
THE 82460GX SERVER/ WORKSTATION CHIP SET

THIS ARTICLE PROVIDES AN INTRODUCTION TO THE MEMORY, I/O, AND GRAPHICS SUBSYSTEMS OF INTEL'S ITANIUM PROCESSOR CHIP SET AND DISCUSSES SEVERAL ASPECTS OF THE PROCESSOR BUS.

Eric Dahlen
Jennifer Gustin
Susan Meredith
Doug Moran
Intel

..... Designing a chip set for a new processor architecture like the IA-64 requires handling multiple aspects. They include implementing the processor interface; providing sufficient I/O bandwidth for servers; supporting Accelerated Graphics Port (AGP) graphics; and providing sufficient memory bandwidth for the processor, I/O, and graphics. Though computer systems must be designed from a systemwide perspective, this article doesn't discuss mechanical aspects of system design such as cooling, packaging, or power delivery.¹

The 82460GX

Figure 1 (next page) shows a block diagram of the 82460GX with its four main interfaces.

- The processor bus (referred to as FSB for frontside bus) runs at 133 MHz and supports up to four processors. The data bus is double-pumped.
- Each of the two memory ports has an independent address, command, and data bus.
- Four 16-bit interconnect ports, referred to as F16 buses, move command and data packets between the I/O bridge chips and the system address chip (SAC).
- A private data bus (PDB) moves data between the SAC and the system data

chip (SDC). The PDB transfers data between the I/O subsystem and memory or between the processor and I/O. To control systemwide data movement, this bus also sends commands from the SAC to the SDC. Note that moving I/O data on the PDB and not on the FSB data pins allows processor data and I/O data to be in flight at the same time, thus increasing total system bandwidth.

Instead of describing each component in detail, we discuss each of the three main subsystems (memory, I/O, and AGP²) and describe each component's role within that subsystem. Table 1 lists the individual chip set components.

The SAC provides the main control for transactions moving through the system. It consists of a bus interface unit (BIU), memory controller (MIU), an interface to the F16 buses (FIU), and an internal interconnect bus (ICU) that connects all the subsystems. All communication between the processor and the chip set take place over the FSB through the BIU.

The BIU drives addresses for I/O transactions on the FSB to permit processor snooping. The FSB moves interrupt messages from the I/O system or between processors. AGP reads and writes are the exception to the rule.

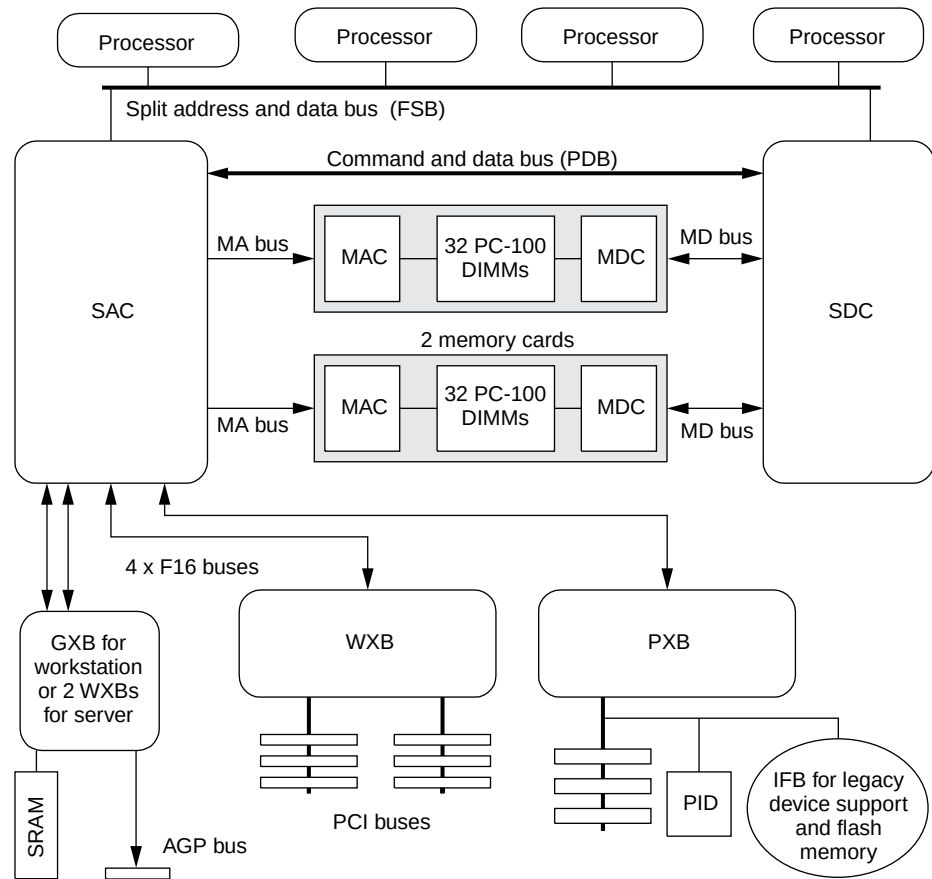


Figure 1. Diagram of a typical 82460GX platform. See Table 1 for a list of components.

Table 1. 82460GX components.

Component	Description
SAC	The system address chip connects to the address and command signals on the FSB; routes requests to the memory or I/O subsystem, and maintains the in-order queue and tracks deferred out-of-order transactions.
SDC	The system data chip contains a 4-ported data buffer and controls the data flow for the chip set. Its buffer has separate read/write ports for the FSB, PDB, and each of the 2 memory ports.
MAC	The memory address chip drives RAS, CAS, and the memory address to the DRAM, and controls the flow of data between the SDC and the MDC.
MDC	The memory data chip multiplexes the data from the DRAM to the SDC and drives write data to the DRAM. It buffers multiple write transactions.
PXB	The PCI expansion bridge contains two 32-bit, 33-MHz PCI buses.
WXB	The wide expansion bridge contains two 64-bit, 66-MHz PCI buses.
GXB	The graphics expansion bridge provides an AGP Pro port for workstations; supports 4x AGP transfers and fast writes as well as standard AGP features.
IFB	The I/O and firmware bridge provides a bridge from PCI to legacy devices such as IDE, keyboard, mouse, floppy, and others, and provides access to the firmware hub, which contains BIOS code.
PID	The programmable interrupt device handles PCI interrupts.

The AGP specification defines reads and writes to target memory directly without requiring them to be kept coherent with the processor's caches, therefore these accesses are not snooped on the FSB. The FSB is also treated as the single point of ordering for PCI and processor traffic.

Memory subsystem

This subsystem consists of the SAC's MIU, the SDC's data buffer, the MAC, the MDC, and the dynamic RAMs. To support large databases for servers, the designers decided that the system must support 64 Gbytes of DRAM, thus requiring 64

DIMMs. The processor bus has a peak bandwidth of 2.1 Gbytes/s. Since I/O data is moved on the PDB and not on the processor bus, there's an additional 2.1 Gbytes/s of peak I/O bandwidth. Therefore, to provide a balanced system, the memory subsystem had to have a total bandwidth of 4.2 Gbytes/s.

The 82460GX supports 32 standard PC100 synchronous DRAM DIMMs on each memory board. Each of the two memory ports has its own address and command bus (MA bus in Figure 1), and its own data bus (MD bus) between the MDC and the SDC. Addresses can be sent to both ports on the same clock cycle, and data can be transferred over both ports on each clock cycle. Each port has a peak bandwidth of 2.1 Gbytes/s, for the required system memory bandwidth of 4.2 Gbytes/s.

For processor and I/O transactions, the BIU sends the MIU (Figure 2) transactions from the FSB. PCI requests move from the FIU to the BIU, so the address can be snooped on the FSB, and then move to the MIU. All noncoherent AGP requests are sent to the MIU directly from the FIU. When a read request is received, the MIU may send it immediately to the MAC on the next clock cycle. This means that the read is done without a guarantee that the data will be used, since snoop results are unknown at that point. However, this reduces latency when getting the read data from the DRAM.

All FSB transactions, whether speculative or not, enter the snoop queue. The SAC waits for snoop results to determine what a transaction requires before it can complete. For instance, a read that hits a dirty line will get a snoop of HITM# (hit modified). In this case instead of a read, the memory system will snarf the data off the FSB and write the modified line to the DRAM. With the snoop results, the transactions enter either the read or the write queue.

AGP transactions bypass the snoop queue and are placed directly into the read or write queue. The Select block arbitrates between entries in these queues and speculative reads to determine which command to send to the memory card. Every clock cycle, the arbiter attempts to send a transaction that best interleaves between the two memory cards. Writes have lower priority than reads and are dispatched when no reads are pending.

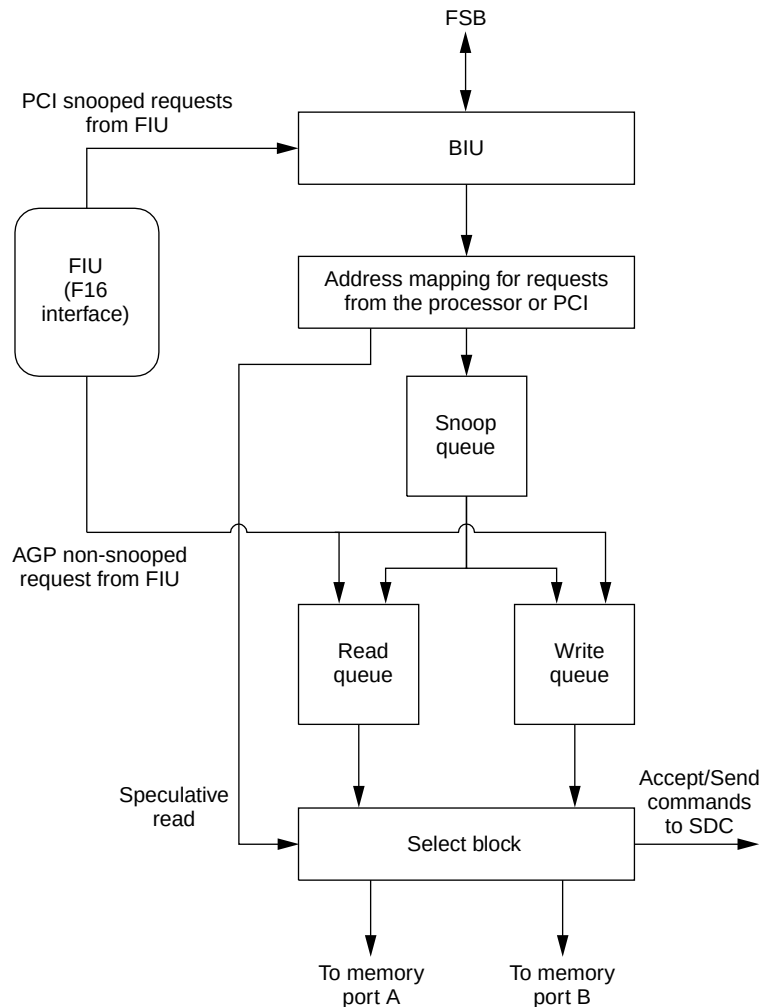


Figure 2. Block diagram of the SAC and memory interface unit (MIU).

As the read/write transaction is issued to the memory card, the SAC sends a string of commands to the SDC that determine three things:

- the response to drive on the FSB,
- which port will receive the incoming data (Accept commands), and
- the port to which it should forward outgoing data (Send commands).

All transactions, whether from the processor, PCI, or AGP, are broken up into these smaller discrete commands for data movement. The SDC compares the tag on any incoming data against its list of Accept commands, places the data into the buffer, and

marks it as valid. Each cycle, the SDC compares its list of Send commands with the buffer entries that have a valid bit set and moves valid data from the data buffer to the correct port.

Since the latency to move write data into the DRAM doesn't affect system performance, writes occur in a leisurely fashion. Write data from the processor is placed in an SDC buffer location. When the data bus between the SDC and MDC is idle, the write data is moved from the SDC to the MDC and buffered there. Only after the write data is in the MDC does the SDC inform the SAC that it can send the Write command to the memory card. The card will then issue the RAS and CAS for the write.

Special cases exist for the memory system.

- Writes may get a HITM#. In this case the memory subsystem would see two separate data transfers on the FSB, merge them together, and then issue one write to the DRAM.
- A write of fewer than 64 bytes is done as a read-modify-write in the SDC. The memory subsystem performs a read, merges the new data into the old line, and writes the results back to the DRAM.
- A read follows directly behind a write to the same address. Here, the read data is provided directly from the SDC's buffer instead of moving the write data out to memory and then reading it back.

I/O subsystem

This subsystem is the key element of a server. It's also critical for workstations, which move large amounts of data over networks and to graphics. The FSB has a peak data bandwidth of 2.1 Gbytes/s, but an effective address bandwidth of 2.8 Gbytes/s. I/O addresses are snooped on the FSB, but the I/O data is moved on the PDB and not on the FSB. Thus the system could have 2.1 Gbytes/s of I/O traffic and 0.7 Gbytes/s of processor traffic.

The I/O subsystem was therefore designed with a peak bandwidth of 2.1 Gbytes/s to handle large amounts of I/O data. This covers the case where the processor is only examining a small fraction of the data, perhaps searching for one field out of large database records or creating routing headers for disk-to-network traffic.

The I/O subsystem must also have enough slots or connections to actually achieve these bandwidths. If gigabit Ethernet and SCSI cards have around 100 Mbytes/s of bandwidth each, the system requires 15 to 20 PCI bus slots. Even with faster cards available for disk connections, having 20 slots provides better interleaving of the accesses across multiple PCI buses.

The subsystem must also provide support for legacy devices such as IDE, keyboard, mouse, USB, and possibly VGA graphics. There must be access to BIOS and other code that is required for booting. The IFB provides all these interfaces and attaches to one of the PCI buses on the PXB.

Given these requirements, the 82460GX provides four 533-Mbyte/s F16 buses. Each F16 attaches to a WXB that has two buses housing either two PCI slots at 66 MHz or four slots at 33 MHz. This allows a typical configuration of eight 66-MHz slots and eight 33-MHz slots on three WXBs. It also allows the PXB on the fourth F16 to have several 33-MHz slots along with the legacy support.

With eight independent PCI buses in the system, the I/O subsystem has a great deal of system parallelism to support multiple transactions that are in simultaneous progress. To support this parallelism, there are separate independent command queues and data buffers. The SAC contains four F16 FIUs. By providing logically separate queue structures for each bus or port, each F16 bus supports a dual-ported device (such as a WXB with two independent PCI buses).

The separate queue structures are required to ensure forward progress of all queues in the presence of peer-to-peer traffic (traffic from any PCI bus to any other PCI bus). The 82460GX maintains PCI ordering rules for peer traffic with respect to all other traffic. Maintaining ordering between traffic to a peer bus and traffic to memory allows the producer/consumer programming model to be supported no matter where the flag and data are placed—in memory or on another PCI bus. Each port has separate inbound and outbound request queues (IRQ and ORQ). The IRQ contains requests from PCI targeting memory or another PCI bus. The ORQ tracks requests from processor to PCI. Resources are tracked independently for each port, so that

one port cannot block the other. For example, if port A is out of read resources but port B of the same F16 isn't, a read to A wouldn't be accepted, but a read to B would be.

Processor reads and writes to PCI are broken up into smaller commands similar to that explained for memory. A read request from the processor is sent to the xXB (PXB, WXB, or GXB). At the same time Accept and Send commands are sent to the SDC. The SDC receives data from the PDB using the Accept command and then forwards it to the FSB due to the Send command. Processor writes are similar, except that the Accept command causes data to be received from the FSB and the Send causes the data to be sent over the PDB to the SAC. The SAC will match the identity of the incoming data with the correct pending write request and forward the command and data packet to the xXB.

PCI-to-memory traffic is handled in much the same fashion. The request originates with the xXB and is sent to the SAC and placed in the IRQ. When it is at the top of the queue, the address will be sent to the FSB for snooping. The BIU and MIU handle this request much like they would a processor request. They break up the read or write into a series of Accept and Send commands to move the data to/from the DRAM and the F16 bus, via the PDB and SAC.

Writes originating on the PCI bus are posted. The xXB accepts the write and all the associated data, then forwards it up the F16. Streams of write traffic are sent as aligned 64-byte requests, so that memory atomically handles the write.

For read performance on the PCI bus, the xXB must trade off prefetching sufficient data to fully supply a streaming PCI agent versus prefetching too much data and wasting bandwidth.

Prefetching too little data will cause delays to the PCI agent, which hurts system performance. Prefetching too much data will keep the memory, FSB, and I/O channels busy with useless data movement, also hurting system performance. The xXB's prefetch algorithm will continuously issue read requests for more data when its buffers have been partially consumed by the requesting PCI agent, thus attempting to always have data ready for that agent.

Graphics subsystem

The GXB component provides a mechanism to use two of the F16 buses for a bridge to the AGP Pro² bus. The GXB and SAC couple the two buses together to present a single logical bus capable of twice the peak bandwidth. (That is roughly 1 Gbyte/s when operating at 133 MHz, making this connection a fair match for AGP 4x mode). The GXB provides a full-featured, 66-MHz AGP Pro interface fully synchronous with the processor and F16 buses, and fully plug-and-play compatible using PCI 2.2 configuration protocols.

The AGP defines the graphics address relocation table (GART) as a mechanism to translate a flat virtual address space used by the AGP device for physical system addresses. The GXB places the GART table in external SRAM directly off the GXB. This provides flexibility for configurations from no GART up to a 1-Gbyte translation aperture. The table is completely independent from main memory, eliminating mutual interference between processor, I/O, and GART-related traffic.

The GXB supports all three AGP data rates (1x, 2x, and 4x), both AGP signaling voltage levels, PIPE# or SBA request generation, the optional fast write protocol, and a full-function PCI interface. PCI and AGP traffic aren't ordered relative to each other, nor do they share any internal resources; thus they use independent queue and buffer space in the GXB core.

AGP requests are segregated into independent high- and low-priority queues in the core, then further segregated into independent read and write queues at the AGP interface. All four AGP request types (high/low priority, read/write) are supported by independent data-buffering resources. The AGP interface logic separates the four AGP request types, allowing them to reorder as allowed in the AGP specification. This provides maximum service for high-priority transactions, which can reorder indefinitely.

Internal resource allocation occurs at the AGP-core interface. Only transactions with sufficient internal data buffer resources and an available slot in their destination queue are forwarded to the GXB core. The GXB optimizes traffic by parsing longer AGP read transactions into requests limited to around 128 bytes each. (Misaligned requests are rounded to the near-

est cache line boundary.) This optimizes the use of the F16 buses in terms of packet latency versus the cost of the packet overhead.

Fence and Flush protocols affect the ordering of low-priority AGP traffic and are fully enforced in the GXB's AGP interface. When a Fence is encountered, the two low-priority queues in the AGP interface unit are synchronized, preventing any reordering across the Fence boundary. This is accomplished via a simultaneous push into the low-priority read or write queue (depending on the transaction type) of the next low-priority transaction and a parallel placeholder in the opposite queue. Hardware ensures that these two symbols are simultaneously popped when the associated low-priority request is passed into the GXB core, which effectively prevents reordering of any transactions received after the Fence command.

The Flush transaction generates a Fence as well as a low-priority read transaction. All writes preceding the Flush must be made visible to the system. This is accomplished by setting the address of the low-priority read generated on behalf of the Flush to that used by the most recent low-priority write request. This pushes all the writes to memory ahead of the read.

Processor bus interface

The Itanium processor bus is similar to the PentiumPro bus with several enhancements. The split-transaction Itanium processor bus has separate address and data buses, snoop, response, and data phases, and transaction tagging. The processor differs from the PentiumPro bus in that it

- has 44 physical address bits, instead of 36;
- transfers data at two times the system clock rate with source-synchronous transfers;
- has a 64-byte line size instead of 32 bytes; and
- delivers interrupts over the FSB instead of using the Advanced Interrupt Controller Architecture (APIC) bus.³

The biggest change in moving to the Itanium processor is the addition of the enhanced defer protocol. On the processor bus, instead

of issuing a completely new transaction, the chip set can return read data by simply initiating a data phase with the assertion of a new signal (IDS#) along with the transaction tag.

Since there's no address overhead for completing deferred reads as there was previously, it's possible to defer all reads and simply return the data to the processor out of order whenever it is available. Deferring reads prevents one long-latency transaction from holding up other transactions.

When there are multiple requests outstanding, there's a good chance that at least two of these requests will hit on the same memory chip, but not be a page hit. Since the second request must wait for the DRAM to precharge and a new RAS/CAS sequence, an in-order bus will have all later reads backing up behind the delayed one. In an out-of-order system, the long-latency request is moved aside, and other reads are allowed to return data as it becomes available.

In the same way, a write that follows a deferred read could have the write data moved on the FSB while the DRAM is accessing read data. An in-order system would require the write data to be moved on the FSB only after the read data had been returned to the processor.

An additional processor change is that interrupts are delivered on the FSB instead of using a separate path like the three-wire APIC bus. I/O interrupts are handled as writes on the PCI bus and later converted by the chip set to interrupt transactions on the FSB.

Using the FSB for interrupts allows another new feature to be added to the chip set: lowest priority interrupt delivery. Through special transactions from the processor, the chip set tracks each processor's priority level. When a new interrupt is received from the I/O system, the chip set can deliver it to the processor that has the lowest priority at the time the interrupt is received. The processor at the lowest priority level should be the processor doing the least critical work, and thus best suited to be interrupted.

Deferring transactions creates an interesting architectural concern for system data coherency under the standard MESI protocol. In general, a processor bus that handles in-order memory accesses will pass ownership of any particular line from one processor to

another. The memory system never “owns” a line, but simply gives the addressed data to any requesting agent, unless overruled by a processor claiming ownership. However, in deferring an access, the chip set becomes sole owner of the line during the time that it’s deferred.

The chip set must guarantee that no two processors get access to the incorrect line and that no committed write is missed by a later read. Assume processor A reads a line and is deferred, followed immediately by B reading the same line and being deferred. Since A doesn’t have the line yet, it won’t respond with a hit to B’s read, nor will A go to the S state instead of the E state because of B’s read. If the chip set were to simply return data to both processors, they would both be in the E state, in violation of MESI rules.

To maintain coherency, the 82460GX will retry any access to an address that it has deferred. In the case just discussed, the second read is retried until the first completes. Later when the second read is re-issued on the FSB after the first read has been completed, the chip set won’t retry it, and the first processor will provide a snoop response of a hit, signaling that both processor caches should go to the shared state. In a case in which the second transaction is a write instead of a read, it will also be retried.

Under the assumption that only cacheable reads need to follow cache-coherency rules, all cache-line reads are checked for address conflicts, but not reads of fewer than 64 bytes. (This also assumes that all cacheable reads are for a full cache line). The processor bus has a signal for writes called WSNP#, which indicates whether the processors should snoop a write. If the processor doesn’t snoop a write, the chip set won’t check for address conflicts either. A write from the processor’s write-combining buffer, which isn’t considered cached or part of the coherency domain by the processors, is one case where writes aren’t snooped.

This brief overview of the 82460GX chip set covers its main components. Note that each subsystem was not designed independently, but with the others in mind. The system has balanced the requirements for each

subsystem against those of the total platform. The 82460GX takes advantage of this approach for greater performance. MICRO

References

1. M. Bell and T. Holman, “Pentium®Pro Processor Workstation/Server PCI Chip Set,” *Proc. Compcon 96, Technologies for the Information Superhighway, Digest of Papers*, IEEE Computer Society Press, Los Alamitos, Calif., 1996, pp. 388-394.
2. Intel Corporation, *Accelerated Graphics Port Interface Specification, Revision 2.0*, May 4, 1998; http://developer.intel.com/technology/agp/agp_index.htm.
3. N. Sarangdhar and G. Singh, “An Overview of the Pentium®Pro Processor Bus,” *Proc. Compcon 96, Technologies for the Information Superhighway, Digest of Papers*, IEEE CS Press, 1996, pp. 383-387.

Eric Dahlen was the design lead/architect for the GXB and graphics port. He also worked on the 82450GX chip set, the first chip set for the PentiumPro processor. Dahlen received a BS degree in electrical engineering from Washington State University.

Jennifer Gustin, Susan Meredith, and Doug Moran were the architecture team for the 82460GX project. Gustin and Meredith also worked on the 82450GX chip set. Moran previously worked for IBM on its first multi-processor PowerPC server. Gustin received a BS degree in electrical engineering from Embry Riddle Aeronautical University. Meredith holds a BS degree in electrical engineering from the University of Louisville. Moran received an MEng degree in electrical engineering from the University of Louisville.

Direct comments about this article to Doug Moran, Intel Corp., 1900 Prairie City Rd., FM6-204, Folsom, CA 95630; douglas.r.moran@intel.com.