Accelerated Graphics Port Interface Specification

Revision 1.0

Intel Corporation

July 31, 1996

Accelerated Graphics Port Interface Specification

Table of Contents

List of Figures

List of Tables

Revision History

Revision	Revision History	Date
1.0	Initial Released Version	7/31/96

1. Introduction

The Accelerated Graphics Port (AGP or A.G.P.) is a high performance, component level interconnect targeted at 3D graphical display applications and is based on a set of performance extensions or enhancements to PCI. This document specifies the A.G.P. interface, and provides some design suggestions for effectively using it in high performance 3D graphics display applications.

1.1 Motivation.

In general, 3D rendering has a voracious appetite for memory bandwidth, and continues to put upward pressure on memory footprint as well. As 3D hardware and software become more pervasive, these two trends are likely to accelerate, requiring high speed access to ever larger amounts of memory, thus raising the bill of material costs for 3D enabled platforms. Containing these costs while enabling performance improvements is the primary motivation for the A.G.P.. By providing up to an order of magnitude bandwidth improvement between the graphics accelerator and system memory, some of the 3D rendering data structures may be effectively shifted into main memory, relieving the pressure to increase the cost of the local graphics memory.

Texture data are the first structures targeted for shifting to system memory for four reasons:

1. Textures are generally read only, and therefore do not have special access ordering or coherency problems.
2. Shifting textures balances the bandwidth load between system memory and local graphics memory, since a well cached host processor has much lower memory bandwidth requirements than does a 3D rendering engine. Texture access comprises perhaps the largest single component of rendering memory bandwidth (compared with rendering, display and Z buffers), so avoiding loading or caching textures in graphics local memory saves not only this component of local memory bandwidth, but also the bandwidth necessary to load the texture store in the first place. Furthermore, this data must pass through main memory anyway as it is loaded from a mass store device..
3. Texture size is dependent upon application quality rather than on display resolution, and therefore subject to the greatest pressure for growth.
4. Texture data is not persistent; it resides in memory only for the duration of the application, so any system memory spent on texture storage can be returned to the free memory heap when the application concludes (unlike display buffers which remain in persistent use).

Other data structures may be moved to main memory but texture data is the biggest win.

Reducing costs by moving graphics data to main memory is the primary motivation for the A.G.P., which is designed to provide a smooth, incremental transition for today’s PCI based graphics vendors as they develop higher performance components in the future.

1.2 Relationship to PCI

The A.G.P. interface specification uses the 66 MHz PCI *(Revision 2.1)* specification as an operational baseline, and provides three significant performance extensions or enhancements to the PCI specification which are intended to optimize the A.G.P. for high performance 3D graphics applications. These A.G.P. extensions are NOT described in, or required by, the PCI specification (*Rev. 2.1*). These extensions are:

- Deeply pipelined memory read and write operations, fully hiding memory access latency.
- Demultiplexing of address and data on the bus, allowing almost 100% bus efficiency.
- AC timing for 133 MHz data transfer rates, allowing for real data throughput in excess of 500 MB/sec..

These enhancements are realized through the use of "sideband" signals. The PCI Specification has not been modified in any way, and the A.G.P. interface specification has specifically avoided the use of any of the "reserved" fields, encodings, pins, etc. in the PCI Specification. The intent is to utilize the PCI design base, while providing a range of graphics-oriented performance enhancements with varying complexity/performance tradeoffs available to the component provider.

A.G.P. neither replaces nor diminishes the necessity of PCI in the system. This high speed port (A.G.P.) is physically, logically, and electrically independent of the PCI bus. It is an additional connection point in the system, as shown in Figure 1-1. It is intended for the exclusive use of visual display devices; all other I/O devices will remain on the PCI bus. The add-in slot defined for A.G.P. uses a new connector body (for electrical signaling reasons) which is not compatible with the PCI connector; PCI and A.G.P. boards are **not** mechanically interchangeable.

The A.G.P. interface specification was developed by Intel, independent of the PCI Special Interest Group, and has been neither reviewed nor endorsed by that group. It is intended to encourage innovation in personal computer graphics technology and products.

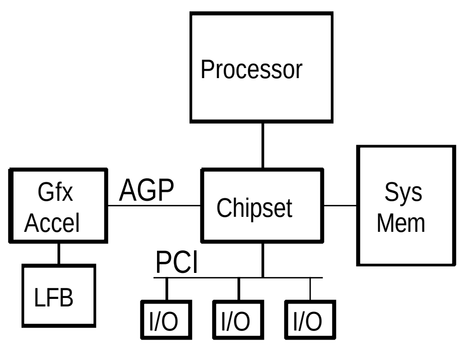


Figure 1-1 System Block Diagram: A.G.P. and PCI Relationship

2. Architectural Context and Scope

This chapter is intended to provide an architectural context for the actual specification of the Accelerated Graphics Port (A.G.P. or AGP) interface, hopefully motivating the design, and making details easier to understand. As such, this chapter does not define any interface requirements, and therefore is not necessary for implementing the A.G.P. interfaces or A.G.P. compliant parts or systems. It also is intended to set the technical scope of the specification in some areas by providing examples of issues beyond the purview of the formal interface specification.

2.1 Two Usage Models: “Execute” & “DMA”

There are two primary A.G.P. usage models for 3D rendering, that have to do with how data are partitioned and accessed, and the resultant interface data flow characteristics. In the “DMA” model, the primary graphics memory is the local memory associated with the accelerator, referred to as ‘local frame buffer’. 3D structures are stored in system memory, but are not used (or “executed”) directly from this memory; rather they are copied to primary (local) memory, to which the rendering engine’s address generator makes it’s references. This implies that the traffic on the A.G.P. tends to be long, sequential transfers, serving the purpose of bulk data transport from system memory to primary graphics (local) memory. This sort of access model is amenable to a linked list of physical addresses provided by software (similar to operation of a disk or network I/O device), and is generally not sensitive to a non-contiguous view of the memory space.

In the “execute” model, the accelerator uses both the local memory and the system memory as primary graphics memory. From the accelerator’s perspective, the two memory systems are logically equivalent; any data structure may be allocated in either memory, with performance optimization as the only criteria for selection. In general, structures in system memory space are not copied into the local memory prior to use by the accelerator, but are “executed” in place. This implies that the traffic on the A.G.P. tends to be short, random accesses, which are not amenable to an access model based on software resolved lists of physical addresses. Since the accelerator generates direct references into system memory, a contiguous view of that space is essential. But, since system memory is dynamically allocated in random 4K pages, it is necessary in the “execute” model to provide an address mapping mechanism that maps random 4K pages into a single contiguous, physical address space.

The A.G.P. supports both the “DMA” and “execute” models. However, since a primary motivation of the A.G.P. is to reduce growth pressure on local memory, the “execute” model is the design center. Consistent with that emphasis, this interface specification requires a physical-to-physical address re-mapping mechanism which insures the graphics accelerator (A.G.P. master) will have a contiguous view of graphics data structures dynamically allocated in system memory.

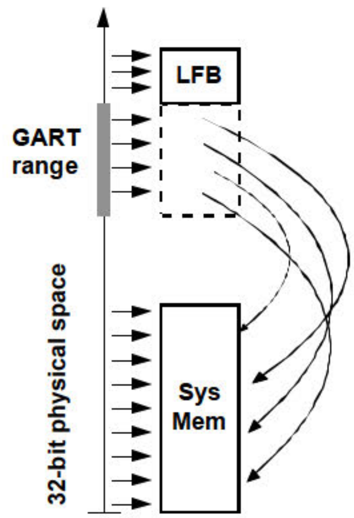


Figure 2-1 Graphics Address Re-mapping Function

This address re-mapping applies only to a single, programmable range of the system physical address space, as shown in Figure 2-1. The 32-bit physical address space shown is common to all system agents. Addresses falling in this range are re-mapped to non-contiguous pages of physical system memory. All addresses not in this range are passed through without modification, and map directly to main system memory, or to device specific ranges, such as the graphics local frame buffer memory shown here.

Re-mapping is accomplished via a memory-based table called the Graphics Address Re-mapping Table (GART), which is set up and maintained by the mapping API described in chapter 6, and used ("walked") by the core logic to perform the re-mapping. In order to avoid compatibility issues and allow future implementation flexibility, this mechanism is specified at a software (API) level. In other words, the actual GART table format is not specified; rather it is abstracted to the API by a HAL or mini-port driver that must be provided with the core logic. While this API does not constrain the future partitioning of re-mapping hardware, the re-mapping function will initially be implemented in the chipset or core logic. *Note: this re-mapping function should not be confused in any way with the system address translation table mechanism. While some of the concepts are similar, these are completely separate mechanisms which operate independently, under control of the operating system.*

2.2 A.G.P. Queuing Models

Both A.G.P. bus transactions and PCI bus transactions may be run over the A.G.P. interface. An A.G.P. compliant device may transfer data to system memory using either A.G.P. transactions or PCI transactions. The core logic can access the A.G.P. compliant master (graphics) device only with PCI transactions. Traffic on the A.G.P. interface may consist of a mixture of interleaved A.G.P. and PCI transactions.

The access request and data queue structures are illustrated in Figure 2-2.

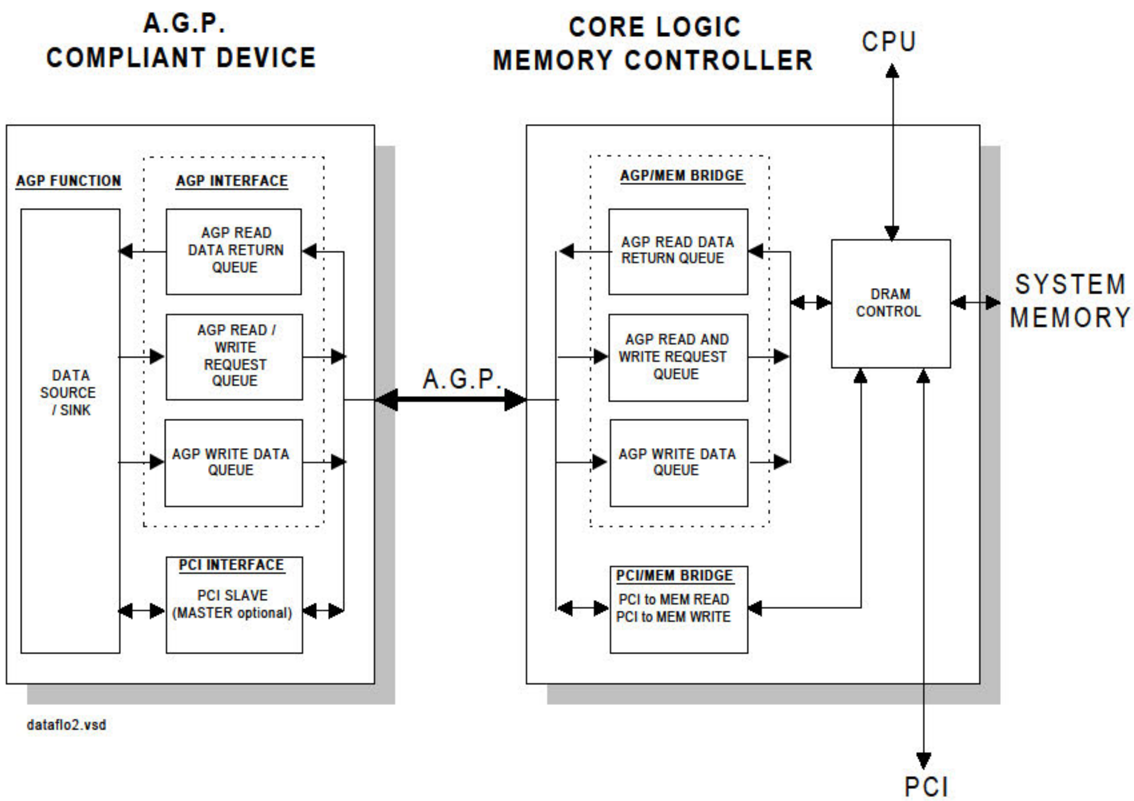


Figure 2-2 A.G.P. Access Queuing Model

A.G.P. transactions are run in a split transaction fashion where the request for data transfer is "disconnected" from the data transfer itself. The A.G.P. compliant device initiates an A.G.P. transaction with an access request. The core logic responds to the access request by directing the corresponding data transfer at a later time. The fact that the access requests are separated from the data transfers allows the A.G.P. compliant device to issue several access requests in a pipelined fashion while waiting for the data transfers to occur. Pipelining access requests results in having several read and/or write requests outstanding in the core logic's ***request queue*** at any point in time. The request queue is divided into high priority and low priority sub-queues, each of which deal with respective accesses according to separate priority and ordering rules. The A.G.P. compliant device tracks the state of the request queue in order to limit the number of outstanding requests and identify data transactions.

The core logic processes the access requests present in its request queue. Read data will be obtained from system memory and returned at the core chipset's initiative via the ***read data return queue***. Write data will be provided by the A.G.P. device at the core logic's direction when space is available in the core logic's ***write data queue***. Therefore A.G.P. transaction traffic will generally consist of interleaved access requests and data transfers.

All PCI transactions on the A.G.P. also have their own queues, separate from the A.G.P. transaction queues. Each queue has its own access and ordering rules. Not shown in figure 2-2 is the core logic queue which handles processor accesses directly to the A.G.P. compliant device, all of which are executed as non-pipelined PCI bus transactions.

2.3 Performance Considerations

Deep pipelining capability allows the A.G.P. to achieve a total memory READ throughput equal to that possible for memory WRITES[1]. This capability, coupled with optional higher transfer rates and address de-multiplexing allows a full order of magnitude increase in memory read throughput over today's PCI implementations. However, many typical desktop platforms will not have sufficient total memory performance to allow full utilization of the A.G.P. capabilities, and ultimately platform issues are likely to predominate in the upper performance limit deliverable through the A.G.P. This makes it very difficult to provide as part of this interface specification a single set of performance guarantees or targets the graphics designers can depend upon. It is clear that in order to optimize a graphics design for the most effective use of the A.G.P., it will need to be targeted at a specific subset of platforms and/or core logic devices.

In an attempt to provide the best possible information for such an optimization, this interface specification defines a few common performance parameters that may be of general interest, and recommends that core logic vendors and/or OEMs provide these parameters for their systems to respective graphics IHVs.

The following are the basic parameters that each core logic set and/or system implementation should provide as a performance baseline for IHVs targeting that platform.

- **Guaranteed Latency**: a useable worst case A.G.P. memory access latency via the HIGH PRIORITY QUEUE, as measured from the clock on which the request (**REQ#**) signal is asserted until the first clock of data transfer. Assumptions: no outstanding A.G.P. requests (pipeline empty); no wait states or control flow asserted by the graphics master - master is ready to transfer data on any clock (inserting *n* clocks of control flow may delay response by more than *n* clocks).

- **Typical Latency**: the typical A.G.P. memory access latency via the LOW PRIORITY QUEUE, as measured from the clock on which the request (**REQ#**) signal is asserted until the first clock of data transfer. Assumptions: no outstanding A.G.P. requests (pipeline empty); no wait states or control flow asserted by the graphics master - master is ready to transfer data on any clock (inserting *n* clocks of control flow may delay response by more than *n* clocks).

- **Mean bandwidth**: deliverable A.G.P. memory bandwidth via the LOW PRIORITY QUEUE, averaged across ~10mS (one frame display time). Assumptions: no accesses to the high priority queue; graphics master maintains optimal pipeline depth of _x_; average access length of _y_; no wait states or control flow asserted by the graphics master.

2.4 Platform Dependencies

Due to the close coupling of the A.G.P. and main memory subsystem, there are several behaviors of the A.G.P. that will likely end up being platform dependent. While the objective of any specification is to minimize such differences, it is apparent that several are probable among A.G.P. compliant core logic and platform implementations. This should not, however, have as much impact as it would in other buses for two reasons:

[1] Memory read throughput on PCI is about half of memory write throughput, since memory read access time is visible as wait states on this unpipelined bus.

- The A.G.P. is a point-to-point connection, intended for use by a 3D graphics accelerator only, and,
- Due to performance issues (section 2.3), A.G.P. compliant graphics devices will likely need to be optimized to a specific subset of platform or core logic implementations anyway.

The purpose of this section is to identify by example some of the areas where behavioral differences are likely, and accordingly establish the scope of this interface specification.

As one example of potential variation, consider the two core logic architectures shown in Figure 2-3. An integrated approach, typical of desktop and volume systems, is shown on the left, and a symmetric multiprocessor partitioning, typical of MP servers, is shown on the right.

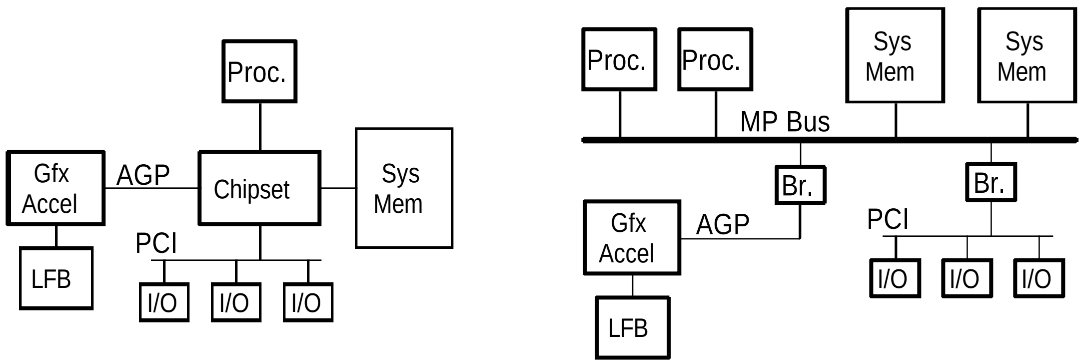


Figure 2-3 Different Core Logic Architectures

The following items are examples of areas where behavioral differences between these or other implementations could well occur.

GART Implementation: for various reasons, different systems may opt for different GART table implementations and layouts. This, however, is not visible since the actual table implementation is abstracted to a common API by the HAL or mini-port driver supplied with the core logic.

Coherency with processor cache: due to the high potential access rate on the A.G.P., it is not advisable from a performance perspective to snoop all accesses. While an attribute in the Graphics Address Re-mapping Table could indicated pages in which snoops were needed, doing this selectively forces a split and later re-combining of the access queue in the integrated chipset, which is very costly with marginal benefit (there are reasonable software solutions to this problem). On the other hand, the MP bus in the MP system could well deal with selective snoops very easily. As a result, processor cache snooping may be chipset dependent. This interface specification does not require snooping, and in general it must not be counted on. Note that all PCI bus transactions are by definition coherent, and always snoop the processor cache.

Bus-to-bus traffic capability: bus masters on either the A.G.P. or the PCI bus will routinely access system memory. However, it is possible to also address targets on the other bus or port, which effectively requires a PCI-to-PCI bridge in the integrated chipset. Pushing WRITES through this bridge is fairly simple, whereas pushing READS through requires a complete bridge implementation, and it is not clear this would ever be utilized. Therefore, this interface specification requires support for WRITES between agents on PCI and the A.G.P.[2], but does not require support for READS between the A.G.P. and PCI, and READS should, in general, not be assumed to work.

Address re-mapping support for PCI compliant bus master accesses to memory: there could be situations where it would be convenient for memory accesses from a PCI bus compliant master to use the GART range address re-mapping services provided to the A.G.P. compliant graphics device. One example, would be a capture device writing to a tiled or non-linear graphics surface allocated in system memory[3]. While this works very naturally in the integrated (desktop) chipset solution, it may well require redundant mapping hardware in the MP solution. Therefore, this interface specification recommends, but does not require address re-mapping support for accesses from a PCI compliant bus master. In the case where re-mapping is not provided for PCI compliant masters, an address in the GART address range must be deemed an access error, and dealt with consistent with the error handling policies of that platform. In ***NO*** case may an address falling in the GART range be propagated through the core logic without re-mapping. Regardless of source, all GART addresses must be EITHER re-mapped, OR trapped and faulted. Note: while no tangible requirement for PCI bus re-mapping support exists today, it may be dangerous to fault physical addresses, and this solution, while possible, is NOT recommended.

Performance: as already discussed in section 2.3, a variety of performance parameters will likely have chipset and/or platform dependencies.

These are examples of platform dependencies that have been determined to fall outside the scope of this interface specification. In general, the scope of this interface specification is limited to the electrical and logical behavior of the actual A.G.P. interface signals, the mechanical definition of an A.G.P. compliant add-in board, the newly defined A.G.P. configuration registers, and software API which controls the graphics address re-mapping function.

---

[2] By way of example, this allows for a video stream generator (e.g., capture, decode, etc.) on PCI to WRITE to the graphics frame buffer on the A.G.P.

[3] Note that there is currently no example of this kind of I/O and buffer management. Today a capture device writes linearly to a linked list of discontiguous pages.

3. Signals and Protocol Specification

3.1 A.G.P. Operation Overview

Memory access pipelining is the major PCI protocol enhancement provided under the Accelerated Graphics Port (A.G.P. or A.G.P.) Interface Specification. A.G.P. pipelined bus transactions share most of the PCI signal set, and are actually interleaved with PCI transactions on the bus. Only memory read and write bus operations targeted at main memory can be pipelined; all other bus operations, including those targeted at device-local memories (e.g., frame buffers), are executed as PCI transactions, as defined in the PCI Rev. 2.1 Specification.

A.G.P. pipelined operation allows for a single ***A.G.P. compliant target***, which must always be the system memory controller, referred to in this document as ***core logic***. In addition to A.G.P. compliant target functions, the core logic must also implement a complete PCI sequencer[4], both master and target. For electrical signaling reasons the A.G.P. is defined as a point-to-point connection, therefore there is also a single ***A.G.P. compliant master***, which, in addition to implementing the A.G.P. compliant master functions, must also provide full PCI compliant target functionality[5] - PCI compliant master functionality is optional.

3.1.1 Pipeline Operation

The A.G.P. interface is comprised of a few newly defined “sideband” control signals which are used in conjunction with the PCI signal set. A.G.P.-defined protocols (e.g., pipelining) are overlaid on the PCI bus at a time and in a manner that a PCI bus agent (non-A.G.P.) would view the bus as idle. Both pipelined access requests (read or write) and resultant data transfers are handled in this manner. The A.G.P. interface uses both PCI bus transactions without change, as well as A.G.P. pipelined transactions as defined herein. Both of these classes of transactions are interleaved on the same physical connection. The access request portion of an A.G.P. transaction (bus command, address and length) is signaled differently than is a PCI address phase. The information is still transferred on the **AD** and **C/BE#** signals of the bus as is the case with PCI[6], but is identified or framed with a new control signal, **PIPE#**, in a similar way to which PCI address phases are identified with **FRAME#**.

The maximum depth of the A.G.P. pipeline is not architecturally constrained but is set to a maximum of 256 by this interface specification. However, the maximum A.G.P. pipeline depth may be reduced further by the capabilities of both master and target. The target provides an implementation dependent number of pipe slots, which is identified at configuration time and made known to the bus master (see section 6.3). The pipeline is then source throttled, since a master is never allowed to have more outstanding requests than the number of pipe slots it has been allocated.

---

[4] The A.G.P. Target must behave as a PCI 2.1 compliant Master and Target with one exception. The A.G.P. Target is not required to adhere to the target initial latency requirements as stated in the PCI 2.1 specification.

[5] The A.G.P. Master must function as a 2.1 compliant PCI target.

[6] Note that there are some optional mechanisms that allow address demultiplexing - using an alternate (non-**AD** bus) mechanism for transferring address - which is described in section 3.1.3.

The notion of intervening in a pipelined transfer enables the bus master to maintain the pipe depth by inserting new requests between data replies. This notion is similar to the software notion of a subroutine call; one context (data reply) running on the physical resource (wires in this case) is temporarily suspended while another context (request queuing) runs, after which the first context is restored and processing continues. This bus sequencing is illustrated in Figure 3-1 below.

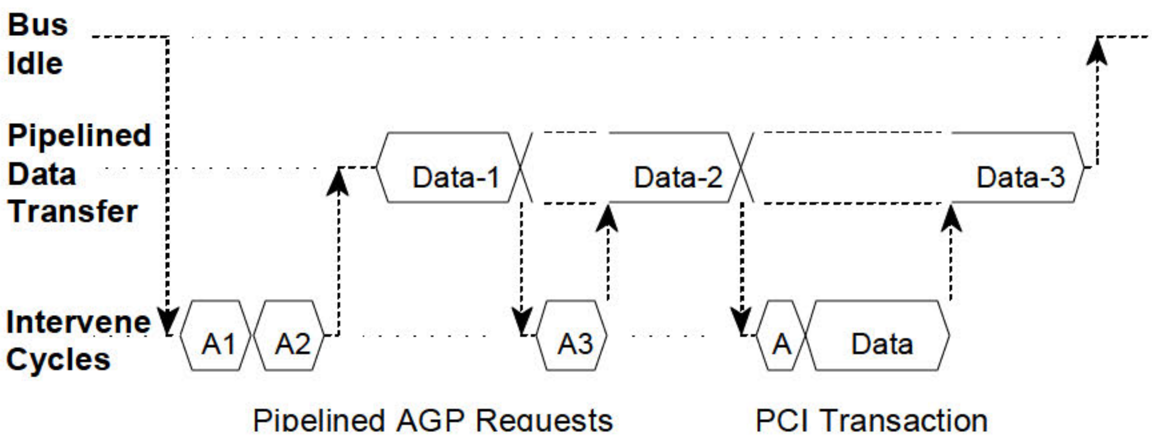


Figure 3-1 Basic A.G.P. Pipeline Concept.

When the bus is in an idle condition, the pipe can be started by inserting one or more A.G.P. access requests consecutively. Once the data reply to those accesses starts, that stream can be broken (or intervened) by the bus master (e.g., graphics controller) to:

- Insert one or more additional A.G.P. access requests.
- Insert a PCI transaction.

This intervene is accomplished with the bus ownership signals, **REQ#** and **GNT#**.

Operation of the bus can also be understood in terms of the 4 bus states shown in Figure 3-2. The operation of the PCI bus can be described by the two states *PCI* and *IDLE*, and the transition lines directly connecting them.

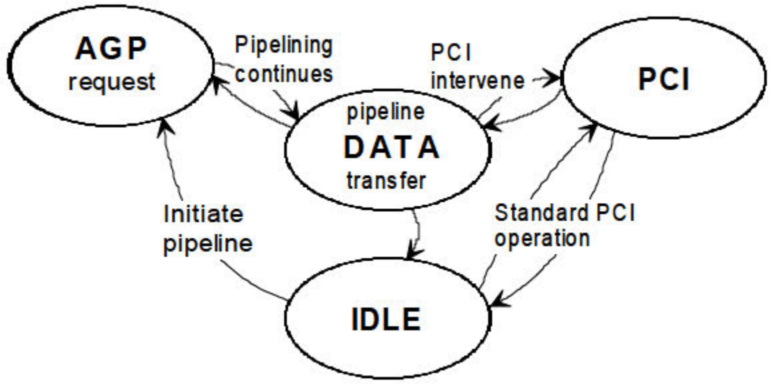


Figure 3-2 A.G.P./PCI Operational States

The A.G.P. pipeline is initiated from the idle state by arbitrating for the bus, and delivering one or more A.G.P. access requests (*A.G.P.* state). These requests are transmitted much like a PCI address phase except that they are timed with **PIPE#** rather than **FRAME#**. When one or more addresses has been transmitted, and **PIPE#** is deasserted, the bus enters the *DATA* state, in which the core logic (sole A.G.P. compliant target) controls the **AD** lines, and transfers data. If a bus master then requests the bus (**REQ#**) the A.G.P. compliant arbiter (always located in A.G.P. compliant target / core logic) suspends pipelined data transfer, and, using the **GNT#** signals, allows the bus master to initiate a bus transaction, driving the bus to either the *A.G.P.* or the *PCI* state depending on whether the master asserts **PIPE#** or **FRAME#**. After this transaction is complete, the bus returns to the *DATA* state and resumes the pipelined transfer. Pipelined data flow may be suspended ONLY at transaction boundaries, never in the middle of a single transaction. While the return of data is pending (a request for data has not been completed) the state machine remains in the DATA state. If new request need to be enqueued while data is pending, the machine transitions from DATA to A.G.P. or PCI depending on what type of request is initiated. Only when all data has been transferred that was previously request, does the machine return to the IDLE condition. For mobile designs, the clock is not allowed to be stopped or changed except when the bus has returned to the Idle state, which means that there are no outstanding requests pending. Flow control issues and protocol for pipelined A.G.P. requests and data transfer are discussed in section 3.2.5

3.1.2 Addressing Modes and Bus Operations

A.G.P. transactions differ from PCI transactions in several important ways.

1. The data transfer in A.G.P. transactions (both reads and writes) is “disconnected” from its associated access request. I.e., request and associated data may be separated by other A.G.P. operations, whereas a PCI data phase is connected to its associated address phase with no possibility of intervening operations. This separation not only allows the pipe depth to be maintained, but also allows the core logic to insure a sufficiently large buffer is available for receiving the write data before tying up the bus on a data transfer that otherwise could be blocked awaiting buffer space. Note that all of the access ordering rules on A.G.P. are based on the arrival order of the access requests, and not the order of actual data transfer.

2. A.G.P. transactions use a completely different set of bus commands (defined below) than do PCI transactions. A.G.P. bus commands provide for access ONLY to main system memory. (PCI bus commands provide for access to multiple address spaces: memory, I/O, configuration.) The address space used by A.G.P. commands is the same 32-bit, linear physical space also used by PCI memory space commands,[7] as well as on the processor bus.

3. Memory addresses used in A.G.P. transactions are always aligned on 8-byte boundaries; 8 bytes is the minimum access size, and all accesses are integer multiples of 8 bytes in length.[8] (Memory accesses for PCI transactions have 4-byte granularity, aligned on 4-byte boundaries.) Smaller or odd size reads must be accomplished with PCI read transaction. Smaller or odd size writes may be accomplished via the **C/BE#** signals, which enable the actual writing of individual bytes within an eight byte field.

[7] This physical memory space may contain a GART range, within which addresses are translated per the description in section 2.1.

[8] The motivation for increasing the addressing granularity from 4 bytes (PCI) to 8 bytes is tied to the typical memory organization used with 64-bit processors. Memories for these systems will generally be 64 bits wide. Therefore, smaller accesses will not provide any performance savings at the memory, and the motivation of A.G.P. is centered around maximizing memory performance for media controllers.

4. A.G.P. access requests have an explicitly defined access length or size. (PCI transfer lengths are defined by the duration of **FRAME#**.)

5. A.G.P. accesses do not guarantee memory coherency. I.e., A.G.P. accesses are not required to be snooped in the processor cache. PCI memory accesses always insure a coherent view of memory, and must be used on accesses where coherency is required.

The format of a complete A.G.P. bus request is shown in Figure 3-3.

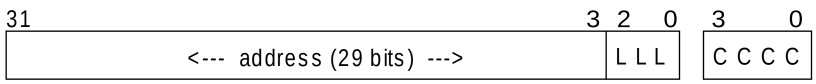


Figure 3-3 Layout of an A.G.P. Access Request

The '*LLL'* field contains the access length in units of Q-words (8 bytes), and displaces the low order 3 bits of address. A length field of "000" indicates that a single Q-word (8 bytes) of data is being requested, while "111" indicates 8 Q-words (64 bytes) are being requested. The 'CCCC' field contains the bus operation or command as itemized in Table 3-1.

CCCC	A.G.P. Operation
0000	Read
0001	Read (hi-priority)
0010	reserved
0011	reserved
0100	Write
0101	Write (hi-priority)
0110	reserved
0111	reserved
1000	Long Read
1001	Long Read (hi-priority)
1010	Flush
1011	reserved
1100	Fence
1101	reserved
1110	reserved
1111	reserved

Table 3-1 A.G.P. Bus Commands

Command definitions are as follows.

Read: starting at the specified address, read *n* sequential Q-words, where *n* = (length_field + 1).

Read (hi-priority): same as *Read*, but the request is queued in the high priority queue. The reply data is returned out of order and within the maximum latency window established for high priority accesses (seesection 2.3). High priority accesses only follow A.G.P. ordering rules with respect to other high priority read accesses.

Write: starting at the specified address, write *n* sequential Q-words, as enabled by the **C/BE#** bits, where *n* = (length_field + 1). Writes obey the bus ordering rules (they may be retired ahead of previously issued reads).

Write (hi-priority): same as *Write*, but indicates that the write data must be transferred from the master within the maximum latency window established for high priority accesses[9] (see section 2.3). High priority write accesses only follow A.G.P. ordering rules with respect to other high write priority accesses.

Long Read: same as *Read* except for access size; in this case *n* = 4 * (length_field + 1), allowing for up to 256 byte transfers.

Long Read (hi-priority): same as *Read (hi-priority)* except for access size which is the same as for *Long Read*.

Flush: similar to read. This command drives all low priority write accesses ahead of it to the point that all the results are fully visible to all other system agents, and then returns a single Q-Word of random data as an indication of its completion (see section 3.2.3). The address and length fields are meaningless for this command.

Fence: creates a boundary in a single master's access stream, around which writes may not pass reads (see section 3.2.3). This command is the only one which does NOT occupy a slot in the A.G.P. pipeline.

Reserved: Must not be issued by a master and may be defined in the future by Intel.

3.1.3 Address Demultiplexing Option

Ideally, to maximize efficiency and throughput on a random access memory bus such as A.G.P., the address is demultiplexed (separate address pins) from the data pins. The A.G.P. interface provides an optional sideband signal set to do this (**SBA[7::0]**), referred to as the ***sideband address port***. However, in order to keep pin cost down it is only an 8-bit wide interface. Implementation of the sideband address port is optional for an A.G.P. compliant master. However, an A.G.P. compliant target (all core logic products) is required to fully support sideband address port operation. This means that a master wishing to use this A.G.P. feature may always depend on target support. Software can query the master and determine if the SBA port will be used by the master or not. When used by the master, software will enable the target to accept requests using the SBA port. See section 6.1 for a description of the SBA status and enable bits.

The sideband address port is used exclusively to transmit A.G.P. access requests (all PCI transactions use the **AD** pins for both data and address), and therefore it is always driven in one direction; from master to target. The semantics of an A.G.P. request transmitted via **AD** pins or **SBA** pins are identical; only the actual syntax of transmission varies. The **SBA** and **AD** pins are never used in any combination to transmit requests; in any given configuration, all A.G.P. requests are transmitted either on **AD** pins or **SBA** pins. A master which uses the sideband address port, has no need of the **PIPE#** signal, which is used only to frame requests on the **AD** pins.

In order to transmit the complete A.G.P. access request across the 8-wire **SBA** port, the request is broken into 3 parts: low order address bits and length, mid-order address bits and command, and high order address bits. These three parts are referred as Type 1, Type 2 and Type 3 respectively. The registers for the last two parts (mid-order address bits (Type 2) and high order address bits (Type 3)) are 'sticky'. Where *sticky*

---

[9] This implies that if the target write queue is full, some access priority must be raised in order to accommodate this access within the latency requirement.

refers to the attribute where they retain what was last loaded into them, so these two parts of the request need only be transmitted if they have changed since the previous request. This exploits the potential locality in the access request stream to minimize the address traffic over the 8 **SBA** signals.

The transmission of each of these 3 Types is accomplished by a separate **SBA** operation. Each operation on the sideband address port delivers a total of 16 logical bits, in 2 phases or transfer ticks of 8 bits each. In 1x transfer mode, each **SBA** operation requires two A.G.P. clocks; while in the 2x transfer mode (source clocked option is active), the entire transfer completes in one A.G.P. clock (See 3.5.1.2.) The **SBA** pins always operate at the same transfer rate as the **AD** pins; either 1x or 2x transfer mode as initialized in the A.G.P. command register (see section 0.) This relationship keeps the 8-bit sideband address port well matched in speed with data transfer on the **AD** pins, since the minimum data transfer size is 8 bytes (2 **AD** ticks), and most A.G.P. access requests will only involve the low order address bits and length, requiring a single **SBA** operation (2 **SBA** ticks).

Table 3-2 below shows the definition and encoding of each of the sideband address port operations. Bold underlines in the encoding column indicate op codes. Each opcode requires two data transfers to move the entire 16 bits to the A.G.P. compliant target. Note that the first piece of data transferred includes the op code. For example for the Length and Lower Address Bits encoding which has an opcode is 0. In this encoding, the first data transferred is the op code (0) and address bits 14 - 08. The second piece of data transferred is address bits 7-3 and the 3 length encoded bits.

Table 3-2 Sideband Address Port Encoding

Encoding	Description
S_7.. ...S_0	Shows alignment of messages on physical sideband wires.
1111 1111 **[1111 1111]**	***Bus Idle:*** used to indicate the bus is idle, also referred to as a NOP. When running at 1x transfer mode, this command is limited to a single clock tick of 8 bits (all ones) while 2x transfer mode requires the full 16 bits as shown here.
0AAA AAAA 14 08 AAAA ALLL 07 03	***Length & Lower Address Bits:*** the A.G.P. access length field (**LLL**), and lower 12 address bits (**A[14::03]**) are transferred across the sideband address port, and a memory access is initiated. The encoding is also referred to as a Type 1 sideband command. The remainder of the A.G.P. access request (**A[31::15]** and bus command) is defined by what was last transmitted using the other two sideband address port commands (Type 2 and Type 3. Note that **AD[2::0]** are assumed to be zero when using this encoding and these bits are not transferred.
10CC CC-A 15 AAAA AAAA 23 16	***Command & Mid Address Bits:*** the A.G.P. bus command (**CCCC**) and mid-order 9 address bits (**A[23::15]**) are transferred across the sideband address port; no memory access is initiated. This encoding is also referred to as a Type 2 sideband command. This command, when followed by the previous command (Type 1) provides for memory access anywhere within a naturally aligned 16 MB 'page'.
110- AAAA 35 32 AAAA AAAA 31 24	***Upper Address Bits:*** the upper 12 address bits (**A[35::24]**) are transferred across the sideband address port; no memory access is initiated. This encoding is also referred to as a Type 3 sideband command. This command, when followed by the two previous commands (Type 2 and Type 1) provides for memory access anywhere within a 32-bit physical address space. The extra four bits (**A[35::32]**) are place holders to avoid aliasing problems in the face of possible address expansion.
1110 **** **** ****	***reserved:*** must not be issued by an A.G.P. compliant master and maybe defined by Intel in the future.

Note that only one sideband address port command (Type 1) actually initiates a memory cycle; the other two (Type 2 and Type 3) simply update respective 'sticky' bits. There is no restriction on the relative ordering in which Type 1, 2 or 3 commands can be issued by the master. If a memory cycle is initiated prior to the initial setting of all access request 'sticky' bits, those bits will be treated as indeterminate. For example, if the first command issued after the port is enabled is a Type 1 command (Type 2 or Type 3 have not occurred yet), the A.G.P. compliant target may use an indeterminate address bits (A15- A31) and command (C3-C0) to access memory. The master is allowed to issue Type 1, 2 or 3 commands in any order and memory accesses are queued anytime a Type 1 is issued. The A.G.P. compliant target receives a Type 1 command it takes the Type 1 information and combines it with previously stored Type 2 and Type 3 information to reconstruct a full address, command and length information to initiate a memory access.

The sideband address port has no associated control or framing signals; command framing is content sensitive (similar to serial interconnects). That is, the port encoding signifies whether there is valid information on the port. A NOP encoding (all 1's) indicates the port is idle and no action is initiated by the master. NOP must be continually transmitted when the port is not in use. The Type 2 and 3 target registers are not affected while NOPs appear on the SBA interface. Since all **SBA** operations must start with the rising edge of the A.G.P. clock, the port idle encoding is 8-bits long in 1x transfer mode, and 16-bits long in 2x transfer mode.

3.2 Access Ordering Rules & Flow Control

3.2.1 Ordering Rules and Implications

This section discusses the ordering relationships between A.G.P. and non-A.G.P. transactions initiated on the A.G.P. interface, between different streams of A.G.P. transactions, between A.G.P. transactions and other system operations (CPU and PCI). These rules apply to operations generated by an A.G.P. compliant Master (and completed by a A.G.P. compliant target). Note that the following rules do not apply to High Priority operations, which will be discussed in section 3.2.4..

A.G.P. Compliant System Ordering Rule:

> There is no ordering relationship between an A.G.P. Compliant Master's operation and any other system operation, including operations generated by host CPU(s), PCI agents, or expansion bus agents.

This means that A.G.P. transactions are only required to follow A.G.P. ordering rules, even when A.G.P. transactions cross into other domains. For example, an A.G.P. compliant master read to a location in memory that is currently locked by the processor is not required to adhere to the processors lock. It is allowed to complete in the face of the lock and a programming error has occurred if this causes an incorrect operation. A.G.P. compliant hardware is not required to ensure consistent data when A.G.P. transactions interact with the rest of the system. For a read this means that the A.G.P. compliant hardware is allowed to get a copy from main memory and is not required to obtain a more recent copy (if available) from the CPU cache. For a write, the A.G.P. compliant hardware can simply write the data to main memory without snooping the CPU cache. If the cache has a modified line, it will over write the A.G.P. write at some point. The A.G.P. compliant hardware is not required to *force* A.G.P. data out of the A.G.P. domain to memory. When an A.G.P. compliant master needs to cause a synchronization event (request an interrupt or set a flag) to occur it must use the FLUSH command to guarantee that previous A.G.P. write operations become visible to the rest of the system. See section 3.2.3 for details.

A.G.P. Compliant Device Ordering Rules

1) The is no ordering relationship between an A.G.P. operation and a PCI transaction.

2) The A.G.P. compliant Target will return a stream of A.G.P. Read data in the same order as requested.

   Example: Reads requested in the order A, B, C, D will return data in the same order as requested A, B, C, D.

3) A.G.P. Write operations are processed by the A.G.P. compliant Target in the order they are requested.

   Example: Writes requested in the order A, B where A and B overlap will cause B to overwrite part of A.

4) Read data returned will be coherent with previously issued A.G.P. Write requests.

   Example: Requests in the order 'Wa, Wb, Rc, Wd, Re' to the same address. Read data returned for Re will be what was written by Wd. (Reads push writes.)

5) An A.G.P. Write operation may bypass previously issued A.G.P. Read operations. Read data returned may reflect data associated with a subsequently issued Write request. (Writes are allowed to pass reads)

Example: Requests in the order 'Wa, Wb, Rc, Wd, Re' to the same address. Read data returned for Rc may be either what was written by Wb or Wd. Wd is returned when Wd passes Rc.

6) PCI transactions initiated by an A.G.P. compliant Master or Target must follow the ordering rules specified in the PCI specification.

Explanation of A.G.P. interface Ordering Rules (explanations correspond to rules with the same number):

1) When an A.G.P. compliant agent is capable of generating both PCI and A.G.P. transactions, the A.G.P. compliant target is not required to maintain any ordering between these two streams. However, the A.G.P. compliant target is required to maintain order within a given stream based on the ordering rules for that stream. For example, a master issues a PCI and an A.G.P. transaction. The order in which the A.G.P. and PCI transactions complete does not matter. These are two different streams of requests and different streams have not ordering relationships.

2) Even though the A.G.P. compliant Target will return a stream of A.G.P. Read data in the same order as requested, this does not mean that the read transactions actually occur at the destination in the same order as requested. For example, a master enqueues Read x and then Read y. Before the read data is returned, a Write to location x and then a Write to location y occurs. Because of the ordering rules defined above, it is possible for the Read to location x to return old or new data and the Read to location y to return old or new data. Note that if the Read to location x returns new data it does not imply that the Read to location y will also return new data. If the Read to location x returned old data, it is possible for the Read to location y to return new data. The value that is returned is determined by the A.G.P. compliant target after the requests have been enqueued and before data is returned to the master. The ordering rules as described above only require that the data being returned to the master be delivered in the same order as requested. The A.G.P. compliant target is allowed to re-arrange Read requests to improve performance, but is never allowed to return data in a different order than requested. Another example is where the master requested Read A, B, C and then D. However, the memory controller is allowed to obtain the data C, B D and then A, but is required to return the data in A, B C and then D.

3) This rule means that A.G.P. write data cannot pass previously written A.G.P. data.

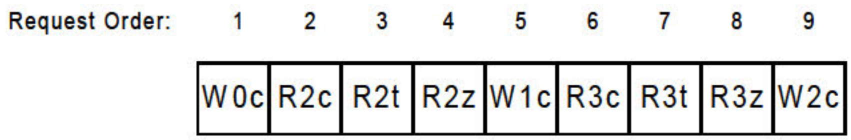
4) Read requests will push Write data (within the same stream) from an A.G.P. compliant master. An A.G.P. read to a location previously written by an A.G.P. write operation will return the latest copy of the data seen by the A.G.P. interface.

5) This rule means that an A.G.P. write issued after an A.G.P. read is allowed to pass the read request and may cause the read to return the new value of the data even though the write request and data transfer occurred on the bus after the read request occurred. To ensure that the old value is returned, the A.G.P. compliant master must not issue the write transaction until after the read data has returned from the read request or issue a FENCE command (which is discussed in section 3.2.3) between the read and the write.

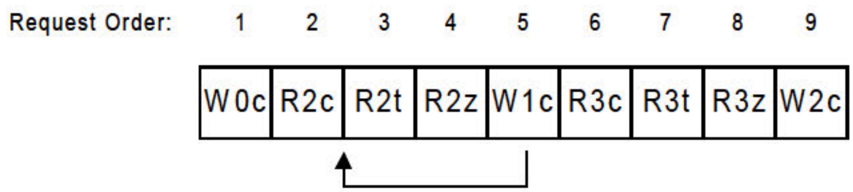
Implications of Allowing A.G.P. Writes to Pass A.G.P. Reads

A potential problem created by allowing A.G.P. Writes to pass A.G.P. Reads is that an A.G.P. Read may return "old" data from a previous A.G.P. Write or "new" data from a following A.G.P. write. An A.G.P. Read sandwiched by A.G.P. Writes may return data for either write — it is indeterminate. This is shown in the following example:

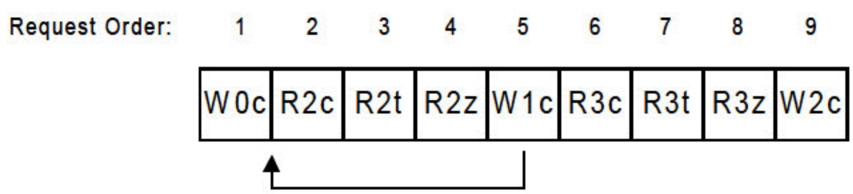
A 3D Graphics controller Master generates the following sequence of pipelined A.G.P. requests. In this example the reads are from the frame buffer, texture buffer, and depth buffer respectively, while the writes are to the frame buffer. This example assumes that all frame buffer accesses are to the same address.



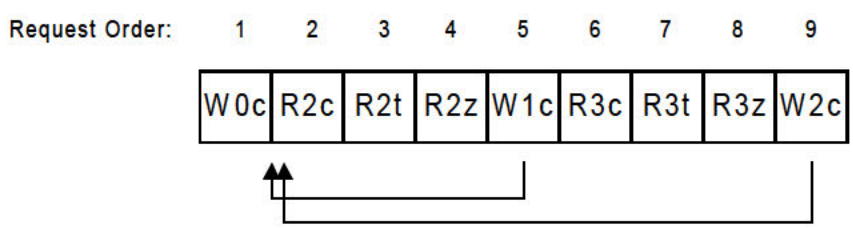
The following diagram shows W1c passing R2t. In this case R2c will return "old" data from the W0c write.



The following diagram shows W1c passing R2c. In this case R2c will return "new" data from the W1c write.



The following diagram shows both W1c and W2c passing R2c. In this case R2c will return "new" data from the W2c write. (In this graphics controller example write W2c is dependent on R2c data returning. So in reality the write request W2c will not be generated before the read data for R2c is returned. However if the requests were pipelined deeper it would be possible for several writes to pass a particular read.)



7) PCI transactions generated by a device on the A.G.P. interface follow the same rules as a device that resides on a PCI bus segment. The PCI agent transactions will follow the same rules as described in the PCI bus specification even though it was initiated on the A.G.P. interface. This agent's transactions have no ordering with respect to any A.G.P. transactions that occur.

A.G.P. Compliant Master Implications of Allowing Writes to Pass Reads

If an A.G.P. compliant Master doesn't care if it gets "old" or "new" data for a given read operation no special action needs to be taken. If an A.G.P. compliant master is particular about getting "new" or "old" data it is the A.G.P. Compliant Master's responsibility to ensure that it gets the correct data. There are various methods to ensure this. Some of these methods are discussed below.

If an A.G.P. Compliant Master must get "new" data it may:

Detect that a conflict exists between a Read request that has already been generated and an internally pending Write request, merge (or substitute) the "new" Write data with the "old" Read data when it is returned; or

Delay the Read request behind the Write request. Since Reads "push" Writes per the ordering rules, the Read will return the "new" data. Since it is desirable to deeply pipeline the A.G.P. requests, actually determining that a conflict exists between a Read and a subsequent Write may be difficult (or impossible) to detect. Once a conflict is detected delaying the Read may stall the pipeline and impact performance.

If an A.G.P. Compliant Master must get "old" data it may:

Issue a FENCE command between the Read and the following Write or

Delay the "new" data Write until the "old" Read data has been returned. This method has the potential for deadlock. A deadlock occurs when delaying a Write causes an A.G.P. Compliant Master's data engine to back-up. If the A.G.P. Compliant Master's Read data return buffers are full the stalled data engine can't remove Read data from the buffers. The "old" Read data can't be accepted and the Write will continue to be delayed, creating the deadlock.

3.2.2 Deadlock Avoidance

An A.G.P. Compliant Master can not make the data transfer phase of a previously issued request dependent on the completion of *any* other A.G.P. or PCI transaction to the device as either a Master or Target.

3.2.3 Flush and Fence Commands

Because of the ordering rules of the A.G.P. interface, the master can not guarantee when write data has reached its final destination. From the A.G.P. compliant master's standpoint a write transaction appears to have completed but the write data may still be pending in the A.G.P. interface. The master also needs the ability to ensure that certain transactions complete on the A.G.P. interface before other transaction are issued. This can be accomplished by delaying the issuing of subsequent requests until previous requests complete, but this defeats the use of pipelining to improve system performance. The FLUSH command causes A.G.P. transactions to become visible to the rest of the system so synchronization events may occur. The FENCE command guarantees what order accesses will complete in, without delaying the issuing of subsequent commands. Each will be discussed in more detail in the following paragraphs. The FENCE and FLUSH commands are low priority commands and have no affect on high priority requests.

Flush:

Under most conditions the master does not care if its transactions are visible to the system or not. But in those cases when it does matter, the FLUSH command is used by an A.G.P. compliant master. The FLUSH command ensures that its low and high priority write transactions have become visible to the rest of the system. Because of the A.G.P. ordering rules, the master can not cause accesses to become visible to the system by using the memory commands like is possible when using PCI commands. Memory commands can only cause data to be returned in a specific order; they place no requirement on the core logic to make accesses visible to the system. However, the core logic must cause A.G.P. accesses to become visible to the rest of the system when the FLUSH command is issued. The FLUSH command behaves similar to a low priority read command except that a single Qword of random data is returned. The return of the random data is the acknowledgment to the master that all previous low and high priority write transactions have become visible to the system. When the FLUSH command completes, the master may safely cause a synchronization event to occur.

Take the case when the A.G.P. compliant master writes data to memory, but does not use the FLUSH command before generating an interrupt. The driver reads its device and determines that data is valid in memory. When it accesses main memory (from the CPU) it may access stale data because the data is still in the A.G.P. domain. In the PCI domain, this sequence was all that was required to guarantee that the correct data would be accessed. However, for A.G.P. this is not sufficient. Since A.G.P. accesses have no ordering with respect to any other accesses in the system (in this example from the CPU), the A.G.P. interface is not required to flush posted write data before completing a read to the A.G.P. interface. Therefore, the posted write data may still reside in the A.G.P. interface and the driver may access stale data. For PCI transactions, the flush of posted data on any read causes loss of performance in the system and generally is only required in certain cases. The FLUSH command provides a mechanism for the master to ensure that the correct data will be accessed when a synchronization event occurs, but does not force the system to flush buffers when not required.

The FLUSH command occupies a slot in the Transaction Request Queue when issued and is retired from the queue when the associated single Qword of data is returned. The only limit to the number of outstanding FLUSH requests are the limits of the transaction request queue itself. It is possible to have the Transaction Request Queue full of FLUSH commands.

Fence:

Because of the A.G.P. ordering rules, the master needs a mechanism that forces writes not to pass previously enqueued read commands. An A.G.P. compliant master uses the FENCE command to demarcate one set of A.G.P. requests from another. The FENCE command affects the order in which they are completed in memory and may not necessarily determine the order in which they complete on the bus. On either side of the demarcation, A.G.P. requests are processed based on the A.G.P. ordering rules. However, ALL requests generated prior to the FENCE command are processed prior to ANY request following the FENCE command. A.G.P. Write requests generated after a FENCE command may not pass any Read requests generated prior to the FENCE command. Read requests issued after the FENCE command may not be combined with or pass any Read request issued prior to the FENCE command.

High Priority requests are exceptions and are allowed to pass the demarcation established by the FENCE command. The FENCE command does not occupy a slot in the Request queue of the A.G.P. compliant master or target. An A.G.P. compliant master may generate an unlimited number of FENCE commands.

3.2.4 Access Request Priority

The A.G.P. bus command set supports two levels of access priority. In general, the high priority queue has the highest priority for memory service, and the low priority queue has lower priority than the processor, but generally higher than any other subsystem for memory service. The high priority queue should be used with caution since it causes additional latency to other requests. For example, the high priority queue may be useful for a graphics controller reading display memory or to avoid overflow/underflow in a data stream having real-time deadlines. The high priority queue is intended for very selective use when an A.G.P. request needs immediate processing.

Requests in the high priority queue may bypass all other (low priority or PCI) requests and may be returned out of order with respect to other streams. Only requests that can tolerate re-ordering (with respect to all accesses other than themselves) should be completed using a high priority command. High priority accesses only have order with respect to the same type of request. For example, high priority read requests only have ordering with respect to other high priority read requests. High priority write accesses only have ordering with respect to other high priority write accesses. Unlike low priority operations, there are no ordering requirements between high priority read and high priority write accesses. The sequence, HPR-A, HPW-B, HPR-C and HPW-D will be used in the following discuss. Read data will be returned in the order in which read accesses were requested. In this example, A will always complete before by C. Write data will always complete in the order requested, in this example write B will always complete before write D. There is no order between read and write high priority operations. In this example, the accesses may complete A, C, B and D; A, B, C and D; B, A, D and C; or B, D, A, and C. However, the order can NEVER be C completes before A or D completes before B.

Both Read and Write requests may be issued as high priority accesses.. The A.G.P. protocol designates read replies as part of either the high or low priority stream, enabling the bus master which originated the access to associate the reply with the correct outstanding request. Writes issued as high priority accesses will have transferred the data across the interface within the maximum latency window established for high priority accesses. This does not imply that the data will have been retired to main memory within this latency window. .

3.2.5 Flow Control

3.2.5.1 Introduction

Flow control on A.G.P. is different than that of PCI. On PCI, the master and target may delay the transfer of data on any data phase. Before each data phase can complete both the master and target must agree that data can be transferred by asserting their respective **xRDY#** signal. When either is not prepared to transfer data, the current data phase is held in wait states. PCI also allows the target to indicate to the master that it is not capable of completing the request at this time (Retry or Disconnect). Only when both agents agree to transfer data does data actually transfer.

On A.G.P., flow control is over blocks of data and not individual data phases. Flow control will be discussed with respect to Initial Blocks and Subsequent Blocks. Some transactions only have initial blocks, this occurs when the entire transaction can be completed within 4 clocks. For transactions that require more than 4 clocks to complete, they are comprised of both an Initial Block and one or more Subsequent Blocks. A block is defined as four A.G.P. clocks and is 8 byte aligned, but is not required to be cacheline aligned. Depending on the transfer mode, the amount of data that is actually transferred may change. But in all cases, the number of clocks between throttle points is always four. Flow control on A.G.P. refers to the initial or subsequent data block. Table 3-3 lists the control signals and which agent drives them and which agent receives them. The table following the diagram lists the flow control of initial and subsequent blocks based on transaction type and which agent is allowed to flow control the data movement.

Flow Control Point > Transaction Type	Target - Initial Data Block	Target - Subsequent Data Block	Master - Initial Data Block	Master - Subsequent Data Block
Low Priority Read Data	**TRDY#**	**TRDY#**	**RBF#**	**IRDY#**
High Priority Read Data	**TRDY#**	**TRDY#**	None	**IRDY#**
Write Data	**GNT#/ST[2::0]**	**TRDY#**	**IRDY#**	None

Table 3-3 A.G.P. Flow Control Points

There are basically three times in which the transfer of data can be delayed:

1. After the data has been requested but before initial data block is returned;
2. During the initial data block of the transaction; and
3. for Subsequent data block(s).

The first case occurs when the master is allowed to delay the return of read data for a Low Priority (LP) transaction by the use of **RBF#** (Read Buffer Full). This signal implies that the Read Buffer is currently full and that the arbiter is not allowed to attempt to return LP read data.

The second case occurs for both the master and target. The target is allowed to delay the completion of the first data phase of the initial block of read data by not asserting **TRDY#**. Since the control and data are moving in the same direction, the master latches the read data and qualifies it with **TRDY#**. When **TRDY#** is asserted, the data is valid on each clock until the block completes. The master is not allowed to flow control the initial block of a High Priority (HP) Read transaction. However, like LP reads, the master is allowed to flow control HP reads like LP reads on Subsequent blocks of data. The master is allowed to delay the first data phase of a write transaction by not asserting **IRDY#**. The agent receiving data is not allowed to insert wait states on the initial block of data for either read or write transactions. The agent sending data is allowed to delay the initial block of data of either a read or write up to a maximum of one clock from when the transaction was allowed to start.

The third or last case, is where both the master and target are allowed to insert waitstates on subsequent blocks of read data. The master is not allowed to insert waitstates on subsequent write transactions, but is allowed to on read transactions (both High and Low Priority).

For the throttle point (TP), there is no specified limit to how long **IRDY#** or **TRDY#** may be deasserted. However, the master must realize that inserting even one wait state at any TP of a read transaction may invalidate the latency guarantee of **all** outstanding high priority requests. If the master inserts a wait state at a TP it can not make any assumptions about what impact the wait state will have on the latency guarantee. For instance, inserting 5 wait states at a TP of Read A (high or low priority) does not mean that outstanding high priority read request B will complete in x + 5 clocks (where x is the latency guarantee provided by the core logic). The target must include any potential **TRDY#** throttle point wait states in its latency guarantee. The specific latency behavior of a target when a master inserts a waitstate is implementation specific. Refer to the data sheet of the specific device to understand this affect.

3.2.5.2 Read Flow Control

Initial Master Flow Control (Low Priority Reads)

RBF# (Read Buffer Full) is an output of the Master and indicates whether it can accept low priority read data or not. What affect the assertion of **RBF#** has on the data transfers depends on the length of the next transaction and the rate at which data is being transferred. If the master has **RBF#** deasserted it must be able to accept the following transactions assuming that the master asserts **RBF#** on the clock in which the grant is received:

For transactions that can be completed in 4 clocks or less, the master is required to accept the entire transaction without wait states regardless of the data transfer mode. When the transaction requires more than 4 clocks to complete, the master is allowed to insert waitstates after each 4 clocks in which data is transferred.

For 1x data transfers, the master must accept the entire transaction without wait states when the length is less than or equal to 16 bytes. When the transfer length is greater than 16 bytes, the master is allowed to flow control after each 16 byte transfer. When the length is 8 bytes or larger, the master has sufficient time to assert **RBF#** to prevent the arbiter from initiating the return of more LP read data.

For 2x data transfers, if a low priority read transaction's length is ***greater*** than 8 bytes the master must accept only the one low priority read transaction, because the master has sufficient time to assert **RBF#** to prevent the arbiter from initiating the return of more read data. When the transfer size is greater than 32 bytes, the master is allowed to flow control after the transfer of each 32 byte block.

For 2x data transfers, if the first low priority read transaction's length is ***equal*** to 8 bytes the master must accept two low priority read transactions. The first transaction must be accepted without flow control. The master must also accept the entire second transaction without flow control when its length is less than or equal to 32 bytes. When the second transaction's length is greater than 32 bytes, the master must accept the initial 32 bytes of the transaction, but is then allowed to flow control the subsequent 32 byte block(s).

Note: The arbiter must delay the assertion of **GNT#** for a subsequent read data transfer so that it is sampled asserted on the same clock edge as the last data phase for the previous read transaction when it is greater than 8 bytes. In order to allow full performance of 8 byte read transfers, the arbiter must pipeline the assertion of **GNT#** in a back-to-back fashion, otherwise dead clocks will appear on the **AD** bus. If the arbiter did not delay the subsequent **GNT#** in this manner the master would need a minimum of 64 bytes of buffering instead of 40 bytes for the 2x transfer mode, for example.

Table 3-4 shows the minimum amount of available data buffering required in the master when **RBF#** is deasserted. The table only applies to 2x data transfer mode[10].

1st Read Transaction	2nd Read Transaction	Buffer space needed to de-assert **RBF#**
8 bytes	$8 \le n \le 32$ bytes	8 + n bytes
8 bytes	n > 32 bytes	40 bytes
16 bytes	don't care	16 bytes
24 bytes	don't care	24 bytes
32 bytes	don't care	32 bytes
> 32 bytes	don't care	32 bytes

Table 3-4 Data Buffering for 2x Transfers

If the master can't accept the above transaction(s) it asserts **RBF#**. The A.G.P. compliant arbiter will not assert subsequent grants for low priority read data while **RBF#** is sampled asserted. In the event that **GNT#** and **RBF#** are asserted on the same clock, the master must be able to accept at least 4 clocks worth of data, the amount of data is dependent on the transfer mode. For the 2x mode, at least 32 bytes of data for the next low priority read transaction must be accepted.

Note: The A.G.P. compliant master has many implementation alternatives that can be predicated by buffer budget and complexity. For example the A.G.P. compliant master could restrict itself to generating only 16 byte low priority read transactions. In this case only 16 bytes of buffering need to be available in order to deassert **RBF#**. If an A.G.P. compliant master restricts itself to 8 and 16 byte low priority read

[10] For 1x data transfer mode, the amount of data buffering required is simply enough to accept the next data transfer or up to 16 bytes, whichever is greater.

transactions, **RBF#** can be deasserted whenever 24 bytes of buffering are available when in 2x transfer mode. An A.G.P. compliant master that does not restrict the size of its low priority read requests needs a minimum of 40 bytes of buffering for 2x transfer mode. Optionally this master could dynamically alter the **RBF#** threshold point based on the size of the next two accesses. It is highly recommended that the master use **RBF#** only in unusual circumstances in which the target is able to provide data quicker than the master is able to consume it. In normal operations, the master should be able to consume that requested data faster than the target is able to provide it. The assertion of **RBF#** to stop the return of data should not be part of the "normal" behavior of the master. Figure 3-4 illustrates the enqueuing of two grants before the arbiter detects that **RBF#** is asserted.

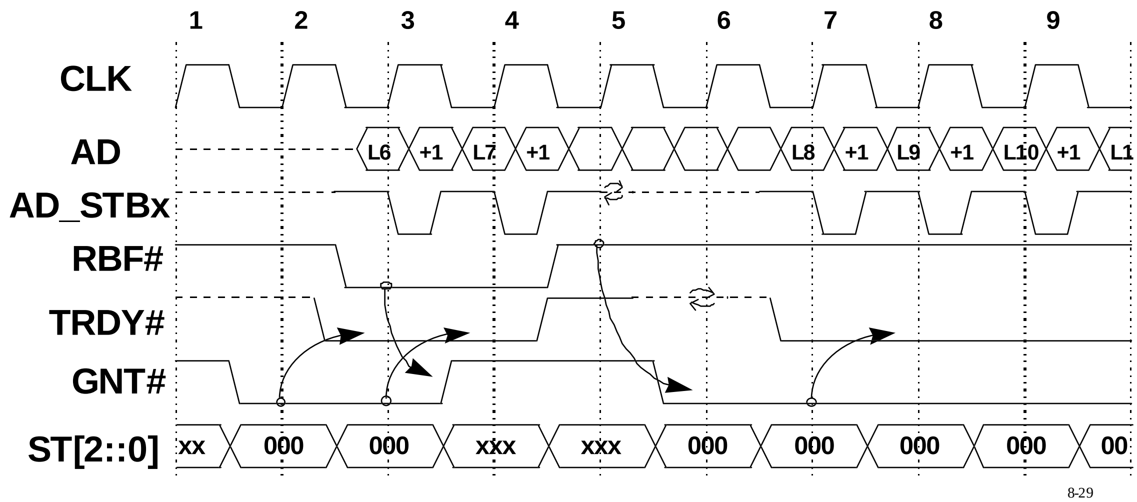


Figure 3-4 Maximum number of GNT#s Queued before the assertion of RBF#

Since **RBF#** is deasserted on clock 1, the A.G.P. compliant arbiter asserts **GNT#**/**ST[2::0]** to indicate the return of a low priority read data (D0) for clock 2. Since the first read data being returned is 8 bytes, the A.G.P. compliant arbiter continues asserting **GNT#**/**ST[2::0]** for clock 3 to allow the second transfer to occur without dead time on the **AD** bus between data transfers of D0 and D1. At this point the arbiter has indicated to the master that two transactions worth of data will be returned. Because **RBF#** is asserted on clock 3, the arbiter is not allowed to initiate the return of any more data after this point until **RBF#** is sampled deasserted again. The arbiter asserts **GNT#** for clock 6, since the master deasserted **RBF#** on clock 5 and the arbiter is ready to return more low priority read data to the master.

The master decodes the initial request (clock 2), determines that sufficient buffer space is not available for a subsequent transaction and asserts **RBF#**. Since **GNT#** and **RBF#** are both asserted on clock 3, the master must accept the second transaction. While the master keeps **RBF#** asserted, the arbiter is not allowed to initiate the return of any new low priority read data. However, the arbiter is allowed to return high priority read data, request (high or low priority) write data from the master, or grant the master permission to initiate requests. (See Figure 3-20.)

Since **GNT#** is asserted on clock 2 (**ST[2::0]** indicates the return of low priority read data), the master starts accepting data and qualifies it with **TRDY#** to determine when it is valid. Note: **TRDY#** is only asserted on the initial data transfer of this transaction since it will complete within four clocks. Once the initial data transfer completes, the master begins accepting data for the second transaction and qualifies that data with **TRDY#**. Note that **TRDY#** must be asserted on the initial data phase of each transaction.

Initial Master Flow Control (High Priority Reads)

The master must always be able to accept read data for all high priority queued transactions that can complete within 4 clocks. When a high priority read request requires more than 4 clocks (multiple blocks) to complete, the master can throttle the transaction (and effectively stall subsequent high priority read data) with **IRDY#** after each data block transfers(this is discussed in the next section). **RBF#** does not apply to high priority read data and **IRDY#** can not be used to initially stall the return of high priority read data.

Throttling

Throttling applies uniformly to both low and high priority read data. Both the target and the master have the ability to throttle read data by adding wait states after each block of data transfers. If either the target or the master wants to throttle the transfer of a subsequent block of data, the target must have **TRDY#** or the master must have **IRDY#** deasserted two 1x clocks prior to when the subsequent block would begin to transfer. This is referred to as the throttle point (TP). Data transfer will resume two 1x clocks after both **IRDY#** and **TRDY#** are sampled asserted. If throttling is not required by either the master or the target, then both **IRDY#** and **TRDY#** will be asserted at the throttle point. A throttle point occurs every 4 clocks. **IRDY#** and **TRDY#** have no meaning between throttle points and may be deasserted. **IRDY#** and **TRDY#** also have no meaning on the last throttle point of a transaction that is equal to or less than a block. Note: that **IRDY#** for the master and **TRDY#** for the target must be actively[11] driven from the first TP until the completion of the last TP. Following the last TP, **xRDY#** must be deasserted and tri-stated. During a TP for a read data transfer, once **xRDY#** is asserted it must remain asserted until the current TP completes, which occurs when both **IRDY#** and **TRDY#** are asserted.

3.2.5.3 Write Data Flow Control

Initial Target Flow Control

The A.G.P. compliant arbiter will only assert **GNT#**/**ST[2::0]** for write data when the target can accept the entire transaction or the initial block. The initial flow control is the same for both high and low priority data write requests.

Initial Master Flow Control

The master samples **GNT#**/**ST[2::0]** asserted for write data and asserts **IRDY#** to begin the write data transfer. The master can delay the beginning of the write data transfer one clock by delaying the assertion of **IRDY#**. Figure 3-5 illustrates the maximum delay in which a master can take when providing write data. **IRDY#** must either be asserted on clock 3 (earliest data can be provided) or clock 4 (the latest in which data can be provided). Once the master asserts **IRDY#** it must transfer all write data associated with the transaction without wait states. Since the master is not allowed to insert waitstates on subsequent TP, **IRDY#** must be deasserted and tri-stated after it is asserted to start the data transfer. On read transactions, **IRDY#** is meaningless except during TPs and must be actively driven until the last TP completes, at which time **IRDY#** must be deasserted and tri-stated.

---

[11] The **xRDY#** signal can be actively driven earlier when the transaction will not complete during the initial block. **xRDY#** is allowed to be deasserted and tri-stated between TPs although the timing diagrams do not illustrate this behavior.

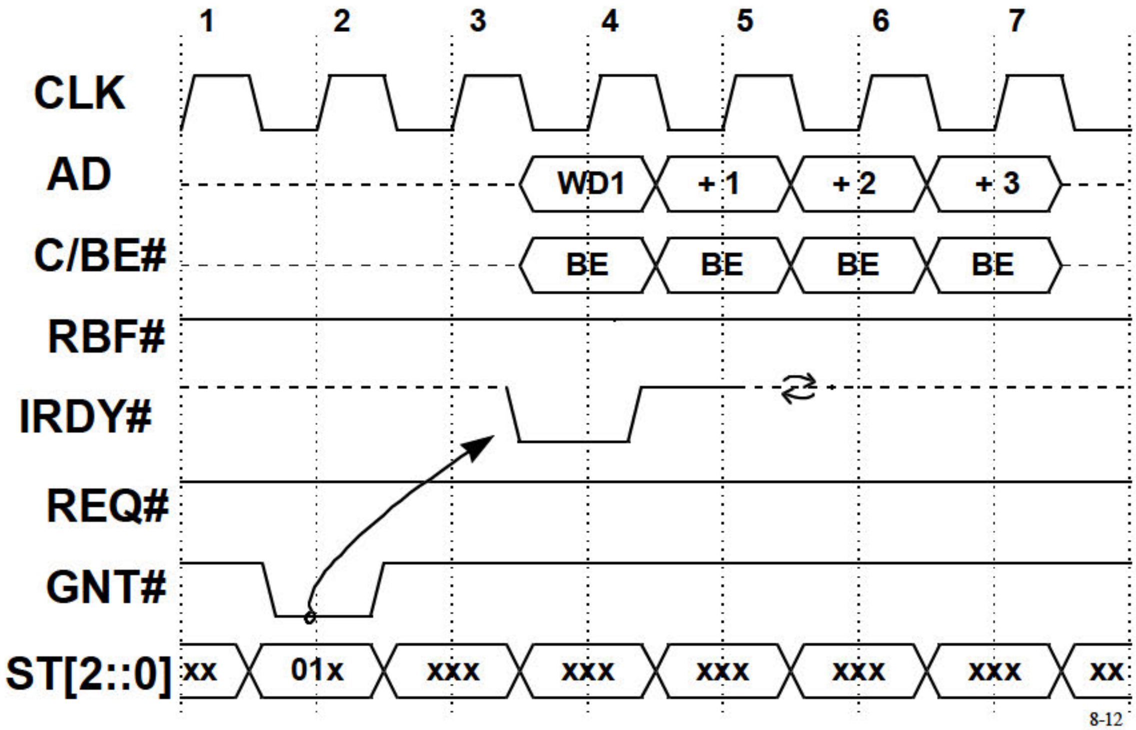


Figure 3-5 Maximum Delay by Master on Write Data

Throttling

Since the master is aware of the quantity of data it wants to send and can generate smaller write request if necessary, the master is not allowed to throttle write data. Write data is only allowed to be throttled by the target. The target is only allowed to throttle after each block is transferred. When the target wants to throttle the transfer of a subsequent block of write data, it must have **TRDY#** deasserted at the throttle point which occurs two 1x clocks prior to when the subsequent block would begin to transfer. The transfer of the subsequent block of data will resume two 1x clocks after **TRDY#** is sampled asserted. If throttling is not required by the target, **TRDY#** will be asserted at the throttle point. **TRDY#** is meaningless (it may be asserted, deasserted or tri-stated[12]) between throttle points but must be actively driven during a TP. When the last TP completes **TRDY#** must be deasserted and tri-stated. **TRDY#** also has no meaning on the last throttle point of a transaction that less than or a multiple of a block. For example, if less than 4 clocks are required to complete the transaction, then the next TP does not occur. In Figure 3-6, the first TP occurs on clock 4 and since the transaction completes before 10, the subsequent TP which would occur on clock 8, is not required and therefore does not exist. In this figure, the TP on clock 4 is the last and **TRDY#** must be deasserted on clock 5 and tri-stated on clock 6.

[12] When tri-stated, it must have been driven deasserted before being tri-stated since this is a sustained tri-state signal.

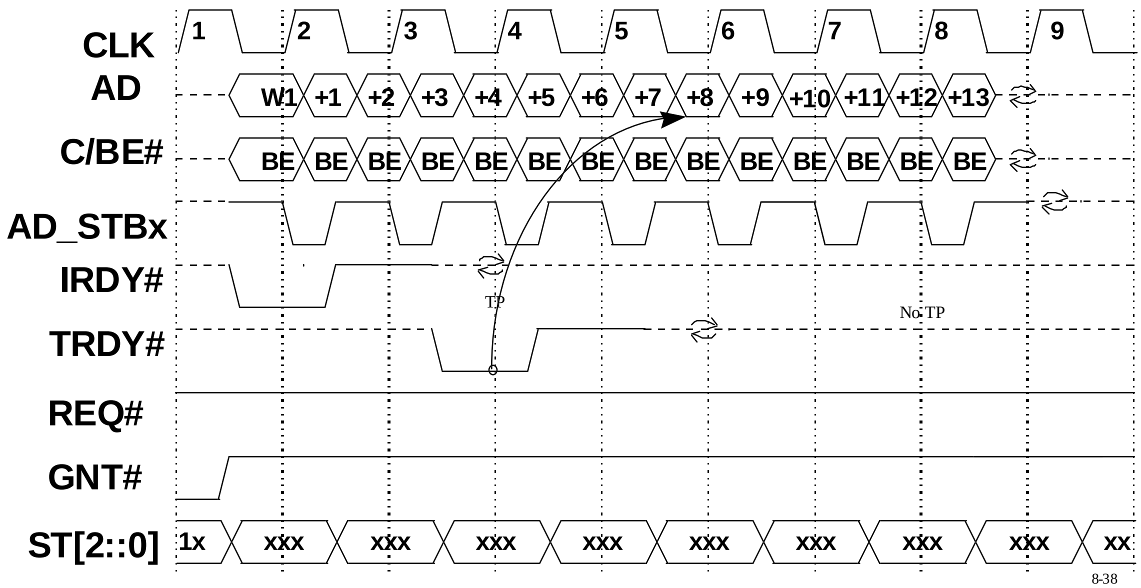


Figure 3-6 Write Data with One TP.

3.2.5.4 1 and 2 Clock Rule for IRDY# and TRDY#

For initial write data, **IRDY#** must be asserted by the master one or two clock edges after **GNT#** is sampled asserted when the **AD** bus is free and is one clock in duration. In the case where **GNT#** is pipelined to the master, **IRDY#** must be asserted on the first or second clock edge when the **AD** bus becomes free to complete the data transfer.

For initial read data, **TRDY#** must be asserted by the target one or two clocks after **GNT#** is sampled asserted when the **AD** bus is free and is one clock in duration. The target can not assert **TRDY#** on the same clock that it asserts **GNT#**. In the case where **GNT#** is pipelined, the one or two clock rule starts from the earliest time that **TRDY#** could be asserted.

3.2.5.5 Other Flow Control Rules

The agent receiving data should assert its flow control signal independent of the sender's flow control. For example, for low priority read data, the master must assert **RBF#** for the initial data block transfer and **IRDY#** for subsequent block data transfers independently of the assertion of **TRDY#**. On transfers of subsequent blocks of read data (where both **IRDY#** and **TRDY#** need to be asserted to continue), once **xRDY#** is asserted in a TP, it must remain asserted until both **IRDY#** and **TRDY#** are asserted on the same clock, which then completes the TP. In other words, once an agent has indicated that it is ready to continue the transfer it can not change its mind. Outside of the TP, the state of **xRDY#** is meaningless.

3.2.6 Source Throttling Address Flow Control

Address flow control for A.G.P. request is source controlled. This means that the A.G.P. compliant master is responsible for not enqueuing more requests than the target is capable of handling. System software reads the *RQ* field in the A.G.P. compliant target status register (see section 0) to learn the maximum number of requests that the target is capable of supporting. Software can also learn the maximum number of request slots supported by the master by reading *RQ* field in the A.G.P. compliant master status register (see section 0). Software then writes the master's *RQ_DEPTH* register in the command register with the value of the number of requests that the master can have outstanding. When the value is more than the master requested, the master limits the number of outstanding requests by design. When the value is less than the master requested, the master is not allowed to enqueue more requests than the maximum value programmed. This guarantees that the A.G.P. compliant target's request queue will never overflow.

The A.G.P. compliant master must track the number of outstanding requests it has issued. A slot in the master's request queue is considered "used" whenever a read, write, or FLUSH command is issued to the target. The request queue slot becomes available again for another request when data associated with that request starts to transfers across the bus. Since a FLUSH command is treated like a LP read, it consumes a slot until the dummy read data is returned. When the number of outstanding requests reaches the allocated limit the master is not allowed to generate further Read, Write, or FLUSH requests until a slot is freed.

3.3 Pin Description

There are 16 new A.G.P. interface signals defined. The A.G.P. compliant target is required to support all 16 signals. The number of A.G.P. signals required by the master determines the performance of the master. There are basically two independent choices the master makes to determine its performance level:

1. How requests are transferred to the core logic; and
2. At what rate will data transfer.

All A.G.P. compliant devices are required to support 4 new signals (**ST[2::0]** and **RBF#**). The following table lists which other A.G.P. signals are required by the master based on A.G.P. functionality supported. The core logic is required to support all 16 signals.

Table 3-5 A.G.P. Signals Required

	Address Queuing on **AD** Bus	Address Queuing on **SBA** Port
1x Data Transfer Rate	**ST[2::0]**, **RBF#**, **PIPE#**	**ST[2::0]**, **RBF#**, **SBA[7::0]**
2x Data Transfer Rate	**ST[2::0]**, **RBF#**, **PIPE#**, **AD_STB0**, **AD_STB1**	**ST[2::0]**, **RBF#**, **SBA[7::0]**, **AD_STB0**, **AD_STB1**, **SB_STB**

In addition to the new A.G.P. signals, all A.G.P. enabled components must have the PCI pin complement on the A.G.P. port as described in Table 3-10. The A.G.P. signals follow the signal type definitions and naming convention used in the PCI Specification.

The following signal type definitions are from the view point of the A.G.P. compliant target.

in *Input* is a input-only signal.

out *Totem Pole Output* is an active driver.

t/s *Tri-State* is a bi-directional, tri-state input/output pin.

s/t/s *Sustained Tri-State* is an active low tri-state signal owned and driven by one and only one agent at a time. The agent that drives a s/t/s pin low must drive it high for at least one clock before letting it float. A new agent cannot start driving a s/t/s signal any sooner than one clock after the previous agent tri-states it. A pull-up is required to sustain the inactive state until another agent drives it, and must be provided by the A.G.P. compliant target.

The following table lists the signal names in the first column, signal types in the second column and the signal descriptions in the third column. In the second column, the direction of a t/s or s/t/s signal is from the view point of the core logic and is represented in the parentheses. For example, **PIPE#** is a sustained tri-state signal (s/t/s) that is always an input for the core logic. The tables below describe their operation and use, and are organized in four groups: Addressing, Flow Control, Status and Clocking.

Table 3-6 A.G.P. Addressing

Name	Type	Description
PIPE#	s/t/s (in)	*Pipelined* request is asserted by the current master to indicate a full width request is to be enqueued by the target. The master enqueues one request each rising edge of **CLK** while **PIPE#** is asserted. When **PIPE#** is deasserted no new requests are enqueued across the **AD** bus. **PIPE#** is a sustained tri-state signal from a *master (graphics controller)* and is an input to the target *(the core logic).*
SBA[7::0]	in	*Sideband Address port* provides an additional bus to pass address and command to the target from the master. **SBA[7::0]** are outputs from a master and an input to the target. This port is ignored by the target until enabled (see section 6.1.9).

Table 3-6 contains two mechanisms to enqueue requests by the A.G.P. compliant master. The master chooses one mechanism at design time or during the initialization process and is not allowed to change during runtime. When **PIPE#** is used to enqueue addresses the master is not allowed to enqueue addresses using the **SBA** port. When the **SBA** port is used **PIPE#** can not be used.

Table 3-7 A.G.P. Flow Control

Name	Type	Description
RBF#	in	*Read Buffer Full* indicates if the master is ready to accept previously requested low priority read data or not. When **RBF#** is asserted the arbiter is not allowed to initiate the return of low priority read data to the master.

Table 3-7 contains the new Flow Control beyond normal PCI flow control already required. If the master is *always* ready to accept return data, the A.G.P. compliant master is not required to implement this signal and the corresponding pin on the target is tied (internally pulled up) in the deasserted state.

Table 3-8 A.G.P. Status Signals

Name	Type	Description
ST[2::0]	out	*Status* bus provides information from the arbiter to a Master on what it may do. **ST[2::0]** only have meaning to the master when its **GNT#** is asserted. When **GNT#** is deasserted these signals have no meaning and must be ignored. 000 Indicates that previously requested low priority read or flush data is being returned to the master. 001 Indicates that previously requested high priority read data is being returned to the master. 010 Indicates that the master is to provide low priority write data for a previous enqueued write command. 011 Indicates that the master is to provide high priority write data for a previous enqueued write command. 100 Reserved (Arbiter must not issue. May be defined in the future by Intel.) 101 Reserved (Arbiter must not issue. May be defined in the future by Intel.) 110 Reserved (Arbiter must not issue. May be defined in the future by Intel.) 111 Indicates that the master has been given permission to start a bus transaction. The master may enqueue A.G.P. requests by asserting **PIPE#** or start a PCI transaction by asserting **FRAME#**. **ST[2::0]** are always an output from the Core logic and an input to the master.

Table 3-8 describes the status signals, their meaning and indicate how the **AD** bus will be used for subsequent transactions. The **AD** bus can be used to enqueue new requests, return previously requested read data, or request the master to provide previously enqueued write data. The **ST[2::0]** are qualified by the assertion of **GNT#**.

Table 3-9 A.G.P. Clock list

Name	Type	Description
AD_STB0	s/t/s (in/out)	**AD** *Bus Strobe 0* provides timing for 2x data transfer mode on the **AD[15::00]**. The agent that is providing data drives this signal.
AD_STB1	s/t/s (in/out)	**AD** *Bus Strobe 1* provides timing for 2x data transfer mode on the **AD[31::16]**. The agent that is providing data drives this signal.
SB_STB	s/t/s (in)	*SideBand Strobe* provides timing for **SBA[7::0]** and is always driven by the A.G.P. compliant master (when supported).
CLK	t/s (in)	*Clock* provides timing for A.G.P. and PCI control signals.

3.4 A.G.P. Semantics of PCI signals

PCI signals, for the most part, are redefined when used in A.G.P. transactions. Some signals have slightly different semantics. **FRAME#, IDSEL, STOP#,** and **DEVSEL#** are not used by the A.G.P. protocol. The exact role of all PCI signals during A.G.P. transactions are described in **Table 3-10**.

Table 3-10 PCI signals in relation to A.G.P.	
FRAME#	Not used. **FRAME#** remains deasserted by its own pull up resistor.
IRDY#	New meaning. **IRDY#** indicates the A.G.P. compliant master is ready to provide *all* write data for the current transaction. Once **IRDY#** is asserted for a write operation, the master is not allowed to insert waitstates. The assertion of **IRDY#** for reads, indicates that the master is ready to transfer a subsequent block of read data. The master is *never* allowed to insert a wait state during the initial block of a read transaction. However, it may insert waitstates after each block transfers. *(There is no* **FRAME#** -- **IRDY#** *relationship for A.G.P. transactions.)*
TRDY#	New meaning. **TRDY#** indicates the A.G.P. compliant target is ready to provide read data for the entire transaction (when transaction can complete within four clocks)a block) or is ready to transfer a (initial or subsequent) block of data, when the transfer requires more than four clocks to complete. The target is allowed to insert wait states after each block transfers on both read and write transactions.
STOP#	Not used by A.G.P..
DEVSEL#	Not used by A.G.P..
IDSEL	Not used by A.G.P..
PERR#	Not used by A.G.P. (Optional for PCI operation per exceptions granted by PCI 2.1 specification.)
SERR#	Same as PCI. *(May be used by an A.G.P. compliant master to report a catastrophic error when the core logic supports a* **SERR#**[13] *pin for the A.G.P. port.)*
REQ#	Same as PCI. (Used to request access to the bus to initiate a PCI or an A.G.P. request.)
GNT#	Same meaning as PCI but additional information is provided on **ST[2::0]**. The additional information indicates that the master is the recipient of previously requested read data (high or low priority), it is to provide write data (high or low priority), for a previously enqueued write command or has been given permission to start a bus transaction (A.G.P. or PCI).
RST#	Same as PCI.
AD[31::00]	Same as PCI.

[13] An A.G.P. **SERR#** signal cannot be tied to the PCI **SERR#** signal because of different clocking domains. The assertion of **SERR#** must meet setup and hold times to **CLK**.

C/BE[3::0]#	Slightly different meaning. Provides command information (different commands than PCI) by the master when requests are being enqueued using **PIPE#**. Provides valid byte information during A.G.P. write transactions and is driven by the master. The target drives to "0000" during the return of A.G.P. read data and is ignored by the A.G.P. compliant master.
PAR	Not used by A.G.P..
LOCK#	Is not supported on the A.G.P. interface for either A.G.P. or PCI transactions.
INTA#, INTB#	Interrupt request signals[14] are the same as PCI and follow the same usage. (Must be level sensitive and shareable.) **INTA#** for a single function device, **INTB#** for a two function device. **INTC#** and **INTD#** are not supported on the A.G.P. connector.

3.5 Bus Transactions

3.5.1 Address Transactions

As described earlier in this A.G.P. interface specification there are two ways to enqueue requests, the **AD** bus or the **SBA** port. If the master chooses the **SBA** port it is not allowed to asserted **PIPE#** for any transactions. If the master uses **PIPE#** to enqueue requests it is not allowed to use the **SBA** port. The following diagrams will first illustrate the enqueuing of addresses on the **AD** bus and then on the **SBA** port.

3.5.1.1 AD Bus

The master requests the permission from the core logic to use the **AD** bus to initiate an A.G.P. request or a PCI transaction by asserting **REQ#**. The arbiter grants permission by asserting **GNT#** with **ST[2::0]** equal to "111" hereafter referred to as *START*. When the master receives START it is required to start the bus operation within two clocks of when the bus becomes available. For example, when the bus is in an idle condition when START is received, the master is required to initiate the bus transaction on the next clock and the one following. When a transaction is currently active on the bus when START is received, the master is required to start the bus operation within 2 clocks of when the bus becomes available. For example, when the current transaction is an A.G.P. read (from target to master), a turn around cycle is required between the last data phase of the read and the start of the request or assertion of **PIPE#** or **FRAME#**. In this example, the master is required to start the request either two or three clocks from the completion of the last data. The next clock after is not allowed since a turn around cycle is required when ownership of the **AD** bus changes. Once this has occurred the master is required to start the transaction either on the next clock or the one thereafter. For a write transaction, the turn around cycle is not required and therefore the master must initiate the transaction the clock following the completion of the last data phase or the clock after. Each of these relationships is described in section 3.6.

[14] These can be tied to the PCI **INTx#** signals since these are o/d signals and are level sensitive.

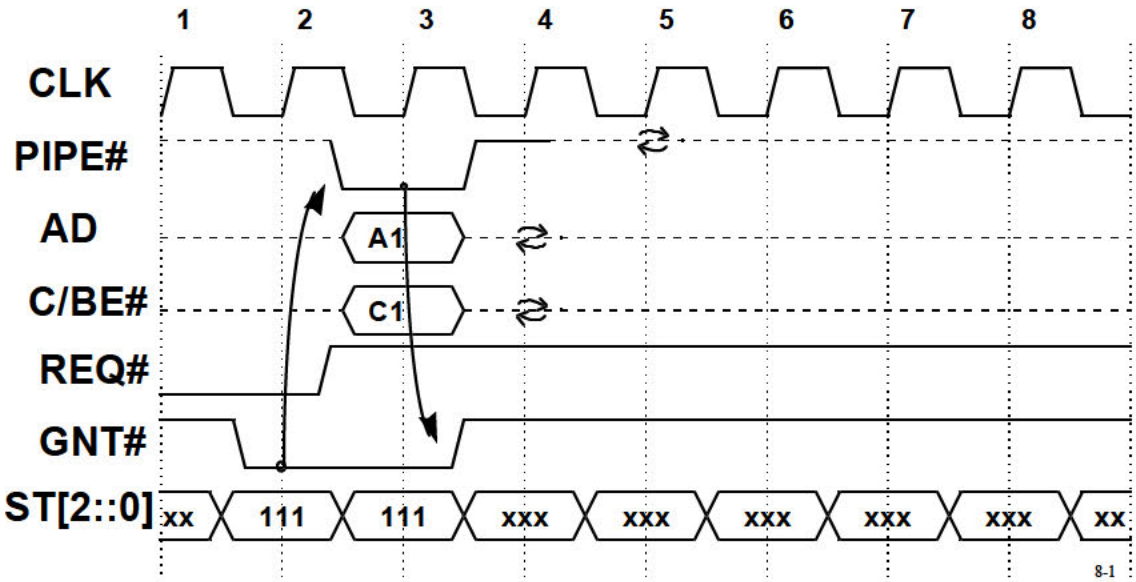


Figure 3-7 Single Address - No Delay by Master

Figure 3-7 illustrates a single address being enqueued by the master. Sometime before clock 1, the master asserted **REQ#** to gain permission to use the **AD** bus. The arbiter grants permission by indicating START on clock 2. A PCI only master is not required to start the transaction within 2 clocks. It is a violation of the A.G.P. protocol if an A.G.P. compliant master delays starting a request (assertion of **PIPE#** or **FRAME#**) by more than 2 clocks. A new request (address, command and length) are enqueued on each clock in which **PIPE#** is asserted. The address of the request to be enqueued is presented on **AD[31::03]**, the length on **AD[2::0]** and the command on **C/BE[3::0]#**. In this figure, only a single address is enqueued since **PIPE#** is asserted for only a single clock. The master indicates that the current address is the last it intends to enqueue when **PIPE#** is asserted and **REQ#** is deasserted which occurs in the figure on clock 3. Once the arbiter detects the assertion of **PIPE#** or **FRAME#** (which occurs on clock 3), it deasserts **GNT#** (on clock 4).

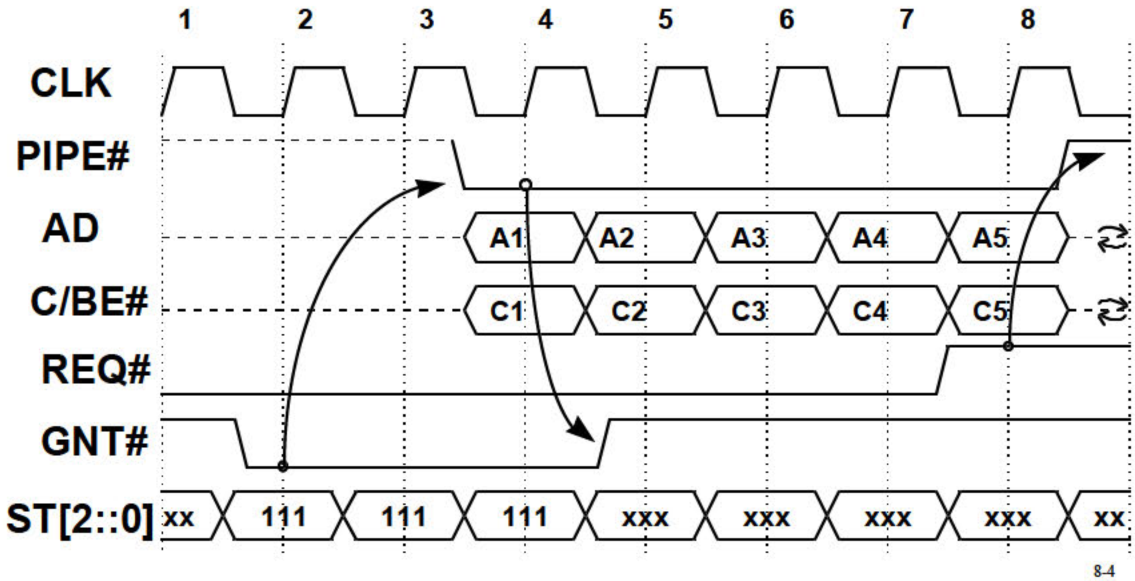


Figure 3-8 Multiple Addresses Enqueued, Maximum Delay by Master

Figure 3-8 illustrates a master that enqueues 5 requests, where the first request is delayed the maximum allowed by the A.G.P. interface specification, which is two clocks from the indication of START. In this case, START is indicated on clock 2, but the master does not assert **PIPE#** until clock 4. Figure 3-7 illustrates the earliest the master could start its request and Figure 3-8 illustrates the latest in which the master is allowed to start the request when the bus is idle. Note that **REQ#** remains asserted on clock 7 to indicate that the current request is not the last one. When **REQ#** is deasserted on clock 8 with **PIPE#** still asserted this indicates that the current address is the last one to be enqueued during this transaction. **PIPE#** must be deasserted on the next clock when **REQ#** is sampled deasserted. If the master desired to enqueue more requests during this bus operation, it would simply continue asserting **PIPE#** until all of its requests are enqueued or until the master has filled all the available request slots provided by the core logic.

The master is not allowed to insert any waitstates while enqueuing requests and the target has no mechanism to stop an address from being enqueued. Once **PIPE#** is asserted, every rising edge of **CLK**, enqueues a new request. The target has no mechanism to stop or delay the master from enqueuing more requests once the arbiter has indicated START. The clock following the last address, the master is required to tri-state the **AD** and **C/BE#** buses, unless the master is to provide write data (as indicated by **GNT#** and **ST[2::0]**) associated with a previously enqueued write request.

3.5.1.2 SBA Port

An A.G.P. compliant master may choose to use **SBA** port to enqueue requests instead of using **PIPE#** and the **AD** bus. The **SBA** port is always driven by the master and if not enqueuing new requests, the master must drive the NOP command on the port which is signaled by driving **SBA[7::0]** to "1111 1111" or FFh for 1x data transfers and "1111 1111 1111 1111" or FFFFh for 2x data transfers. The target ignores the **SBA** port until enabled to decode it. All commands on the **SBA** port always come in pairs except for the NOP case when 1x data transfer is done. In this case, a NOP is a single clock in duration. Unlike the enqueuing requests on the **AD** bus, the master does not request use of the port, but simply sends the request at anytime(when a request slot is available). If a subsequent command is near a previous command then only the lower address bits and length need to be transferred. The target will use previous upper address bits and command to initiate a new memory access. With this abbreviated addressing, the **AD** bus can be completely utilized transferring small pieces of data that are in close to each other. In the diagrams, the notion of "R1H and R1L" indicate that this is request 1 high and request 1 low. High refers to the upper 8 bits (where the OP code resides) and Low refers to the lower 8 bits. A request can be a Type 1, Type 2 or a type 3 command as described in section 3.1.3.

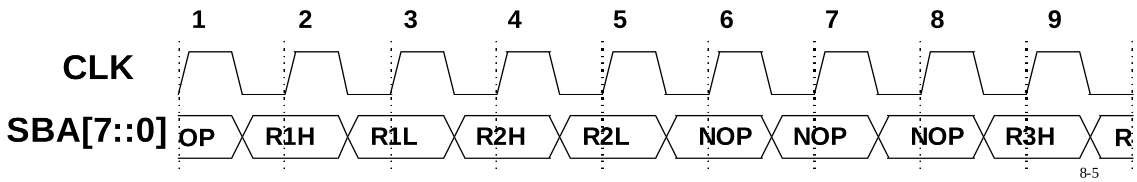


Figure 3-9 1x SideBand Addressing

In Figure 3-9 the master sends the NOP encoding on clock 1 and sends the high bits of the of a Type x (1, 2 or 3) on clocks 2, 4 and 9 and the low order bits on clocks 3 5, and 10. The master send NOPs on clocks 6, 7 and 8 to indicated that the **SBA** port does not contain a new request. There is no specific sequence in which Type 1, 2 or 3 encodings are required to transfer the **SBA** port. In this figure, every non-NOP time could be a Type 1 or only Type 3 commands. Recall that memory accesses are only initiated when a Type 1 encoding is decoded by the target. A Type 2 simply stores updated middle addresses and command in the Type 2 register of the target. A Type 3 encoding updates the upper address bit in the Type 3 register. Only when a Type 1 command is received does the target reconstruct an address by using the Type 3 and Type 2 registers with the Type 1 value and enqueues it to the memory controller.

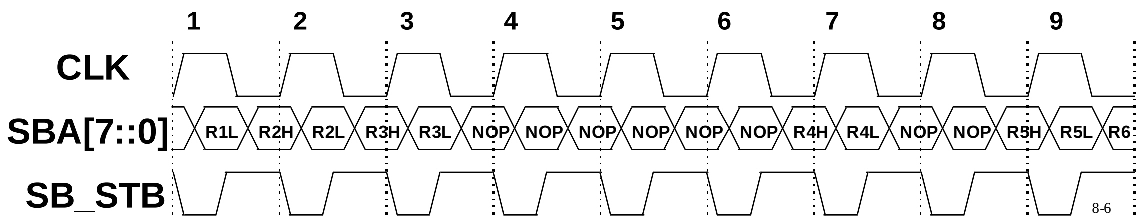


Figure 3-10 2x Side Band Addressing

Figure 3-10 illustrates the **SBA** port operating in 2x mode. In this mode, a new address is transferred across the **SBA** port each **CLK**. This figure is the same as the previous one except that both pieces of the encoding, the high and low portions, transfer across the port during a single **CLK** period.

3.5.2 Basic Data Transactions

As described earlier, data transfers across the port as independent transactions from the request that initiated the data movement. The following sections will discuss data movement of 1x and 2x transfer modes, with each discussing basic read and write data transfers.

3.5.2.1 1x Data Transfers

Read Data

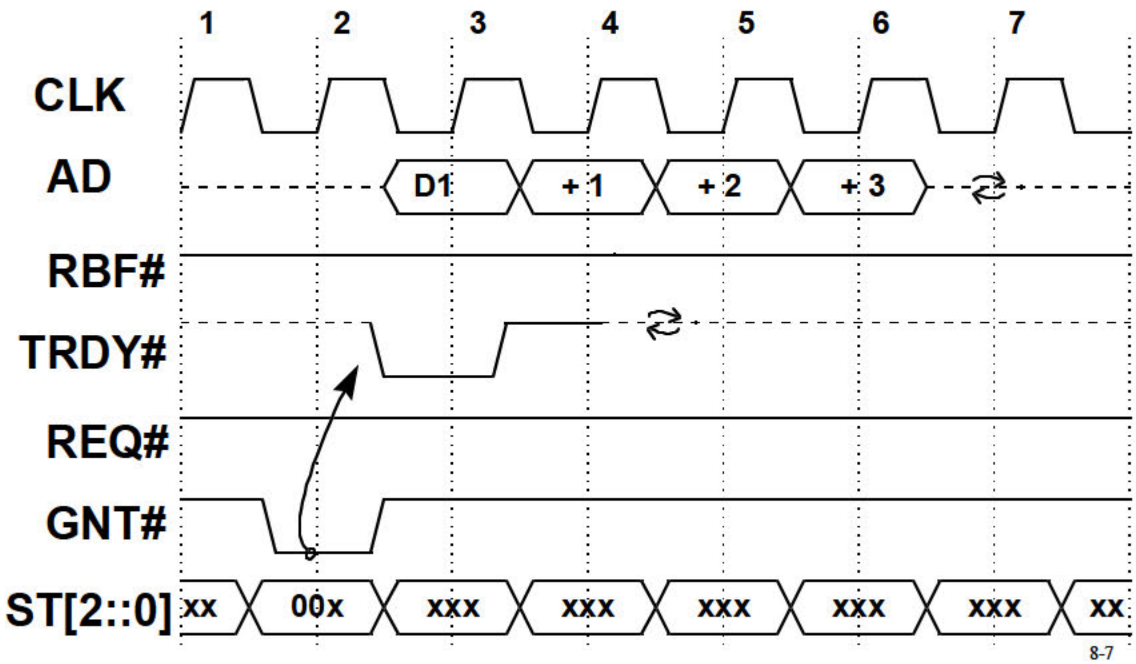


Figure 3-11 Minimum Delay by Target of Read Transaction

Figure 3-11 illustrates the returning of read data that was previously requested by the master. The bus is in an idle condition and the arbiter indicates to the master that the next transaction to appear on the **AD** bus is read data for the master. This is indicated by the assertion of **GNT#** with the **ST[2::0]** being 00x. To signal Low Priority Read data returning, the **ST** encoding would be "000", and High Priority Read data being indicated by "001". In the diagrams where the **ST** encoding is 00x, the data being moved could be low or high priority data. In those cases, that it makes a difference to the type of read data being returned the **ST** encodings will be either "000" or "001".

The master is informed that the read data is coming, when **GNT#** is asserted and **ST[2::1]** equals "00" which occurs on clock 2. The master knows that the next time **TRDY#** is asserted, that the **AD** bus contains valid data. Once **GNT#** has been asserted for read data, the master starts latching the **AD** bus on each rising clock and qualifies the data with **TRDY#**. When **TRDY#** is deasserted, the data is not valid. The entire transaction will complete without waitstates, once **TRDY#** is sampled asserted and the transaction completes within 4 clock. In this figure, only 16 bytes of data are being transferred to the master. Notice that **TRDY#** is a single pulse and that there is no **IRDY#** handshake as is done on PCI. When the transfer size of the read data can complete within 4 clocks , neither the master nor target are allowed to do flow control (waitstates) on the transaction. The **C/BE#** bus does not contain valid byte enables since the smallest addressable size of memory is 8 bytes and all 8 bytes are always returned. The **C/BE#** bus is driven by the A.G.P. compliant target to "0000" and the byte enables are ignored by the master. Once **TRDY#** has been asserted, it must be deasserted by the following clock (unless it will be asserted again) and tri-stated. This is shown in this figure by a solid line being driven high, then on the next clock the signal is tri-stated. The signal is held in this state by a pull-up. This is referred to as a sustained tri-state signal and is the same as **TRDY#** as defined by PCI.

Figure 3-11 illustrates the earliest the target can return data to the master once **GNT#** has been asserted indicating a read data transfer. Notice that there is no **PIPE#** or **SBA** port in the figure, the transaction in which data is returned to the master is the same no matter how the request was transferred to the target.

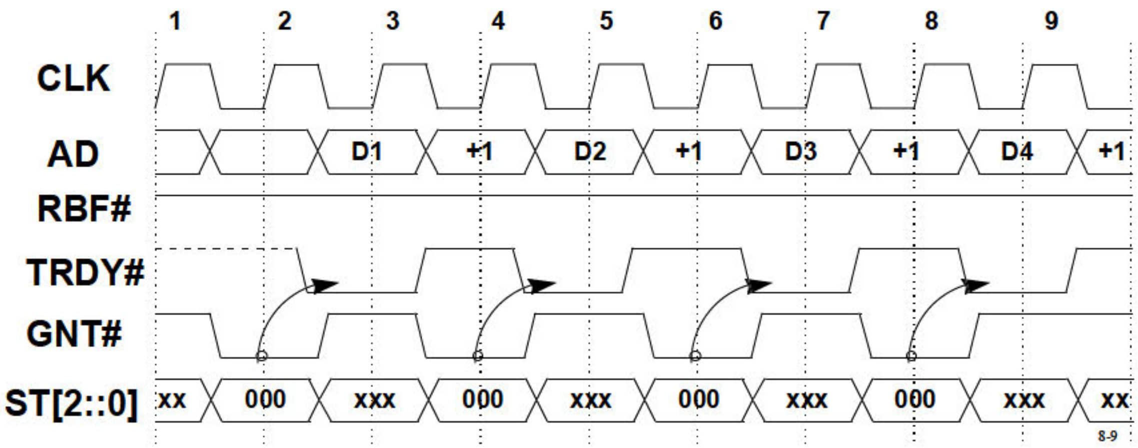


Figure 3-12 Minimum Delay on Back to Back Read Data

Figure 3-12 illustrates a stream of 8 byte read operations being returned to the master. This figure shows that the arbiter is indicating to the master that read data is being returned on every clock. Remember that the minimum transfer size is 8 bytes and in 1x transfer mode, it requires two clocks to return the data. Therefore enqueuing **GNT#**'s earlier accomplishes nothing. The arbiter will not assert **GNT#** for a new transaction until the last clock of the current read transaction.

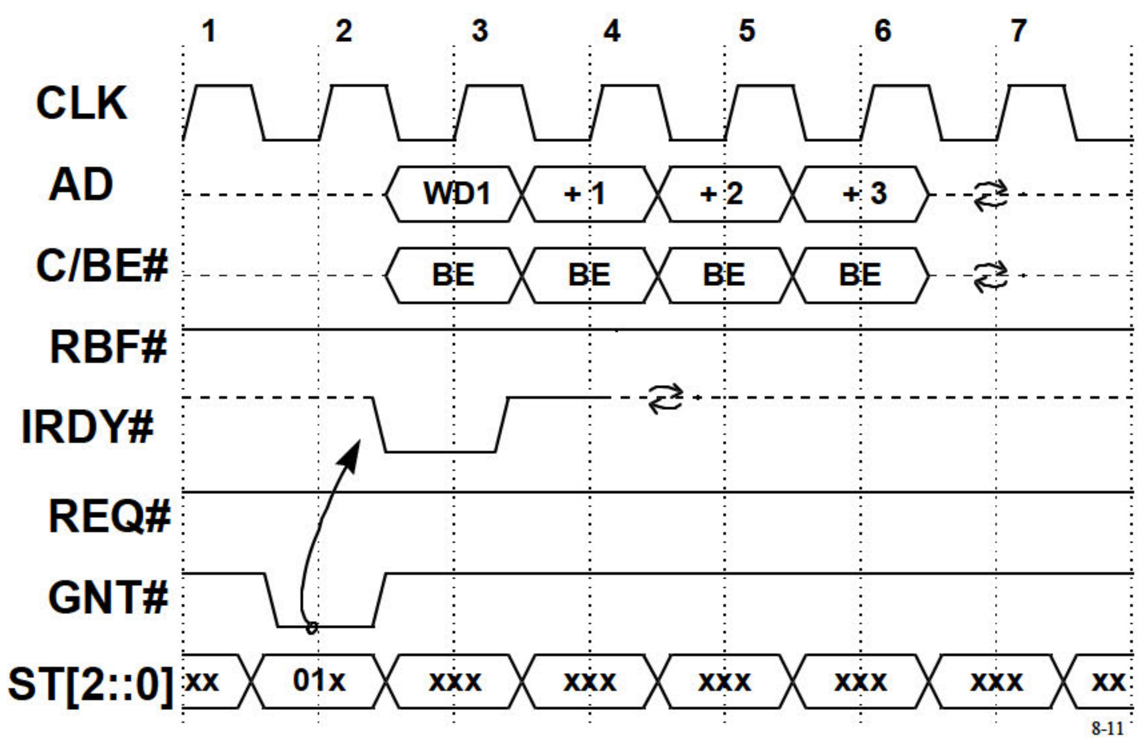


Figure 3-13 Master does not Delay Providing Write Data

Figure 3-13 shows a basic write data transfer. The arbiter indicates to the master that write data should be provide to the core logic which is indicated by the assertion of **GNT#** and **ST[2::0]** being "010 or 011". The first being a low priority write data and the second indicating that it is a high priority write data. In this example the signaling is the same and therefore the "01x" value is used. The master is required to provide the write data within 2 clocks of the indication from the arbiter. In this example, the master provides the data immediately because the bus was idle. The assertion of **IRDY#** is for a single pulse and goes with the first piece of data to indicate to the target that data is valid. Once **IRDY#** has been asserted, data transfers 4 bytes per CLK until the transaction has completed (for transactions that complete within four clocks). In this example the transaction is 16 bytes and completes in 4 clocks. The master is required to deassert and then tri-state **IRDY#** after it was asserted. The data is transferred on the **AD** bus while the **C/BE[3::0]#** provide the byte enables. The byte enables indicate which byte lanes carry meaningful data. The target is not allowed to delay the movement of write data (initial data block) after **GNT#** and **ST** bus indicate a write data transfer.

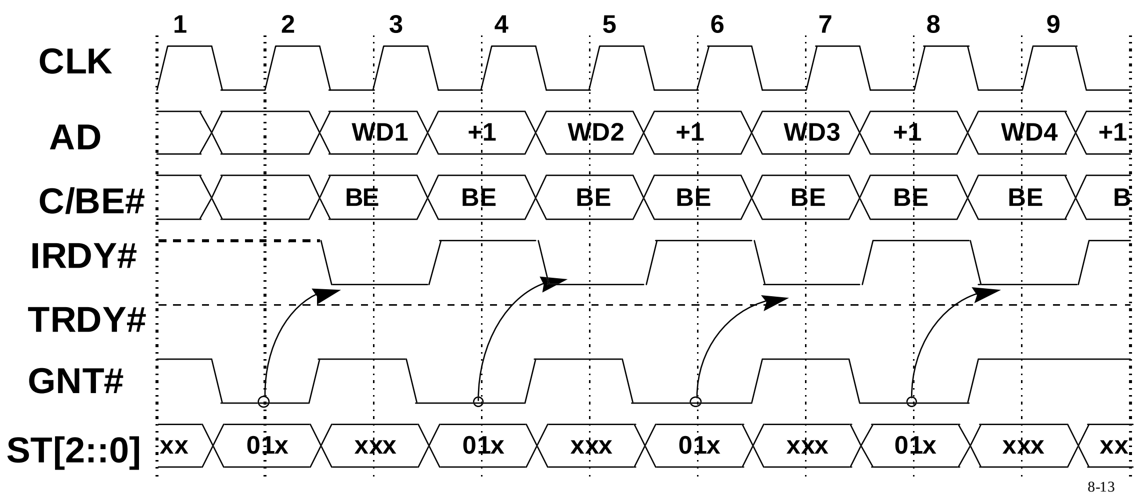


Figure 3-14 Back to Back Write Data Transfers - no Delays

Figure 3-14 is an example of back to back write data transfers. Each of these transactions are 8 bytes and could be either high priority or low priority write data transfers. On clock 2, the arbiter indicates to the master to provide previously requested write data to the core logic. Since these are small transfers the arbiter provides a **GNT#** on every other clock. Since a new transaction begins on clock 3, 5, 7 and 9, the master asserts **IRDY#** on these clocks to indicate that the first piece of data of each transaction is valid on the **AD** bus.

3.5.2.2 2x Data Transfers

This section discusses 2x data transfers. Basically this is the same as 1x clocking except an entire 8 bytes are transferred during a single **CLK** period. This requires that two 4 byte pieces of data are transferred across the **AD** bus per **CLK** period. First a read data transfers will be discussed and then a write transfer.

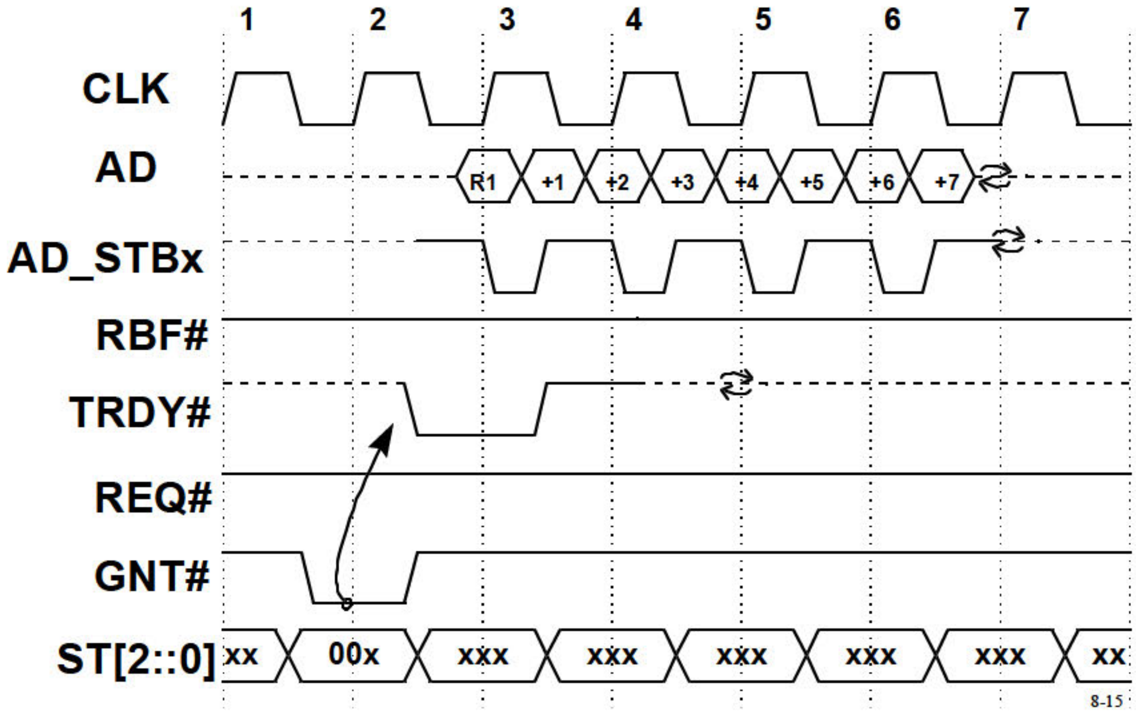


Figure 3-15 2x Read Data no Delay

Figure 3-14 is the same as Figure 3-11 except that 16 bytes are transferred in 4 clocks, while in this figure 32 bytes are transferred during the same 4 clocks. The control signals are identical. The **AD_STBx** signal has been added when data is transferred at 8 bytes per **CLK** period. **AD_STBx** represents **AD_STB0** and **AD_STB1** and are used by the 2x interface logic to know when valid data is present on the **AD** bus. The control logic (**TRDY#** in this case) indicates when data can be used by the internal consumer of the data.

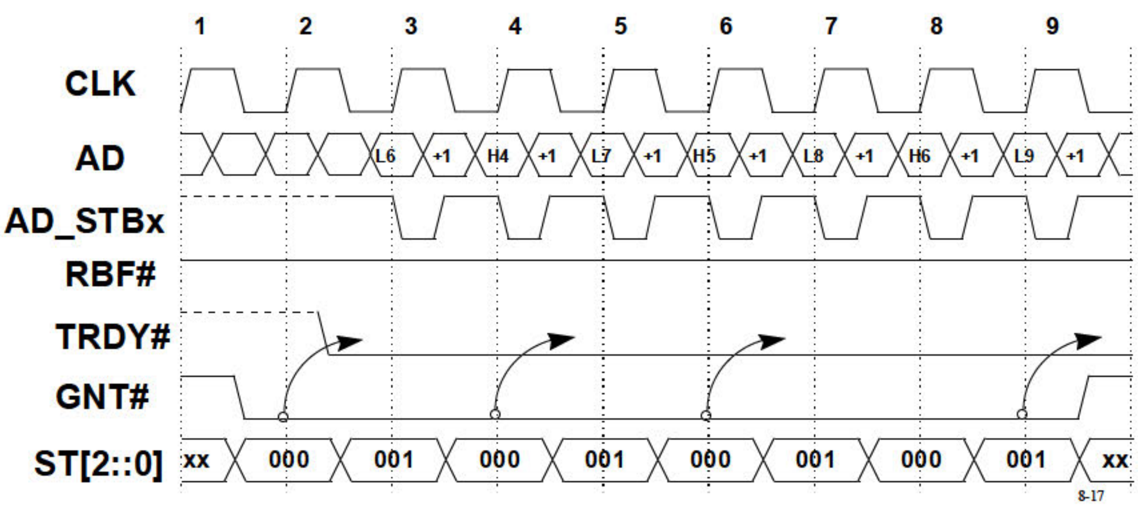


Figure 3-16 2x Back to Back Read Data - No Delay

Figure 3-16 shows back to back 8 byte read transactions. The **ST[2::0]** toggle between "000"and "001" to illustrate that they are actually changing. However, they are not required to change between HP and low priority in order to do back to back transactions. In this diagram, **TRDY#** must be asserted on each clock since a new transaction starts on each clock.

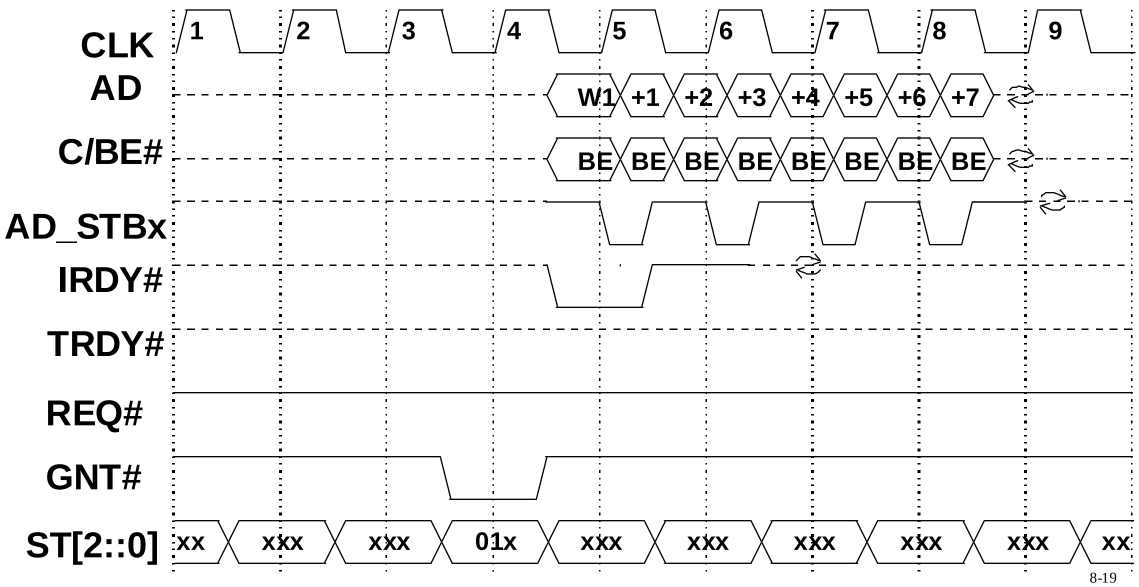


Figure 3-17 2x Basic Write No Delay

Figure 3-17 is a basic write transaction that transfers data at the 2x rate. This figure is the same as Figure 3-13 (1x basic write) and there is no difference in the control signals and only more data is moved. The normal control signals determine when data is valid or not. This diagram moves 32 bytes of data in the same time as 16 bytes are moved in Figure 3-13.

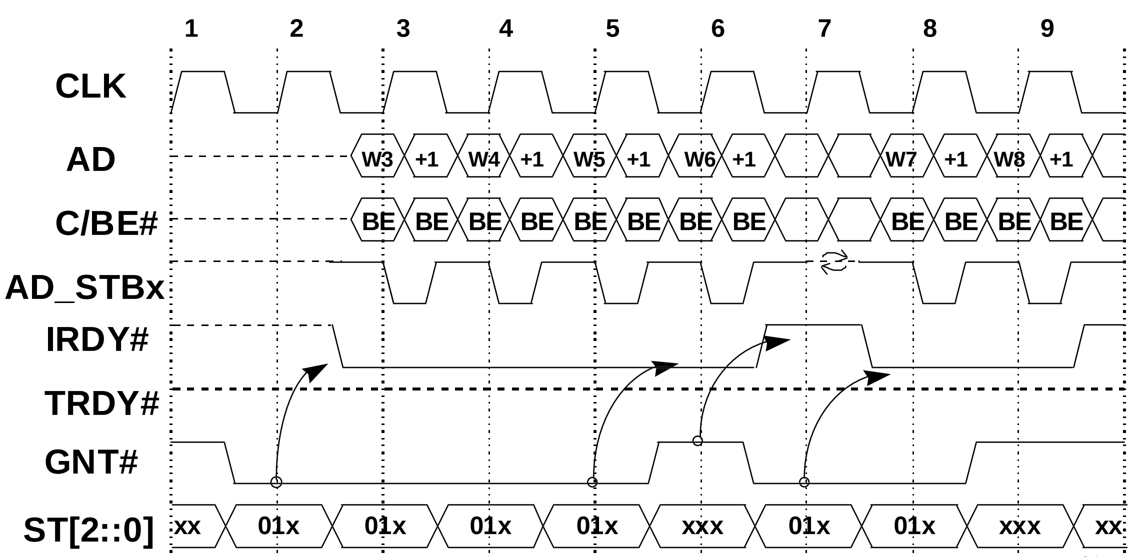


Figure 3-18 QuadWord Writes Back to Back - No Delays

Figure 3-18 instead of a single transfer with 32 bytes of data, these diagrams illustrates multiple 8 byte write operations. When the transactions are short, the arbiter is required to give grants on every clock or the **AD** bus will not be totally utilized. In this example a new write is started on each rising clock edge except clock

7, because the arbiter deasserted **GNT#** on clock 6. Since a new transaction is start on each **CLK**, the **IRDY#** signal is only deasserted on clock 7.

3.5.3 Flow Control

3.5.3.1 Initial Data Block

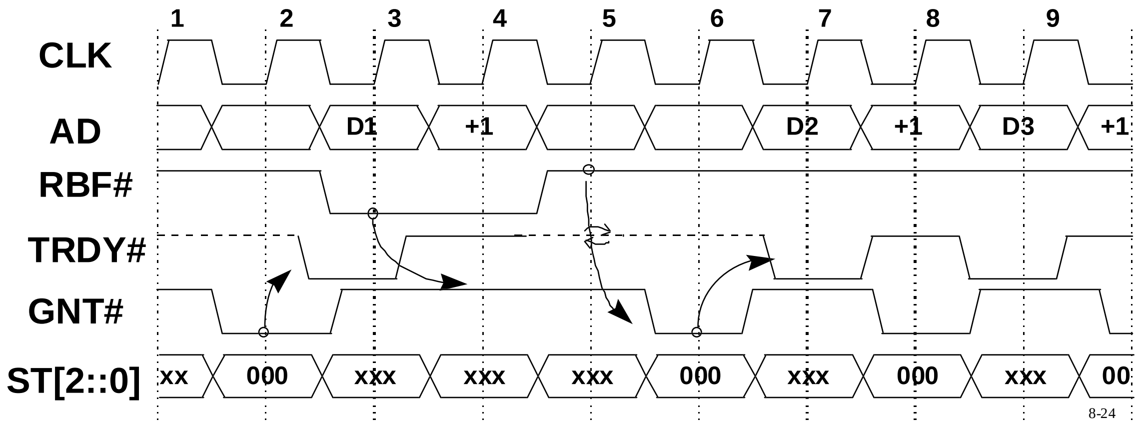


Figure 3-19 Master's Data Buffer is Full - No Delay on Read Data

Figure 3-19 is a diagram where the master indicates to the target that it can accept the current transaction but has no buffer available for any additional transactions at this time. This is indicated by the master by asserting **RBF#** on clock 3. In this figure, the master asserts **RBF#** the clock following the assertion of **GNT#** to prevent the arbiter from returning additional LP read transactions. As illustrated in the previous diagram, the arbiter only asserts **GNT#** on every other clock at 1x data rates because that is all that is required to keep the **AD** bus 100% busy. Enqueuing the **GNT#** earlier only causes the master to provide more buffering for the return of read data and does not improve performance or efficiency of the bus.

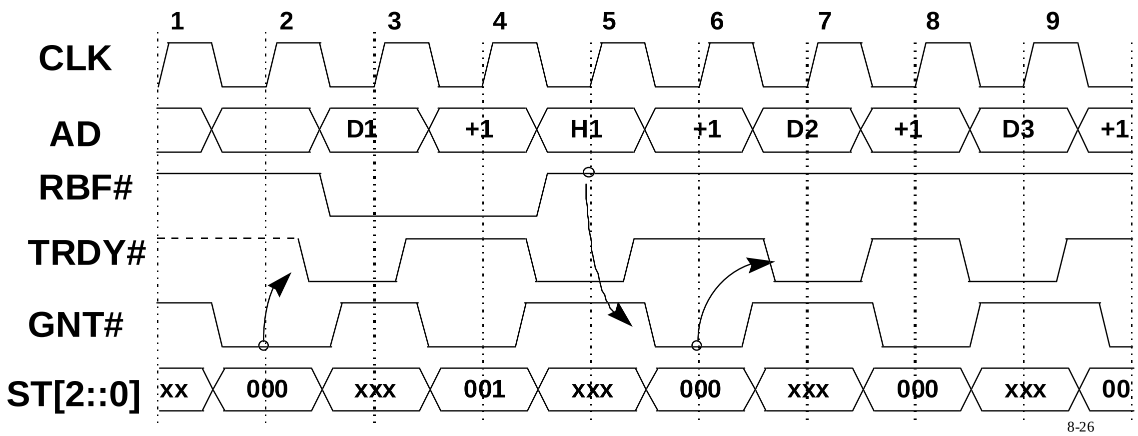


Figure 3-20 High Prority Read Data Returned While RBF# Asserted

Figure 3-20 is the same as Figure 3-19 except that the arbiter returns high priority read data when the master indicates that it is not capable of accepting read data by asserting **RBF#**. The target is allowed to return High Priority (HP) data at any time, including when the master is not able to accept LP read data. **RBF#** only applies to “LP read data and not to HP read data. The master indicates that it is ready to continue accepting LP read data on clock 5 by deasserting **RBF#**. The arbiter indicates that HP read data is being returned on clock 4 by asserting **GNT#** and **ST[2::0]** are "001". The data transfer for HP read is the same as a read data, with **TRDY#** being asserted on the initial data phase.

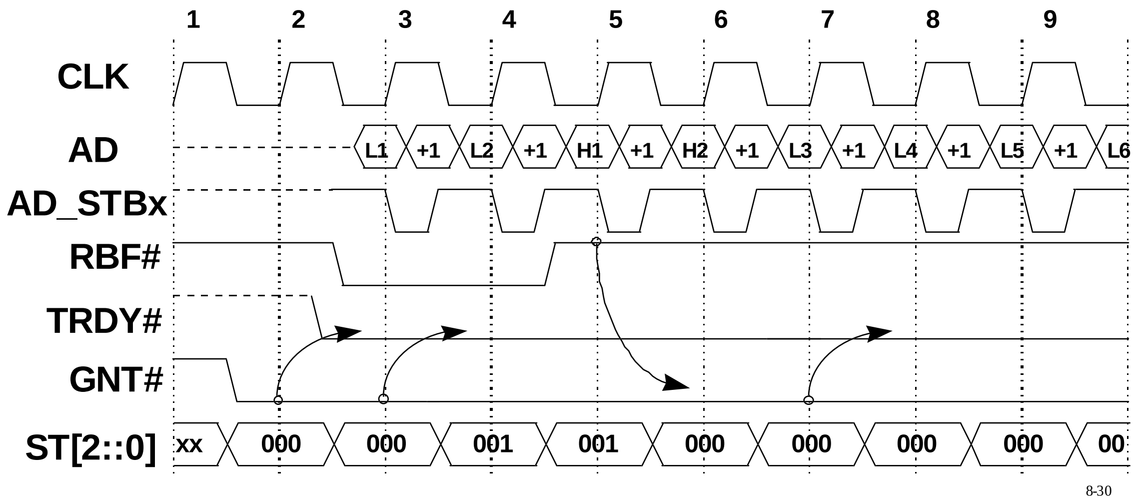


Figure 3-21 2x Read Data with RBF# Asserted

Figure 3-21 shows the master asserting **RBF#** indicating to the target that the masters read data buffer is full and it is not capable of accepting any new data. In order to ensure that 8 byte read operations can occur without delays on the **AD** bus, the arbiter must enqueue a new grant on each clock. In this case, the master must be able to accept 2 transactions worth of data because the arbiter is driving the second grant on the same clock in which the master is driving **RBF#**. Therefore, the master must provide a minimum of buffer for two transactions, as in this case 8 + 32 bytes of buffering. In general the master for read transactions (assuming the master is capable of asserted **RBF#** the clock following the assertion for **GNT#** indicating that read data is being returned. If the master delays the assertion of **RBF#** or desires to minimize the frequency in which it stalls the return of read data, more buffering should be provided. This figure, interleaves HP read data when the master indicates that it can not accept any more LP read data. If HP data was not pending, the **AD** bus would have been dead on clocks 5 and 6.

3.5.3.2 Subsequent Data Block

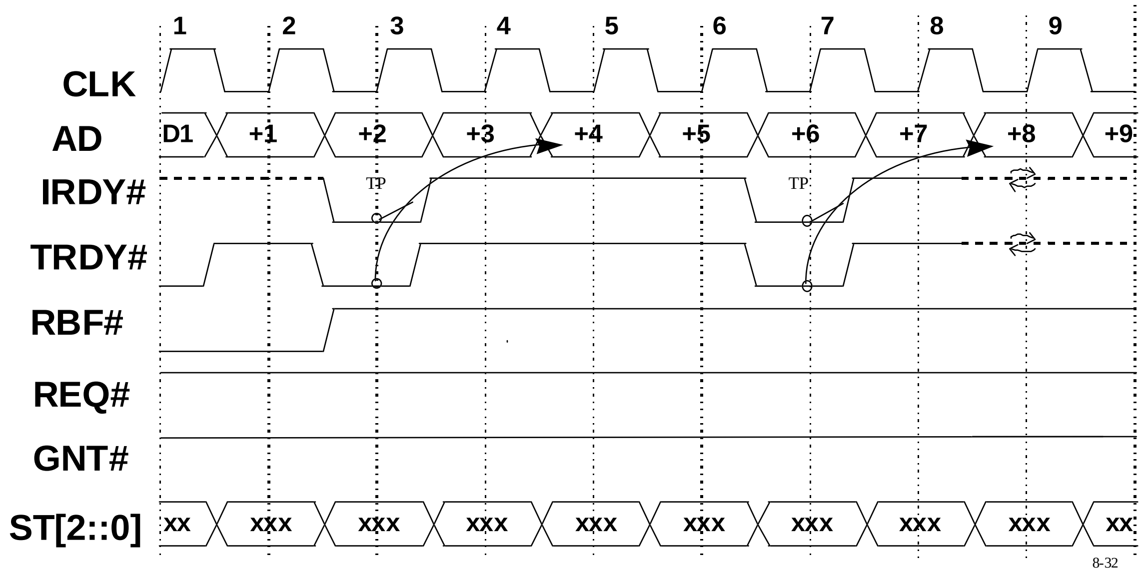


Figure 3-22 Throttle Point for Subsequent Data Block - No Delay

For a transaction that requires more than four clocks to complete, both the master and target are allowed to insert waitstates. This point is referred to as a throttle point (TP in the diagrams). After every four clocks that data transfers, both the master and target are allowed to delay the next block of data from being transferred. Neither agent can cause the subsequent block from transferring, but can only delay it. In Figure 3-22 the first TP point occurs on clock 3, while the second occurs on clock 7. The TP point always occurs two clocks before the subsequent block of data would start to transfer. Another way to describe it for the 1x data transfers, the TP occurs on the clock in which the 9th-12th bytes transfer for the first TP and 25th - 28th bytes for the second.

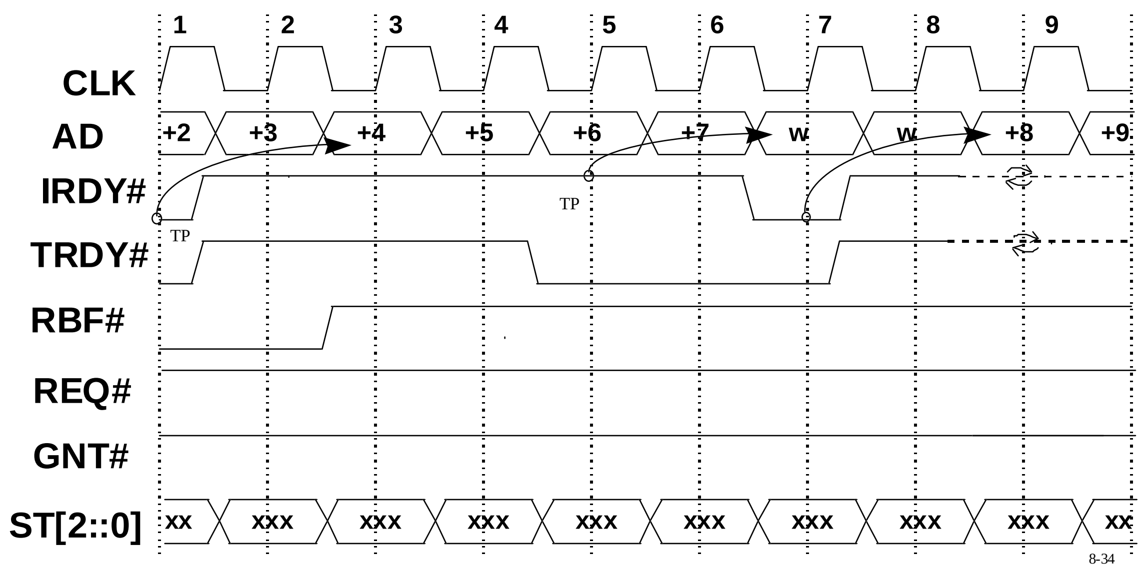


Figure 3-23 Master Delays Subsequent Data Block

Figure 3-23 illustrate the TP where the target indicated that it is ready to transfer data while the master delays the transfer by two clocks. When both **IRDY#** and **TRDY#** are asserted which occurs on clock 7, the subsequent block of data begins to transfer two clocks later. Note that once **TRDY#** is asserted it must remain asserted until **IRDY#** is asserted, at which point both must be deasserted. A wait state is inserted on the bus on clock 7 and 8 because **IRDY#** was not asserted on clocks 5 and 6. The TP starts on clock 5 and ends on clock 7. **xRDY#** must be actively driven during the entire TP.

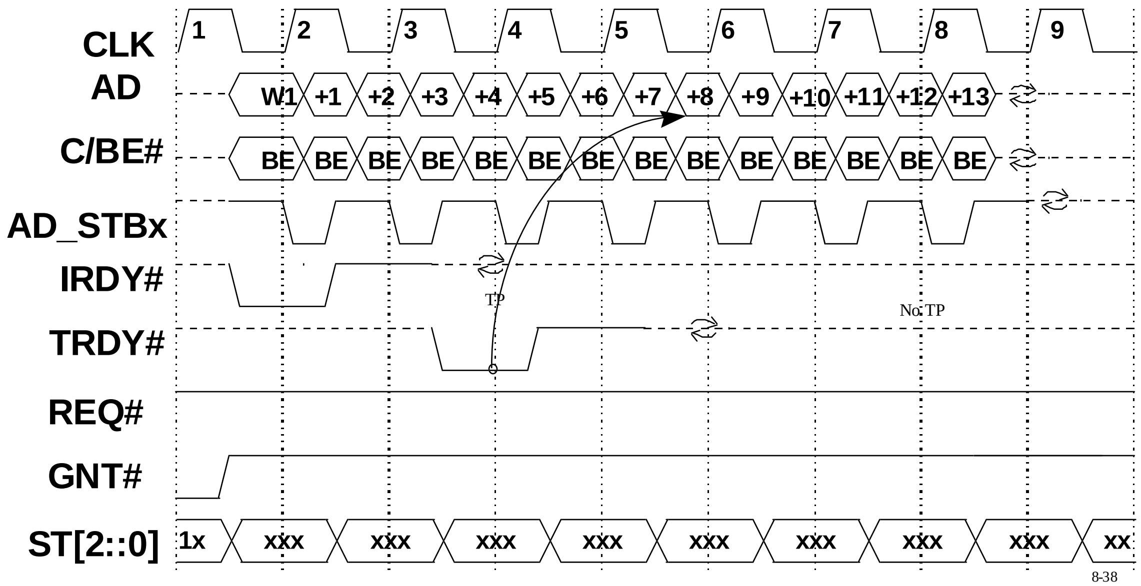


Figure 3-24 Write with Subsequent Block - No Delay

Figure 3-24 is a write transaction that requires more than four clocks to complete and therefore has a subsequent block on data. This figure shows the transaction completing without waitstates because the target was ready for the subsequent block of data by asserting **TRDY#** for clock 4. Since the transaction does not cross into a second subsequent block, the throttle point which would occur on clock 8 is meaningless. Only the target is allowed to flow control a write once it has started to transfer data.

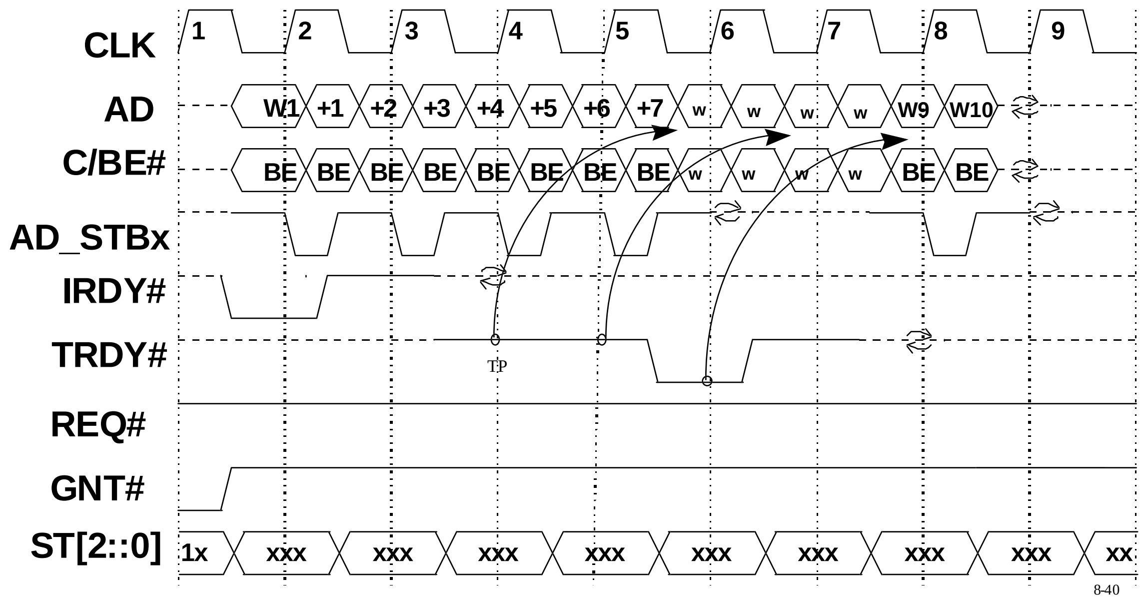


Figure 3-25 Target Delays Subsequent Write Data Block

Figure 3-25 is the same diagram as Figure 3-24 **Write with Subsequent Block - No Delay** except that the target is not ready in the case. Because **TRDY#** was deasserted on clocks 4 and 5, waitstates are inserted on clocks 6 and 7. Because **TRDY#** was asserted on clock 6, the first data phase of the second block is transferred on clock 8. The "w" on clocks 6 and 7 indicate that data could be driven or not driven but must be valid on clock 8. The master is not required to drive meaningful data for clocks 6 and 7 since **TRDY#** indicates that the target will not accept the data until clock 8.

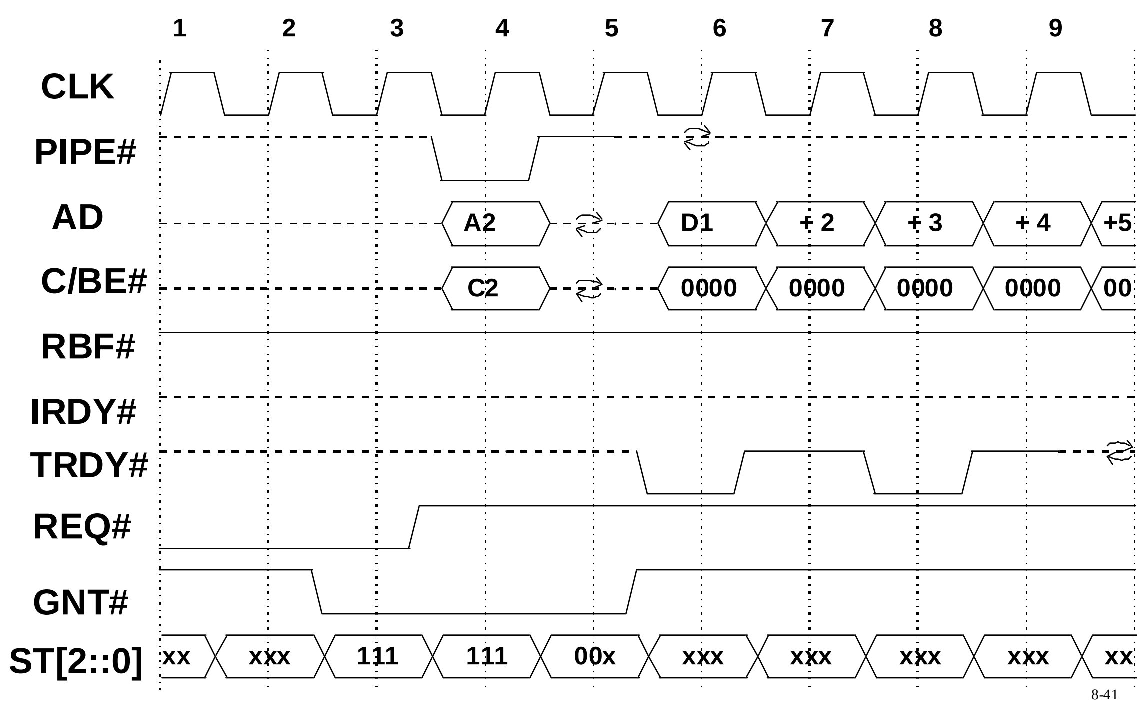


Figure 3-26 Earliest Read Data Returned after a Request

Figure 3-26 illustrates the earliest that read data can be returned to a master following the enqueuing of an address by the master. A turn around cycle is required when ownership of the **AD** bus occurs. The data being returned on clock 6 was requested sometime ago and is not related to the address being enqueued on clock 4. Clock 5 is the earliest the master can be informed of the return of read data. Figure 8-44 illustrates when more requests are enqueued but the **GNT#** for the data transfer remains the same. Figure 8-45 illustrates the target inserting a wait state before providing the data. Again notice that **TRDY#** is only asserted for a single clock. Once **GNT#** has been asserted indicating the return of read data, the master watches for the assertion of **TRDY#** to indicate that the data for the first data phase is valid. Subsequent data phases complete one per clock for the entire transaction when the transaction completes within four clocks. For transactions that require more than four clocks to complete, a subsequent block is required and a TP is valid which occurs on clock 8.

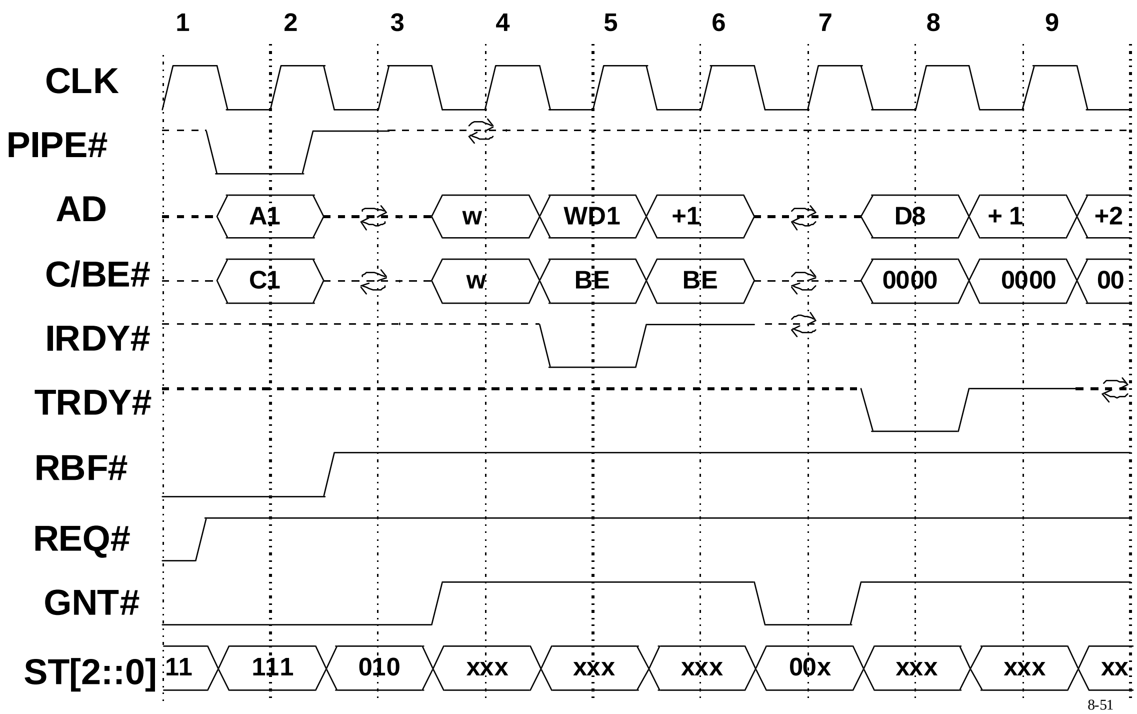


Figure 3-27 Request followed by Write and then a Read

Figure 3-27 shows the sequence of enqueuing a single address, followed by write data being transferred, followed by read data being transferred. A turn around cycle is required on the bus each time ownership of the **AD** bus changes. The turn around on clock 3 occurs because the master did not know until clock 3 that it is to provide write data. If more requests had been enqueued then the turn around access could have been avoided. The master delayed the write data by one clock. Figure 8-51 and Figure 3-29 show multiple requests followed by write data where there is no turn around cycle.

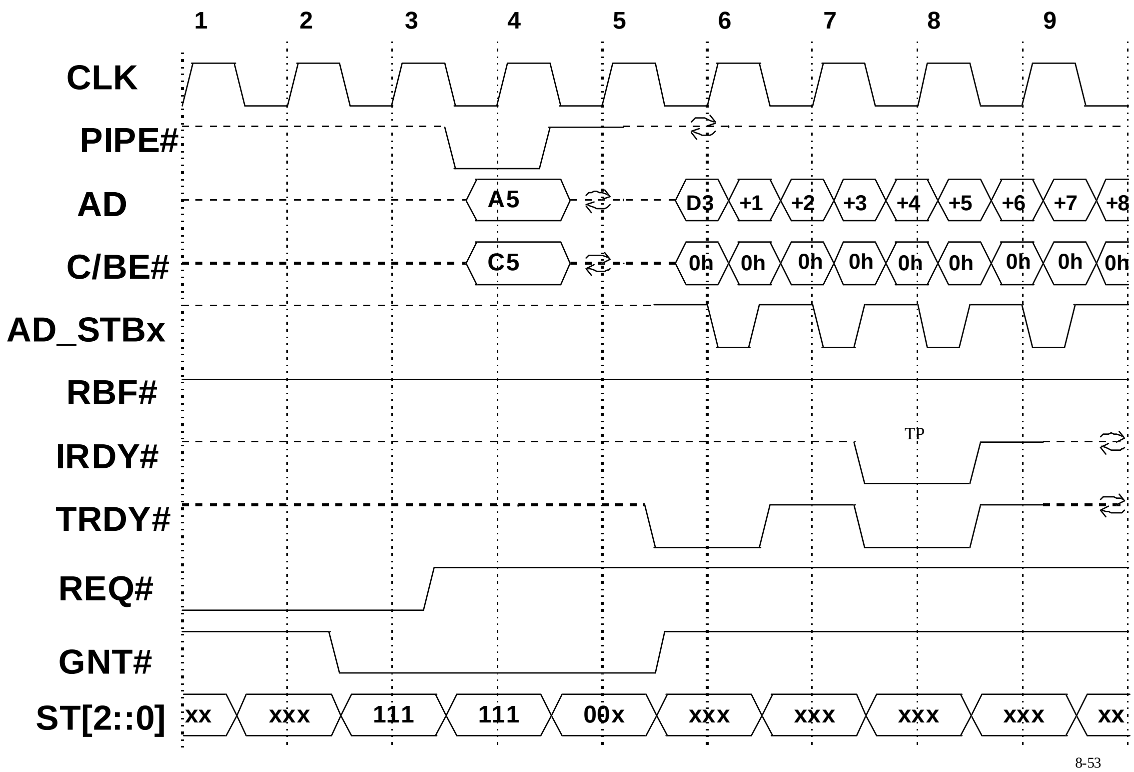


Figure 3-28 Request followed by Read Data no Delay by Target.

Figure 3-28 shows the earliest that read data can be returned following a request being enqueued. One turn around clock is required since ownership of the **AD** bus occurs. Even without the turn around cycle the arbiter is not able to give a grant to start the read data transfer because, the **ST** bus must be held until the transaction starts, which occurs on clock 4. The arbiter then changes the encoding for clock 5.

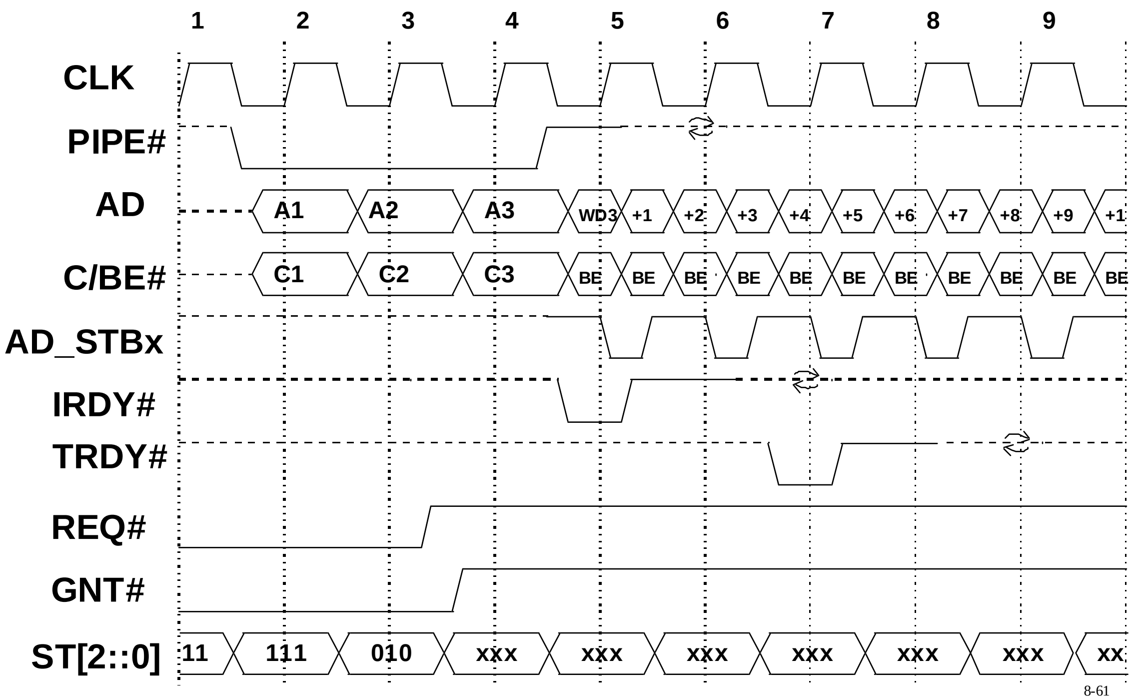


Figure 3-29 Request followed by Write - no AD Turn around

Figure 3-29 illustrated how combining the process of enqueuing new requests and then moving write data eliminates a turn around cycle on the bus. Since **GNT#** is asserted on clock 3 to indicate that the master should provide write data immediately after the requests have been enqueued. The master does not tri-state the **AD** or **C/BE** buses, but starts driving write data. The assertion of **IRDY#** indicates that the write transaction has started and valid write data will appear on each clock. The target indicates that it is ready to accept the second block of data on clock 9 by asserting **TRDY#** on clock 7.

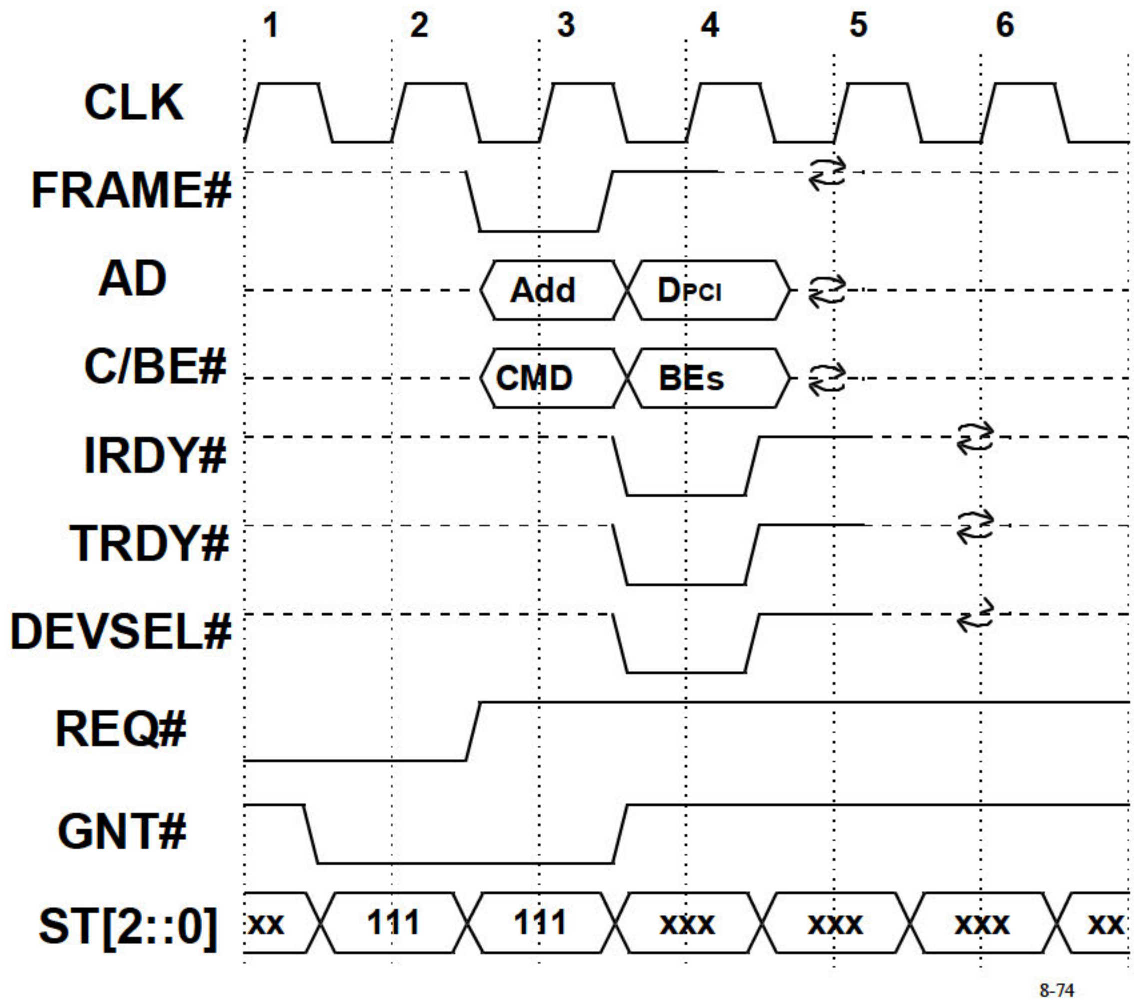


Figure 3-30 Basic PCI Transaction on A.G.P.

Figure 3-30 is a basic PCI transaction on the A.G.P. interface. If the PCI agent is a non A.G.P. compliant master, it ignores the **ST[2::0]** signals and everything appears to be a PCI bus. For those masters that are APG aware, then the **ST** bus indicates that permission to use the interface has been granted to initiate a request and not to move A.G.P. data.

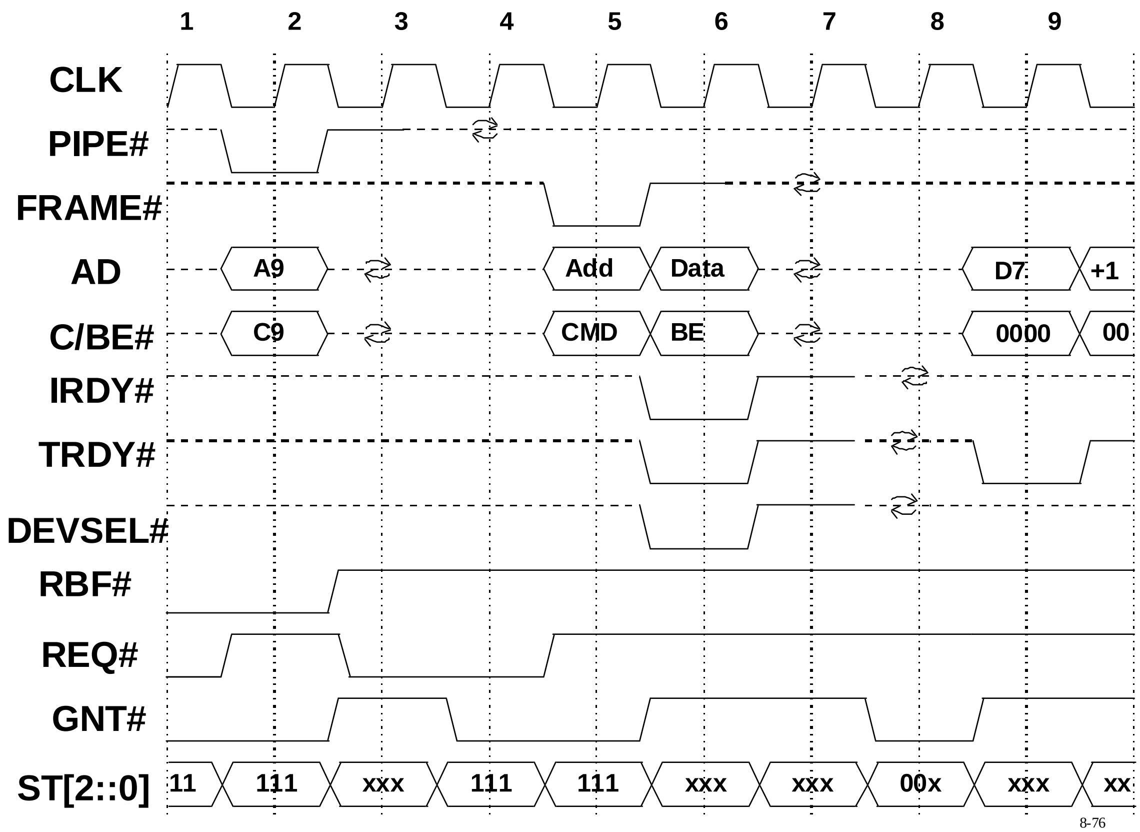


Figure 3-31 PCI Transaction Between A.G.P. Request and Data

Figure 3-30 shows the quickest that the A.G.P. compliant master can enqueue a request, perform a PCI transaction and then start returning read data. Two clocks are required between the completion of the A.G.P. request and the start of a PCI transaction, this is required because the A.G.P. **REQ#** line must be deasserted when **PIPE#** is asserted to indicate that the current request is the last to be enqueued. The earliest the A.G.P. compliant master can request a PCI transaction is on clock 3, and the earliest the arbiter can grant permission is on clock 4, which allows the PCI compliant master to initiate its request on clock 6. The two clocks between the PCI transaction and the read data is caused because of potential contention on **TRDY#**. This can occur when the PCI compliant master is the core logic and the target is the A.G.P. compliant master.

3.6 Arbitration Signaling Rules

3.6.1 Introduction

This section describes the rules that the A.G.P. compliant master's **REQ#** signal and the A.G.P. compliant arbiter's **GNT#** signal need to follow for correct A.G.P. operation. These rules are a necessary part of the A.G.P. protocol.

The rules associated with the master's **REQ#** output signal provide an early indication to the A.G.P. compliant arbiter as to when an access request transaction will complete. The arbiter may take advantage of this to eliminate idle bus clocks between transactions.

The rules associated with the **GNT#** signal minimize the amount of read data buffering required in the master while allowing back-to-back 8 byte transactions without idle bus clocks. In order to achieve back-to-back data transactions the arbiter may pipeline grants . The master must be able to accept them. Some of the rules in this section are necessary to limit the number of pipelined transactions that can be outstanding. This section will not attempt to describe the arbitration priority algorithms. This is a chipset specific implementation issue.

3.6.2 A.G.P. Compliant Master's **REQ#**

The A.G.P. compliant master asserts its **REQ#** signal when it wants to issue either a PCI cycle or enqueue requests using **PIPE#**. The master will de-assert **REQ#** depending on the type of access request. When issuing an A.G.P. access request over the **AD** bus using **PIPE#** the master must keep its corresponding **REQ#** asserted until one clock prior to de-asserting **PIPE#** asserted and **REQ#** deasserted on the same clock edge is an early indication that the current access request transaction is the last and **PIPE#** will deasserted one clock later. The arbiter may utilize this to avoid idle bus clocks when asserting **GNT#** for a subsequent transaction. This rule implies that **REQ#** will be deasserted for at least one clock between back-to-back **PIPE#** access request transactions. The master should concatenate as many address requests as possible into a single **PIPE#** access request transaction.

When an A.G.P. compliant master or a pure PCI compliant master issues a PCI transaction using **FRAME#** (and no other access requests are pending) it will de-assert **REQ#** when it asserts **FRAME#**. If another access request is pending the master will keep its **REQ#** asserted.

These rules are summarized in Table 3-11:

		Current Access Request	
		PCI using **FRAME#**	A.G.P. using **PIPE#**
Next Access Request	PCI	Keep **REQ#** asserted.	De-assert **REQ#** one clock prior to de-asserting **PIPE#**.
	A.G.P.	Keep **REQ#** asserted	De-assert **REQ#** one clock prior to de-asserting **PIPE#**. Concatenate if possible.
	None	De-assert **REQ#** when asserting **FRAME#**.	De-assert **REQ#** one clock prior to de-asserting **PIPE#**.

Table 3-11 A.G.P. Arbitration Rules

Figure 3-8 **Multiple Addresses Enqueued, Maximum Delay by Master** shows an access request using **PIPE#**. The master deasserts **REQ#** one clock prior to de-asserting **PIPE#**. Simultaneous SideBand and AD Access Request Generation is Not Allowed. An A.G.P. compliant master that is configured to issue commands over the sideband signals is not allowed to generate commands with **PIPE#** over the **AD** bus.

3.6.3 **GNT#** and **ST[2::0]**

The A.G.P. compliant arbiter will assert **GNT#** to initiate PCI or A.G.P. (non-sideband) activity. The **ST[2::0]** signals are only meaningful while **GNT#** is asserted and are used to communicate the type of PCI or A.G.P. activity being initiated. The **ST[2::0]** encodings are shown in Table 3-8 **A.G.P. Status Signals**.

3.6.4 **GNT#** for Single Transactions

For PCI and A.G.P. access requests **GNT#** will stay asserted until the arbiter samples either **FRAME#** or **PIPE#** asserted. The A.G.P. compliant master must drive **PIPE#** or **FRAME#** so that it is asserted either one or two clocks after the clock it samples **GNT#** asserted. Therefore **GNT#** will be asserted to an A.G.P. compliant master for a minimum of two clocks and a maximum of three clocks (for a single access request when the bus is idle). If the A.G.P. compliant master does not assert **PIPE#** or **FRAME#** from either the same clock that **GNT#** is first sampled asserted or the following clock, the arbiter may de-assert **GNT#** and consider the master inoperative. A pure PCI compliant master[15] on the A.G.P. bus may take longer to assert **FRAME#** after sampling its **GNT#** asserted. A pure PCI compliant master may be considered inoperative if it doesn't drive **FRAME#** within 16 idle bus clocks after **GNT#** is asserted.

For read and write data transfers, **GNT#** will be asserted along with the corresponding **ST[2::0]** signals for one clock cycle per transaction. This is summarized in Table 3-12.

A.G.P. Transaction Type	**GNT#** Duration
PCI Cycle	Until **FRAME#** sampled asserted
AD Access Request	Until **PIPE#** sampled asserted
Read Data	One 1x clock period per transaction
Write Data	One 1x clock period per transaction

Table 3-12 GNT# Duration

Figure 3-31 shows an A.G.P. compliant master asserting **REQ#** to run a PCI cycle. The master samples **GNT#** asserted on clock edge #4 with **ST[2::0]** encoding of '111' indicating permission to generate either a PCI cycle or an A.G.P. request. The master is allowed to take one or two clocks to assert **FRAME#**. In this example the master asserts **FRAME#** one clock after sampling **GNT#** asserted. Since no subsequent access request is pending the master de-asserts **REQ#** at the same time it asserts **FRAME#**. The arbiter samples **FRAME#** asserted on clock edge #3 and de-asserts **GNT#** clock 6. In this case **GNT#** is asserted for two clocks. If the master would have taken an additional clock to assert **FRAME#** the arbiter would have asserted **GNT#** for three clocks. Once the arbiter asserts **GNT#** (**ST**=111), the arbiter will continue driving it until either **PIPE#** or **FRAME#** are sampled asserted.

Figure 3-11 shows a read data transaction. The arbiter asserts **GNT#** for a single clock with an **ST[2::0]** encoding of '00x' indicating permission for the target to drive either high or low priority read data. Both the master and the target sample **GNT#** asserted on clock edge #2. The master must be ready to accept data on the next clock edge. The target is allowed to take one or two clocks to assert **TRDY#** and begin driving read

[15] A PCI 2.1 compliant master that does not generate A.G.P. transactions.

data. In this example the target asserts **TRDY#** and begins driving read data one clock after sampling **GNT#** asserted. **GNT#** is only asserted for one clock since this is a single read transaction consisting of four data phases.

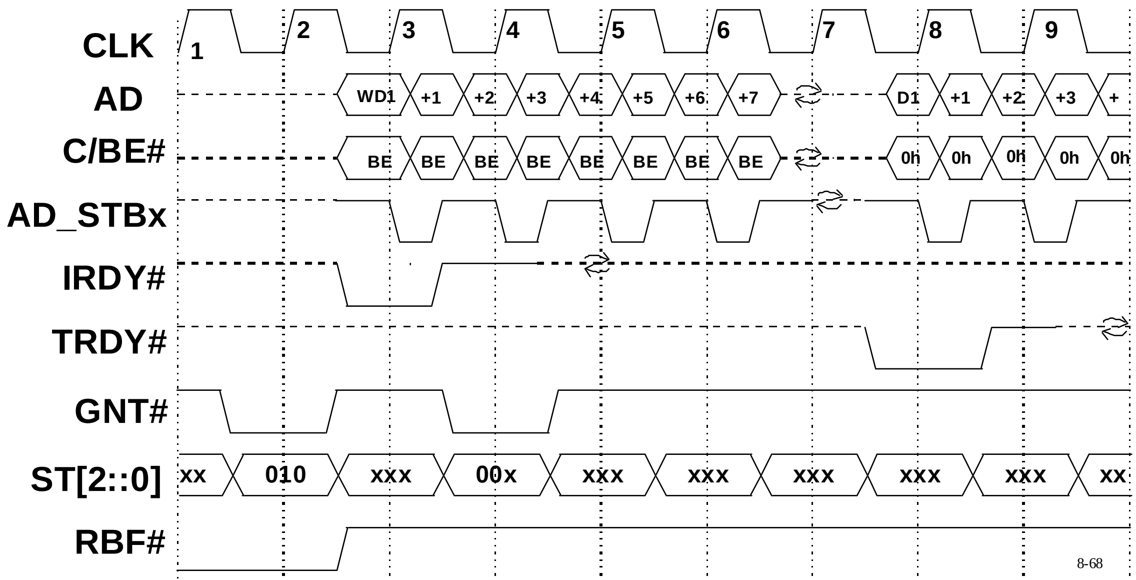


Figure 3-32 Write Data Followed by Read

Figure 3-32 shows a 32 byte write followed by a read. The arbiter asserts **GNT#** on clock edge #2 for a single clock with an **ST[2::0]** encoding of '010' indicating permission for the master to drive low priority write data. Both the master and the target sample **GNT#** asserted on clock edge #2. The target must be able to accept write data on the next clock. The master is allowed to take one or two clocks to assert **IRDY#** and begin driving write data. In this example the target asserts **IRDY#** and begins driving write data one clock after sampling **GNT#** asserted. **GNT#** is only asserted for one clock since this is a single write transaction consisting of eight data phases.

3.6.5 **GNT#** Pipelining

In order run back-to-back 8 byte data transactions (in 2x data transfer mode) without idle bus clocks between transactions the arbiter must pipeline **GNT#**s. The arbiter limits the number of outstanding **GNT#**s resulting from pipelining, to minimize the master's **GNT#** tracking logic. The master must be able to support the same number of outstanding pipelined **GNT#**s. The rules associated with attaining these goals are documented in this section.

When **GNT#** is pipelined the new bus driver is responsible for correctly sequencing from the current transaction to the next. If an idle bus clock is required between transactions to allow for bus turnaround, the new bus driver is responsible for guaranteeing the turn around bus clock.

- If **GNT#** is pipelined for an access request or for write data the master is responsible for correctly sequencing from the previous transaction to the next.

- When **GNT#** is pipelined for read data the target is responsible for correctly sequencing from the previous transaction to the next.

The rules governing the earliest point that **GNT#** may be pipelined for the next transaction is solely dependent on the current transaction type. If the current transaction is read data, the arbiter must wait to drive

GNT# for the next transaction such that **GNT#** is first sampled asserted on the last data phase of the current read. The last data phase is defined as the last rising 1x clock edge of the data transaction. This rule (along with proper use of the **RBF#** signal) minimizes the amount of low priority read data buffering required in the master. For a sequence of back-to-back 8 byte data transactions (in 2x data transfer mode) **GNT#** will be asserted on every 1x clock edge since, by definition, every 1x clock edge is the last data phase of a transaction.

If the current transaction is write data, **GNT#** for the next transaction can be asserted on the clock immediately following the **GNT#** for the current write data while there are less than **four** outstanding write data **GNT#s.** The arbiter tracks the number of outstanding write data **GNT#**s and will only assert a **GNT#** for a subsequent transaction if there are less than 4 outstanding write data **GNT#**s. The arbiter increments its write data **GNT#** counter when it asserts **GNT#** for write data and decrements the counter when it samples **IRDY#**[16] asserted by the master for a write data transaction. The *master must be able to handle five pipelined* **GNT#s** (this assumes that a master doesn't consider a **GNT#** "canceled" until the data transaction has finished, one request currently being handled and four more enqueued). This rule allows back-to-back 8 byte write data transactions to proceed when the master takes two clocks to assert the initial **IRDY#** after sampling **GNT#** asserted.

If the current transaction is a **PIPE#** request, **GNT#** for a data transaction can be asserted immediately following the **GNT#** for the current access request. Since **REQ#** will stay asserted (but doesn't indicate another request) until one clock prior to **PIPE#** de-assertion, it is impossible to pipeline a **GNT#** for another PCI or **PIPE#** access request if the current transaction is a **PIPE#** access request. Note that a **GNT#** for a **PIPE#** access request could immediately be followed by up to four **GNT#**s for write data transfers (or three writes and one additional transaction). The master's **GNT#** pipeline logic must be able to handle this case.

If the current transaction is a PCI cycle, **GNT#** for the next transaction can be asserted immediately following the **GNT#** for the current PCI cycle. Note that a **GNT#** for a PCI cycle could immediately be followed by up to four **GNT#**s for write data transfers (or three writes and one additional transaction). The master's **GNT#** pipeline logic must be able to handle this case. An A.G.P. pipelined transaction is not allowed to start (after a PCI transaction) until the bus is IDLE (**FRAME#** and **IRDY#** deasserted) for one clock. Table 3-13 entries refer to the earliest clock edge off which the arbiter can drive **GNT#** asserted for the next cycle.

---

[16] The count is 4 when a latched version of **IRDY#** is used to decrement the number of outstanding grants. Since the target could use either a latched or unlatched version the target is required to handle 4 outstanding pipelined transactions.

Current AD Activity

Next AD Activity	PCI	A.G.P. Command	Read Data	Write Data
PCI or A.G.P. Command	**FRAME#** of current transaction sampled asserted.	**REQ#** sampled asserted after being deasserted.	2nd to last data phase of current transaction.	Immediately following **GNT#** for current write while <4 outstanding **GNT#**s.
Read Data	**FRAME#** of current transaction sampled asserted. (Depends on **RBF#**)	**PIPE#** of current transaction sampled asserted. (Depends on **RBF#**)	2nd to last data phase of current transaction to allow max. of 40 bytes of buffering in master. See section describing **RBF#** signal. (Depends on **RBF#**)	Immediately following **GNT#** for current write (Depends on **RBF#**) while <4 outstanding **GNT#**s.
Write Data	**FRAME#** of current transaction sampled asserted.	**PIPE#** of current transaction sampled asserted.	2nd to last data phase of current transaction.	Immediately following **GNT#** for current write while <4 outstanding **GNT#**s.

Table 3-13 Current/Next AD Activity

Figure 3-16 shows a sequence of back-to-back 8 byte read data transactions in 2x data transfer mode. The target samples **GNT#** asserted on clock edge #2 and responds by asserting **TRDY#** and driving read data (L6) on the following clock. The arbiter can assert the **GNT#** for the second read data transaction (H4) on clock edge #3 since that is the last data phase of the L6 read data transaction. **GNT#** is asserted on every clock edge so that an 8 byte read data transaction can occur on every clock edge.

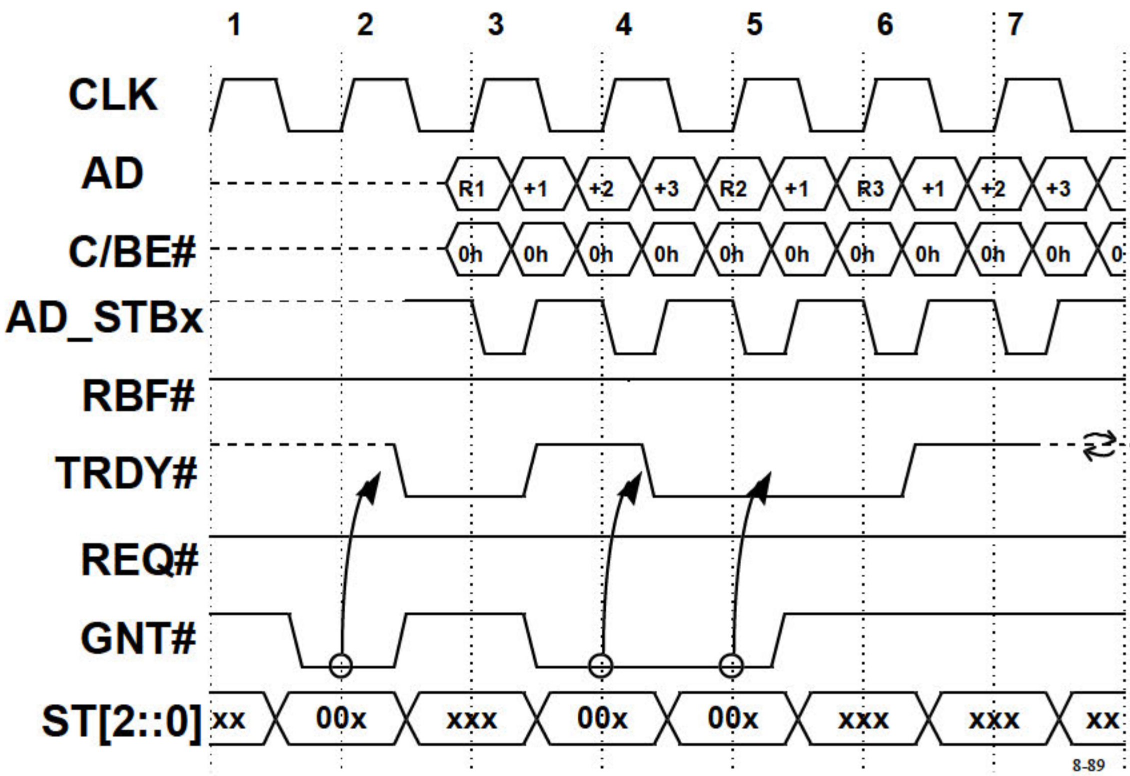


Figure 3-33 GNT# assertion for 16, 8, and then 16 byte Read Transfers

Figure 3-33 shows a sequence of 2x read data transactions. **GNT#** for the second read transaction (R2) is asserted on the clock edge #4 which is the last data phase of the R1 read transaction. **GNT#** for the third read transaction (R3) is asserted on clock edge #5 which is the last data phase of the R2 read transaction.

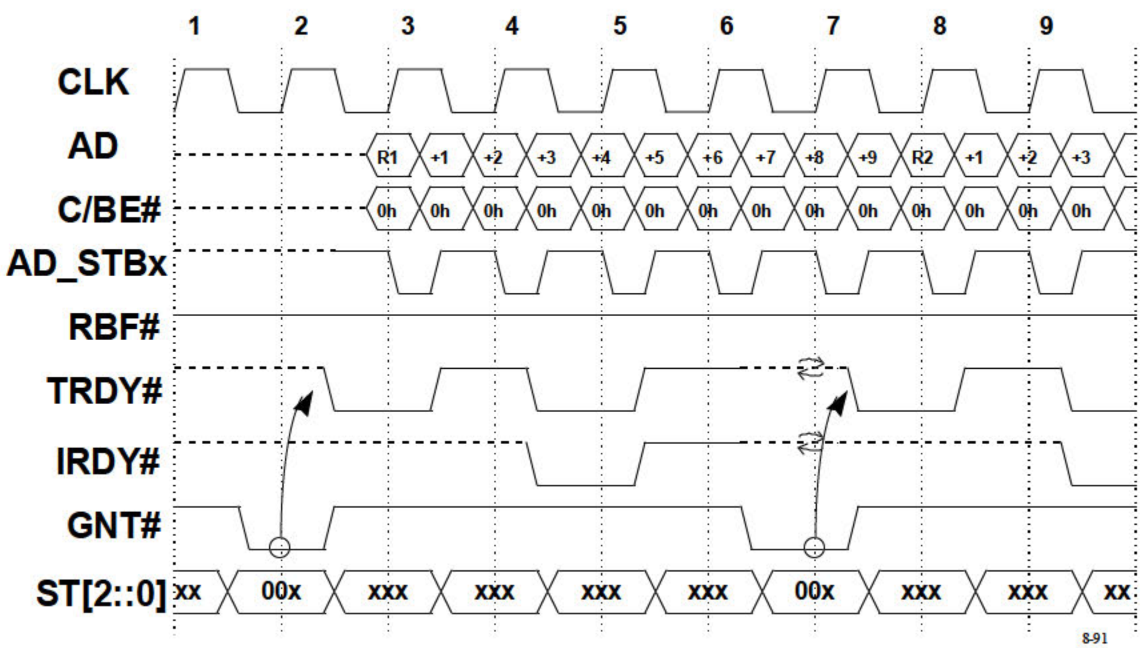


Figure 3-34 GNT# Assertion for next Read Data after long Data Transfer

Figure 3-34 shows a 40 byte read transaction followed by another read transaction in 2x data transfer mode. **GNT#** for the second read data transaction (R2) is asserted on clock edge #7 which is the last data phase of the R1 read transaction.

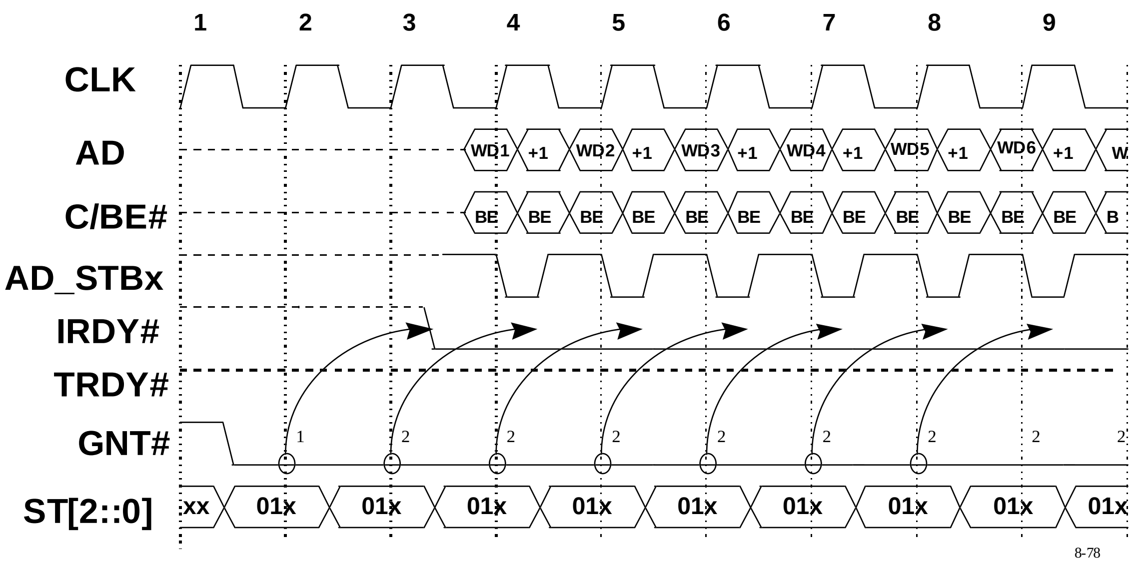


Figure 3-35 GNT# assertion for Back to Back Write Data Transfers

Figure 3-35 shows back-to-back 8 byte write data transactions in 2x data transfer mode. The following figures show that a maximum of 3 transaction are outstanding and will transfer data. The reason that it is only 3 and not 4 is that these diagrams assume that the arbiter is not using the latched version of **IRDY#**. When the latched version is used then all the number of grants outstanding are increased by one, since the arbiter delays the decrement. However, the arbiter can have 4 actually outstanding otherwise dead clocks can occur on the bus.

The master samples **GNT#** asserted on clock edge #2 and asserts **IRDY#** and drives write data W1 two clocks after sampling (clock edge #4). On clock edge #2 the arbiter increments its write **GNT#** counter to 1. Since the **GNT#** counter is less than three the arbiter asserts **GNT#** for write data W2 on clock edge #3 and the arbiter increments the write **GNT#** counter to 2. Since the **GNT#** counter is still less than three the arbiter asserts **GNT#** for write data W3 on clock edge #4. Even though **GNT#** is asserted on clock edge #4 the write **GNT#** counter does not increment since **IRDY#** for W1 is sampled asserted on clock edge #4. The arbiter continues asserting **GNT#** on every clock edge sustaining the back-to-back 8 byte transfers since the write **GNT#** counter is always less than three. In fact it is this waveform that established the need to allow up to three outstanding write **GNT#**s.

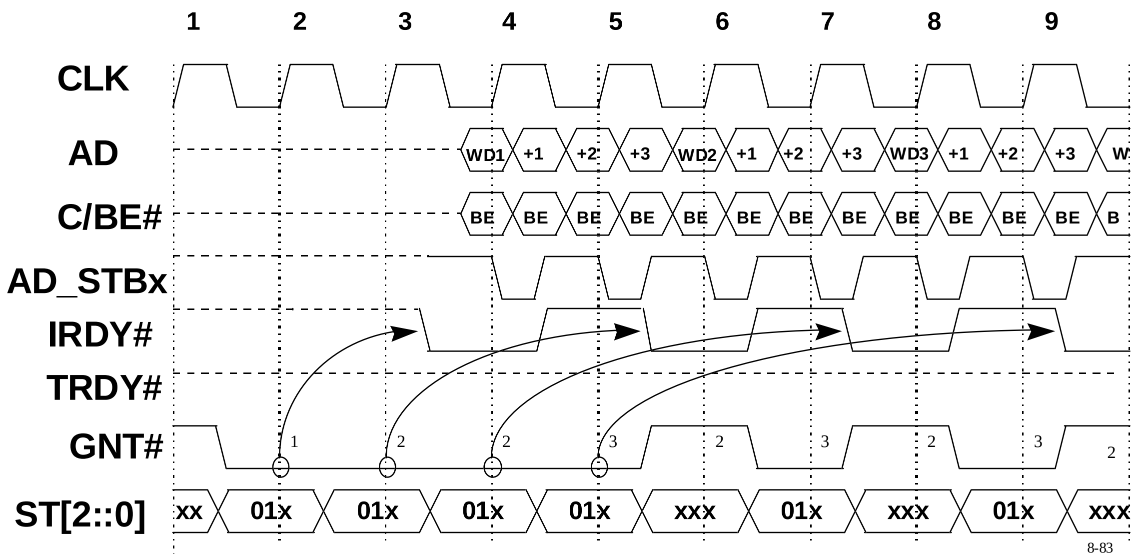


Figure 3-36 Back to Back GNT# with Delay on Initial Transfer

Figure 3-36 shows a sequence of 16 byte write data transaction in 2x data transfer mode. The master asserts **IRDY#** and drives write data W1 two clocks after sampling **GNT#** asserted on clock edge #2. On clock edge #2 the arbiter increments its write **GNT#** counter to 1. Since the **GNT#** counter is less than three the arbiter asserts **GNT#** for write data W2 on clock edge #3 and the arbiter increments the write **GNT#** counter to 2. Since the **GNT#** counter is still less than three the arbiter asserts **GNT#** for write data W3 on clock edge #4. Even though **GNT#** is asserted on clock edge #4 the write **GNT#** counter does not increment since **IRDY#** for W1 is sampled asserted on clock edge #4. Since the write **GNT#** counter is still less than three the arbiter asserts **GNT#** for write data W4 on clock edge #5. Since there is no **IRDY#** asserted on clock edge #5 the write **GNT#** counter increments to three and the arbiter is prohibited from asserting **GNT#** for W5 on clock edge #6. **IRDY#** for W2 is asserted on clock edge #6 decrementing the write **GNT#** counter to two. This allows the arbiter to assert **GNT#** for W5 on clock edge #7. This again increments the write **GNT#** counter to three and prohibits **GNT#** assertion for W6 on clock edge #8. Note that on clock edge #5 four **GNT#** have been pipelined to the master and the first transaction is still underway. This is the worst case scenario that the master's **GNT#** pipeline logic needs to account for.

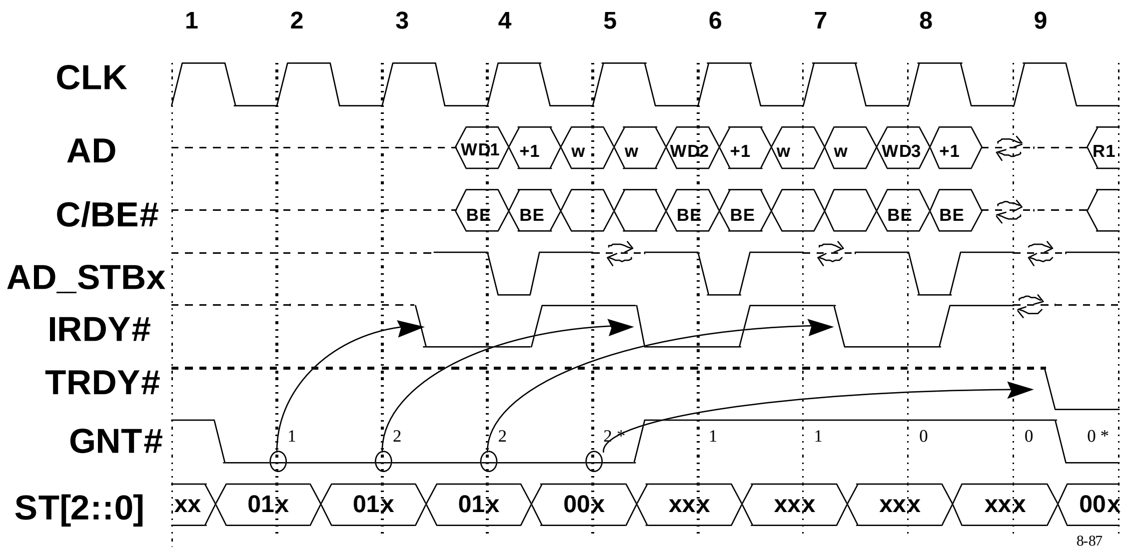


Figure 3-37 Pipelined GNT#s - Read and Writes (part 1)

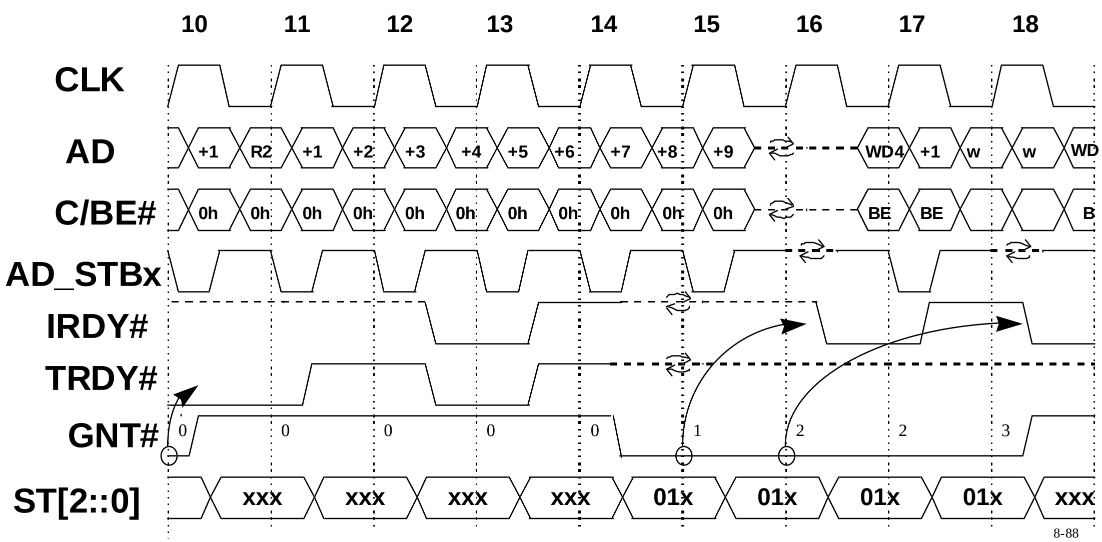


Figure 3-38 Pipelined GNT#s - Read and Writes (part 2)

Figure 3-37 shows the first half of a long sequence of write data transactions mixed with read data transactions. While Figure 3-38 shows the conclusion of the transaction. These need to be viewed as a single figure for the following discussion. The first three **GNT#**s are for write data transactions. The master inserts a wait state between the write data transactions. The **GNT#** asserted on clock edge #5 is for read data transaction R1. Note that the **GNT#** for R1 on clock edge #5 did not cause the write **GNT#** counter to increment from two to three. The write **GNT#** counter only increments for **GNT#**s associated with write data transactions. The arbiter de-asserts **GNT#** on clock edge #6 and waits to assert **GNT#** for read data R2 on clock edge #10 which is the last data phase of read data transaction R1. Note that by this time the write **GNT#** counter decremented to zero by sampling **IRDY#** asserted on clock edges #6 and #8. Note also that the write **GNT#** counter does not increment on clock edge #10 since the **GNT#** is for a read data transaction. The target is responsible for inserting the idle clock for bus turnaround between transactions W3 and R1. Read data transaction R2 is a 40 byte transaction so the next **GNT#** assertion is delayed by the arbiter until clock edge #15 which is the last data phase of R2. The **GNT#** on clock edges #15 is for write data transaction W4. This causes the write **GNT#** counter to increment. The master is responsible for inserting the idle clock for bus turnaround between transactions R2 and W4. The arbiter asserts **GNT#** for W5, W6 and W7 on clock edges #16,17 and 18# respectively. The arbiter is prohibited from asserting **GNT#** on clock edge #19 for another transaction since the write **GNT#** counter is at three.

3.6.6 **GNT#** Interaction with **RBF#**

The A.G.P. compliant arbiter will not assert **GNT#** for a low priority read data transaction if the **RBF#** signal is asserted. In the case where **RBF#** is asserted on the same clock edge as **GNT#** is asserted, the master is required to accept that transaction. The arbiter must deassert **GNT#** immediately upon sampling **RBF#** asserted so that no further low priority read data transactions are signaled. **RBF#** only prohibits **GNT#** from being asserted for low priority read data transactions. **GNT#** assertion for high priority read data, write data, and access requests can still be generated even though **RBF#** is asserted.

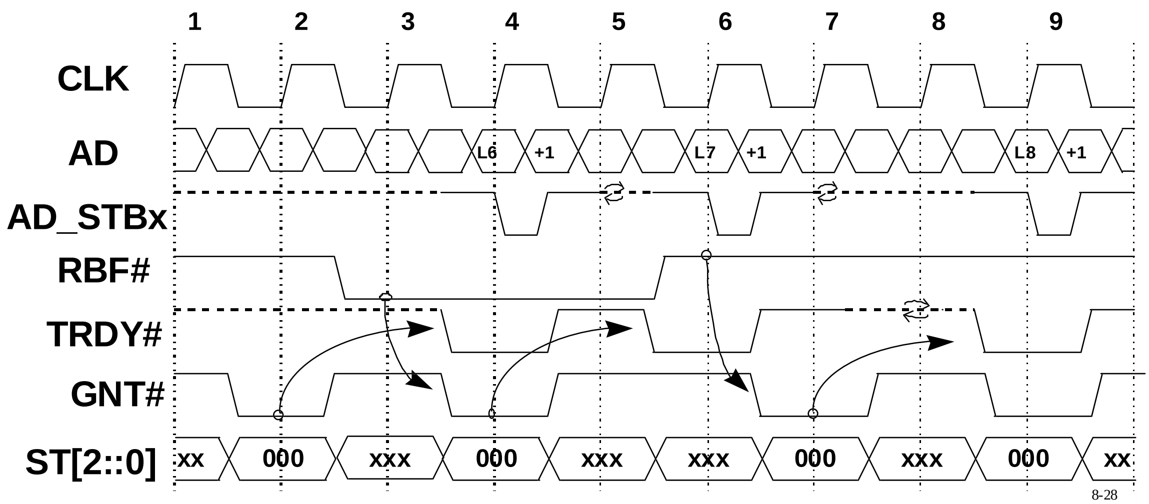


Figure 3-39 LP GNT# Pipelining Stopped While RBF# is Asserted

Figure 3-39 shows the master asserting the **RBF#** signal indicating that it can't accept further low priority read data. The master samples **GNT#** asserted on clock edge #2 with **ST[2::0]** indicating a low priority read data transaction. The master asserts **RBF#** on clock edge #3 because it doesn't have sufficient buffer space to take the next two low priority read transactions. The arbiter has already asserted **GNT#** on clock edge #3 which is the last data phase of L6. The master must accept the **GNT#** on clock edge #3 for read data transaction L7. The arbiter samples **RBF#** asserted on clock edge #3 and deasserts **GNT#** until it samples **RBF#** deasserted on clock edge # 5. Note that if the arbiter didn't deassert **GNT#** immediately upon sampling **RBF#** asserted on clock edge #3, then **GNT#** would be asserted on clock edge #4. This would increase the minimum amount of low priority read data buffering required in the master.

3.7 A.G.P. Sequencer State Machine Equations

This section will provide a concise statement of protocol rules to allow a designer to verify the correctness of his A.G.P. compliant bus sequencer. TBD.

3.7.1 Error Reporting

The error reporting philosophy of PCI applies to A.G.P. as well. This philosophy requires any device involved in the transfer or storage of permanent or retained system state (e.g., disk) to detect and report parity errors, and to compute parity. Devices involved with transient data only (e.g., graphics) are not so required. Since the sole purpose of the A.G.P. is the connection of video display devices, no provision has been made in the current spec for detecting or reporting of any bus errors. The current thinking is that if this becomes an issue in the future, bus parity detection and reporting will be added in essentially the same way as it is currently done on the PCI bus.

Admittedly this approach begs the question of address errors on writes to system memory (a problem that did not occur on PCI when graphics devices were targets only), as well as the possibility of data corruption on ancillary channels such as VBI. Nonetheless, this position seems to be congruent with current graphics accelerator practice, and therefore remains the current plan.

4. Electrical Specification

4.1 Overview

4.1.1 Introduction

As with the protocol enhancements, the A.G.P. electrical specification extends PCI to provide a much higher performance graphics interface. Most of the electrical interface requirements are based on the PCI spec, and this document will reference that spec wherever possible.

The A.G.P. physical interface is optimized for a point to point topology using a 3.3V signaling environment. The baseline performance level utilizes a 66 MHz clock, to provide a peak bandwidth of 266MB/sec.

A.G.P. includes an option for higher performance levels, providing peak bandwidths of 533MB/sec[17]. This optional mode uses a double-clocked data technique to transfer twice the data per each A.G.P. clock. This A.G.P. mode, referred to as *2X transfer mode (*also *2X mode* or *A.G.P.-133[18])*, requires additional interface timing strobes and different signal timings from the 1X mode. Components supporting the 2X transfer mode must also support the 1X mode. 2X mode requirements are a superset of the 1X mode timings.

The next section establishes a conceptual framework by which to understand the two modes.

4.1.2 1X Transfer Mode Operation

The 1X mode, 66 MHz A.G.P. interface can be designed using common I/O buffer technology. Since A.G.P. is a point to point interface, adjustments have been made in the system timing budgets that relax component input requirements relative to PCI.

Conceptually, the 1X mode A.G.P. operation is similar to PCI. All timings are referenced to a single clock, the A.G.P. clock.

4.1.3 2X Transfer Mode Operation

The A.G.P. protocol provides Qword access granularity. Qword transfers normally take two clock cycles in the 1X mode. Likewise, sideband address commands normally take two clocks per each 16-bit command. The 2X transfer mode provides a mechanism for doubling the data transfer rate of the **AD**, **C/BE#** and **SBA** signals[19]. With 2X transfer, Qword transfers only require one clock cycle, and sideband commands only require one clock per 16-bit command.

---

[17] 533MB/sec results from the actual minimum clock period of 15ns.

[18] "133" connotes a 133MHz peak transfer rate.

[19] The clock mode of these signals is controlled as a unified group; independent control of the clocking mode for sub-groups is not provided.

The 2X transfer mode is implemented as a timing layer *below* the 1X protocol's flow control mechanisms. This timing layer, referred to as the *inner loop*, specifies timing relationships for the reliable transfer of data from the output latches at the transmitting device to the input latches at the receiving device. The logical protocol mechanisms operate above this layer, via an *outer loop*, to control the actual transfer of data between the data queues. A model showing these various time domains is shown in Figure 4-1.

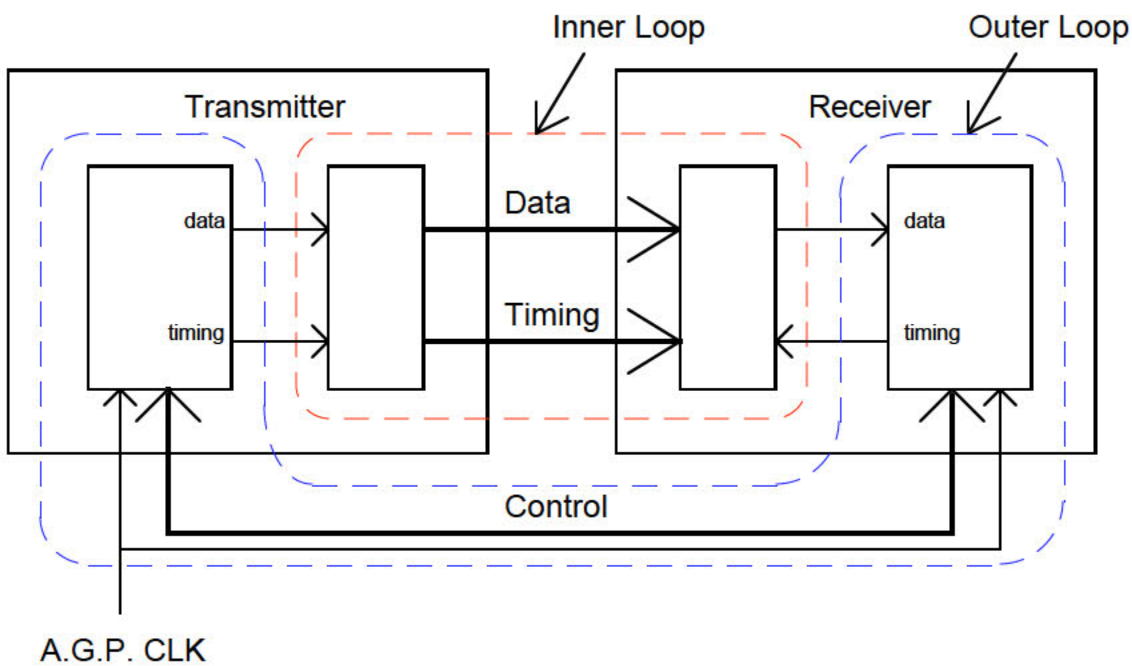


Figure 4-1 2X Mode Time Domains

The outer loop of both devices operates from a common A.G.P. clock, and outer loop controls are specified relative to this clock (as in the 1X operation mode). The inner loop timings use additional source synchronous timing signals to realize the data transfer. Note that the only clock defined in the system is the A.G.P. clock.

Source timed strobes, where the device sourcing the data also sources a timing signal for use by the receiver, are used since data transport delays are nulled out at the receiver. These source synchronous strobes are derived from the common A.G.P. clock at the transmitter, placed at the center of the output data valid window, and used by the receiver to directly capture data at the interface.

The inner loop and outer loop timing dependencies are defined by a set of relationships between the strobes and the A.G.P. clock. The relationship allows for a *deterministic* transfer of data between the inner and outer loops. These timing dependencies are specified in such a way as to allow implementation flexibility at the receiver. A tradeoff can be made between the latency through the inner loop and the implementation technology and/or design complexity. Refer to the A.G.P. Design Guide for more information.

The timing model presented above contains four different time domains, to be detailed in the following sections:

- Transmit/Receive Outer Loop
- Transmit to Receive Inner loop

- Transmit Outer to Inner Loop
- Receive Inner to Outer Loop

4.1.3.1 Transmit/Receive Outer Loop

The outer loop between the devices uses the 1X mode A.G.P. timings for bi-directional control information transfer between the transmitter and receiver.

4.1.3.2 Transmit to Receive Inner loop

Transfer of data[20] between the transmit and receive inner loop circuits is accomplished using a timing strobe sent from the transmitter to the receiver, with a set of simple timing relationships between the data and strobe. Both edges of the strobe are used to transfer data, with the first half of data corresponding to the falling edge of the strobe, and the second half corresponding to the rising edge.

The transmit strobe edges are positioned near the center of the minimum data valid window, to give the receiver a good input data sampling window for all the various system timing skew cases. A minimum data valid *before* strobe edge (tDvb), and a minimum data valid *after* strobe edge (tDva) are specified. The transmit strobe/data timings are shown in Figure 4-2.

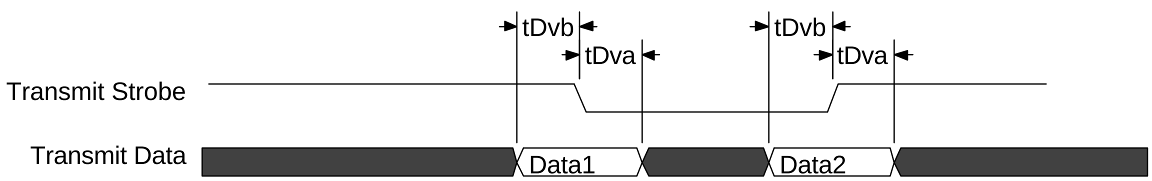


Figure 4-2 Transmit Strobe/Data Timings

The receive strobe input is directly used to capture data into the device. Again, *both* edges of the strobe are used to capture data, since new data is valid on each edge. A minimum setup (tDsu) and hold time (tDh) relative to the strobe edges is therefore required, as shown in Figure 4-3.

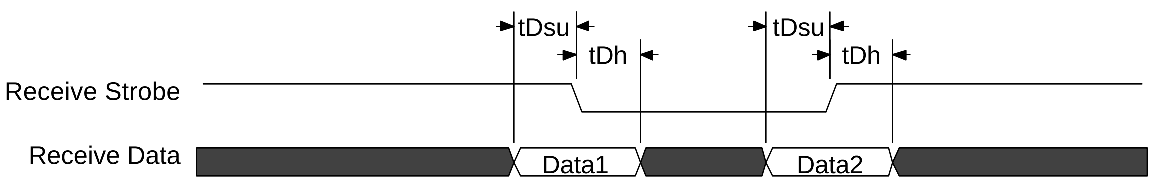


Figure 4-3 Receive Strobe/Data Timings

[20] *Data* refers to any of the 2X capable signal groups: AD[31:0], C/BE[3:0]# or SBA[7:0].

4.1.3.3 Transmit Outer to Inner Loop

The next timing relationship to understand is the relationship between the outer loop and inner loop at the transmitter. These timing specs are needed to create a deterministic data relationship between the inner loop transfer and the related outer loop flow control events (e.g. **IRDY#**). The relationship is specified by relating the output strobe to the A.G.P. clock, as shown in Figure 4-4. Note that two clock periods, T1 and T2, are shown since the strobe pulse is permitted to cross the A.G.P. clock boundary.

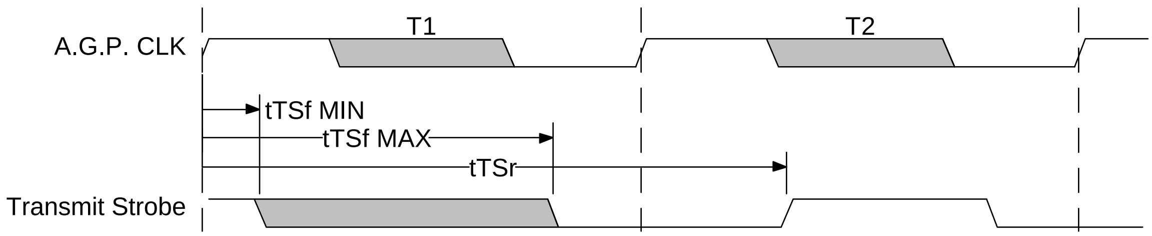


Figure 4-4 Transmit Strobe/Clock Timings

To guarantee a deterministic relationship between inner loop data transfer and the corresponding outer-loop flow control, the strobe's falling edge is required to occur within the T1 clock period, as seen at the receiver. This requirement dictates a min *and* max spec for the clock to strobe falling edge (tTSf). The clock to strobe rising edge only requires a max spec (tTSr). Actually, all the transmit strobe specs are driven by receiver requirements and system skews. The receiver requirements will be discussed in more detail in the next section.

Figure 4-5 shows the composite inner and outer loop timing relationships for the transmitter.

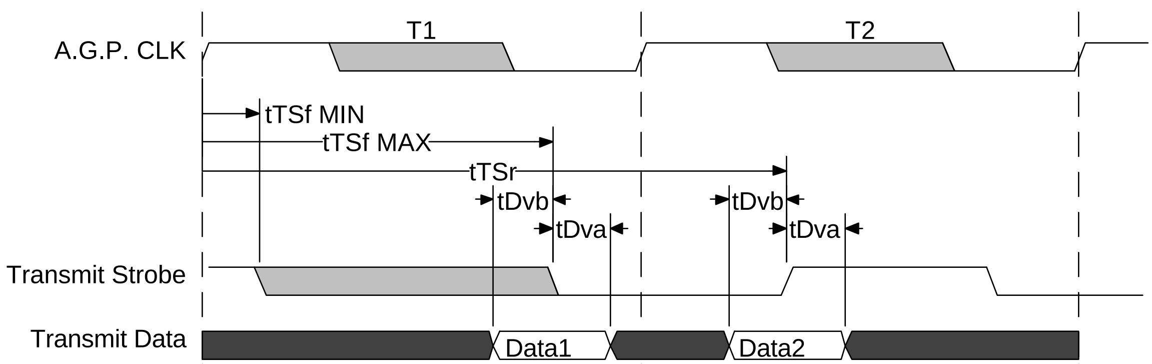


Figure 4-5 Composite Transmit Timings

4.1.3.4 Receive Inner to Outer Loop

The last, and most complex, set of timings to understand is the receiver inner to outer loop relationships. To better understand these timings, a model of the inner to outer loop transfer interface is needed. Refer to the receive transfer timing in Figure 4-6.

In the case of the **AD** bus interface, after the rising edge of the receive strobe, a Qword of valid data will be available. Data will be transferred from the inner loop to the outer loop based on a A.G.P. clock event. The requirement is to define a circuit to reliably affect this transfer for all system conditions.

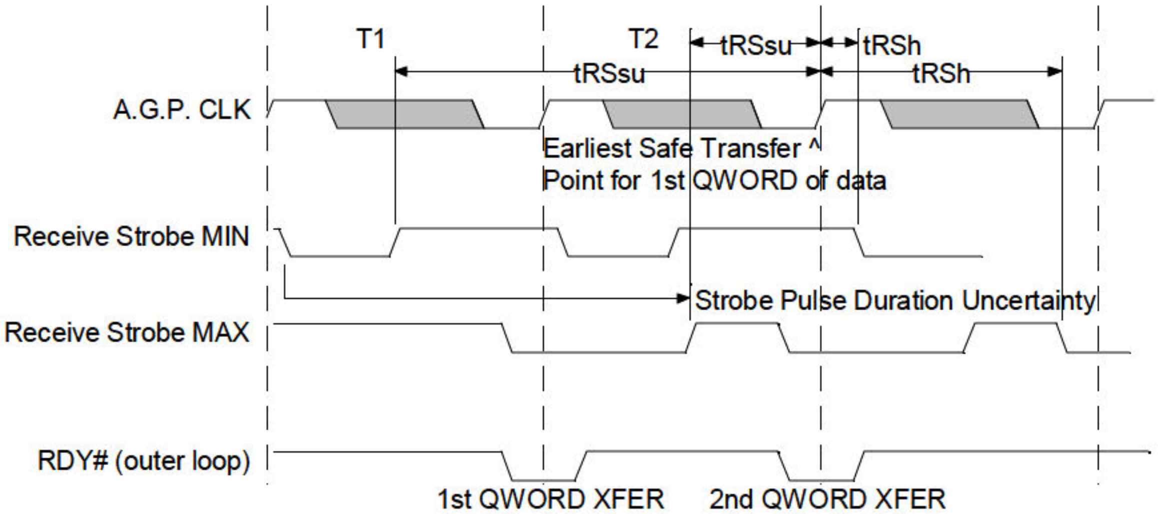


Figure 4-6 Receiver Inner to Outer Loop Transfer Timing

Figure 4-6 depicts the possible minimum and maximum strobe relationships at the receiver. Two consecutive data transfers occur, with the first one starting in T1. In the minimum strobe delay case, both strobe edges occur in T1, and in the maximum strobe delay case the second edge does not occur until T2. Thus, there is an uncertainty window for the strobe duration which can cross the clock boundary.

Due to this uncertainty window, the earliest *safe* transfer point from the inner loop to the outer loop occurs at the end of T2. Notice that in the min case a second set of strobes occurs in T2, since a second QWORD of data is being transferred. Therefore, to prevent data from being overwritten before the safe transfer point, at least *two* stages of edge-triggered latches must exist in the inner loop input circuitry.

The latched data will be transferred to the outer loop block based on the A.G.P. clock. The inner loop latched data must be guaranteed to remain stable at the point of transfer. The minimum setup spec on the receive strobe to clock (tRSsu) exists to ensure that data from the output of an inner loop latch has a defined setup time to the input of the outer loop's latch. Likewise, the minimum hold spec on the strobe (tRSh) exists to ensure that data from the output of an inner loop latch has a defined hold time relative to the input of the outer loop's latch.

Note that tRSsu and tRSh are associated with the strobe edge for a particular T1 cycle data transfer. Due to the minimum to maximum strobe variation, it is possible for tRSsu to extend *across* a clock edge into the T1 cycle. Also, the tRSh hold time is provided for data transfer from the *second* previous rising strobe event, not the immediately prior one.

The receive strobe spec values were chosen to allow most implementations to transfer the data at the *earliest* safe point, at the end of T2. Note that the *actual* transfer point is not specified, only the earliest viable point. An implementation may elect to increase the effective setup time by additional pipeline delay stages. Refer to the A.G.P. Design Guide for more details.

The composite receive timings are shown in Figure 4-7.

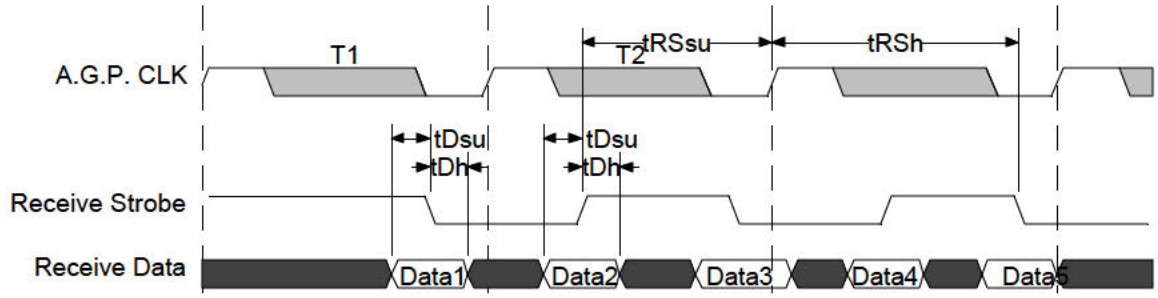


Figure 4-7 Composite Receive Timings

4.1.3.5 SB_STB Synchronization

The 2X inner loop receiver timing specifications ensure reliable transfer between the time domains. However, to affect such transfer implicit knowledge of the current clock cycle (T1 or T2) is needed, to determine which data to transfer to the outer loop. The **AD** bus outer loop protocol directly provides the required information, as each **xRDY#** event signals a new T1. However, the sideband address port, **SBA**, has no such inherent synchronization information.[21] To allow for the receiver circuitry to unambiguously synchronize with the correct clock event, an additional synchronization protocol is defined for the sideband port.

Note that this synchronization protocol has no relationship to the normal command protocol defined for the **SBA** signals. The A.G.P. compliant target protocol sequencer should be designed to be unaffected by this synchronization protocol. In other words, commands over the **SBA** port are undefined until synchronization has occurred.

Whenever the **SB_STB** signal has been idle, including after a reset, prior to sending any sideband commands the master must transmit a special synchronization cycle. This sync cycle is used by the receiver to determine proper phasing information (T1 or T2) at the interface.

The sync cycle is defined as **SBA[7::0]** being driven to FEh for a single 1X clock cycle, with timings adhering to the 1X mode requirements. Following the sync cycle, the **SBA** port state is undefined relative to the 1X clock, and is only valid at the first valid 2X command strobe point. This first valid 2X command strobe point occurs *exactly* two clocks after the sync cycle event is sampled by the target (cycle T1 in Figure 4-8), at which point 2X timing operation has commenced. The first valid command[22] occurs on this first T1 cycle.

The 2X timing operation must continue, unless **SB_STB** is stopped. Prior to stopping the **SB_STB** signal, a minimum of *four* clock cycles of NOPs must be transmitted. (Note this is equivalent to four 16-bit NOP commands in 2X mode). If stopped, **SB_STB** must be driven high and either held high or tristated. Once stopped, **SB_STB** must remain stopped for a *minimum* of *eight* clock cycles prior to any new sync event. The **SB_STB** synchronization protocol is shown below.

---

[21] The strobe to clock AC timings specs do not permit a synchronization based on sampling the strobe by the 1X clock, they only allow for deterministic data transfer from strobe-based data latches to 1X-clock based latches.

[22] The first command sent may be any of the defined sideband address commands, including a NOP.

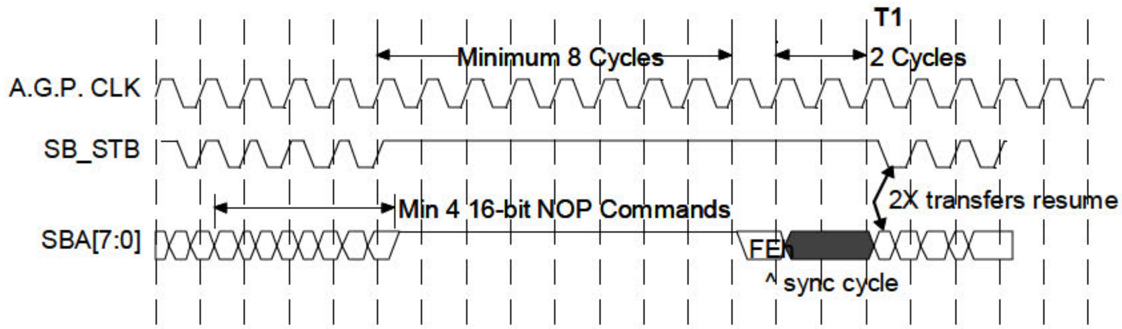


Figure 4-8 SB_STB Synchronization Protocol

4.2 Component Specification

This chapter defines the electrical characteristics of A.G.P. interface. Most of the specifications are focused on the A.G.P. modes, and will not repeat all requirements defined in the PCI Rev. 2.1 Specification.

I/O buffer design technology to meet these requirements is beyond the scope of this specification. Some general guidelines may be found in the A.G.P. Design Guide.

4.2.1 DC SPECIFICATIONS

The A.G.P. interface is optimized for a 3.3V operating environment and is based on PCI 3.3V signaling, with changes to allow higher data transfer rates. DC parameters differing from PCI 66 specifications are highlighted in bold in the table below.

4.2.1.1 A.G.P. 1X Mode DC Specification

Table 4-1: DC Specifications for A.G.P. 1X Signaling

Symbol	Parameter	Condition	Min	Max	Units	Notes
Vddq	**I/O Supply Voltage**		**3.0**	**3.6**	**V**	**1**
V_{ih}	Input High Voltage		0.5Vddq	Vddq+ 0.5	V	
V_{il}	Input Low Voltage		-0.5	0.3Vddq	V	
Vipu	Input Pull-up Voltage		.7Vddq			2
I_{ih}	Input High Leakage Current	V_{in} = 2.7		70	μA	3
I_{il}	Input Leakage Current	0 < V_{in} < Vddq		±10	μA	3
V_{oh}	Output High Voltage	I_{out} = -500 mA	.9Vddq		V	
V_{ol}	Output Low Voltage	I_{out} = 1500uA		.1Vddq	V	
C_{in}	**Input Pin Capacitance**			**8**	**pF**	**4**
C_{clk}	CLK Pin Capacitance		5	12	pF	

- NOTES:
- 1. Vddq only specifies the voltage at the A.G.P. interface; component supply voltages are independent of Vddq. Component I/O buffers must be powered only from Vddq, and isolated from any other power supplies on the component. Vddq for both A.G.P. compliant master and A.G.P. compliant target must be driven from the same power rail.
- 2. This specification should be guaranteed by design. It is the minimum voltage to which pull-up resistors are calculated to pull a floated network. Applications sensitive to static power utilization should ensure that the input buffer is conducting minimum current at this input voltage.
- 3. Input leakage currents include hi-Z output leakage for all bi-directional buffers with tri-state outputs.
- 4. Absolute maximum pin capacitance for an A.G.P. input is 8 pF (except for **CLK**) with an exception granted to motherboard-only devices, which could be up to 16 pF, in order to accommodate PGA packaging. This would mean, in general, that components for expansion boards would need to use alternatives to ceramic PGA packaging (i.e., PQFP, BGA, etc.).

4.2.1.2 A.G.P. 2X Mode DC Specification

The parameters below are the incremental requirements for the A.G.P. 2X mode interface. Note that the primary change is the addition of a voltage reference, which allows for a differential input buffer with common reference voltage. Implementation of a differential input buffer is not a requirement if alternate design approaches can be used to meet all other requirements.

Table 4-2: DC Specifications for A.G.P. 2X Signaling

Symbol	Parameter	Condition	Min	Max	Units	Notes
Vref	Input reference voltage		0.39Vddq	0.41Vddq	V	1,2
Iref	Vref pin input current	$0 < V_{in} < Vddq$		+10	uA	2
VIH	Input High Voltage		Vref+0.2		V	
VIL	Input Low Voltage			Vref-0.2	V	
Cin	Input Pin Capacitance			8	pf	
Δ Cin	Strobe to data Pin Capacitance delta		-1	2	pf	3

Notes:

1. A.G.P. allows differential input receivers to achieve the tighter timing tolerances needed for 133MT/s. Nominal value of Vref is 0.4Vddq which can be designed with 2% resistor to achieve the specified min and max values. The value of Vref is intended to specify near the center point of the VIL/VIH range. For example, at nominal Vddq (3.3V), Vref is 1.32V +/- 2.5%. A single input interface buffer can be designed to meet the VIL/VIH levels of both the A.G.P. and PCI specifications. As in other A.G.P. specifications, note that the Vddq references the I/O ring supply voltage, and not the component supply.

2. A differential input buffer is not a required implementation, as long as all other specifications are met. However component designs requiring a reference are required to adhere to the Vref and Iref specifications, to facilitate a common reference circuit for motherboard-only A.G.P. designs. (A common reference circuit is not applicable to add-in card designs, since Vref is not supplied via the connector.)

3. Delta Cin is required to restrict timing variations resulting from differences in input pin capacitance between the strobe and associated data pins. This delta only applies between signal groups and their associated strobes: **AD_STB1**=>**AD[31::16]** & **C/BE[3::2]#**; **AD_STB0**=>**AD[15::0]** & **C/BE[1::0]#**; **SB_STB**=>**SBA[7::0]**.

4.2.2 AC Timings

The A.G.P. timings are specified by two sets of parameters, one corresponding to the A.G.P. 1X operation, and the second for the optional A.G.P. 2X transfer mode operation. The A.G.P. 2X specs are in *addition* to the A.G.P. 1X specs. The A.G.P. 1X specs still apply to all outer-loop control signals during A.G.P. 2X operation.

4.2.2.1 A.G.P. 1X Timing Parameters

Table 4-3: A.G.P. 1X AC Timing Parameters

Symbol	Parameter	Min	Max	Units	Notes
Clock:					
t_{CYC}	CLK cycle time	15	30	ns	1
t_{HIGH}	CLK high time	6		ns	
t_{LOW}	CLK low time	6		ns	
-	CLK slew rate	1.5	4	V/ns	2
t_{LOCK}	PLL Lock Time		1000	us	3
Transmitter Output Signals:					
t_{VAL}	**CLK to control signal and Data valid delay**	**1.5**	**6**	**ns**	**4**
t_{ON}	**Float to Active Delay**	**1.5**	**6**	**ns**	
t_{OFF}	**Active to Float Delay**	**1**	**14**	**ns**	
	Output slew rate	1.5	4	V/ns	2
Receiver Input Signals:					
t_{SU}	**Control signals setup time to CLK**	**5**		**ns**	**4**
t_H	**Control signals hold time to CLK**	**0.5**		**ns**	**4**
Reset Signal:					
t_{RST}	Reset active time after power stable	1		ms	
$t_{RST-CLK}$	Reset active time after **CLK** stable	100		us	
$t_{RST-OFF}$	Reset active to output float delay		40	ns	
---	**RST#** Slew Rate	50	n/a	mV/ns	5

NOTES:

1. In general, A.G.P. compliant devices must work with any stable clock frequency from 33MHz to 66 MHz. Changes in the clock frequency are allowed for mobile applications, however only a change between the normal operating state and a clock stopped state are supported. The clock may only be stopped in a low state. Devices allowing for clock stop operation must be static designs, and maintain all software visible configuration information during the clock stop state. Also, during clock stop operation, devices designed for mobile applications must control signal state as specified in the forthcoming *PCI Power Management Specification* to minimize system power dissipation. The state of all new A.G.P. signals should be handled identically to existing PCI signals of the same output type (e.g. S/T/S are floated by component, pulled-up to Vddq by central resource).

2. Rise and fall times are specified in terms of the edge rate measured in V/ns. This slew rate must be met across the minimum peak-to-peak portion of the clock waveform as shown in figure 4-7 of the PCI spec.

3. Mobile A.G.P. systems can stop the A.G.P. clock during normal operation. PLLs on mobile components must relock within the specified time from a stable A.G.P. clock. Mobile A.G.P. compliant devices must allow for the clock to stop without any loss of data or configuration information. When the clock is restarted, a bus master can not issue a new transaction or bus request until the tLOCK time period has been met.

4. In A.G.P. 2X mode, tVAL, tDsu and tDh values for the **AD**, **C/BE#**, and **SBA** signals are *superseded* by the A.G.P.-2X parameters tDvb, tDva, tDsu, tDh.

5. The minimum **RST#** slew rate applies only to the rising (deassertion) edge of the reset signal, and ensures that system noise cannot render an otherwise monotonic signal to appear to bounce in the switching range.

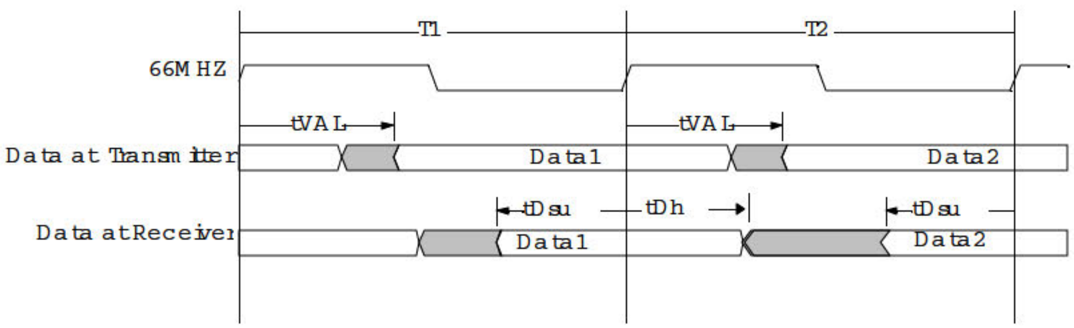


Figure 4-9 A.G.P. 1X Timing Diagram

4.2.2.2 A.G.P. 2X AC Timing Parameters

The parameters below apply only to the inner loop 2X transfer mode signals (**AD**, **C/BE#**, **SBA**) during 2X operation. The data specs below *replace* the corresponding data specs from the A.G.P. 1X table for these signals. The A.G.P. 1X parameters apply to all other signal operation.

Table 4-4 A.G.P. 2X AC Timing Parameters

Symbol	Parameter	Min	Max	Units	Notes
Transmitter Output Signals:					
t_{TSf}	**CLK** to transmit strobe falling	2	12	ns	
t_{TSr}	**CLK** to transmit strobe rising		20	ns	
t_{Dvb}	Data valid before strobe	1.7		ns	
t_{Dva}	Data valid after strobe	1.7		ns	
t_{ONd}	**Float to Active Delay**	**-1**	**9**	**ns**	
t_{OFFd}	**Active to Float Delay**	**1**	**12**	**ns**	
t_{ONS}	Strobe active to strobe falling edge setup	6	10	ns	
t_{OFFS}	Strobe rising edge to strobe float delay	6	10	ns	
Receiver Input Signals:					
t_{RSsu}	Receive strobe setup time to **CLK**	6		ns	
t_{RSh}	Receive strobe hold time hold time from **CLK**	1			
t_{Dsu}	Data to strobe setup time	1		ns	
t_{Dh}	Strobe to data hold time	1		ns	

NOTES:

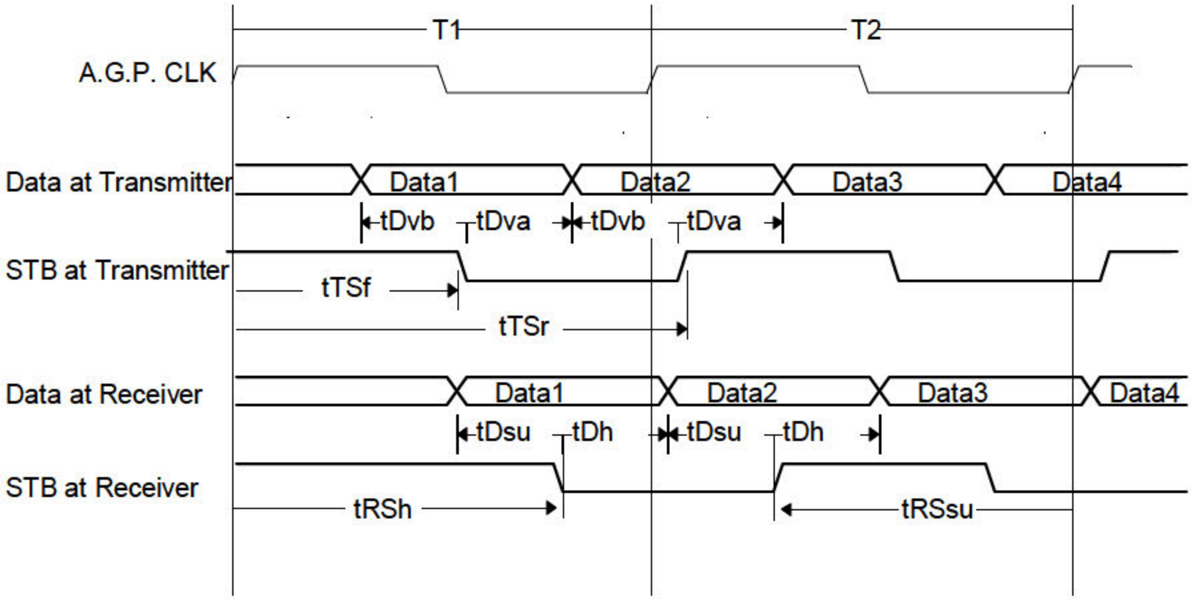


Figure 4-10 A.G.P. 133 Timing Diagram

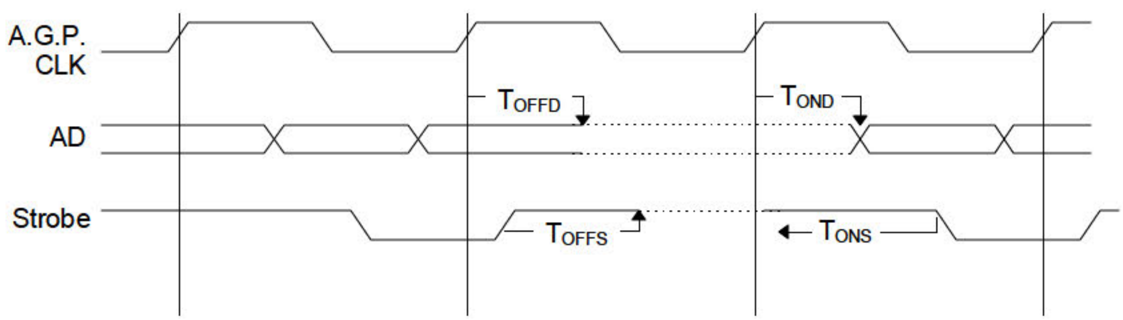


Figure 4-11 Strobe/Data Turnaround Timings

4.2.2.3 Measurement and Test Conditions

Unless otherwise specified, the reference point for all AC timings measurements is 0.4VddqThe output loading is 10pf

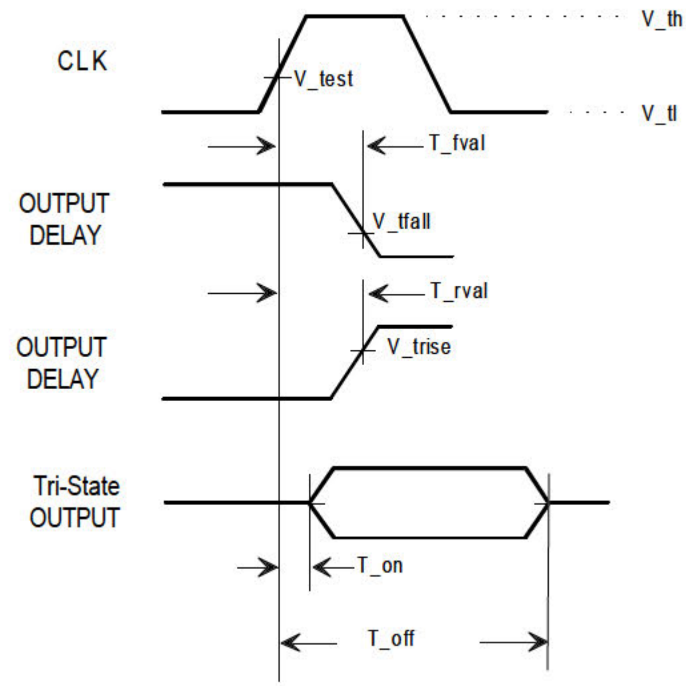


Figure 4-12 Output Timing Measurement Conditions

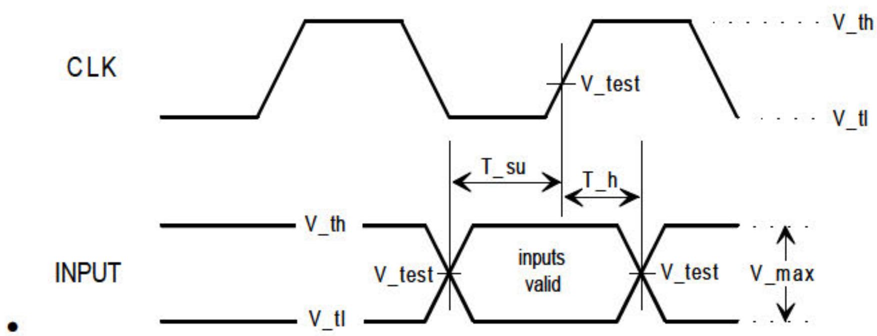


Figure 4-13 Input Timing Measurement Conditions

Table 4-5 Measurement and Test Condition Parameters

Symbol	3.3V Signaling	Units	Notes
V_{th}	0.6Vddq	V	1
V_{tl}	0.2Vddq	V	1
V_{test}	0.4Vddq	V	
V_{trise}	0.285Vddq	V	2
V_{tfall}	0.615Vddq	V	2
V_{max}	0.4Vddq	V	1
Input Signal Slew Rate	1.5-4.0	V/ns	3

NOTES:

- 1. The test for the 3.3V environment is done with 0.1*Vddq of overdrive. V_{max} specifies the maximum peak-to-peak waveform allowed for testing input timing.
- 2. V_{trise} and V_{tfall} are reference voltages for timing measurements only. Developers need to design buffers that launch enough energy into a 50Ω transmission line so that correct input levels are guaranteed after the first reflection.
- 3. Outputs will be characterized and measured at the package pin with the load shown in Figure 4-16. Input signal slew rate will be measured between 0.3Vddq and 0.6Vddq.

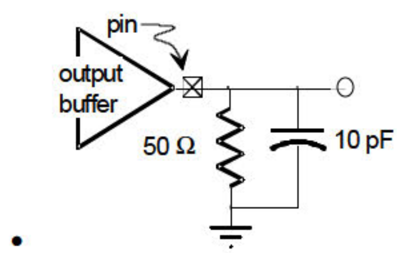


Figure 4-14 T_{val}(max) Rising Edge

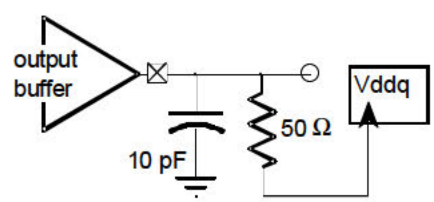


Figure 4-15 T_{val}(max) Falling Edge

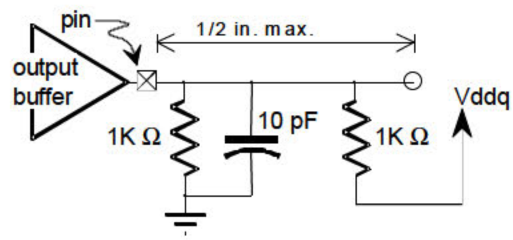


Figure 4-16 T_{val} (min) and Slew Rate

4.2.3 Signal Integrity Requirement

Table 4-6 Signal Integrity Requirements[1]

		A.G.P. 1X		A.G.P. 2X			
Symbol	**Parameter**	**Min**	**Max**	**Min**	**Max**	**Units**	**Notes**
-	Output Slew Rate	1.5	4	1.5	4	V/ns	2,3
Tset	Output settling time to +/- 10% of rail				7	nS	

NOTES:

- 1. Output buffer (OB) loading conditions under which measurements are made: 1) the OB driving a 6 inch transmission line with a characteristic impedance range from 65Ω ±15Ω, 2) one CMOS type input loading attached on the other side of an ideal transmission line
- 2. As measured at the receiver input
- 3. For mobile or other system designs without a metal enclosure, to minimize EMI problems it is recommended that the output slew rate not exceed 2.5V/ns.

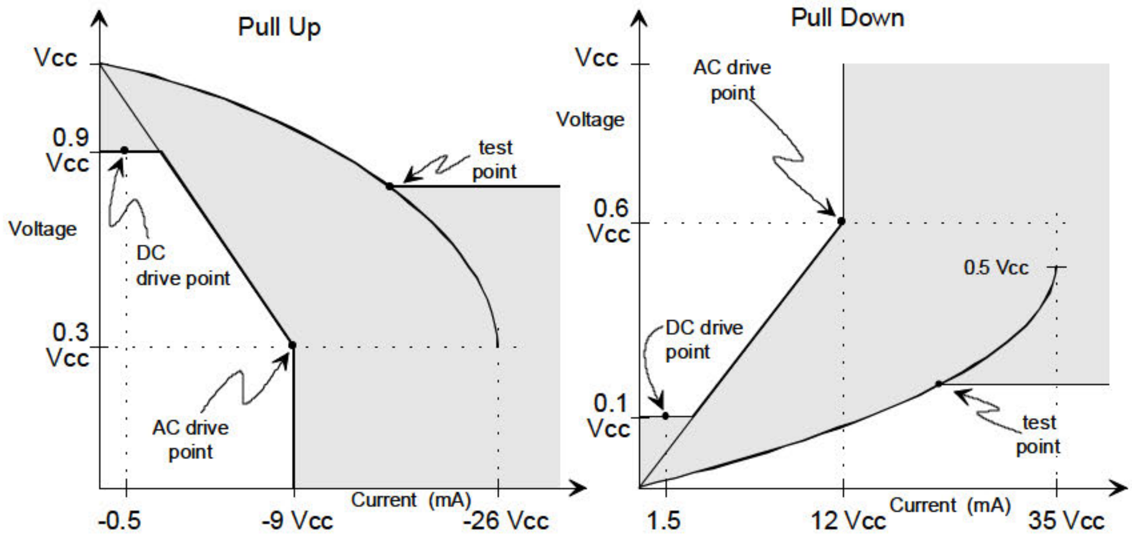
4.2.4 Driver Characteristics

To provide optimal performance in a point-to-point environment, A.G.P. requires a driver that is roughly *half* the strength of the PCI buffer. The output driver must be able to deliver an initial voltage swing of at least the VIL/VIH value to the receiver through the bus with a known characteristic impedance. Since no external transmission line termination mechanism is specified on the A.G.P. interface, under this environment, the signal at the device pins can transition beyond Vddq and VSS voltages by a considerable amount due to signal reflection on the line. The I/O buffer must be designed to maintain acceptable signal quality levels. Output slew rate and settling time specifications are included for this purpose.

In 2X mode, the data and strobe output buffers should be designed with rise and fall delay matching to within 1ns on all process, temperature, and voltage conditions. This is required for delay matching on the data and strobe paths for source synchronous data transfer mechanism.

The minimum and maximum drive characteristics of A.G.P. output buffers are defined by V/I curves. These curves should be interpreted as traditional "DC" transistor curves with the following exceptions: the "DC Drive Point" is the only position on the curves at which steady state operation is intended, while the higher current parts of the curves are only reached momentarily during bus switching transients. The "AC Drive Point" (the real definition of buffer strength) defines the minimum instantaneous current curve required to switch the bus with a single reflection. From a quiescent or steady state, the current associated with the AC drive point must be reached within the output delay time, T_{val}. Note however, that this delay time also includes necessary logic time. The partitioning of T_{val} between clock distribution, logic, and output buffer is not specified; but the faster the buffer (as long as it does not exceed the max rise/fall slew rate specification), the more time is allowed for logic delay inside the part. The "Test Point" defines the maximum allowable instantaneous current curve in order to limit switching noise and is selected roughly on a 50Ω load line.

Adherence to these curves should be evaluated at worst case conditions. The minimum pull up curve should be evaluated at minimum Vddq and high temperature. The minimum pull down curve should be evaluated at maximum Vddq and high temperature. The maximum curve test points should be evaluated at maximum Vddq and low temperature.



Equation C:

$$I_{oh} = (49.0/Vddq)*(V_{out}-Vddq)*(V_{out}+0.4Vddq)$$

$$\text{for } Vddq > V_{out} > 0.7\ Vddq$$

Equation D:

$$I_{ol} = (128/Vddq)*V_{out}*(Vddq-V_{out})$$

$$\text{for } 0v < V_{out} < 0.18\ Vddq$$

Figure 4-17 V/I Curves for 3.3V Signaling

Inputs are required to be clamped to BOTH ground and Vddq (3.3V) rails. When dual power rails are used, parasitic diode paths could exist from one supply to another. These diode paths can become significantly forward biased (conducting) if one of the power rails goes out of spec momentarily. Diode clamps to a power rail, as well as output pull-up devices, must be able to withstand short circuit current until drivers can be tri-stated.

4.2.5 Receiver Characteristics

A differential input receiver is recommended for the 1X and 2X operation. The voltage reference is specified at 0.4Vddq. This reference voltage is compatible with the PCI 66 MHz VIL/VIH specification. A differential buffer is not a strict requirement, as long as an implementation can meet all other AC/DC input specs.

To reduce the current consumption of the Vref supply, the differential input buffer must be designed with low input leakage current such that the combined load on Vref of all inputs is less than 10ua. The differential input buffer must be designed to have sufficient gain to convert an input differential voltage (~200mv) to a full internal CMOS voltage swing, without introducing additional skews.

4.2.6 Maximum AC Ratings and Device Protection

- All A.G.P. input, bi-directional, and tri-state output buffers should be capable of withstanding continuous exposure to the waveform shown in Figure 4-18. It is recommended that these waveforms be used as qualification criteria against which the long term reliability of each device is evaluated. This level of robustness should be guaranteed by design; it is not intended that this waveform should be used as a production test.
- These waveforms are applied with the equivalent of a zero impedance voltage source, driving through a series resistor directly into each A.G.P. input or tri-stated output pin. The open-circuit voltage of the voltage source is shown in Figure 4-18, which is based on the expected worst case overshoot and undershoot expected in actual A.G.P. busses. The resistor values are calculated to produce the worst case current into an effective (internal) clamp diode. Note that:
- The voltage waveform is supplied at the resistor shown in the evaluation setup, NOT the package pin.
- Any internal clamping in the device being tested will greatly reduce the voltage levels seen at the package pin.

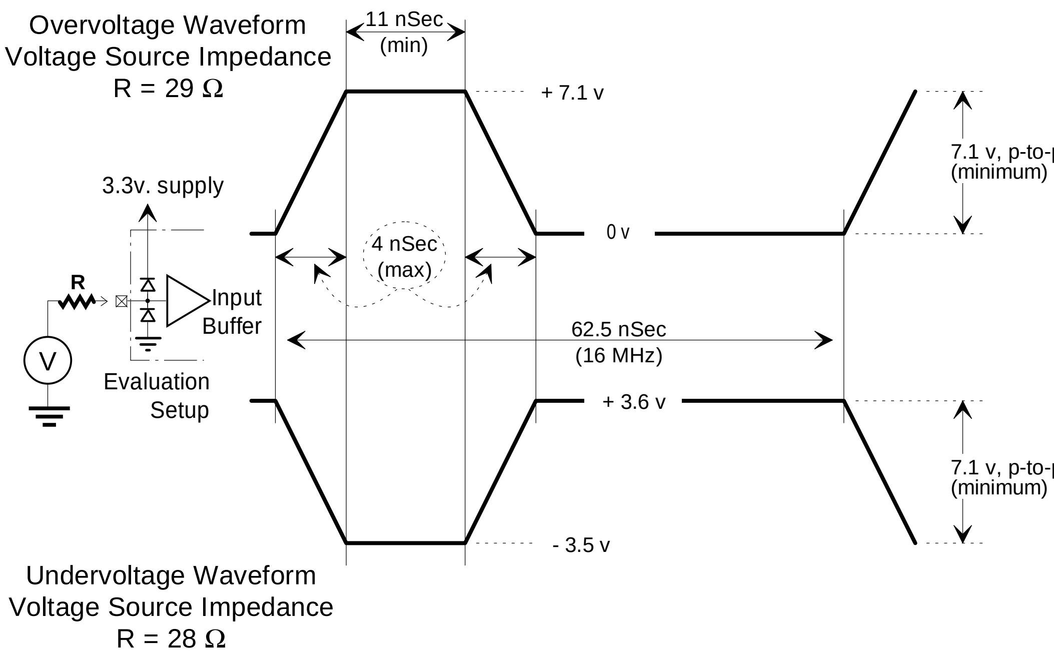


Figure 4-18 Maximum AC Waveforms for 3.3V Signaling

4.2.7 Component Pinout Recommendations

All A.G.P. signals on the graphics chip should be located to facilitate meeting the add-in card requirements as specified in section 4.3.. The component pinout should be ordered to match the connector pinout and component side of the add-in card as defined in Chapter 5 of this document. This alignment will minimize signal crossing, minimize overall trace lengths and aid in matching the trace lengths within the groups.

The strobe signals must be grouped with their associated data group:

- **AD_STB0** with **AD[15::0]** and **C/BE[1::0]#**
- **AD_STB1** with **AD[31::16]** and **C/BE[3::2]#**
- **SB_STB** with **SBA[7::0]**

Motherboard component pinouts should also be defined based on the above recommendations.

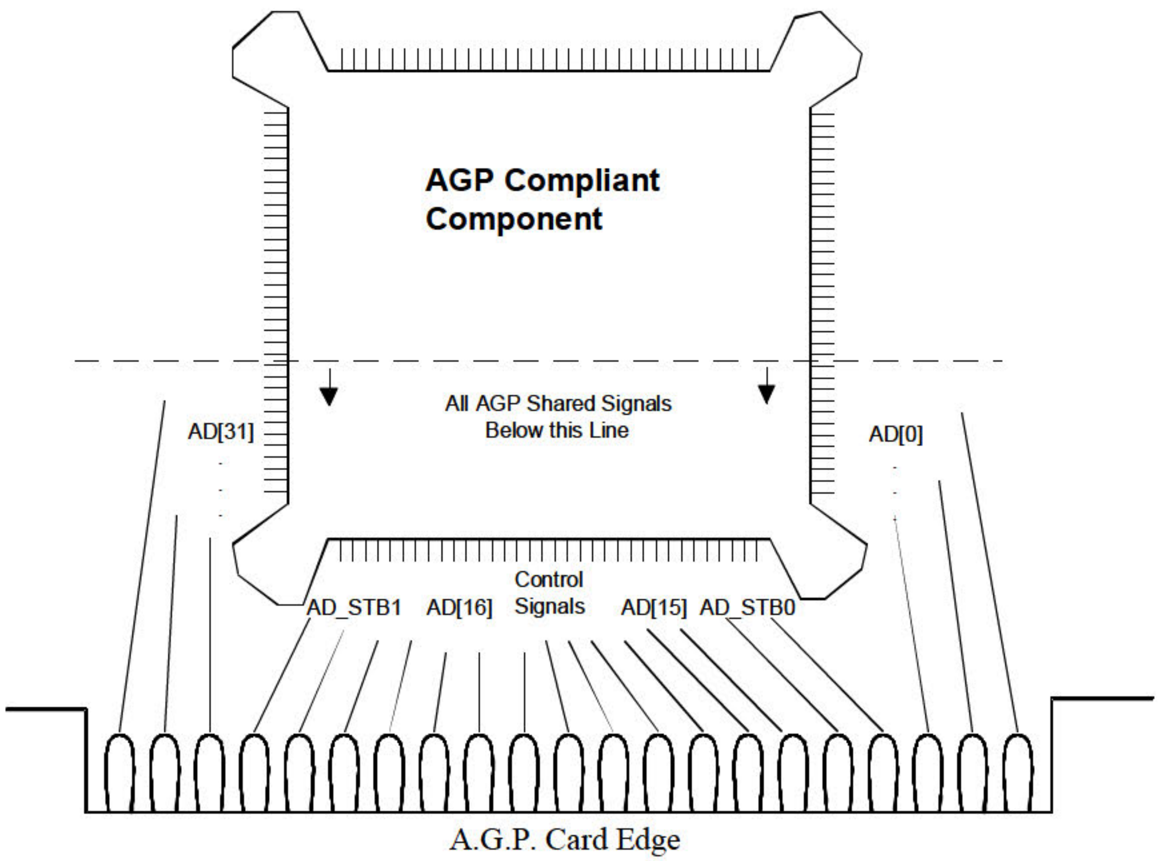


Figure 4-19 Recommended Component Pinout

4.3 Motherboard Specification

4.3.1 System Timing Budget

The table below summarizes the system timing parameters for 1X mode signals.

Table 4-7: System Timing Summary

Timing Element	Parameter	Max	Units	Notes
Tcyc	Cycle Time	15	ns	
Tval	Valid Delay	6	ns	
Tprop	Prop Delay	3	ns	
Tsu	Input Setup	5	ns	
Tskew-total	Total Skew	1	ns	1
Tskew-mb	Motherboard Skew	.9	ns	
Tskew-add-in	Add-in Card Skew	.1	ns	

NOTES:

1. Tskew is the sum of all skews (motherboard and add-in).

The table below summarizes all system interconnect delays. Note that the individual motherboard and add-in card components are repeated later in their respective sections.

Table 4-8: Interconnect Delay Summary

Symbol	Parameter	Max[1]	Units	Notes
t_{PROP}	Signal propagation	**3**	ns	2
$t_{PROP-MB}$	Signal propagation, motherboard	**2.15**	ns	
$t_{PROP-CONN}$	Signal propagation, connector	**.15**	ns	
$t_{PROP--CARD}$	Signal propagation, add-in card	**.7**	ns	
$t_{TRMATCH}$	Total Trace mismatch between Data and Strobe	**.7**	ns	3,4
$t_{TRMATCH-MB}$	Trace mismatch, motherboard	**.5**	ns	4
$t_{TRMATCH-CARD}$	Trace mismatch, card	**.2**	ns	4

NOTES:

1. Signal propagation delays are measured as the difference between the driver driving a 10pf lumped load vs. the driver driving an 80ohm transmission line terminated by a 10pf lumped load.
2. Tprop is the sum of all other propagation delays
3. Ttrmatch is the sum of all trace mismatches.
4. Trace mismatch applies between signal groups and their associated strobes: **AD_STB1**=>**AD[31::16]** & **C/BE[3::2]#**; **AD_STB0**=>**AD[15::0]** & **C/BE[1::0]#**; **SB_STB**=>**SBA[7::0]**. The trace mismatch spec only applies between the strobe and data signals within a group, not between data signal within a group or between groups.

4.3.2 Clock Skew

The maximum total system clock skew is 1 nsec for both 1X and 2X clock modes. This 1 nsec includes skew and jitter which originates on the motherboard, add-in card and clock synthesizer. Clock skew must be evaluated not only at a single threshold voltage, but at all points on the clock edge that fall in the switching range defined in Table 4-9 and Figure 4-20 Clock Skew Diagram.[23] This is measured between the pins of the two A.G.P. complaint components (not at the connector).

[23] The system designer may need to address an additional source of clock skew. This clock skew occurs between two components that have clock input trip points at opposite ends of the V_{il} - V_{ih} range. In certain circumstances, this can add to the clock skew measurement as described here. In all cases, total clock skew must be limited to the specified number.

- The total skew & jitter is allocated so that 0.1 nsec originates from the add-in card routing, and 0.9 nsec originates from the motherboard routing and clk synthesizer (the motherboard designer shall determine how the 0.9 nsec is allocated between the board and the synthesizer). To correctly evaluate clock skew, the system designer must take into account clock distribution on the add-in board as specified in section 4.3.2

Table 4-9 Clock Skew Parameters

Symbol	A.G.P. 1X (3.3V) Signaling	A.G.P. 2X (3.3V) Signaling	Units
V_{test}	0.4Vddq	0.4Vddq	V
T_{skew}	1 (max)	1 (max)	ns

-

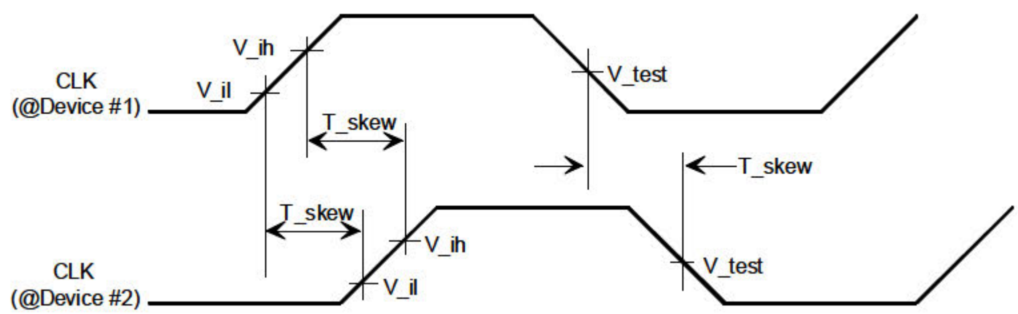


Figure 4-20 Clock Skew Diagram

4.3.3 Reset

- A.G.P. devices are reset using the PCI reset signal (**RST#**). The assertion and deassertion of **RST#** is asynchronous with respect to **CLK**. The rising (deassertion) edge of the **RST#** signal must be monotonic (bounce free) through the input switching range and must meet the minimum slew rate specified in Table 4-3. The specification does not preclude the implementation of a synchronous **RST#**, if desired. The timing parameters for reset are contained in Table 4-3, with the exception of the T_{fail} parameter. This parameter provides for system reaction to one or both of the power rails going out of spec. If this occurs, parasitic diode paths could short circuit active output buffers. Therefore, **RST#** is asserted upon power failure in order to float the output buffers.
- The value of T_{fail} is the minimum of:
- • 500 ns (maximum) from either power rail going out of specification (exceeding specified tolerances by more than 500 mV)
- • 100 ns (maximum) from the 5V rail falling below the 3.3V rail by more than 300 mV.
- The system must assert **RST#** during power up or in the event of a power failure. **RST#** should be asserted as soon as possible during the power up sequence, or as soon as the "power good" signal indicates a power failure. After **RST#** is asserted, A.G.P. complaint components must asynchronously disable (float) their outputs, but are not considered reset until both T_{rst} and $T_{rst\text{-}clk}$ parameters have been met. **RST#** therefore should not be deasserted until both T_{rst} and $T_{rst\text{-}clk}$ parameters have been met.

4.3.4 Interconnect Delay

Table 4-10: Motherboard Interconnect Delays

Symbol	Parameter	Max[1]	Units	Notes
$t_{PROP-MB}$	Signal propagation, motherboard	**2.15**	ns	
$t_{PROP-CONN}$	Signal propagation, connector	**.15**	ns	
$t_{TRMATCH-MB}$	Trace mismatch, motherboard	**.5**	ns	2

Notes:

1. Signal propagation delays are measured as the difference between the driver driving a 10pf lumped load vs. the driver driving an 80ohm transmission line terminated by a 10pf lumped load.
2. Trace mismatch applies between signal groups and their associated strobes: **AD_STB1**=>**AD[31::16]** & **C/BE[3::2]#**; **AD_STB0**=>**AD[15::0]** & **C/BE[1::0]#**; **SB_STB**=>**SBA[7::0]**. The trace mismatch spec only applies between the strobe and data signals within a group, not between data signal within a group or between groups.

4.3.5 Physical Requirements

The AC timings and electrical loading on the A.G.P. interface are optimized for one host component on the motherboard and one A.G.P. compliant agent either on the motherboard or through a connector. The interface is a point to point network, with a maximum electrical length of 3ns. The board routing should use layout design rules consistent with high speed digital design . The followings summarize the board layout restrictions on the A.G.P. interface.

4.3.5.1 Interface Signaling

All A.G.P. signals are +3.3V compatible signals. No +5V signals are specified in the A.G.P. bus environment. The master and target device must be capable of supporting the 3.3V signaling environment. The interrupt signals from the A.G.P. bus must interface to the PCI bus interrupt controller. This controller and the PCI compliant devices may be a +5V devices. It is the requirement of the motherboard designer to properly interface the A.G.P. interrupts to the PCI bus. This can be done is several ways. One way is to pullup the PCI interrupts to 3.3V only, allowing the A.G.P. interrupts to connect directly to the PCI interrupts. Alternatively, the A.G.P. interrupts can be buffered to the PCI bus, thus isolating the 5V environment from the A.G.P. bus.

4.3.5.2 Pullups

A.G.P. control signals require pullups to Vddq on the motherboard (or, optionally integrated on motherboard chipset) to ensure they contain stable values when no agent is actively driving the bus. These signals include

FRAME#, **TRDY#**, **IRDY#**, **DEVSEL#**, **STOP#**, **SERR#**, **PERR#**, **LOCK#**, **INTA#, INTB#**[24]**,** **PIPE#**, **AD_STB[1::0]** and **SB_STB.**

The pull-up value requirements are shown in the table below.

Table 4-11: Pull-up Resistor Values

Rmin	Rtypical	Rmax	Notes
4K Ω	8.2K Ω @ 10%	16K Ω	

NOTES:

4.3.5.3 Signal Routing and Layout

A.G.P. signals must be carefully routed on the motherboard to meet the timing and signal quality requirements of this interface specification. The following are some general guidelines that should be followed. Trace lengths included in this section are guidelines only. It is recommended that the board designer simulate the routes to verify that the specification is met.

The total flight time allowed for the A.G.P. bus is 3ns. The timing budget for the components of the flight path is identified in table 4-3. The motherboard prop delay budget of 2.15ns restricts the total trace length on the motherboard to less than approximately 10 inches.

The trace lengths for signals within a group must be matched to meet the total mismatch requirement given in Table 4-10, of 0.5ns. This means that the traces within a group on the motherboard must be matched to within approximately 2 inches, and are recommended to be matched as closely as possible to provide timing margin.

4.3.5.4 Impedances

The motherboard impedances should be controlled to minimize the impact of any mismatch between the motherboard and the add-in card. A impedance of 65Ω ± 15Ω is strongly recommended, otherwise signal integrity requirements may be violated.

4.3.5.5 Vref Generation

The motherboard must generate Vref locally for any motherboard component which requires it. Vref should be generated from the A.G.P. interface Vddq rail, not the component power supply. Vref should be properly decoupled to ground to manage switching currents. Such decoupling is platform dependent, and therefore not specified.

4.3.5.6 Crosstalk Consideration

For 66 and 133MT/s transfer, noise due to crosstalk must be carefully controlled to a minimum. Refer to the A.G.P. Design Guide for typical values.

---

[24] **INTA#** and **INTB#** are special cases. The motherboard should ensure these signals can not float to other motherboard inputs, while meeting the interface signaling requirements on section 4.3.5.1. If **INTA#** or **INTB#** are connected to inputs on the add-in card, the add-in card design must ensure those inputs can not float.

4.3.5.7 Line Termination

No external line termination mechanisms are specified on the A.G.P. interface, but may be used to meet signal integrity requirements as long as these elements do not inhibit agents from meeting their performance specifications. Circuit design techniques may be used to handle signal reflection and over/undershoot, such as clamping devices and slew rate controlled output buffers, to achieve acceptable signal integrity.

4.4 Add-in Card Specification

4.4.1 Clock Skew

The clock trace on the add-in card shall be routed to achieve an interconnect delay of 0.6 ± 0.1 ns. System designers will assume the delay of 0.6 nsec while designing the motherboard for minimum clock skew. The tolerance of ± 0.1 nsec is the clock skew contribution allocated for the add-in card, as specified in section 4.3.2..

4.4.2 Interconnect Delay

Table 4-12: Add-in Card Interconnect Delays

Symbol	Parameter	Max[1]	Units	Notes
$t_{PROP--CARD}$	Signal propagation, add-in card	**.7**	ns	
$t_{TRMATCH-CARD}$	Trace mismatch, card	**.2**	ns	2

NOTES:

1. Signal propagation delays are measured as the difference between the driver driving a 10pf lumped load vs. the driver driving an 80ohm transmission line terminated by a 10pf lumped load.

2. Trace mismatch applies between signal groups and their associated strobes: **AD_STB1**=>**AD[31::16]** & **C/BE[3::2]#**; **AD_STB0**=>**AD[15::0]** & **C/BE[1::0]#**; **SB_STB**=>**SBA[7::0]**. The trace mismatch spec only applies between the strobe and data signals within a group, not between data signal within a group or between groups.

4.4.3 Physical Requirements

4.4.3.1 Pin Assignment

Pins labeled *Vddq* are special power pins for defining and driving the A.G.P. signal rail on the board. On the board, the A.G.P. compliant component's I/O buffers must be powered from these special pins only — not from the other +5V or 3.3V power pins.[25]

4.4.3.2 Signal Routing and Layout

A.G.P. signals must be carefully routed on the graphics card to meet the timing and signal quality requirements of this interface specification. The following are some general guidelines that should be followed. Trace lengths included in this section are guidelines only. The card designer must simulate the routes to verify that the specification is met.

The total flight time allowed for the A.G.P. bus is 3ns. The timing budget for the components of the flight path is identified in Table 4-7. The add-in card prop delay budget of .7ns restricts the total trace length on the add-in card to be approximately 3 inches.

The trace lengths for signals within a group must be matched to meet the total mismatch requirement given in Table 4-12, of .2ns. This means that the traces within a group on the add-in card must be matched to within approximately 0.9 inches, and are recommended to be matched as closely as possible to provide timing margin. Refer to Figure 4-19 for add-in card component placement recommendations.

4.4.3.3 Impedances

The add-in card impedances should be controlled to minimize the impact of any mismatch between the motherboard and the add-in card. A impedance of $65\Omega \pm 15\Omega$ is strongly recommended, - otherwise signal integrity requirements may be violated.

4.4.3.4 Vref Generation

The add-in card must generate Vref locally for any add-in card component which requires it. Vref should be generated from the A.G.P. interface Vddq rail, not the component power supply.

4.4.3.5 Power supply delivery.

The power supply to the add-in card for core supply (VCC) and I/O supply voltage(Vddq) must be separated on the die, package and add-in card. Also there is a power level sequencing of core supply (VCC) and I/O supply voltage (Vddq) that needs to be considered which will be detailed in the next version of the specification.

---

[25] Any clamp diodes on A.G.P. signal pins must only connect to the Vddq rail and not to the 3.3V or 5V Vcc rails.

5. Mechanical Specification

5.1 Introduction

A.G.P. provides a high performance graphics interface. The A.G.P. connector was defined to meet this requirement and provide a robust implementation for an A.G.P. expansion card. This section defines an A.G.P. connector intended for high volume, high performance desktop systems. The connector must be low cost, reliable, electrically robust, and manufacturable in high volume from multiple sources. The A.G.P. connector effectively replaces one of the planar PCI connectors. The PCI connector that was previously utilized for graphics is now replaced by the A.G.P. connector which provides a higher performance.

The A.G.P. expansion card is based on the PCI expansion card design with the same maximum dimensions and configuration. It is easily implemented in existing chassis designs from multiple manufacturers. The A.G.P. expansion card requires a mounting bracket for card location and retention which is the same as the PCI ISA Retainer. The bracket is the interface between the card and the system that provides for cable escapement just as in PCI implementations. (See PCI Rev. 2.1, page 169 for ISA bracket and page 178 for ISA retainer.) The bracket shall be supplied with the card so that the card can be easily installed in the system.

5.2 Expansion Card Physical Dimensions and Tolerances

The A.G.P. card, like the PCI card, is designed to fit most existing chassis. The maximum component height on the primary side of the A.G.P. expansion card is not to exceed 0.570 inches (14.48 mm). The maximum component height on the backside of the card is not to exceed 0.105 inches (2.67 mm). Datum A on the illustrations is used to locate the A.G.P. card to the planar and to the frame interfaces; the back of the frame and the card guide. Datum A is carried through the locating key on the card edge and the locating key on the connector.

See Figure 5-1 and Figure 5-2 for A.G.P. expansion card physical dimensions.

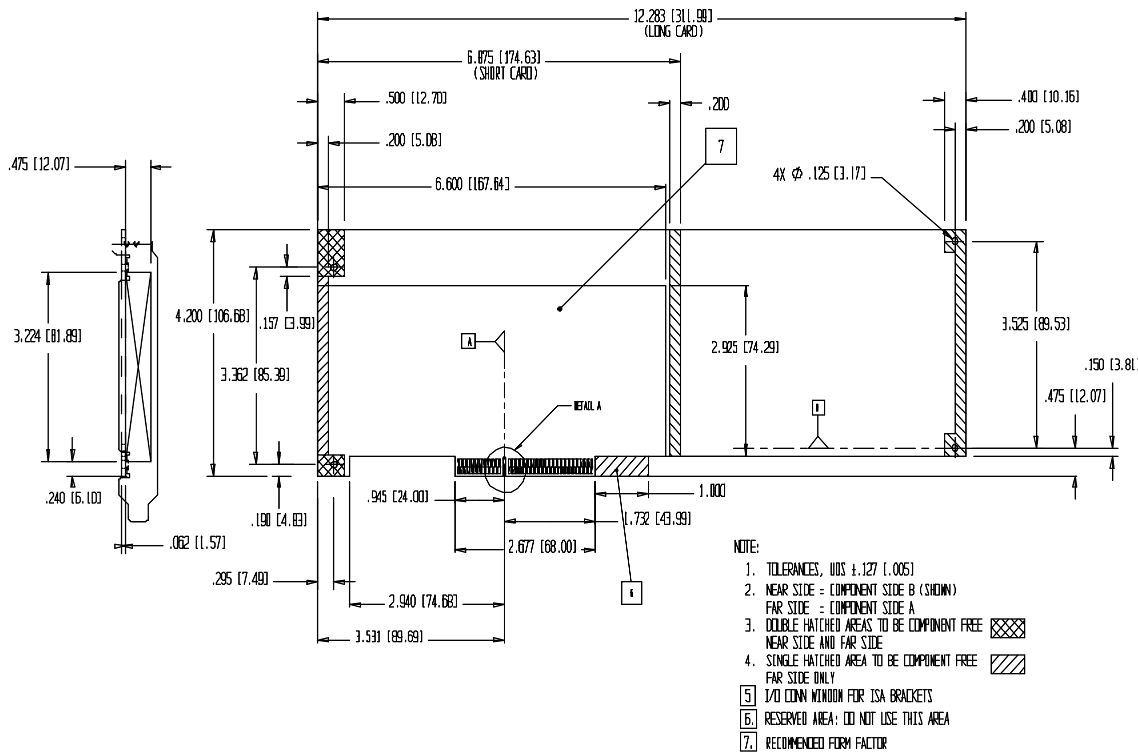


Figure 5-1 A.G.P. Card Edge Connector

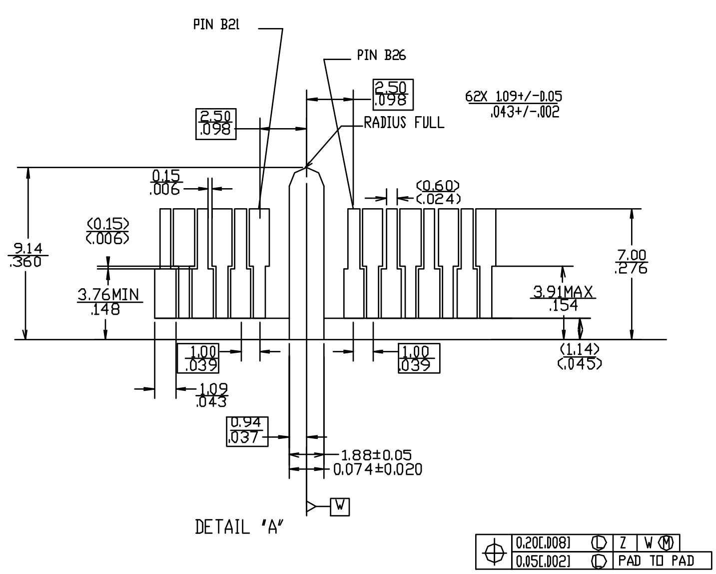


Figure 5-2 **Detail A, A.G.P. Card Edge Finger Layout**

5.3 Connector:

The connector shall hold the card at right angles to the system board. All dimensions are metric, inch dimensions are shown for reference only. The connector must accommodate a 1.57 mm (0.062 in) thick card (see Figure 5-3). Key width dimension of 1.78 mm (.070 inch) is measured prior to draft.

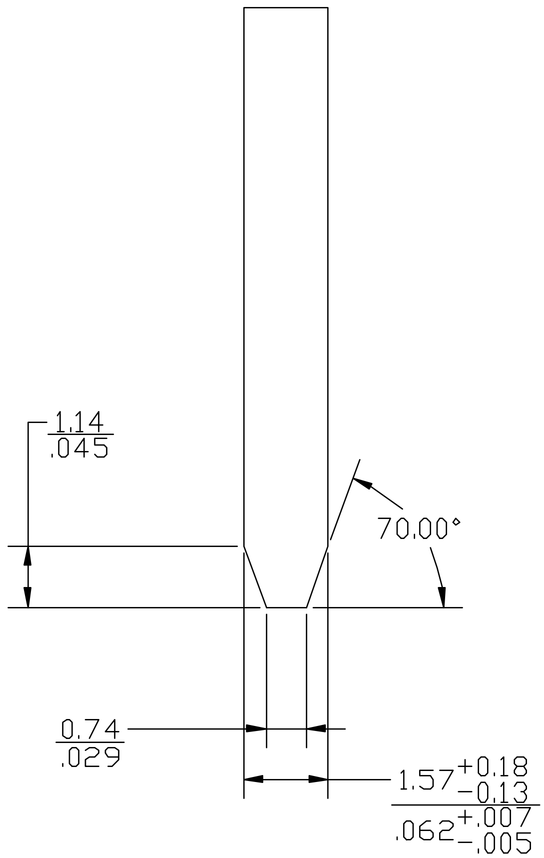


Figure 5-3 A.G.P. Card Edge Connector Bevel

5.4 Connector Physical Description

This section provides a physical description of the A.G.P. connector. The connector is intended for high volume, high performance desktop systems. In the connector drawings, the recommended board layout details are given as nominal dimensions. Layout detail tolerances should be consistent with the connector supplier's recommendations and good engineering practice. See

Figure 5-5 for connector dimensions and layout recommendations.

- The connector specification detail and connector supplier information is available from the A.G.P. homepage (http://www.teleport.com/~agfxport/).
- Caution: The connector is NOT hot unpluggable. Be sure system and Motherboard power is off.Unplugging an A.G.P. card with power and/or signals enabled at the connector may cause irreparable damage to the card and/or system boards. Disabling power and signals at the connector is a highly recommended standard practice for existing systems using ISA, EISA and PCI expansion cards.
- Caution: It is highly recommended that A.G.P. plug in cards are plugged and unplugged "straight" into the connector, without rocking. Using a "rocking" motion to seat and/or unseat the A.G.P. card may cause damage to the system board A.G.P. connector and/or damage pads on the add-in card. This is consistent with good practice and recommendations for the presently used PCI connector.

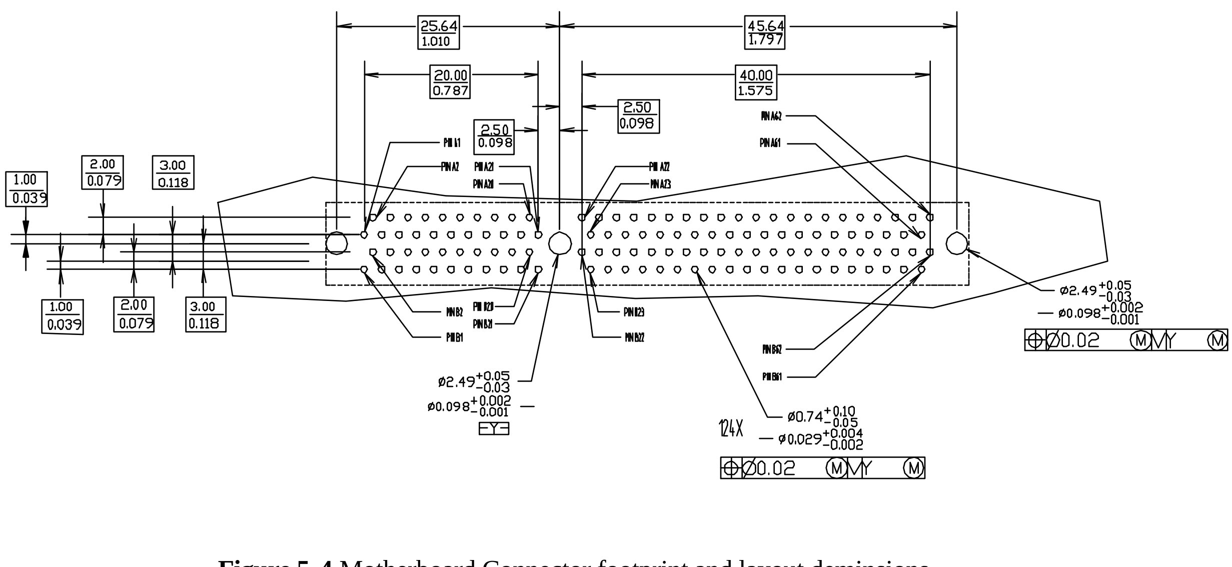


Figure 5-4 Motherboard Connector footprint and layout deminsions.

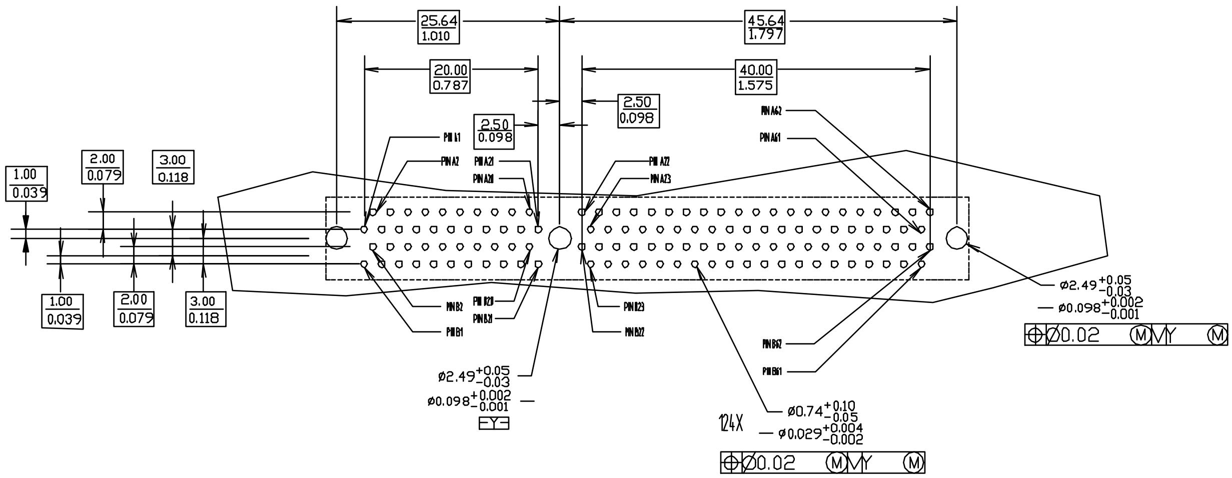


Figure 5-5 **Motherboard Connector Layout Recommendation**

5.5 Planar Implementation

5.5.1 ATX Planar Implementation

Planar implementations are supported by the A.G.P. expansion card design. For illustration purposes, the planar mounted expansion connector is detailed in Figure 5-6. This example shows an ATX form factor planar. The A.G.P. connector effectively replaces one of the planar PCI connectors. The PCI connector that was utilized for graphics is now replaced by the A.G.P. connector which provides a higher performance A.G.P. interface. The principles outlined can be applied to locate the A.G.P. connector in any motherboard.

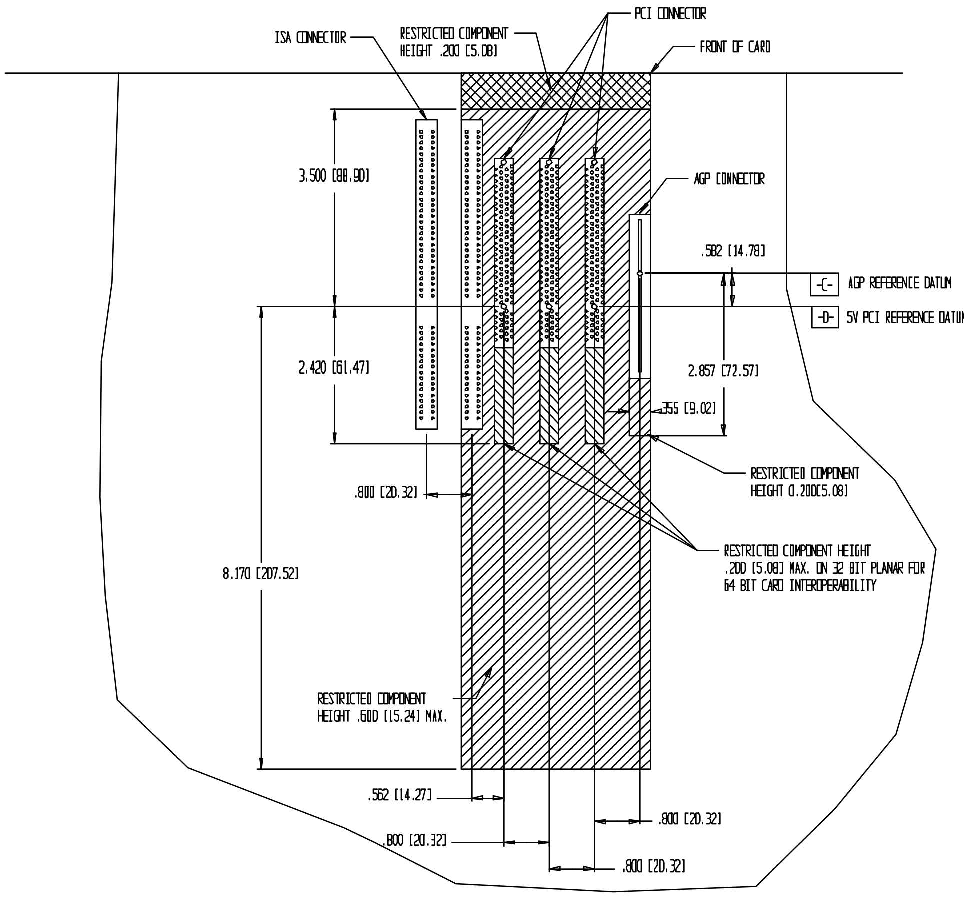


Figure 5-6 Typical ATX Implementation

5.5.2 Low Profile Planar Implementation

A.G.P.'s electrical and mechanical requirements prevent it from being placed on a low profile chassis's riser card, and requires that it be supported directly from the motherboard. To do this, the A.G.P. video solution can be implemented either integrated as video down on the motherboard, or as an A.G.P. expansion slot, but both cannot be supported simultaneously.

Integrating A.G.P. video down on the motherboard offers the fewest restrictions on the placement and routing of the circuit and connectors. This solution requires no special treatment in the A.G.P. interface specification.

5.5.3 Pin List

	A.G.P.1	
Pin#	B	A
1	Spare	**12V**
2	**5.0V**	Spare
3	**5.0V**	***Reserved ****
4	***USB+***	***USB-***
5	GND	GND
6	INTB#	INTA#
7	CLK	RST#
8	REQ#	GNT#
9	VCC3.3	VCC3.3
10	ST0	ST1
11	ST2	Reserved
12	**RBF#**	PIPE#
13	GND	GND
14	**Spare**	Spare
15	SBA0	SBA1
16	VCC3.3	VCC3.3
17	SBA2	SBA3
18	SB_STB	Reserved
19	GND	GND
20	SBA4	SBA5
21	SBA6	SBA7
22	KEY	KEY
23	KEY	KEY
24	KEY	KEY
25	KEY	KEY
26	AD31	AD30
27	AD29	AD28
28	VCC3.3	VCC3.3
29	AD27	AD26
30	AD25	AD24
31	GND	GND
32	**AD_STB1**	Reserved
33	AD23	C/BE3#
34	Vddq3.3	Vddq3.3
35	AD21	AD22
36	AD19	AD20
37	GND	GND
38	AD17	AD18
39	C/BE2#	AD16
40	Vddq3.3	Vddq3.3
41	**IRDY#**	**FRAME#**
42		
43	GND	GND
44		
45	VCC3.3	VCC3.3
46	**DEVSEL#**	**TRDY#**
47	**Vddq3.3**	STOP#
48	PERR#	**Spare**
49	GND	GND
50	SERR#	PAR
51	C/BE1#	AD15
52	Vddq3.3	Vddq3.3
53	AD14	AD13
54	AD12	AD11
55	GND	GND
56	AD10	AD9
57	AD8	C/BE0#
58	Vddq3.3	Vddq3.3
59	**AD_STB0**	Reserved
60	AD7	AD6
61	GND	GND
62	AD5	AD4
63	AD3	AD2
64	Vddq3.3	Vddq3.3
65	AD1	AD0
66	SMB0	SMB1

* - This reserved pin should be connected to GND

Note: **IDSEL** is not a pin on the A.G.P. connector. A.G.P. add-in card manufacturer's should connect the **AD16** signal to the **IDSEL** pin on an A.G.P. compliant Master.

6. System Configuration and A.G.P. Initialization

A.G.P. configuration and initialization operations are of three general types.

1. Power On Startup Test (POST) code allocates resources to all devices in the system. (BIOS)

2. The operating system activates A.G.P. features. (Not BIOS)

3. The final runtime memory management activity is carried out by Microsoft's DirectDraw.

The first two of these steps are described here. Refer to Microsoft documentation for details on the third.

6.1 POST-time initialization

Conventional bus enumeration software in the PowerOnSartupTest (POST) code identifies all system devices *(includes A.G.P. compliant devices),* creates a consistent system address map and allocates system resources to each device. An A.G.P. compliant device (master or target) must provide all required fields in the device's PCI configuration header, including Device ID, Vendor ID, Status, Command, Class code, Revision ID and Header type. (See Section 6.1 of the PCI 2.1 specification for more detail.) By supporting the PCI header, this allows conventional bus enumeration software to function correctly while being completely unaware of A.G.P. features.

6.1.1 A.G.P. Compliant Master Devices

A.G.P. compliant master devices have a certain amount of memory resources that must be placed somewhere in the system memory address map using a PCI base address register. These memory resources fall into two categories, Prefetchable and Non-prefetchable address regions. Prefetchable memory space is where the Linear Framebuffer is mapped to provide performance improvements. Non-prefetchable memory space is where control registers and FIFO-like communication interfaces are mapped. Each of these address regions should have their own base address register. Refer to page 196 of the PCI 2.1 specification for a description of PCI base address registers.

6.1.2 A.G.P. Compliant Target Devices

A.G.P. compliant target devices require a certain amount of address space for A.G.P. memory that must be placed somewhere in the system address map using a PCI base address register. Non-prefetchable control registers when supported by the target are provided by a second base address register.

To enable the use of existing enumeration code (unmodified) to handle A.G.P. compliant devices, Figure 6-1 is a logical view of how an A.G.P. compliant target appears. The area inside the dotted line represents the A.G.P. compliant target and core logic chipset. The corelogic (in this example) includes ports to the System memory, Processor, PCI and A.G.P.. The two main functions in the figure are the Host Bus Bridge and the PCI to PCI Bridge. The Host Bus Bridge is the interface that exists in all corelogic that spawn a PCI bus segment. The PCI to PCI Bridge function, facilitate the configuration of the second I/O port (A.G.P.) of the corelogic without requiring new enumeration code. With the corelogic presenting the interface (required to follow the PCI to PCI Bridge Architecture Specification (1.0)), this provides a way to determine what device resides on the A.G.P. port, what system resources it requires, and the mechanism to allocate those resources.

A benefit of this is that the host bridge obtains the mapping information (without special software) to route requests to the correct destination (i.e., PCI, A.G.P. or main memory).

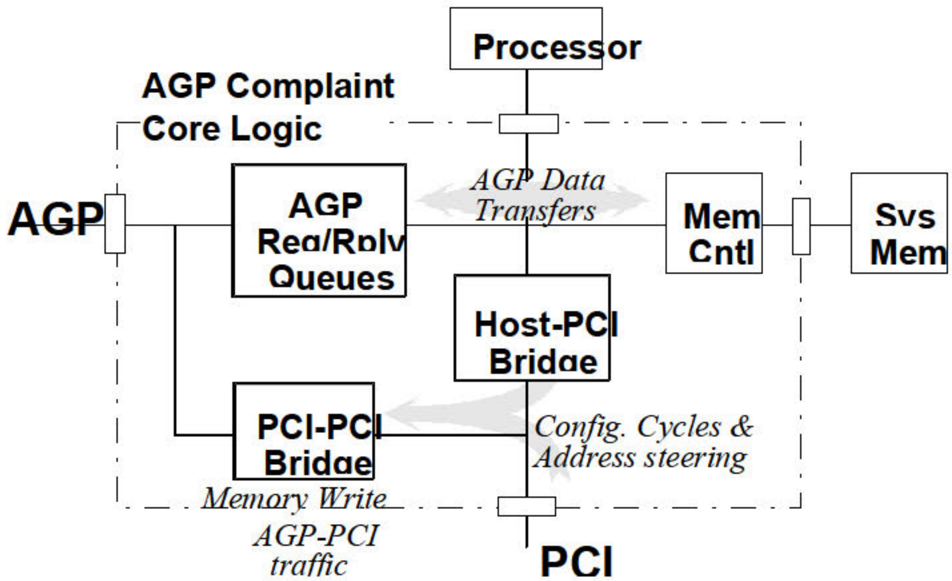


Figure 6-1 Configuration View of an A.G.P. Target

Note: the PCI to PCI Bridge function in the figure is NOT a complete and fully functional PCI to PCI Bridge. This implementation DOES not allow all access types to transverse between the A.G.P. and PCI segments. Only Memory write transactions (PCI commands Memory Write and Memory Write and Invalidate) that initiate on A.G.P. are forwarded to the Primary PCI bus segment and on Primary PCI bus are forwarded to A.G.P. segment. All other PCI commands (I/O (read and write), configuration (read and write) and Memory reads (Memory Read, Memory Read Line and Memory Read Multiple, Special Cycles, and Interrupt Acknowledge) are not allowed to cross the interface. When one of these commands is issued and has destination on the other interface the corelogic can treat this as a programming error. How it completes the access is chipset specific, one option would be to have the corelogic (acting as a PCI compliant target) to simply ignore the request and allow it to terminate with Master-abort.

The A.G.P. compliant target's base address registers should reside in the Host-to-PCI bridge because the Windows operating system can load a Host-to-PCI bridge driver, whereas there is no provision for loading a driver for a PCI-to-PCI bridge.

6.1.3 Boot-time VGA Display Device(s)

Most A.G.P. graphics accelerators will have a VGA display device. This means some systems may have more than one VGA device. Conventional BIOS codes select one VGA device by first searching the ISA bus, then PCI add-in card slots *(includes A.G.P. connector)*, then motherboard devices *(includes motherboard A.G.P. compliant devices)*.

> **Reminder:** Boot-time bus enumeration software is unaware of A.G.P. features.

6.1.4 Operating System Initialization

The operating system initializes A.G.P. features by performing the following operations:

1. Allocate memory for the A.G.P. remapping table

2. Initialize the A.G.P. compliant target's address remapping hardware

3. Set the A.G.P. compliant target and master data transfer parameters

4. Set host memory type for A.G.P. memory

5. Activate policy limiting the amount of A.G.P. memory

An A.G.P. chipset driver API will be used for the second item. Refer to the appropriate Microsoft device driver interface kit for details.

The third item requires access to configuration registers defined later in this interface specification. Setting bit 4 (Status Register) at offset 6 indicates the device implements New Capabilities mechanism as described by PCI[26]. The New Capabilities structure is implemented as a linked list of registers containing information for each function supported by the device. A.G.P. status and command registers are included in the linked list. The structure for the A.G.P. specific ID and structure is illustrated in Figure 6-2.

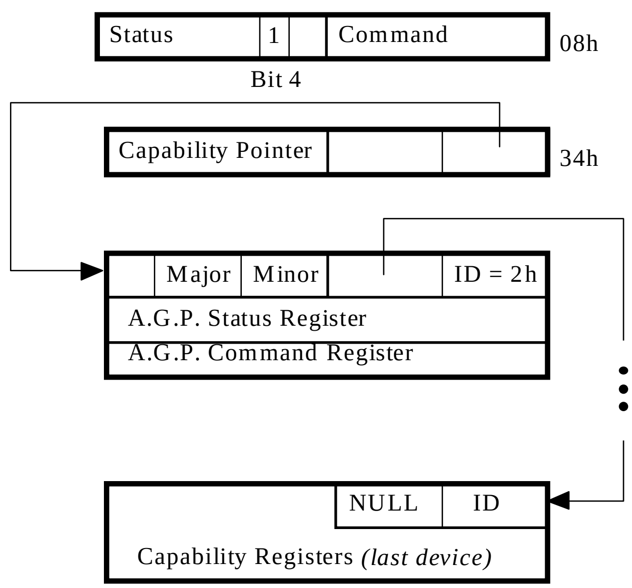


Figure 6-2 Location of A.G.P. Capabilities

Configuration registers are used by the OS to initialize A.G.P. features. These features must be supported by both A.G.P. compliant master and target devices in the following registers. The explanatory text describes the specific behavior of the target and master with respect to each function.

[26] An ECR defining the "New Capabilities" to the PCI 2.1 Specification is in process.

6.1.5 PCI Status Register

Bit	Field	Description
31:5		See PCI 2.1 specification
4	**CAP_LIST**	If the **CAP_LIST** bit is set, the device's configuration space implements a list of capabilities. This bit is *Read Only* register.
3:0		See PCI 2.1 specification

6.1.6 Capabilities Pointer - (offset 34h)

Bits	Field	Description
31:8	Reserved	Always returns 0 on read, write operations have no effect
7:0	**CAP_PTR**	This field contains a byte offset into the device's configuration space containing the first item in the capabilities list and is a *Read Only* register.

CAP_PTR gives the location of the first item in the list, which, in this example, is for the A.G.P. compliant device. Device capabilities may appear in any order in the list. The CAP_PTR register and the Capability Identifier register are *Read Only* with reserved fields returning zero when read.

6.1.7 Capability Identifier Register (Offset = CAP_PTR)

Bits	Field	Description
31:24	Reserved	Always returns 0 on read; Write operations have no effect.
23:20	**MAJOR**	Major revision number of A.G.P. interface specification this device conforms to.
19:16	**MINOR**	Minor revision number of A.G.P. interface specification this device conforms to.
15:8	**NEXT_PTR**	Pointer to next item in capabilities list. Must be *NULL* for final item in list.
7:0	**CAP_ID**	The value 02h in this field identifies the list item as pertaining to A.G.P. registers.

The first byte of each list entry is the capability ID. The PCI Special Interest Group assigned A.G.P. an ID of 02h. The **NEXT_PTR** field contains a pointer to the next item in the list. The **NEXT_PTR** field in final list item must contain a NULL pointer.

6.1.8 A.G.P. status register (offset CAP_PTR + 4)

Bits	Field	Description
31:24	**RQ**	The **RQ** field contains the maximum number of A.G.P. command requests this device can manage.
23:10	Reserved	Always returns 0 when read, write operations have no effect
9	**SBA**	If set, this device supports side band addressing.
2:8	Reserved	Always returns 0 when read, write operations have no effect
1:0	**RATE**	The **RATE** field indicates the data transfer rates supported by this device. A.G.P. compliant devices must report all that apply. *<Bit 0: 1X, Bit 1: 2X >* *Note: The* **RATE** *field applies to* ***AD*** *and* ***SBA*** *buses.*

The A.G.P. status register is a *Read Only* register. Writes have no affect, and reserved or unimplemented fields return zero when read.

6.1.9 A.G.P. command register - (offset CAP_PTR + 8)

Bits	Field	Description
31:24	**RQ_DEPTH**	<u>Master</u>: The **RQ_DEPTH** field must be programmed with the maximum number of pipelined operations the master is allowed to enqueue in the target. Value set in this field must be equal to or less than the value reported in the **RQ** field of target's status register. <u>Target</u>: The **RQ_DEPTH** field is reserved.
23:10	Reserved	Always returns 0 when read, write operations have no effect
9	**SBA_ENABLE**	When set, the side address mechanism is enabled in this device.
8	**AGP_ENABLE**	<u>Master</u>: Setting the **AGP_ENABLE** bit allows the master to initiate A.G.P. operations. When cleared, the master cannot initiate A.G.P. operations. <u>Target</u>: Setting the **AGP_ENABLE** bit allows the target to accept A.G.P. operations. When cleared, the target ignores incoming A.G.P. operations. *Notes: 1. The target must be enabled before the master .* *2. The* **AGP_ENABLE** *bit is cleared by* **AGP_RESET**.
7:3	Reserved	Always returns 0 when read, write operations have no effect
2:0	**DATA_RATE**	One *(and only one)* bit in the **DATA_RATE** field must be set to indicate the desired data transfer rate. *<Bit 0: 1X, Bit 1: 2X>*. The same bit must be set on both master and target. *Note: The* **DATA_RATE** *field applies to* **AD** *and* **SBA** *buses*

The A.G.P. command register is a read/write register, with reserved fields returning zero when read and writes having no affect. All bits in the A.G.P. command register are initialized to zero at reset.

7. Appendix A

[Preliminary – subject to change]

1. API	Application Programming Interface.
2. Bus 0	Compatibility PCI bus *(where ISA bridge resides).*
3. Bus *n*	PCI/A.G.P. bus. *n* = 1 when no PCI to PCI Bridges present on Bus 0.
4. Chipset	Motherboard chipset that provides connections to: Host Bus, Compatibility PCI bus, and A.G.P. interface.
5. DirectX	Microsoft API's for accessing graphics and audio hardware.
6. DirectDraw	Microsoft graphics API.
7. Display surface	Memory area containing graphics data object.
8. Fence	Means of synchronizing A.G.P. write operations with subsequent A.G.P. read operations.
9. Flush	Operation that makes an A.G.P. compliant target's accesses to system graphics memory visible to other parts of the system.
10. A.G.P. bus	Abbreviation for PCI/A.G.P. bus.
11. A.G.P. compliant master	An A.G.P. compliant master interface that is capable of generating A.G.P. pipelined read/write operations per this interface specification.
12. A.G.P. port	Connection point on a chipset where an A.G.P. compliant master may be attached.
13. A.G.P. operation	Memory read/write operation subject to A.G.P. ordering rules and protocol. A.G.P. operations that are initiated by **PIPE#** or **SBA** bus.
14. A.G.P. compliant target	An A.G.P. compliant target interface that is capable of interpreting A.G.P. addresses (e.g. the chipset) per this interface specification.
15. Local graphics memory	Memory local to the graphics controller.
16. Non-prefetchable memory	PCI registers that have side effects.
17. Prefetchable Memory	PCI memory and memory mapped registers that are free from side-effects.
18. Uncacheable Memory	Host caching protocol used on I/O operations, non-prefetchable regions or prefetchable regions not supported by hardware coherency.
19. Ordering rules	Rules specifying when the effect of a read or write operation can be observed by another operation.
20. Paging	Movement of data between disk and other memory levels of the virtual memory system.
21. Pipelining	A.G.P. read/write operations use a *split transaction* like paradigm

	where one or more addresses may be transferred during one bus operation and data is transferred during another.
22. POST or POST Code	Initialization software executed out of the startup ROM.
23. SPI	System Programming Interface.
24. Synchronization	A.G.P. ordering rules may allow certain memory operations to occur in to provide improved performance and may complete in a different order than initiated by the master. Synchronization operations provide additional control of the completion order.
25. System graphics memory	System memory accessible via the A.G.P. port by the graphics controller.
26. WriteBack	A type of Host cache coherency used for application memory.

8. Appendix B

This appendix contains other timing diagrams that are interesting but not used in the body of this interface specification. These figures are believed to be correct, however errors may exist and the rules and wording in the main interface specification have precedence.

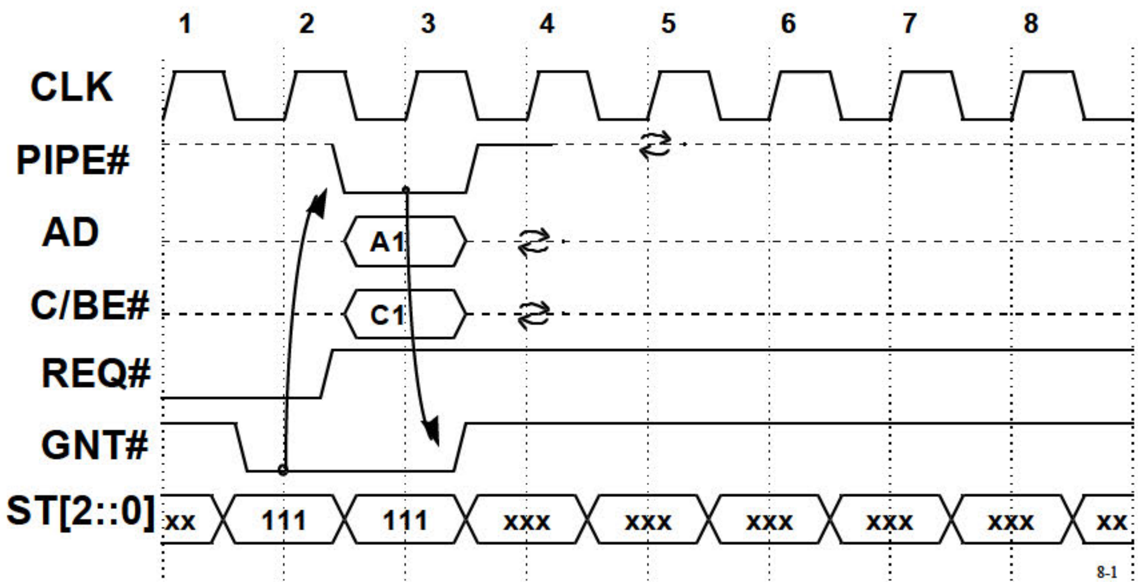


Figure 8-1 Single Address - No Delay by Master

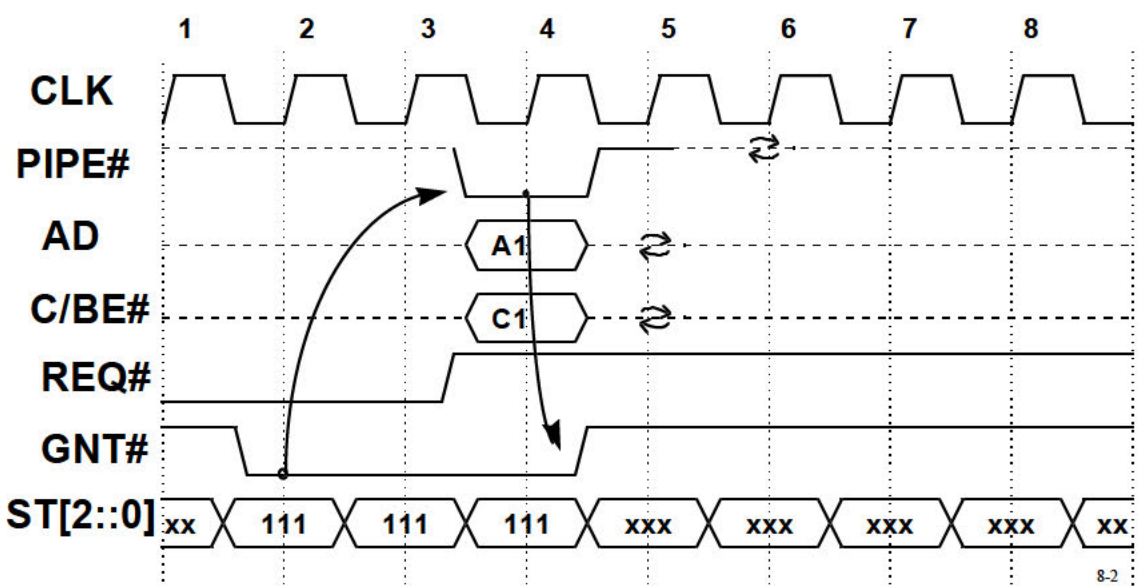


Figure 8-2 Single Address - Maximum Delay by Master

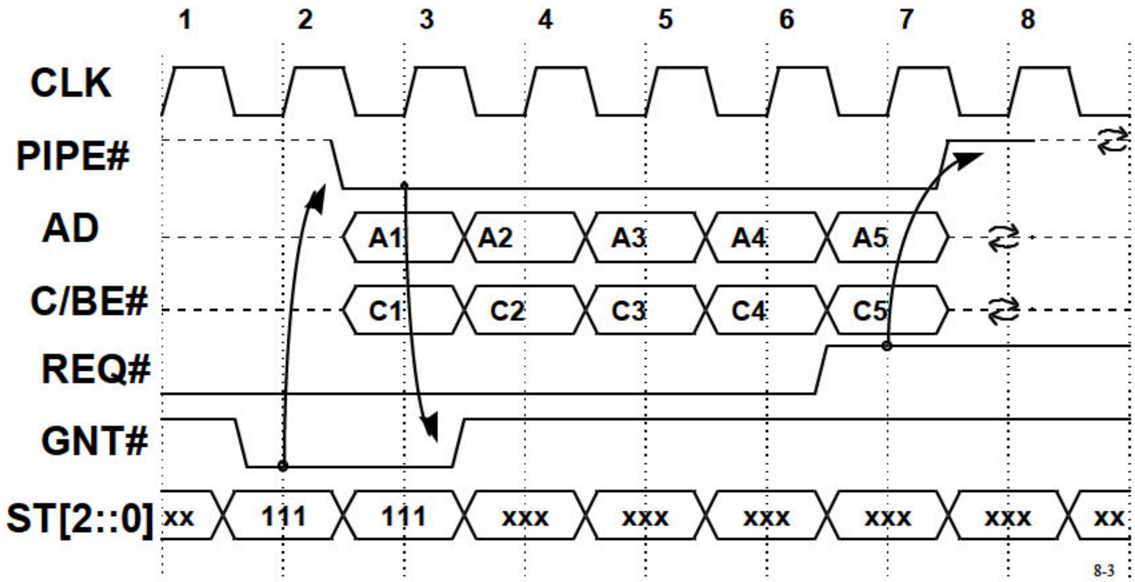


Figure 8-3 Multiple Address - No Delay by Master

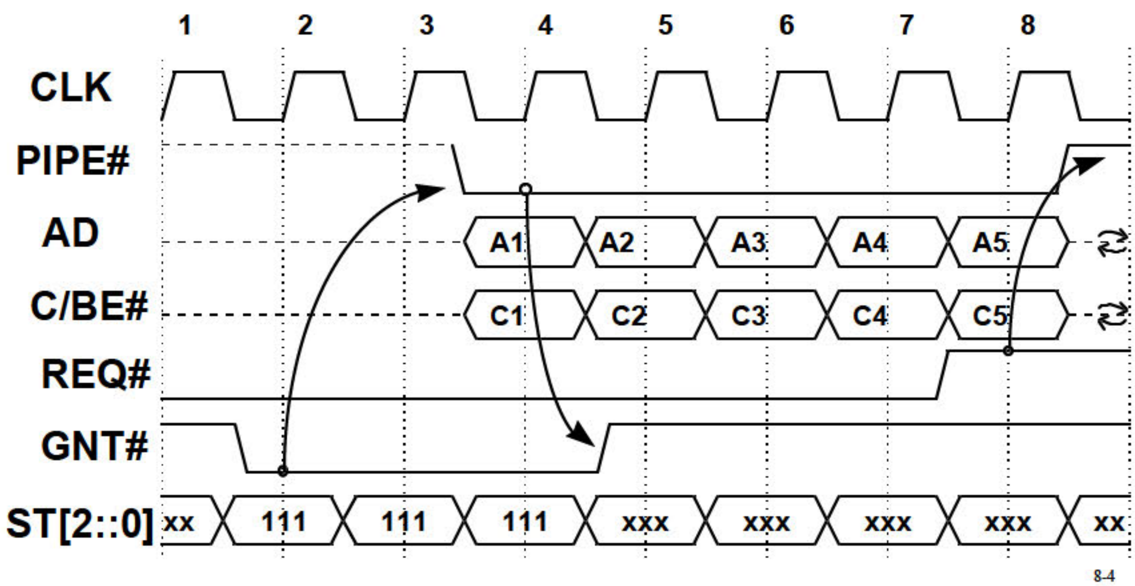


Figure 8-4 Multiple Address - Maximum Delay by Master

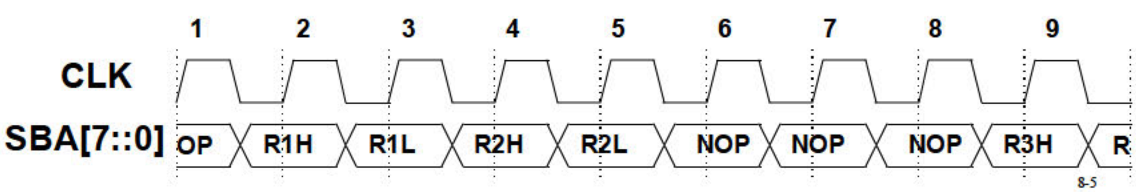


Figure 8-5 1x SideBand Addressing

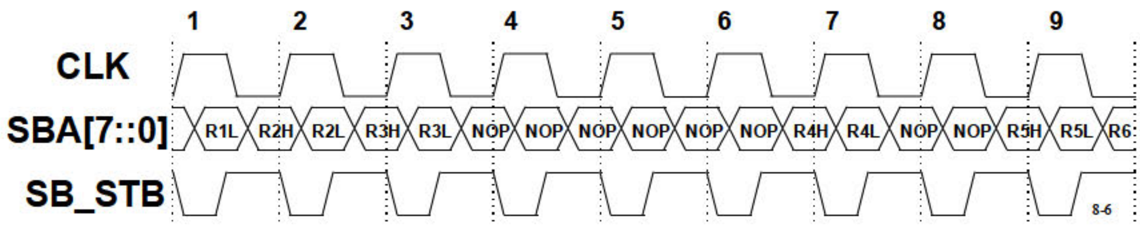


Figure 8-6 2x SideBnd Addressing

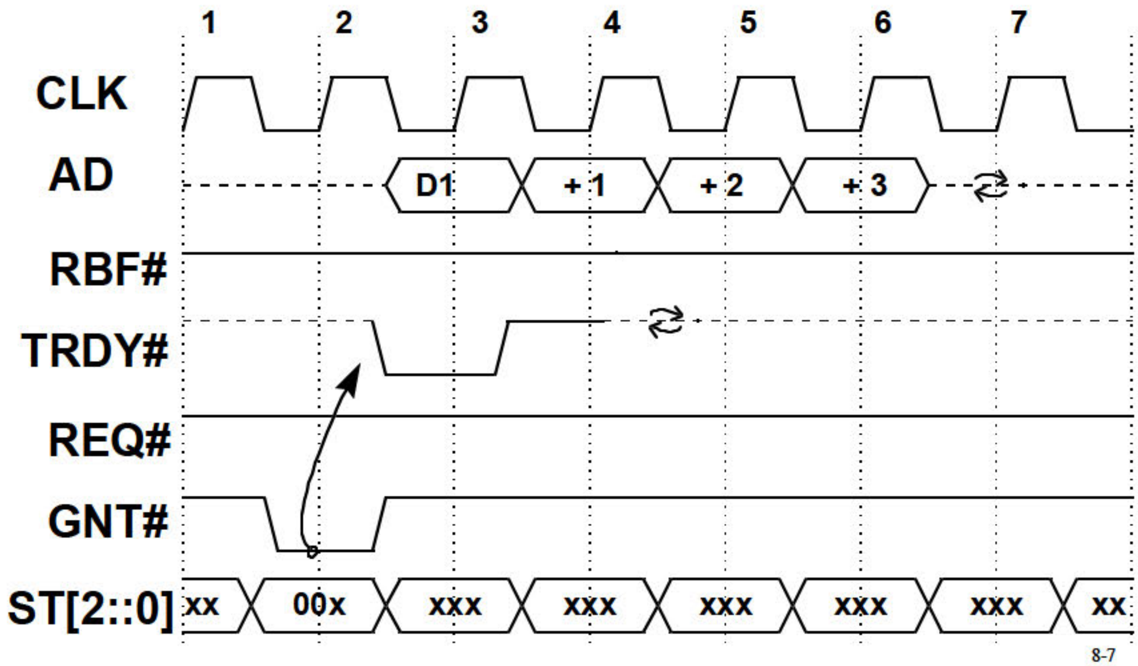


Figure 8-7 Minimum Delay by Target of Read Transaction

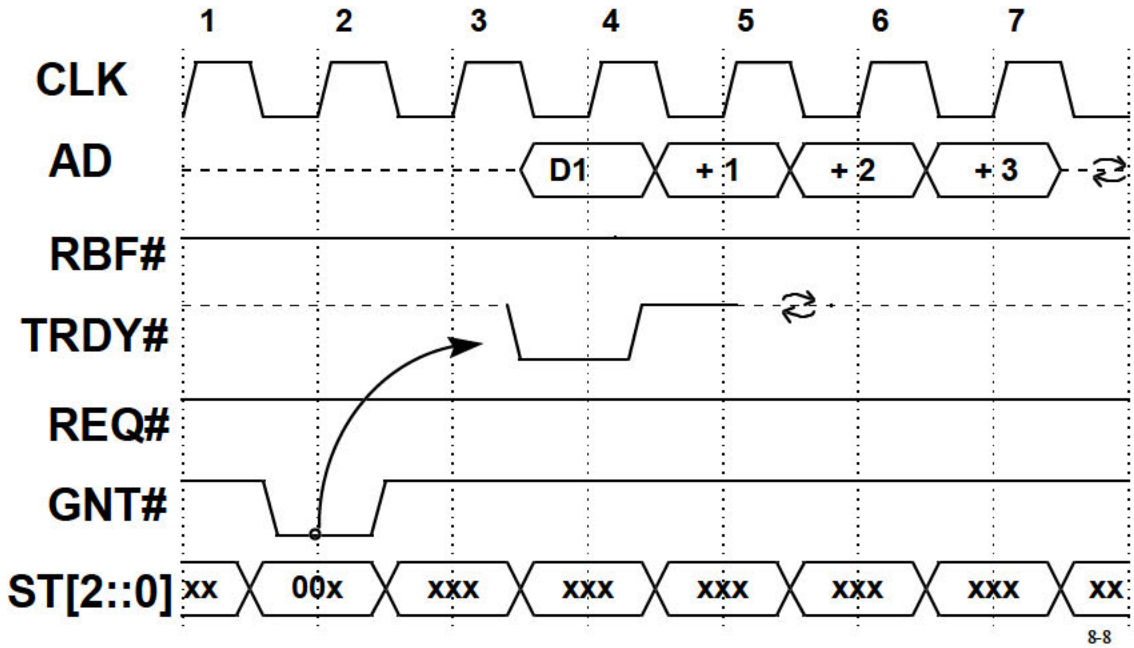


Figure 8-8 Maximum Delay by Target of Read Transaction

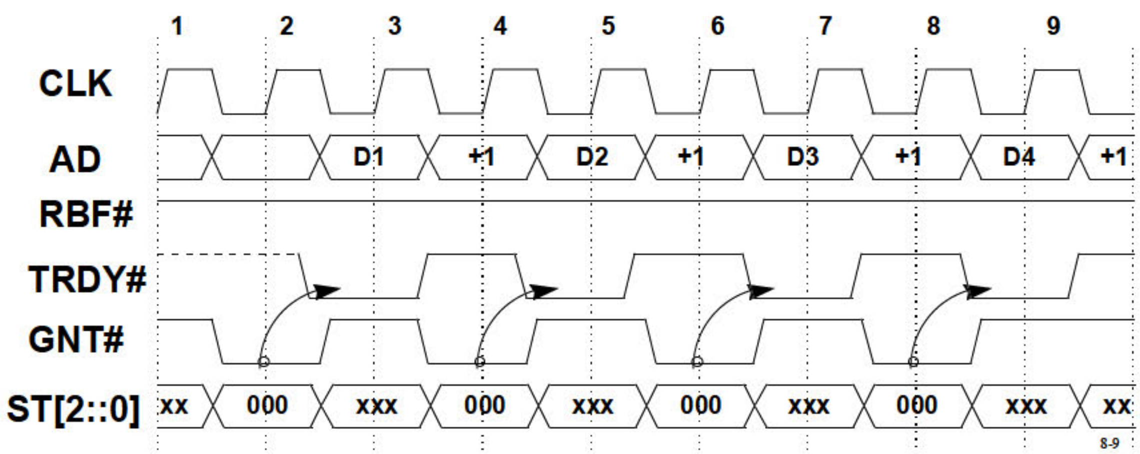


Figure 8-9 Minimum Delay of Back to Back Read Data

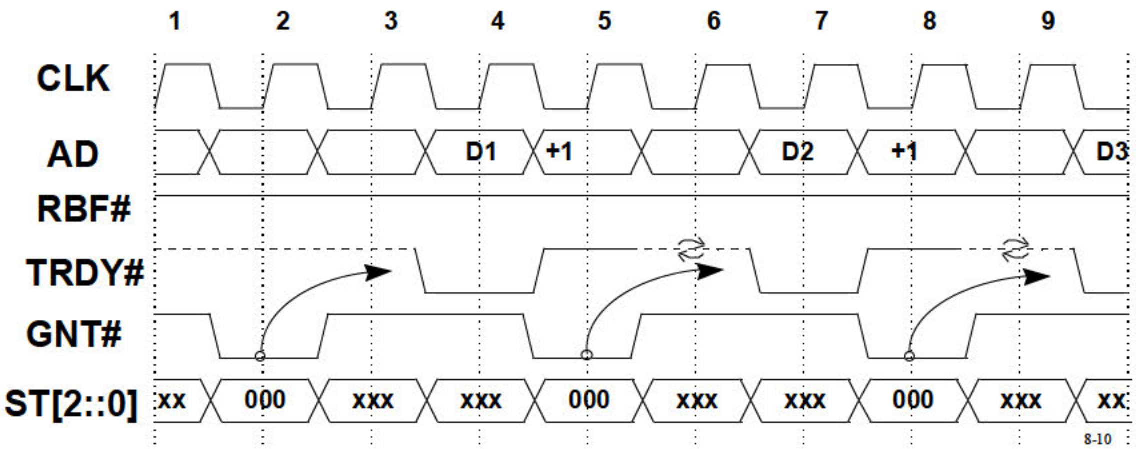


Figure 8-10 Maximum Delay of Back to Back Read Data

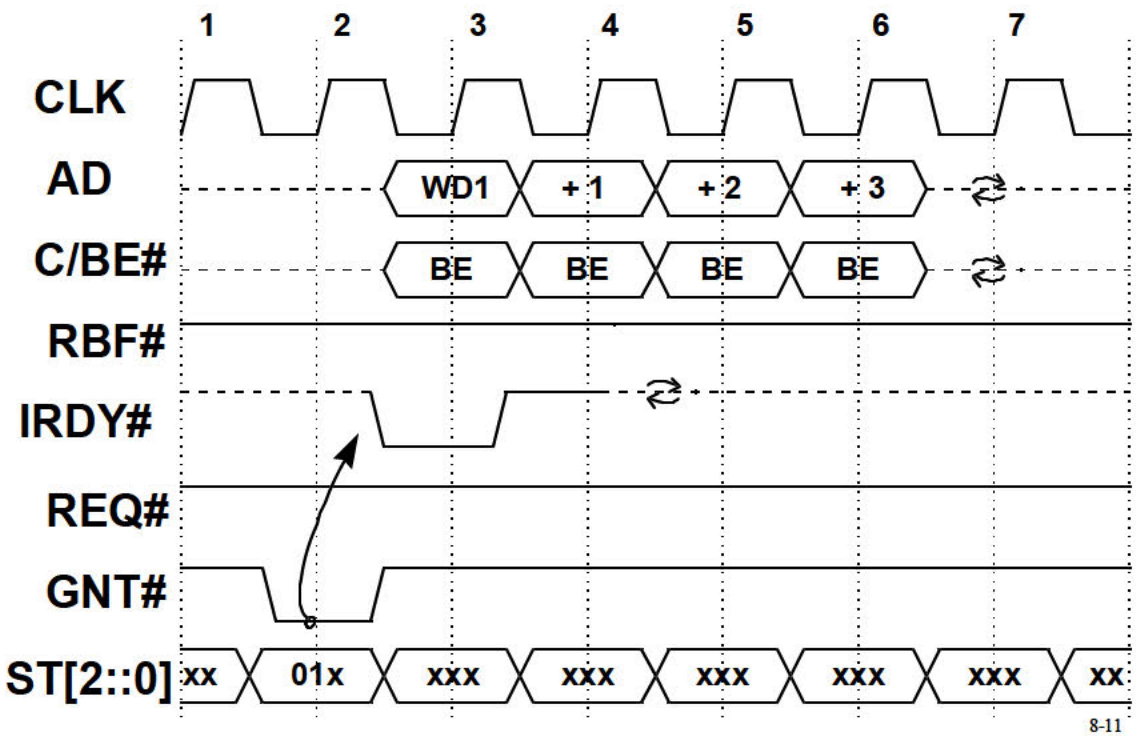


Figure 8-11 Minimum Delay by Master of Write Data

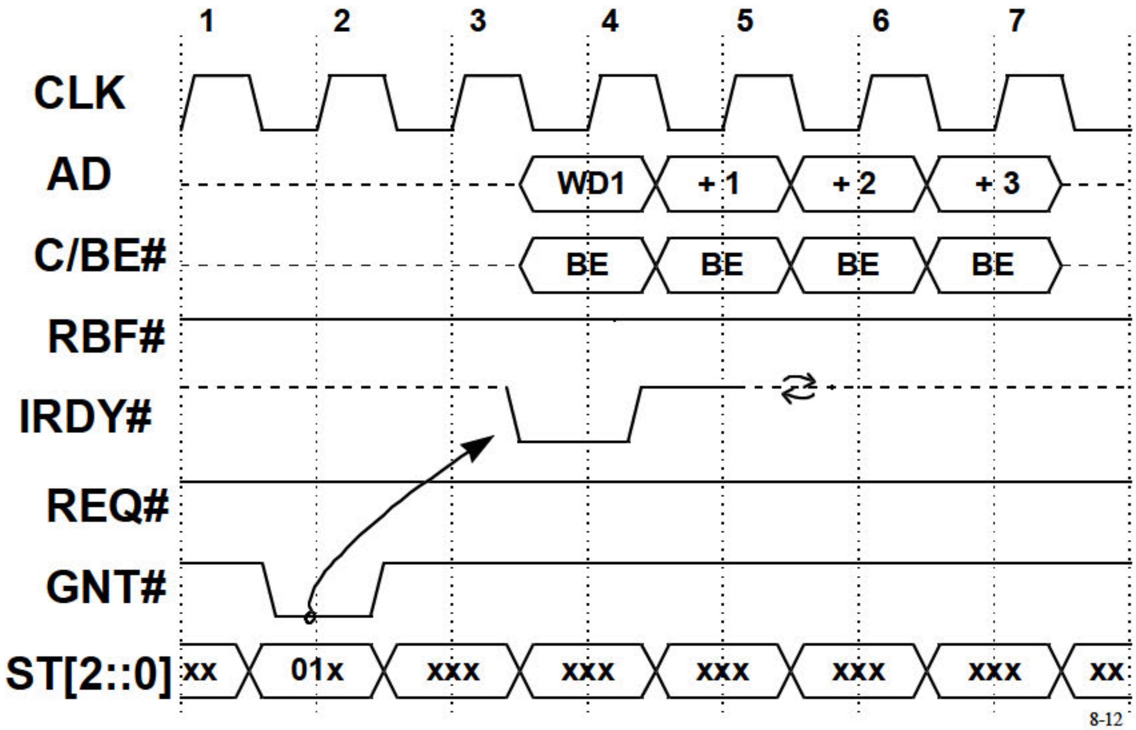


Figure 8-12 Maximum Delay by Master of Write Data

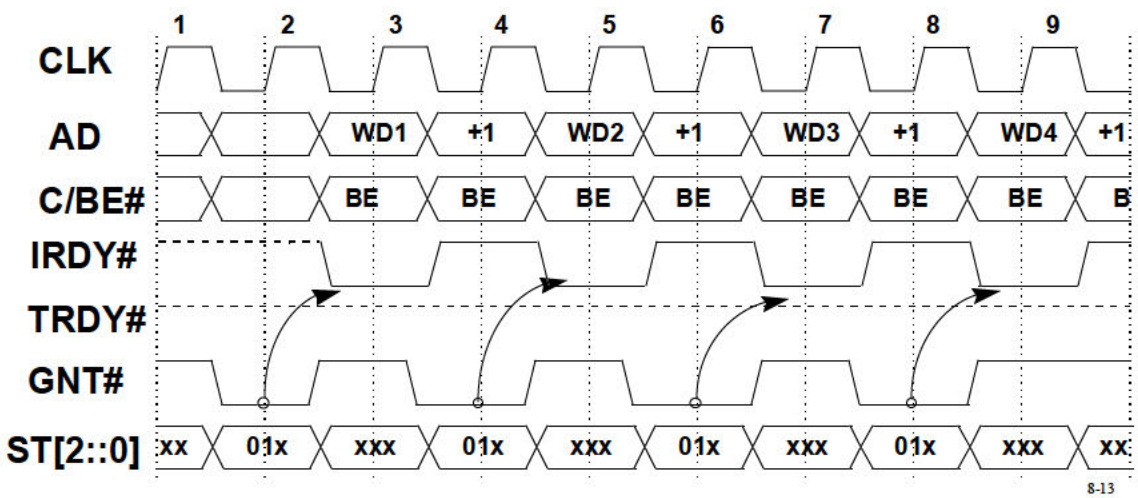


Figure 8-13 Minimum Delay by Master of Back to Back Write Data

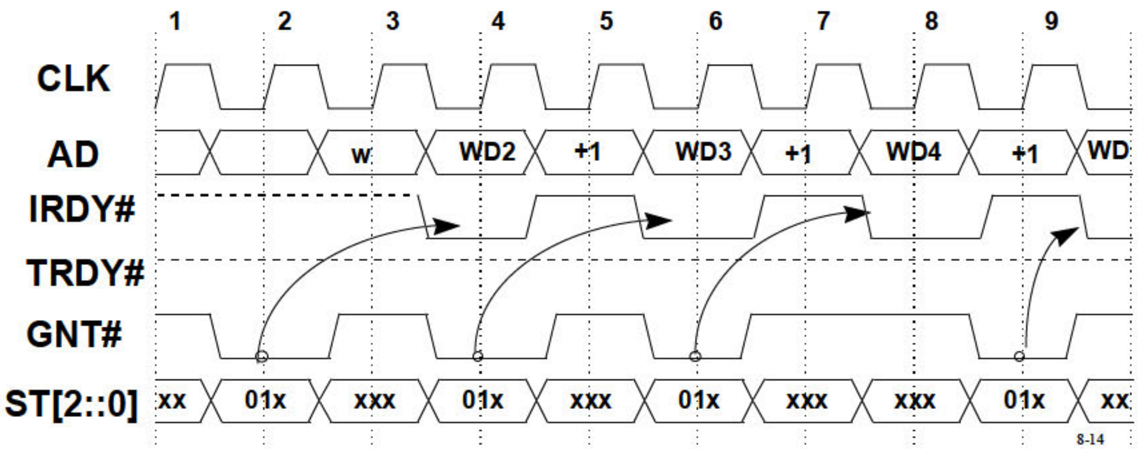


Figure 8-14 Master Delays Initial Write No Delay

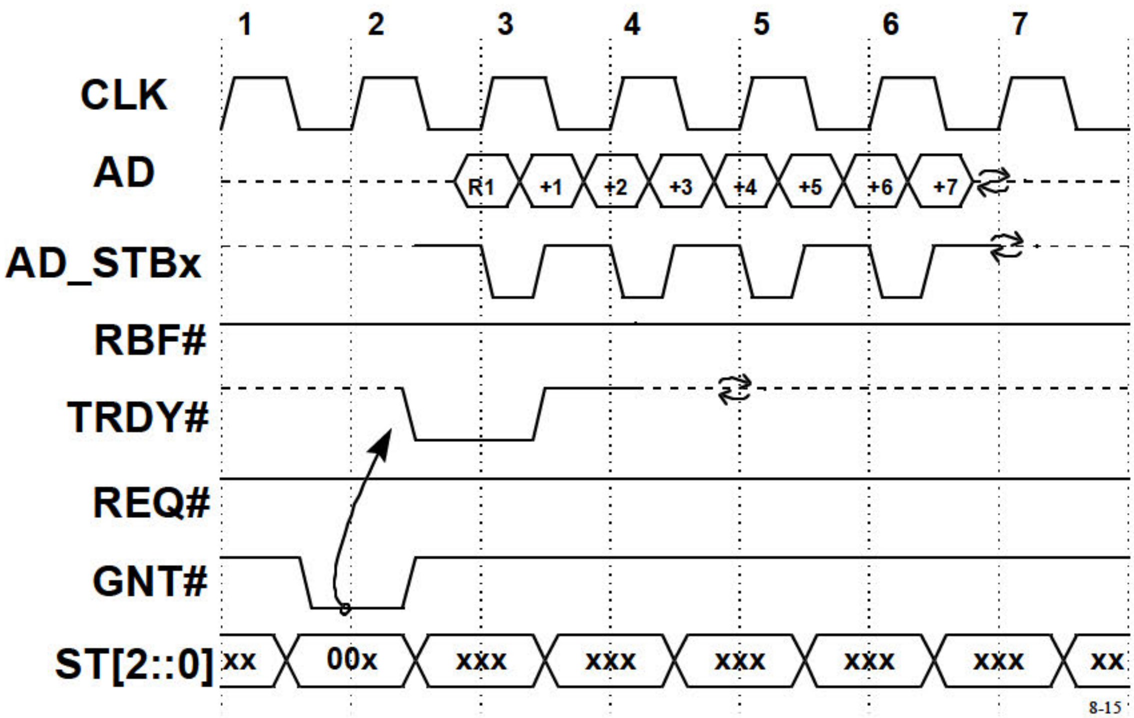


Figure 8-15 2x Read Data - No Delay

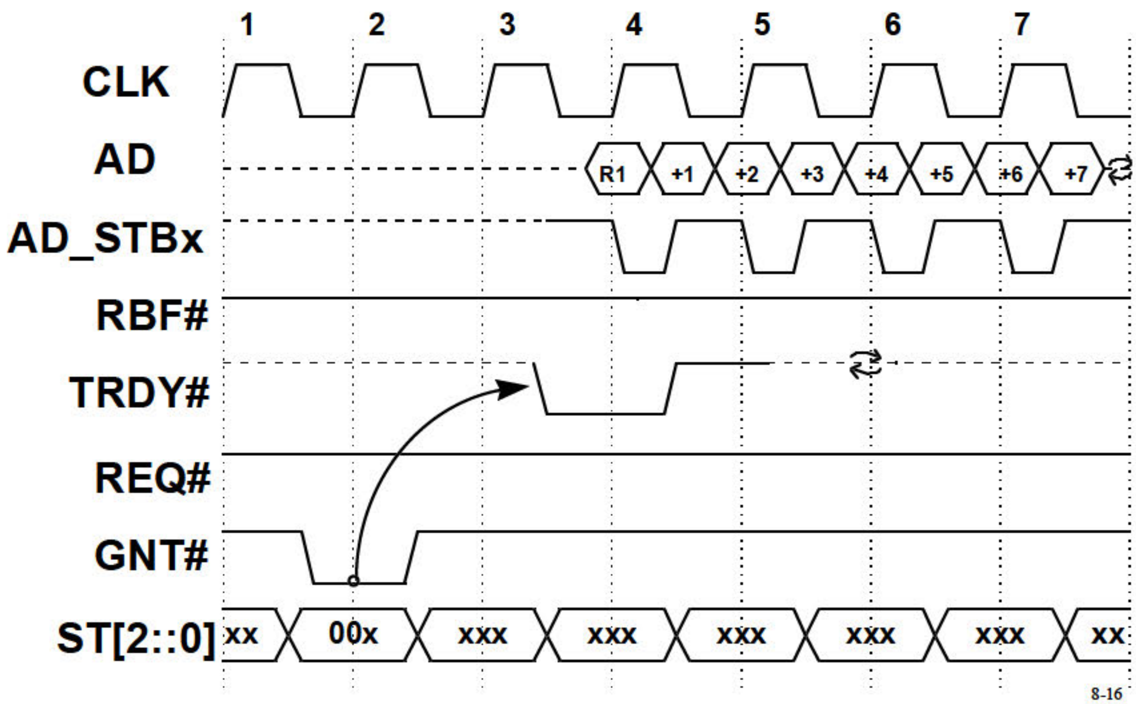


Figure 8-16 2x Read Data - With Delay

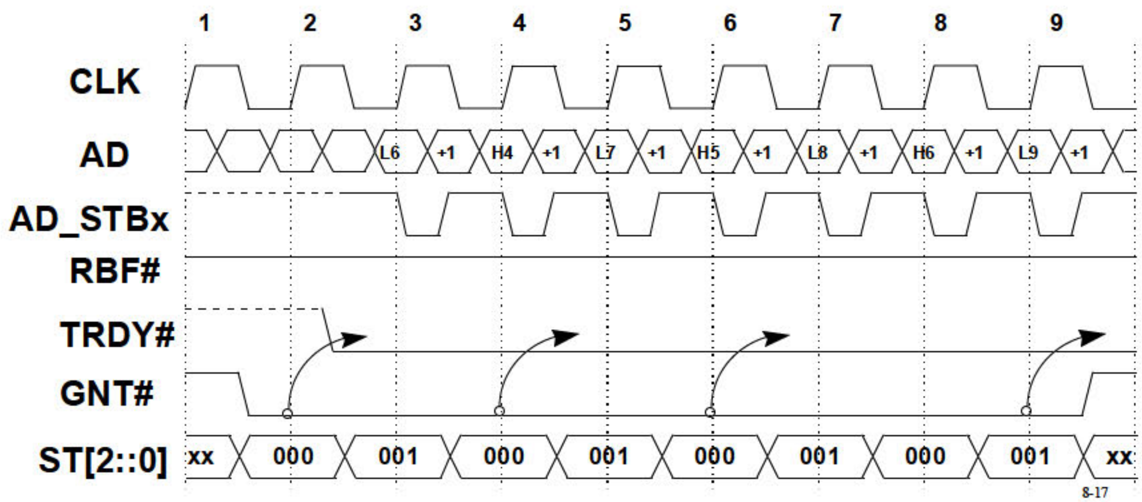


Figure 8-17 2x Back to Back Read Data - No Delay

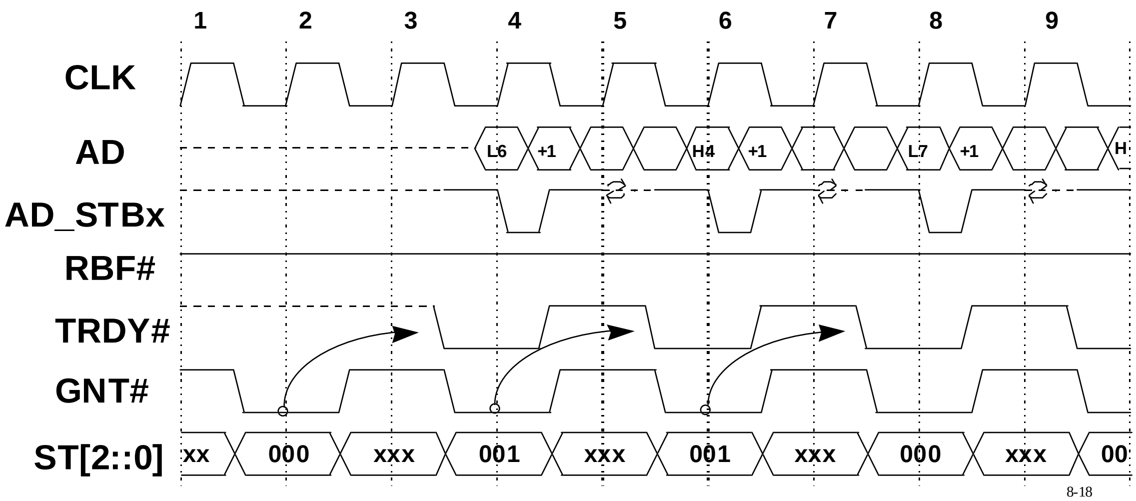


Figure 8-18 2x Back to Back Read Data - Maximum Delay

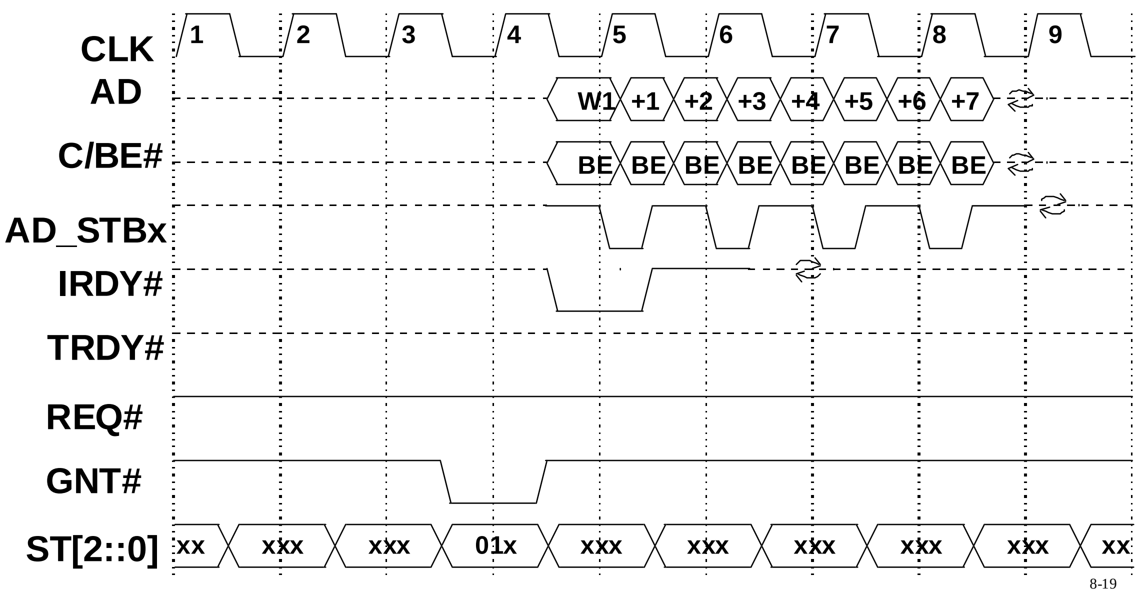


Figure 8-19 2x Write Data - No Delay

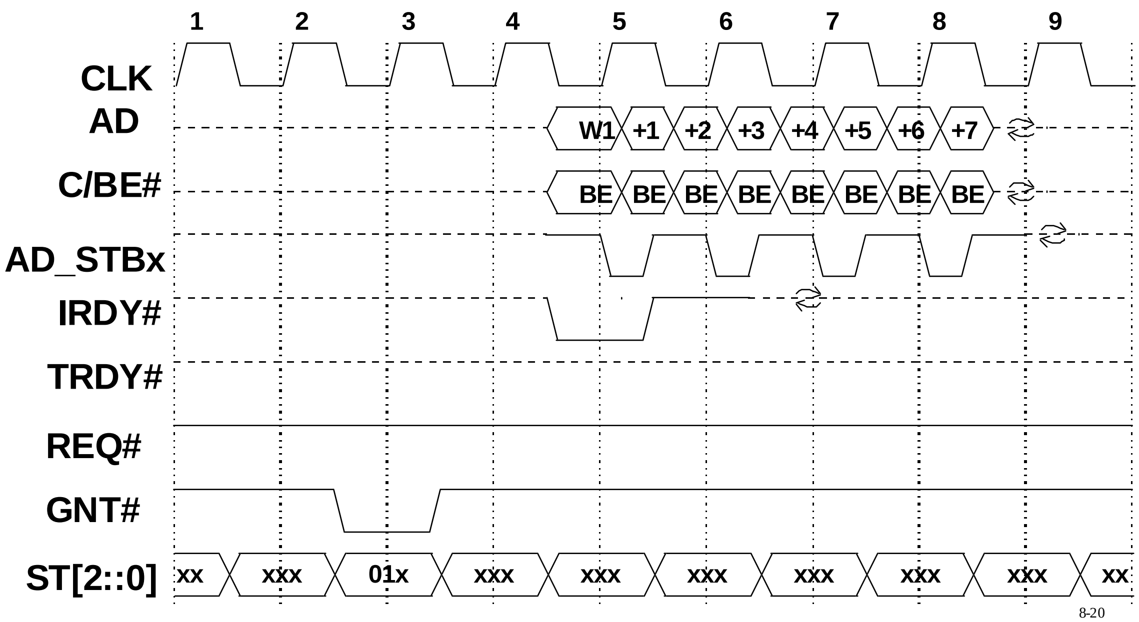


Figure 8-20 2x Write Data - Maximum Delay

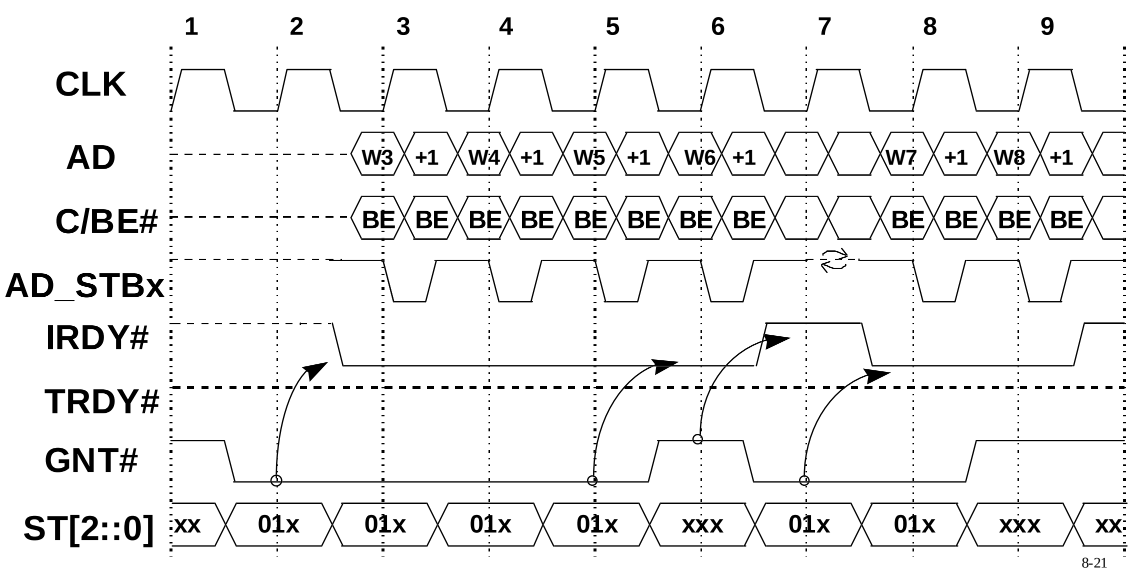


Figure 8-21 2x Back to Back Writes - No Delay

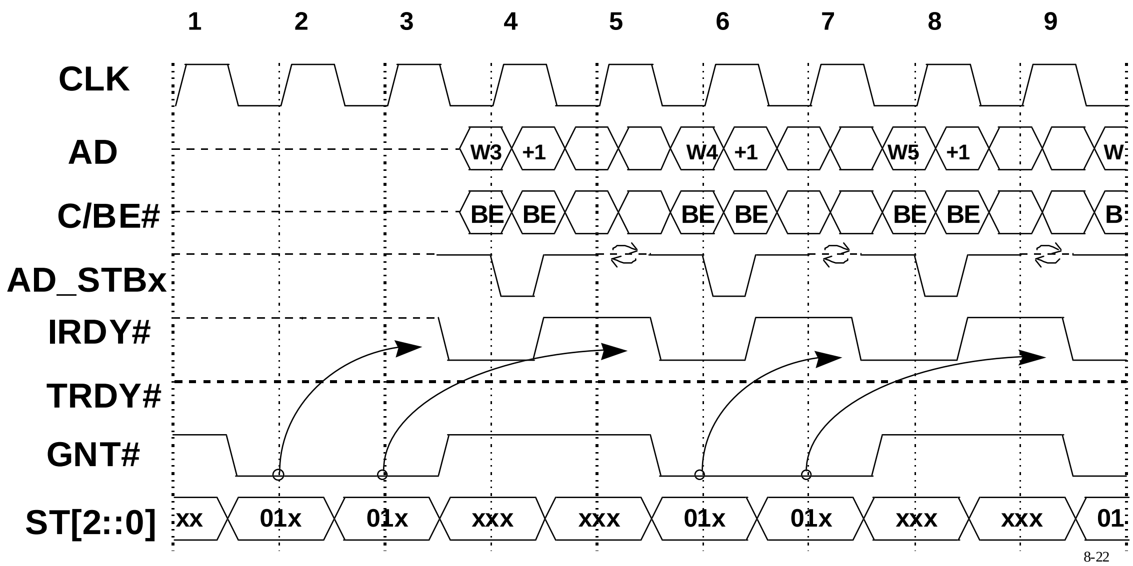


Figure 8-22 2x Back to Back Writes - Maximum Delay

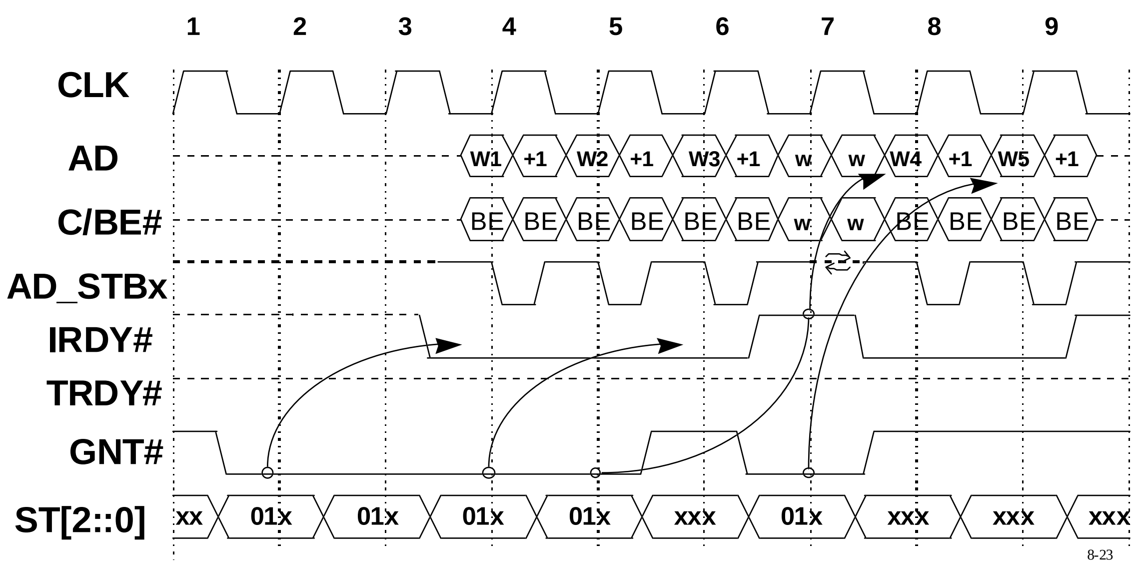


Figure 8-23 2x Writes, Initial Transaction with Delay, Subsequent Transactions No Delay

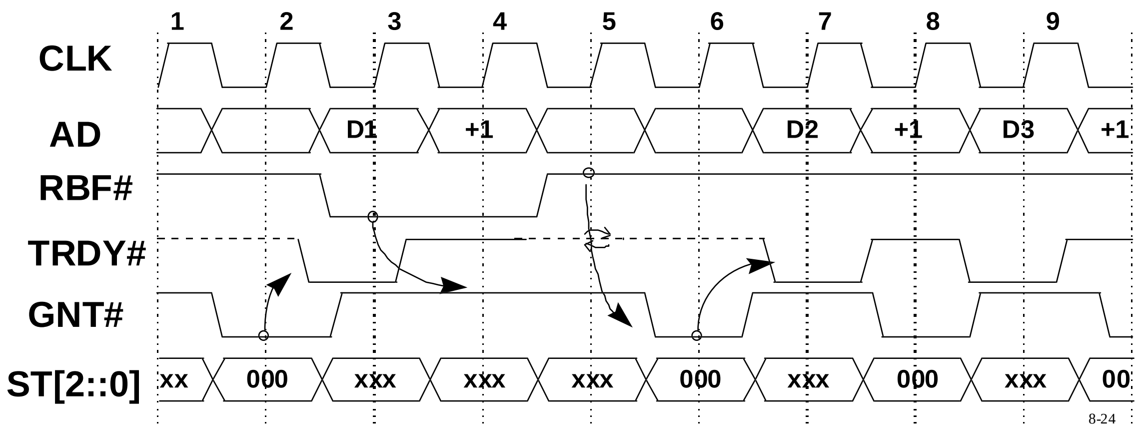


Figure 8-24 Master Data Buffer Full - Minimum Delay for Read Data

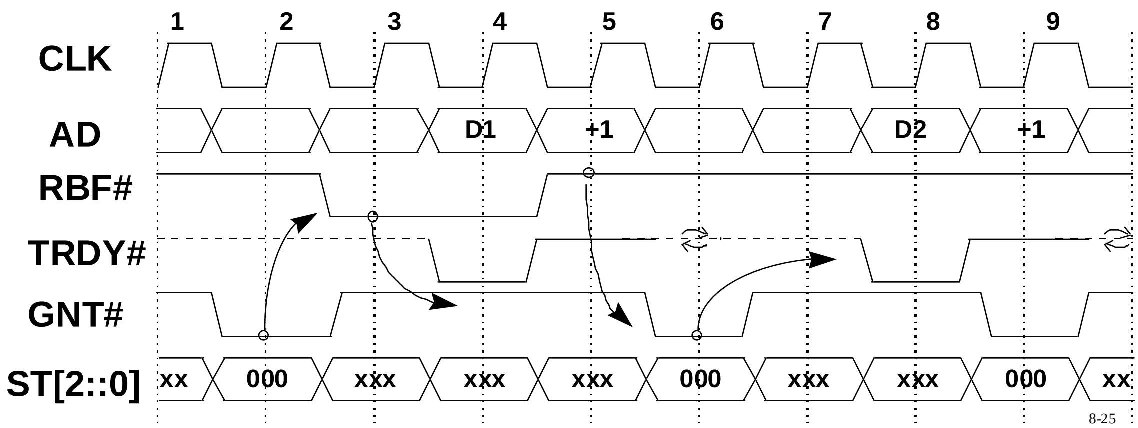


Figure 8-25 Master Data Buffer Full - Maximum Delay for Read Data

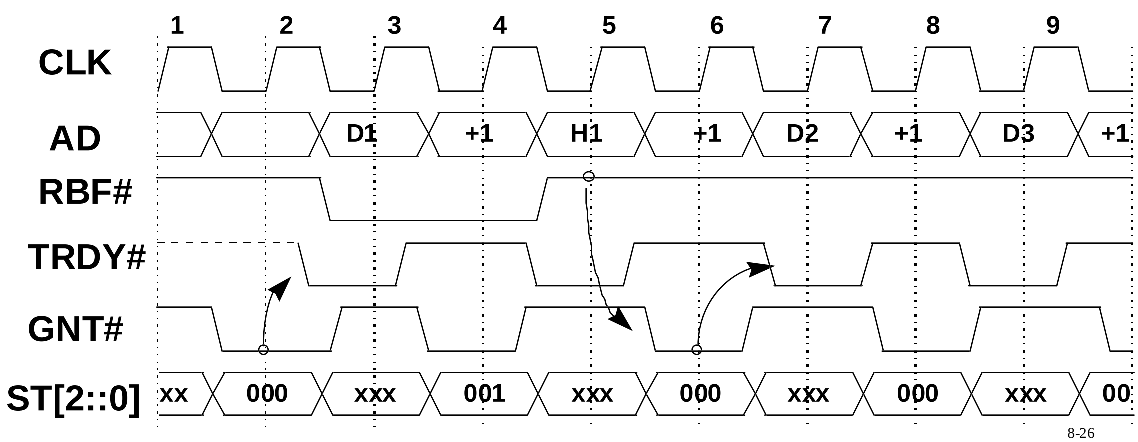


Figure 8-26 RBF# Asserted, HP Read Data Returned

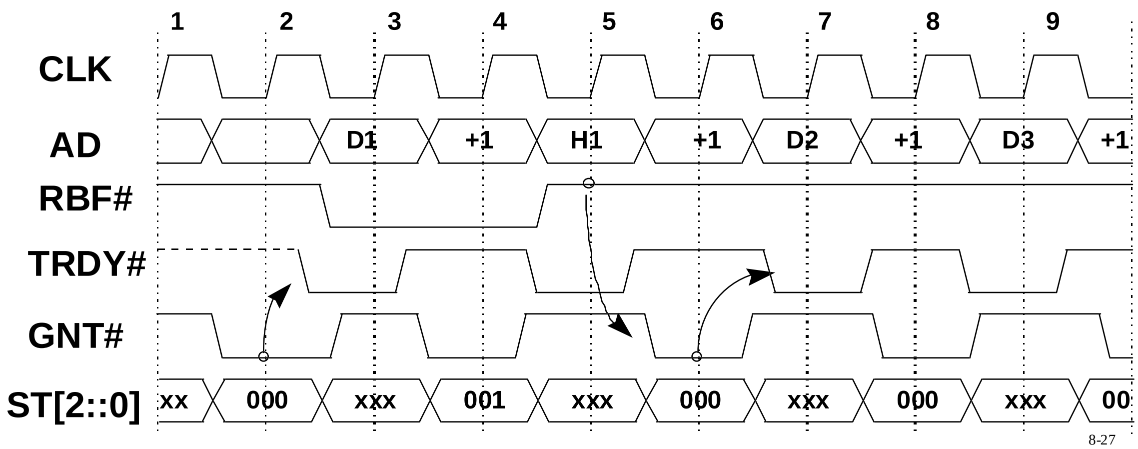


Figure 8-27 RBF# Asserted, Maximum Delay by Target, HP Read Data Returned

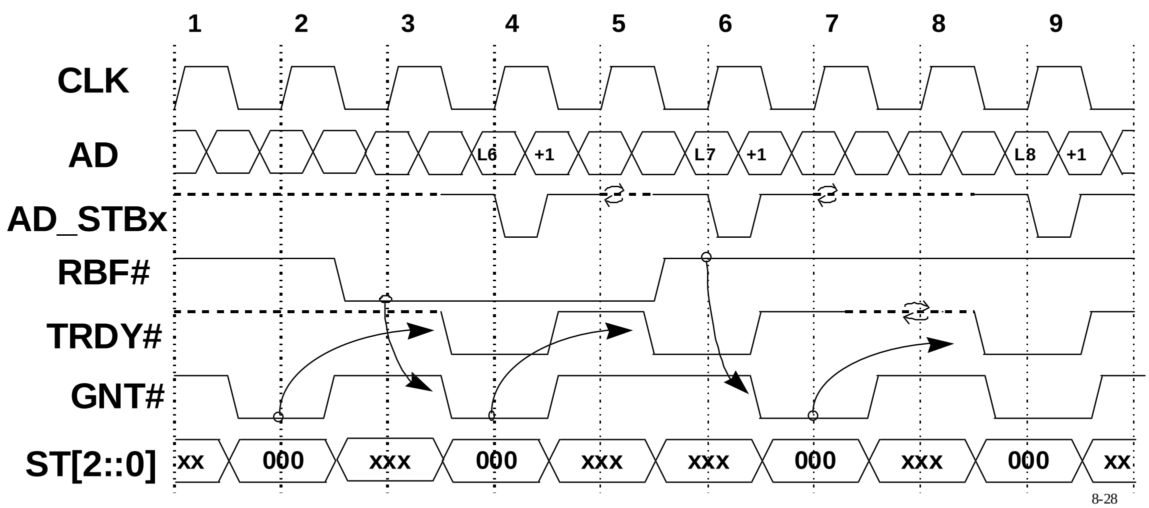


Figure 8-28 2x Read Data, RBF# Asserted, Maximum Delay By Target

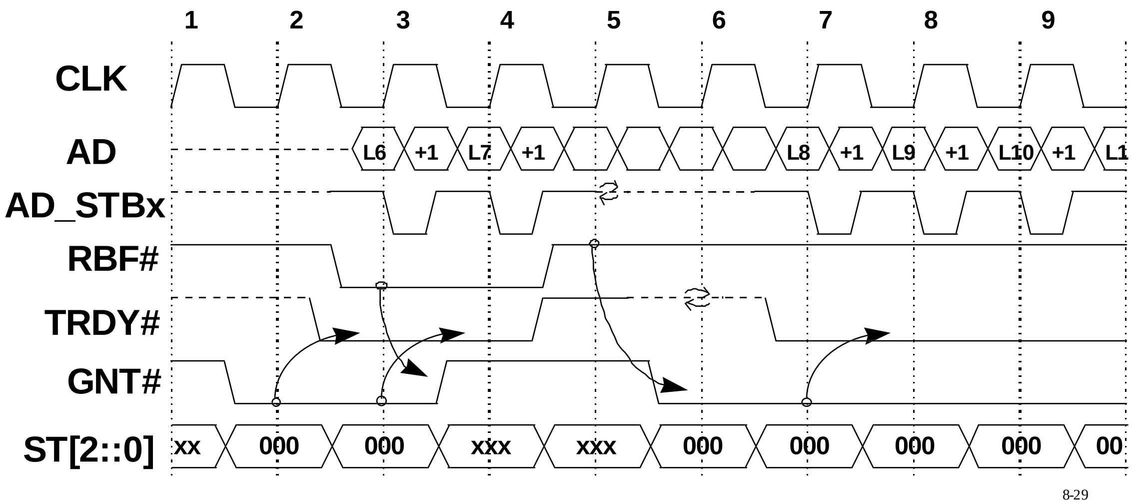


Figure 8-29 2x Read, RBF# Asserted, No HP Read Data

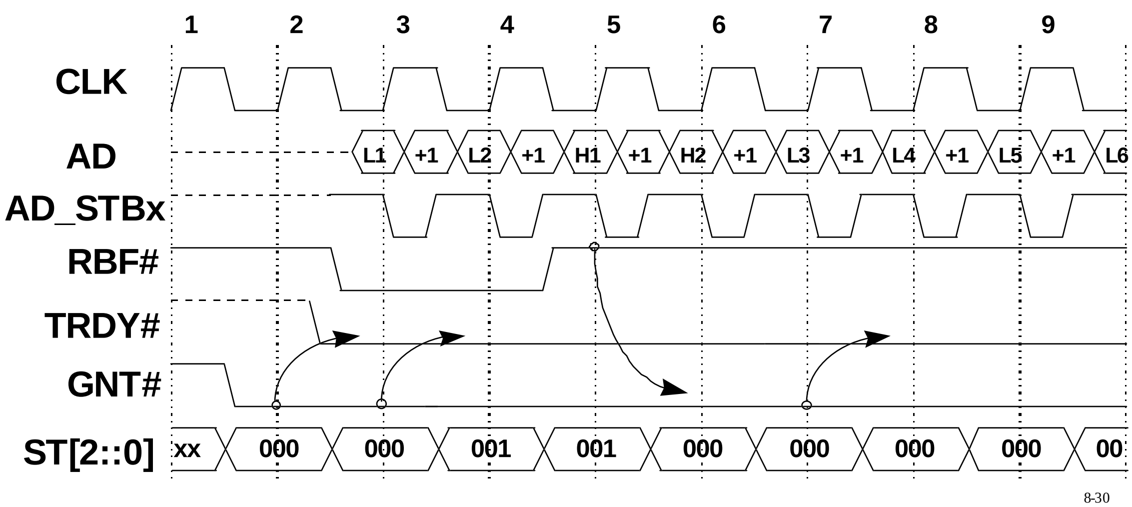


Figure 8-30 2x Read Data, with Delay, RBF# Asserted, HP Read Data Returned

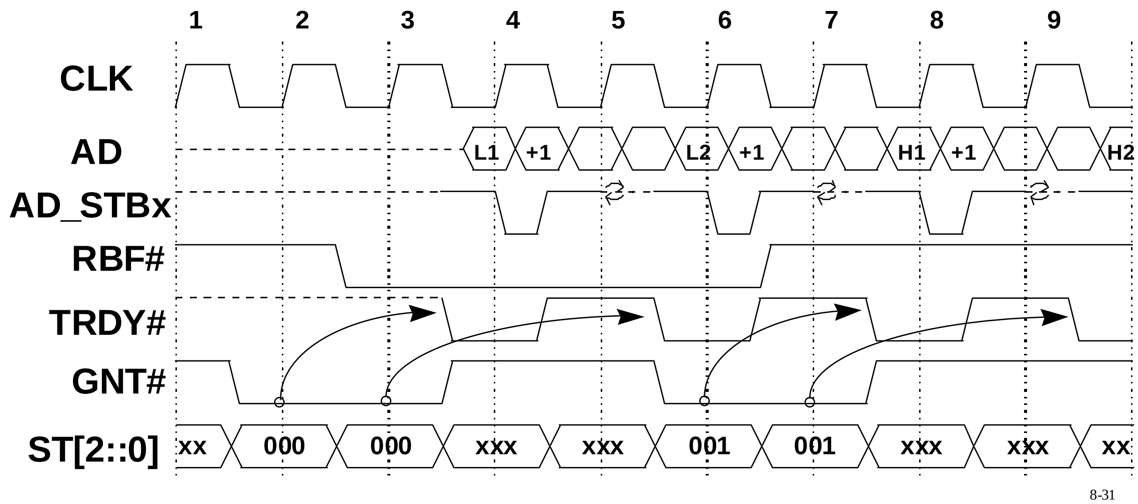


Figure 8-31 2x Read Data, RBF# Asserted, HP Read Data Delayed

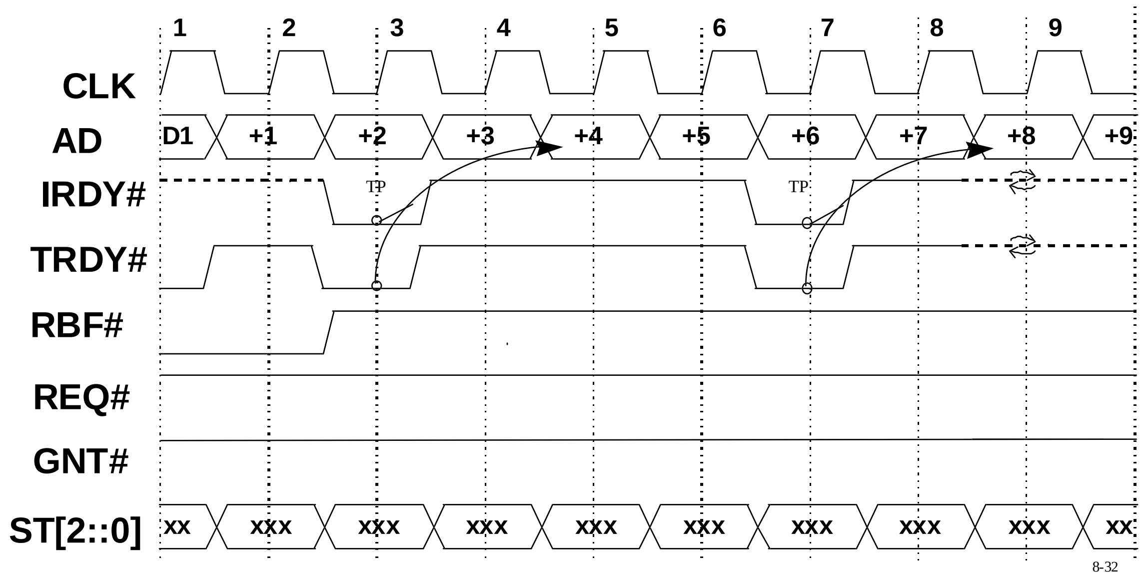


Figure 8-32 Read Data, No Delay Subsequent Block

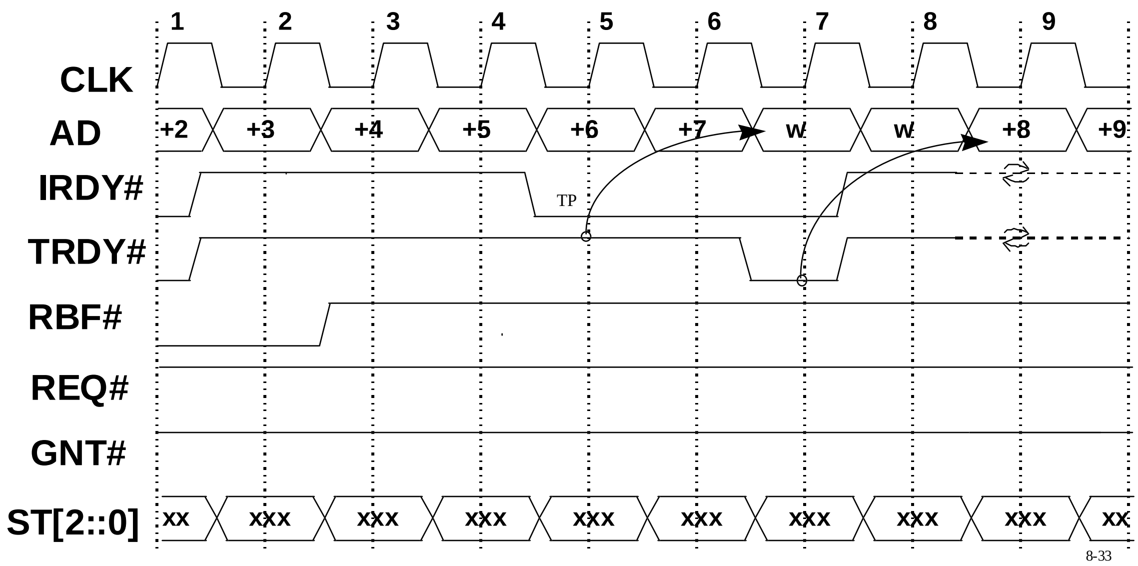


Figure 8-33 Read Data, Master Ready - Target Delays Subsequent Block

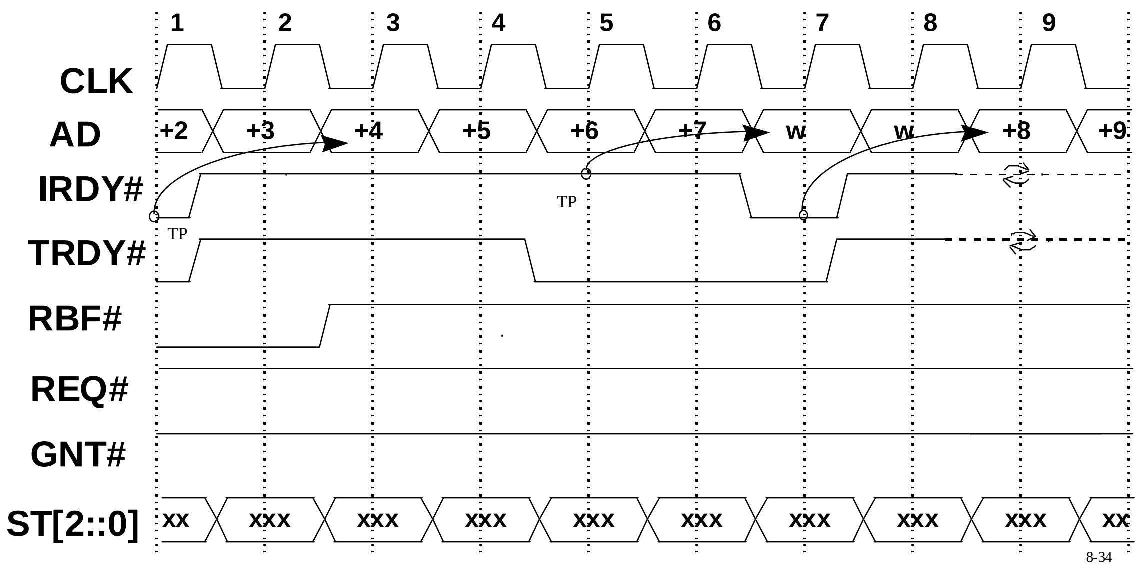


Figure 8-34 Read Data, Target Ready - Master Delays Subsequent Block

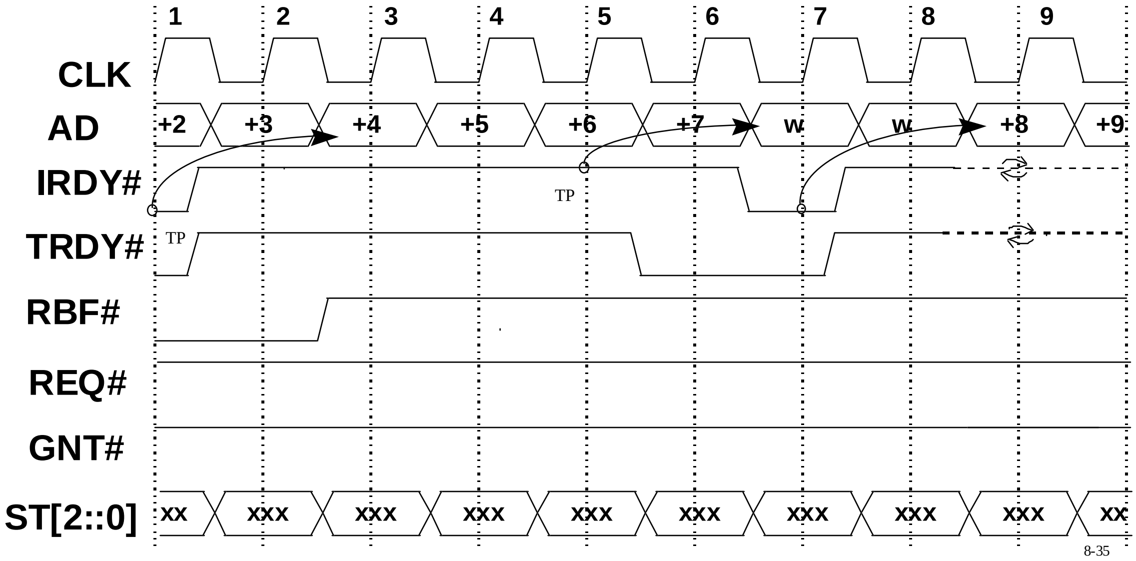


Figure 8-35 Read Data, Target Delays 1 clock - Master Delays 2 clocks Subsequent Block

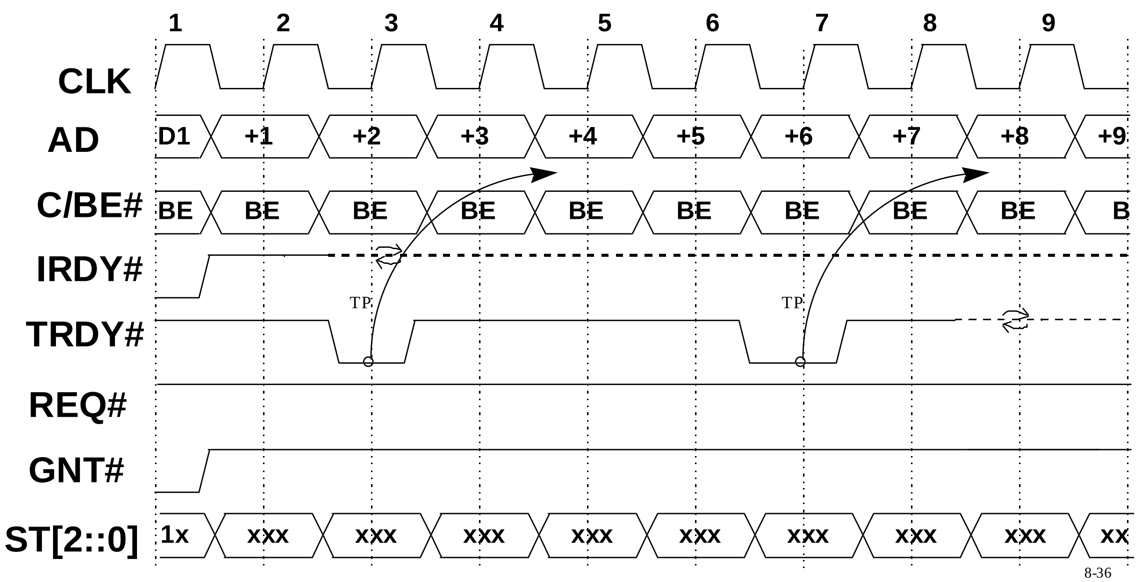


Figure 8-36 Write Data, No Delay Subsequent Block

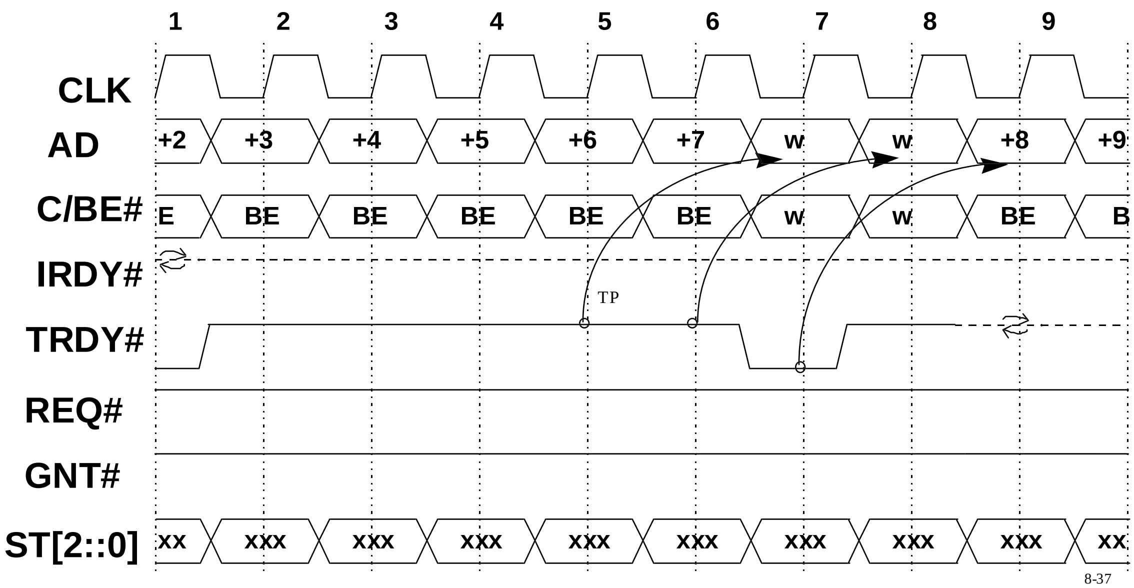


Figure 8-37 Write Data, Target Delays Subsequent Block 2 Clocks

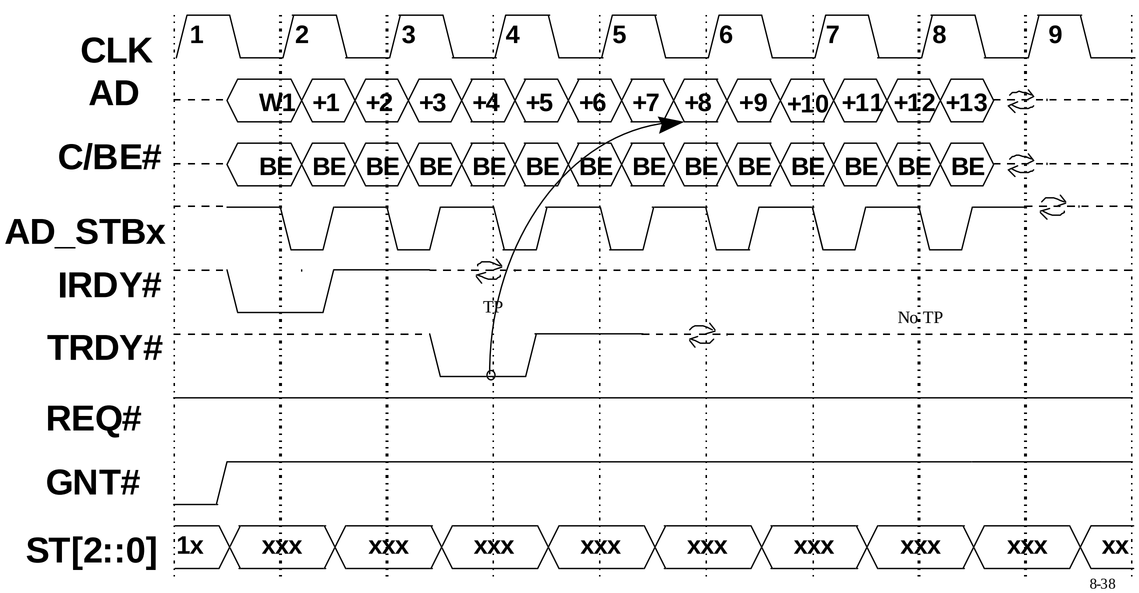


Figure 8-38 2x Write No Delay Initial or Subsequent Block

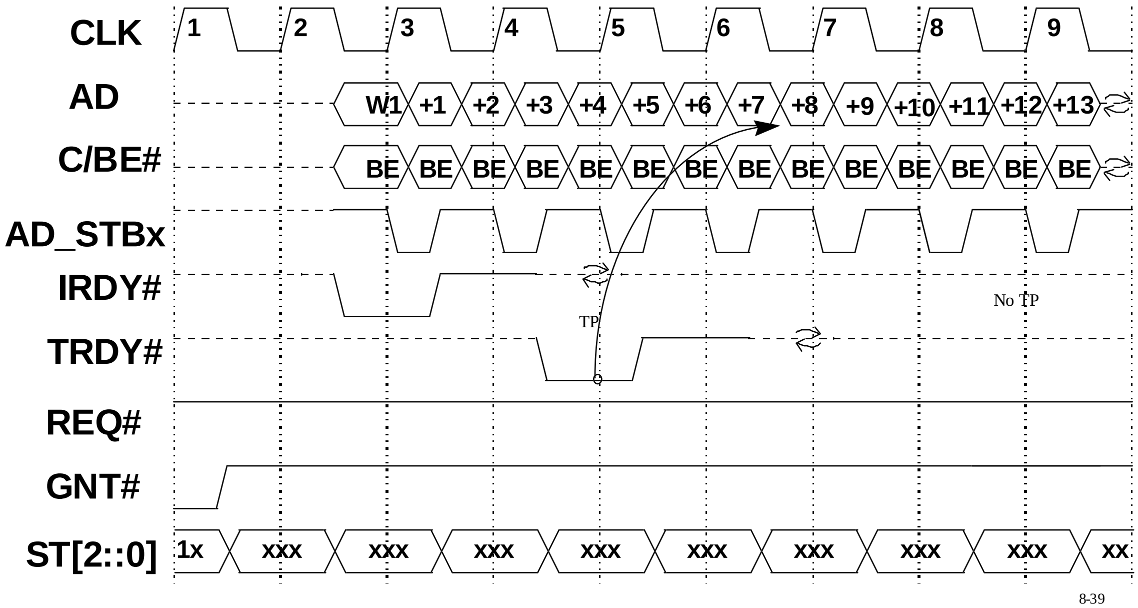


Figure 8-39 2x Write, Initial Delay, No Delay Subsequent Block

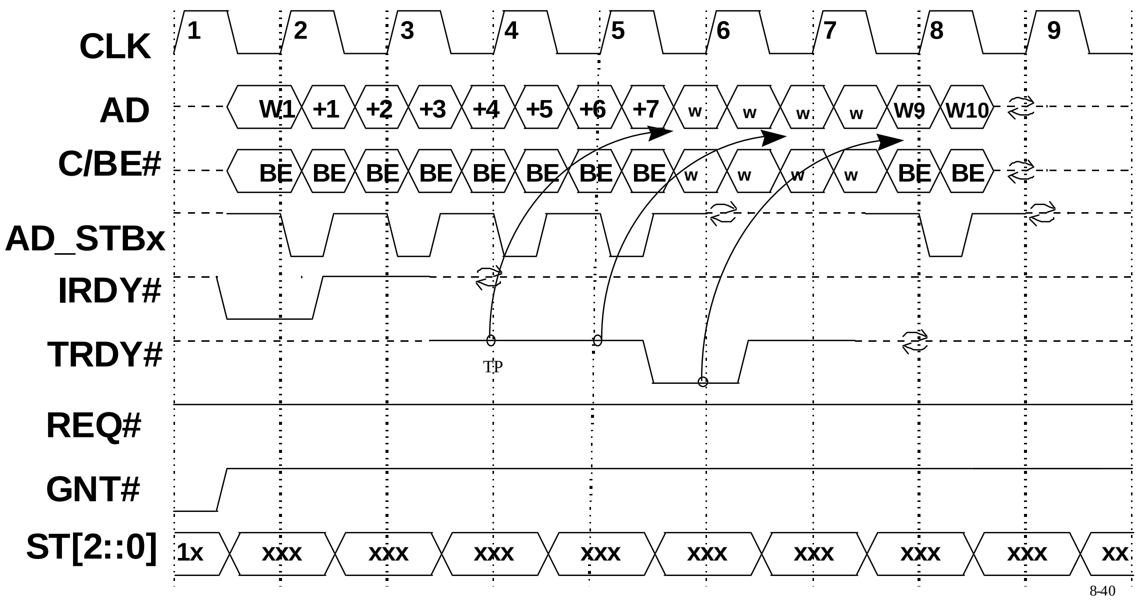


Figure 8-40 2x Write, No Initial Delay, 2 clocks Delay Subsequent Block

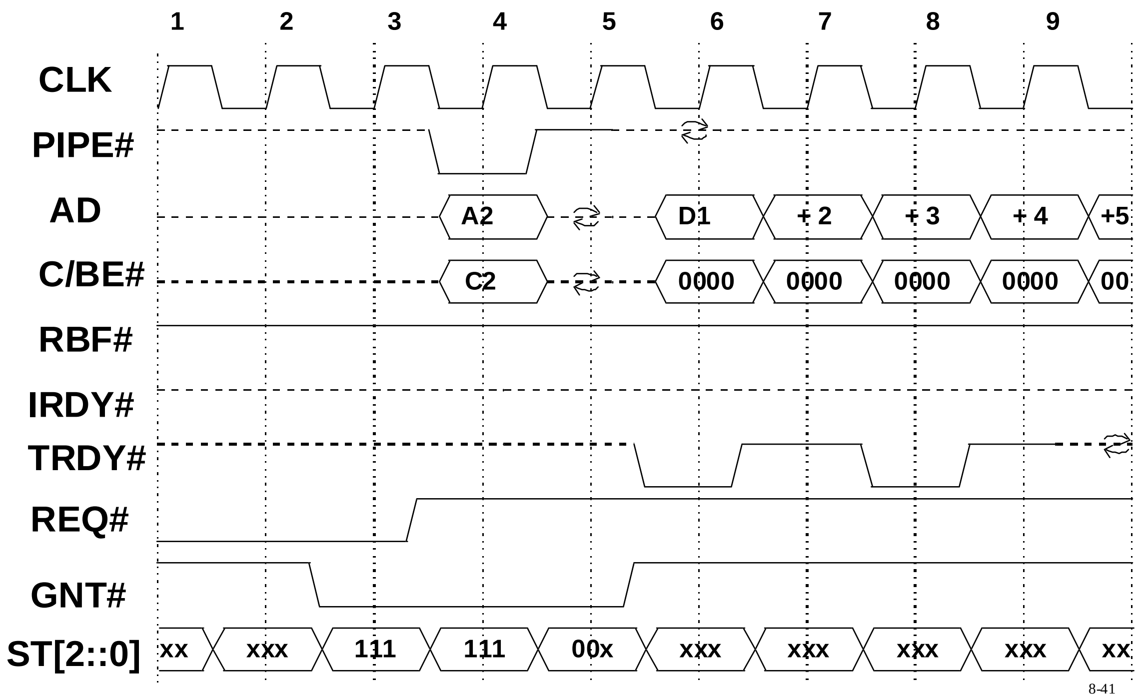


Figure 8-41 Earliest Read Data after Request

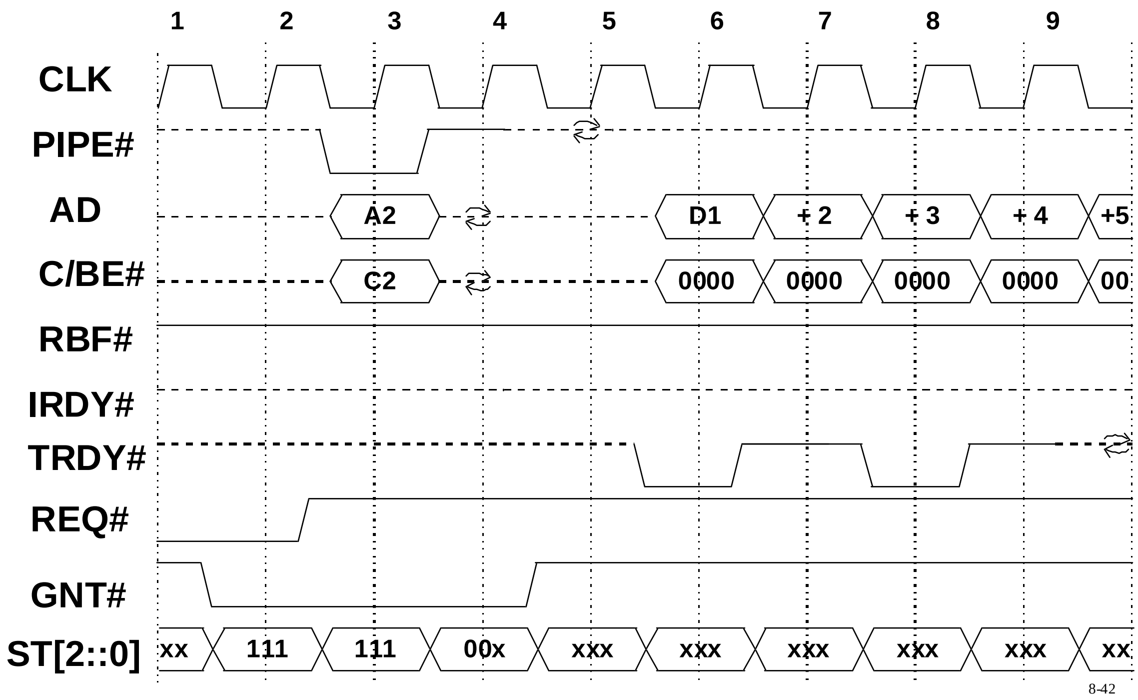


Figure 8-42 Maximum Delay Read Data after Request

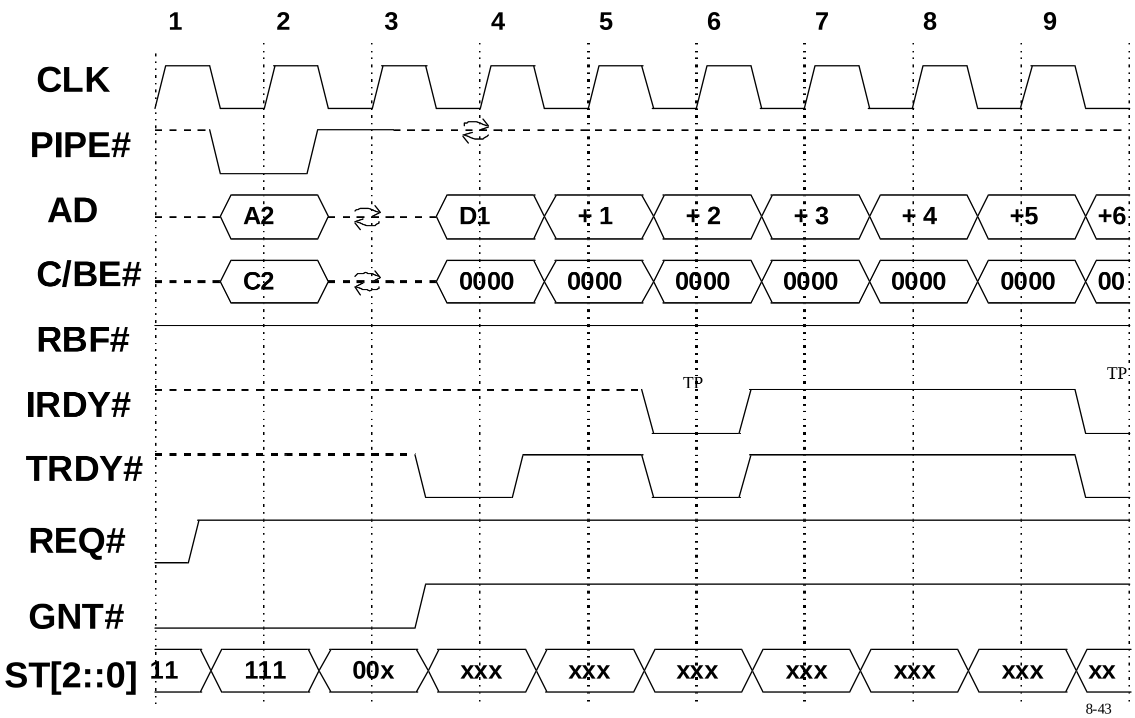


Figure 8-43 Request followed by Long Data Read

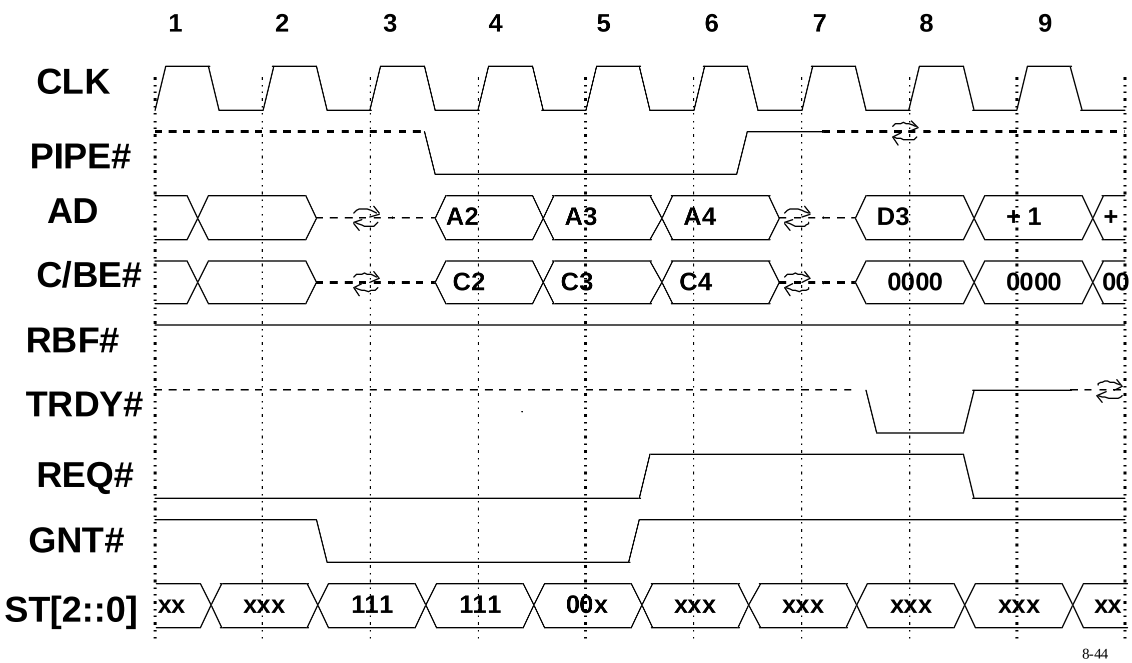


Figure 8-44 Early GNT# for Read Data

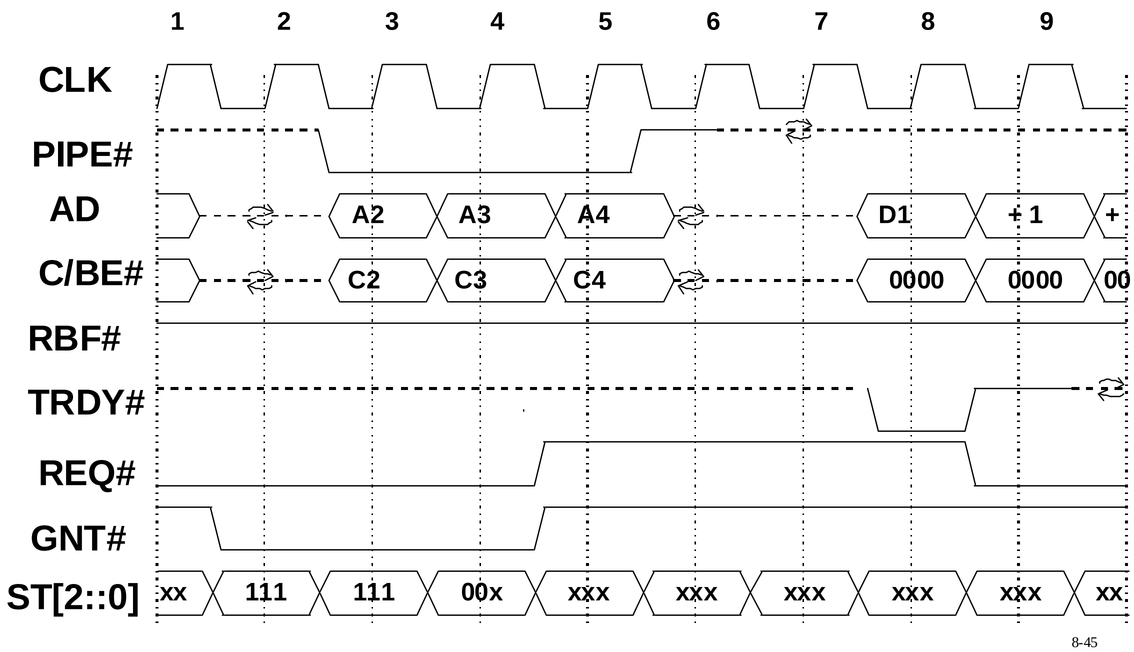


Figure 8-45 Early GNT# for Read Data, Read Data Delayed

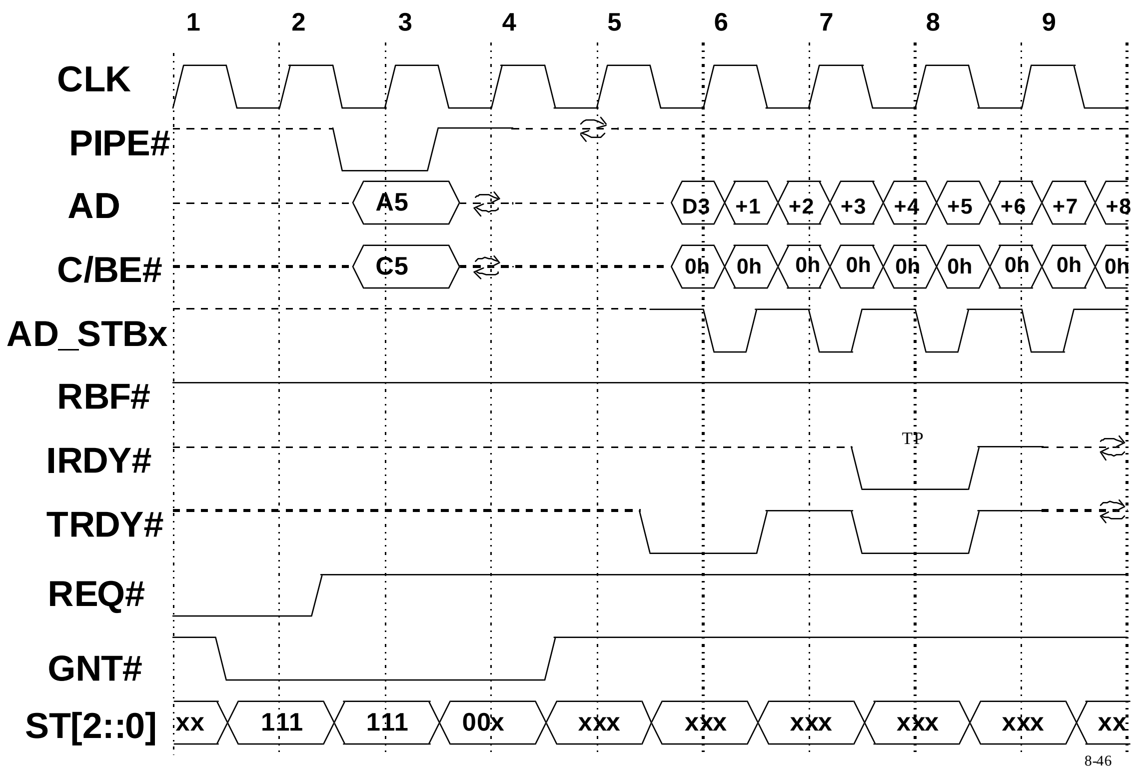


Figure 8-46 2x Read, Early GNT#, Delayed Read Data

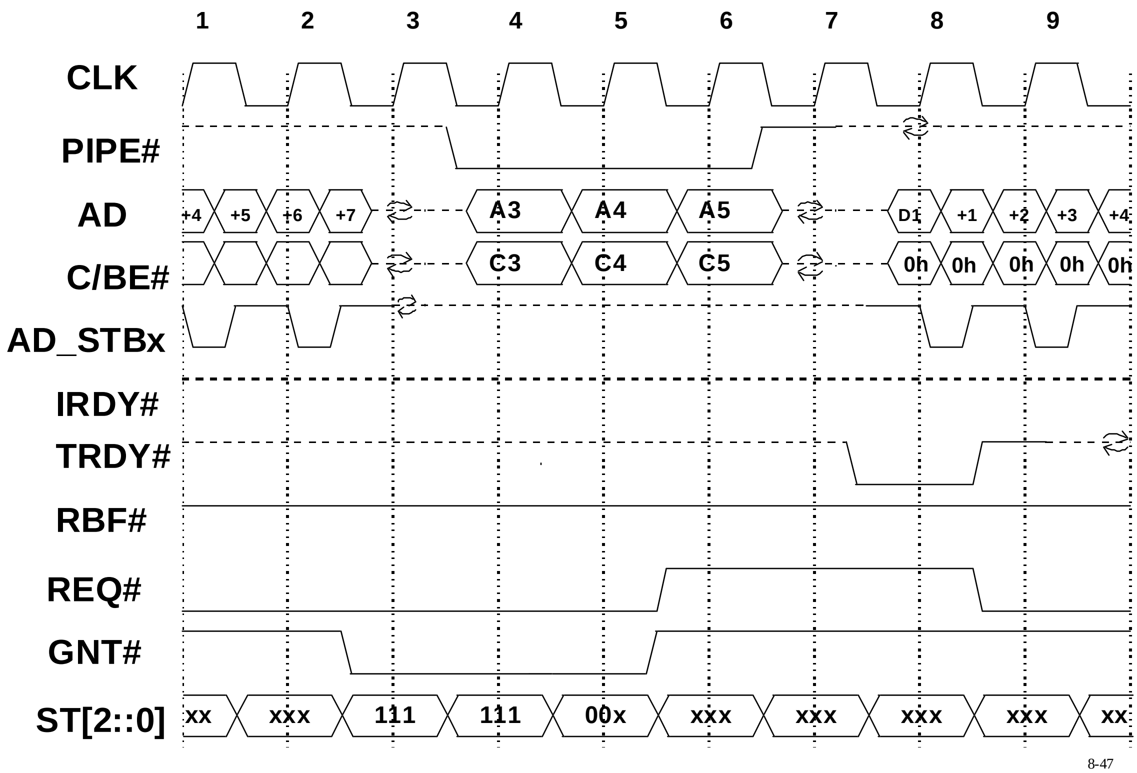


Figure 8-47 Data Transfers Intervened with Requests

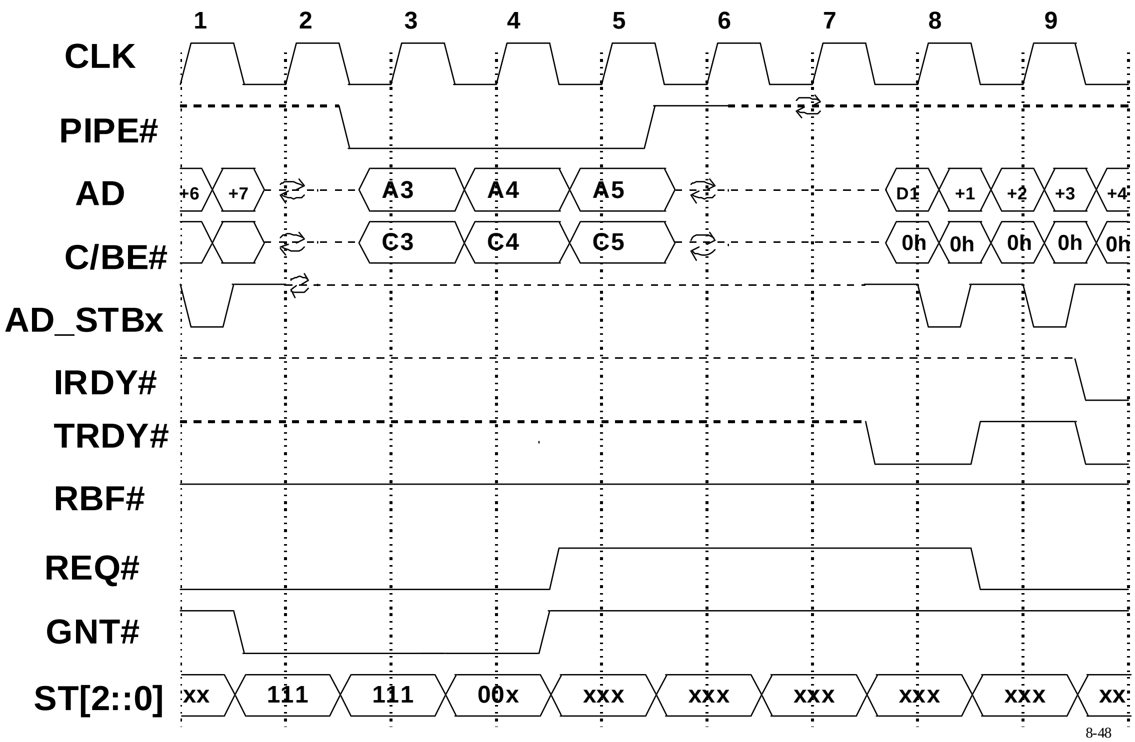


Figure 8-48 Intervened Request, Subsequent 2x Read Data Delayed

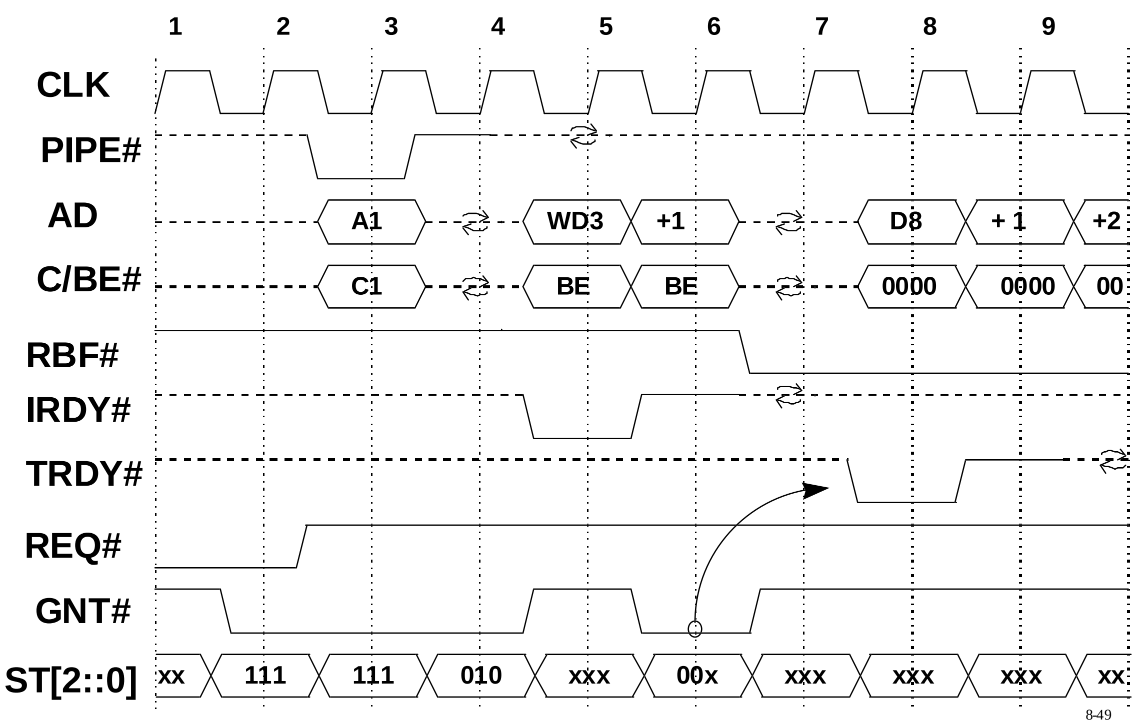


Figure 8-49 Request followed by Write Data, followed by Read Data

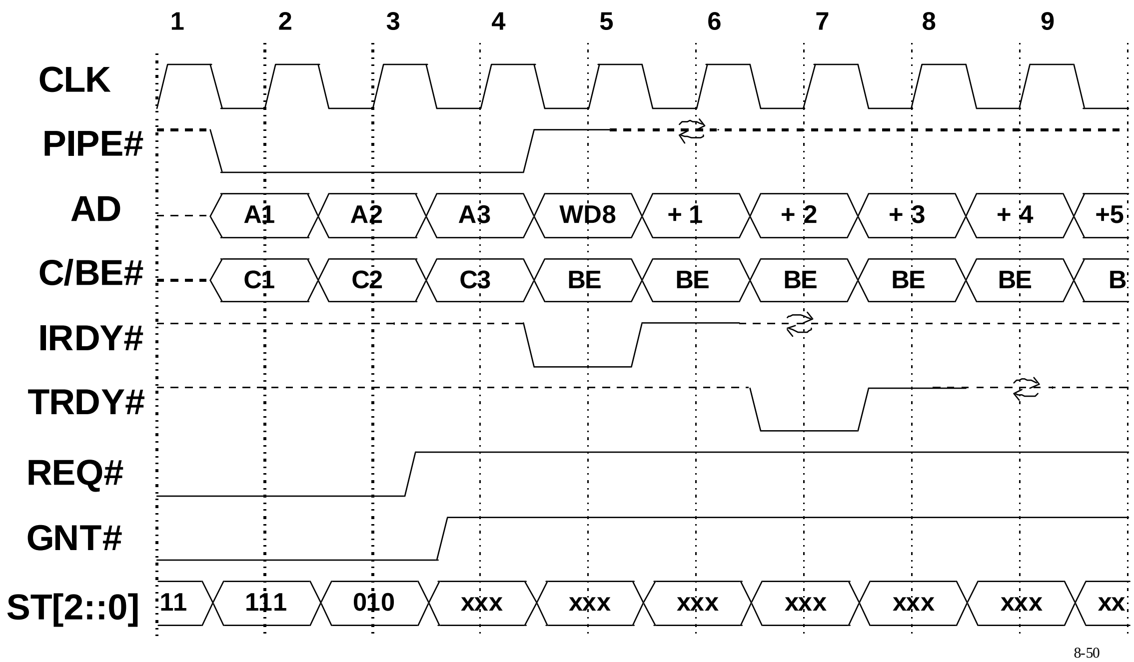


Figure 8-50 Multiple Requests followed by Write Data - No Turn-Around Cycle

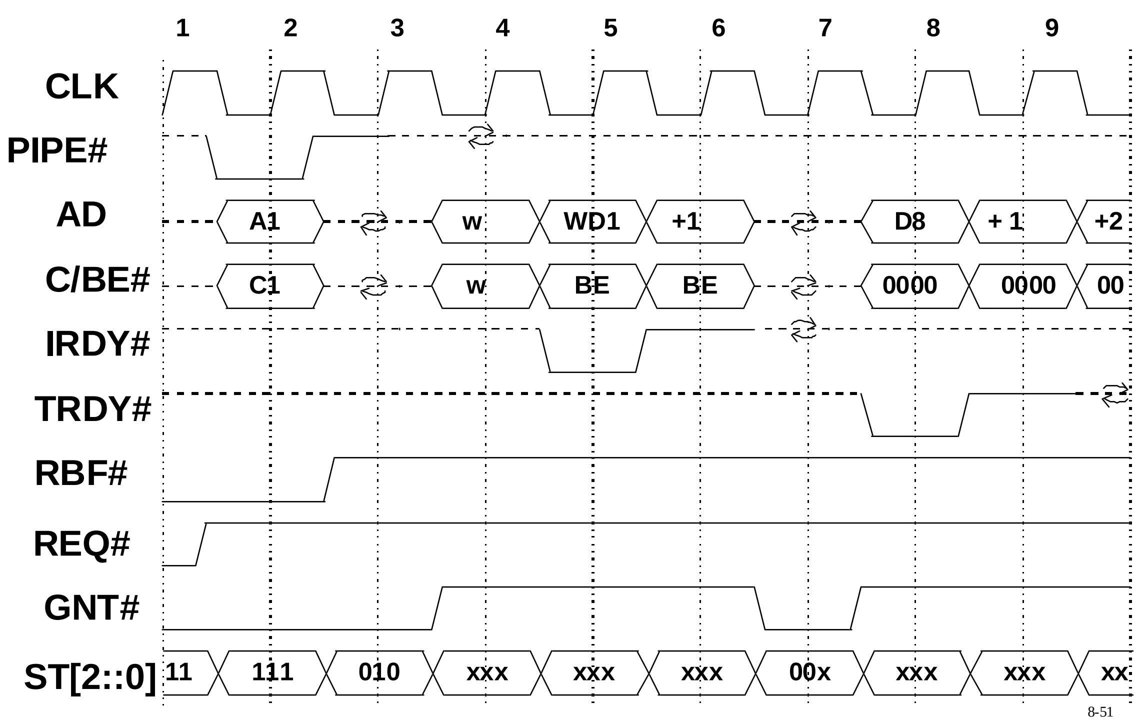


Figure 8-51 Request followed by Write Data Delayed, followed by Read Data - No Delay

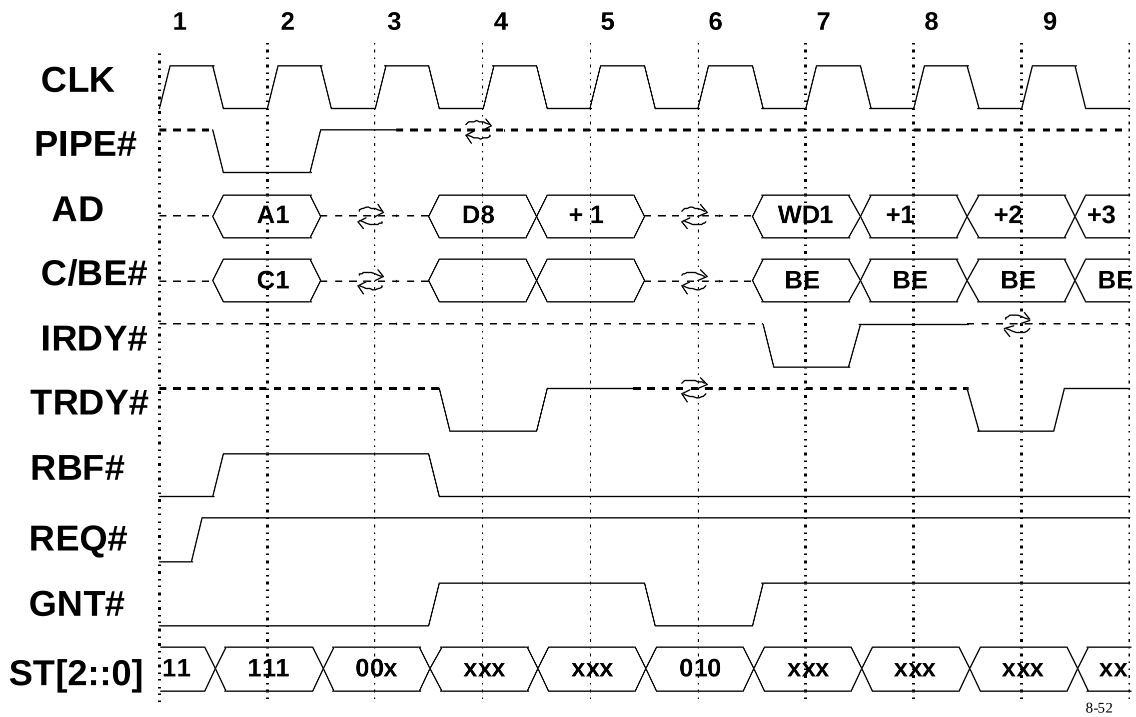


Figure 8-52 Request followed by Read Data, followed by Write Data

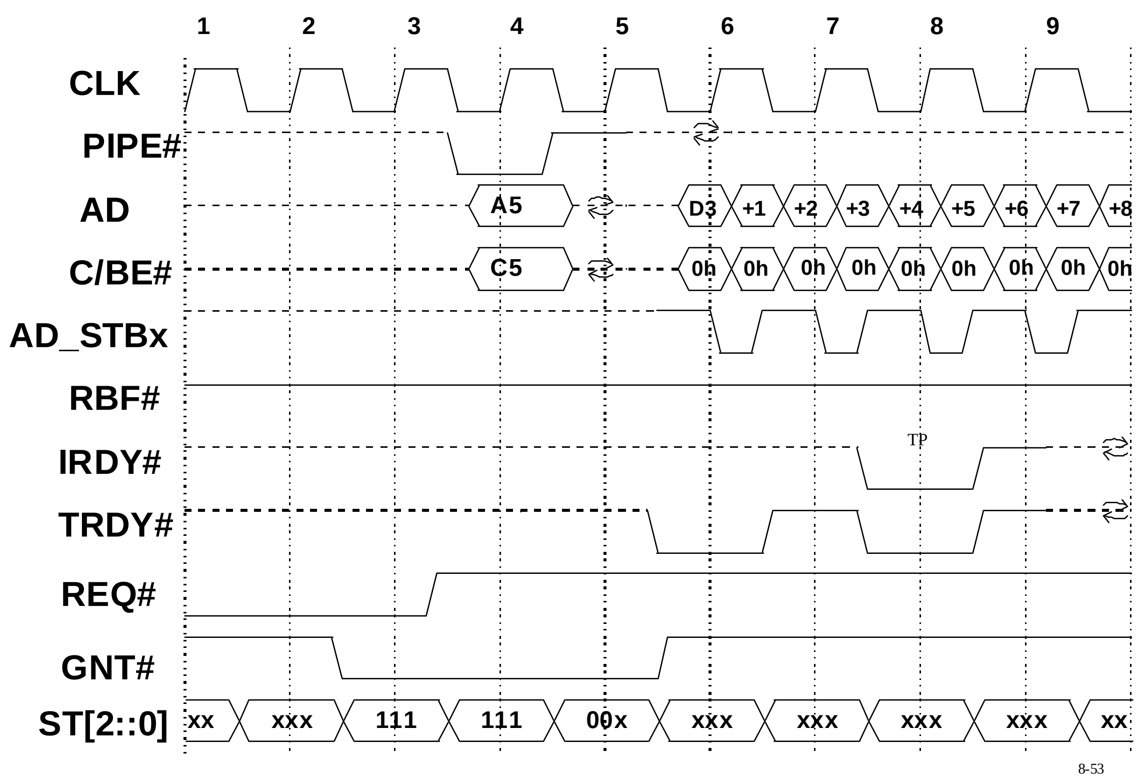


Figure 8-53 Request followed by 2x Read Data No Delay

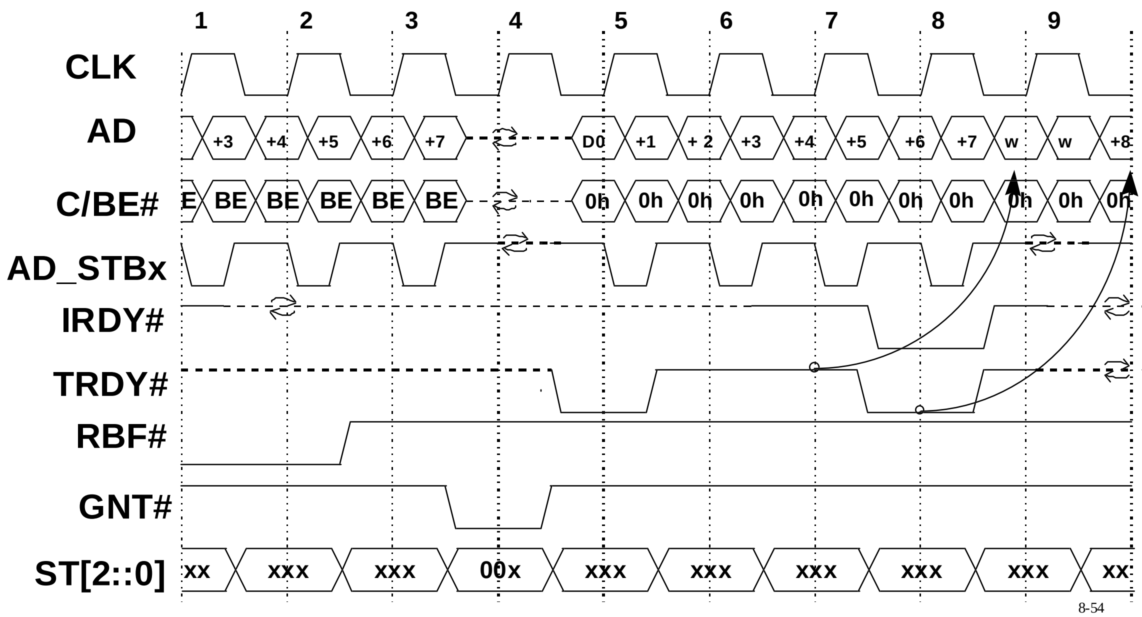


Figure 8-54 2x Write followed by 2x Read with Subsequent Block Delayed

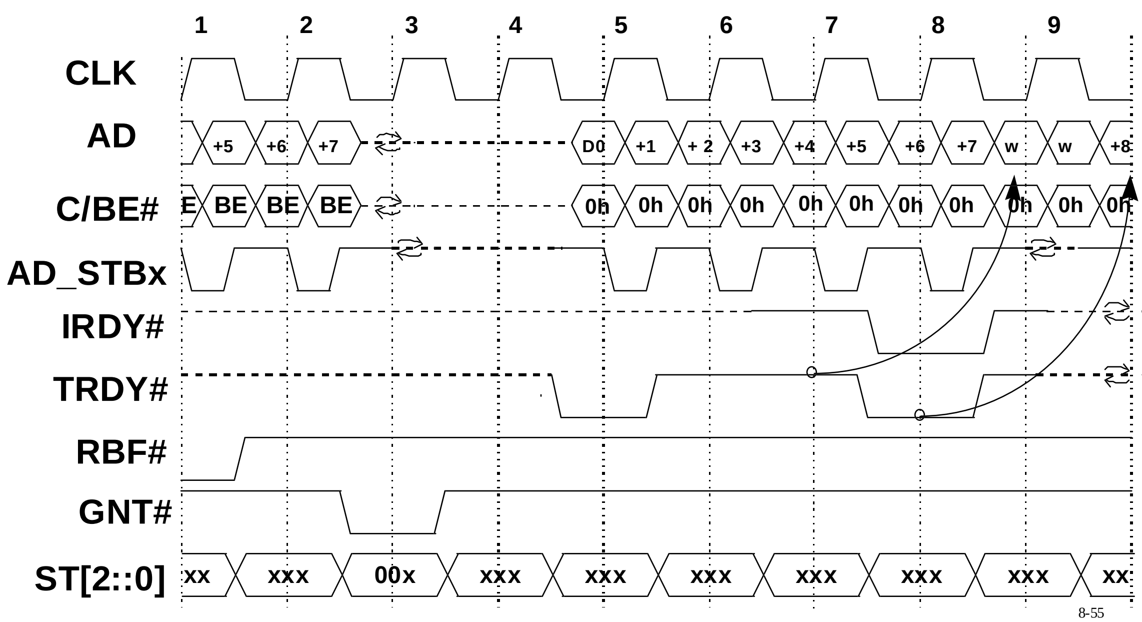


Figure 8-55 2x Write followed by 2x Read with Initial and Subsequent Blocks Delayed

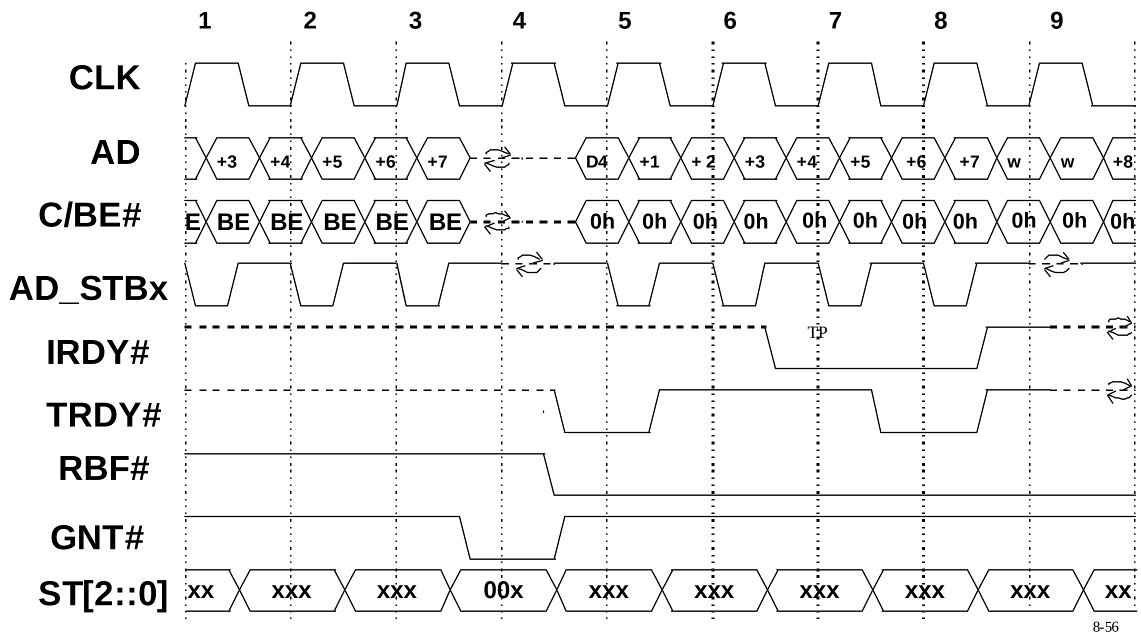


Figure 8-56 2x Write, 2x Read No Initial Delay, Subsequent Block Delayed by Target

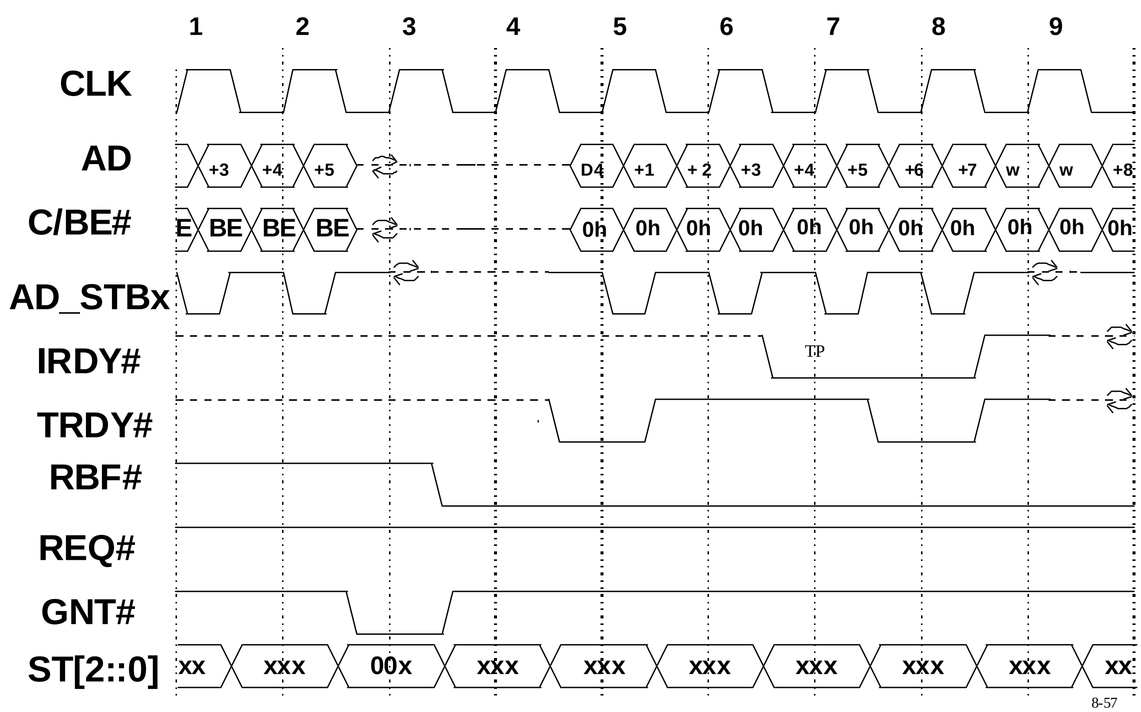


Figure 8-57 2x Write followed by 2x Read, Initial and Subsequent Block Delayed by Target

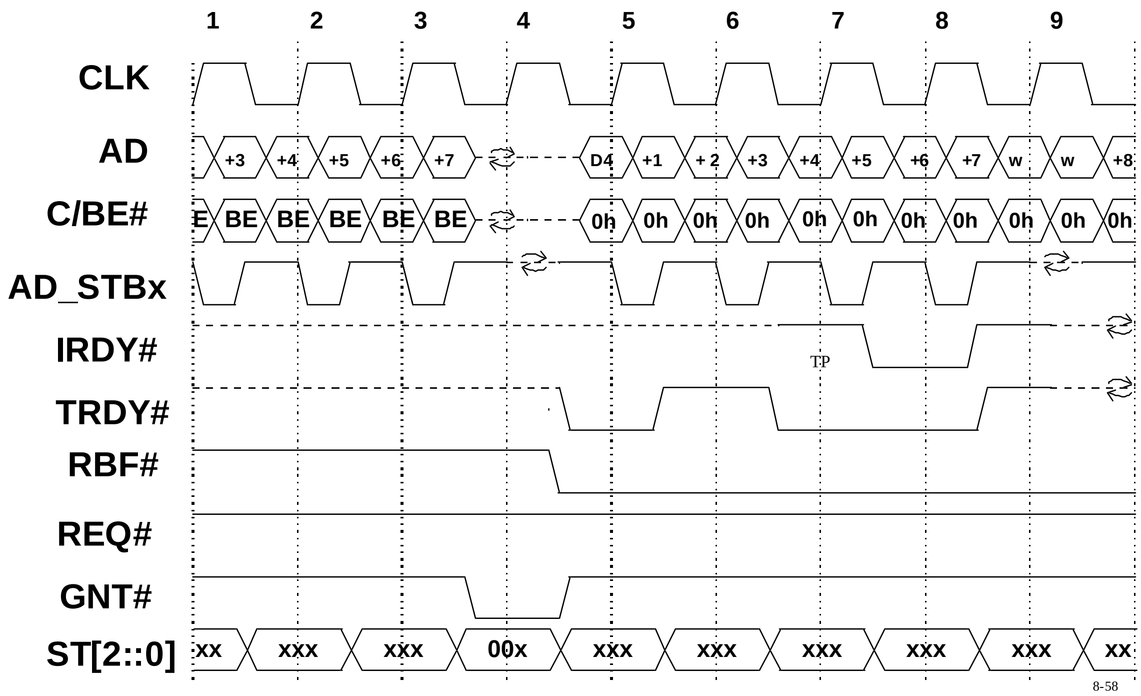


Figure 8-58 2x Write followed by 2x Read, Master Delays Subsequent Block

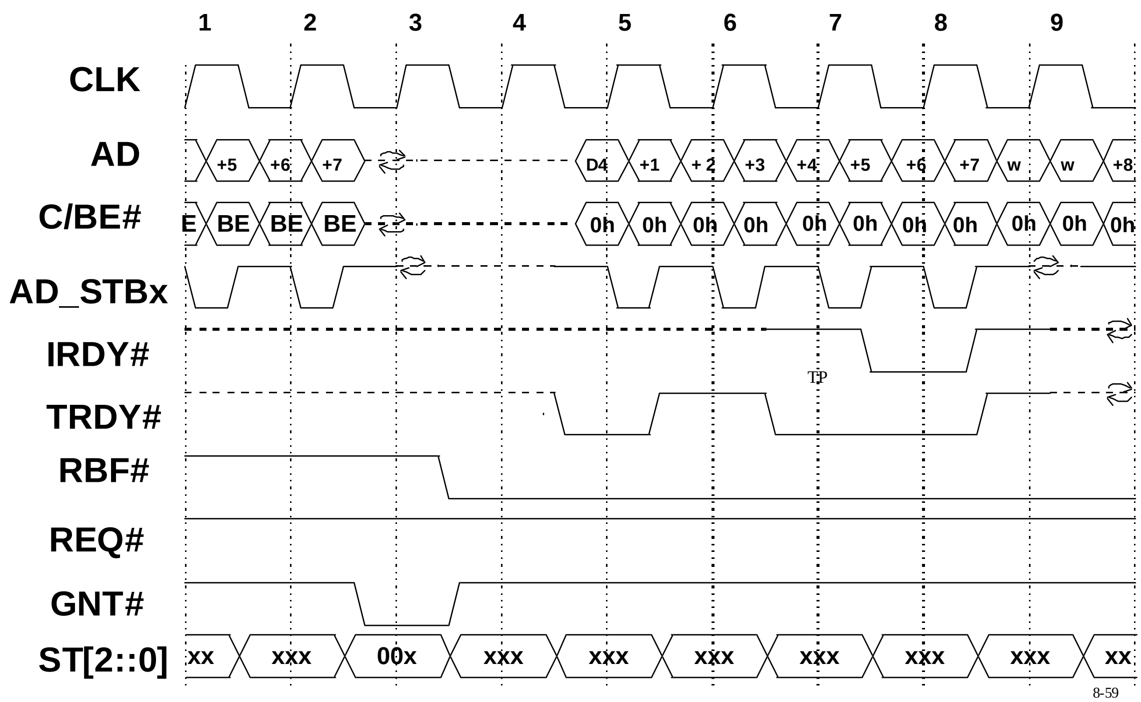


Figure 8-59 2x Write followed by 2x Read Initial and Subsequent Blocks delayed

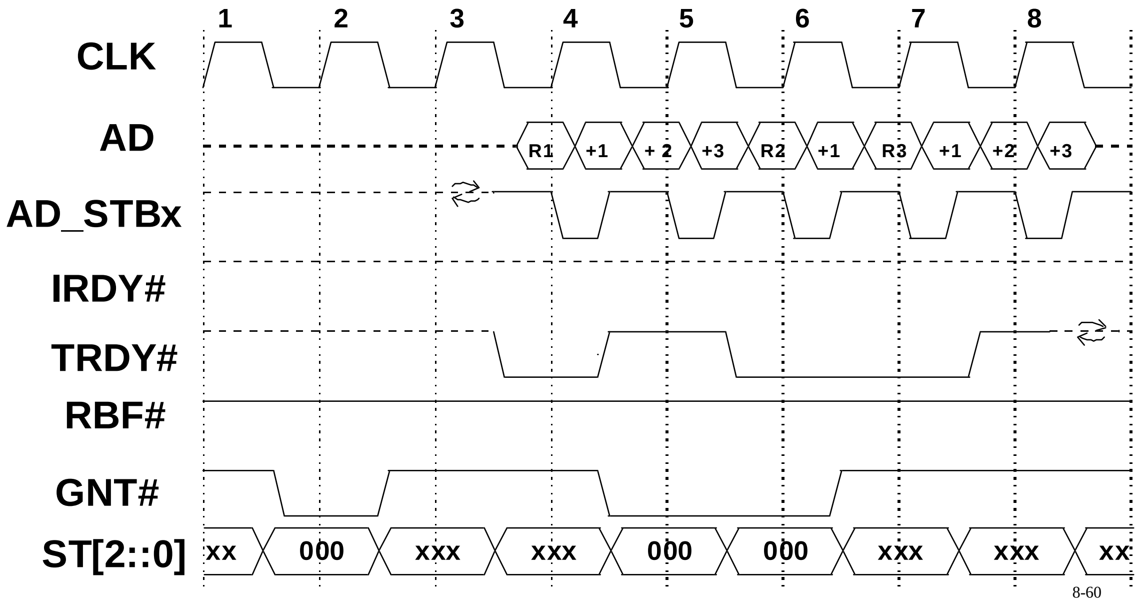


Figure 8-60 2x Reads Back to Back Different Lengths

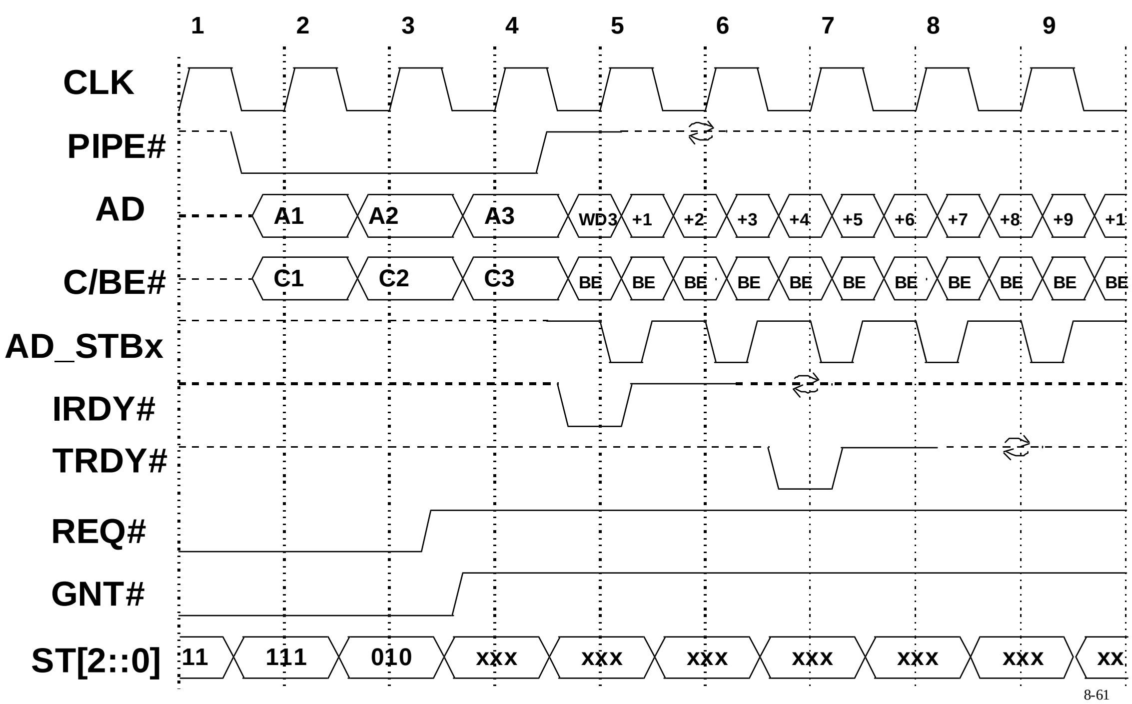


Figure 8-61 Multiple Requests followed by 2x Write No Delay Initial or Subsequent Blocks

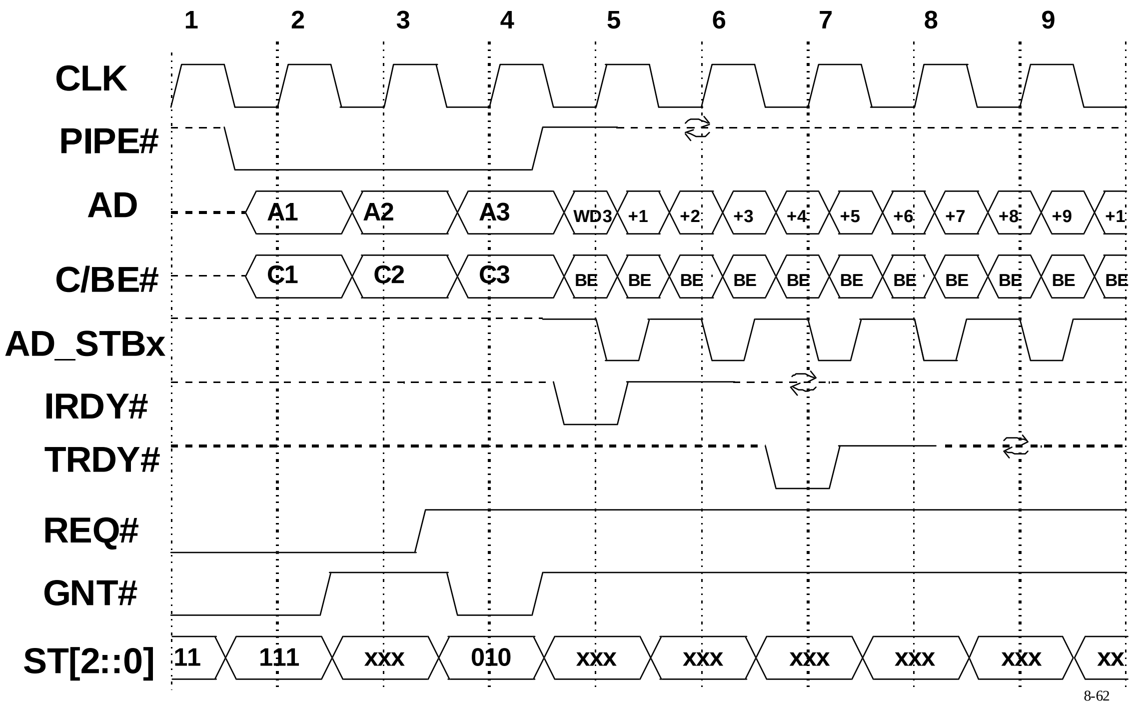


Figure 8-62 Requests followed by 2x Write Early GNT# no Delay

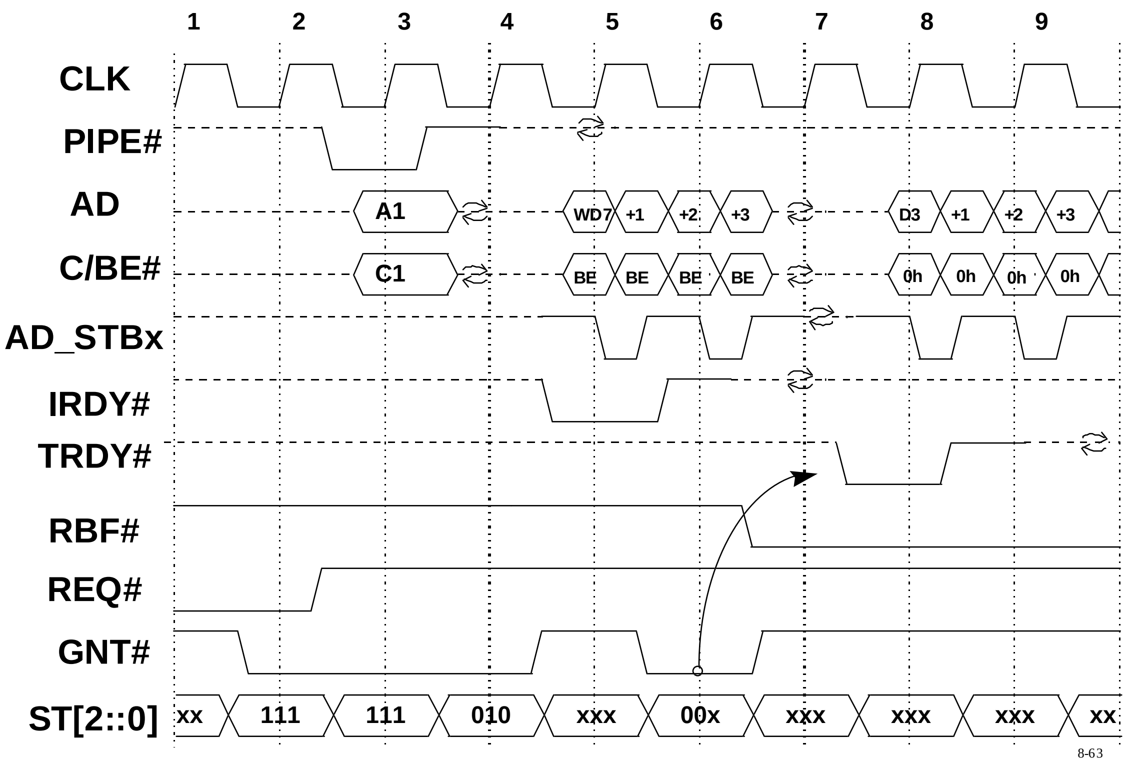


Figure 8-63 Single Request, 2x Write, 2x Read Early GNT#

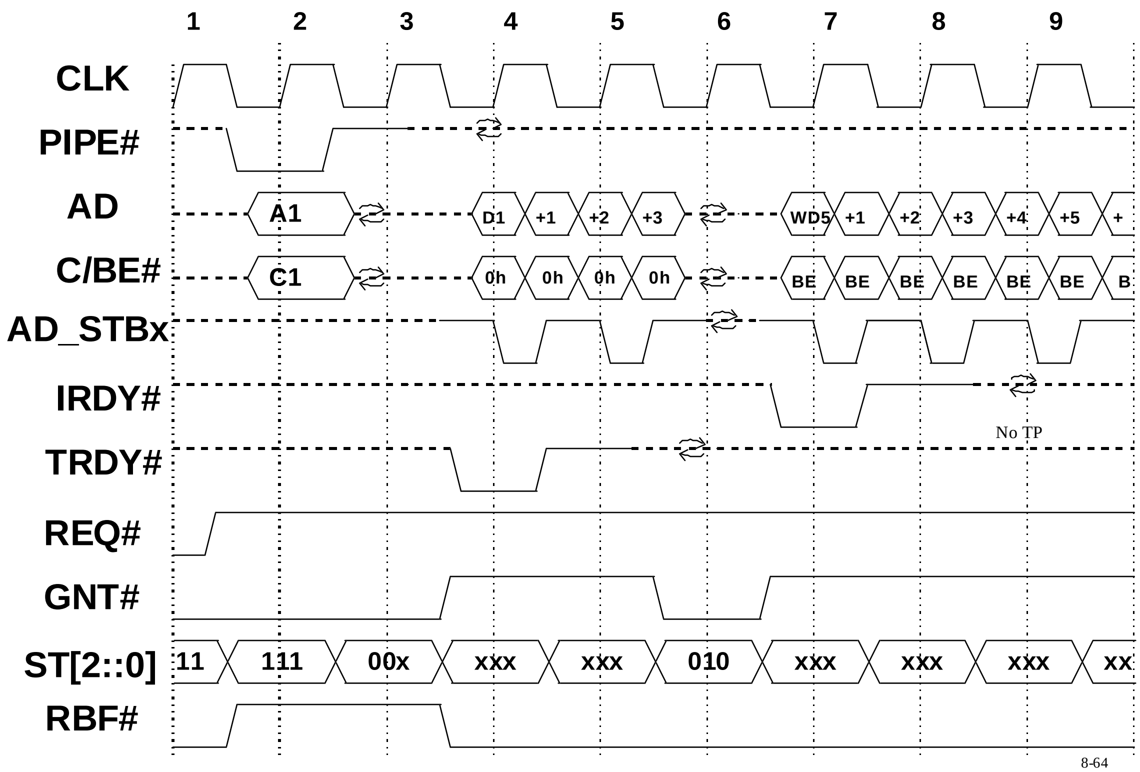


Figure 8-64 Request followed by 2x read, followed 2x Write Late GNT#

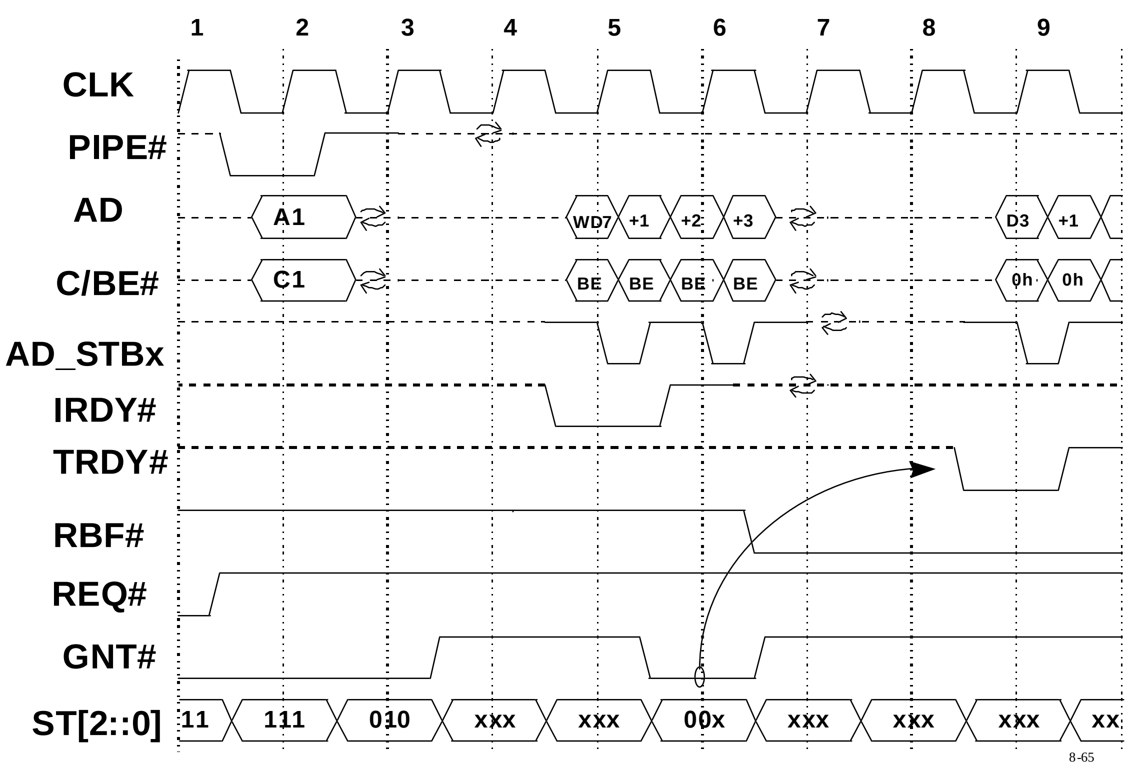


Figure 8-65 Single Request followed by 2x Write and 2x Read with Delays Early GNT#

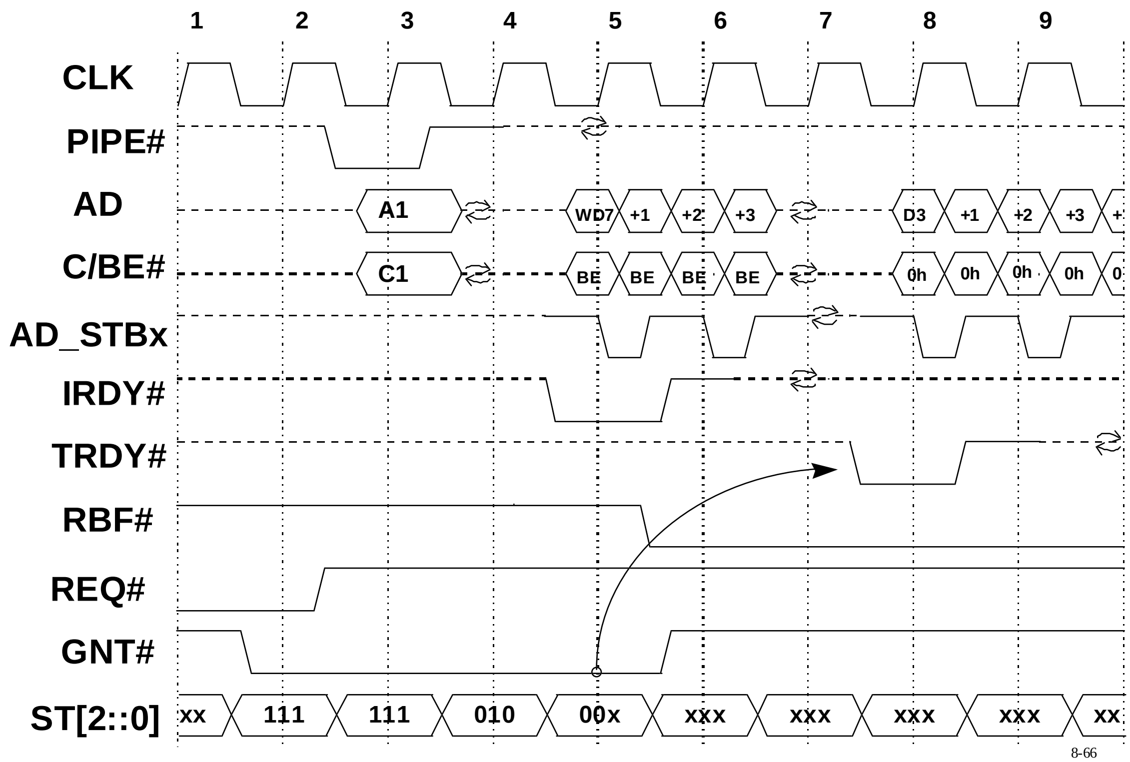


Figure 8-66 Single Request followed by 2x Write and 2x Read, Early GNT#

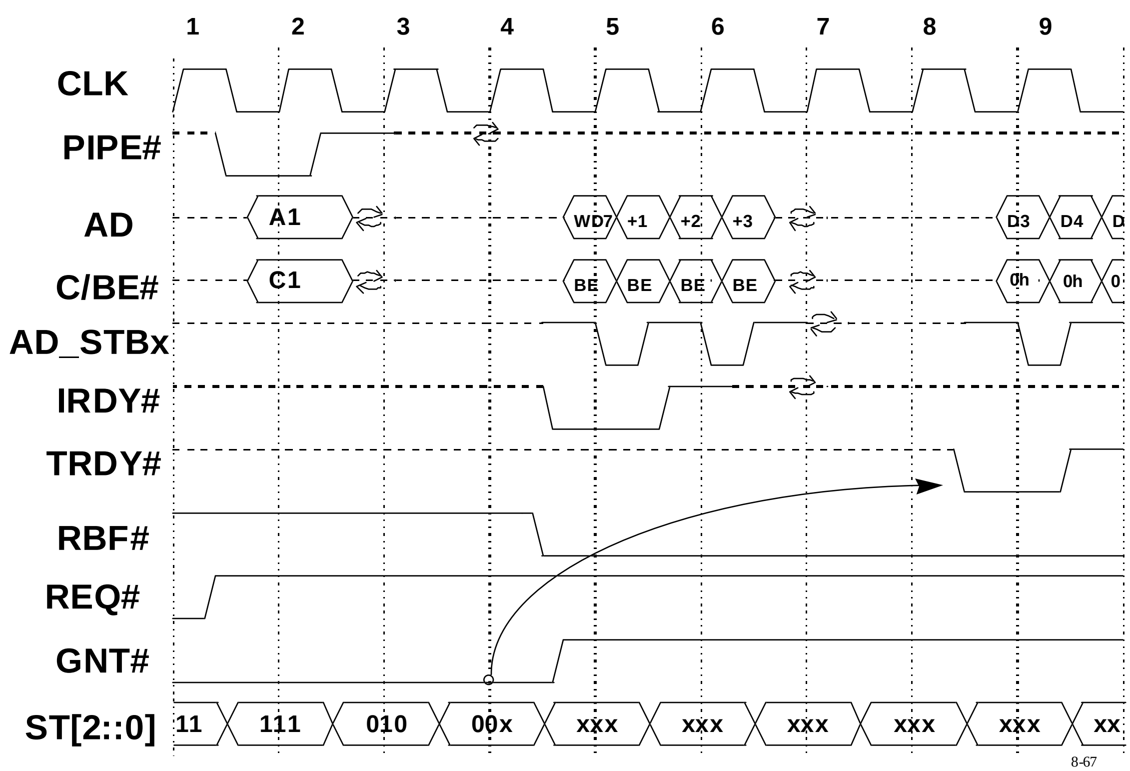


Figure 8-67 Single Request, Early GNT# for 2x Write (delay) followed by 2x Read (delay)

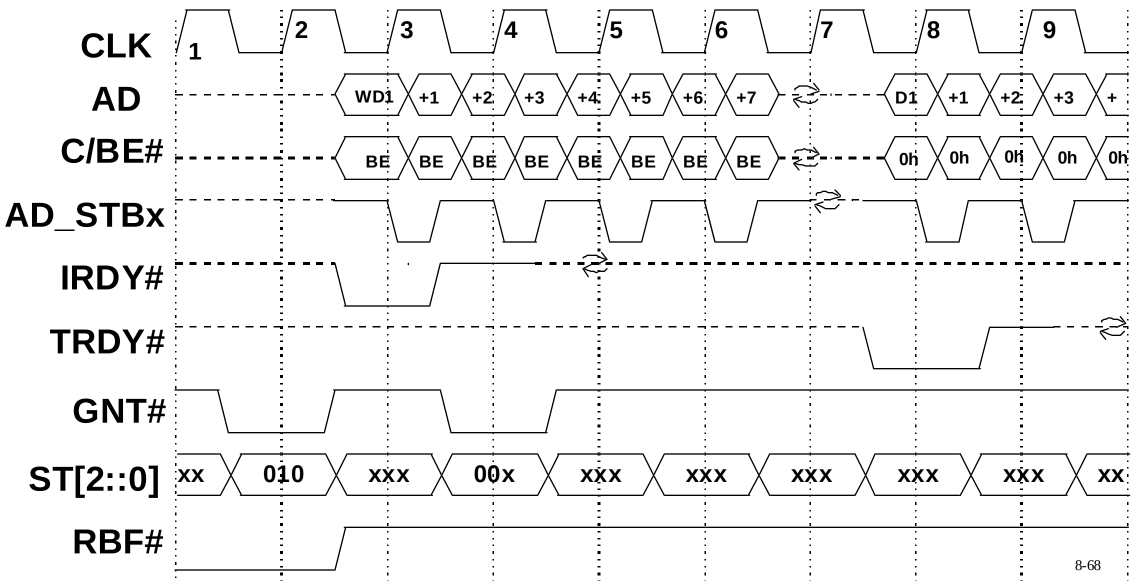


Figure 8-68 2x Write followed by 2x Read, GNT# Delayed 1 Clock

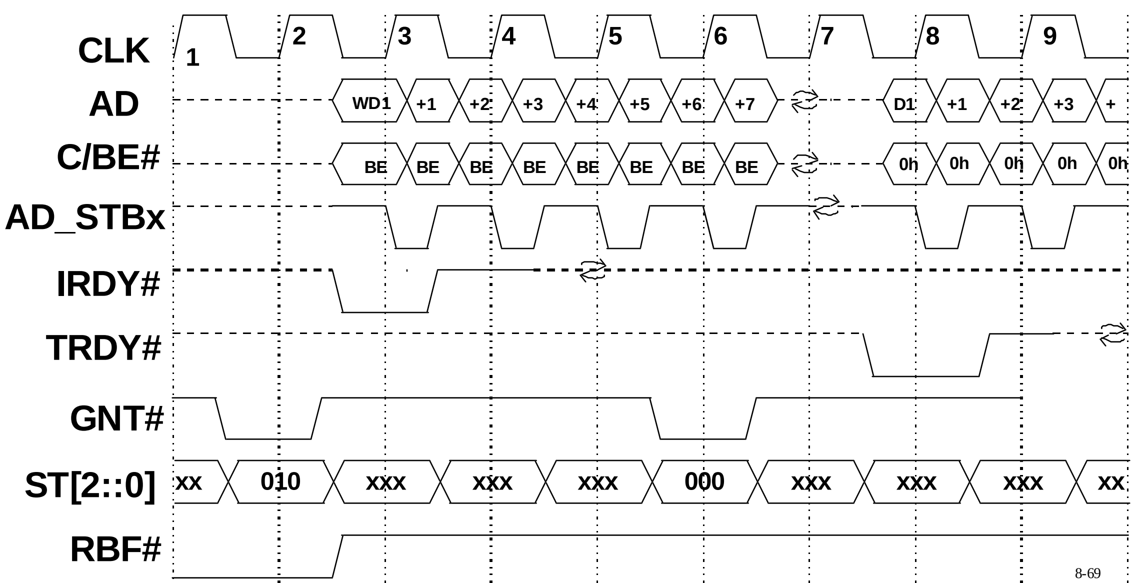


Figure 8-69 2x Write followed by 2x Read, GNT# No Delay

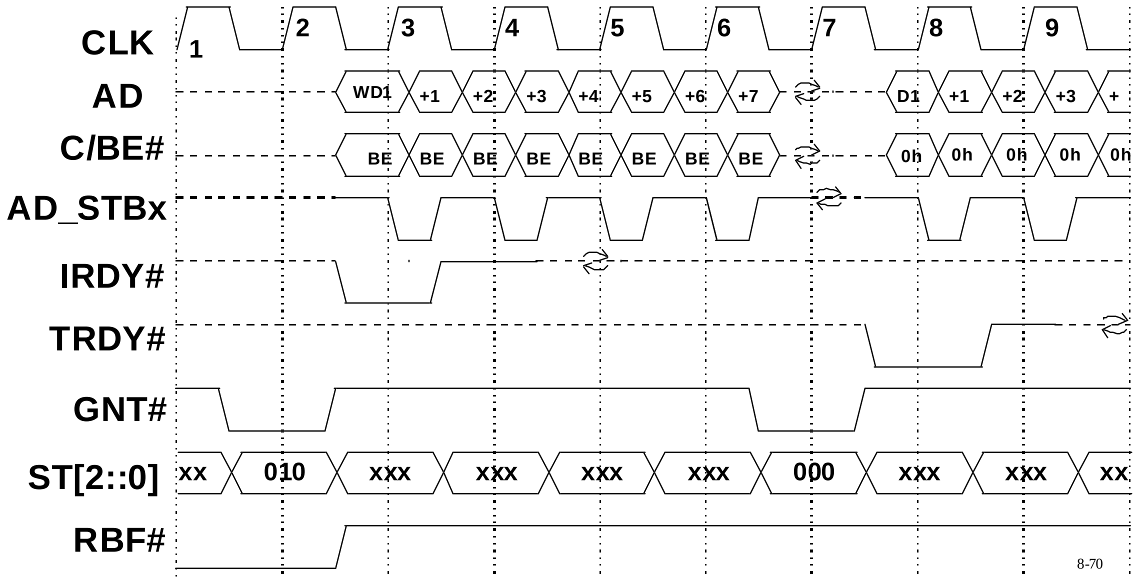


Figure 8-70 2x Write followed by 2x Read, GNT# Delayed Maximum with no Dead Clock

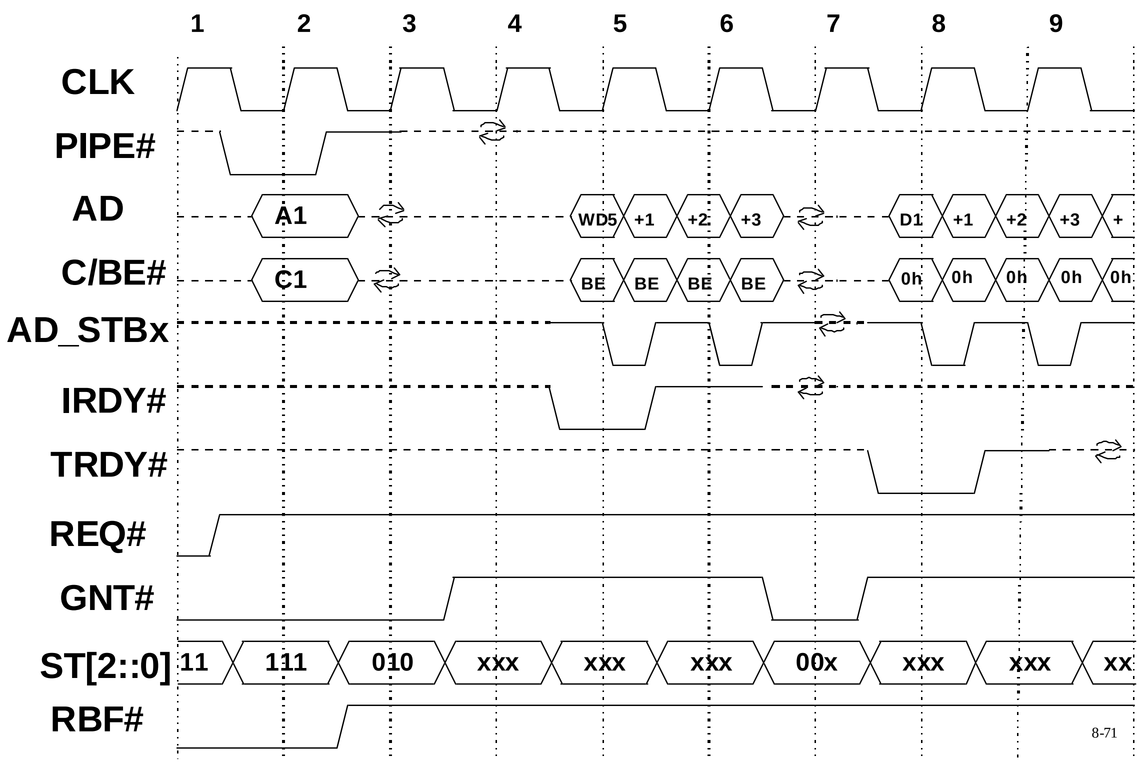


Figure 8-71 Single Request, Maximum 2x Write Delay, 2x Read Late GNT#

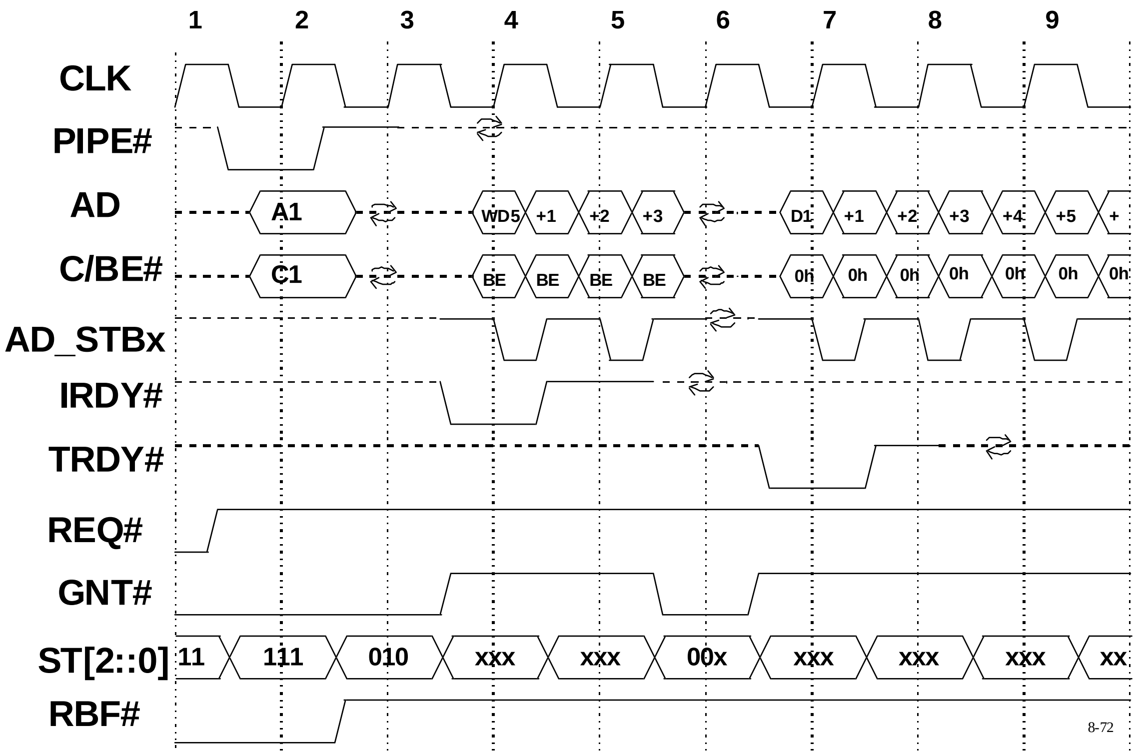


Figure 8-72 Single Request, 2x Write, 2x Read Late GNT#

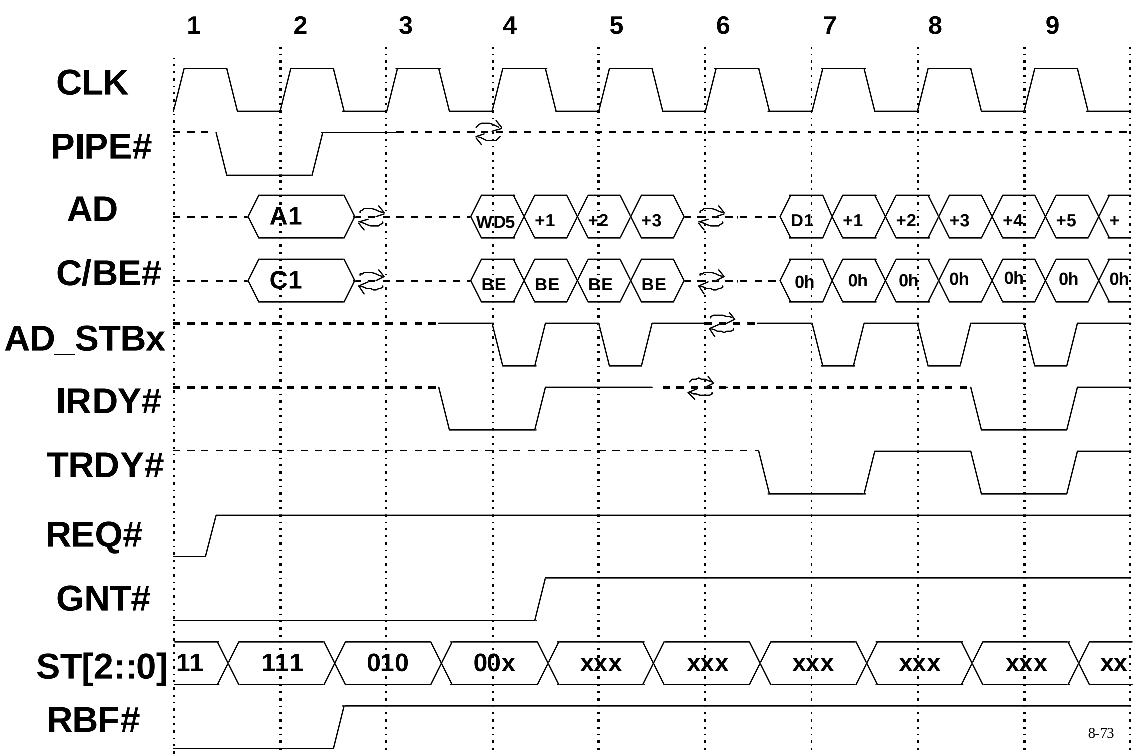


Figure 8-73 Single Request, 2x Write, 2x Read Earliest GNT#s

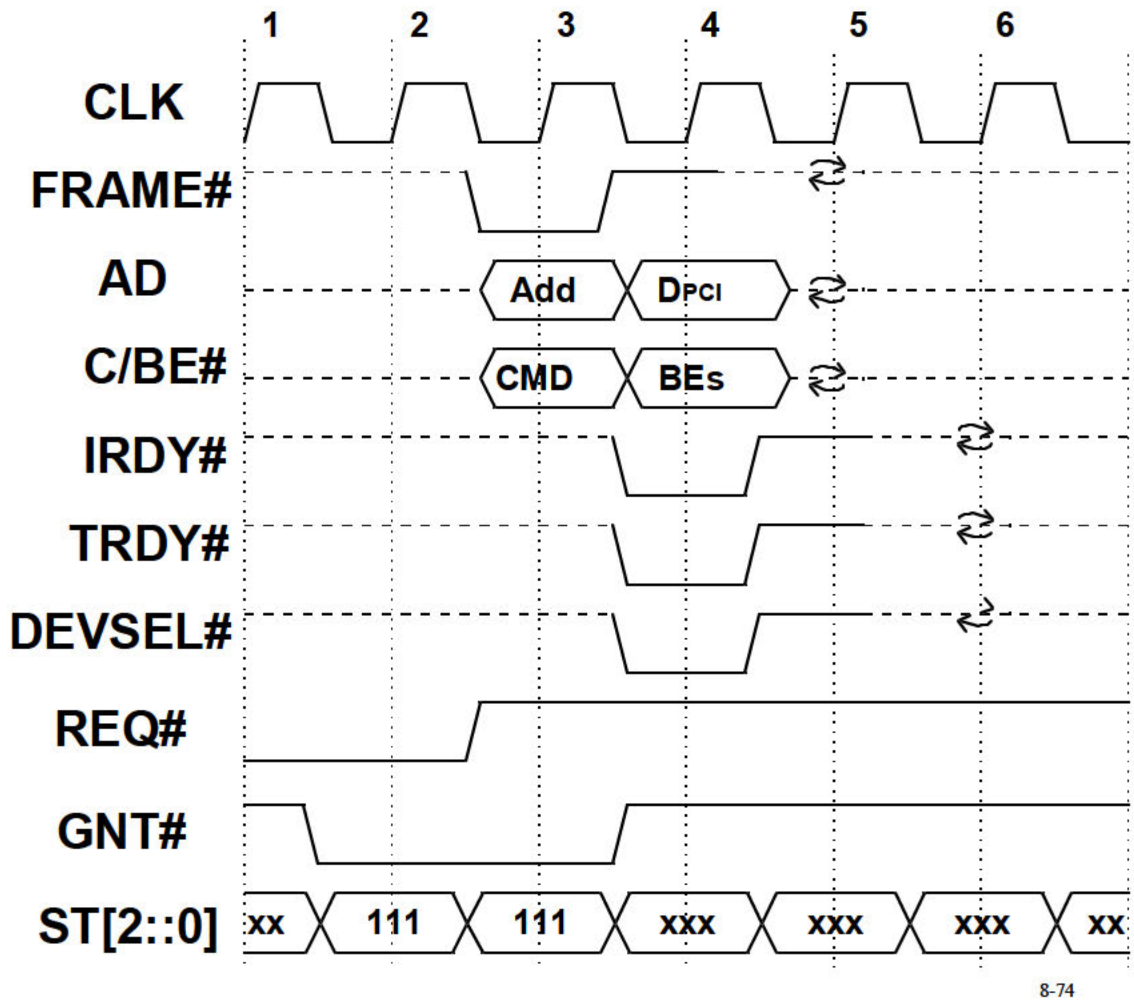


Figure 8-74 PCI Write Transaction on A.G.P.

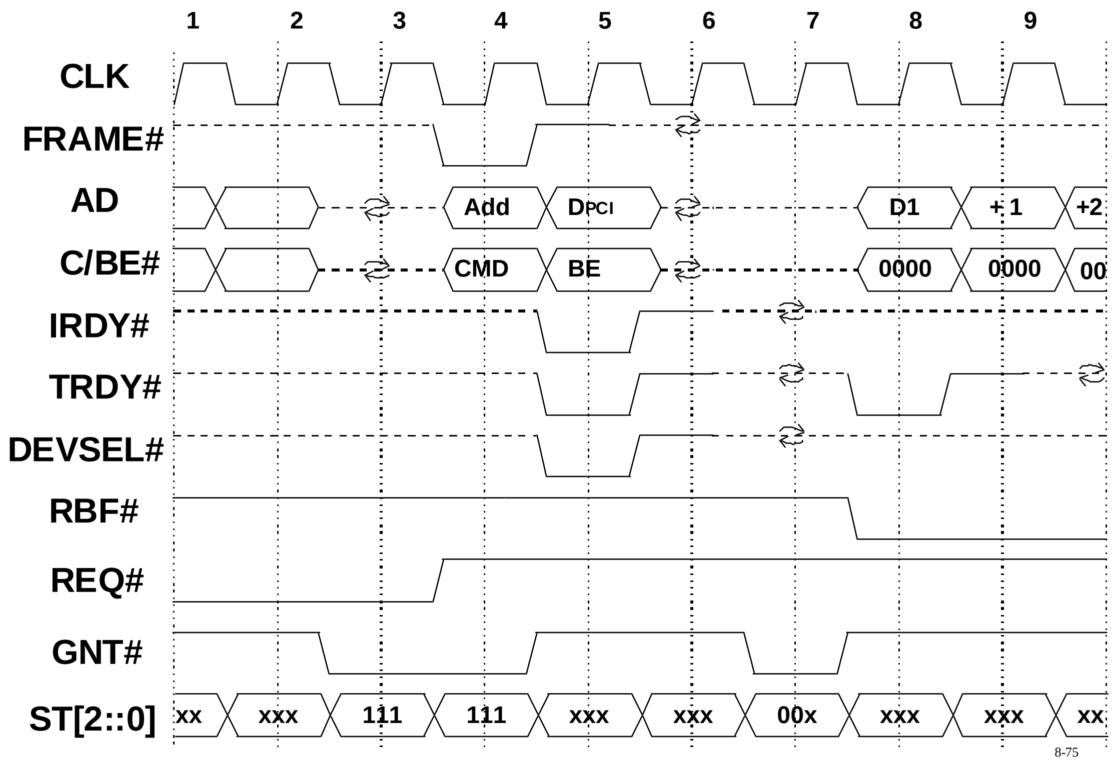


Figure 8-75 PCI Transaction Between 2 A.G.P. Data Transfers

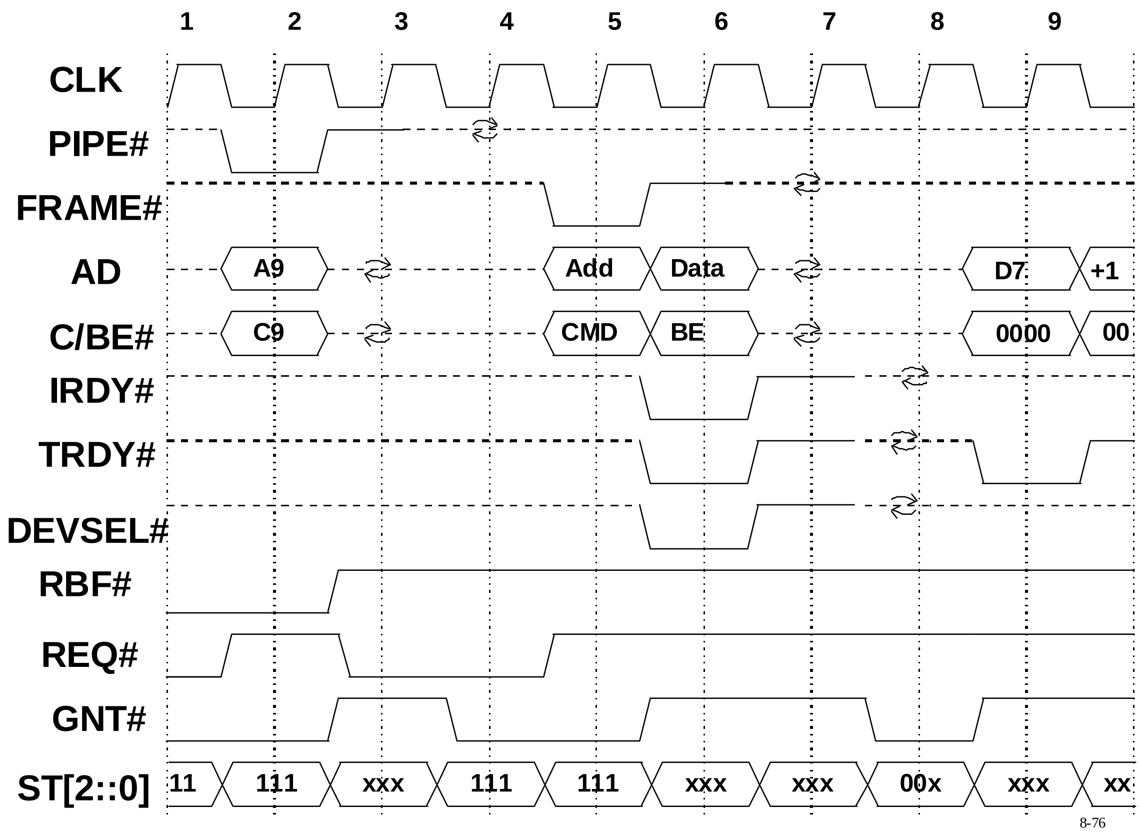


Figure 8-76 Single Request, PCI Transaction followed by Read Data (No Delays)

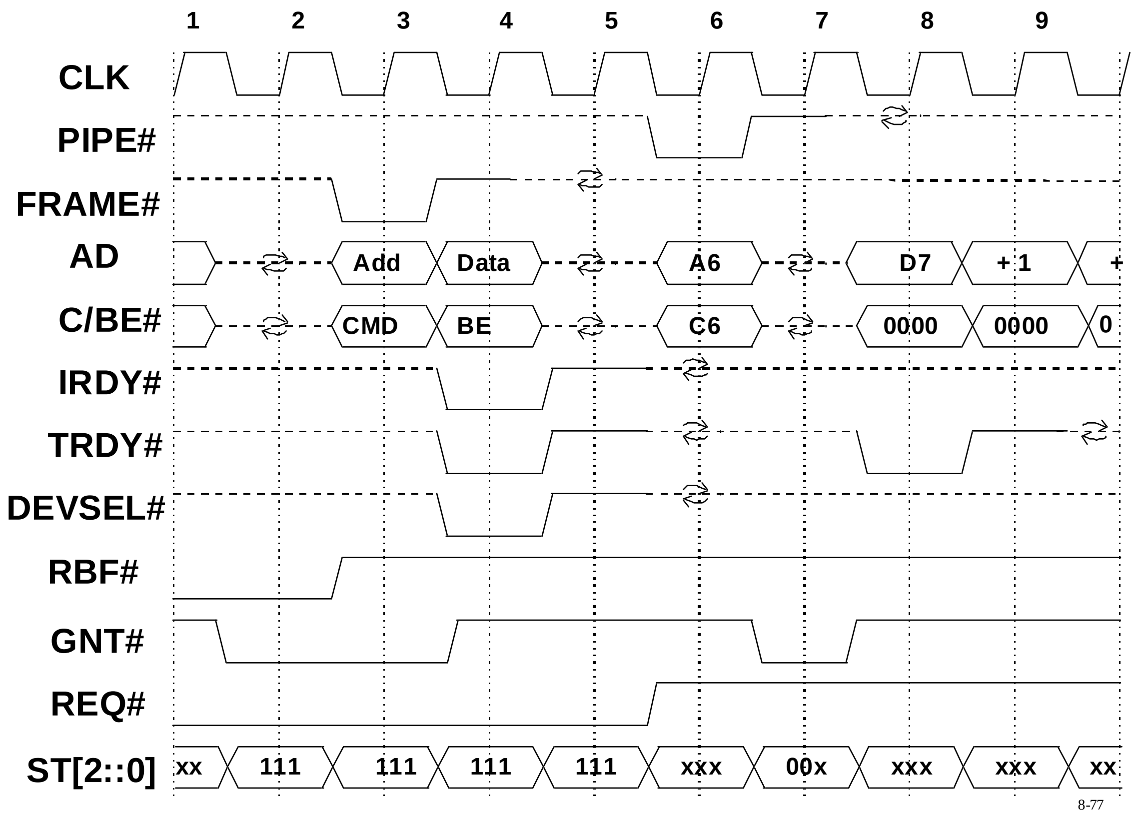


Figure 8-77 PCI Transaction followed by A.G.P. Request, Read Data (No Delays)

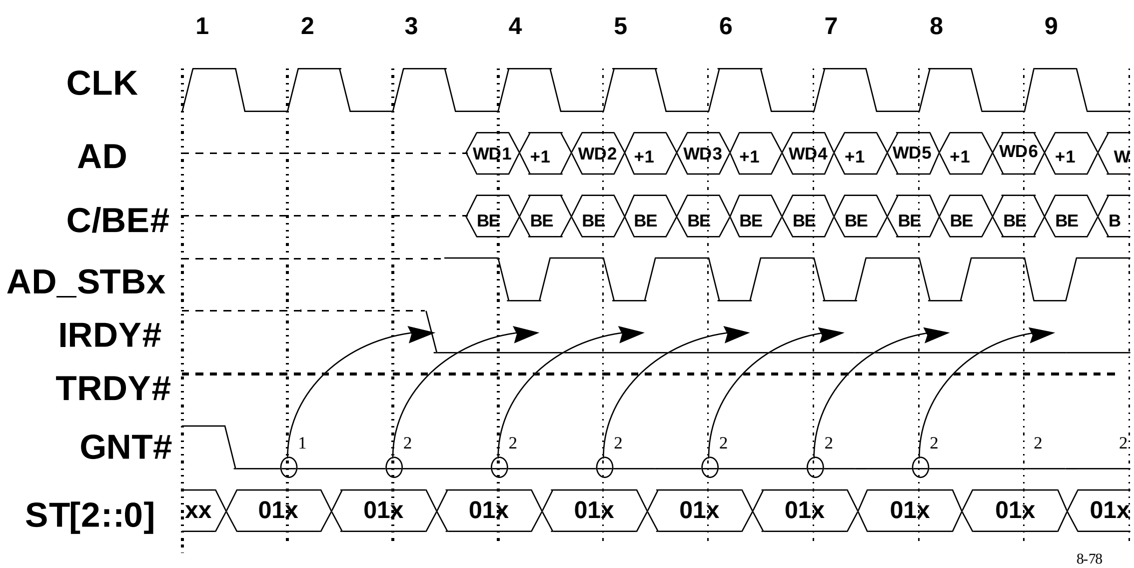


Figure 8-78 BtoB 8 Byte Writes, Initial Write with Max Delay, Subsequent Writes No Delays

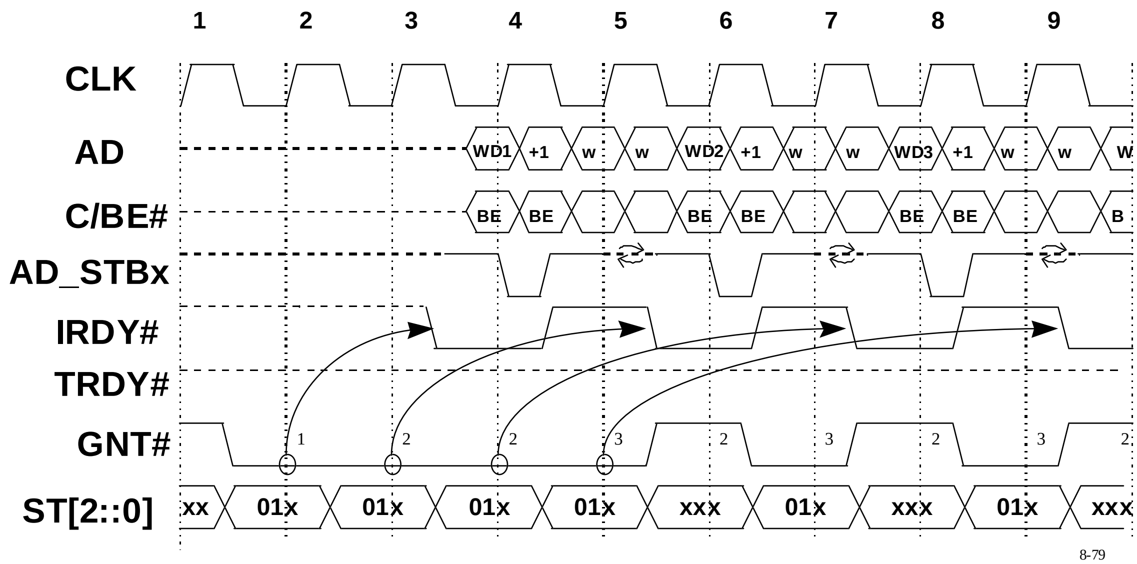


Figure 8-79 8 Byte Write, Initial Max Delay, Subsequent Max Delay (Part 1)

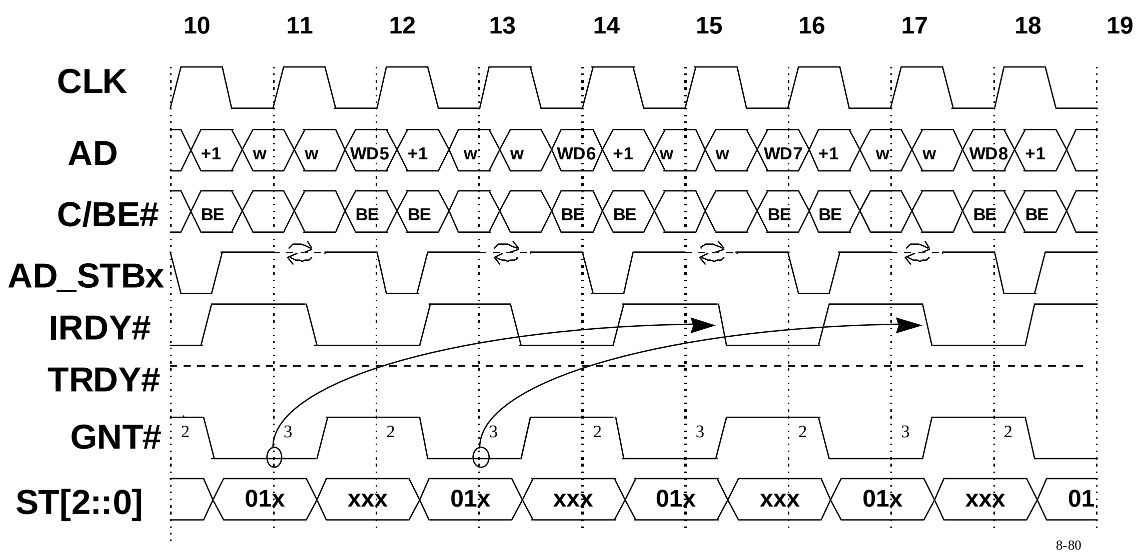


Figure 8-80 8 Byte Write, Initial Max Delay, Subsequent Max Delay (Part 2)

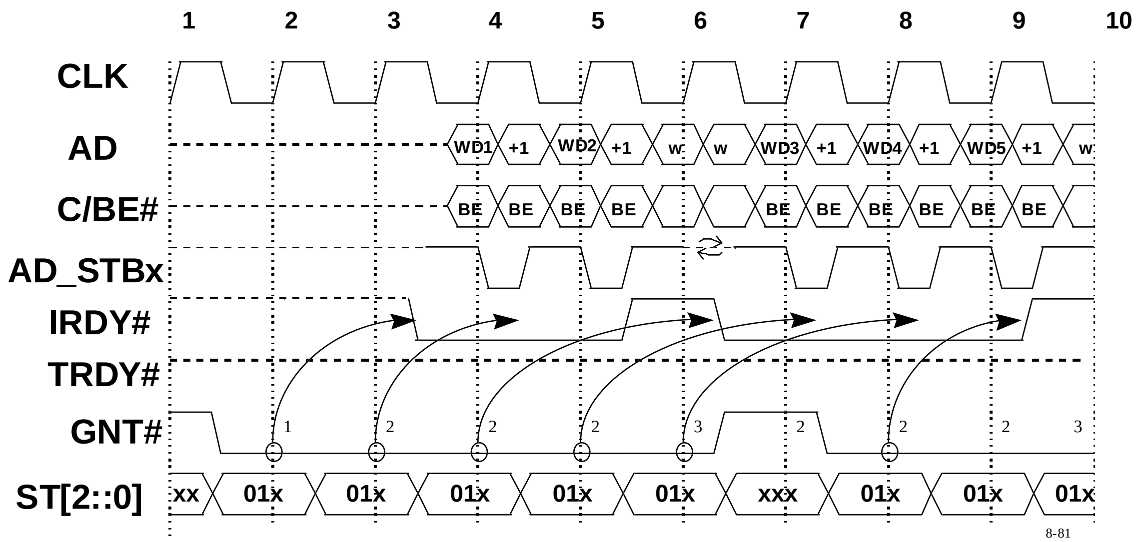


Figure 8-81 BtoB 8 Byte Writes, Initial Write with Delay, Subsequent Writes Delays (Part 1)

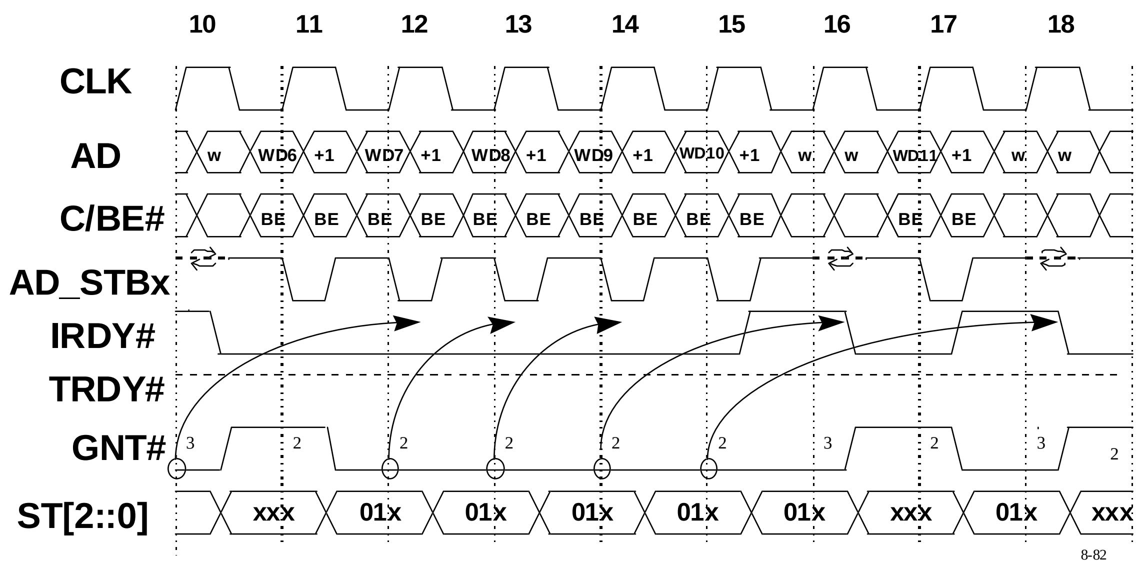


Figure 8-82 BtoB 8 Byte Writes, Initial Write with Delay, Subsequent Writes Delays(Part 2)

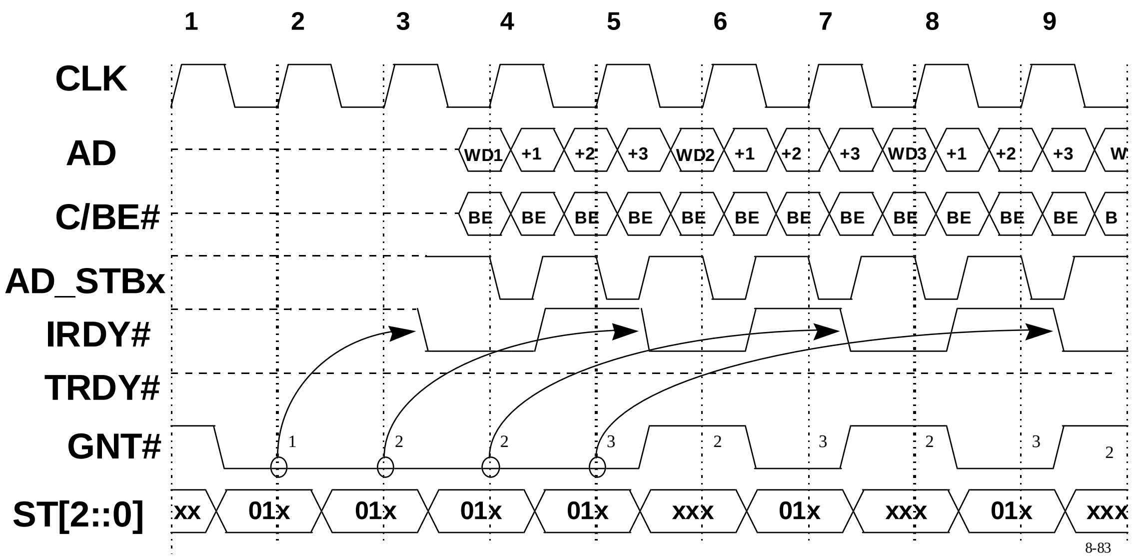


Figure 8-83 16 Byte Writes, Initial Max Delay, No Subsequent Delay

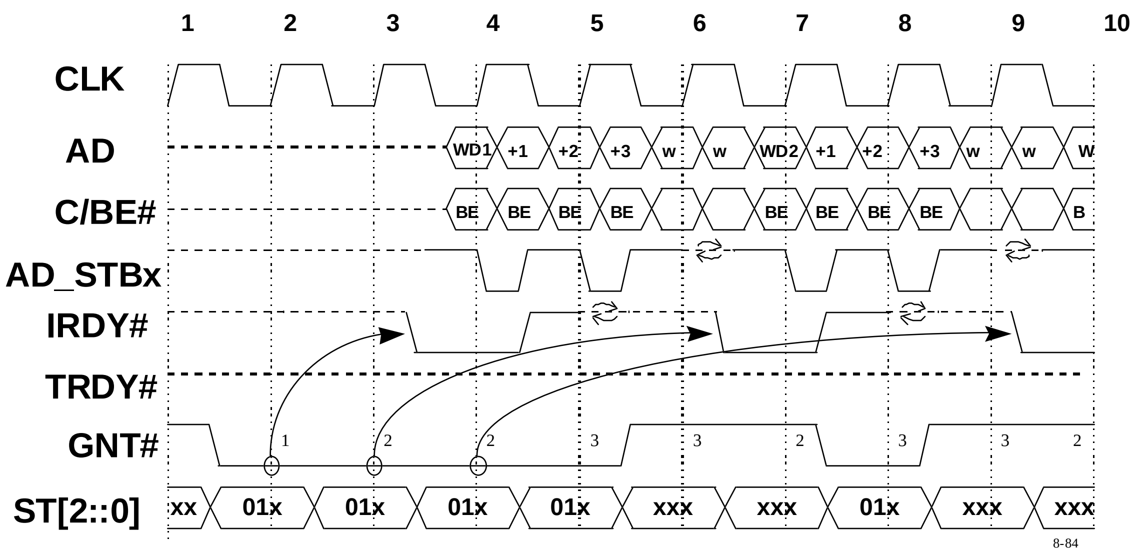


Figure 8-84 16 Byte Writes, Initial Max Delay, Subsequent Max Delays

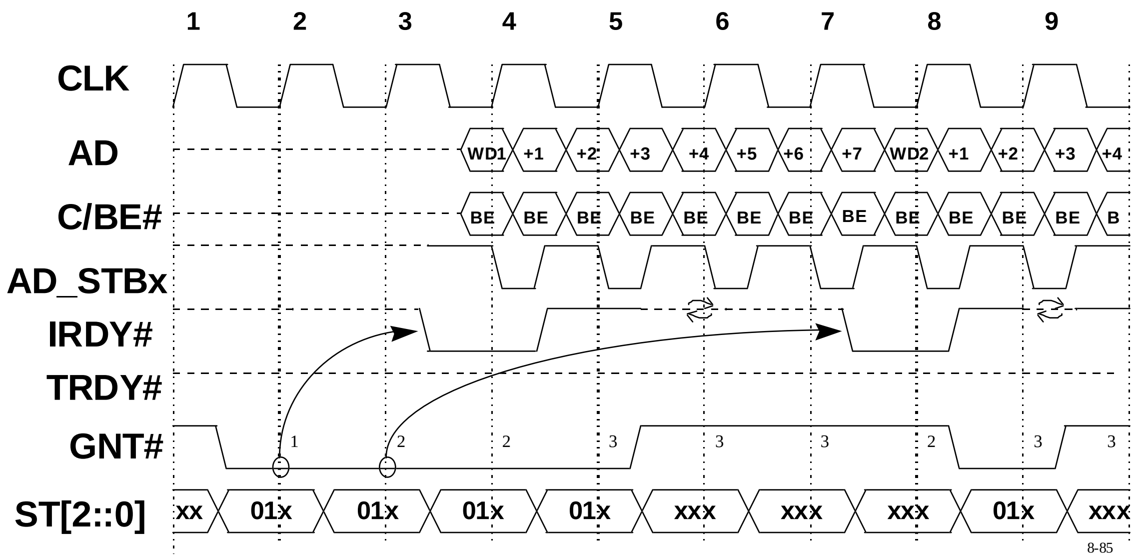


Figure 8-85 32 Byte Writes, Initial Max Delay, No Subsequent Delay

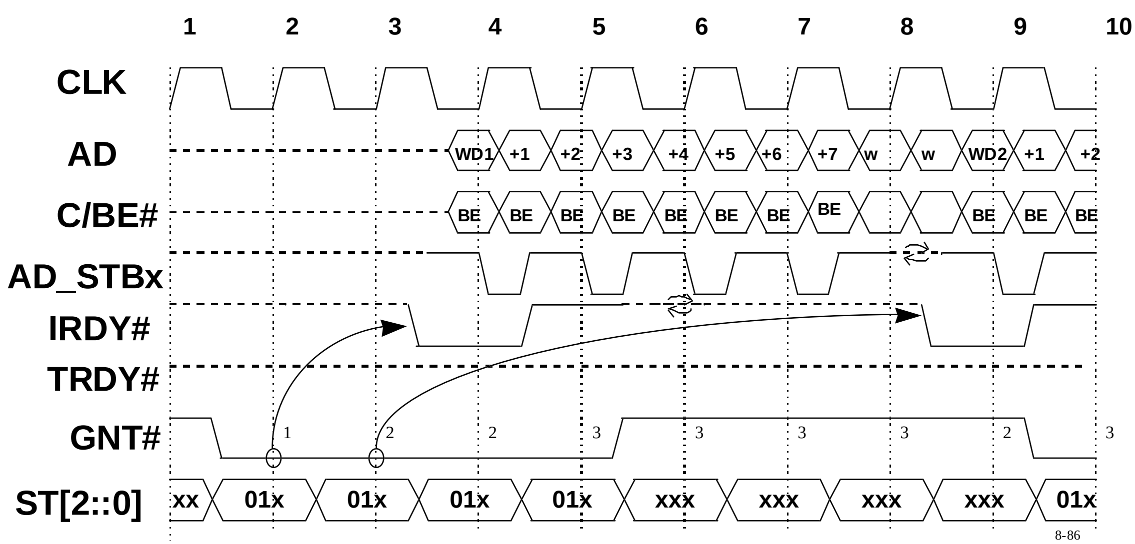


Figure 8-86 32 Byte Writes, Initial Max Delay, Subsequent Max Delays

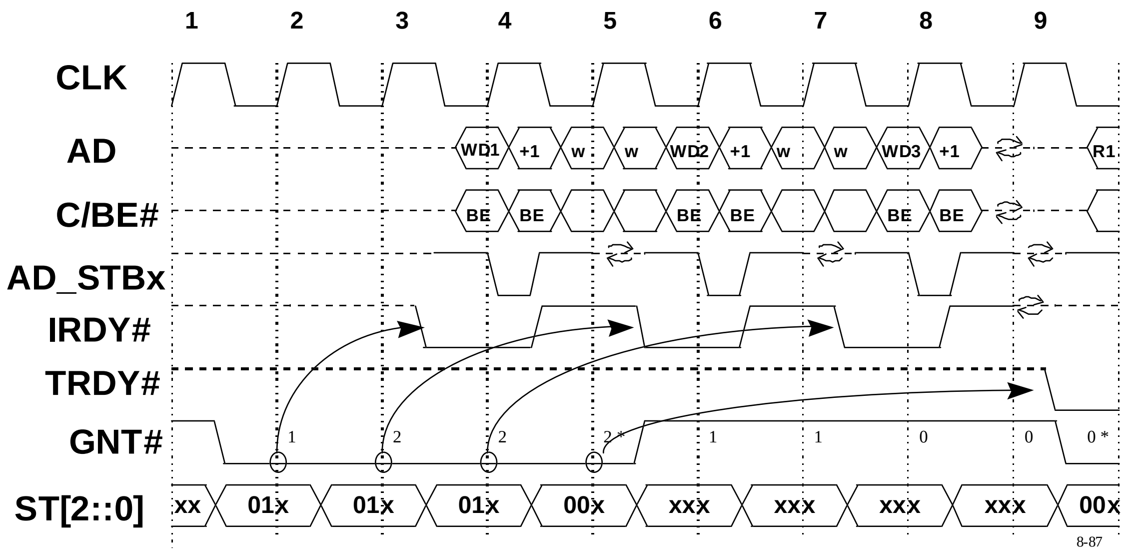


Figure 8-87 Three 8 Byte Writes, 8 Byte Read, 40 Byte Read, Three 8 Byte Writes (Part 1)

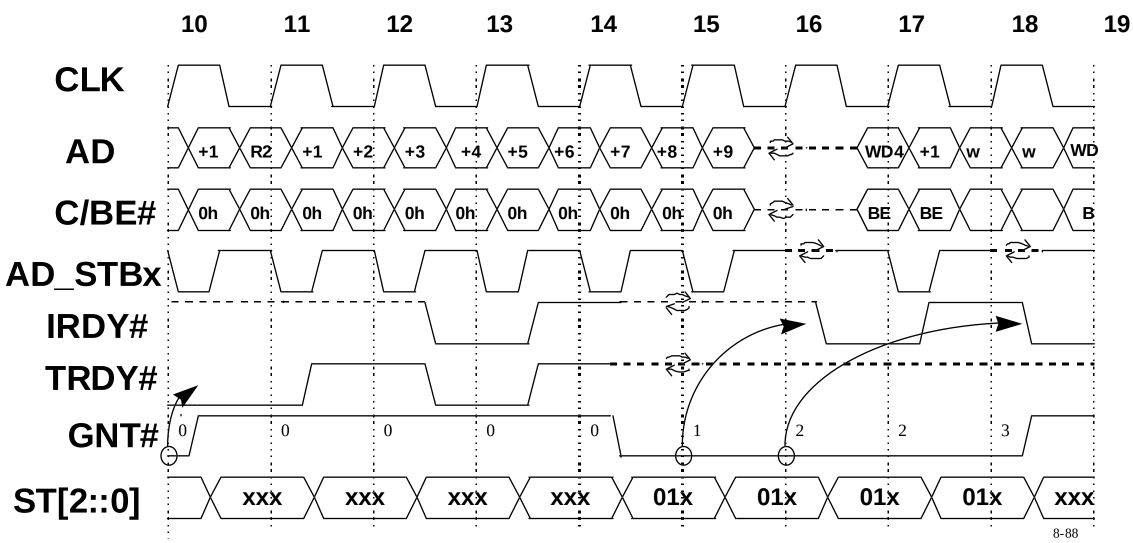


Figure 8-88 Three 8 Byte Writes, 8 Byte Read, 40 Byte Read, Three 8 Byte Writes (Part 2)

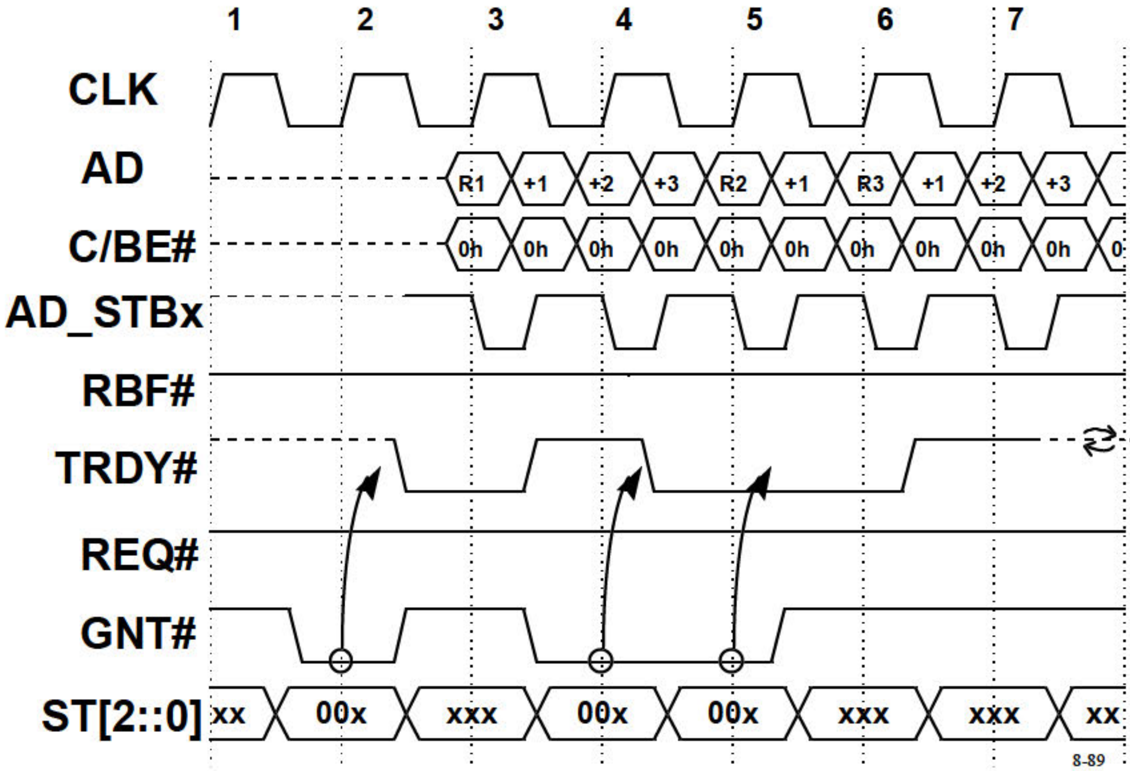


Figure 8-89 16 Byte Read, 8 Byte Read, 16 Byte Read

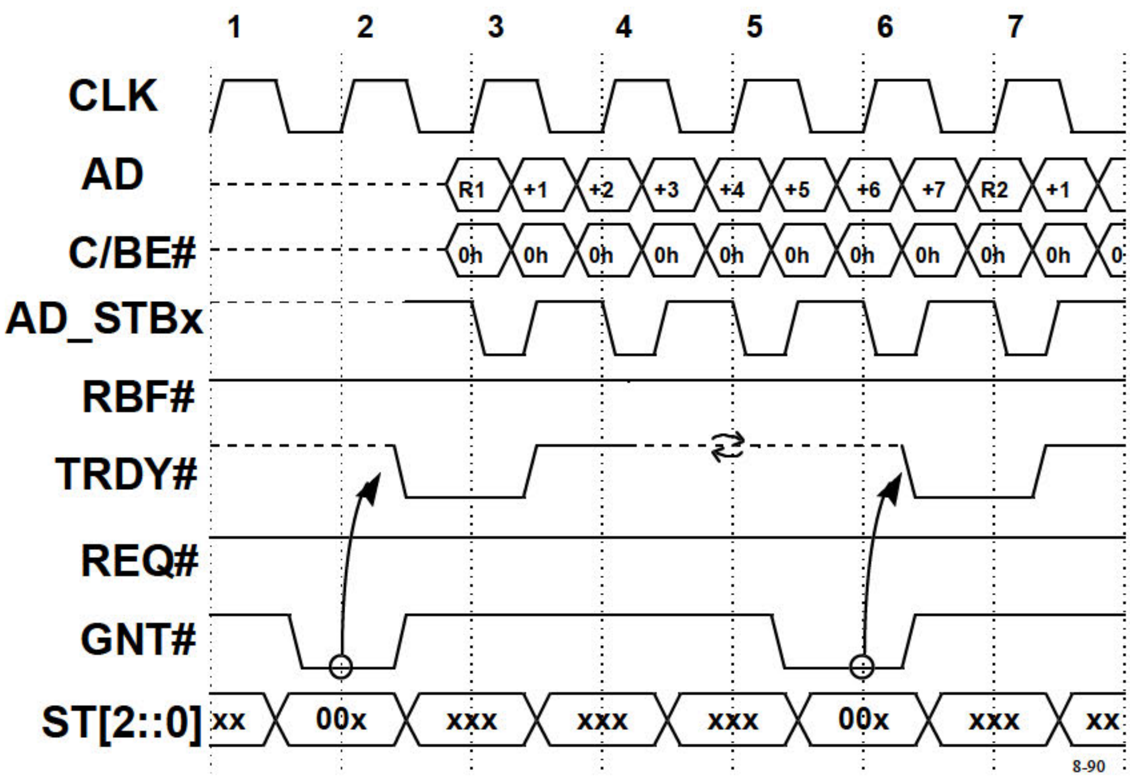


Figure 8-90 Back to Back 32 Byte Reads

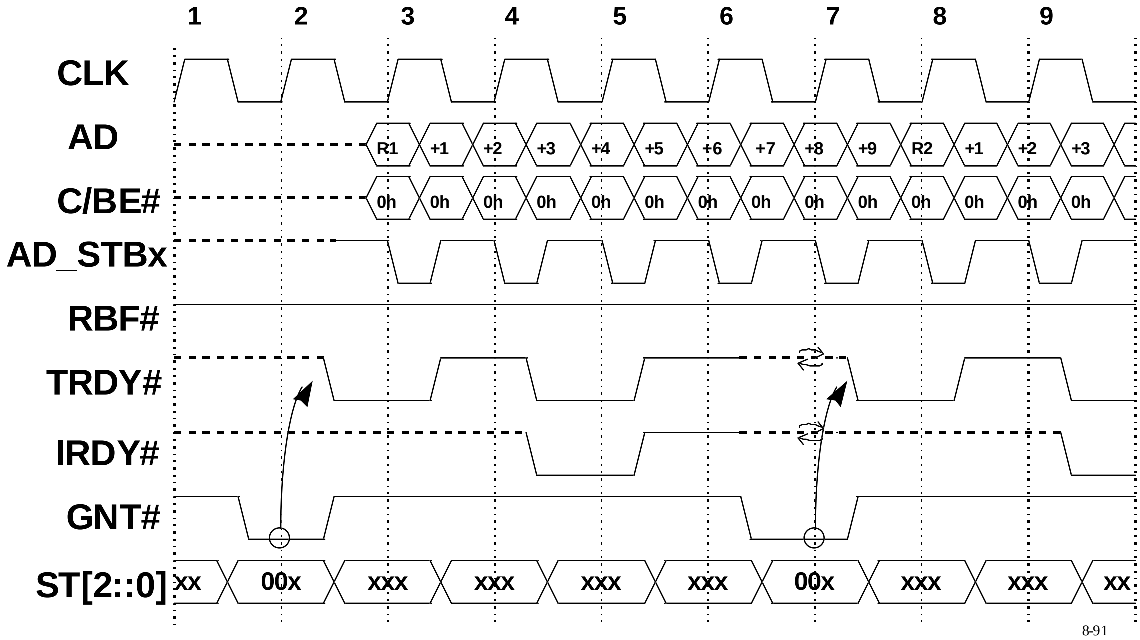


Figure 8-91 Back to Back 40 Byte Reads