Bus Operations

This chapter covers NexBus processor cycles, NexBus5 system bus cycles and cache-coherency operations. The processor bus cycles are conducted primarily on NexBus although their effects can also be seen on the L2 SRAM bus. The NxCLK clock, shown in the timing diagrams accompanying this text, runs at half the frequency of the processor's internal clock.

☞ In this chapter, the term "clock" refers to the *NexBus clock* not to the processor clock, as is meant elsewhere throughout this book.

The notation regarding *Source* in the left-hand column of the timing diagrams shown in this section indicates the chip or logic that generates the signal. When signals are driven by multiple sources, all sources are shown, in the order in which they drive the signal. In some cases, signals take on different names as outputs are NANDed in group-signal logic. In these cases, the signal source is shown with additional notations, where the additional notations indicate the device or logic that originally caused the change in the signal.

Level-2 Asynchronous SRAM Accesses

Figure 18 in the *Nx586 Hardware Architecture* chapter compares the basic clock timing for the processor, its L1 caches, and the L2 cache. An L1 cache miss may cause an access to the L2 cache, which resides off-chip on a dedicated 64-bit bus. Figure 28 shows a read, write, and read to the L2 cache. Transfers can begin on any processor clock and occur at the peak rate of eight bytes every two processor clocks.

In addition, Figure 28 shows a read followed by a write followed by a read cycle. Reads (or writes) can be back-to-back without dead cycles. An idle cycle is shown after the last read. The processor clock, which runs at twice the rate of the NexBus clock (NxCLK), is represented here by its two phases, PH1 and PH2. These phases are not visible at the pins except through the delayed outputs, XPH1 and XPH2. The data-sampling point is shown as the falling edge of PH2, which is relative to the rising edge of NxCLK. Two pins for COE* are shown, A and B. Both pins are identical in function and transition on the rising edge of PH1. The two pins are made available for loading considerations.

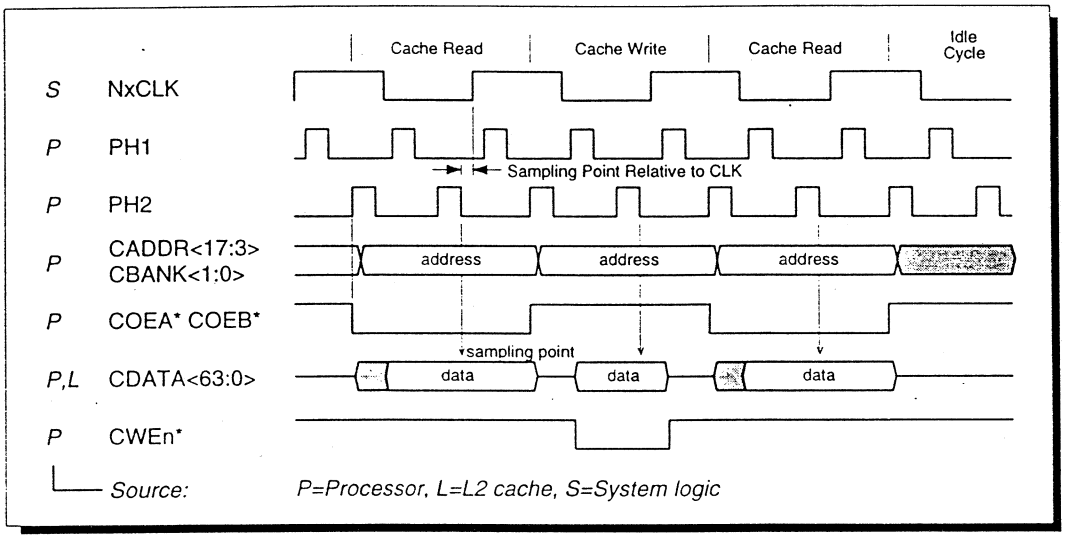

Figure 28 Level-2 Asynchronous Cache Read and Write

The L2 cache controller provides data to the processor in 3 CPU phases. In other words, the cache cycle time is 1.5 CPU clocks (one clock is equal to two phases). Data is provided to either the CPU core or the L1 cache in 1.5 CPU clocks. L2 cache address generation occurs before the cycle starts. A total of 7.5 clocks are required for a cache line fill, as shown is figure 29.

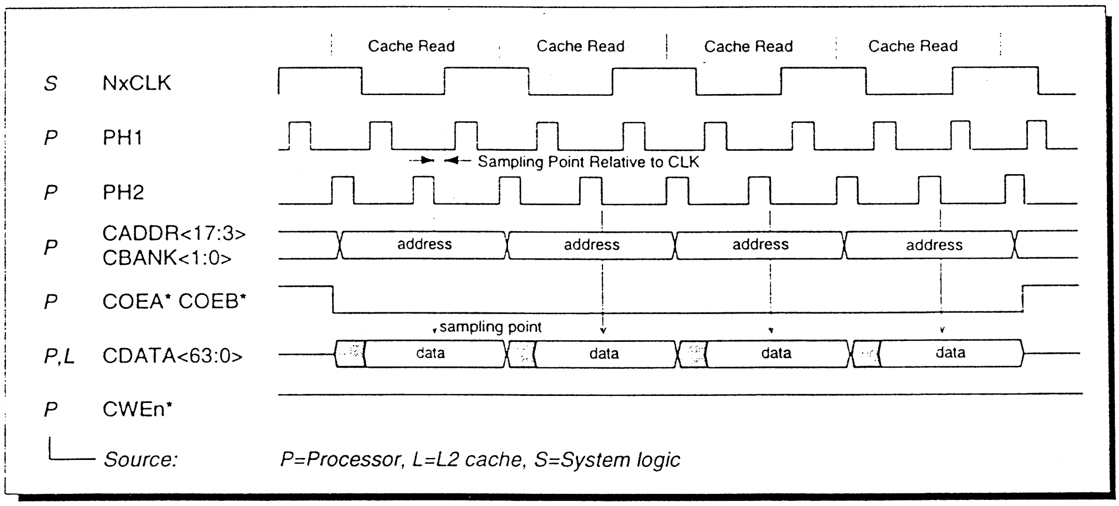

Figure 29 Level-2 to Level-1 Asynchronous Cache Line Fill.

Level-2 Synchronous SRAM Accesses

The type of SRAMs required for synchronous mode are "Synchronous Flow Through" with wide I/O (32 or 36 bits). A single clocking pin, CKMODE is used to initiate the read/write operations. At the rising edge of CKMODE, all addresses, write-enables, chip selects and data are registered within the SRAM. It is assumed that new signals can be applied to the SRAMs prior to data out valid. Read data is sampled on the next rising edge of CKMODE (approximately two PH2 clocks later). A dead cycle for bus turn around time is provided during read followed by write cycles (approximately one PH2 clock). Figure 30 shows the signal relationships for the synchronous SRAM mode.

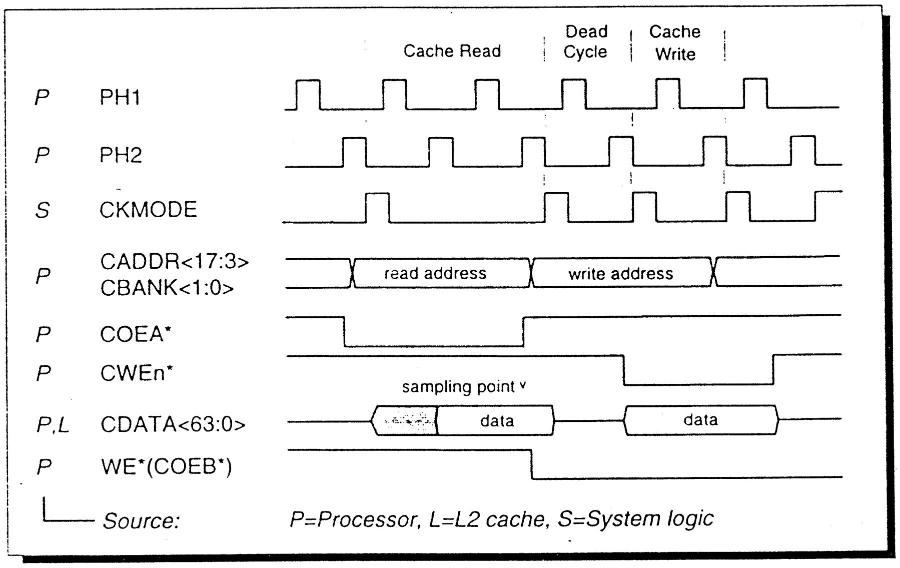

Figure 30 Level-2 Synchronous Cache Read and Write cycles

NexBus and NexBus^5 Arbitration and Address Phase

Processor operations on NexBus/NexBus^5 may or may not begin with arbitration for the bus. To obtain the bus, the processor asserts NREQ*, LOCK*, and/or AREQ* to the arbiter, which responds to the arbitration winner with GNT*. Automatic re-grant occurs when the arbiter holds GNT* asserted at the time the processor samples it, in which case the processor need not assert NREQ*, LOCK*, or AREQ* and can immediately begin its operation.

NREQ*, when asserted, remains active until GNT* is received from the arbiter. In systems using the systems logic that interfaces directly to NexBus, NREQ* is typically treated the same as AREQ*; when NexBus control is granted, control of all other buses is also granted at the same time.

LOCK* is asserted during sequences in which multiple bus operations should be performed sequentially and uninterrupted. This signal is used by the arbiter to determine the end of such a sequence. Cache-block fills are not locked; they are implicitly treated as atomic reads. Arbiters may allow a master on another system bus to intervene in a locked NexBus transaction. To avoid this, the processor asserts AREQ*. LOCK* is typically software-configured to be asserted for read-modify-writes and explicitly locked instructions.

AREQ* is asserted to gain control of the NexBus^5 or any other buses supported by the system. This signal always remains active until GNT* is received.

When GNT* is received, the processor places the address of a qword (for memory operations) on NxAD<31:3> or the address of a dword (for I/O operations) on NxAD<15:2>. It drives status bits on NxAD<63:32> and asserts its ALE* signal to assume bus mastership and to indicate that there is valid address on the bus. The processor asserts ALE* for only one bus clock. The slave uses the GALE signal generated by system logic to enable the latching of address and status from the NexBus^5.

NexBus Basic Operations

The Nx586 supports two local bus interfaces, NexBus and NexBus^5. NexBus is considered a true CPU or processor local bus. Where as, NexBus^5 is a NexGen proprietary system bus. During RESET* active, the XCVERE* pin is sampled for the local bus mode. XCVERE* determines what type of bus is generated by the processor. When pulled high, the Nx586 will generate the NexBus standard which requires external transceivers to connect the processor to the NexBus^5 system bus. Figure 31 and 32 show the NexBus transceiver control signals for basic QWORD Read and Write operations. AD<63:0> is the multiplexed NexBus processor local bus while NxAD<63:0> is the multiplexed NexBus^5 system local bus.

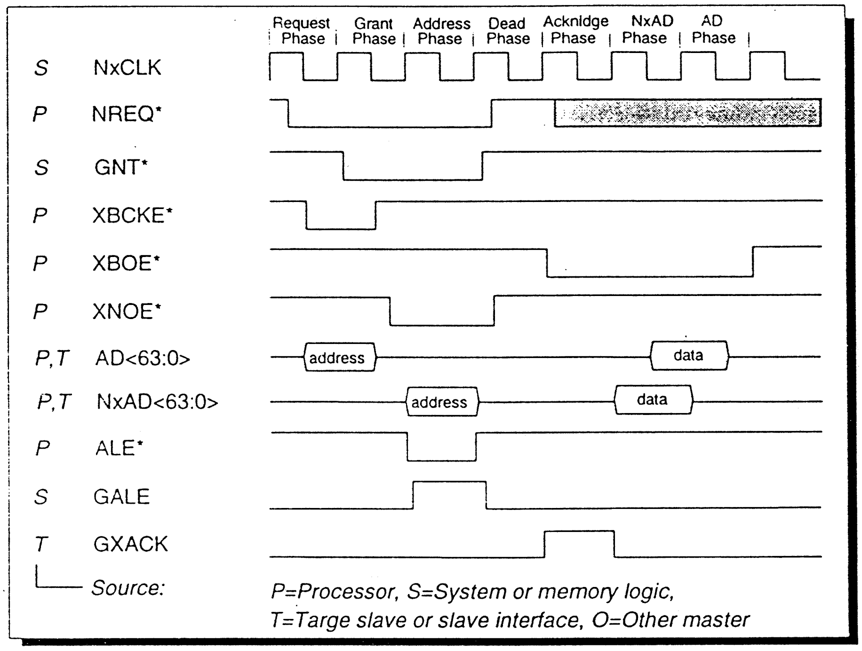

Figure 31 Fastest NexBus Single-Qword Read

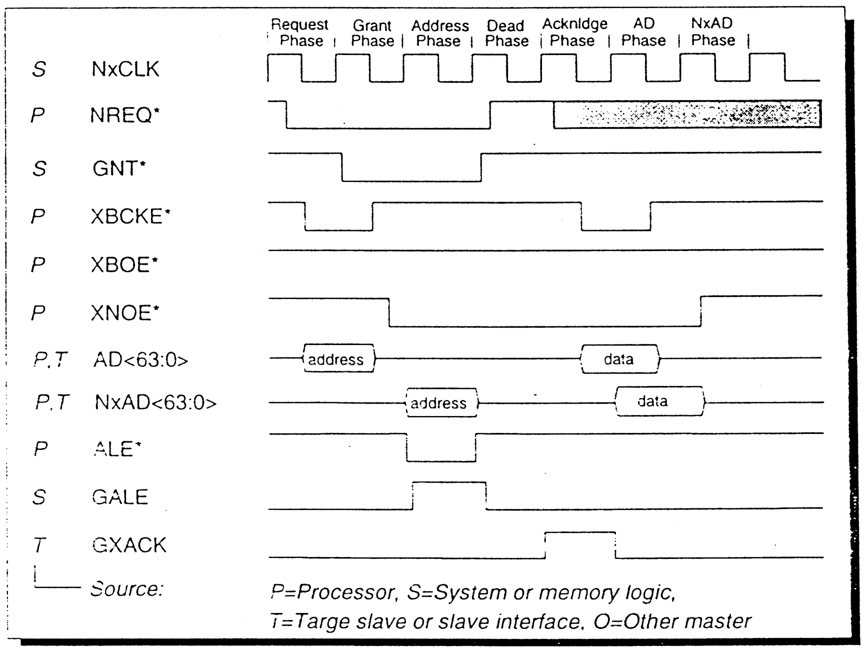

Figure 32 Fastest NexBus Single-Qword Write

NexBus5 Single-Qword Memory Operations

Figure 33 shows the fastest possible NexBus5 single-qword read. The notation regarding *Source* indicates the logic that originated the signal as an output. In this figure and others to follow, the source of group signals (such as GXACK) is shown with additional symbols indicating the device or logic that output the originally activating signal. For example, the source of the GXACK signal is shown as "S,P", which means that system logic (S) generated GXACK but that the processor (P) caused this by generating XACK*. In some timing diagrams later in this section, bus signals take on different names as outputs cross buses through transceivers or are ORed in group-signal logic; in these cases, the source of the signals is shown with additional symbols indicating the logic that originally output the activating signals.

The data phase of a fast single-qword read starts when the slave responds to the processor's request by asserting its XACK* signal. The processor samples the GXACK and GXHLD signals from system logic to determine when data is placed on the bus. The processor then samples the data at the end of the bus clock after GXACK is asserted and GXHLD is negated. The operation finishes with an idle phase of at least one clock.

This protocol guarantees the processor and other caching devices enough time to recognize a modified cache block and to assert GDCL in time to cancel a data transfer. A slave may not assert XACK* until the second clock following GALE. However, the slave must always assert XACK* during or before the third clock following GALE, since otherwise the absence of an active GXACK indicates to the systems logic interface between the NexBus5 and other system buses (called the *alternate-bus interface)* that the address must reside on the other system bus. In that case, the systems logic interface to that other bus assumes the role of slave and asserts GXACK.

Figure 33 shows when GBLKNBL may be asserted. If appropriate, the slave must assert GBLKNBL no later than it asserts XACK*, and it must keep GBLKNBL asserted until it negates XACK*. It must negate GBLKNBL at or before it stops placing data on the bus. Although not shown, OWNABL must also be valid (either asserted or negated) whenever GXACK is asserted. In the example shown in Figure 34, the slave asserts GXACK at the latest allowable time, thereby effectively inserting one wait state. The slave may or may not drive the NxAD<63:0> signals during the wait states. The processor will not drive them during the data phase of a read operation.

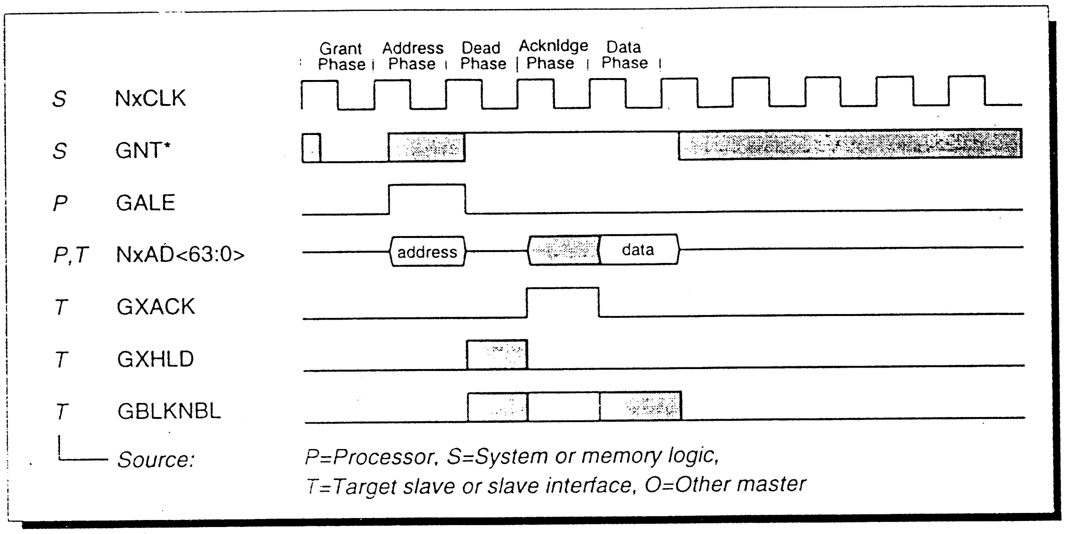

Figure 33 Fastest NexBus5 Single-Qword Read

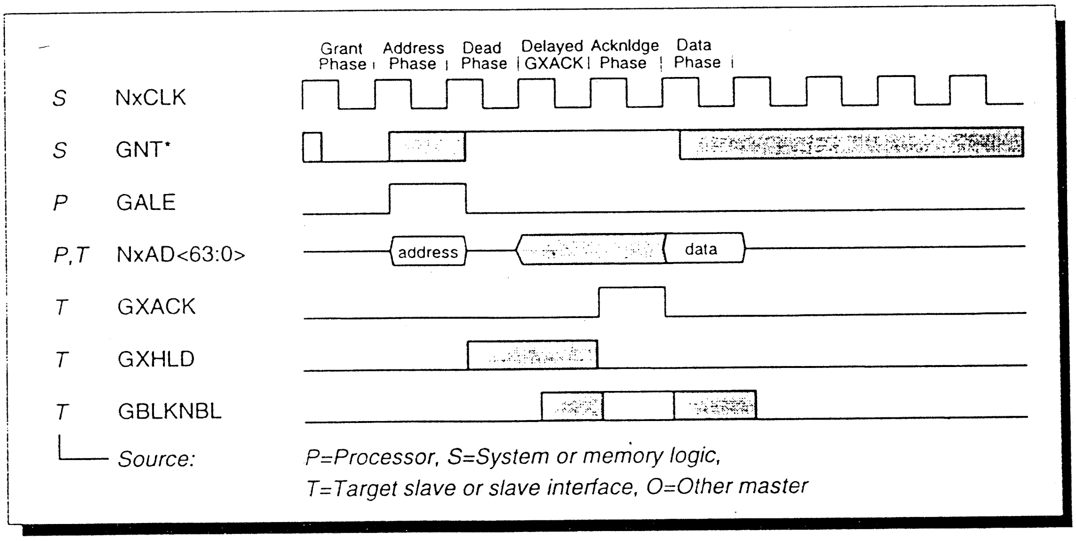

Figure 34 Fast NexBus5 Single-Qword Read with a delayed GXACK

If the slave is unable to supply data during the next clock after asserting XACK*, the slave must assert its XHLD* signal at the same time. Similarly, if the processor is not ready to accept data in the next clock it asserts its XHLD* signal. The slave supplies data in the clock following the first clock during which GXACK is asserted and GXHLD is negated. The processor strobes the data at the end of that clock. A single-qword read with wait states is shown in Figure 35 and 36. For such an operation, the slave must negate XACK* after a single clock during which GXACK is asserted and GXHLD is negated, and it must stop driving data onto the bus one clock thereafter. The processor does not assert XHLD* while GALE is asserted, nor may either party to the transaction assert XHLD* after the slave negates GXACK. In the example shown in Figure 35, the slave asserts GXACK at the latest allowable time, thereby inserting one wait state, and GXHLD is asserted for one clock to insert an additional wait state. The slave may or may not drive the NxAD<63:0> signals during the wait states. The processor will not drive them during the data phase of a read operation.

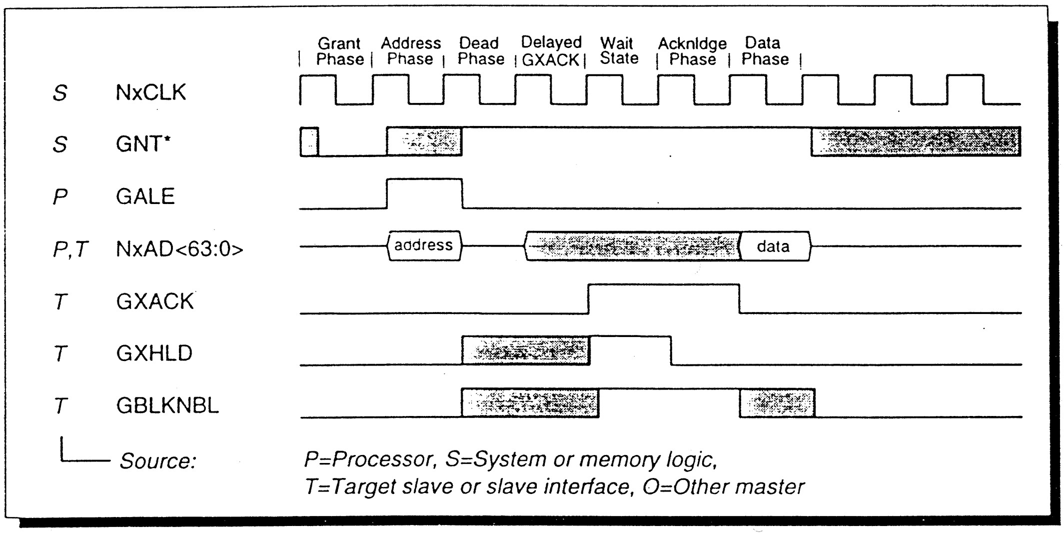

Figure 35 NexBus5 Single-Qword Read with Wait States using a delayed GXACK

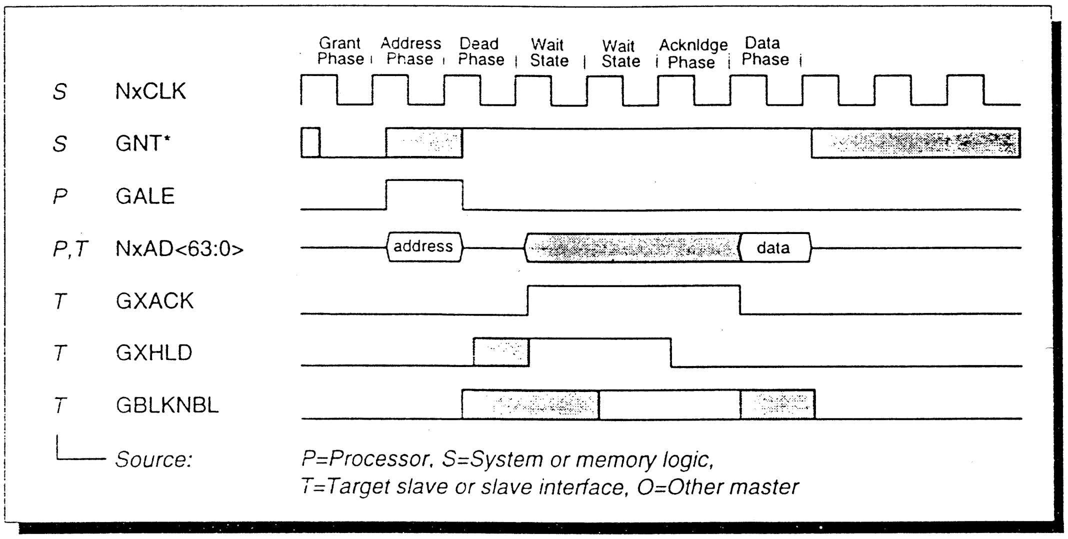

Figure 36 NexBus[5] Single-Qword Read with Wait States using GXHLD only

A single-qword write operation is handled similarly. Figure 37 illustrates the fastest write operation possible. Figure 38 shows a single-qword write with wait states. After the bus is granted, the processor puts the address and status on the bus and asserts ALE*. As in the read operation, the slave must assert its XACK* signal during either the second or third clock following the assertion of GALE. If the slave is not ready to strobe the data at the end of the clock following the assertion of GXACK, it must assert its XHLD* signal. The processor places the data on the bus in the clock after the assertion of GXACK, which may be as soon as the third clock following the assertion of GALE. The slave samples GXHLD to determine when the data is valid. The processor will drive data as soon as it is able, and it continues to drive the data for one (and only one) clock after the simultaneous assertion of GXACK and negation of GXHLD. As in the read operation, the slave's XACK* is asserted until the clock following the trailing edge of GXHLD.

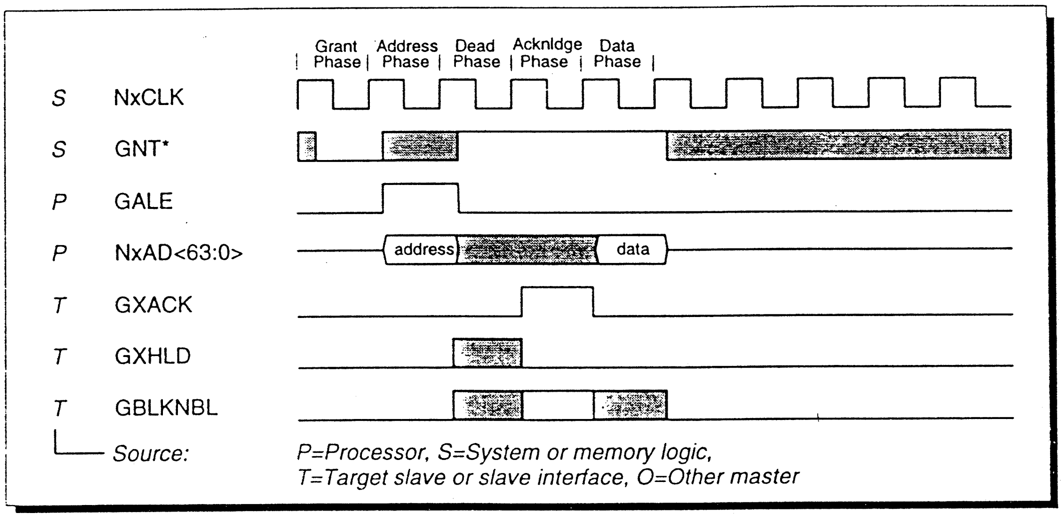

Figure 37 Fastest NexBus5 Single-Qword Write

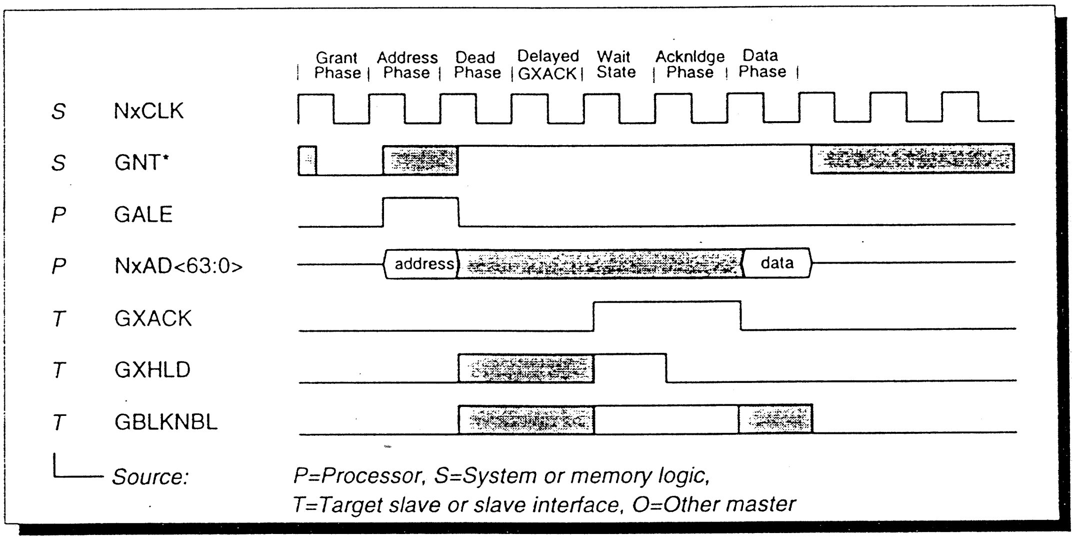

Figure 38 NexBus5 Single-Qword Write With Wait States

NexBus^5 Cache Line Memory Operations

The processor performs cache line fill or block operations with memory at a much higher bandwidth than the single-qword operations described in the previous section. Block operations, both reads and writes, are done only in four-qword increments (32-bytes). All cache line reads are cache fills.

Cache line reads and writes are indicated by the assertion of BLKSIZ* during the address/status phase of the bus operations, as previously defined for single-qword operations.

A cache line operation consists of a single address phase followed by a multi-transfer data phase. The data transfer may begin with *any* qword in the block, as indicated by the address bits, but it then proceeds through additional qwords of the specified contiguous data in any order.

NexBus^5 I/O Operations

I/O operations on the NexBus^5 are performed exactly like single-qword reads and writes, with three exceptions. First, the I/O address space is limited to 64K bytes. Second, the 16-bit I/O address is broken into two fields: fourteen address bits and two byte-enable bits. I/O addresses do not use BE<7:2>* (which must be set to all 1's) but instead specify a quad address on NxAD<2>. Third, data is always transferred on NxAD<15:0>, and NxAD<63:16> is undefined during the data transfer phase of an I/O operation.

I/O operations are indicated by driving 010 (data read) and 011 (data write) on NxAD<48:46> and all zeros on NxAD<31:16> when GALE is asserted. I/O space is always non-cacheable, so a slave should never assert GBLKNBL when responding to an I/O operation.

NexBus^5 Interrupt-Acknowledge Sequence

When an interrupt request is sensed by external interrupt-control logic, the request is signaled to the processor by the control logic, the processor acknowledges the interrupt request (during which sequence the controller passes the interrupt vector), and the processor services the interrupt as specified by the vector. The hardware mechanism is described above in the *Hardware Architecture* chapter.

An interrupt-acknowledge sequence, shown in Figure 39, consists of two back-to-back locked reads on NexBus^5, where the operation type (NxAD<48:46>) is 000 and the byte enable bits BE<7:0>* = 11111110. The first (synchronizing) read is used latch the state of the interrupt controller. It is indicated by NxAD<2> = 1 (I/O-byte address 4). The second read is used to transfer the 8-bit interrupt vector on NxAD<7:0> to the processor, which uses it as an index to the interrupt service routine. This read is indicated by NxAD<2> = 0 (I/O-byte address 0). During these two reads only the least significant bit of the address field is driven to a valid state. The most significant bits are undefined. After the interrupt is serviced, the request is cleared and normal processing resumes.

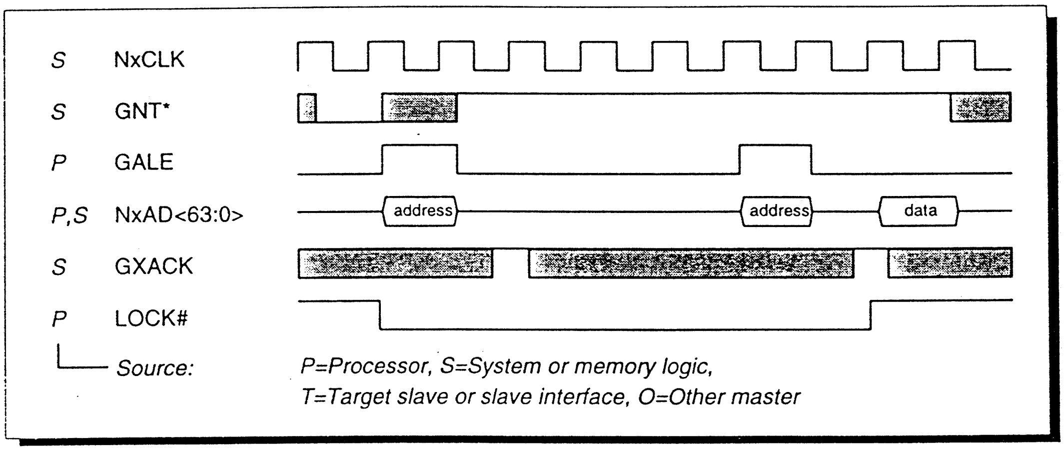

Figure 39 Interrupt Acknowledge Cycle

NexBus5 Halt and Shutdown Operations

Halt and shutdown operations are signaled on the NexBus5 by driving 001 on NxAD<48:46> during the address/status phase, as shown in Figure 40. The halt and shutdown conditions are distinguished from one another by the address that is simultaneously signaled on the byte-enable bits, BE<7:0>* on NxAD<39:32>. The processor does not generate a data phase for these operations.

Type of Bus Cycle	*NxAD<48> M/IO**	*NxAD<47> D/C**	*NxAD<46> W/R**	*NxAD<39:32> BE<7:0>**	*NxAD<31:3>*	*NxAD<2>*
Halt	0	0	1	11111011	undefined	0
Shutdown	0	0	1	11111110	undefined	0

Figure 40 Halt and Shutdown Encoding

For the halt operation, the processor places an address of 2 on the bus, signified by BE<7:0>* bits (NxAD<39:32>) = 11111011. NxAD<2> = 0 and NxAD<31:3> are undefined. After this, the processor remains in the halted state until NMI*, RESETCPU*, or RESET* becomes active.

For the shutdown operation, the processor places an address of 0 on the bus, signified by BE<7:0>* bits (NxAD<39:32>) = 11111110. NxAD<2> = 0 and NxAD<31:3> are undefined. An external system controller should decode the shutdown cycle and assert RESETCPU*. After this, the processor performs a soft reset, RESETCPU*; that is, the processor is reset, but the memory contents, including modified cache blocks, are retained.

Because the Nx586 processor has a 64-bit data bus rather than a 32-bit data bus, eight total byte-enable bits (BE<7:0>*) are specified for quadword wide bus.

Obtaining Exclusive Use Of Cache Blocks

The processor can obtain ownership of a cache block either *preemptively* or *passively*. Preemptive ownership is gained by asserting OWN* during the address/status phase of a read or write operation. Whenever the processor needs to write a cache block that is either cached in the shared or invalid state, it performs a preemptive read-to-own operation by asserting OWN* during a single-qword write or four-qword block read.

Passive ownership is normally gained when the processor performs a block read, because other NexBus5 caching devices must snoop block reads. If any part of a block addressed by the processor's read operation resides in another NexBus5 device's cache, regardless of state, that device asserts SHARE* after the assertion of GALE but not later than the clock during which the first qword of the block is transferred. SHARE* remains asserted through the entire data transfer. If the processor sees GSHARE negated during a block read when it samples the first qword of the block, it knows that it has the only copy. It can therefore cache the block in the exclusive state rather than the shared state, if and only if OWNABL is asserted by system logic.

If another NexBus5 caching device is unable to meet this timing in the fastest possible case, it must assert XHLD* to delay the operation until it is able to perform the cache check. While it is possible to put a caching device on NexBus5 that is unable to check its cache and report SHARE* correctly, but instead always asserts SHARE*, this has a very negative effect on system efficiency. It is also possible to design a device that invalidates its cache block during any block read hit, in which case only the efficiency of that one device is impaired.

If the processor addresses a non-cacheable block on a system bus other than NexBus5, the systems logic interface between the NexBus5 and the other system bus (called the *alternate-bus interface)* must indicate this by negating GBLKNBL, and it may not perform block reads or writes to such a block. If the block on the other bus is cacheable, it can only be cached in the shared state, since standard system buses (such as VL bus and ISA bus) do not support the MESI caching protocol, and it is not possible to cache their memory addresses in the exclusive state.

The OWNABL signal from system logic is used to indicate cacheability of locations on other system buses. Whenever OWNABL is negated during a bus operation, the processor will not cache the block in the exclusive state even if the processor asserted OWN*; instead, it may cache the block in the shared state if other conditions permit it.

GBLKNBL and GSHARE must be asserted by system logic at the same time that OWNABL is negated. The timing of these three signals is identical: they should be valid whenever GXACK is asserted. They may be (but need not be) asserted ahead of XACK*, and may (but, except for GSHARE, need not) be held one clock after the negation of XACK*. This timing differs from that of GSHARE, since when OWNABL is asserted GSHARE is not required to be valid until the clock following the negation of GXHLD—i.e., coincident with the data transfer.

NexBus5 Intervenor Operations

The examples given above assume that the addressed data does not reside in a modified cache block. When an operation by another NexBus5 master results in a cache hit to a modified block in the processor, the processor intervenes in the operation by asserting DCL*. The timing for DCL* is the same as that for SHARE*: the NexBus master samples GDCL on the same clock in which it samples NexBus5 data. An asserted GDCL indicates to the master that data cached by the processor is modified. To meet the fastest timing requirements, the processor asserts DCL* no later than the third clock following the assertion of GALE. If a MESI write-back caching device is unable to determine in a timely manner whether a transaction hits in its cache, it must assert XHLD* to delay the transfer.

If a block write operation by another master hits a modified cache block in the processor, the processor does not assert DCL*, since such a block write replaces all of a cache block. Instead, the processor invalidates the block.

An addressed slave that sees GDCL asserted during the first qword transfer of an operation must abort the operation by negating GXACK. It may then perform a block write-back starting with the first qword. Immediately after the operation is completed, as determined by the negation of GXACK, the NexBus5 Arbiter must grant the bus to the intervenor by asserting GNT*. The arbiter must not grant the bus to any other requester, even if the previous master has asserted AREQ* and/or LOCK*, because DCL* has absolutely the highest priority. Upon seeing GNT* asserted, the intervenor (whether the processor or another master) immediately updates the memory by performing a block write, beginning at the qword address specified in the original operation. The intervenor negates DCL* before performing the first data transfer, but not before it asserts ALE*. During this memory update, the master must sample the data it requested (if the operation was a read) as it is sent to memory on NexBus5 by the intervenor. If the master is not ready to sample the data, it can assert XHLD*, as can both the intervenor and the slave; all three parties to the operation examine GXHLD to synchronize the data transfer.

Modified Cache-Block Hit During Single-Qword Operations

During single-qword reads that hit in a modified cache block, the NexBus5 sequence looks like a normal single-qword read from the memory followed by a block write by the intervenor. Figure 41 illustrates the timing. The fastest time is shown for the operation, while both the fastest and slowest possible times are shown for the leading edge of GDCL. For a slow device intervening in a fast operation, GDCL is available to be sampled on the same clock as the first qword of data is available.

In Figure 41, two sources are shown for GALE and NxAD<63:0>, and one source (Sp) has a subscript. The source is the chip or logic that outputs the signal. The subscript for the source indicates the chip or logic that originally caused the change in the signal.

During single-qword writes, the master with the modified cache block asserts DCL* to indicate that the single write will be followed by a block write. If the single write included only some of the bytes of the qword, the intervenor records this fact, and during the subsequent block write it outputs byte-enable bits indicating the other bytes of the qword. For example, if the byte-enable bits of the single write were 00000111, the intervenor outputs 11111000. In other words, the intervenor updates only those bytes that were not written by the master. Except for such intervening write-back operations, block writes must have all byte-enable bits asserted (00000000). During block write-backs, byte-enable bits apply only to the first qword, so all bytes of the final three qwords are written.

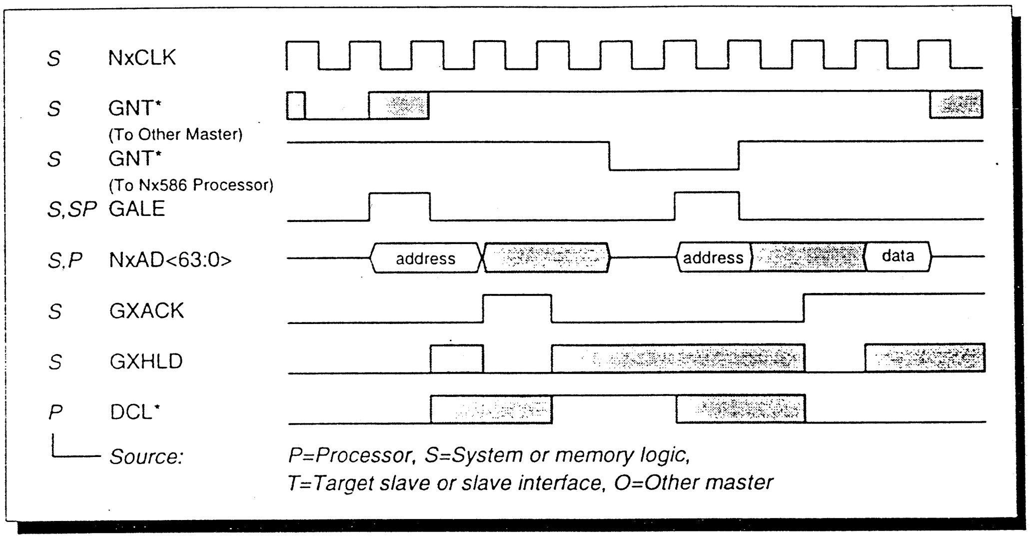

Figure 41 NexBus5 Single-Qword Read Hits Modified Cache Block

Modified Cache-Block Hit During Four-Qword (Block) Operations

As described above for single-qword operations, a block read by another NexBus5 master may hit a modified cache block in the processor. When this happens, the processor responds exactly as for a single-qword operation: it asserts DCL*, waits for the assertion of GNT* following the negation of GXACK, and proceeds with a block write-back. It writes the entire four-qword block back to memory. The original bus master must sample the data in this second block operation while it is transferred to memory. The master may insert wait states by asserting XHLD*. Since the processor, as intervenor, begins its write-back with the address requested by the master, if the original block read is a four-qword operation, the master can intercept the data as it is transferred to memory and find it in the expected order.

Block writes can hit in a modified or exclusive cache block only if the operation was initiated by the DMA action of a disk controller, not by the processor. Since only complete block writes are permitted, no write-back is required and the processor invalidates its cache block.

THIS PAGE INTENTIONALLY LEFT BLANK