receive interrupt requests from the various sources, prioritize them, and route them in order of priority to the microprocessor.

In a multiprocessing system, extra support is needed to handle the processing of interrupts by multiple processors. With multiple processors available to process interrupts, the PIC now needs to decide which processor to deliver each interrupt to and keep track of these assignments.

Also, multiprocessing systems must be able to handle interprocessor interrupts. In addition to the interrupt sources present in a uniprocessor system, each processor is now a potential interrupt source and may direct interrupt requests to any of the other processors in the system. Interprocessor interrupts provide communication between processors for reasons such as power management or to restart or shutdown the system.

8259 Interrupt Process. Any x86 multiprocessing interrupt solution must maintain compatibility with the 8259 programmable interrupt controller that has been the industry standard interrupt controller since the PC/AT. Both APIC and OpenPIC support 8259-type interrupt control. An overview of the 8259's interrupt process will provide a basic understanding of the data exchange that occurs during an interrupt in a uniprocessor system. The interrupt process for an 8259-type interrupt consists of the following steps:

1. The I/O device issues a hardware interrupt to the 8259 PIC by raising the signal on the PIC's pin to which it is connected.

2. The PIC prioritizes concurrent interrupts using a hardware prioritization scheme where the priority of each interrupt request is dependent upon which interrupt request pin the I/O device is connected to on the 8259. After determining the highest priority interrupt request, the PIC drives the interrupt line to the processor to inform it of the interrupt request. Since the processor can deal with only one interrupt at a time, all other pending interrupts are masked, or ignored.

3. The processor responds to the 8259 with an interrupt acknowledge bus cycle.

4. The PIC responds to the processor's interrupt acknowledge cycle by sending the interrupt message across the data bus to the processor. Included in the interrupt message is the interrupt source and the pre-programmed interrupt vector (8-bit pointer

to the interrupt handler).

5. The processor branches to the vector address and executes the corresponding interrupt handler.

6. Upon completion of the interrupt handler, the processor issues an end of interrupt (EOI) command to the PIC.

7. The PIC clears the completed interrupt and unmasks any pending interrupts. If pending interrupts exist, the PIC begins processing of the next hardware interrupt.

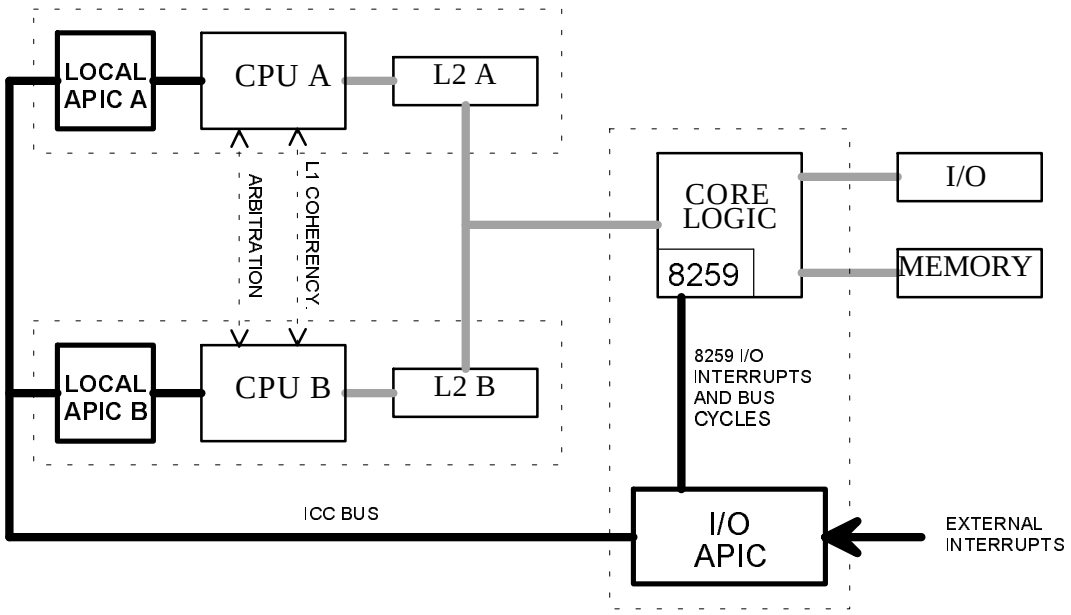

Diagram 4: APIC INTERRUPT HANDLING IN A PC SMP SYSTEM

(Dotted lines show potential integration)

APIC Interrupt Handling

Intel's solution for the handling of interrupts in a multiprocessing application is its Advanced Programmable Interrupt Controller (APIC). In hopes of establishing APIC as the industry standard, Intel is providing board designs and working with core logic chipset manufacturers in addition to embedding APIC technology in its latest Pentium** and P6** processors and chipsets.[1]

APIC Architecture. The APIC architecture is distributed between two basic functional units, the local APIC and the I/O APIC as seen in Diagram 4. Together, the APIC units are responsible for

[1] Electronic Engineering Times - Issue 840 - *AMD, Cyrix push open-server spec*

delivering interrupts from interrupt sources to interrupt destinations throughout the multiprocessing system. The local and I/O units may be implemented in hardware as a discrete chip containing both units (e.g. Intel's 82489DX interrupt controller), or they may be integrated with other system components. For example, the local APIC is embedded in Intel's Pentium microprocessors, and the I/O APIC can be built into the core logic chipset.

As a discrete chip, the 82489DX may be used as either the local unit or the I/O unit, but not both since one local APIC is required for each processor. Depending on the number of interrupt lines in the system, one or more I/O APICs may be used. In order to distinguish the APIC units, each local and I/O APIC has an ID register that is programmed during system initialization. Up to 240 total APIC devices (maximum of 32 local APICs) can be supported. The APIC IDs are used by software to specify destination information and to access APIC's communication bus.

The local and I/O APIC units communicate through the Interrupt Controller Communications (ICC) bus. In the case of an I/O interrupt, the I/O APIC unit receives the interrupt request, determines who it will direct it to, formats an interrupt message, and sends it over the ICC bus. The local unit that is addressed by its ID accepts the message sent by the I/O unit and delivers it to its processor. The use of a unique ICC bus provides two advantages: 1) interrupt-related traffic is off-loaded from the system bus and 2) processors can share in the handling of interrupts without the software's knowledge.

Interprocessor interrupts are also passed along the ICC bus. These interrupts are handled in an APIC system as a bus message from the originating processor's local APIC to the local APIC(s) of the intended processor(s). The originating local APIC formats the message in the same manner as an I/O APIC would, including information such as the destination ID and interrupt handling vector.

APIC Interrupt Modes. The APIC architecture defines three different interrupt modes: PIC mode, Virtual-Wire mode, and Symmetric I/O mode. The first two modes, PIC and Virtual-Wire, provide support for PC/AT compatible system initialization. At least one of these two modes must be implemented in order to boot-up the system. The Symmetric I/O mode is required for all multiprocessing APIC systems and provides the APIC interrupt handling support once all microprocessors have been initialized.

PIC Mode and Virtual-Wire Mode. During system initialization, the boot processor works to enable the other processors. Until the rest of the processors are initialized to the system, 8259-type interrupt support for a uniprocessor system is required. Both the PIC mode and the Virtual-Wire mode provide this support by enabling direct communication between the 8259 and the boot processor. The difference between these two modes is in their implementation.

The PIC mode is implemented as a standard 8259 configuration where the 8259 is connected to the processor's interrupt pin (not shown in APIC diagram), and all APIC components are bypassed. In Virtual-Wire mode, the 8259 communicates with the processor through the local APIC unit. It is not connected directly to the processor as in PIC mode, but to one of the local APIC's pins (not shown in diagram). The local APIC acts as a virtual-wire by simply passing the interrupt signals between the 8259 and the processor. In this mode, the I/O APIC is not used.

Since I/O APICs are ignored when operating in PIC and Virtual-Wire modes, interrupt sources connected to the I/O APICs must also be connected to the 8259 to be heard. The operating system must have the ability to enable the special connections for PIC and Virtual Wire modes and disable them for Symmetric I/O mode operation. After the system is initialized, the operating system switches the interrupt mode to Symmetric I/O mode to support the multiple processors.

Symmetric I/O Mode. In Symmetric I/O mode, I/O interrupts may be configured to request interrupts to the 8259 or the I/O APIC. If an interrupt request is directed to the 8259, the request is forwarded to the I/O APIC. Once the I/O APIC has received an interrupt message, either from the 8259 or directly, it uses the APIC protocol to prioritize and route interrupts across the ICC bus to the processor. This process flow is similar to the 8259 process except that an extra step is added for 8259 interrupts to forward the interrupt information to the I/O APIC and that the APIC protocol is used for prioritizing and sending the message.

Interrupt Servicing in APIC Symmetric I/O Mode. There are two methods available in APIC's symmetric I/O Mode to determine which processor will get to service an interrupt request: fixed and lowest priority. In fixed delivery mode, the interrupt is unconditionally delivered to all the processors' local APICs whose ID matches the destination information supplied with the interrupt. This may be one or more processors. When the lowest priority method is used, the APIC will usually direct the interrupt to the processor that is executing the lowest-priority process in the system. However, greater efficiency can sometimes be obtained by delivering an interrupt not to the lowest priority processor, but to the processor that has most recently handled an interrupt from the same source -- the focus processor. In this case, the interrupt handler's code will often still reside in the focus processor's cache.

After determining which processor(s) will service the interrupt request, the controller must direct the interrupt to the correct processor. Unlike the 8259 interrupt process where the PIC must wait for an interrupt acknowledge from the processor, in an APIC system, the APIC unit can send the source information and interrupt vector along with the interrupt request. The delivery scheme that APIC uses is simply to broadcast the interrupt message on the ICC bus. Any local APIC that matches the destination ID(s) will receive and service the interrupt.

Since any APIC unit is capable of sending an interrupt message (I/O APICs for I/O interrupts and local APICs for interprocessor interrupts), but only one message can be delivered at any time on the common ICC bus, the local and I/O APIC units need to arbitrate for control of the bus. The arbitration scheme used is a "rotating priority". One or more units may start sending an interrupt message simultaneously. If this occurs, the rotating priority arbitration scheme is used and the unit with the highest priority, the winner, is granted control of the bus and is allowed to complete its interrupt message. After completing its message, the winner sets its priority level to zero (the lowest), and all the other units increment their priority level by one. In this manner, the APIC units rotate through the highest priority standing. The exception to this arbitration protocol is the transmission of special messages, such as an end-of-interrupt, which will get control of the bus right away.

The main disadvantage to using APIC for interrupt control is that the APIC architecture is proprietary and for non-Intel microprocessors is hardware intensive. For I/O APIC support,

designers have the option of buying the discrete 82489DX controller or licensing the circuit from Intel for inclusion in the chipset. If non-Intel microprocessors are being used, an 82489DX must be purchased for each processor for local APIC control, adding to system costs and design complexity.

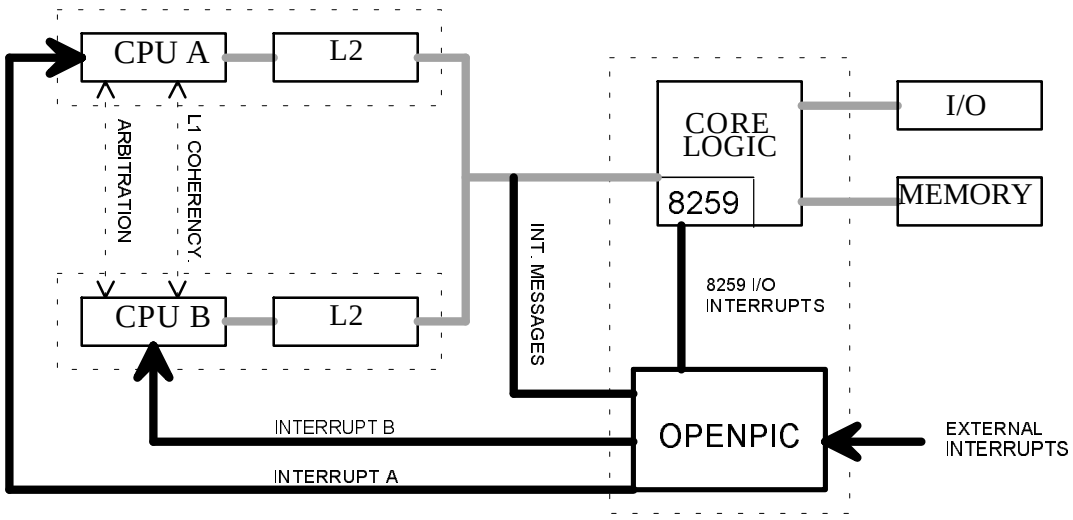

Diagram 5: OPENPIC INTERRUPT HANDLING IN A PC SMP SYSTEM

(Dotted lines show potential Integration)

OpenPIC Interrupt Handling

With the goal of offering an alternative to Intel's proprietary APIC technology as the industry standard, AMD** and Cyrix** have jointly developed an Intel-independent interrupt control solution for SMP systems. The AMD/Cyrix strategy is to provide an open x86 multiprocessing specification, OpenPIC (Open Programmable Interrupt Controller), at no cost to computer manufacturers. The OpenPIC Architecture supports design into systems using x86 vendors' microprocessors, such as the Intel Pentium.

OpenPIC Architecture. This open multiprocessing specification describes the required function a "generic" interrupt controller must have in order to support the OpenPIC architecture. The OpenPIC architecture is based upon an expansion of the standard 8259 interrupt controller protocol, supporting up to 2048 interrupt sources and up to 32 processors. As in an 8259 configuration, the controller has a unique connection to each processor's interrupt pin, as shown in Diagram 5, and uses the system bus for communicating the interrupt message (source and vector).

An OpenPIC controller may be implemented as a stand-alone device, or it may be integrated into the core logic chipset. Unlike the distributed APIC architecture consisting of I/O and

local units, OpenPIC control is centralized. Since the OpenPIC controller communicates directly with each processor via its interrupt pin, no additional per processor hardware is required.
OpenPIC Interrupt Modes. In order to provide support for PC/AT compatible system initialization, the OpenPIC architecture defines two modes of operation, 8259 Pass-Through mode and Symmetric Multiprocessing (SMP) mode.

8259 Pass-Through Mode. In 8259 pass-through mode, the 8259 interrupt request output will be passed directly through the OpenPIC device to the boot processor. This provides support for PC/AT compatible interrupts during boot-up. This mode is similar to APIC's Virtual-Wire mode. In both cases, the advanced interrupt controller acts simply as a hard-wired connection. After initialization is complete, the system will disable the 8259 Pass-Through mode and switch to SMP mode.

Symmetric Multiprocessing Mode. In SMP mode, there are two ways the OpenPIC controller receives interrupt requests: via the 8259 and directly from the I/O devices. It is the responsibility of the OpenPIC controller to prioritize the requests and distribute the interrupts to the processors.

Interrupt Servicing in OpenPIC SMP Mode. OpenPIC's SMP mode interrupt process flow is very similar to the standard uniprocessor 8259 process, with differences due to a different source prioritization scheme and the need to designate which processor is to service the interrupt.

In an OpenPIC system, source interrupts are prioritized using the following method. Each interrupt source has a software programmable priority value, ranging from 0 (lowest) to 15 (highest). These values are programmed during initialization, but can be changed later. If concurrent interrupt requests are received by the controller, they are processed in order of their priority. Each processor has a task priority, also ranging from 0 to 15. Task priorities are set dynamically by software. In order for the controller to deliver an interrupt to a processor, the source's priority must be greater than the processor's task priority.

OpenPIC has two modes available for determining which processor(s) will receive the interrupt: directed mode and distributed mode. Directed mode is like APIC's fixed mode where the interrupt is delivered to specific processors as specified by the destination information. Directed mode may be used for single or multiple destination interrupts. Distributed mode is a dynamic delivery mode for multiple destination I/O interrupts. In distributed mode, the controller uses an implementation-specific algorithm to determine which processors it may distribute the interrupt to. The delivery modes available to the controller are dependent upon the interrupt's source and destination information as shown in Table 2.

Interrupt Type	**Single Destination**	**Multiple Destination**
I/O Interrupt	Directed	Distributed
Interprocessor Int.	Directed	Directed

Table 2: OpenPIC Delivery Modes.

Once the OpenPIC controller determines which processor(s) will service the interrupt, the controller drives the interrupt line(s) to the processor(s) to notify them of the interrupt request. This delivery method is very different from APIC's approach where the interrupt message is

broadcast on the common ICC bus and any processor matching the destination information will service the interrupt.

Interprocessor interrupts are supported in an OpenPIC design by hardware connections between each processor and one of the OpenPIC controller's external interrupt request pins. To initiate an interprocessor interrupt, the originating processor drives its interrupt request line to the OpenPIC controller. Once the request is received by the controller, the interrupt is handled in the same way as I/O device interrupts. Self-interrupts, where the processor specifies its own ID as the destination are also supported.

Summary

Multiprocessing systems may be symmetric or asymmetric and must be controlled by a multiprocessing-capable operating system. The three major design issues that must be handled in any multiprocessing system are: cache coherency, arbitration, and interrupt handling. Depending upon the system's configuration, cache coherency and arbitration may be handled by the core logic, L2 caches, or the processors. Interrupts are handled by interrupt controllers.

Three symmetric multiprocessing system configurations were discussed: Private Cache SMP, Shared Look-Through Cache SMP, and Shared Look-Aside Cache SMP. The private cache configuration provides the highest performance along with the highest costs and is usually used for servers. The SLTC SMP approach reduces costs by sharing a look-through L2 cache. SLAC SMP systems provide the lowest cost solution by sharing a look-aside L2 cache. Although the SLAC SMP strategy provides the lowest performance of the three, it is the most commonly used configuration today due to its lower costs.

In a multiprocessing system, it is the responsibility of the interrupt controller to receive interrupt requests from I/O devices and/or processors, to determine which processor(s) shall service each interrupt, and to route the interrupts to the correct microprocessors. Two multiprocessing interrupt solutions are currently available: Intel's APIC and AMD/Cyrix's OpenPIC. The APIC solution consists of a proprietary distributed architecture. System designers have the option of purchasing discrete APIC devices or licensing the circuit from Intel for integration in the chipset. The OpenPIC specification defines only the function required by a centralized "generic" controller; it does not stipulate the use of any specific device(s) or charge any licensing fees for use of the architecture.

References

1) Shanley, T., 80486 System Architecture, Mindshare, Inc., Richardson, TX, 1994.

2) Shanley, T., and Anderson, D., ISA System Architecture, Third Edition, Mindshare, Inc., Richardson, TX, 1995.

3) Anderson, D. and Shanley, T., Pentium Processor System Architecture, Mindshare, Inc., Richardson, TX, 1993.

4) Intel MultiProcessor Specification, Version 1.4, Intel Corporation Literature Center, Mt. Prospect, IL, 1995.

5) An APIC-Based Symmetric Multiprocessor System Design, Version 1.0, Intel Application Note #AP-474, Intel Corporation Literature Center, Mt. Prospect, IL, 1994.

6) Pentium Family User's Manual, Vol. 3: Architecture and Programming Manual, Intel Corporation Literature Center, Mt. Prospect, IL, 1994.

7) OpenPIC Multiprocessor Interrupt Controller Register Interface Specification, Revision 1.2, Advanced Micro Devices and Cyrix Corporation, 1995.
