ACCELERATED GRAPHICS PORT

Preliminary Draft of Accelerated Graphics Port Interface Specification

Preliminary Draft of Revision 2.0

Intel Corporation

December 10, 1997

Contents

Figures

Tables

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 the PCI bus. 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 (*PCI Local Bus Specification*) specification as an operational baseline, and provides four 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 Local Bus Specification*. 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.
- New AC timing in the 3.3 V electrical specification that provides for one or two data transfers per 66-MHz clock cycle, allowing for real data throughput in excess of 500 MB/s.
- A new low voltage electrical specification that allows four data transfers per 66-MHz clock cycle, providing real data throughput of up to 1 GB/s.

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.

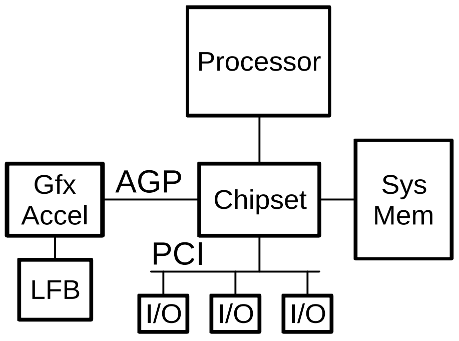


Figure 1-1: System Block Diagram: A.G.P. and PCI Relationship

1.3 Terminology

This revision of the A.G.P. interface specification refers to various devices that are compatible with the A.G.P. interface (for example, masters and targets). The *A.G.P. Interface Specification, Revision 1.0* refers to such devices as A.G.P.-compliant devices or A.G.P.-enabled devices (for example, A.G.P.-compliant masters, A.G.P.-compliant targets, and A.G.P.-enabled masters). This revision of the interface specification simplifies this terminology by using terms such as A.G.P. device, A.G.P. master, and A.G.P. target to refer to devices that are compatible with the A.G.P. interface specification.

2. Architectural Context and Scope

This chapter provides 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. It also is intended to set the technical scope of the interface specification in some areas by providing examples of issues beyond the purview of the formal interface specification.

2.1 Two Usage Models: "Execute" and "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 (the "DMA" operation) to which the rendering engine's address generator makes its 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. Because the accelerator generates direct references into system memory, a contiguous view of that space is essential; however, 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 remapping mechanism which insures the graphics accelerator (an A.G.P. master) will have a contiguous view of graphics data structures dynamically allocated in system memory.

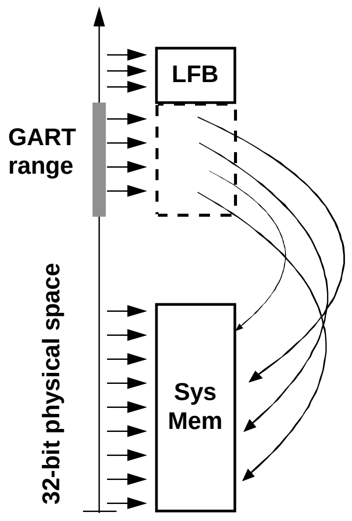


Figure 2-1: Graphics Address Remapping Function

This address remapping 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 remapped 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.

Remapping is accomplished via a memory-based[1] table called the Graphics Address Remapping Table (GART) and used ("walked") by the corelogic to perform the remapping. 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 miniport driver that must be provided with the corelogic. While this API does not constrain the future partitioning of remapping hardware, the remapping function will initially be implemented in the chipset or corelogic. *Note: this remapping 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 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. master (graphics) device may transfer data to system memory using either A.G.P. transactions or PCI transactions. The corelogic can access the A.G.P. master 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.

[1] The physical location of the table is chipset specific and may reside in main memory or in GART specific memory.

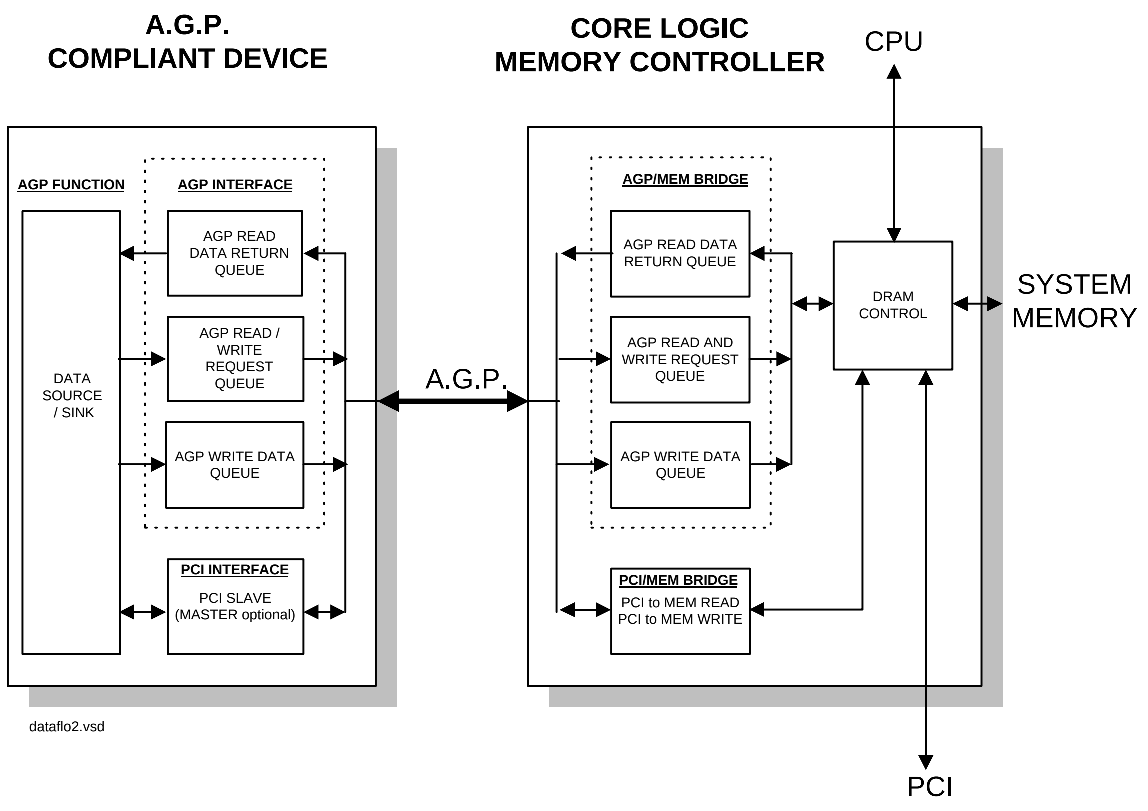


Figure 2-2: A.G.P. Access Queuing Model

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. master initiates an A.G.P. transaction with an access request. The corelogic 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. master 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 corelogic’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. master tracks the state of the request queue in order to limit the number of outstanding requests and identify data transactions.

The corelogic processes the access requests present in its request queue. Read data will be obtained from system memory and returned at the corelogic’s initiative via the ***read data return queue***. Write data will be provided by the A.G.P. device at the corelogic’s direction when space is available in the corelogic’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. 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 corelogic queue which handles processor accesses directly to the PCI target interface of the A.G.P. master, 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 which is possible for memory WRITE[2] transactions. 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 corelogic 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 corelogic vendors and/or OEMs provide these parameters for their systems to respective graphics IHVs.

The following are the basic parameters that each corelogic 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) and no waitstates 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) and no waitstates 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 ~10 ms (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 waitstates or control flow asserted by the graphics master.

[2] 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.

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. corelogic and platform implementations. This should not, however, have as much impact as it would in other buses for two reasons:

- The A.G.P. is a point-to-point[3] connection, intended for use by a 3D graphics accelerator only, and,
- Due to performance issues (Section 2.3), an A.G.P. graphics master will likely need to be optimized to a specific subset of platform or corelogic 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 corelogic 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.

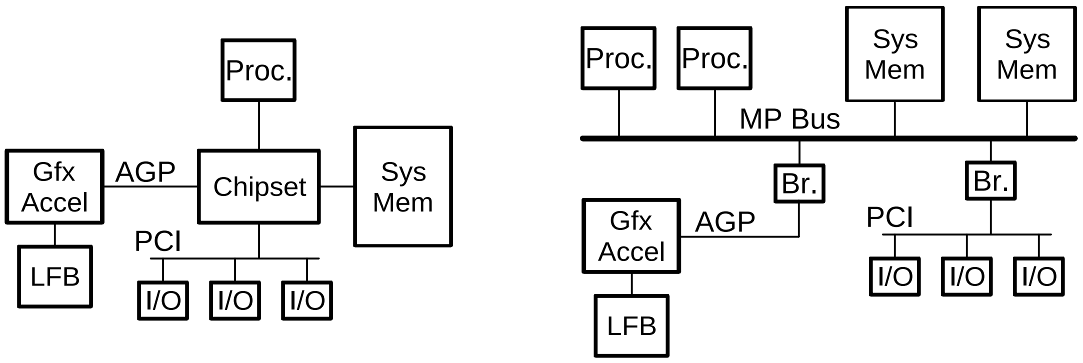


Figure 2-3: Different Corelogic Architectures

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 miniport driver supplied with the corelogic.

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. Selective snooping in an integrated corelogic architecture presents serious queue

[3] This means that active communication can only occur between two A.G.P. agents that reside on the interface, where one agent is referred to as the A.G.P. target and the other the A.G.P. master. The simplest implementation is to only have two devices attached to the bus. Attaching more than two devices to the interface is not precluded as long as there is only one active master and one active target. Any other device must not respond to or interfere with the interface operation. When more than two devices are attached to the interface, the system designer is responsible to ensure that all requirements of this interface specification are met, since the component and/or add-in card designer has no control how the devices are used.

management problems, while the MP bus in an MP corelogic architecture could well deal with selective snoops very easily. As a result, processor cache snooping on A.G.P. accesses is chipset dependent, and may not be counted upon in general. The exact coherency requirements for this interface specification are as follows:

- Processor cache coherency must be guaranteed by the corelogic for all PCI transactions (transactions initiated with the **FRAME#** signal) regardless of which bus segment (A.G.P. or PCI) they originate on. This is consistent with normal PCI operation. Coherency means that at the moment a PCI/A.G.P. memory request is serviced (completed), the subject data are fully consistent with any valid contents of the processor caching mechanism (excluding processor write combining buffers) for any of the target locations in the access.
- For A.G.P. transactions (transactions initiated with the **PIPE#** signal or on the de-multiplexed address bus, **SBA** port), there are no specific coherency requirements, and behavior is chipset dependent. It is entirely possible that some implementations will return stale read data, or allow write data to be overwritten, e.g., by a processor cache write back. Other implementations may provide full coherency support. For this reason, **any device driver managing an A.G.P. device is required to insure that A.G.P. transactions targeted at cacheable memory are safe**. In practice, this caution applies mostly to A.G.P. transactions **outside** the GART address range, since memory allocated **inside** the GART address range by the normal A.G.P. memory allocation procedure will be of a cache type (e.g., WC) consistent with the hardware's native ability to provide coherent access. In general, an A.G.P. device driver may determine the level of chipset coherency support via the chipset ID available in the Microsoft* Windows* 95/Windows NT* registry.

  Note: Options for supporting A.G.P. coherent access to cacheable memory (i.e., snooping required) are under consideration. Specific requirements on this topic are subject to change/elaboration.

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 (PCI) 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 only requires support for memory WRITES between a PCI master on the PCI bus and a (PCI) target on the A.G.P.[4] bus. Support of any other transaction between the two interfaces is optional and in general, should not be assumed to work.

Address Re-Mapping Support for PCI Bus Master Accesses to Memory

GART range address remapping support is provided by the chipset for graphics devices attached to the A.G.P. interface. Remapping support is, in general, **not** provided for devices attached to standard PCI bus(es). However, to provide a consistent addressing model across the system, this interface specification ***requires*** the following processor dependent[5], address remapping support for accesses presented at a PCI port. 1) Chipsets that allow the processor to generate physical addresses in the GART range (i.e., GART address remapping is supported for processor accesses as well as for A.G.P. accesses) ***must*** support an

[4] 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. interface.

[5] The chipset implementation of GART remapping is related to the memory management implementation of the attached processor, including, for example, the behavior of MTRRs. However, this reference to processor implementation is made as an example for clarification purposes only. This specification does not place any requirements on processor behavior, nor stipulate any particular processor behavior.

identical remapping service for the PCI port. 2) Chipsets that do not support remapping for processor accesses (i.e., the processor resolves its own GART range addresses to valid physical memory ranges) must NOT do any remapping of addresses presented at the PCI port.

In the case where GART remapping is not provided at the PCI port, any address in the GART address range that is presented at the PCI port must be deemed an access error, and dealt with consistent with the error handling policies of that platform. In general, out of bounds or otherwise erroneous PCI requests do not receive any response and result in a master abort. In ***NO*** case may an address falling in the GART range be propagated through the corelogic without remapping.

Device drivers managing PCI bus master devices should always use the standard system call to de-reference addresses being passed to the hardware. This will guarantee that the correct address is used for the current platform and level of chipset support. Conversely, device drivers managing A.G.P. bus master devices should always de-reference virtual addresses via the new or alternate system call associated with the new A.G.P. VMM services described in the Memphis DDK. This will guarantee that the linear GART address is always used on the A.G.P. port, independent of platform differences described here. PCI bus master device drivers should ***never*** use this new de-referencing call.

Monochrome Device Adapter Support

Existing PCI platforms typically support coexistence of a VGA device with a second monochrome device adapter (MDA), for debug and software development. A.G.P. corelogic implementations may elect to support this capability depending on their specific market needs. Possible implementations include static detection of MDA adapters present off the PCI bus, and rerouting of MDA accesses to the PCI bus under BIOS control; or snooping of A.G.P. directed accesses to dynamically detect disabling of MDA resources on the A.G.P. device and subsequent re-routing to the PCI bus. To ensure interoperation with possible corelogic implementations, additional requirements for A.G.P. graphics controllers with respect to MDA resources are specified in this document (Chapter 6).

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 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. add-in board, and the A.G.P. configuration registers which control the graphics address remapping function.

3. Signals and Protocol Specification

3.1 Pin Description

A.G.P. protocol defines 21 new signals beyond what PCI uses. Which of these signals are implemented depends on which features of the bus are supported and whether the device is an A.G.P. master or target. The corelogic is required to support all signals required to:

1. Allow the A.G.P. master to enqueue requests and
2. Transfer data at 1x or 2x.

The corelogic may optionally support:

1. Data transfers at 4x or
2. Fast Write (FW) transactions to the A.G.P. master.

The A.G.P. master may optionally choose:

1. How it enqueues requests or
2. The rate at which it transfers data or
3. If it supports FW transactions.

The A.G.P. signals follow the signal type definitions and naming convention used in the *PCI Local Bus Specification*. The following signal type definitions are from the view point of the A.G.P. target:

in	*Input* is an input-only signal.
out	*Totem Pole Output* is an active driver.
t/s	*Tri-State* is a bidirectional, 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 an s/t/s pin low must drive it high for at least one clock before letting it float. A new agent cannot start driving an 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 central resource (A.G.P. target or motherboard).

The following tables list 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 corelogic 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 corelogic. The following tables describe the operation and use of each signal and are organized in four groups: A.G.P. Requests, A.G.P. Flow Control, A.G.P. Status, and A.G.P. Clocking.

Table 3-1: A.G.P. Requests

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 corelogic).*
SBA[7::0]	in	*SideBand Address port* provides an additional bus to pass requests (address and command) to the target from the master. **SBA[7::0]** are outputs from the master and an input to the target. This port is ignored by the target until enabled (see Section 6.1.10).

Table 3-1 contains two mechanisms to enqueue requests by the A.G.P. 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 requests, the master is not allowed to enqueue requests using the **SBA** port. When the **SBA** port is used to enqueue requests, **PIPE#** cannot be used to enqueue requests.

Table 3-2: 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. This signal must be pulled up by the central resource (A.G.P. target or motherboard).
WBF#	in	*Write Buffer Full* indicates if the master is ready to accept FW[6] data from the corelogic or not. When **WBF#** is asserted, the corelogic arbiter is not allowed to initiate a transaction to provide FW data. This signal must be pulled up by the central resource (A.G.P. target or motherboard).

Table 3-2 contains the A.G.P. flow control beyond normal PCI flow control already required. When the master is *always* ready to accept the return of LP (Low Priority) Read data, the A.G.P. master is not required to implement **RBF#** and the corresponding pin on the target is tied (pulled up) in the deasserted state by the central resource. When the master is always ready to accept the first block of FW data, the A.G.P. master is not required to implement **WBF#** and the corresponding pin on the target is tied (pulled up) in the deasserted state by the central resource.

[6] Fast Write data is typically data generated by the CPU and *pushed* to the A.G.P. master. See Section 3.5.3.5 for details.

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

Name	Type	Description
ST[2::0]	out	*Status* bus provides information from the arbiter to the 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 corelogic and an input to the master.

Table 3-3 describes the status signals, which 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 send previously enqueued write data. The **ST[2::0]** signals are qualified by the assertion of **GNT#**.

Table 3-4: 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 **AD[15::00]**. The agent that is providing data drives this signal.
AD_STB0#	s/t/s (in/out)	**AD** *Bus Strobe 0 compliment* and **AD_STB0** provide timing for 4x data transfer mode on **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 **AD[31::16]**. The agent that is providing data drives this signal.
AD_STB1#	s/t/s (in/out)	**AD** *Bus Strobe 1 compliment* and **AD_STB1** provide timing for 4x data transfer mode on **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]** (when supported) and is always driven by the A.G.P. master. When the SideBand Strobes have been idle, a synch cycle needs to be performed before a request can be enqueued. (See Section 4.1.2.10 for details).
SB_STB#	s/t/s (in)	*SideBand Strobe compliment* and **SB_STB#** provide timing for **SBA[7::0]** (when supported) when 4x timing is supported and is always driven by the A.G.P. master.
CLK	t/s (out)	*Clock* provides timing for A.G.P. and PCI control signals.

Table 3-4 describes the clock signals used on the A.G.P. interface and when they are used. The basic **CLK** signal is used to time all control signals on the interface and is also used to transfer data in the 1x mode. The other two strobes are used to transfer data on the **AD** bus or the **SBA** port. Since the **AD** bus is 32 bits wide, two copies of the **AD_STB** are required. When the 4x mode is used, the compliments of the strobes are also required.

Table 3-5: USB Signals

Name	**Type**	**Description**
USB+	t/s	*USB Positive Differential Data Line,* used to send USB data and control packets to external peripheral devices (typically a USB capable video monitor in this application). For complete details on the USB signaling characteristics and requirements, see the *Universal Serial Bus Specification*.
USB-	t/s	*USB Negative Differential Data Line.* See above reference.
OVRCNT#	Note 1	*USB Overcurrent Indicator* is low when too much current has been taken from the 5 volt power supply (Vbus) line on the monitor connector. Otherwise, the line is between 2.4 volts and Vddq.

Notes

1. Overcurrent indication can be provided either by an active sense circuit or by a passive sensing of the USB power out (Vbus) after a fuse. An example of such a passive sensing circuit is a resistor divider of 10 KΩ ±5% from Vbus to **OVRCNT#** and 15 KΩ ±5% from **OVRCNT#** to ground. Implementations may vary. Boards which do not provide power to the monitor cable must pull this line to Vddq through a pull-up resistor.

Table 3-6: Power Management on A.G.P.

Name	Description
PME#	*Power Management Event* is not used by the A.G.P protocol, but is used by the PCI target interface when being power managed by the operating system. Refer to the *PCI Bus Power Management Interface Specification* for the definition of **PME#** and how a device is power managed by the operating system.

Table 3-6 is included in this interface specification to point out the PCI Power Management signal which is used to control the power used by the device. Since an A.G.P. device is also a PCI target, the device follows the *PCI Bus Power Management Interface Specification* when supporting power management. Care needs to be taken when supporting power management to ensure correct operation and compliance with the power management specification.

Table 3-7: Special Interface Signals

Name	Description
TYPEDET#	*Type detect* indicates whether the interface is 1.5 volt or 3.3 volt. Refer to Section 4.3.4 for a description of this signal.

3.1.1 Semantics of PCI Signals

PCI signals, for the most part, are used in a similar manner when doing an A.G.P. transaction. **FRAME#, IDSEL**, **STOP#**, and **DEVSEL#** are not used by the A.G.P. pipelined protocol but are used (except **IDSEL**) for an FW transactions. **IDSEL**, **LOCK#**, **INTC#**, and **INTD#** are not supported on the A.G.P. connector. The exact role of all PCI signals during A.G.P. transactions is described in Table 3-8.

Table 3-8: PCI Signals in Relation to A.G.P.

PCI Signal	Relationship With A.G.P.
FRAME#	Not used during an A.G.P. pipelined transaction and is kept in the deasserted state by the central resource. **FRAME#** is used for FW transactions.
IRDY#	Indicates the A.G.P. 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 accept a subsequent block of read data. The master is *never* allowed to insert a waitstate during the initial block of a read transaction. However, it may insert waitstates after each subsequent block transfers. (There is no **FRAME#** -- **IRDY#** relationship for A.G.P. transactions.*)* For FW transactions, **IRDY#** is driven by the corelogic to indicate when the write data is valid on the bus. The corelogic is allowed to insert waitstates on block boundaries but not on individual data phases.
TRDY#	Indicates the A.G.P. target is ready to provide read data for the entire transaction (when transaction can complete within four clocks) 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 waitstates after each block transfers on both read and write transactions. For FW transactions, the A.G.P. master uses **TRDY#** to indicate if and when it is willing to transfer a subsequent block.
STOP#	Not used during an A.G.P. transaction and is kept in the deasserted state by the central resource. For FW transactions, **STOP#** is used to signal Disconnect or Target-abort terminations. The PCI target termination of Retry is not supported when performing FW protocol.
DEVSEL#	Not used during an A.G.P. transaction and is kept in the deasserted state by the central resource. For FW transactions, it is used when the transaction cannot complete during the block.
IDSEL	Not part of the A.G.P. connector and is generated internally by the graphics PCI target interface. (See Section 3.1.2 for details.)
PERR#	Not used during an A.G.P. transaction and is kept in the deasserted state by the central resource. (Optional for PCI operation per exceptions granted by the *PCI Local Bus Specification.*)
SERR#	Same as PCI. (May be used by an A.G.P. master to report a catastrophic error when the corelogic supports an **SERR#**[7] pin for the A.G.P. port.)

[7] 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**.

Table 3-8: PCI Signals in Relation to A.G.P.

PCI Signal	Relationship With A.G.P.
REQ#	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 it has been given permission to start a bus transaction (A.G.P. or PCI).
RST#	Same as PCI.
AD[31::00]	Same as PCI.
C/BE[3::0]#	Slightly different meaning. Provides command information (different commands than PCI) from 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. master.
PAR	Not valid during an A.G.P. transaction, but must be actively driven by the current owner of the **AD** bus.
LOCK#	Is not supported on the A.G.P. interface for either A.G.P. or PCI transactions.
INTA#, INTB#	Interrupt request signals[8] 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.

Each signal can be required, optional, or not applicable, depending on the type of device (graphics or corelogic), which interface (PCI or A.G.P.) is supported, and the type of agent (master or target). Table 3-9 is a summary of the functionality supported by function and agent. Table 3-10 is a summary of the PCI signals supported by function and agent. Table 3-11 is a summary of the A.G.P. signals supported by function and agent. For example, the second column represents a graphics agent that supports the PCI interface as a target. The row labeled Support indicates whether the interface is required or optional. For the example, the PCI target interface is required by a graphics agent. The gray boxes indicate that the signal is not applicable to the function.

Table 3-9: Summary of Interfaces Based on Function and Agent

Device	**Graphics**				**Corelogic**			
Interface	**PCI**		**A.G.P.**		**PCI**		**A.G.P.**	
Agent	Target	Master	Master	FW_{Target}	Target	Master	Target	FW_{Master}
Support	R	O	O	O	R	R	R	O

Legend:

R	Required
O	Optional

[8] These can be tied to the PCI **INTx#** signals since these are o/d signals and are level sensitive. However, care must be taken to ensure electrical interface requirements are met. Refer to Section 4.3.4 for details.

Table 3-10: Summary of PCI Signals Based on Function and Agent

Device	**Graphics**				**Corelogic**			
Interface	**PCI**		**A.G.P.**		**PCI**		**A.G.P.**	
Agent	Target	Master	Master	FW_{Target}	Target	Master	Target	FW_{Master}
FRAME#	R	R	▨	R	R	R	▨	R
IRDY#	R	R	R	R	R	R	R	R
TRDY#	R	R	R	R	R	R	R	R
STOP#	R	R	▨	R	R	R	▨	R
DEVSEL#	R	R	▨	R	R	R	▨	R
IDSEL	R[1]	▨	▨	▨	R[3]	▨	▨	▨
PERR#	R	R	▨	▨	R	R	▨	▨
SERR#	O	O	▨	▨	O	O	▨	▨
REQ#	▨	R	O[2]	▨	▨	R	R	I
GNT#	▨	R	R	▨	▨	R	R	I
RST#	R	R	R	R	R	R	R	R
AD[31::00]	R	R	R	R	R	R	R	R
C/BE[3::0]#	R	R	R	R	R	R	R	R
PAR	R	R	NS	NS	R	R	NS	NS
LOCK#	NS	NS	NS	NS	NS	NS	NS	NS
INTA#	O	O	O	▨	R	O	R	▨
INTB#	O	O	O	▨	R	O	R	▨
CLK	R	R	R	R	R	R	R	R
PME#	O	▨	▨	▨	O	▨	▨	▨

Legend:

R	Required
O	Optional
I	Internal signal
NS	Not supported
▨	A shaded cell indicates that the signal is not applicable to the function.

Notes: 1 – **IDSEL** is not a signal on the connector. See Section 3.1.2 for details of how **IDSEL** is implemented.

2 – **REQ#** is required when requests are enqueued on the **AD** bus.

3 – **IDSEL** is typically an internal signal for the corelogic.

Table 3-11: Summary of A.G.P. Signals Based on Function and Agent

Device	**Graphics**				**Corelogic**			
Interface	PCI		A.G.P.		PCI		A.G.P.	
Agent	Target	Master	Master	FW$_{Target}$	Target	Master	Target	FW$_{Master}$
ST[2::0]		O[5]	R			R	R	
PIPE#			O[1]				R	
SBA[7::0]			O[1]				R	
RBF#			O[2]				R	
WBF#				O[4]				R
AD_STB0			2x or 4x	R			R	R
AD_STB0#			4x	4x			4x	4x
AD_STB1			2x or 4x	R			R	R
AD_STB1#			4x	4x			4x	4x
SB_STB			O				R	
SB_STB#			O[3]				4x	

Legend:

R	Required
O	Optional
I	Internal signal
NS	Not supported
▨	A shaded cell indicates that the signal is not applicable to the function.

Notes: 1 – Either **PIPE#** or **SBA[7::0]** must be supported; not both.

2 – **RBF#** is not required by the A.G.P. master if it can always accept the return of LP Read data.

3 – **SB_STB#** is required when using the **SBA** port and data transfers at 4x.

4 – **WBF#** is not required by the A.G.P. master if it can always accept the first block of FW data.

5 – **ST#** signals are only optional to the PCI master if no A.G.P. support is implemented.

3.1.2 Configuration of an A.G.P. Master

Initialization of an A.G.P. device is done via the configuration mechanism defined by the *PCI Local Bus Specification*. This interface specification does not define a new mechanism. An A.G.P. master is composed of a PCI target interface and an A.G.P. master interface. (Optionally, the device can also include a PCI master interface when required.) The PCI target interface follows the *PCI Local Bus Specification*. This requires the device to respond to a PCI configuration transaction when a configuration command (read or write) is decoded and **AD01** and **AD00** are both "0" and the device's **IDSEL** is asserted. Since **IDSEL** is not a signal in the A.G.P. connector, it must be connected to **AD16** at the component. The designer of the A.G.P. master must be careful as to how this connection is made. It must be connected internally for A.G.P. operation while it must be connected externally for PCI operation. The next two sections will describe how this connection must be made based on the targeted market segment of the device.

3.1.2.1 Device for A.G.P. Only Operation

When the device is designed for exclusive operation on the A.G.P. interface, the device does not have an external **IDSEL** pin. In this implementation, the device asserts **DEVSEL#** when the bus command is configuration (read or write). **AD16** is a "1" while **AD1** and **AD0** are "00". Under all other conditions, the device's configuration space has not been selected and the device does not assert **DEVSEL#** to claim the access. System software will scan all configuration spaces supported by asserting a different **AD** signal between **AD16** and **AD31** while performing a PCI configuration read or write command. A device located on that segment can only assert **DEVSEL#** for a single configuration space which is uniquely identified by having its **IDSEL** asserted when **AD1**-**0** are "00". The exception is for a multi-function device that has bit 7 of the header type field set. In this case, the different functions are selected based on the function number decoded (**AD10-AD8**). See the *PCI Local Bus Specification* for more details.

3.1.2.2 Device for Both PCI and A.G.P. Operation

When a device is designed to be used on both A.G.P. and PCI bus segments, then the device needs to have two modes of operation. When in the A.G.P. mode, the device generates **DEVSEL#** as described in the A.G.P. only implementation. When used in a PCI mode of operation, the device must provide an external **IDSEL** that is connected to one of the **AD** signals. Which **AD** signal it is connected to is determined by the system designer and NOT by the component designer. In this case, the device must be "strapped"[9] to indicate which mode it is operating in. Note: that besides how **DEVSEL#** is generated during configuration accesses, the device must also know the strength in which it drives its output buffers. Note: A.G.P. requires only half the strength of a PCI buffer, since A.G.P. is a point-to-point connection and not a bus environment like PCI.

[9] Refers to the condition where information is conveyed to the device by sampling a pin to determine how the device should behave. For example, if the pin is tied to Vcc the device supports mode X, otherwise it supports mode Y.

3.2 Operation Overview

Memory access pipelining is one of the major PCI protocol enhancement provided under this 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 Local Bus Specification*. The other enhancement to the PCI protocol is the acceleration of memory write transactions from the corelogic to the A.G.P. master device acting like a PCI target. This is called fast write (FW). When the FW protocol is enabled, the PCI write data target at the A.G.P. master transfers at the same rate as the A.G.P. data transfers. PCI memory write targeted at the corelogic from the A.G.P. master transfers at the PCI data rate. FW flow control is more like A.G.P. than PCI. See Section 3.5.2.2.2 for details.

A.G.P. pipelined operation allows for a single *A.G.P. target*, which must always be the system memory controller, referred to in this document as *corelogic*. In addition to A.G.P. target functions, the corelogic must also implement a complete PCI sequencer[10], 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. master*, which, in addition to implementing the A.G.P. master functions, must also provide full PCI target functionality[11]. PCI master functionality is optional.

3.2.1 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[12], 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 6.1.10). 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.

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.

[10] The A.G.P. Target must behave as a PCI 2.1 compliant master and target with two exceptions. The A.G.P. target is not required to adhere to either the target initial latency requirements or the target subsequent latency requirements as stated in the *PCI Local Bus Specification*.

[11] The A.G.P. master must function as a PCI compliant PCI target.

[12] There are some optional mechanisms that allow address demultiplexing, such as using an alternate (non-**AD** bus) mechanism for transferring address, which is described in Section 3.5.1.1.

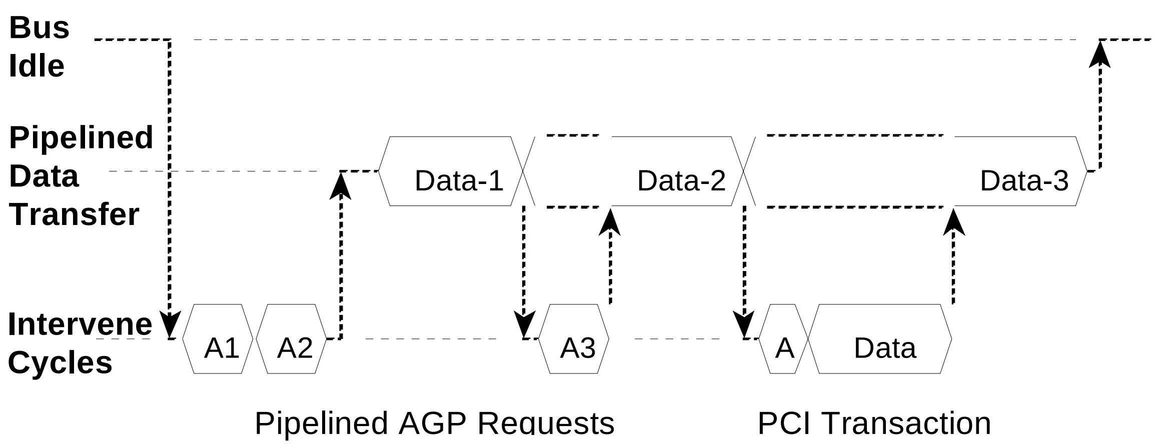


Figure 3-1: Basic A.G.P. Pipeline Concept

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 four 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.

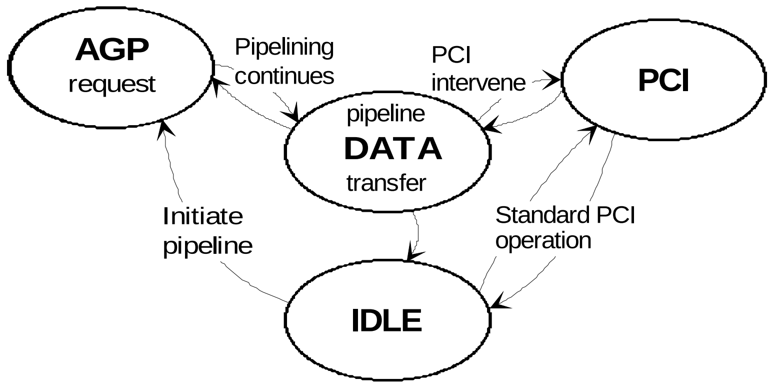


Figure 3-2: A.G.P./PCI Operational States

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 corelogic (the sole A.G.P. target) controls the **AD** lines and transfers data. If a bus master then requests the bus (**REQ#**), the A.G.P. arbiter (always located in A.G.P. target/corelogic) 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 needs to be enqueued while data is pending, the machine transitions from DATA to A.G.P. or PCI state depending on what type of request is initiated. Only when all previously requested data has been transferred, does the machine return to the IDLE state. 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 3.5.2.

3.2.2 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. pipelined transactions (both reads and writes) is "disconnected" from its associated access request. That is the 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 corelogic 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. All access ordering rules on the A.G.P. interface are based on the arrival order of the access requests, and not the order of actual data transfer.
2. Transactions use a completely different set of bus commands (see Section 3.3) 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, and configuration.) The address space used by A.G.P. commands is the same linear physical space also used by PCI memory space commands[13], 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.[14] (Memory accesses for PCI transactions have 4-byte granularity, aligned on 4-byte boundaries.) Smaller or odd size reads must be accomplished with a 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.
4. Pipelined access requests have an explicitly defined access length or size. (PCI transfer lengths are defined by the duration of **FRAME#**.)
5. Pipelined accesses do not guarantee memory coherency. That is, 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.

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

[14] 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 graphics accelerators.

3.3 Bus Commands

Bus commands indicate to the corelogic the type of pipelined transaction the master is requesting on the A.G.P. interface. Bus commands are encoded on **C/BE[3::0]#** when enqueuing a request on the **AD** bus or encoded in a type 2 command when using the **SBA** port. (See Section 3.5.1 for details.) The format of a complete A.G.P. bus request is shown in Figure 3-3.

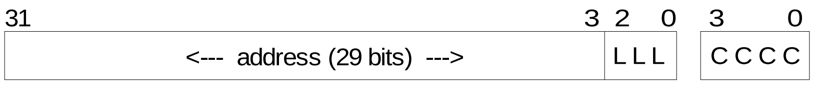


Figure 3-3: Layout of an A.G.P. Access Request

The "LLL" field contains the access length in units of Qwords (8 bytes), and displaces the low order 3 bits of address. A length field of "000" indicates that a single Qword (8 bytes) of data is being requested, while "111" indicates eight Qwords (64 bytes) are being requested. The "CCCC" field contains the bus operation or command as itemized in Table 3-12. A brief description of each command follows the table.

Table 3-12: A.G.P. Bus Commands

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	Dual Address Cycle (DAC)
1110	reserved
1111	reserved

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

Read (hi-priority): same as *Read*, but the request is queued in the high priority queue. The reply data may be returned out of order with respect to other requests. However, the reply data will be returned in order with respect to other HP Read requests, and within the maximum latency window established for high priority accesses (see Section 2.3). High priority read 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 Qwords, 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[15] (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 a 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 Qword of random data as an indication of its completion (see Section 3.4.3). The address and length fields are meaningless for this command.

Fence: creates a boundary in a single master's LP access stream around which writes may not pass reads (see Section 3.4.3). This command is the only one which does NOT occupy a slot in the A.G.P. pipeline.

Dual Address Cycle: is used by the master to transfer a 64 bit address to the corelogic when using the **AD** bus. When using the SBA[16] port to enqueue requests, the DAC command is not valid and must be implemented as a reserved command. The master is required to use two clock periods to transfer the entire address using **AD[31::00]** and **C/BE[3::0]#**. During the first clock, the master provides the lower address bits (A31-A03) and the length encoding on (A2-A0), just like a 32 bit request, but provides the DAC command (1101) encoding on **C/BE[3::0]#** instead of the actual command. The second clock of the request contains the upper address bits (A63-A32) on **AD[31::00]** and the actual command on **C/BE[3::0]#**.

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

[15] 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.

[16] The SBA port uses the Type 4 command when requesting data above the 4 GB boundary.

3.4 Access Ordering Rules

3.4.1 Ordering Rules and Implications

The A.G.P. ordering rules are different than the CPU ordering rules and those of PCI. Since each type of transaction has different ordering rules, this section will discuss the interaction between A.G.P. streams and other operations. These interactions can be broken down into three basic categories:

1. Different System Streams (A.G.P., CPU, and PCI)
2. Different A.G.P. streams (high priority (HP) and low priority (LP))
3. Same stream (Read and Write)

These rules only apply to operations that are initiated or completed on the A.G.P. interface. The interaction between transactions that are not initiated or completed on the A.G.P. interface is beyond the scope of this interface specification and will not be discussed.

Different System Streams (A.G.P., CPU, and PCI)

There is no ordering relationship between an A.G.P. master's operation and any other system operation, including operations generated by host CPU(s), PCI agents, or expansion bus agents. This means that an A.G.P. transaction is only required to follow A.G.P. ordering rules even when the A.G.P. transaction crosses into other domains. For example, an A.G.P. master initiates a read to a location in system memory that is currently locked by the processor. The corelogic is not required to honor the CPU's lock. The corelogic is allowed to complete the request in the face of the lock and if this causes incorrect operation, then a programming error has occurred.

The A.G.P. target 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 an A.G.P. target is allowed to complete the request from system memory without checking the CPU cache for a more recent copy. For a write, the A.G.P. target can simply write the data to system memory without snooping the CPU cache. If the cache has a modified line at the same location, it will overwrite the A.G.P. write at some point. The A.G.P. target is not required to *force* A.G.P. data out of the A.G.P. domain into system memory or even into buffers that participate in ordering rules of system memory. When an A.G.P. master needs to cause a synchronization event (request an interrupt or set a flag) to occur, it uses the Flush command. The Flush command forces previously issued LP A.G.P. write transactions to be forced into system memory or into buffers that participate in ordering rules of the system. This means that LP write data already transferred across the bus or write data that has been issued but not transferred.

There is no relationship between A.G.P. and PCI transactions on the A.G.P. interface except that they use the same wires. This means that the order in which they complete does not matter.

PCI transactions initiated by an A.G.P. master or target must follow the ordering rules specified in the *PCI Local Bus Specification*. This includes the A.G.P. defined FW operations.

When an A.G.P. agent is capable of generating both PCI and A.G.P. transactions, the A.G.P. target is not required to maintain any ordering between these two streams. However, the A.G.P. 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 no ordering relationships. The same rules are followed when a PCI transaction is generated by a device on the A.G.P. interface or by a device that resides on a PCI bus segment. The PCI agent's transactions will follow the same rules as described in the *PCI Local Bus Specification* even though it was initiated on the A.G.P. interface. This agent's transaction has no ordering with respect to any A.G.P. transactions that occur.

Different A.G.P. Streams (HP and LP)

There is no relationship between different A.G.P. streams. This means that the order that HP and LP transactions complete with respect to each other does not matter[17].

> If Requests are made in the following order: "HP1, HP2, LP3, LP4, and HP5",they may complete as: "HP1, HP2, HP5, LP3, LP4" or "LP3, LP4, HP1, HP2, HP5" to give two examples.

Same Stream (Read and Write)

Requests of the same type (LP Read, HP Read, HP write, or LP write) must complete in the same order that they were requested with respect to themselves. The A.G.P. target will return a stream of A.G.P. Read data in the same order as requested.

> Reads requested in the order "A, B, C, D" will return data in the same order as requested – "A, B, C, D".

Even though the A.G.P. 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. The A.G.P. target is allowed to re-arrange Read requests to improve performance, but is never allowed to return data in a different order than requested.

> The master requests 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 the same order as requested.

The A.G.P. target will complete a stream of write data to system memory in the same order as requested. This rule means that A.G.P. write data cannot pass previously written A.G.P. data.

> Writes requested in the order "A then B" where A and B overlap will cause B to overwrite part of A.

For HP Requests, there is no relationship between read and write requests. An HP Read request can pass and complete before a previously enqueued HP write request. An HP write request can pass and complete before a previously enqueued HP Read request.

> HP transactions requested as "Ra and then Wb" to the same address can complete as either '"Ra then Wb", or "Wb then Ra". This means that the read may return old or new data.

For LP transactions, there is a relationship between read and write requests. Read data returned will be coherent with previously issued A.G.P. Write requests. Write data is allowed to pass previously enqueued read requests. Therefore, when the read data is returned, it may be the old or new data.

> LP transactions requested in the order "Wa, Wb, Rc, Wd, Re" to the same address. Read data returned for Re will be what was written by Wd. (Read requests push write requests.)

[17] The High Priority stream latency guarantee provided by the corelogic may place some constraints on the order in which the corelogic retires requests.

LP Read requests will push LP Write data from an A.G.P. master. An A.G.P. LP Read to a location previously written by an A.G.P. LP write operation will return the latest copy of the data seen by the A.G.P. interface.

> LP transactions requested 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.

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. 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 3.4.3) between the read and the write.

A master enqueues LP Read x and then LP Read y. Before the read data is returned, an LP Write to location x and then an LP 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. 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.

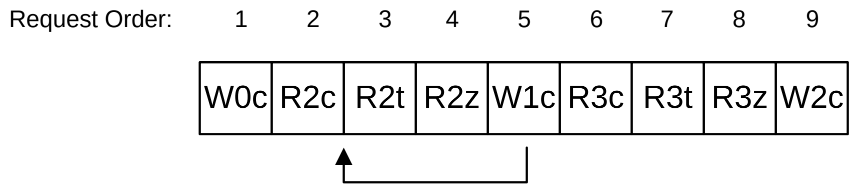
Implications of Allowing A.G.P. Writes to Pass A.G.P. Reads

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:

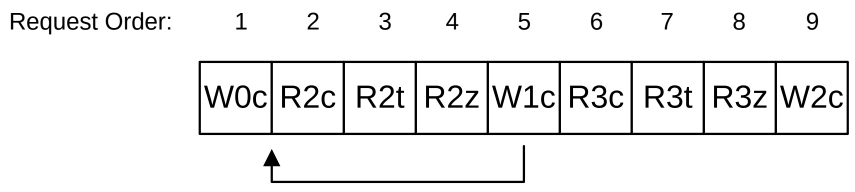
A 3D graphics controller master generates the following sequence of pipelined A.G.P. Requests. In this example, the reads are from the frame buffer, texture buffer, and depth buffer respectively, while the writes are to the frame buffer. This example assumes that all frame buffer accesses are to the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

Request Order:	1	2	3	4	5	6	7	8	9
	W0c	R2c	R2t	R2z	W1c	R3c	R3t	R3z	W2c

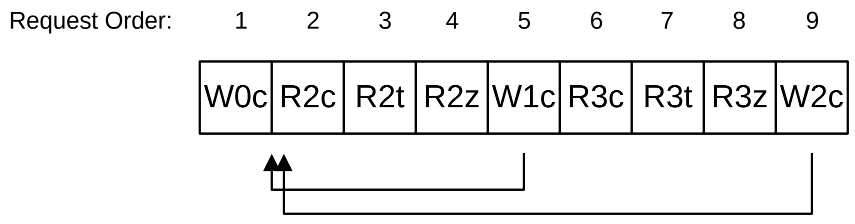
The following diagram shows W1c passing R2t. In this case, R2c will return "old" data from the W0c write.



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.)



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

If an A.G.P. master does not 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. master is particular about getting “new” or “old” data, it is the A.G.P. master’s responsibility to ensure that it gets the correct data. There are various methods to ensure this. Some of these methods are discussed next.

If an A.G.P. 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 LP Read request behind the LP Write request. Since LP Reads “push” LP Writes per the ordering rules, the LP 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 an LP Read and a subsequent LP Write, may be difficult (or impossible). Once a conflict is detected, delaying the LP Read may stall the pipeline and impact performance.

If an A.G.P. Master Must Get “Old” Data it May:

Issue a Fence command between the LP Read and the following LP Write or delay the “new” LP Write data until the “old” LP Read data has been returned. The latter method has the potential for deadlock. A deadlock occurs when delaying an LP Write causes an A.G.P. master’s data engine to back-up. If the A.G.P. master’s LP Read data return buffers are full, the stalled data engine cannot remove LP Read data from the buffers. The “old” LP Read data cannot be accepted and the LP Write will continue to be delayed, creating the deadlock. For HP transactions, the only way to guarantee old data is to delay the write until the read completes, since Fence is an LP command and does not apply to HP requests.

Summary Of Ordering Rules

X Pass Y?	HIGH	LOW	PCI
Read -> Read	No	No	No
Write -> Write	No	No	No
Read -> Write	Yes	No	No
Write -> Read	Yes	Yes	No

3.4.2 Deadlock Avoidance

An A.G.P. master cannot 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.

3.4.3 Flush and Fence Commands

Because of the ordering rules of the A.G.P. interface, the master cannot guarantee when LP write data has reached its final destination. From the A.G.P. master's standpoint, an LP write transaction appears to have completed but the LP 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 transactions 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. LP transactions to become visible to the rest of the system so synchronization events may occur. The Fence command guarantees what order LP accesses will complete in, without delaying the issuing of subsequent LP commands. Each will be discussed in more detail in the following paragraphs. The Fence and Flush commands are LP 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. master. The Flush command ensures that its LP write transactions have become visible to the rest of the system. Because of the A.G.P. ordering rules, the master cannot cause accesses to become visible to the system by using the memory commands as is possible with PCI commands. Memory commands can only cause data to be returned in a specific order; they place no requirement on the corelogic to make accesses visible to the system. However, the corelogic must cause A.G.P. LP accesses to become visible to the rest of the system when the Flush command is issued. The Flush command behaves similar to an LP 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 LP 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. master writes LP 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 system 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 LP 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 LP data is returned. The only limit to the number of outstanding Flush requests is the limit 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 LP writes not to pass previously enqueued LP Read commands. An A.G.P. master uses the Fence command to demarcate one set of LP A.G.P. Requests from another. The Fence command affects the order in which A.G.P. Requests 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* LP requests generated prior to the Fence command are processed prior to *any* LP request following the Fence command. A.G.P. LP Write requests generated after a Fence command may not pass any LP Read requests generated 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 LP Request Queue of the A.G.P. master or target. An A.G.P. master may generate an unlimited number of Fence commands.

3.4.4 Access Request Priority

The A.G.P. bus command set supports two levels of access priority. In general, the HP Queue has the highest priority for memory service, and the LP Queue has lower priority than the processor, but generally higher than any other subsystem for memory service. The HP Queue should be used with caution since it causes additional latency to other requests. For example, the HP 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 HP Queue is intended for very selective use and only when an A.G.P. Request needs immediate processing.

Requests in the HP Queue may bypass all other (LP 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 an HP command. HP accesses only have order with respect to the same type of request. For example, HP Read requests only have ordering with respect to other HP Read requests. HP write accesses only have ordering with respect to other HP write accesses. Unlike LP operations, there are no ordering requirements between HP Read and HP write accesses. The sequence, HPR-A, HPW-B, HPR-C, and HPW-D, will be used in the following discussion. Read data will be returned in the order in which read accesses were requested. In this example, A will always complete before 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 HP operations. In this example, the accesses may complete as: A, C, B, and D; or A, B, C, and D; or A, B, D and C; or B, A, D, and C; or B, D, A, and C; or A, B, D 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 HP 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 HP accesses will have transferred the data across the interface within the maximum latency window established for HP accesses. This does not imply that the data will have been retired to main memory within this latency window.

3.5 Bus Transactions

A.G.P. has two types of operation on the interface, enqueueing requests and transferring data. Each is a separate and distinct bus operation. The following sections will discuss the actual transaction and not how an agent requests permission to initiate the transaction. In this section, the **REQ#** and **GNT#** signals have been intentionally been omitted to simplify the diagrams. How permission is obtained for an agent to use the interface and when it is required to start a transaction, once permission has been granted, is discussed in Section 3.6.

In describing how each transaction is initiated and completed, timing diagrams will be used. To help understand the relationship of significant signals involved in A.G.P. transactions, a brief discussion about how the diagrams are drawn is useful. When a signal is drawn as a solid line, it is actively being driven by the current master or target. When a signal is drawn as a dashed line, no agent is actively driving it. However, it may still be assumed to contain a stable value if the dashed line is at the high rail. Tri-stated signals are indicated to have indeterminate values when the dashed line is between the two rails (e.g., **AD** or **C/BE#** lines). When a solid line becomes a dotted line, it indicates the signal was actively driven and now is tri-stated. When a solid line makes a low to high transition and then becomes a dotted line, it indicates the signal was actively driven high to precharge the bus and then tri-stated. A turnaround cycle is required on all signals that may be driven by more than one agent. The turnaround cycle is required to avoid contention when one agent stops driving a signal and another agent begins driving the signal. This is indicated on the timing diagrams as two arrows pointing at each others' tail.

3.5.1 Enqueueing Requests

There are two ways to enqueue requests on the A.G.P. interface, the **SBA** port, or the **AD** bus. An A.G.P. master can use either mechanism to enqueue requests, and bit 9 of the Status register (see Section 6.1.9) indicates which mechanism it will use. When the **SBA** port is used to enqueue requests, the **AD** bus cannot be used. When the **AD** bus is used to enqueue requests, the **SBA** port cannot be used. The corelogic is required to support both mechanisms and this allows a master to choose which of the two mechanisms it will implement. Since the master is only allowed to use one of the two mechanisms, the corelogic is allowed to disable the other mechanism. Bit 9 of the corelogic's *Command* register is programmed to indicate which mechanism the master will use to enqueue requests.

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 **SBA** port. However, in order to keep pin cost down, it is only an 8-bit wide interface.

The **SBA** 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 corelogic. 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 that uses the **SBA** port has no need of the **PIPE#** signal which is used only to frame requests on the **AD** pins.

3.5.1.1 Address Demultiplexing Option

In order to transmit the complete A.G.P. access request across the 8-wire **SBA** port, the request is broken into three parts (optionally four parts): low order address bits and length, mid-order address bits and command, and high order address. These three parts are referred to as Type 1, Type 2, and Type 3 respectively. All Type commands are "sticky" except Type 1. Where *sticky* 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 eight **SBA** signals. Figure

3-4 is an A.G.P. Request and illustrates which bits go with which Type. The Types that have gray shading behind them are "sticky" in the corelogic and when a Type 1 is sent, the entire request is reconstructed and sent to the memory controller. Note that Type 2 includes address bits 16-23 and the four command bits.

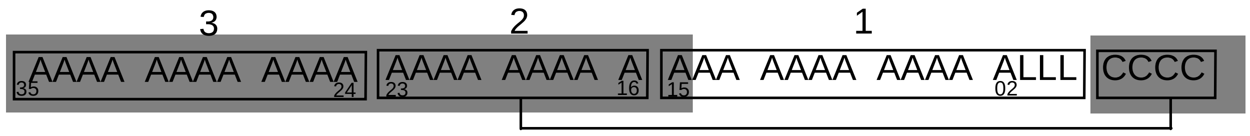


Figure 3-4: A.G.P. Request

The transmission of each of these three Types is accomplished by a separate **SBA** operation. Each operation on the **SBA** port delivers a total of 16 logical bits, in two 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. In 4x transfer mode, two complete transfers can complete in a single A.G.P. clock. When only a single transfer is required, the master must drive the NOP command during the second transfer.

When addresses greater than 36 bits are needed, a Type 4 command can be used if enabled. Bit 5 in the Command register determines if the extended address bits are supported or not. When bit 5 is set, the device supports up to address bit 47. When cleared (0), the extended address bits are not supported and the master is not allowed to initiate a Type 4 command. Type 4 command is the same as a Type 3 command except Type 4 presents address bits 36 through 47.

The **SBA** port always operates at the same transfer rate as the **AD** bus; either 1x, 2x, or 4x transfer mode as initialized in the A.G.P. Command register. This relationship keeps the 8-bit **SBA** port well matched in speed with data transfer on the **AD** bus, since the minimum data transfer size is 8 bytes (two **AD** ticks), and most A.G.P. access requests will only involve the low order address bits and length, requiring a single **SBA** operation (two **SBA** ticks).

Table 3-13 shows the definition and encoding of each of the **SBA** 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. target. Note that the first piece of data transferred includes the op code. For example, for the Length and Lower Address Bits encoding the 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 three length encoded bits.

Table 3-13: SBA 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 operating at 2x or 4x transfer mode, the *Bus Idle* encoding requires the full 16 bits.
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 **SBA** 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 **SBA** 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 CCRA 15 AAAA AAAA 23 16	***Command & Mid Address Bits:*** the A.G.P. bus command (**CCCC**[18]) and mid-order 9 address bits (**A[23::15]**) are transferred across the **SBA** 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". Note that the "R" indicates a reserved bit that must be driven by the master as a "0" and is ignored by the target.
110R AAAA 35 32 AAAA AAAA 31 24	***Upper Address Bits:*** the upper 12 address bits (**A[35::24]**) are transferred across the **SBA** 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 36-bit physical address space. The master drives **A[35::32]** to zero when its bit 5 of the A.G.P. Command register is cleared (0). When bit 5 is set (1), the master must actively drive these bits. The target ignores these bits when its corresponding bit 5 is cleared. Note: the 'R' indicates a reserved bit that must be driven as a "0" by the master and is ignored by the target.
1110 AAAA 39 36 AAAA AAAA 47 40	***Extended Address Bits:*** the extended 12 address bits (**A[47::36]**) are transferred across the **SBA** port; no memory access is initiated. This encoding is also referred to as a Type 4 sideband command. This command, when followed by the three previous commands (Type 3, Type 2, and Type 1) provides for memory access anywhere within a 48-bit physical address space. The master must actively drive all bits when this command is used. Note that bit 5 on the master's Command register must be set to use this command.
1111 **0***** **** ****	***reserved:*** must not be issued by an A.G.P. master and may be defined by Intel in the future. Any encoding when S[7:4] are 1111 and S3, S2, S1, or S0 is a 0 is considered reserved.

[18] The A.G.P. master when using the SBA port must treat the DAC command as a reserved command.

Note that only one **SBA** port command (Type 1) actually initiates a memory cycle; the other port commands (Type 2, Type 3, and Type 4) simply update respective "sticky" bits. There is no restriction on the relative ordering in which Type 1, 2, 3, or 4 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 has not yet occurred), the A.G.P. target may use an indeterminate address bits (A15- A31) and command (C3-C0) to access memory[19]. When the A.G.P. target receives a Type 1 command it takes the Type 1 information and combines it with previously stored Type 2 and Type 3 (and optionally Type 4) information to reconstruct a full address, command, and length information to initiate a memory access. Note that the **SBA** port only supports 48 bits of addressing while the **AD** bus supports all 64. When the **SBA** port is used, it is assumed by both the A.G.P. master and target that address bits 63 through 48 are zero.

The **SBA** 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, 3, and, when enabled, Type 4 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 and 4x transfer modes. In 2x mode, a single 16 bit NOP is all that is required to realign itself to **CLK**. In 4x mode, the master may be required to send two NOP commands before starting a new request to align itself with respect to **CLK**.

SBA Port Operation

Using the **SBA** Port to enqueue requests allows the **AD** bus to be used exclusively for data transfers. Table 3-14 lists the signals needed to enqueue requests using the **SBA** Port. Note that the **SBA** Port operates at the same rate as the data transfers on the **AD** bus. The minimum number of signals needed to enqueue requests on the **SBA** Port is eight which is when the data transfer rate is 1x. The maximum number of signals needed is 10 which occurs when the data transfer rate is 4x.

Table 3-14: SBA Signals

Data Transfer Rate	**Signals Required**
1x	**SBA[7::0]**
2x	**SBA[7::0]**, **SB_STB**
4x	**SBA[7::0]**, **SB_STB**, and **SB_STB#**

[19] When bit 5 is set in the Command register, Type 4 may also be indeterminate. When bit 5 is cleared, Type 4 "sticky" bit must not be used and the address sent to memory must have address bits 32-47 set to zero. In either case, bits 48-63 are assumed to be zero.

Request Enqueueing 1x Data Transfer Mode

Figure 3-5 illustrates the use of the **SBA** Port to enqueue requests. R1, R2, R3, or R4 in the figure could use Type 1, 2, 3, or 4 commands. A Type 4 command can only be used when enabled (see Sections 6.1.9 and 6.1.10 for details).

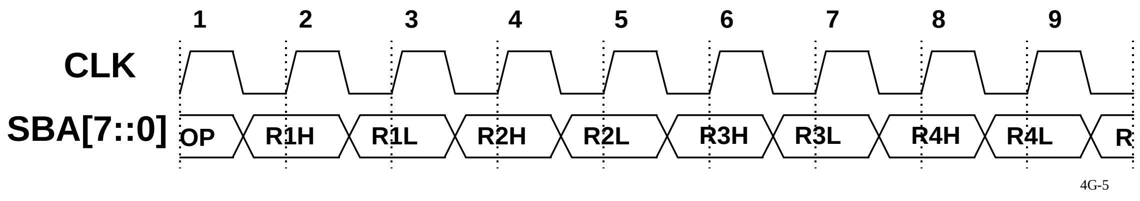


Figure 3-5: SBA Port Request - 1x

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 or 4x 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 of requests on the **AD** bus, the master does not request use of the port, but simply sends the request at any time (when a request slot is available). If a subsequent command is near a previous command, only the lower address bits and length need to be transferred. The target will use the previously sent 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 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, Type 3, or Type 4 command as described in Section 3.5.1.1.

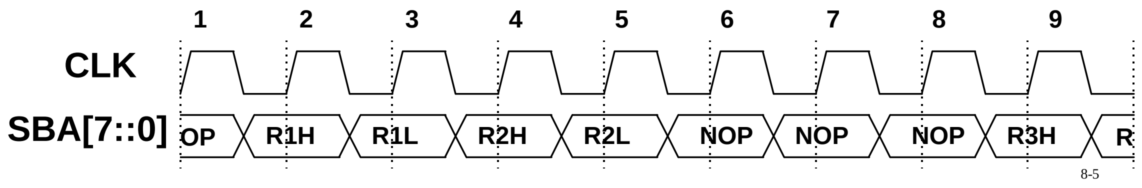


Figure 3-6: 1x Sideband Addressing

In Figure 3-6, the master sends the NOP encoding on clock 1 and sends the high bits of a Type x (1, 2, 3, or 4) on clocks 2, 4, and 9 and the low order bits on clocks 3, 5, and 10. The master sends NOPs on clocks 1, 6, 7, and 8 to indicate that the **SBA** Port does not contain a new request. There is no specific sequence in which Type 1, 2, 3, or 4 encodings are required to transfer across the **SBA** Port. In this figure, every non-NOP time could only be Type 1 or 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 the command in the Type 2 register of the target. A Type 3 encoding updates the upper address bits 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 (and, when enabled, Type 4 registers) with the Type 1 value and enqueue it to the memory controller.

Request Enqueueing 2x Data Transfer Mode

Figure 3-7 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.

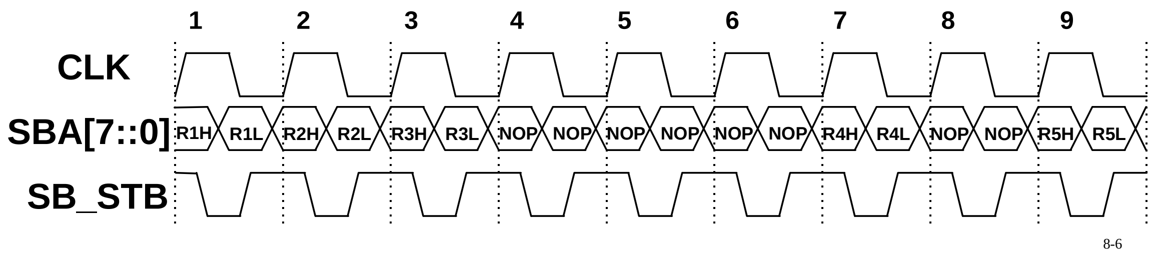


Figure 3-7: 2x Sideband Addressing

Notice that NOPs come in pairs since the full 16 bits are required to be transferred, unlike the 1x mode when only 8 bits were required to be transferred. The high portion of the command is transferred on the falling edge of **SB_STB** while the low portion is transferred on the rising edge. There is no relationship to commands being enqueued by **SB_STB** and **CLK** when the data transfer is at a level other than 1x.

Request Enqueueing 4x Data Transfer Mode

Figure 3-8 is the same as Figure 3-7 except the requests are enqueued at the 4x data transfer rate rather than the 2x transfer rate. In this case, the entire sequence was completed in five clocks instead of 10 clocks. In this mode, two strobes are required to transfer the data. In this mode, only the falling edge of the strobes are used to transfer data. A request is enqueued on each falling edge of the strobe, R1H is latched on the first falling edge of **SB_STB**, and the R1L is latched on the first falling edge of **SB_STB#**. Only a single Type 1 can be enqueued per **CLK**. For example, R1 or R2 can be a Type 1 but not both. However, R2 and R3 can both be Type 1.

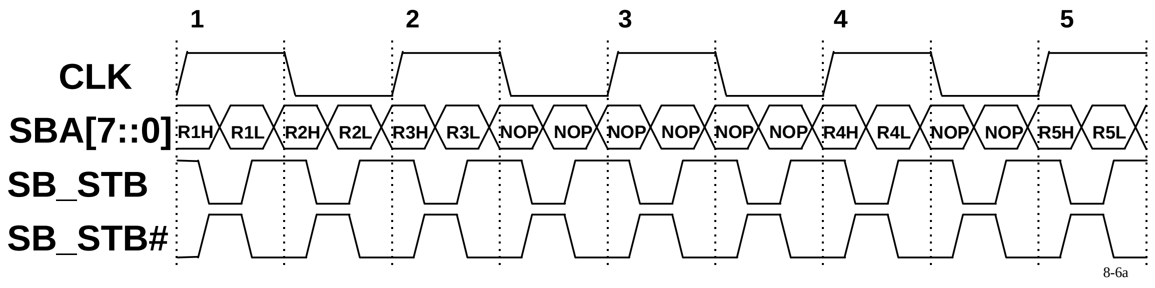


Figure 3-8: 4x Sideband Addressing

3.5.1.2 AD Bus

When implementing the **SBA** port is not feasible due to pin or performance constraints, enqueuing requests via the **AD** bus may be the preferred choice. A new request is enqueued on each clock in which **PIPE#** is asserted and is used in conjunction with the **AD** bus. Requests are only enqueued on the rising edge of **CLK** when **PIPE#** is asserted, regardless of the data transfer rate selected for data movement. The master indicates to the corelogic when the last request is enqueued by deasserting **REQ#** during the last clock in which a request is enqueued. The next clock the master is required to have **PIPE#** deasserted. The master is not allowed to insert waitstates while enqueueing requests.

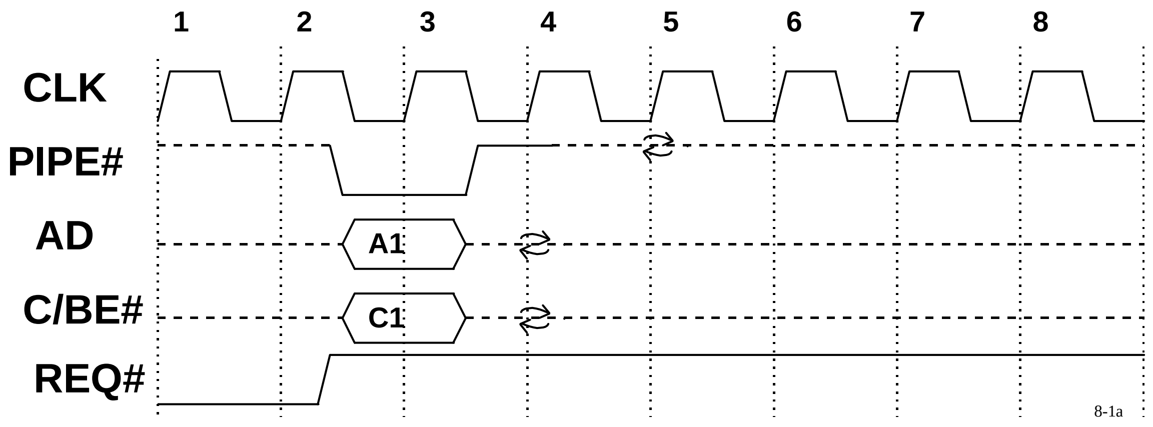


Figure 3-9: Single Address

Figure 3-9 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. How and when the arbiter grants the bus will be discussed in Section 3.6. A new request (address, command, and length) is 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.

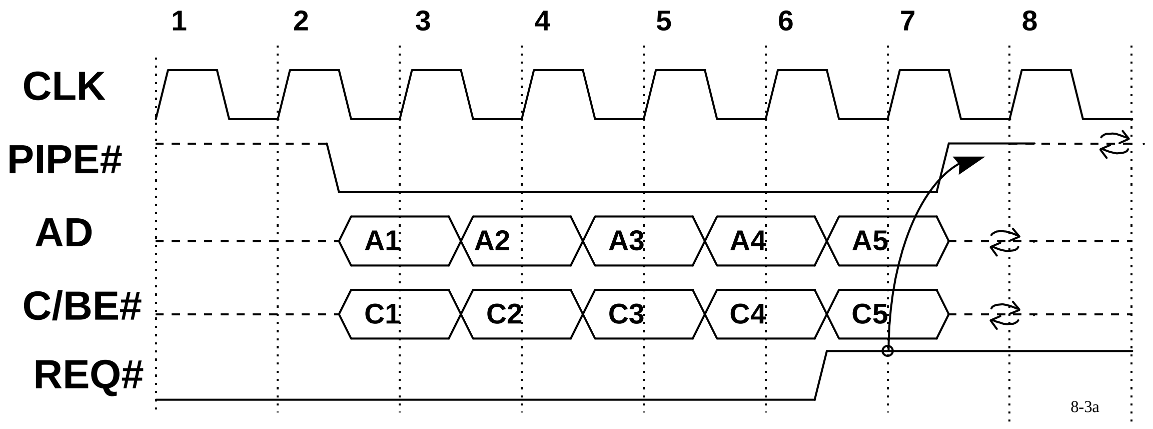


Figure 3-10: Multiple Addresses Enqueued, Maximum Delay by Master

Figure 3-10 illustrates a master that enqueues five requests. Note that **REQ#** remains asserted until clock 7 to indicate that the master desires to continue enqueueing requests and the current request is not the last one. When **REQ#** is deasserted with **PIPE#** asserted on clock 8, it indicates that the current request 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 corelogic.

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 clock following the last request, the master is required to tri-state the **AD** and **C/BE#** buses. Once the master has deasserted **REQ#**, it is not allowed to reassert it and continue asserting **PIPE#**. Once **REQ#** is deasserted, **PIPE#** must be deasserted on the next clock.

3.5.1.3 64-bit Requests on the AD Bus

The A.G.P. master can enqueue a request using the **AD** bus by asserting **PIPE#** when initiating the transaction. When the master initiates a 32-bit request, it enqueues a new request on each clock in which **PIPE#** is asserted. The last request is enqueued when **REQ#** is deasserted and **PIPE#** is asserted. When bit 5 of the master's A.G.P. Command register is set, it is enabled to make 64-bit requests. When the A.G.P. master enqueues a request that uses a 64 bit address, it is required to use two clocks to transfer the request. During the first clock, the master provides the lower address bits (A31-A2) and the length encoding (A2-A0) when the command is DAC (**C/BE[3::0]#** is 1101). The following clock, the master provides the upper address bits (A63-A32) and the actual command on **C/BE[3::0]#**. **REQ#** is deasserted during the final clock of the last request to be enqueued. Note that the A.G.P. master is not allowed to use DAC to enqueue a request when the upper address bits are all zero. In other words, the master must use a single address cycle (SAC) when accessing a location that resides within the lower 4 GB of memory and is only allowed to use a DAC when accessing addresses above that boundary.

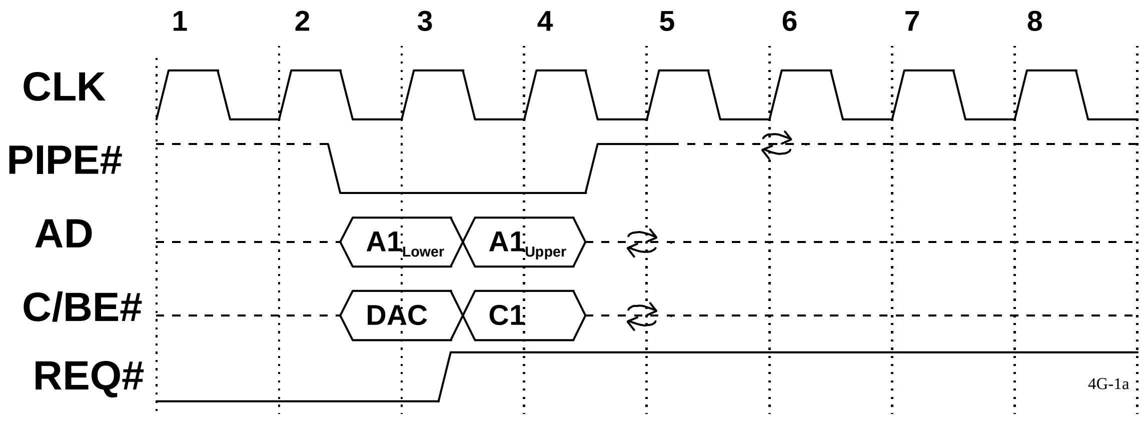


Figure 3-11: Single 64-bit Request

Figure 3-11 illustrates a single request being enqueued that uses a 64 bit address or DAC command. The transaction is the same as Figure 3-9 except the DAC command is used during the first clock in which **PIPE#** is asserted and it remains asserted for two clocks. In this case, **PIPE#** is asserted for two clocks even though only a single address is enqueued. Notice that **REQ#** is deasserted during the second address phase of the transaction and not during the initial clock of the request. The transaction appears to have enqueued two requests, but since the first command was DAC, two clocks are used to enqueue the complete request.

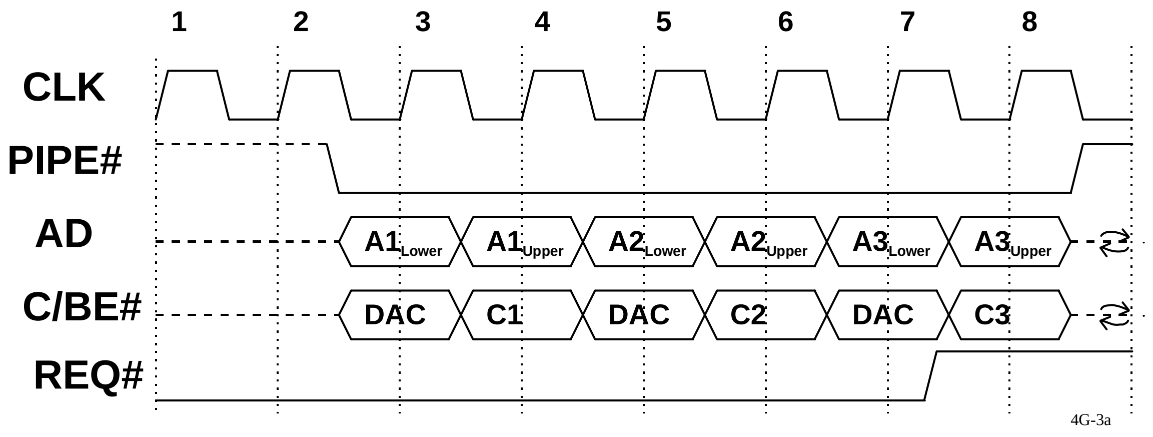


Figure 3-12: Multiple 64-bit Requests

Figure 3-12 illustrates multiple 64-bit requests being enqueued during a single request transaction. Notice that all these requests are using the DAC and require two clocks for each request to be transferred. The target (and arbiter) know that the last request is enqueued on clock 8 because **REQ#** is deasserted when **PIPE#** is asserted. The master always provides the lower part of the address and the length encoding on the first clock and the upper bits on the subsequent clock. The master is not allowed to insert or delay a request once **PIPE#** is asserted.

The A.G.P. master is allowed to mix SAC and DAC within the same transaction on the interface. An A.G.P. master is allowed to enqueue both requests that access data above and below the 4 GB boundary in the same bus transaction.

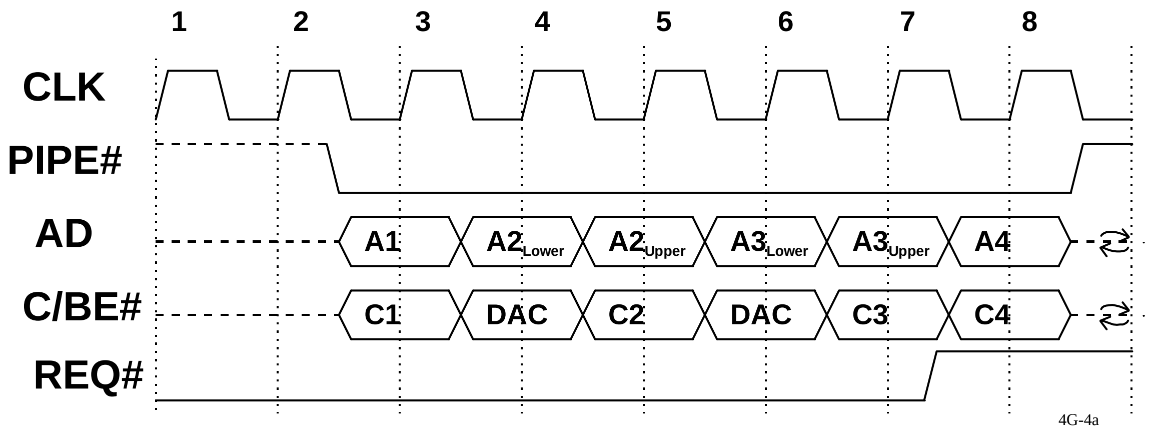


Figure 3-13: 32-bit and 64-bit Requests

Figure 3-13 illustrates SAC and DAC requests being enqueued during the same transaction. In this case, the first and fourth request use the SAC while the second and third use the DAC command. For this transaction, two transactions address data in the lower 4 GB of the address space and the other two address data above the 4 GB boundary. Like all requests that are enqueued using the **AD** bus, **REQ#** is deasserted during the final clock of the request transaction. One request slot is consumed when either a DAC or SAC is enqueued.

3.5.2 Flow Control

3.5.2.1 Address Flow Control

Address flow control for an A.G.P. Request is source controlled. This means that the A.G.P. 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. target's Status register (see Section 6.1.10) 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. master Status register. 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. target's request queue will never overflow.

The A.G.P. 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 transfer across the bus. Since a Flush command is treated like an 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.

3.5.2.2 Data Flow Control

Flow control on A.G.P. is different than on PCI. On PCI, the master and target may delay transferring 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 waitstates. 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 an initial block, which means that the entire transaction completes within four clocks. For transactions that require more than four clocks to complete, they are comprised of both an initial block and one or more subsequent blocks. A block is defined as the amount of data that can be transferred in four A.G.P. clocks which must be 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. can occur at three different times of a transaction after it has been requested:.

1. Before the arbiter indicates that the data transfer will begin.
2. After the arbiter indicates the data movement will occur but before the data actually starts transferring.
3. Between the completion of the current block and the start of a subsequent block.

For example, while the initial block is being transferred, flow control can occur that will cause a subsequent block of data to be delayed. The stream type and whether the transaction is a read or write will determine whether one or more of the three options are available. Flow control also is different for a master than for a target. Table 3-15 is for the master while Table 3-16 is for the target or corelogic. The first row in the tables lists the transaction type while the first column lists the times when flow control can occur.

Table 3-15: A.G.P. Master Flow Control Conditions

Transaction Type	LP Read	HP Read	A.G.P. Write	FW[20]
Before Transfer	**RBF#**	None	None	**WBF#**
Initial Block	None	None	**IRDY#**	None
Subsequent Block	**IRDY#**	**IRDY#**	None	**TRDY#/STOP#**

Table 3-15 lists the flow control conditions for a master; each column in the table will be discussed separately. In the table, when a signal name is listed it indicates that the master can use that signal to cause data not to transfer. When "None" is entered in the table, then the master has no mechanism to stall the data from transferring.

Column 2 lists whether the master can cause the transfer of LP Read data to be delayed or not.

Row 2 is the time when the master can prevent the arbiter from initiating the return of previously requested LP Read data to the master. The table indicates that the master can assert **RBF#** to prevent the arbiter from initiating the return of LP Read data. Note, once the arbiter has indicated that LP Read data is being returned, **GNT#** asserted and **ST[2::0]** is "000", **RBF#** has no affect on the return of data for the current transaction and only applies to the return of data for the next LP Read transaction.

Row 3 is the time when the arbiter has indicated that the next transaction on the interface will be the return of LP Read data for the initial block of data. In this case, the master has no mechanism to delay the data from being transferred.

Row 4 is the time when the transaction will not complete during the current transfer block but requires at least one more subsequent block to complete. For example, the requested data is 24 bytes in length and the transfer rate is 1x. This transfer requires the initial block (16 bytes) and an additional two clocks (8 bytes) of the subsequent block to complete the transfer. When this occurs, the master is allowed to insert waitstates between the completion of the current block and the start of a subsequent block. The number of clocks the master is allowed to insert waitstates is not defined. The master deasserts **IRDY#** to cause the return of a subsequent block of data to be delayed.

Column 3 lists whether the master can cause the transfer of HP Read data to be delayed or not.

Row 2 is the time when the master can prevent the arbiter from initiating the return of previously requested HP Read data to the master. The entry in the table indicates that there is no mechanism for the master to delay the return of HP Read data. The reason for this behavior is that HP accesses can stall the CPU; and, therefore, it is reasonable to require the master to accept the data when the corelogic can provide it.

Row 3 is the time when the arbiter has indicated that the next transaction on the interface will be the return of HP Read data for the initial block of data. In this case, the master has no mechanism to delay the data from being transferred.

Row 4 is the time when the transaction will not complete during the current transfer block. This is the same as the LP Read flow control for subsequent block.

Column 4 lists whether the master can cause the transfer of A.G.P. write data to be delayed or not. Since there is no difference between HP and LP write data transfers, they are combined in the table.

Row 2 is the time when the master can prevent the arbiter from initiating the return of previously requested write data to the corelogic. The table indicates that the master has no mechanism to delay the transfer of write data once the master has enqueued the request.

[20] This transaction type is a modified PCI operation with flow control over blocks and not data phases.

Row 3 is the time when the arbiter has indicated that the next transaction on the interface will be the transfer of write data for the initial block of data. In this case, the master can delay providing the write data by one clock by keeping **IRDY#** deasserted. Note that when the arbiter has indicated that the next transaction on the interface is write data, the master must provide the data within two clocks of when the bus becomes available to complete the request.

Row 4 is the time when the transaction will not complete during the current transfer block. The master has no mechanism to stall a subsequent block of write data. This is because the master may choose the transfer size of the write request; and, therefore, there is no reason to allow the master to stall the transfer.

Column 5 lists whether the master can delay the transfer of PCI write data in an accelerated way (FW) from the corelogic to the A.G.P. master acting as a PCI target. See Section 3.5.3.5 for details of the behavior of an FW transaction.

Row 2 is the time when the master can prevent the arbiter from initiating the transfer of memory write data from the corelogic to the A.G.P. master. This data is typically initiated by the CPU. Since the master did not request the data, the write buffers of the master may be full and initiating the request just to have the transaction (PCI) terminated with retry wastes bus bandwidth. Therefore, the master can assert **WBF#** to indicate to the corelogic that it cannot accept any FW data.

Row 3 is the time when the arbiter has indicated internally to the corelogic that FW data is the next transaction on the interface. Since the master does not know that the transaction is coming, it must decode the PCI transaction to determine if it is the target of the request. The A.G.P. master must accept the first four clocks worth of data regardless of whether the device has completed the address decode or not.

Row 4 is the time when the transaction will not complete during the current transfer block. The master can insert waitstates between the completion of the current block and the start of a subsequent block. The A.G.P. master is also allowed to use PCI termination of *Disconnect* at any time after the initial block.

Table 3-16: A.G.P. Target Flow Control Conditions

Transaction Type	**LP Read**	**HP Read**	**A.G.P. Write**	**FW**
Before Transfer	**GNT#/ST[2::0]**	**GNT#/ST[2::0]**	**GNT#/ST[2::0]**	**GNT#/ST[2::0]**
Initial Block	**TRDY#**	**TRDY#**	None	**IRDY#**
Subsequent Block	**TRDY#**	**TRDY#**	**TRDY#**	**IRDY#**

Table 3-16 lists the flow control conditions for the corelogic and each column will be discussed. In the table when a signal is listed, the target can use that signal to cause data not to transfer. When "None" is entered in the table, the target has no mechanism to stall the data from transferring.

Row 2 is the time when the target can delay the data movement regardless of the transaction type. Since the corelogic is a combination of A.G.P. target and A.G.P. arbiter, it can use the arbitration signals to delay the transfer of data when appropriate. The corelogic can choose when the data traverses the bus subject to the availability of the of A.G.P. master's buffering (**RBF#** and **WBF#**) since it is providing or accepting data. The arbiter uses **GNT#** and **ST[2::0]** to delay when data will transfer.

Columns 2 and 3 are the return of read data and are the same for the corelogic.

Row 3 is the time when the corelogic can delay the return of the initial block of read data. The entries in the table indicate that the corelogic is allowed to delay the return of read data for 1 clock when the bus is available to move the data. The corelogic delays the return of read data by deasserting **TRDY#** for one clock.

Row 4 is the time between the transfer of the current block and the start of a subsequent block. The corelogic is allowed to delay the return of a subsequent block of read data. The amount of delay that the corelogic can insert between blocks is undefined, and the corelogic inserts the delay by deasserting **TRDY#**.

Column 4 is the movement of previously enqueued write data.

Row 3 is the time when the arbiter has indicated that write data will transfer and before the transaction actually starts. Per the table, the corelogic has no mechanism to delay the acceptance of the data. Therefore, if the corelogic is not capable of accepting the data, it must not allow the arbiter to indicate that write data is to be provided.

Row 4 is the time between the transfer of the current block and the start of a subsequent block. The corelogic is allowed to delay the acceptance of a subsequent block of write data. The amount of delay that the corelogic can insert between blocks is undefined and the corelogic inserts the delay by deasserting **TRDY#**.

Column 5 is the movement of PCI write data from the corelogic to the A.G.P. master that was not requested by the A.G.P. master. Note that FW transactions are only supported when enabled and the data transfer rate is greater then 1x. When the transfer rate is 1x or FW is disabled, PCI memory writes from the corelogic to the A.G.P. master must be done using PCI protocol and follow the PCI rules.

Row 3 is the time when the arbiter has indicated that write data will transfer and before the transaction actually starts. Per the table the corelogic can delay providing the data to the A.G.P. master by deasserting **IRDY#**; however, the delay is limited to a single clock.

Row 4 is the time between the transfer of the current block and the start of a subsequent block. The corelogic is allowed to delay providing a subsequent block of write data. The amount of delay that the corelogic can insert between blocks is undefined and the corelogic inserts the delay by deasserting **IRDY#**.

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 waitstate at any TP of a read transaction may invalidate the latency guarantee of **all** outstanding high priority requests. If the master inserts a waitstate at a TP, it cannot make any assumptions about what impact the waitstate will have on the latency guarantee. For instance, inserting five waitstates at a TP of read A (high or low priority) does not mean that outstanding high priority read request B will complete in x + five clocks (where x is the latency guarantee provided by the corelogic). The target must include any potential **TRDY#** throttle point waitstates 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.

3.5.2.2.1 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 four clocks or less, the master is required to accept the entire transaction without waitstates regardless of the data transfer mode. When the transaction requires more than four clocks to complete, the master is allowed to insert waitstates after each four clocks in which data is transferred.

For 1x data transfers, the master must accept the entire transaction without waitstates 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 be able to 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.

Table 3-17 shows the minimum amount of buffering required in the master when **RBF#** is deasserted. This table only applies to 2x data transfer mode[21].

Table 3-17: Data Buffering for 2x Transfers

1st Read Transaction	2nd Read Transaction	Buffer Space Needed to Deassert RBF#
8 bytes	$8 \leq n \leq 32$ 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 3-18 shows the minimum amount of buffering required in the master when **RBF#** is deasserted. This table only applies to 4x data transfer mode.

[21] 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.

Table 3-18: Data Buffering for 4x Transfers

1st Read Transaction	2nd Read Transaction	Buffer Space Needed to Deassert RBF#
8 bytes	$8 \leq n \leq 64$ bytes	8 + n bytes
8 bytes	n > 64 bytes	72 bytes
16 bytes	$16 \leq n \leq 64$ bytes	16+n bytes
16 bytes	n > 64 bytes	80 bytes
24 bytes	don't care	24 bytes
32 bytes	don't care	32 bytes
64 bytes	don't care	64 bytes
> 64 bytes	don't care	64 bytes

If the master cannot accept the above transaction(s), it asserts **RBF#**. The A.G.P. arbiter will not assert a subsequent grant 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 four clocks worth of data when the next transfer is an LP Read data. The amount of data buffering required for one full block of data is dependent on the transfer mode. For the 2x transfer mode, one block is 32 bytes, and, for the 4x transfer mode, it is 64 bytes.

Note: The A.G.P. master has many implementation alternatives that can be predicated by buffer budget and complexity. For example, the A.G.P. 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. master restricts itself to 8 and 16 byte low priority read transactions, **RBF#** can be deasserted whenever 24 bytes of buffering are available when in 2x transfer mode. An A.G.P. 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 and 80 bytes for 4x 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 3-14 illustrates the enqueuing of two grants before the arbiter detects that **RBF#** is asserted.

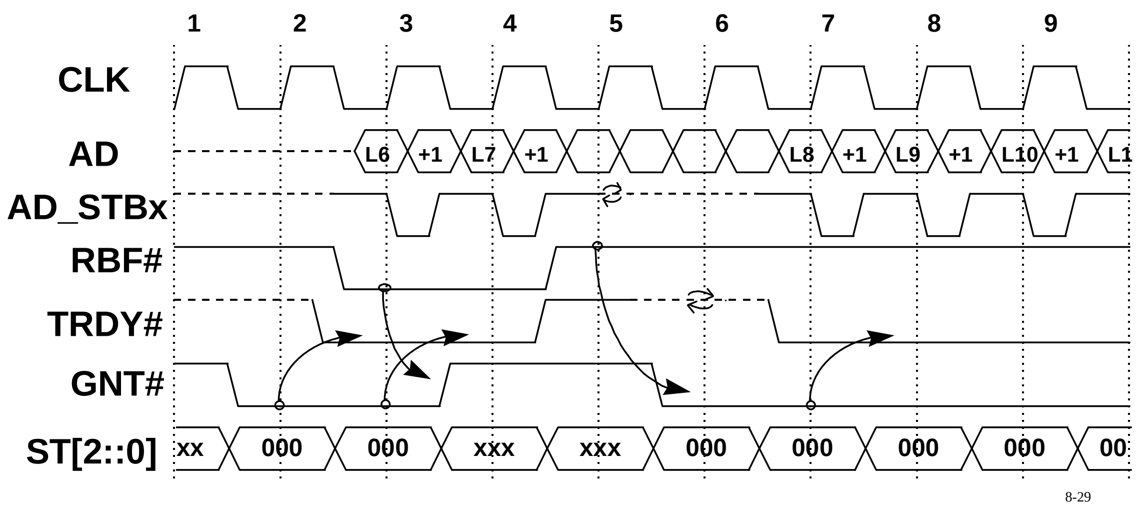


Figure 3-14: Maximum Number of GNT#s Queued Before the Assertion of RBF#

Since **RBF#** is deasserted on clock 1, the A.G.P. 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. 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 LP read 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 Section 3.6).

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 that **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 four clocks. When a high priority read request requires more than four 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. **RBF#** does not apply to high priority read data and **IRDY#** cannot 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 waitstates 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.

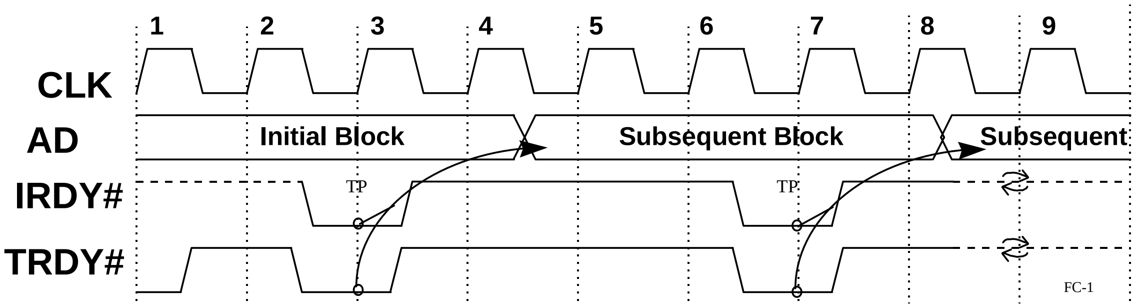


Figure 3-15: Read Transaction with TP with No Waitstate

Figure 3-15 illustrates the basic elements of an A.G.P. read transaction. The transaction was initiated sometime before clock 1 and the corelogic initiates the transaction by asserting **TRDY#** on clock 1 indicating that the initial block of data is starting to transfer. (The figure assumes that **RBF#** is deasserted.) In this figure, the initial block of data is transferred on clocks 1 through 4. The first opportunity at which either the corelogic or the master is allowed to request a waitstate is on clock 3 and is called the initial TP. Both agents are ready to continue the transaction on clock 5 since the TP completes on clock 3 when both **IRDY#** and **TRDY#** are asserted.

A new throttle point occurs four clocks after the previous TP completes. The initial TP occurs two clocks after the initial block starts transferring. **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#** and **TRDY#** must be actively driven during each TP. After a TP and before the next TP, **xRDY#** can be actively driven or tri-stated. When actively driven, the state can be asserted or deasserted. When tri-stated, the last agent to actively drive the signal is required to actively deassert it before tri-stating. When the transaction requires more than four clocks to complete, **xRDY#** is allowed to be actively driven before the first TP.

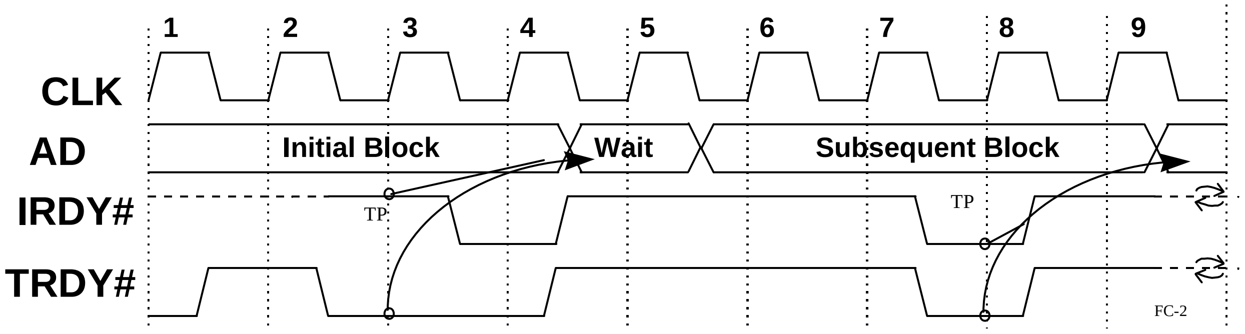


Figure 3-16: Read Transaction with TP with a Waitstate

Figure 3-16 is the same as Figure 3-15, except that the A.G.P. master inserts one waitstate at the first TP. Since the first TP does not complete until clock 4, the subsequent block does not begin to transfer data until clock 6. The second TP completes with no delay and is the same as in the previous figure. Because the TP does not complete on clock 3, a waitstate occurs on the **AD** bus on clock 5. If the master had continued to keep **IRDY#** deasserted on clock 4, a second waitstate would have occurred on clock 6.

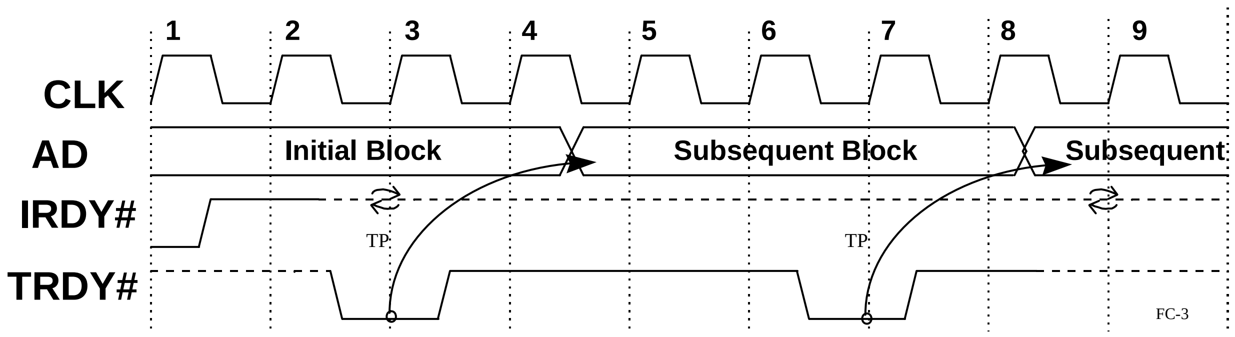


Figure 3-17: Write Transaction with TP with No Waitstate

Figure 3-17 shows an A.G.P. Write data transaction. In this case, the A.G.P. master asserts **IRDY#** to indicate that it is providing the write data. Notice that **IRDY#** is deasserted on clock 2 and is then tri-stated. The A.G.P. master is not allowed to cause the delay of write data once the transaction has been initiated. Therefore, for Write transactions, only the target is involved in TPs. In this case, the corelogic determines when subsequent blocks of write data are transferred. In this figure, the corelogic is ready and completes the TPs at the earliest time possible. Note that the corelogic is not required to deassert **TRDY#** after the first TP on clock 3, but could continue to drive **TRDY#** asserted. Notice on clock 8, that the corelogic deasserted and tri-stated **TRDY#**. This could mean that the transaction will complete in the next block, or the corelogic simply deasserted and tri-stated **TRDY#**. It could re-assert **TRDY#** in clock 10 if the burst continued.

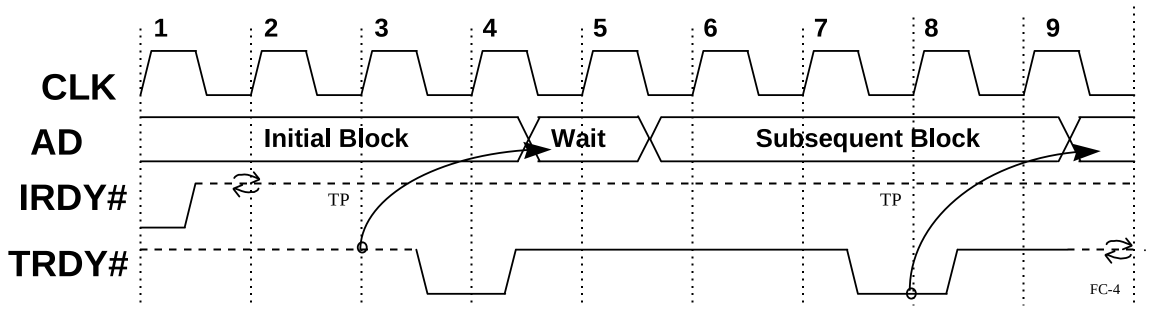


Figure 3-18: Write Transaction with TP with a Waitstate

Figure 3-18 is the same as Figure 3-17 except in this case the corelogic inserted a waitstate at the initial TP which occurs on clock 3. Since **TRDY#** was not asserted until clock 4, a waitstate is inserted on the **AD** bus on clock 5. What the A.G.P. master drives on the **AD** bus during the "wait" time does not matter. If it wanted to, the A.G.P. master could continue to drive the previous data, drive dummy data, or drive the initial data of the next block during the wait time. However, two clocks after **TRDY#** is asserted, the master must provide the subsequent block of data with no delays.

3.5.2.2.2 Write Data Flow Control

Initial Target Flow Control

The A.G.P. 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 3-19 illustrates the maximum delay which a master can introduce when providing write data. **IRDY#** must either be asserted on clock 3 (the earliest in which 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 waitstates. Since the master is not allowed to insert waitstates on a 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. When the transfer requires a subsequent block, the A.G.P. master is required to actively drive **IRDY#** during the entire TP. Once the last TP completes, the master must deasserted and tri-stated **IRDY#**.

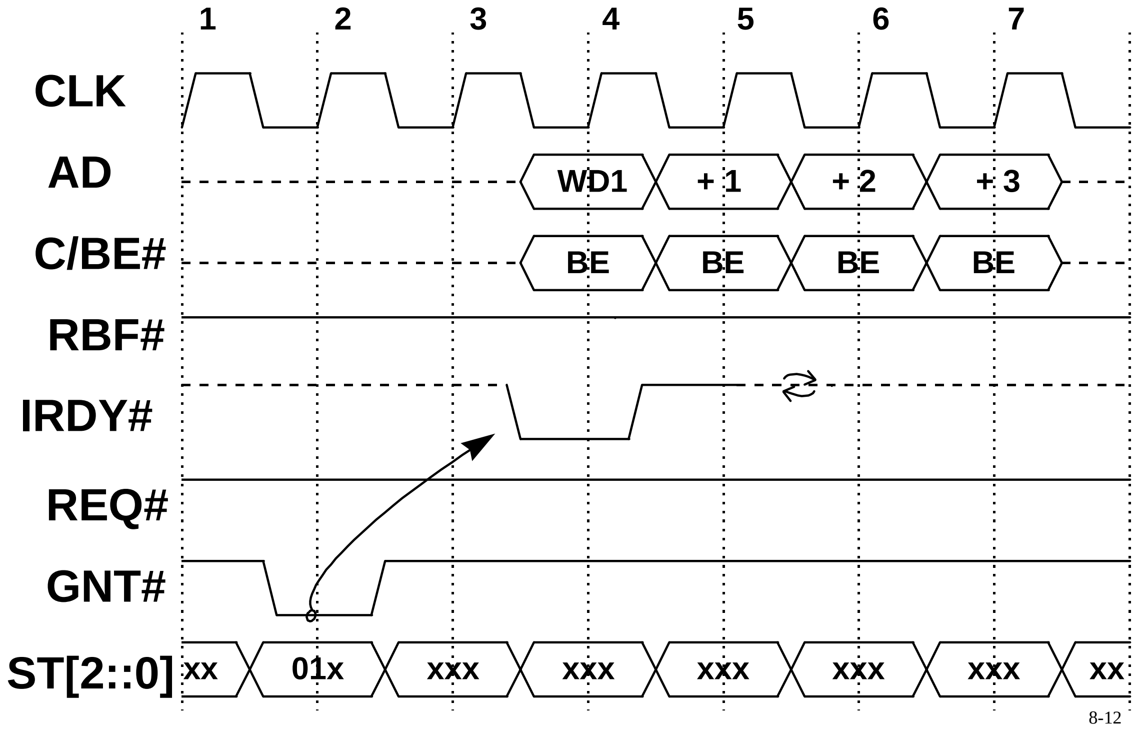


Figure 3-19: Maximum Delay by Master on Write Data

Throttling

Since the master is aware of the quantity of data it wants to send and can generate a smaller write request if necessary, thus 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 between blocks. When the target wants to throttle the transfer of a subsequent block of write data, it must have **TRDY#** deasserted at the TP 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 TP. **TRDY#** is meaningless (it may be asserted, deasserted, or tri-stated[22]) between TPs 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 TP of a transaction that takes less than or a multiple of a block. For example, if fewer than four clocks are required to complete the transaction, then the next TP does not occur. In Figure 3-20, the first TP occurs on clock 4 and since the transaction completes before clock 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.

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

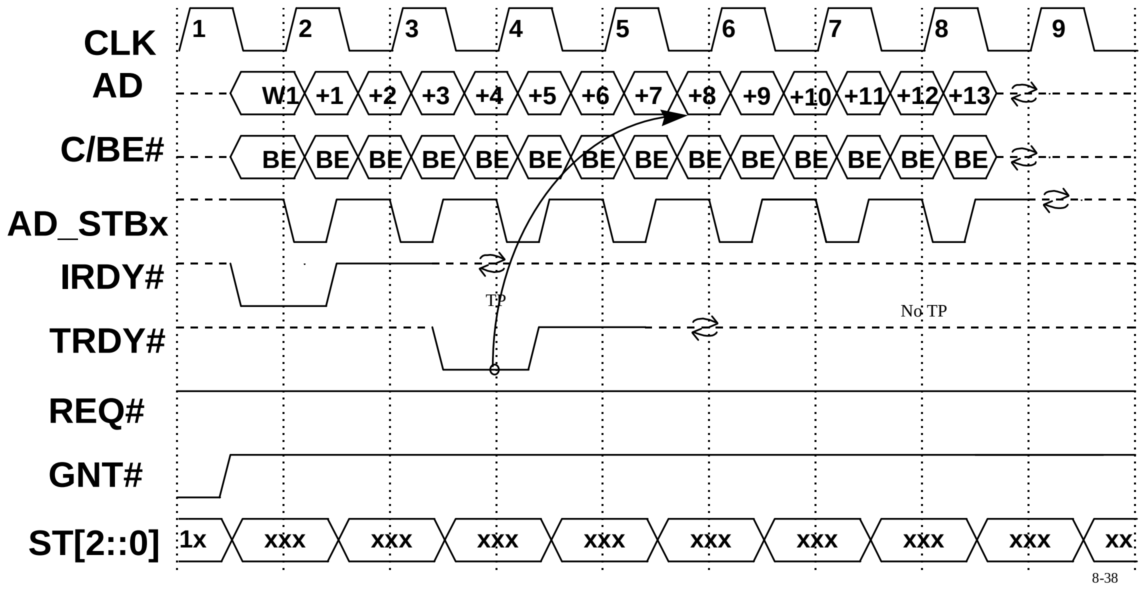


Figure 3-20: Write Data with One TP

One and Two Clock Rule for IRDY# and TRDY#

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

For initial read data, **TRDY#** must be asserted for one clock by the target one or two clocks after **GNT#** is sampled asserted when the **AD** bus is free. The target cannot 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.

3.5.2.3 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 cannot change its mind. Outside of the TP, the state of **xRDY#** is meaningless.

3.5.3 Data Transactions

As described earlier, data transfers across the interface as an independent transaction from the request that initiated the data movement. The following sections will discuss the movement of data with transfer rates of 1x, 2x, and 4x. In addition to the discussion of the transfer rate, the discussion will include basic read and write transfers. The pipeline A.G.P. transactions are always initiated by the A.G.P. master and not the corelogic. (However, the corelogic does initiate the data movement of the transaction in response to the master's request.) A discussion about an enhanced unsolicited data transfer initiated by the corelogic to the A.G.P. master (acting as a PCI target) will follow and is called Fast Write (FW) data transfer mode. FW mode is limited to PCI memory write commands.

3.5.3.1 1x Data Transfers

Data transfers between the A.G.P. master and the A.G.P. target in the 1x transfer mode use **CLK** to qualify both the control signals and the data. The transfer of data is initiated by the A.G.P. master making a request to the corelogic, while the corelogic initiates the actual data transfer

Figure 3-21 illustrates the return 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"; for high priority read data, the **ST** encoding would be "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 which type of read data being returned, the **ST** encodings will be either "000" or "001".

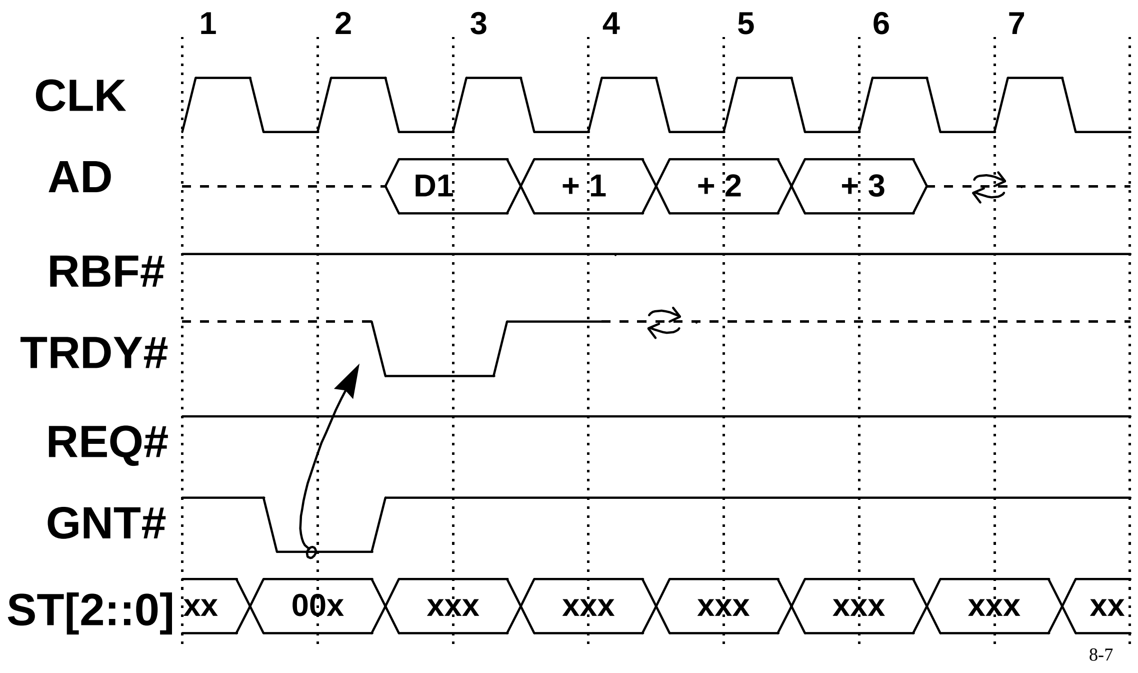


Figure 3-21: Minimum Delay by Target of Read Transaction

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 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. Once **TRDY#** is asserted and the entire transaction will complete within four clocks, no waitstates can be inserted. Notice that **TRDY#** is a single clock pulse and that there is no **IRDY#** handshake as is done on the PCI bus. When the transfer size of the read data can complete within four clocks, neither the master nor target is 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. 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 the PCI specification.

Figure 3-21 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.

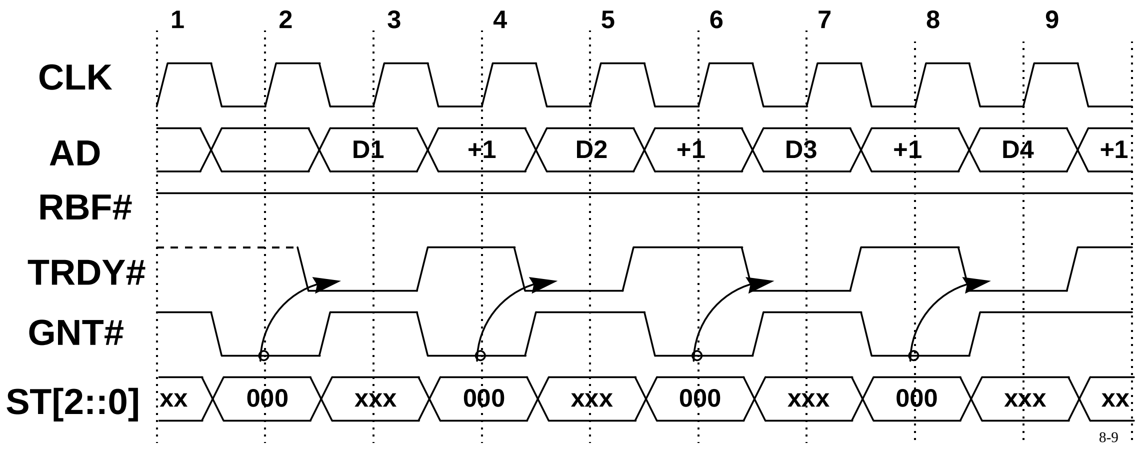


Figure 3-22: Minimum Delay on Back-to-Back Read Data

Figure 3-22 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.

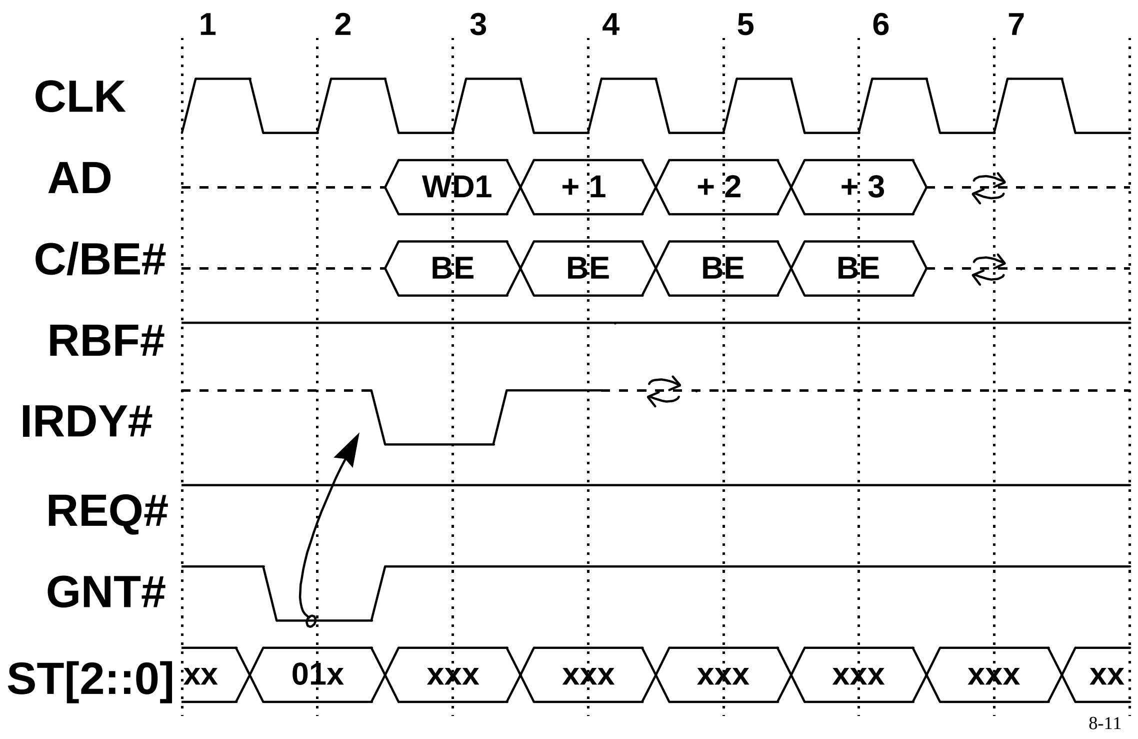


Figure 3-23: Master Does Not Delay Providing Write Data

Figure 3-23 shows a basic write data transfer. The arbiter indicates to the master that write data should be provided to the corelogic by the assertion of **GNT#** and **ST[2::0]** being “010” or “011”. The first being a low priority write data and the second being 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 two 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 a single clock 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 at 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 four 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 the **ST** bus indicate a write data transfer.

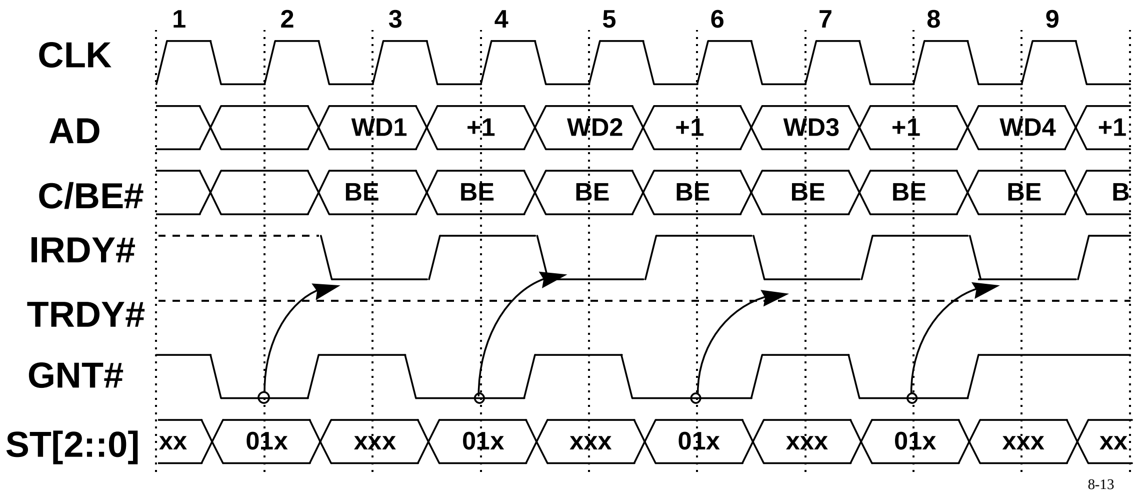


Figure 3-24: Back-to-Back Write Data Transfers - No Delay

Figure 3-24 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 corelogic. 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.

3.5.3.2 2x Data Transfers

This section discusses 2x data transfers. Basically, 2x clocking 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 read data transfer will be discussed; then a write transfer will be discussed. The control signals are identical to the 1x read signals, except that **AD_STBx** has been added when data is transferred at 8 bytes per **CLK** period. **AD_STBx** represents **AD_STB0** and **AD_STB1** which are used by the 2x interface logic to determine when valid data is present on the **AD** bus. The falling edge of **AD_STBx** is used by the receiving agent to latch the first four bytes of data residing on the **AD** bus and the rising edge is used to latch the second 4 bytes of the transaction. Note that **CLK** is not used to latch data but is used to qualify the control signals.

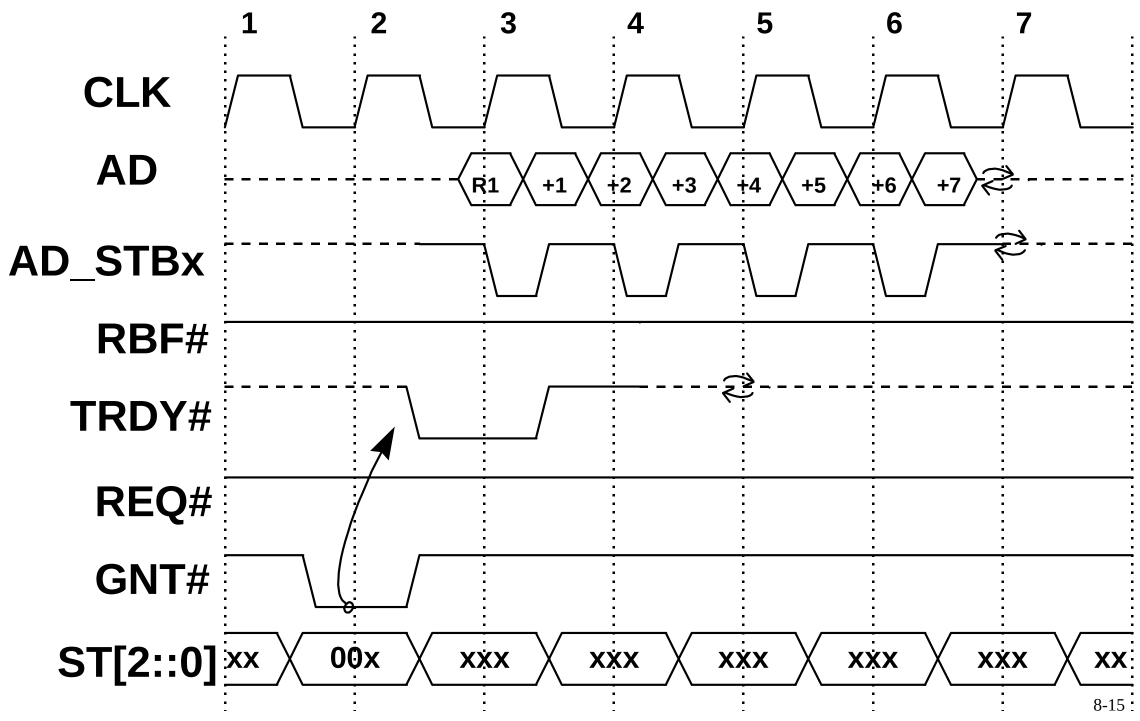


Figure 3-25: 2x Read Data - No Delay

Figure 3-25 is the same as Figure 3-21 except that 16 bytes are transferred in four clocks; while in this figure, 32 bytes are transferred during the same four clocks. The control logic (**TRDY#** in this case) indicates when data can be used by the internal consumer of the data.

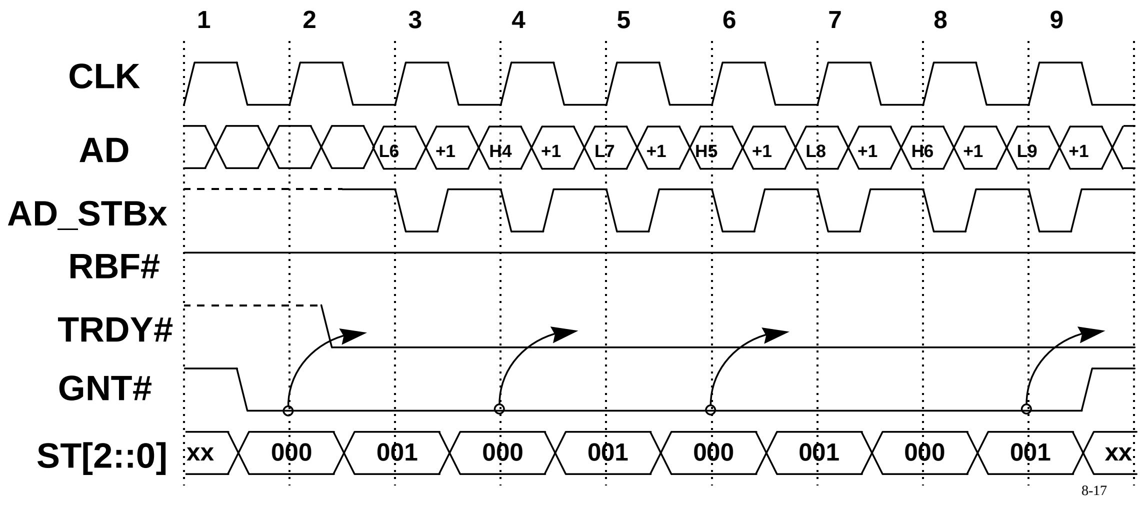


Figure 3-26: 2x Back-to-Back Read Data - No Delay

Figure 3-26 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 high priority 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.

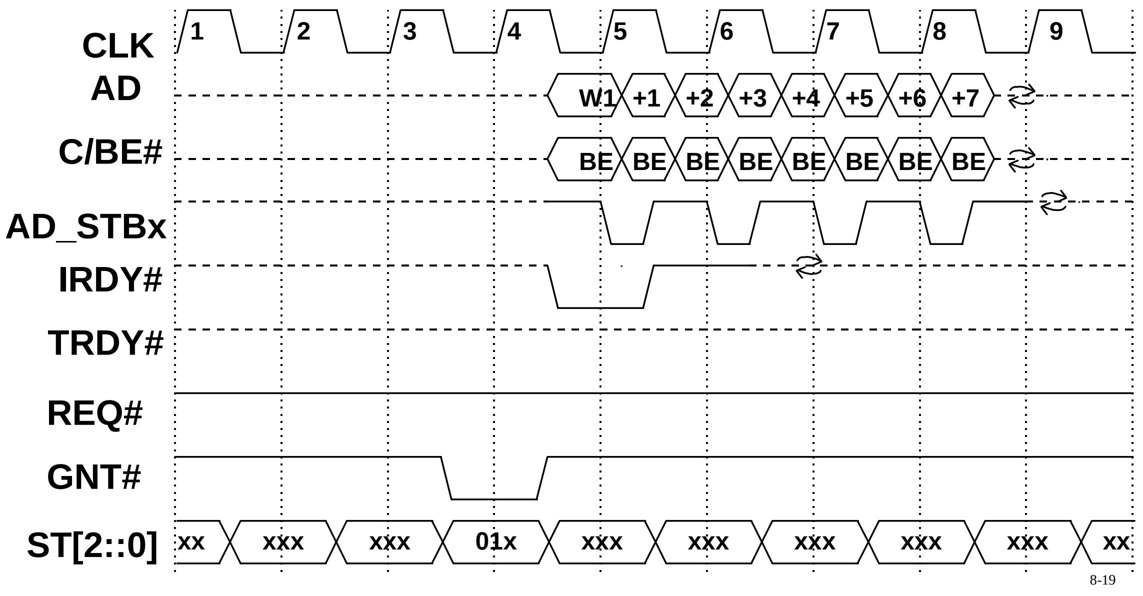


Figure 3-27: 2x Basic Write - No Delay

Figure 3-27 is a basic write transaction that transfers data at the 2x rate. This figure is the same as Figure 3-23 (1x basic write). There is no difference in the control signals; and only more data is moved. The normal control signals determine when data is valid. This diagram shows 32 bytes of data being moved in the same time as 16 bytes are moved in Figure 3-23.

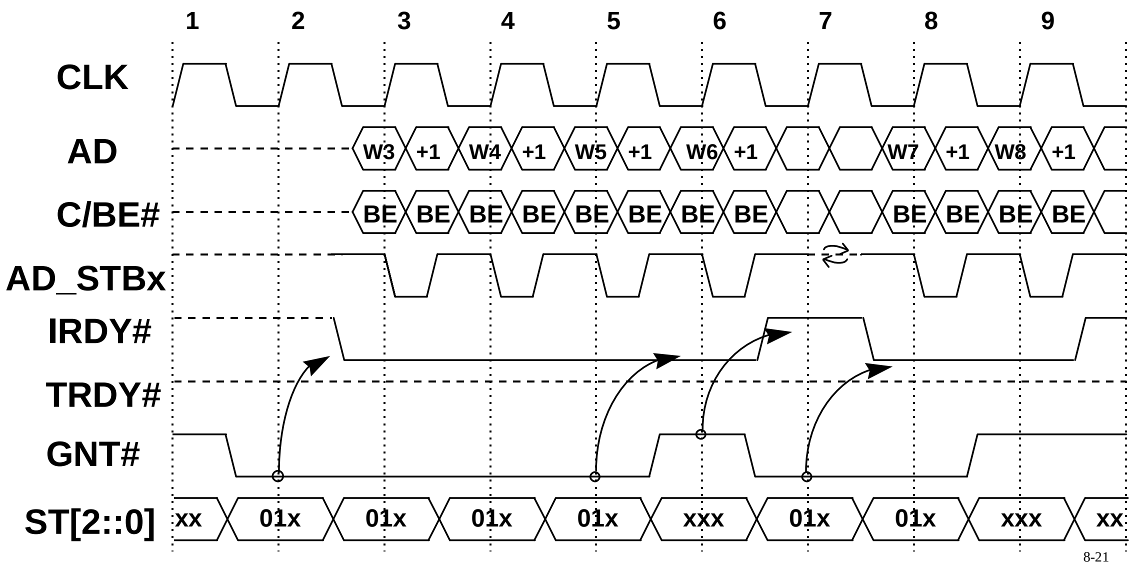


Figure 3-28: QuadWord Writes Back-to-Back - No Delay

Figure 3-28 illustrates multiple 8 byte write operations while previous diagrams illustrated 32 byte transfers. 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 started on each **CLK**, **IRDY#** is only deasserted on clock 7.

3.5.3.3 Relationship Between xRDY# and AD_STBx

All 2x diagrams in this section and Appendix B (except Figure 3-29) are drawn from the worst case timings point of view of the receiver. Therefore, it appears that data is transferred the clock after **xRDY#** is asserted; when, per Section 4.1.2.9, the data can actually occur the clock in which **xRDY#** is asserted.

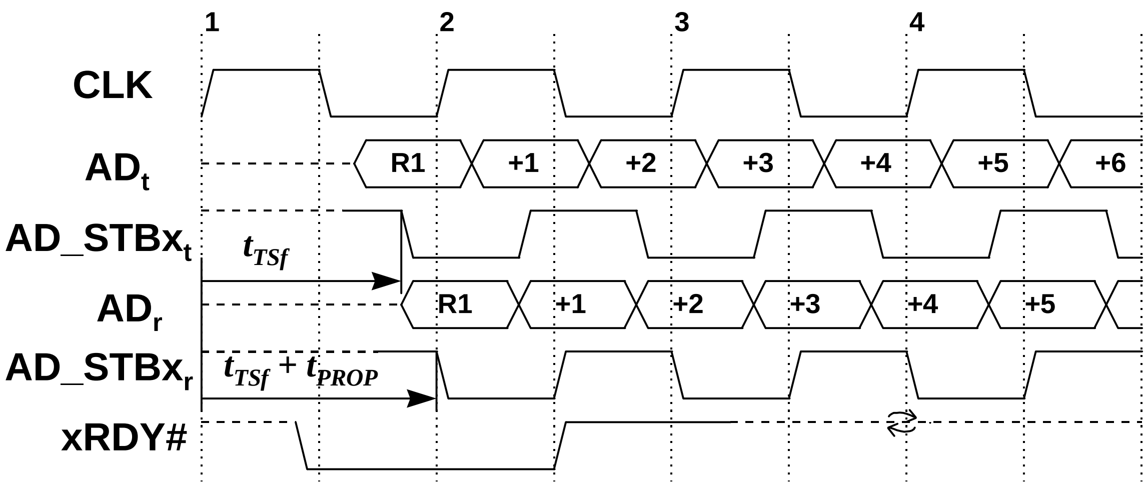


Figure 3-29: Maximum Shift Between xRDY# and AD_STBx

In Figure 3-29, signal names with a subscript t, indicate the signal at the transmitter of the data, while the subscript r indicates the signal at the receiver. **AD_STBx_t** is the strobe driven by the transmitter and is required to be valid within t_{TSf} which is 2-12 ns[23] after the rising edge of **CLK**. **AD_STBx_r** is the strobe as seen by the receiving agent. It is delayed t_{PROP} from the transmitter's delay by up to an additional 2.5 ns[24]. The maximum total of these delays puts the falling edge of **AD_STBx_r** on the rising edge of clock 2. This makes it appear as though data R1 is driven off of clock 2 with no propagation delay, while in fact it was driven from clock 1 with nearly a full clock of delay. Section 4.1.2.9 discusses 2x transactions, when data is valid and when it may be used.

Figure 4-7 clearly shows that the first data transfers during T1 and illustrates a more typical value where t_{TSf} and t_{PROP} are not at the maximum delays allowed. When the agent providing the data uses the minimum times, both the falling and rising edges of **AD_STBx** can occur during T1 or clock 1 in the figure. The agent that receives the data is required to handle the data transferring anywhere between the minimum to the maximum delays of the transmitting agent and the motherboard.

23 See Table 4-7.

24 See Table 4-18.

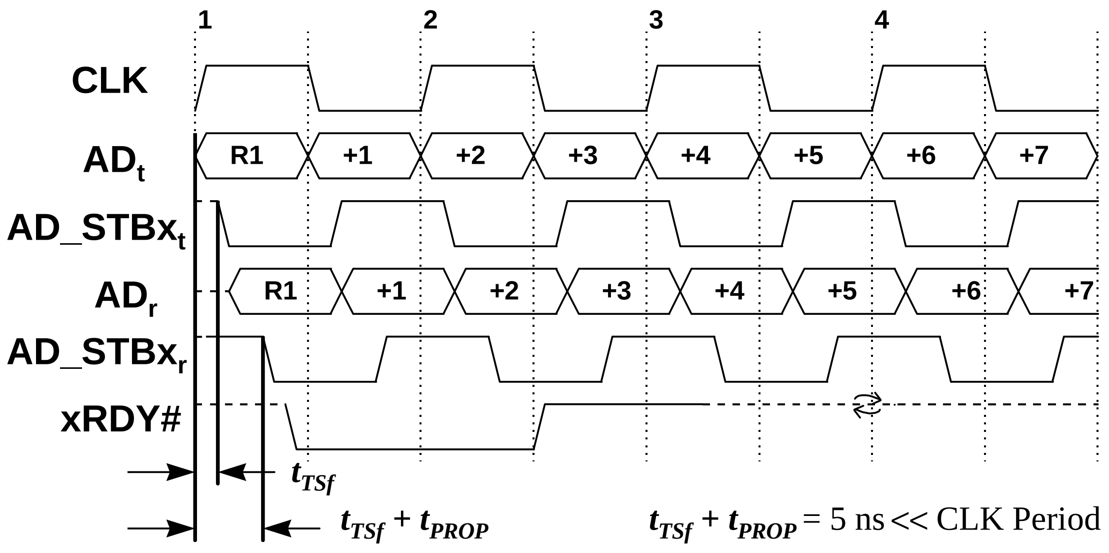


Figure 3-30: Minimum Shift Between xRDY# and AD_STBx

Figure 3-30 is the same as Figure 3-29 except that the minimum times have been used for the agent providing the data. Notice that in this case, both the falling and rising edges of **AD_STB** occur during clock 1 while in Figure 3-29 they occur in clock 2. When the agent providing the data is between the minimum time and the maximum time, then the falling edge occurs in clock 1 while the rising edge occurs in clock 2.

3.5.3.4 4x Data Transfers

This section discusses 4x data transfers. Basically this is the same as 2x clocking except an entire 16 bytes can be transferred during a single **CLK** period. This requires that four 4 byte pieces of data are transferred across the **AD** bus per **CLK** period. First a read data transfer will be discussed, and then a write transfer. The control signals are identical as for the 2x read, except that **AD_STBx#** has been added when data is transferred at 16 bytes per **CLK** period. **AD_STBx#** represents the compliment of **AD_STB0** and **AD_STB1** and are used with **AD_STB0** and **AD_STB1** by the 4x interface logic to know when valid data is present on the **AD** bus. The receiving agent has two choices of how it uses the four strobes to latch data.

The first choice is to use only the falling edge of each strobe to latch data. The first falling edge of **AD_STB0** is used to determine when the receiving agent latches the first four bytes of data residing on the **AD** bus, and the first falling edge of **AD_STB0#** is used to latch the second 4 bytes of the transaction. The second falling edge of **AD_STB0** is used to determine when the receiving agent latches the third four bytes of data residing on the **AD** bus, and the second falling edge of **AD_STB0#** is used to latch the fourth 4 bytes of the transaction. Note that the rising edges of **AD_STBx** or **AD_STBx#** are never used to latch data when in the 4x transfer mode.

The second choice is to use the strobes as differential pairs and not as four separate signals. The compliment pairs are **AD_STB0** and **AD_STB0#**; and **AD_STB1** and **AD_STB1#**. When this choice is used, an internal latch signal is created in which data is latched on both the falling and rising edges. Assume for illustration purposes that the internal signal is **AD_STBx** in Figure 3-31. The first falling edge of **AD_STBx** is used to determine when the receiving agent latches the first 4 bytes of data residing on the **AD** bus and the first rising edge of **AD_STBx** is used to latch the second 4 bytes of the transaction. The second falling edge of **AD_STBx** is used to determine when the receiving agent latches the third 4 bytes of data residing on the **AD** bus and the second rising edge of **AD_STBx** is used to latch the fourth 4 bytes of the transaction. Note that the rising edges of the internal signal are used to latch data when in the 4x transfer mode.

Note that **CLK** is not used to latch data but is used to qualify the control signals. **AD_STBx** is always pulled up and **AD_STBx#** is always pulled down by the central resource.

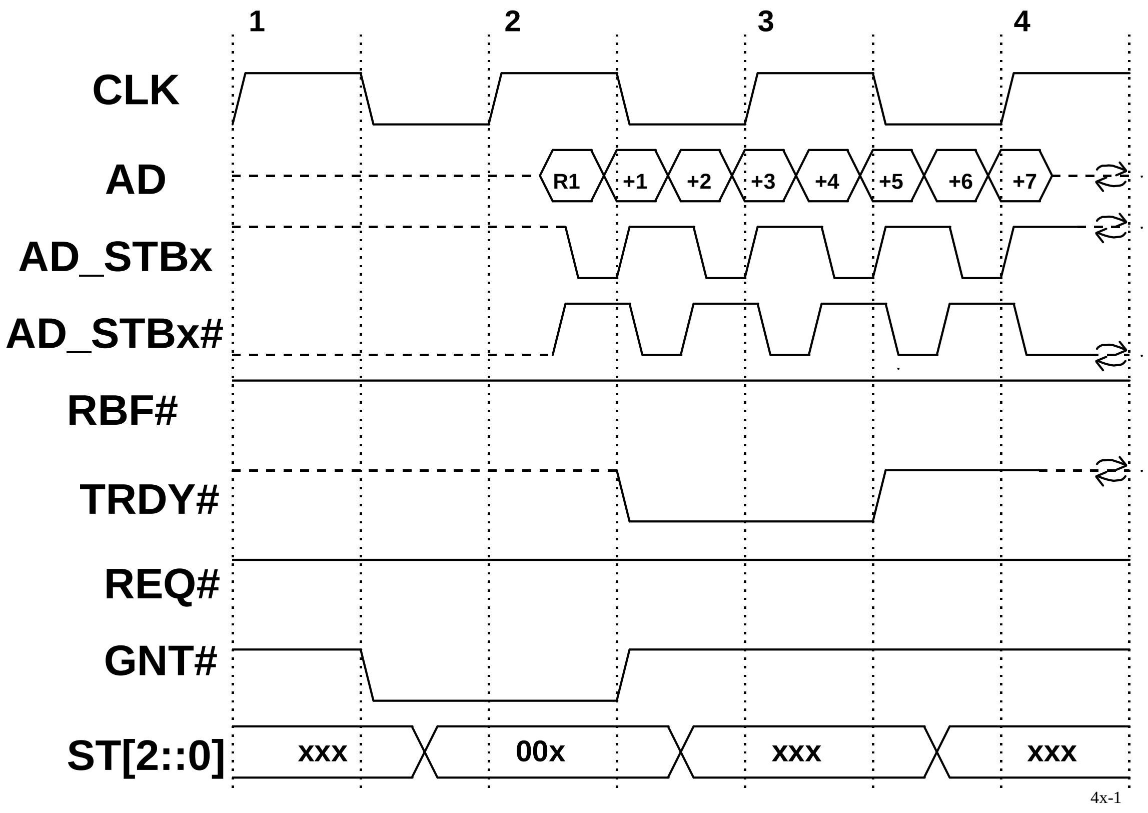


Figure 3-31: 4x Read Data - No Delay

Figure 3-31 is the same as Figure 3-25 except that data transfers at double the speed (4x). The same clock numbers are used to illustrate the differences. The control signals are identical, since flow control is over four clocks regardless of the transfer mode. R1 is transferred across the bus on the first falling edge of **AD_STBx**. The data phase marked as +1 is transferred on the falling edge of **AD_STBx#**. The data phases marked +2 and +3 are the same as the first two data phases.

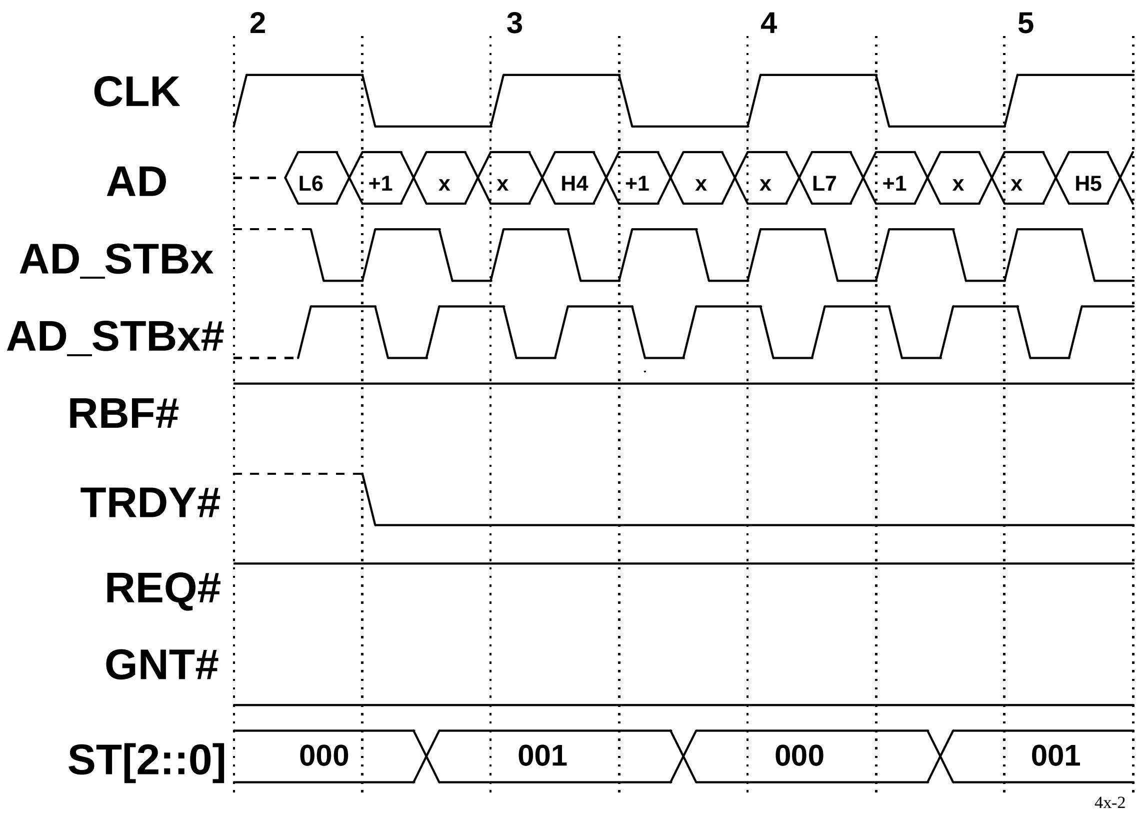


Figure 3-32: 4x Back-to-Back Read Data - No Delay

Figure 3-32 is the same as Figure 3-26 except that data transfers at double the speed (4x). The difference in this figure is that only 8 bytes of data has been requested. Since 16 bytes can be transferred per clock period, the corelogic provides the 8 bytes requested and then transfers 8 bytes of meaningless data. The master is responsible for discarding any data that it does not require. Note that the strobes toggle during the third and fourth data phases even though the data is meaningless which is indicated with an X when data would have been valid. Therefore, when the 4x transfer mode is supported, the master should ask for a minimum of 16 bytes, otherwise, bus bandwidth is wasted. However, if 8-byte transactions are requested, the corelogic will provide the requested 8 bytes and then will provide an additional 8 bytes of meaningless data.

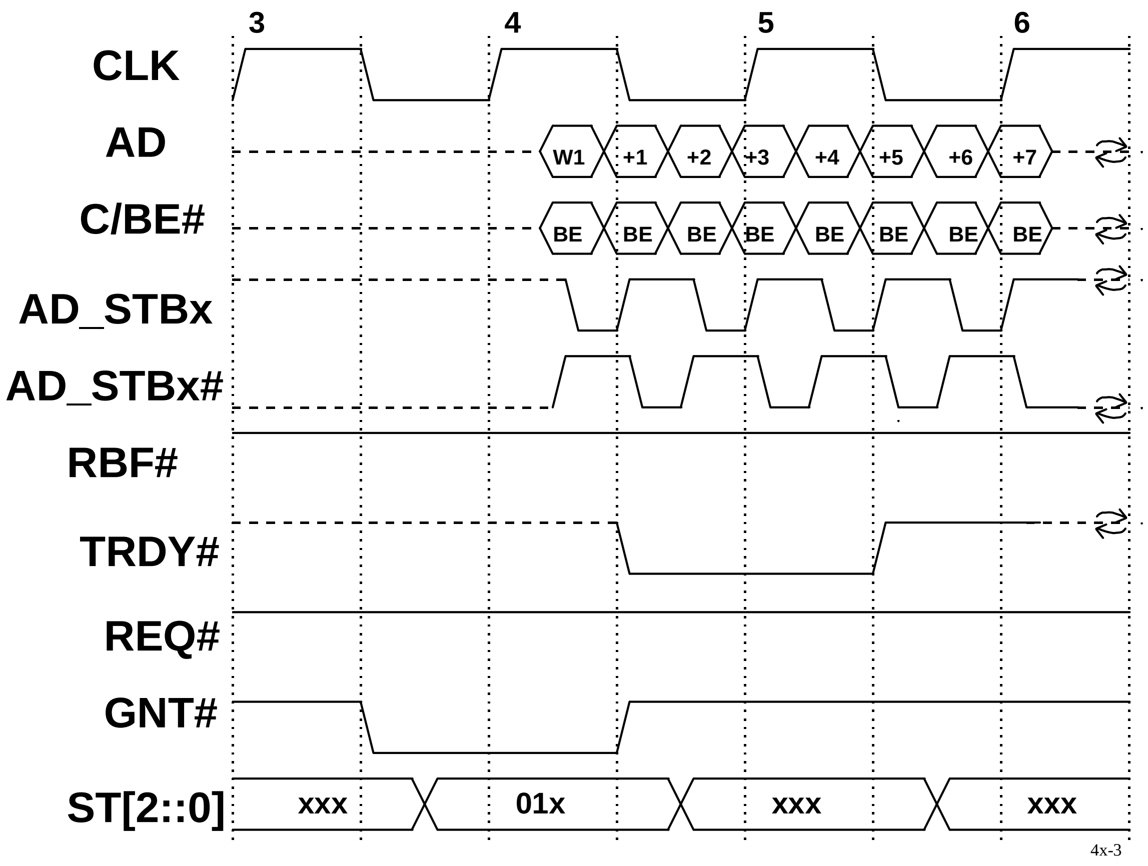


Figure 3-33: 4x Basic Write - No Delay

Figure 3-33 is the same as Figure 3-27 except data transfers at twice the speed. This transaction completes in half the time as the 2x transaction, because the master requested more than 8 bytes for this transaction. The control signals are the same for both figures.

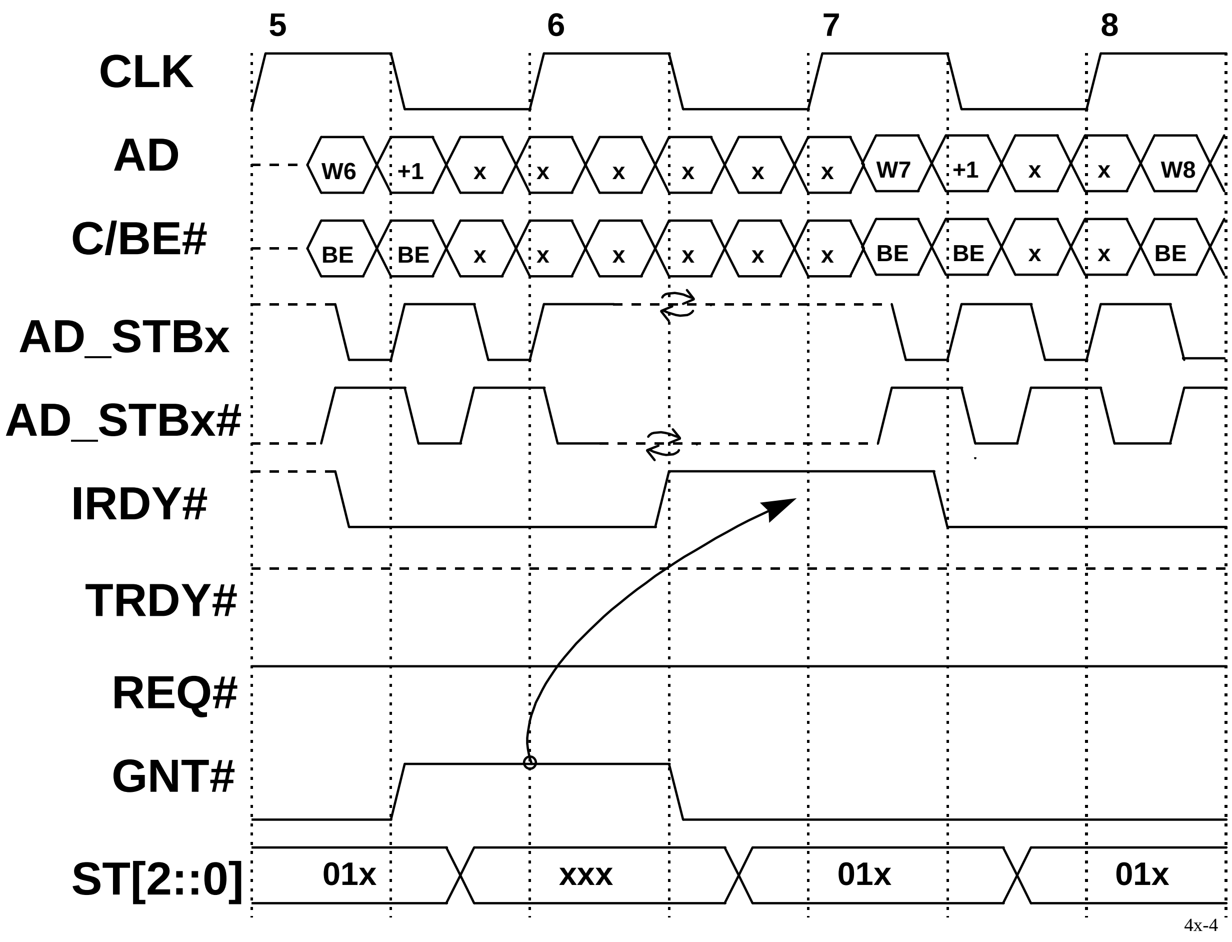


Figure 3-34: 4x QuadWord Writes - No Delay

Figure 3-34 is the same as Figure 3-28 except that data transfers at double the speed. However, both transactions take the same number of clocks to complete because these are 8 byte transactions. The third and fourth data phases of each transaction transfer meaningless data. For this transaction, the byte enables determine which byte lanes provide meaningful data, and since the third and fourth data phases provide meaningless data, the byte enables are deasserted.

Refer to the other 2x diagrams to determine how 4x transactions work under the other conditions. Note that flow control the pipelining of **GNT#**s is the same for 1x, 2x, or 4x data transfers. There is no change in basic control timing in 4x mode. However, it should be highlighted that in 4x mode, the maximum transfer size in one clock is 16 bytes. To support back-to-back 16 byte transfers without an idle bus clock between transactions, the arbiter must pipeline **GNT#** in the same fashion as **GNT#** is pipelined for back-to-back 8 byte transfers in 2x mode.

3.5.3.5 Fast Write Transfers

The Fast Write (FW) transaction is from the corelogic to the A.G.P. master acting as a PCI target. This type of access is required to pass data/control directly to the A.G.P. master instead of placing the data into main memory and then having the A.G.P. master read the data. For 1x transactions, the protocol simply follows the PCI bus specification. However, for higher speed transactions (2x or 4x), FW transactions will follow a combination of PCI and A.G.P. bus protocols for data movement which is defined in this section. PCI protocol will be followed to initiate the transaction, while flow control will follow the A.G.P. block style and not PCI data phase style. Termination of the transaction is like PCI with some modifications to relationships between signals. For example, PCI requires **IRDY#** to be asserted when **FRAME#** is deasserted. However, for FW transactions, this relationship is not required.

One new signal is needed when using FW protocol – Write Buffer Full (**WBF#**). When **WBF#** is asserted, it indicates to the corelogic that the PCI target's write buffers are full and that initiating an FW transaction to the target is not allowed. When **WBF#** is deasserted, the target is indicating to the corelogic that it can accept at least five clocks worth of data before it will terminate the transaction. (Note: five clocks are required when the first of two back-to-back transactions is short – see Figure 3-53 for details.)

The corelogic uses PCI signals to perform FW transactions to the A.G.P. master (acting as a PCI target). For FW transactions, the behavior of the PCI signals has been modified and do not follow the PCI specification in their function or requirements. For example, there is no relationship between **FRAME#** and **IRDY#** for FW transactions.

FRAME# is used to signal the start and duration of a transaction. On the first clock in which **FRAME#** is sampled asserted, the corelogic has placed the address on the **AD** bus and the command on the **C/BE#** bus. Only PCI memory write commands (Memory Write and Memory Write and Invalidate) are allowed for FW transactions. I/O and Configuration Write commands must be completed using PCI protocol. The first clock in which **FRAME#** is deasserted indicates the last clock in which data may be transferred. This means that **FRAME#** is allowed to be deasserted while **IRDY#** is deasserted.

IRDY# is used by the corelogic to indicate to the target that a block of data is beginning to transfer. The corelogic provides up to four clocks of data without inserting waitstates starting with the clock in which **IRDY#** is first asserted.

C/BE[3::0]# indicate which byte lanes carry meaningful data. Like PCI, any combination of byte enables is legal including none. When the corelogic initiates an FW transaction that transfers less data than an entire block (FW-2x 8 bytes, FW-4x 16 bytes), it deasserts the byte enables for the lanes that do not have valid data. The target must qualify the data it latches with the byte enables to determine if valid data was latched.

TRDY# is used by the A.G.P. master (acting as a PCI target) to indicate to the corelogic if it is willing to transfer a subsequent block of data. The target cannot terminate an FW transaction with Retry as it can with PCI. The target uses **WBF#** to prevent the corelogic from initiating an FW transaction when its write buffers are full. The target can request the master to stop the current transaction like PCI, but with slightly different meanings and protocol. A target of an FW transaction can terminate the request after the initial block transfers with Disconnect (with and without) data, Target-Abort or with a modified version of Master-Abort. Each of these terminations will be discussed and illustrated in later sections.

DEVSEL# is used by the target to indicate that it owns the target control signals and must be asserted with or before the target can drive **TRDY#** or **STOP#**. There are some cases, in which the target must suppress the assertion of **DEVSEL#**. This occurs when an FW transaction is short, and is to avoid contention of **DEVSEL#** for back to back transactions. When a transaction requires multiple blocks to complete, the target is required to have **DEVSEL#** asserted by slow decode time, otherwise the corelogic will assume that there is no target. If this condition occurs, the corelogic completes the transaction with Master-Abort semantics. Master-Abort termination on an FW transaction can only occur after the initial block of data transfers. Therefore, the initial four clocks of data is lost if an FW Master-Abort is signaled.

STOP# is used by the target to request the corelogic to stop the FW transaction after the current block complete (disconnect without data) or after the next block (disconnect with data). The target is allowed to terminate the transaction with Target-Abort when the target cannot complete the transaction as requested. The target is allowed to restrict how it is accessed using FW transactions (i.e., only Dword accesses or contiguous byte enables).

The generation and checking of parity is not supported on FW transactions.

The following sections will describe the FW bus protocol in detail and illustrate interesting cases of FW transactions themselves and their interaction with other bus transactions, both PCI and A.G.P. Note that the following figures only illustrate 2x data transfers. However, FW protocol is supported for 4x data transfers.

3.5.3.5.1 FW Basic Transaction

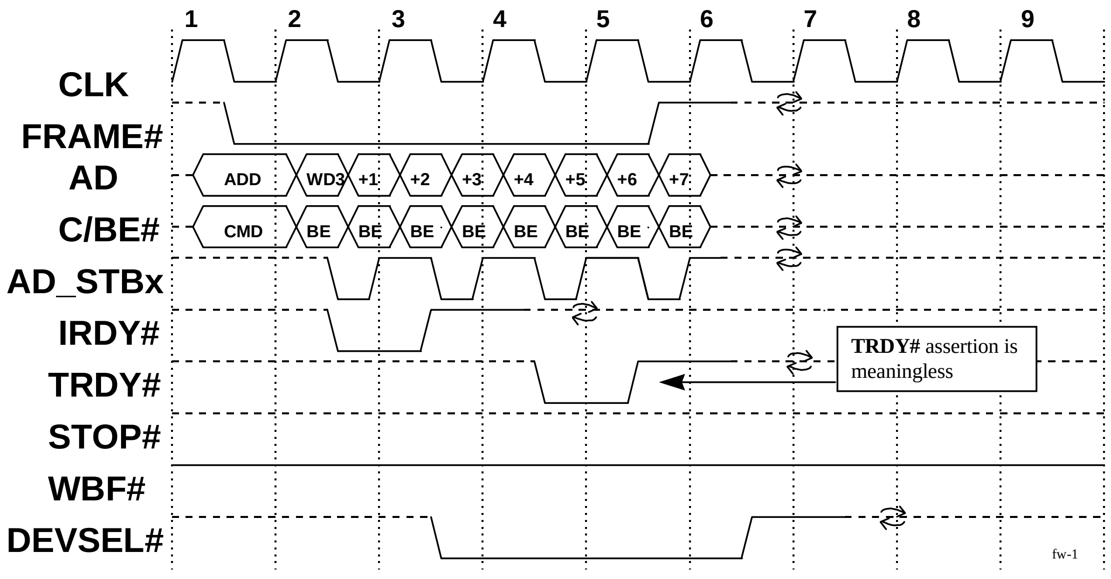


Figure 3-35: Minimum Delay for AD_STBx

Figure 3-35 is an example of an FW transaction. The corelogic, when it has memory write data and has been enabled to do FW transactions, requests use of the **AD** bus by asserting its **REQ#**. This is not shown in the diagram since the corelogic's **REQ#** is an internal signal since the arbiter is part of the corelogic. When the corelogic has been granted access to the bus (internal **GNT#** is asserted and the bus is Idle) and **WBF#** is deasserted, it starts the transaction by placing the memory write command on **C/BE[3::0]#**, the address on **AD[31::00]**, and asserting **FRAME#** which occurs on clock 2. The corelogic on the next clock places the actual data on the **AD** bus and asserts **IRDY#**. The first Dword of data actually transfers on the first falling edge of **AD_STBx** and the second Dword transfers on the rising edge. In this figure, both transfers occur during clock 2. The target (A.G.P. master) is required to accept the first block of data before it can insert waitstates or terminate the transaction because **WBF#** is deasserted on clock 1. The target accepts the first block of data and indicates to the master that it is willing to accept the next block by the asserting **TRDY#** (for a single clock) on clock 5. If the master wishes to continue the transaction, it would keep **FRAME#** asserted on clock 6 which is illustrated in Figure 3-37. Since the master deasserts **FRAME#** on clock 6, the assertion of **TRDY#** on clock 5 was meaningless. In this example, the target does not know that a second block of data is not required to complete the transaction until **FRAME#** is deasserted on clock 6. The target asserts **TRDY#** for clock 5 to allow the master to continue the burst (transfer a subsequent block) without waitstates.

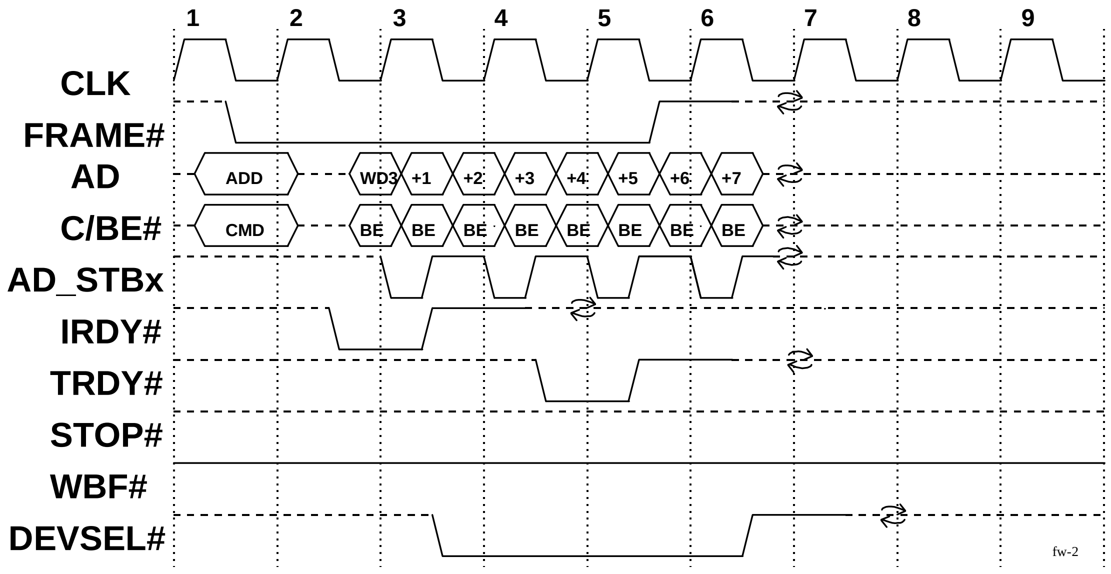


Figure 3-36: Maximum Delay for AD_STBx

Figure 3-36 is the same as Figure 3-35 except that the corelogic takes the maximum delay for the assertion and deassertion of **AD_STBx** while Figure 3-35 shows a minimum time. The rest of the transaction is the same with a single block of data being transferred. This figure illustrates that the actual data transfer can occur entirely in the second clock after the assertion of **FRAME#** or that (as in this figure) that part of the data occurs in first clock after the assertion of **FRAME#** and the rest in the second clock. Since the data only transfers on the edge of **AD_STBx** and not on the rising edge of **CLK** when **IRDY#** is asserted, care needs to be taken when latching data for FW transactions. The falling edge of **AD_STBx** can occur on the rising edge of **CLK**. This condition occurs when the corelogic takes the maximum time of 12 ns to assert **AD_STBx**, the system can use an additional 3 ns to propagate the signal to the target. Therefore, the target can receive **AD_STBx** 15 ns after the rising edge of **CLK**, which is the period of **CLK**. The rest of the timing diagrams will assume a more typical value than the maximum. Therefore, both edges of **AD_STBx** will occur in the same period of **CLK**; but note that this is not required and the target must be able to accept the maximum delay allowed by this interface specification.

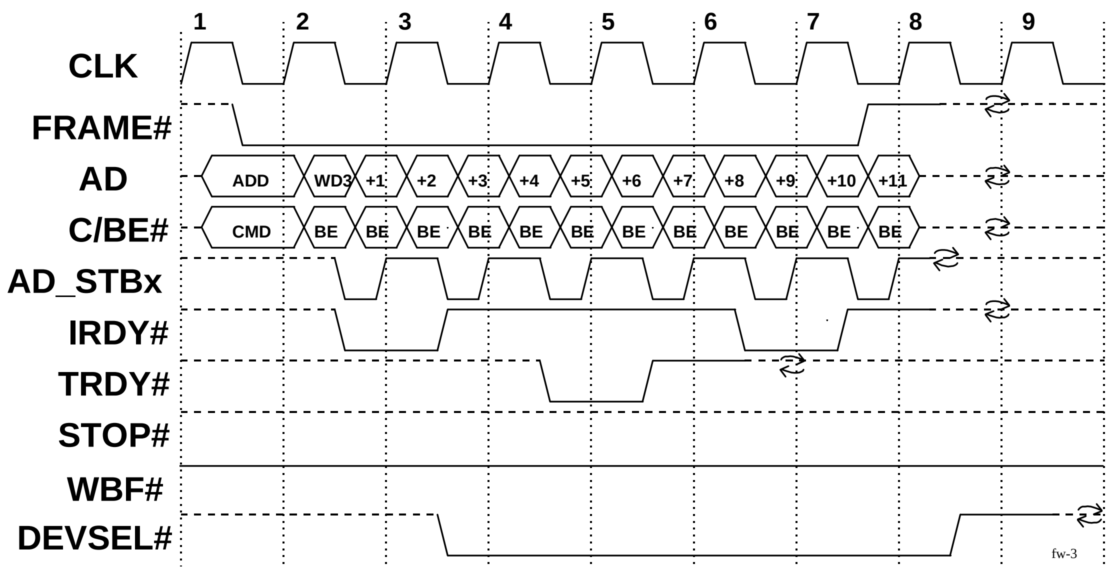


Figure 3-37: Transaction with Subsequent Block

Figure 3-37 is the same as Figure 3-35 except the corelogic continues the transaction past the initial block of data. The assertion of **TRDY#** on clock 5 has meaning and indicates that the target is ready to transfer the second block of data. Since **TRDY#** is asserted on clock 5, the corelogic is allowed to transfer data for the second block starting on clock 7. The target knows that the transaction is ending on clock 8 because **FRAME#** is deasserted. The next TP would have occurred on clock 9 if **FRAME#** had remained asserted. The state of **IRDY#**, after it is asserted indicating the start of a block transfer, is meaningless until two clocks after the completion of the next TP (**TRDY#** is asserted). In this example, **IRDY#** is meaningless on clocks 4, 5, and 6.

3.5.3.5.1.1 FW Transactions with Waitstates

FW transactions are like A.G.P. transactions and not like PCI transactions with respect to waitstates. The corelogic is allowed to insert up to one waitstate between the address phase and the first clock of the data transfer. The target cannot insert any waitstates during the initial block transfer. It uses **WBF#** to prevent the corelogic from initiating an FW transaction. Both agents are allowed to insert waitstates between subsequent data blocks. The number of waitstates between subsequent blocks is not defined.

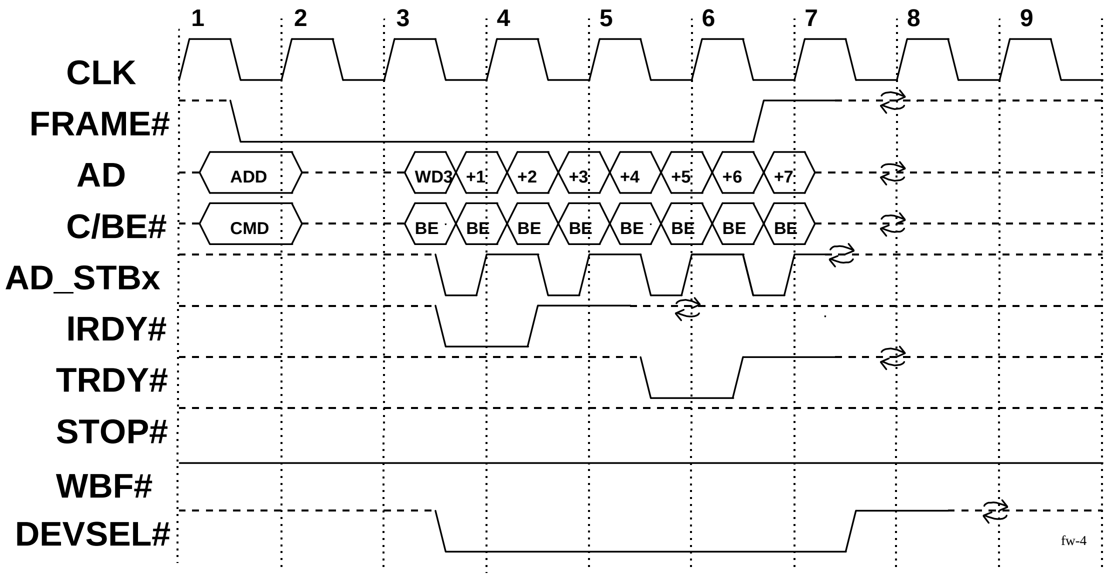


Figure 3-38: Master Delays Data One Clock

Figure 3-38 is an example where the corelogic inserts a waitstate (maximum delay) to assert **IRDY#** indicating that the data is valid on the interface. The master starts the transaction as in Figure 3-35, but in this case delays providing the data by one clock. This is indicated by not asserting **IRDY#** until clock 4 while in Figure 3-35 the corelogic asserts **IRDY#** on clock 3. Beyond this the two figures are the same.

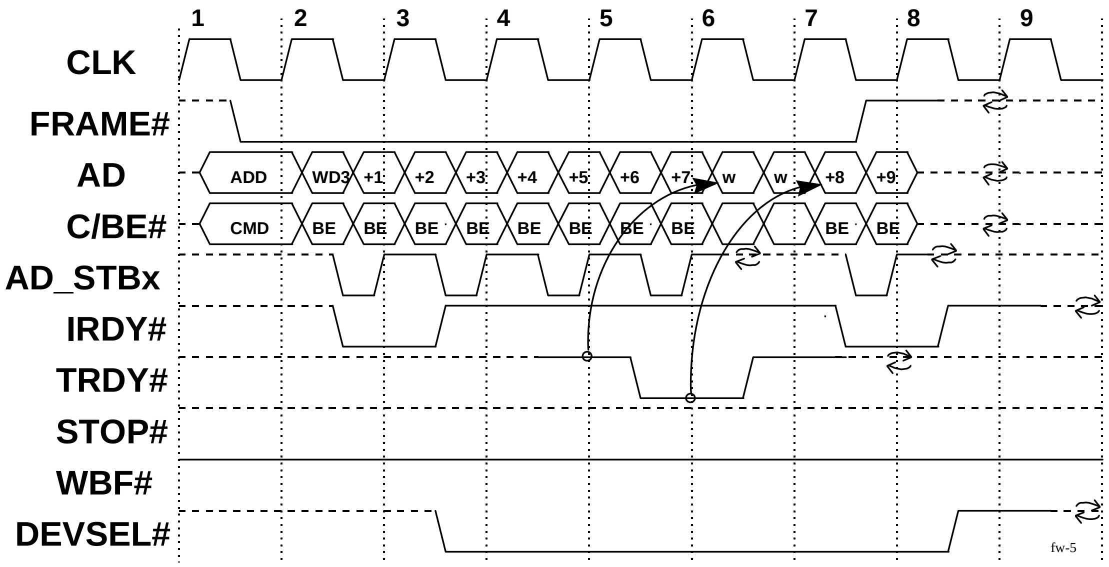


Figure 3-39: Target Delays Subsequent Block

Figure 3-39 is the same as Figure 3-37 except the target inserts one waitstate between the first and second blocks of data. Because **TRDY#** is deasserted on clock 5, a waitstate is inserted on the **AD** bus on clock 7 if **FRAME#** remains asserted on clock 6. Because **TRDY#** and **FRAME#** are asserted on clock 6, the target is ready to accept data on clock 8. The corelogic provides data and asserts **IRDY#** on clock 8 starting the transfer of the second block of data. This is the only case when an FW transaction follows the standard PCI **FRAME#** - **IRDY#** rule. This occurs because the master transfers only one Qword of a subsequent block. In all other cases, **FRAME#** will be deasserted when **IRDY#** is deasserted.

3.5.3.5.2 FW Transaction with Different Terminations

This section will discuss the different terminations that can occur for an FW transaction. Each of the terminations that can be used with PCI will be discussed. These include Retry, Disconnect with Data, Disconnect without Data, Target-Abort, Master-Abort, and Normal terminations.

3.5.3.5.2.1 Retry

The target termination known as Retry on PCI is not supported for FW transactions. The target does not require this termination because it has **WBF#**. **WBF#** prevents the corelogic from initiating an FW transaction to the graphics agent and, therefore, has no need of Retry termination. Refer to the Write Buffer Full discussion in the FW followed by FW section for details about **WBF#** operation.

3.5.3.5.2.2 Disconnect With Data

The PCI target termination know as Disconnect With Data is supported for FW transactions. This is the preferred implementation of the two Disconnect options since it minimizes the wasted clocks on the interface. Disconnect With Data is signaled on the bus when the target claims the access by asserting **DEVSEL#** and then asserts both **STOP#** and **TRDY#** at the TP which occurs on clock 5. **STOP#** is used to request the master to stop the transaction and **TRDY#** is used to indicate that the target is willing to transfer the next block of data.

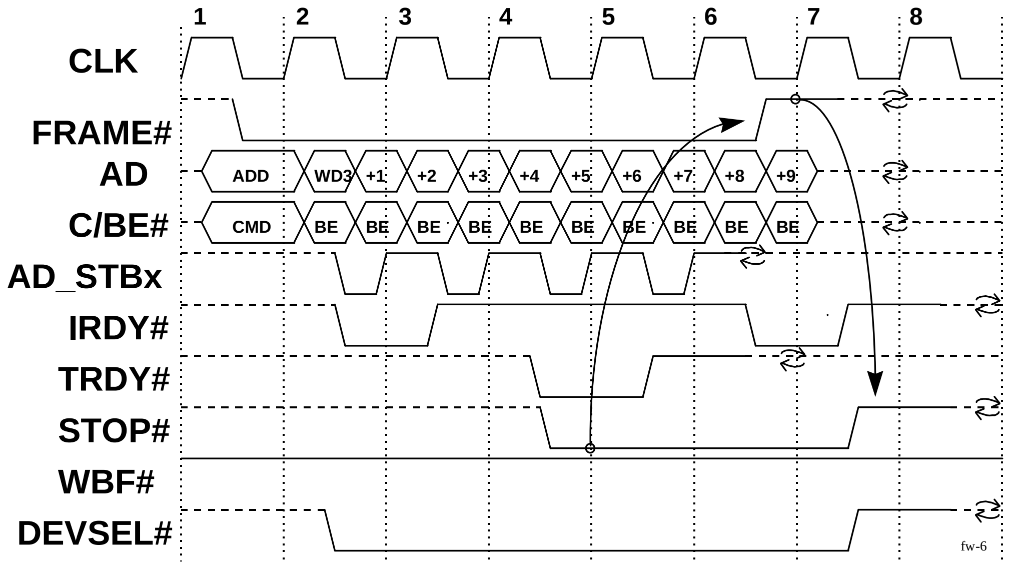


Figure 3-40: Target Stops Transaction After Second Block

Figure 3-40 is a transaction where the target is only willing to accept two blocks of data. In this case, the assertion of **TRDY#** on clock 5 indicates that the target is willing to accept the second block of data. But since **STOP#** is also asserted on clock 5, the target is indicating that it is not willing to accept a third block of data. In this case, the master may have intended to complete the transaction on clock 7 anyway, or is required to stop it prematurely because **STOP#** was asserted on clock 5. Regardless of the master’s intent, the transaction ends on clock 7 which is indicated by **FRAME#** being deasserted on clock 7. The target is required to accept up to four clocks of data per block when it asserts **TRDY#** indicating it is willing to accept the next block. In this case, if the corelogic had desired to continue, it could have transferred data on clocks 9 and 10 before it is required to stop the transaction because **STOP#** was asserted on clock 5. The target is required to keep **STOP#** asserted until it samples **FRAME#** deasserted, at which time it is required to deassert **STOP#** and tri-state it (per standard s/t/s requirements).

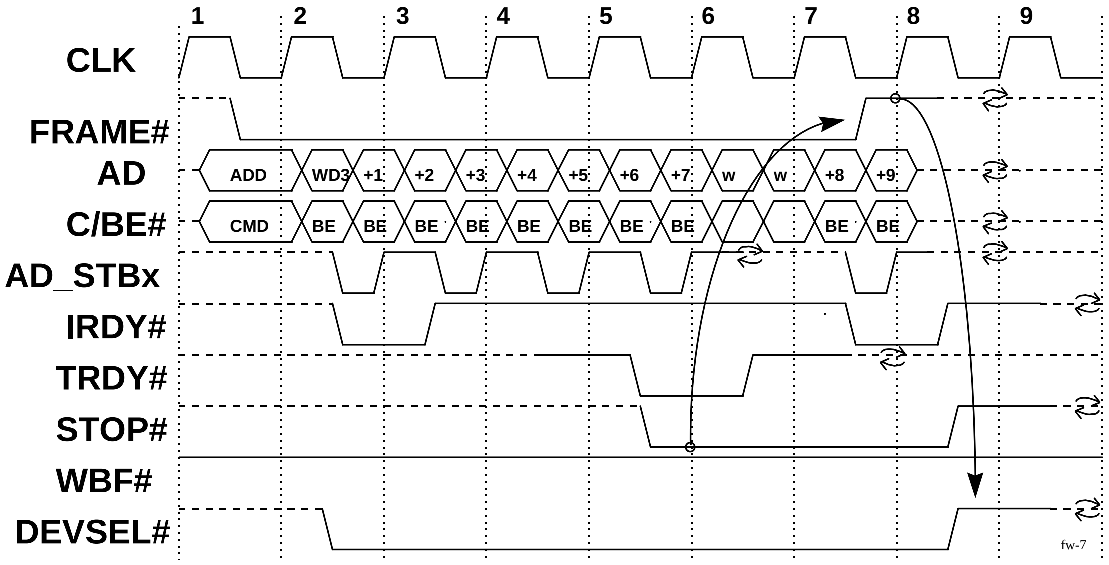


Figure 3-41: Target Delays and Then Stops Transaction After Second Block

Figure 3-41 is the same as Figure 3-40 except in this case the target inserts a waitstate between the blocks of data. In this case, the assertion of **STOP#** is required to be delayed one clock. If **STOP#** is asserted on clock 5 with **TRDY#** deasserted, that would indicate that the target is not willing to transfer the second block of data. As shown in this figure, the target is willing to accept the second block after a waitstate, but is not willing to accept the third block of data. Again, the master may have been intending to stop during the second block anyway because **FRAME#** is deasserted before clock 11. (**IRDY#** is asserted when **FRAME#** is deasserted because the corelogic is transferring one Qword in the subsequent block. If it had been two, three, or four Qwords, **FRAME#** would be deasserted when **IRDY#** was also deasserted.)

3.5.3.5.2.3 Disconnect Without Data

The PCI target termination known as Disconnect Without Data is supported for FW transactions. This is not the preferred implementation of the two Disconnect options, since it requires clocks on the bus in which no data is transferred. Disconnect Without Data is signaled on the bus when the target claims the access by asserting **DEVSEL#** and then asserts **STOP#** but keeps **TRDY#** deasserted at the TP which occurs on clock 5. The TP completes when either **TRDY#** or **STOP#** is asserted. **STOP#** is used to request the master to stop the transaction and **TRDY#** is used to indicate that the target is not willing to transfer the next block of data.

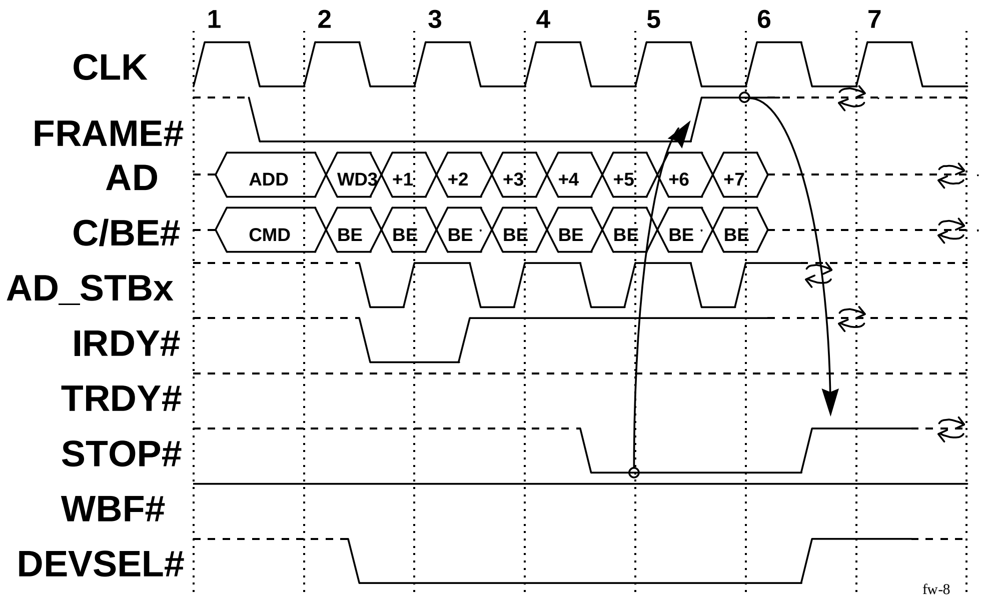


Figure 3-42: Target Stop Transaction After First Block

Figure 3-42 is a case when the target accepts the first four clocks worth of data since **WBF#** is deasserted, but is not willing to accept the second block of data because **STOP#** is asserted on clock 5. In this case, the corelogic is required to deassert **FRAME#** on clock 6 to indicate the last data phase. Caution needs to be taken by the arbiter that it does not assert **GNT#** for a different transaction until all shared signals have been deasserted and tri-stated in preparation for the next transaction. In the case of FW transactions, the bus will appear to be in the Idle condition one clock before it actually reaches that state. Therefore, the arbiter needs to track what type of access is currently on going and then delay the assertion of **GNT#** for a new transaction until it ensures that no contention occurs on the shared signals. Avoiding contention on the **AD** and **C/BE#** buses is relatively easy, but avoiding contention of **TRDY#** and **IRDY#** may be more difficult.

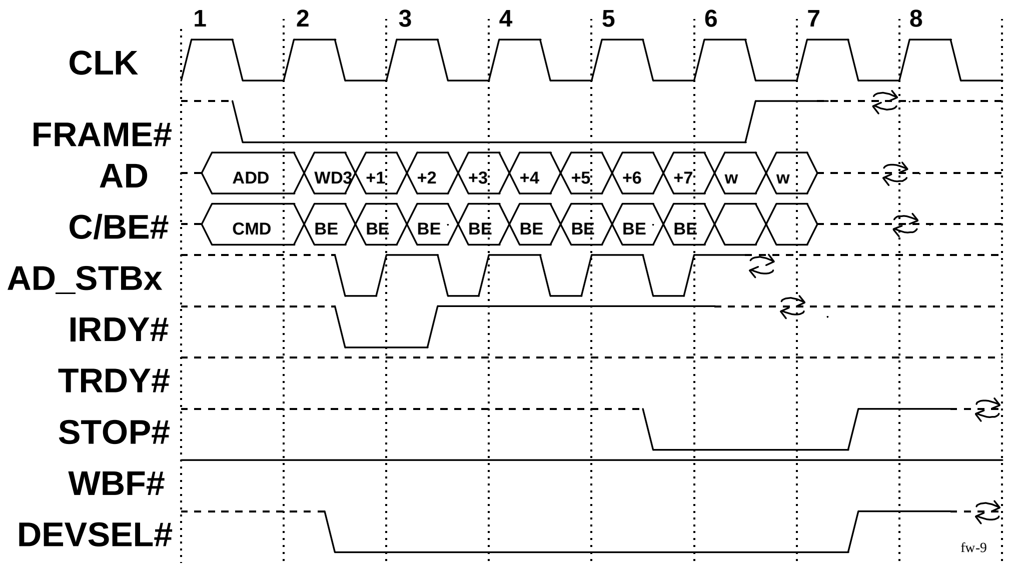


Figure 3-43: Target Delays and Then Stops Transaction After First Block

Figure 3-43 is the same as Figure 3-42, except that the target inserts one waitstate before it indicates that it is incapable of continuing the burst. In this case, a waitstate is inserted on clock 7 because **TRDY#** was deasserted on clock 5 and the corelogic deasserts **FRAME#** on clock 7 because **STOP#** was asserted on clock 6. The TP for this transaction completes on clock 6 because **STOP#** is asserted. Once **STOP#** is asserted, it must remain asserted until **FRAME#** is sampled deasserted which occurs on clock 7.

The master has indicated to the target that some data in the next block will be transferred because **FRAME#** is asserted on clock 6. If the master would insert a waitstate between blocks, it is allowed to delay the deassertion of **FRAME#** even though **STOP#** is asserted on clock 6. The master is required to complete the current transaction as soon as possible.

3.5.3.5.2.4 Target-Abort

The PCI target termination know as Target-Abort is supported for FW transactions. It has the same meaning as for PCI, in that the target can never complete the current request and the master is required to not repeat it again. This is an error condition and is signaled on the interface by deasserting **DEVSEL#** (after it was asserted) with **TRDY#** deasserted and **STOP#** asserted.

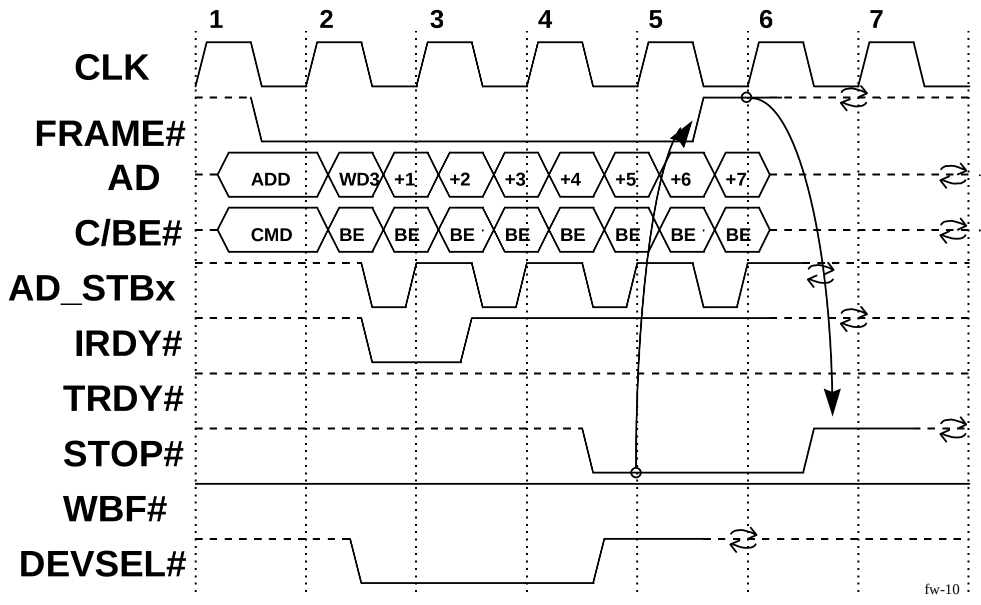


Figure 3-44 Target-Abort

The target of the FW transaction claims the access by asserting **DEVSEL#** on clock 3, in Figure 3-44, when it has completed the address and command decodes. The target is required to accept the first block of data before it can request the transaction to stop. In this case, the target has determined that it cannot complete the transaction and requests the master to stop when the transfer of the first block completes. The target deasserts **DEVSEL#**, keeps **TRDY#** deasserted, and asserts **STOP#** on clock 5 to signal a Target-Abort. Since **STOP#** is asserted on clock 5, the master is required to deassert **FRAME#**. The target is required to keep **STOP#** asserted until it samples **FRAME#** deasserted, which occurs on clock 6 in the example. Once **FRAME#** is deasserted, the target then deasserts and tri-states **STOP#**. The target could have delayed the signaling of Target-Abort by keeping **DEVSEL#** asserted and **STOP#** and **TRDY#** deasserted.

3.5.3.5.2.5 Master-Abort

The PCI termination know as Master-Abort is supported for FW transactions. It has the same meaning as PCI but can occur when data transfers for the transaction. Since the target is required to accept the first four clocks worth of data (**WBF#** deasserted), the true meaning of a PCI Master-Abort cannot be signaled. However the same signaling is used. The difference is that four clocks of data are transferred before the master knows that there is no target accepting the data. FW Master-Abort termination is signaled on the interface the same way it is on PCI in that **DEVSEL#** is not asserted by slow decode time. (Subtractive decoding is not support on the A.G.P. interface.) The master knows that waitstates are not being inserted by the target between the initial and subsequent blocks, when both **TRDY#** and **DEVSEL#** are deasserted by the slow decode sample point.

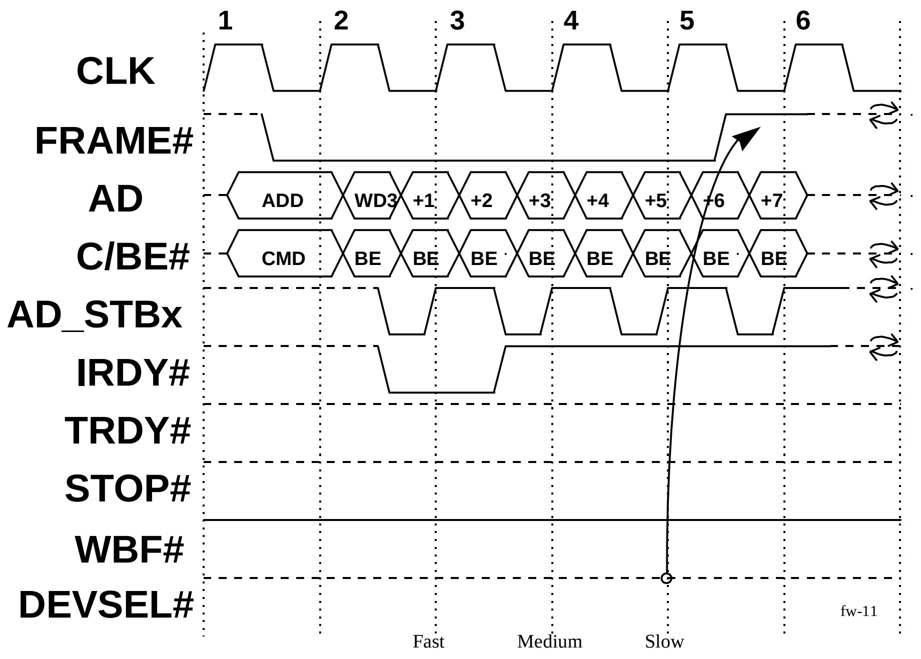


Figure 3-45 Master-Abort

In Figure 3-45 the target fails to assert **DEVSEL#** by clock 5, in this case the master knows that no target is going to responding. The data transferred during the first block is dropped. Subsequent blocks are treated as a separate transaction, since separate memory write operations can be combined into a single bus transaction. The target is required to assert **DEVSEL#** by clock 5 in order to perform target termination or to insert waitstates.

3.5.3.5.2.6 Normal

PCI normal termination is where the master was able to transfer all the data that it desired. This means that the target did not assert **STOP#** to request the master to end the transaction. This is the typical termination of an FW transaction. A normal completion is shown in Figure 3-35 and Figure 3-37.

3.5.3.5.3 Back to Back Transactions

Because the interface supports multiple bus protocols, this section will discuss the relationships when an FW transaction precedes or follows another transaction. Table 3-19 lists all the combinations where an FW precedes or follows another transaction. The first column is the transaction sequence. For example, the first row indicates that an FW is followed by another FW. In this case, column two indicates that a turn-around cycle is not required between the two transactions. The third column refers the reader to a figure that illustrate this example. The fourth and fifth columns indicate which agent is the initiator and target of the first transaction. The last two columns indicate the agents for the second transaction. Following the table each of these cases will be illustrated and discussed.

Table 3-19: Back to Back Transactions

	Turn-around		First Transaction		Second Transaction	
Transaction Type	Required	Figure	Initiator	Target	Initiator	Target
FW to FW	No	Figure 3-46 to Figure 3-53	Corelogic	A.G.P. Master	Corelogic	A.G.P. Master
FW to PCI$_c$ Read	No	Figure 3-54	Corelogic	A.G.P. Master	Corelogic	A.G.P. Master
PCI$_c$ Read to FW	Yes	Figure 3-55	Corelogic	A.G.P. Master	Corelogic	A.G.P. Master
FW to A.G.P. Read	Yes*	Figure 3-56 to Figure 3-58	Corelogic	A.G.P. Master	Corelogic	A.G.P. Master
A.G.P. Read to FW	Yes	Figure 3-59	Corelogic	A.G.P. Master	Corelogic	A.G.P. Master
FW to A.G.P. Write	Yes*	Figure 3-60 to Figure 3-61	Corelogic	A.G.P. Master	A.G.P. Master	Corelogic
A.G.P. Write to FW	Yes	Figure 3-62	A.G.P. Master	Corelogic	Corelogic	A.G.P. Master
FW to PCI$_a$ Read	Yes	Figure 3-63	Corelogic	A.G.P. Master	A.G.P. Master	Corelogic
PCI$_a$ Read to FW	Yes	Figure 3-64	A.G.P. Master	Corelogic	Corelogic	A.G.P. Master
FW to PCI$_a$ Write	Yes	Figure 3-65	Corelogic	A.G.P. Master	A.G.P. Master	Corelogic
PCI$_a$ Write to FW	Yes	Figure 3-66	A.G.P. Master	Corelogic	Corelogic	A.G.P. Master
FW to PIPE#	Yes	Figure 3-67	Corelogic	A.G.P. Master	A.G.P. Master	Corelogic
PIPE# to FW	Yes	Figure 3-68 to Figure 3-69	A.G.P. Master	Corelogic	Corelogic	A.G.P. Master

* Figure 3-57 and Figure 3-61 require two turn-around cycles.

3.5.3.5.3.1 FW to FW

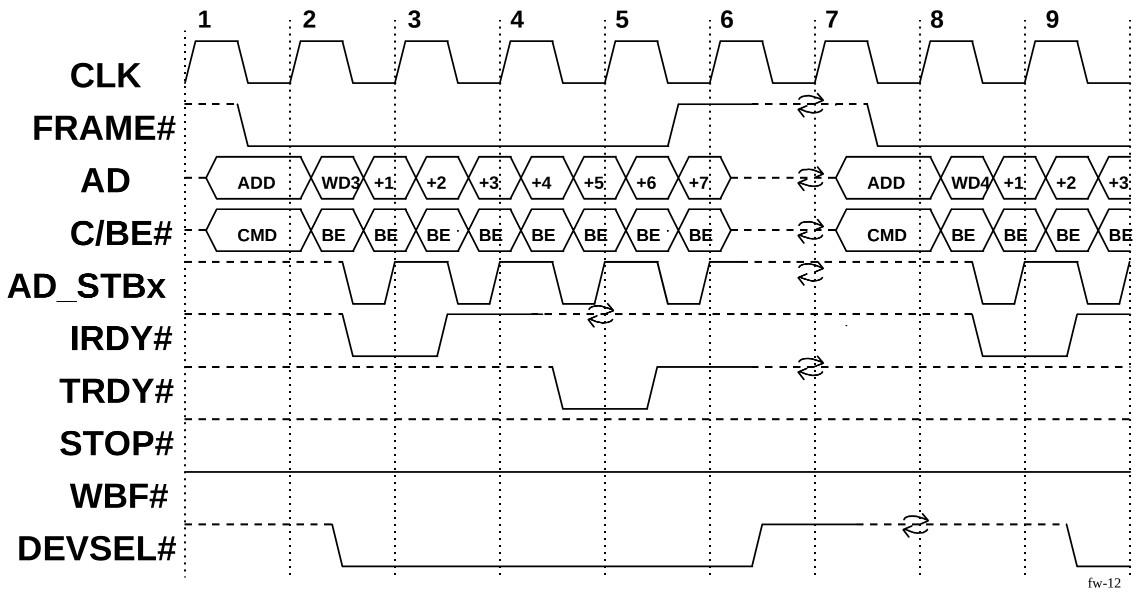


Figure 3-46: Back-to-Back Transaction with Turn-around Cycle

Figure 3-46 is an example of back-to-back transactions where a dead clock is placed between the transactions. Notice that most of the shared signals have a turn-around time on clock 7, such that the second transaction is not required to be from the same master as the previous transaction. However, in this figure, they are both from the corelogic. This condition may be required when the corelogic uses the maximum time to assert **AD_STBx**. The timing could be such that it is impossible to do back-to-back transactions without a dead clock between transactions.

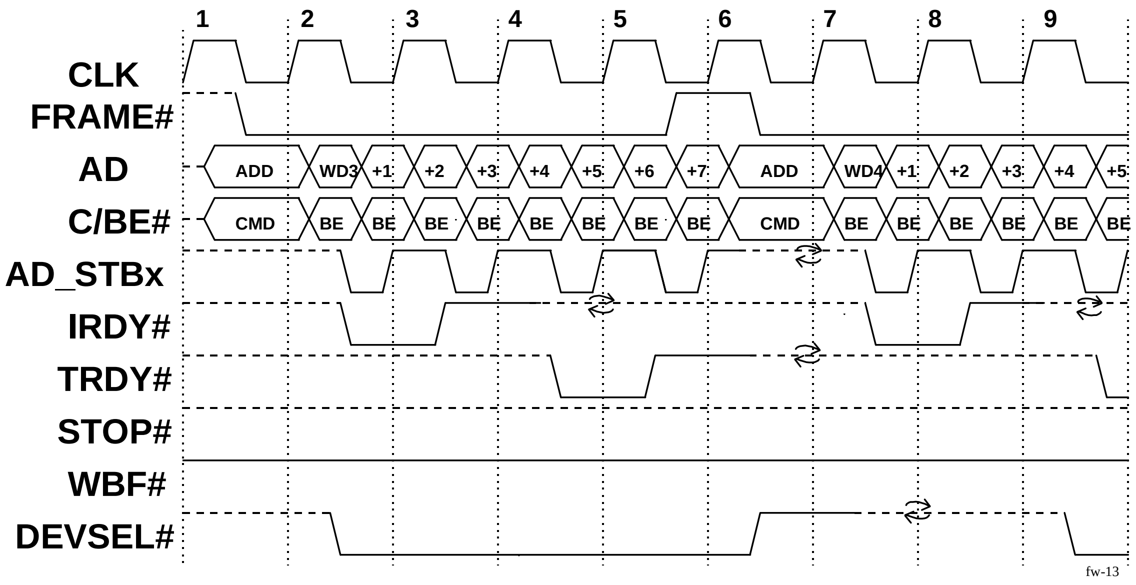


Figure 3-47: Fast Back-to-Back Transaction

Figure 3-47 is the same as Figure 3-46 except that the dead clock has been removed from between the transactions. As mentioned earlier, this type of transaction may not be possible if the maximum delay is used by the corelogic in driving the strobes and data lines. Since it is possible for the corelogic to do this access, the target (the A.G.P. master acting as a PCI target) is required to handle it when issued. Since ownership of the shared signals does not change, a turn-around cycle is not required. The A.G.P. master, when functioning as the PCI target for FW protocol, must be able to handle fast back-to-back transactions like the PCI requirement for targets. In this case, this type of transaction can only be initiated when both accesses are from the same master.

3.5.3.5.3.1.1 Write Buffer Full

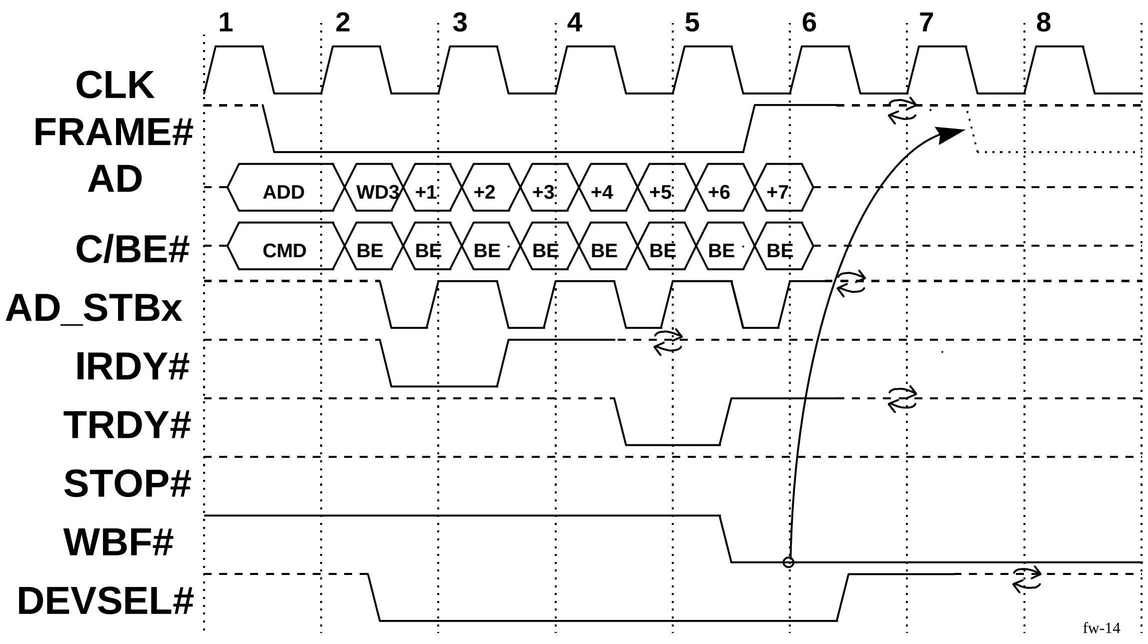


Figure 3-48: WBF# Asserted Prevents Second Transaction

Figure 3-48 shows an FW transaction completing normally. However, notice that **WBF#** is asserted by the A.G.P. master on clock 6 which prevents the corelogic from initiating a new transaction on clock 7 or after. In this case, the corelogic was not doing a fast back-to-back transaction and would have asserted **FRAME#** on clock 8 if **WBF#** had been deasserted on clock 7. In this case, the target indicates, by asserting **WBF#,** that its write buffers are full and it cannot accept a new memory write transaction. If the corelogic has more write data buffered that needs to be delivered to the target, it must not initiate the FW transaction until **WBF#** is deasserted. (Note that when FW protocol is enabled, the corelogic is not allowed to initiate a PCI memory write[25] transaction using standard PCI protocol.)

[25] Can be either Memory Write or Memory Write and Invalidate commands.

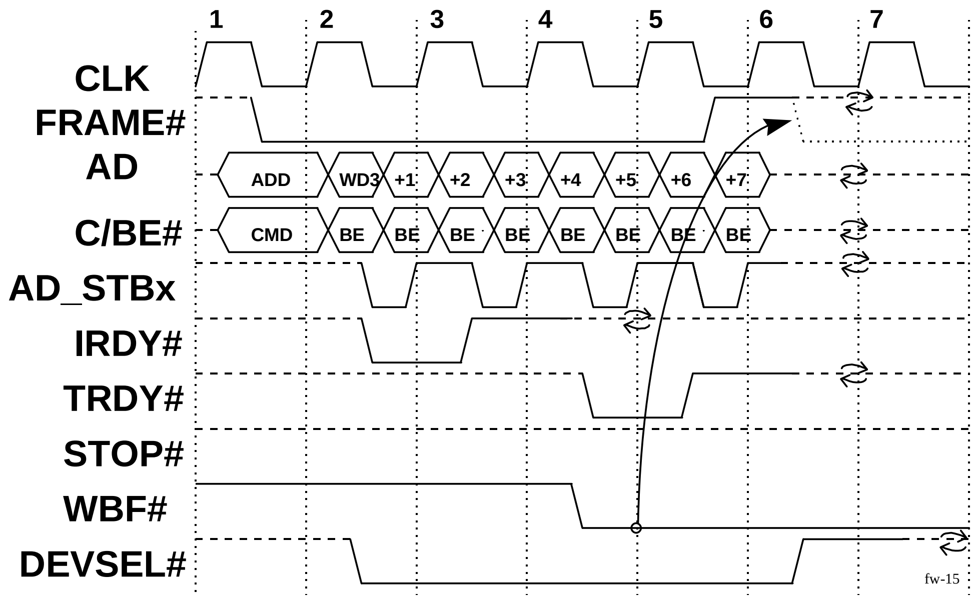


Figure 3-49: Second Transaction Prevented Because WBF# is Asserted

Figure 3-49 is the same as Figure 3-48 except the second transaction could be done as fast back-to-back (no turn-around between accesses). In this case, **WBF#** is asserted one clock earlier to ensure that the second transaction is not allowed to be initiated. If **WBF#** had been delayed one clock, the second transaction would have been allowed to occur as illustrated in Figure 3-53. With the proper use of **WBF#**, the target is only required to have five clocks of buffering which is shown in Figure 3-53. The target is required to accept the first four clocks of any transaction before it is allowed to insert waitstates or do a target termination (Disconnect or Target-Abort). A target termination of Retry is not allowed since this termination means that no data was transferred for the current transaction. This would mean that the master initiated the transaction even though **WBF#** was asserted or the target did not accept the four clocks worth of data. In either case, this is a violation of the FW protocol.

3.5.3.5.3.1.2 DEVSEL# Operation and FW Transactions

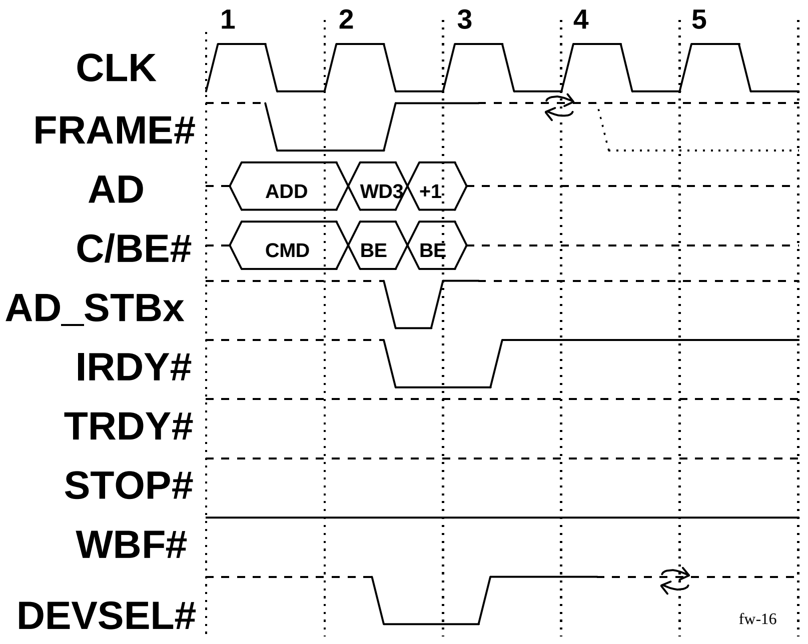


Figure 3-50: Relationship of DEVSEL# and Short Transaction

Figure 3-50 is an interesting case, in which the amount of data transferred is small. This causes some interesting conditions that need to be reviewed. In this case, the entire transaction can be completed in two clocks. The first clock is for the address and command, while the second clock is for the actual data transfer. In this case, since **WBF#** is deasserted, the corelogic knows that the entire transaction can complete without a TP. In this case, the target may not have completed the address decode before the data has completely transferred. The assertion of **DEVSEL#** in this condition is optional. The target is only required to assert **DEVSEL#** before or with the assertion of **TRDY#** or **STOP#**. Since this transaction does not reach a TP, the assertion of **DEVSEL#** is optional. The target **must** accept the first four clocks of any transaction independent of completing the address decode. Once the decode completes and the device is not the target of the transaction, it discards the data that was latched. This is a requirement if there is more than one target interface active when the corelogic is the master. This can occur when the A.G.P. master contains more than a single function; in other words, when the A.G.P. master is a multifunction device that presents multiple PCI configuration spaces to the system. (See Section 6.3 for details.) In this case, the corelogic believes there is a single device and assumes that it is targeting a single device and is allowed to do fast back-to-back accesses. If the first access was to function 0 and the second to function 1, both devices must latch the transaction and store the write data until a full address decode can be completed. When this has occurred, the device not selected by the address simply discards the latched data and does not assert **DEVSEL#** (claiming ownership of the current transaction).

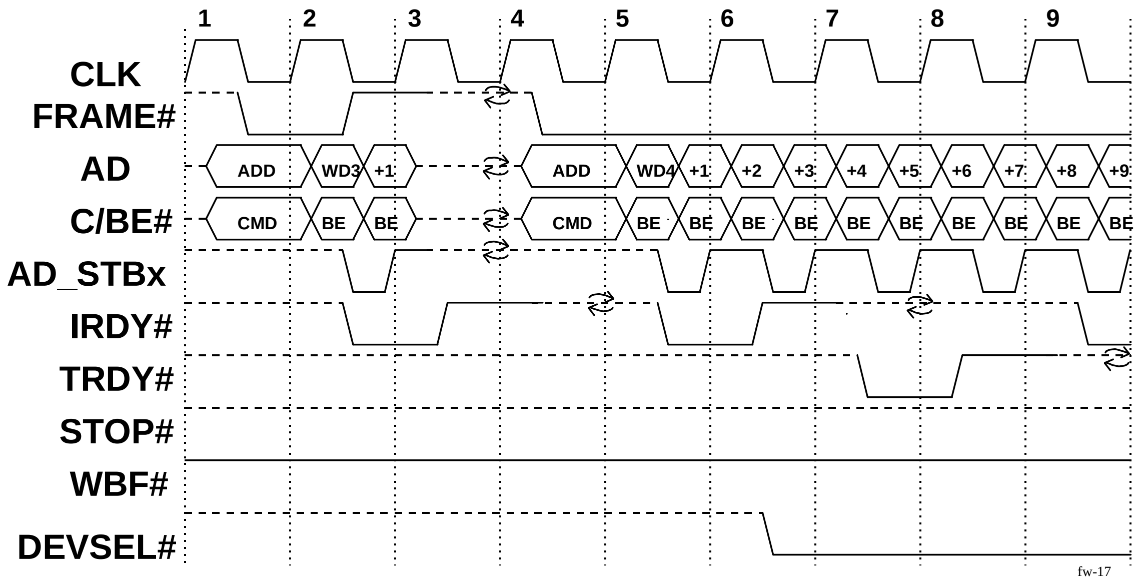


Figure 3-51: Suppression of DEVSEL# on a Short Transaction

Figure 3-51 is a back-to-back transaction where the initial transaction is short. In this case, a turn-around cycle is placed between the transactions. Note that the extra clock is not required in all cases. Notice that **DEVSEL#** was not asserted for the first transaction since it completes before reaching a TP.

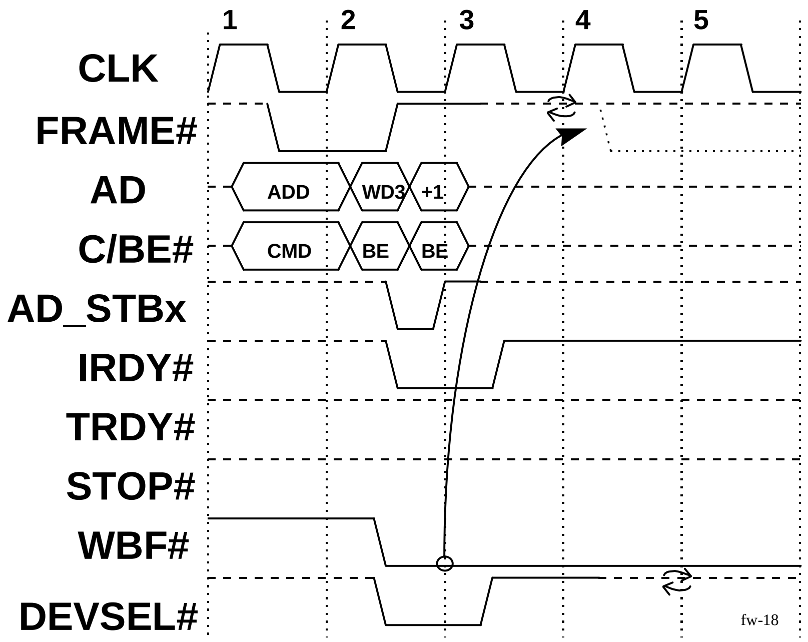


Figure 3-52: WBF# Asserted When First Transaction is Short

Figure 3-52 is the same as Figure 3-48 except that in this case the first transaction is short. **WBF#** must be asserted as soon as the transaction starts in order to prevent a subsequent transaction from being initiated.

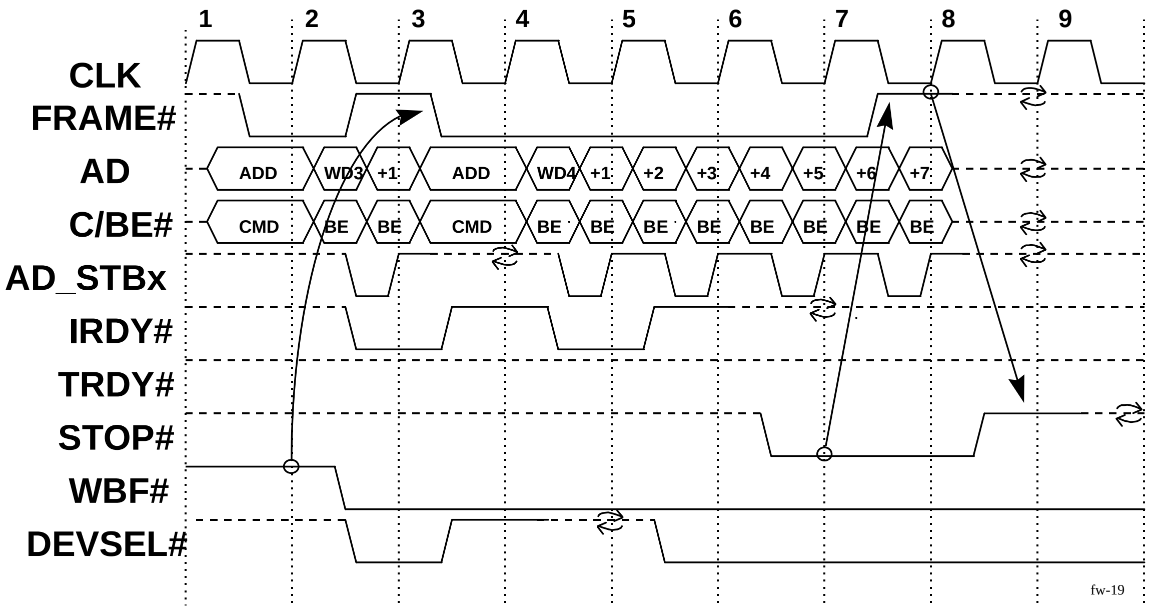


Figure 3-53: Back-to-Back Transaction When First Transaction is Short

Figure 3-53 is the case where the target cannot prevent two transactions from completing. In this case, the first transaction is so short that the corelogic cannot detect **WBF#** asserted until clock 3 which is the same clock in which the second transaction is initiated using a fast back-to-back transaction. For this type of sequence, the A.G.P. master, acting as a PCI target doing FW protocol, is required to provide enough buffering to accept five clocks worth of data. In this case, it requires two Dwords for the first transaction and an additional eight Dwords for the subsequent transaction. Note that the target is only allowed to insert waitstates or terminate the transaction at block boundaries which occur every four clocks. If the master had inserted a waitstate on the initial transaction (delayed the assertion of **IRDY#**) or the transaction was longer than a two clocks, **WBF#** could be detected before the second transaction was initiated. Since this transaction is possible, the target must provide sufficient buffering for it to occur.

3.5.3.5.3.2 FW to PCI$_c$ Read

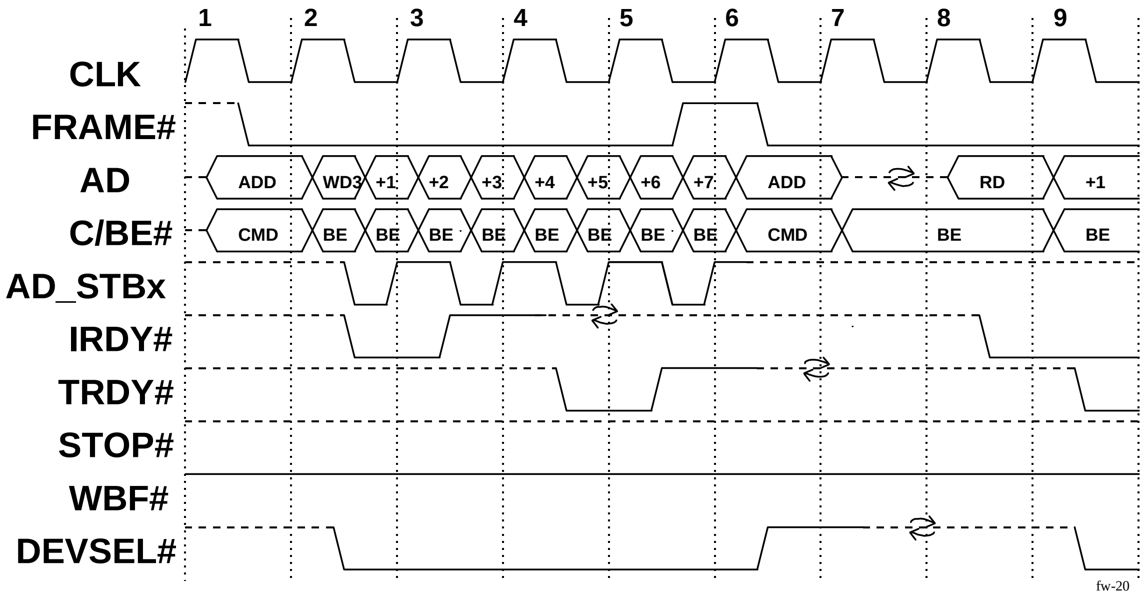


Figure 3-54: FW Followed by PCI Read by Corelogic

When an FW transaction is followed by a corelogic PCI read transaction no turn-around cycle is needed for the **AD** or **C/BE#** buses since the corelogic is the master for both transactions. Figure 3-54 illustrates that there is no contention on any of the shared signals, therefore the need to put a turn-around cycle between the transactions is not required. However, the corelogic is allowed to insert multiple dead clocks if it desires. Notice that PCI transaction occurs on the interface at 1x transfer rates and use a **IRDY#** - **TRDY#** handshake to transfer data.

3.5.3.5.3.3 PCI$_c$ Read to FW

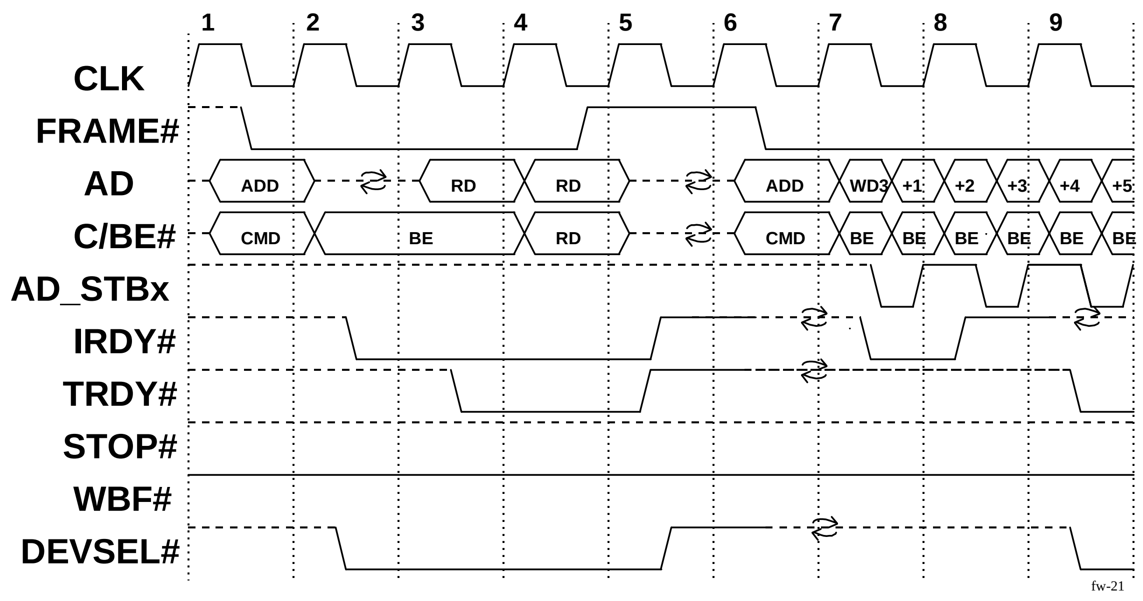


Figure 3-55: PCI Read by Corelogic Followed by FW

Figure 3-55 shows that a turn-around cycle is needed on the **AD** bus since the graphics agent was driving it at the end of the PCI Read transaction and the corelogic will drive it for the next transaction. All other signals have sufficient time for a turn-around to prevent contention.

3.5.3.5.3.4 FW to A.G.P. Read

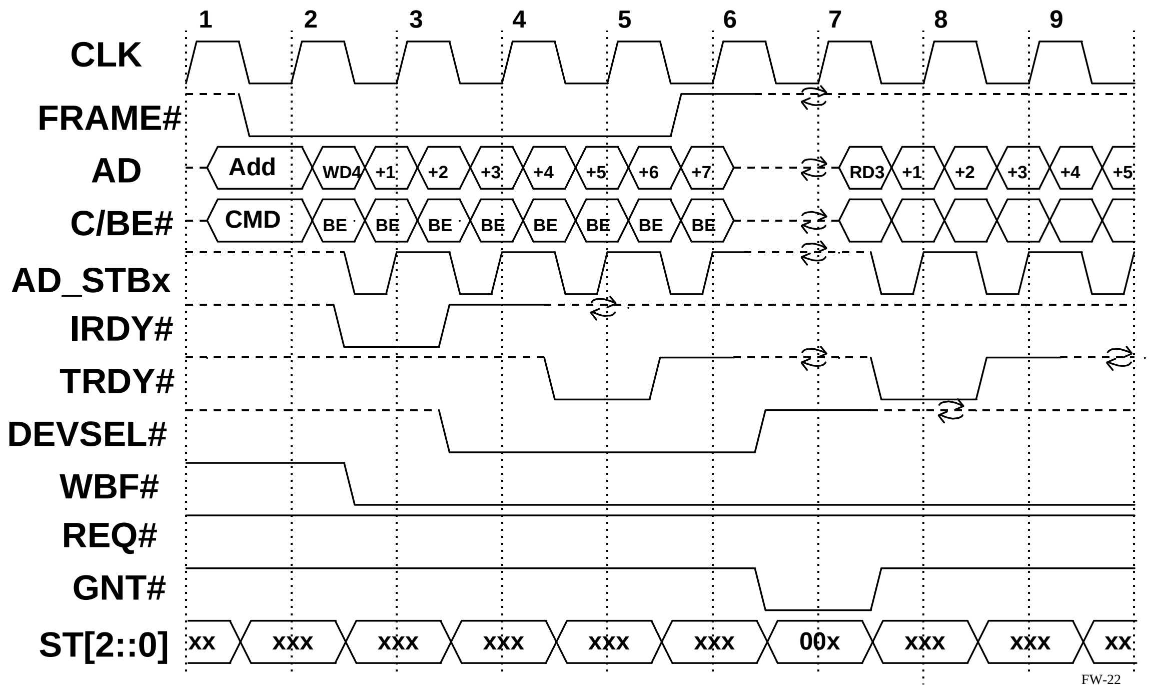


Figure 3-56: FW Followed by an A.G.P. Read Data

Figure 3-56 is the same as Figure 3-59 except the transaction order is reversed. A turn-around is required on clock 7 for **TRDY#** since it changes ownership. While in Figure 3-59 no turn-around is required even though ownership of **TRDY#** changes.

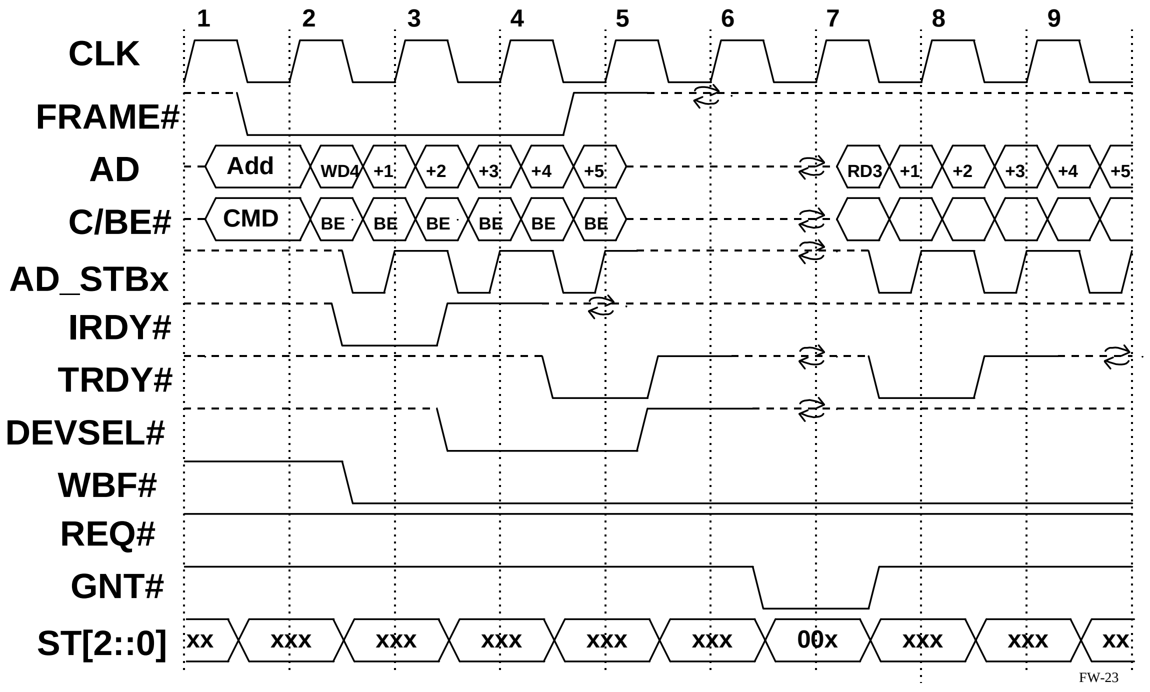


Figure 3-57: Two Dead Clocks Between FW and an A.G.P. Read Data

When an A.G.P. read transaction follows an FW transaction that has three clocks of data, two turn-around cycles are required. In Figure 3-57, the graphics agent does not know the length of the transfer and asserts **TRDY#** on clock 5 to indicate that it is willing to continue the burst without waitstates. However, the corelogic transfers the entire transaction during the initial block and does not require a subsequent block. Since **TRDY#** was asserted, the graphics agent must deassert it and then tri-state it. The corelogic cannot control **TRDY#** until clock 7. Therefore, the **AD** bus has two dead clocks before the corelogic can initiate the second transaction. This same condition can occur whenever **FRAME#** is deasserted and **TRDY#** is asserted. For example, in Figure 3-56 if the graphics agent had inserted a waitstate on clock 5, **TRDY#** would have been asserted on clock 6 when **FRAME#** was deasserted.

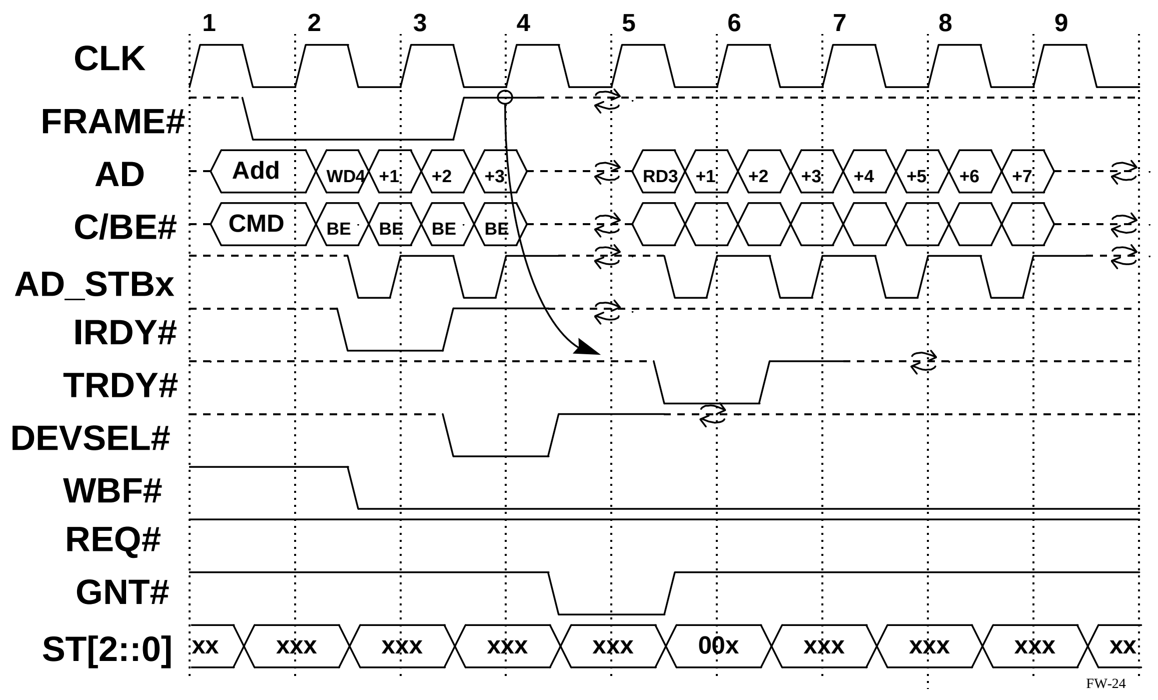


Figure 3-58: Short FW Followed by an A.G.P. Read Data

When the FW transaction completes with fewer than three clocks of data only a single turn-around is required. In Figure 3-58, the graphics agent does not assert **TRDY#** on clock 5 because **FRAME#** is deasserted on clock 4. The corelogic indicates that the current clock is the final data phase when **FRAME#** is deasserted.

3.5.3.5.3.5 A.G.P. Read to FW

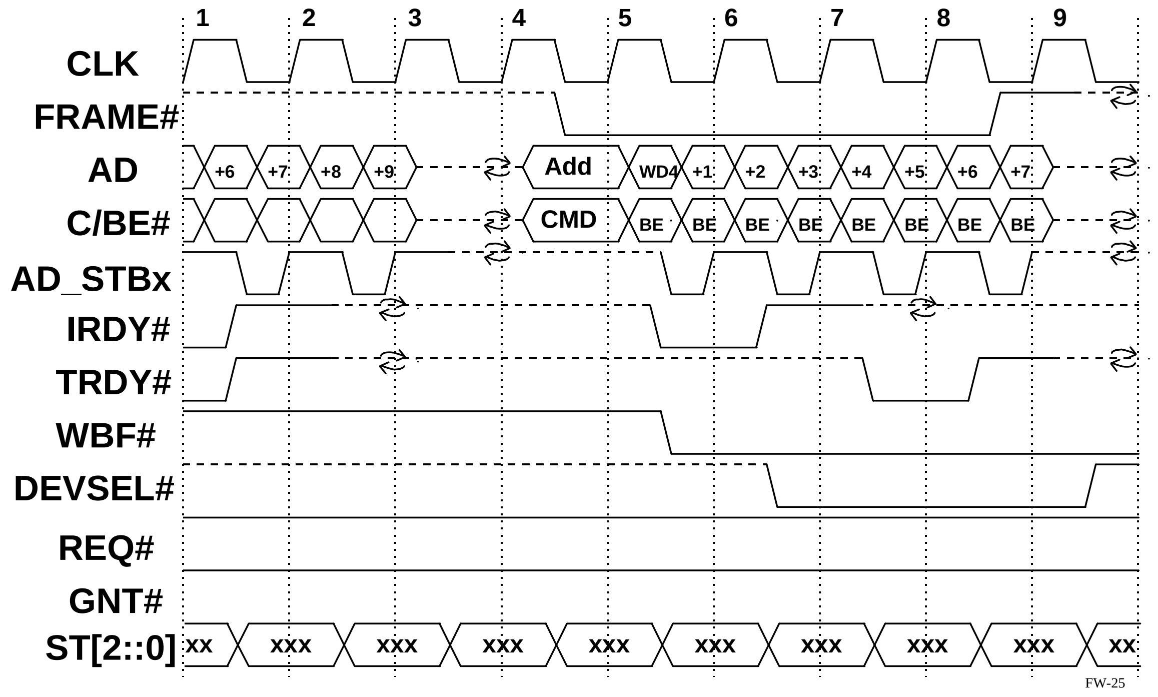


Figure 3-59: A.G.P. Read Followed by an FW Transaction

Figure 3-59 is an A.G.P. read data transaction followed by an FW. In this case, the corelogic is driving the **AD** bus for both transactions. However, **IRDY#** is required to have a turn-around cycle since ownership changes. In this case, the second transaction is the FW and has an address phase which gives **IRDY#** time to switch ownership. A turn-around cycle is required when bus protocols change, therefore, a turn-around cycle occurs on clock 4 because of protocol requirements and not to avoid contention. The same is true for **AD_STBx#** but is not required.

3.5.3.5.3.6 FW to A.G.P. Write

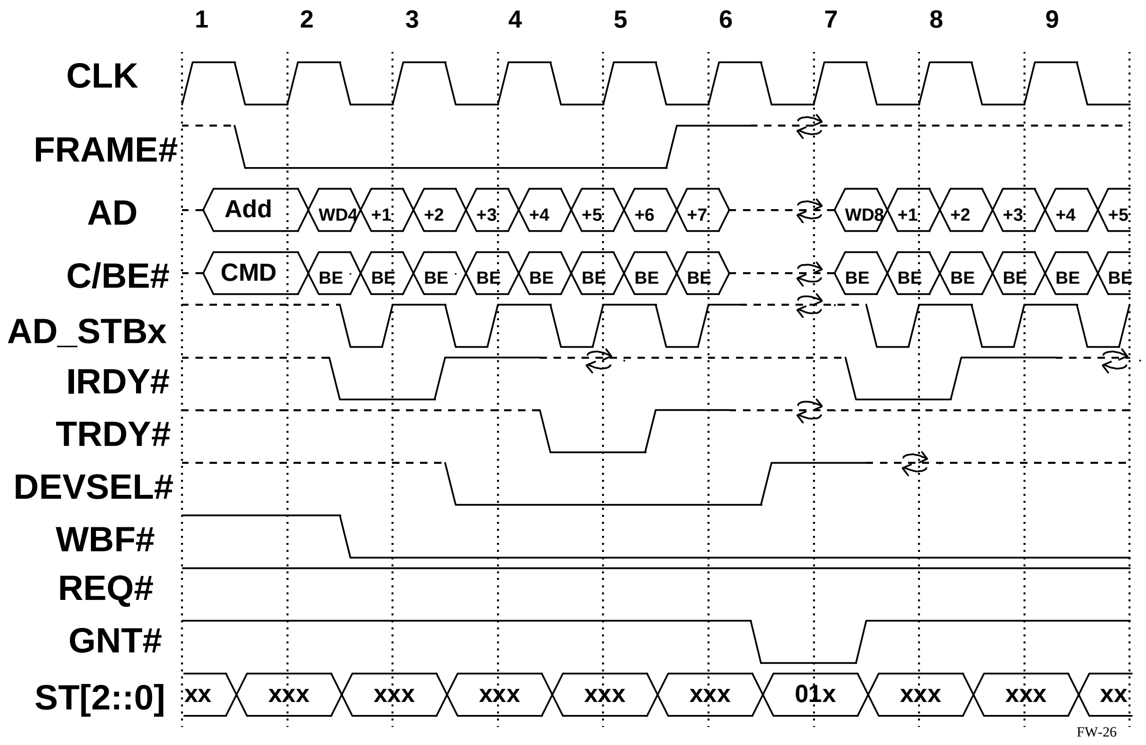


Figure 3-60: FW Followed by an A.G.P. Write Data

Figure 3-60 shows the same two transactions as Figure 3-62, except they are in reverse order. In this case, multiple signals are required to have turn-around cycles to remove contention. This is different than the previous figure because there is no address phase on the second transaction. In the previous case, the other signals had a chance to turn around before they were driven by the second agent. Note that the arbiter is allowed to assert **GNT#** for a write transaction, when **FRAME#** (or **PIPE#**) is asserted on the previous transaction as described in Table 3-19. Therefore, the arbiter could assert **GNT#** in this figure on clocks 3, 4, 5, 6, or 7.

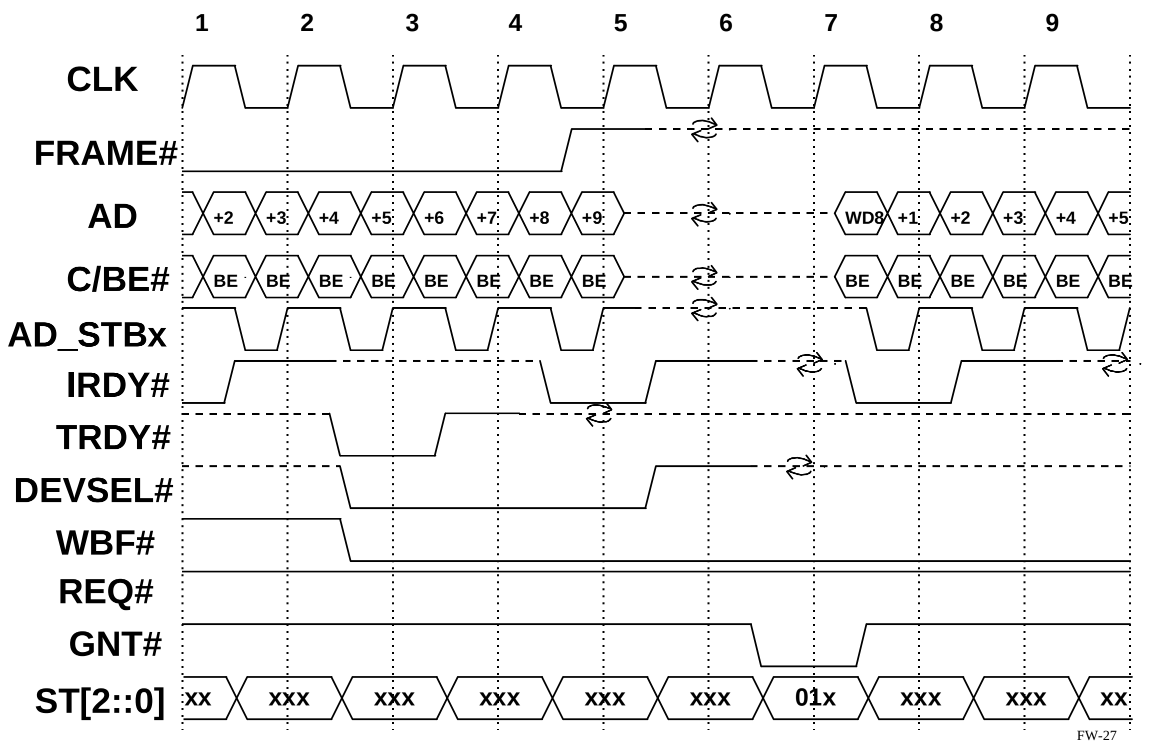


Figure 3-61: Long FW Followed by an A.G.P. Write Data

Figure 3-61 is the same as Figure 3-63 except that in this figure the FW transfers one clock of data in the second block. When this occurs, the corelogic asserts **IRDY#** on clock 5 indicating that the second block of data is starting to transfer and since **FRAME#** is deasserted it is the last clock in which data will transfer. Because ownership of **IRDY#** occurs between these transactions, the arbiter is required to ensure that two clocks of turn-around occur before the next transaction can start.

3.5.3.5.3.7 A.G.P. Write Followed by FW

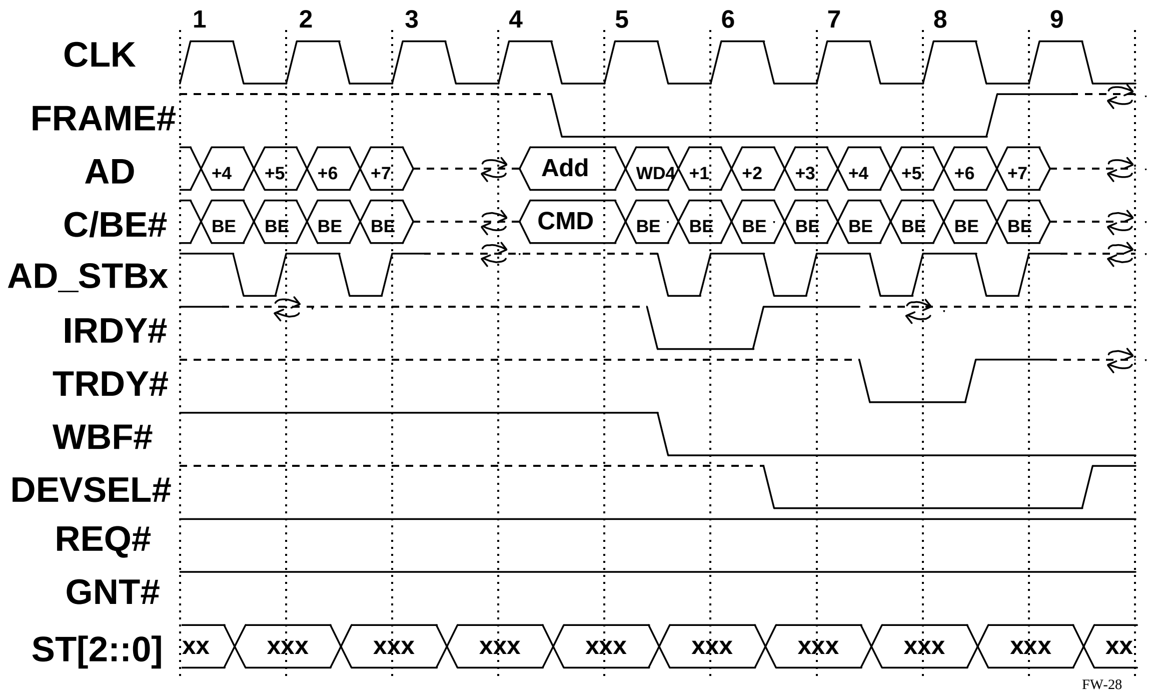


Figure 3-62: A.G.P. Write Followed by FW

Figure 3-62 is an A.G.P. write transaction followed by an FW transaction. In this case, the turnaround is required because the **AD** bus is owned by different agents. Notice that no other signal has any requirement for a turn-around. In this case, the write data is being provided by the A.G.P. master while the FW data is provided by the corelogic. If the A.G.P. write transaction had been short, **IRDY#** may have also required a turn-around cycle.

3.5.3.5.3.8 FW Followed by Graphics PCI Master Read

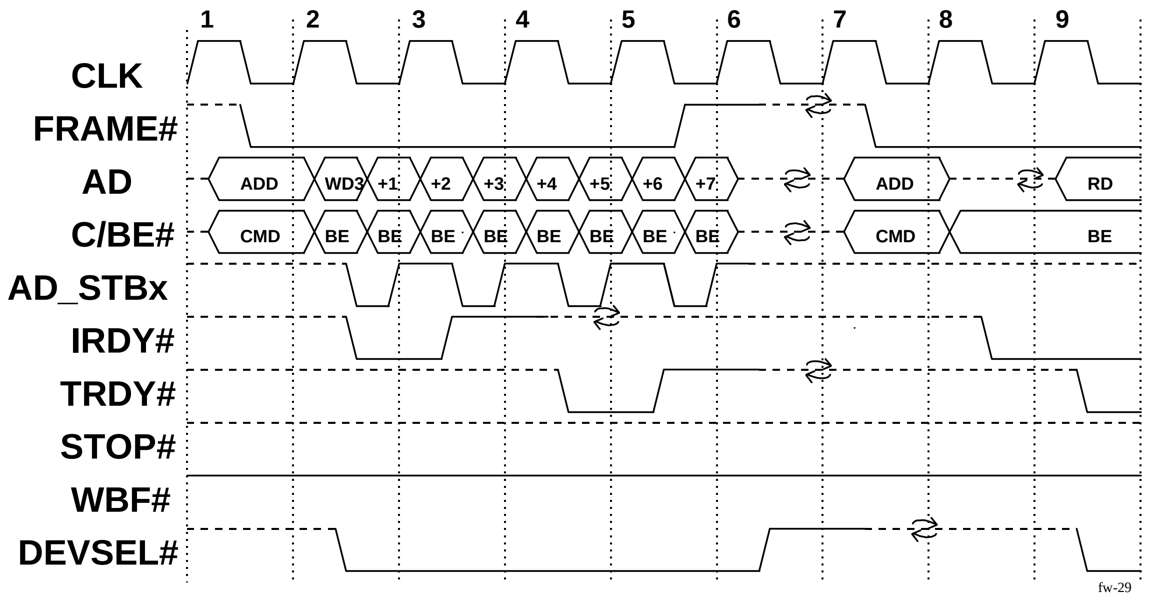


Figure 3-63: FW Followed by Graphics PCI Master Read

Figure 3-63 is an FW transaction followed by a graphics PCI master read transaction and indicates that a turn-around is needed since a change of ownership of the **AD** and **C/BE#** buses occurs.

3.5.3.5.3.9 Graphics PCI Master Read Followed by FW

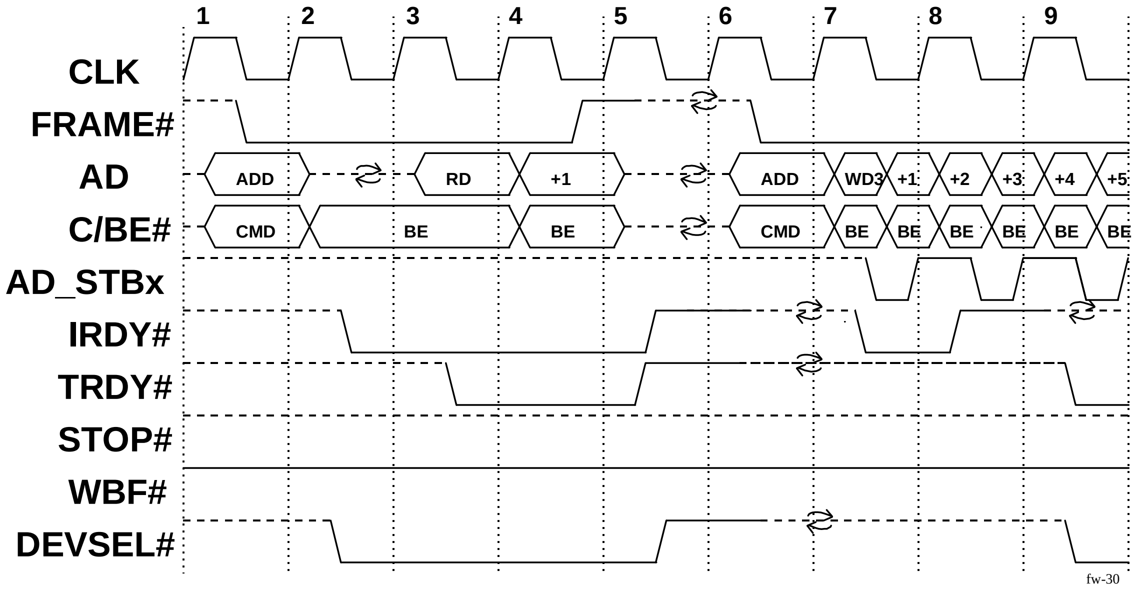


Figure 3-64: Graphics PCI Master Read Followed by FW

In Figure 3-64, the **AD** bus is owned by the same agent and therefore does not need a turn-around. However, the **C/BE#** bus changes ownership from the graphics agent as master to the corelogic as master for the FW transaction. With this turn around cycle, **IRDY#** and **TRDY#** have sufficient time not to have contention.

3.5.3.5.3.10 FW Followed by Graphics PCI Master Write

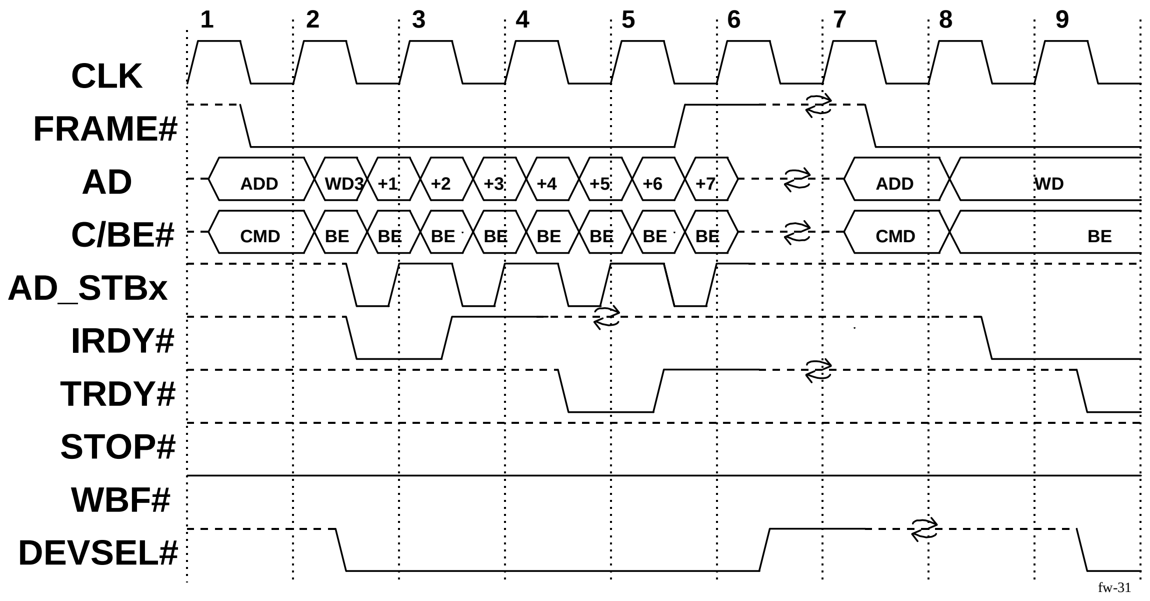


Figure 3-65: FW Followed by Graphics PCI Master Write

When different agents are bus masters for a back to back transaction, a turn-around cycle is needed and occurs on clock 7 in Figure 3-65.

3.5.3.5.3.11 Graphics PCI Master Write Followed by FW

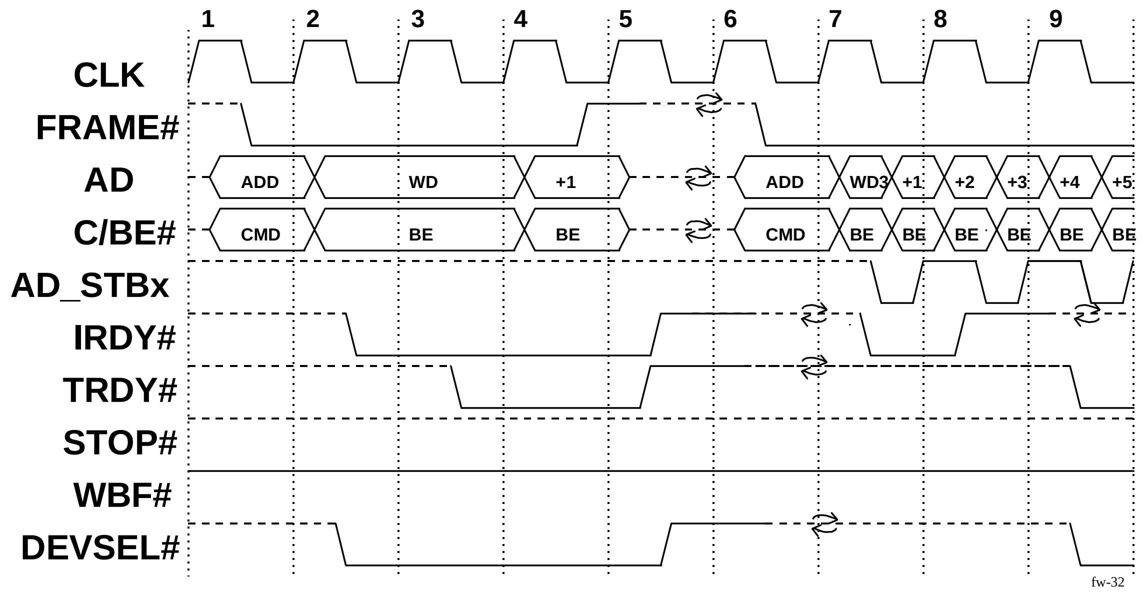


Figure 3-66: Graphics PCI Master Write Followed by FW

Ownership of the **AD** and **C/BE#** buses and **FRAME#** changes, and, therefore, they need a turn-around cycle between bus transactions which occurs on clock 6 in Figure 3-66.

3.5.3.5.3.12 FW Followed by PIPE#

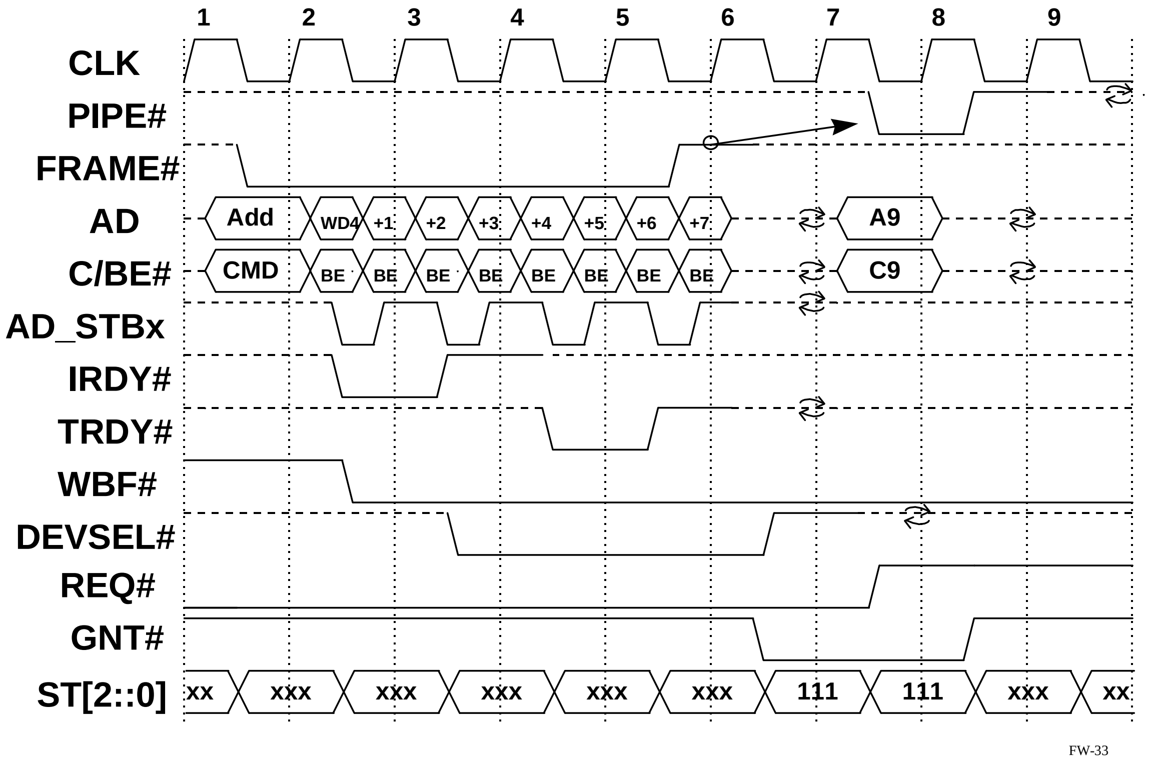


Figure 3-67: FW Followed by an A.G.P. Request

Figure 3-67 is an FW followed by an A.G.P. Request using the **AD** bus. In this case, a turn-around cycle is required on the **AD** bus since different agents are driving it. In this case, the corelogic was driving the **AD** bus for the FW transaction and the A.G.P. master drives it for the A.G.P. Request. The arbiter asserts the **GNT#** for the A.G.P. master when it samples **FRAME#** deasserted on the FW transaction. In this figure, the A.G.P. master starts as quickly as it can. The A.G.P. master is allowed to delay the assertion of **PIPE#** one clock if it desires.

3.5.3.5.3.13 PIPE# Followed by FW

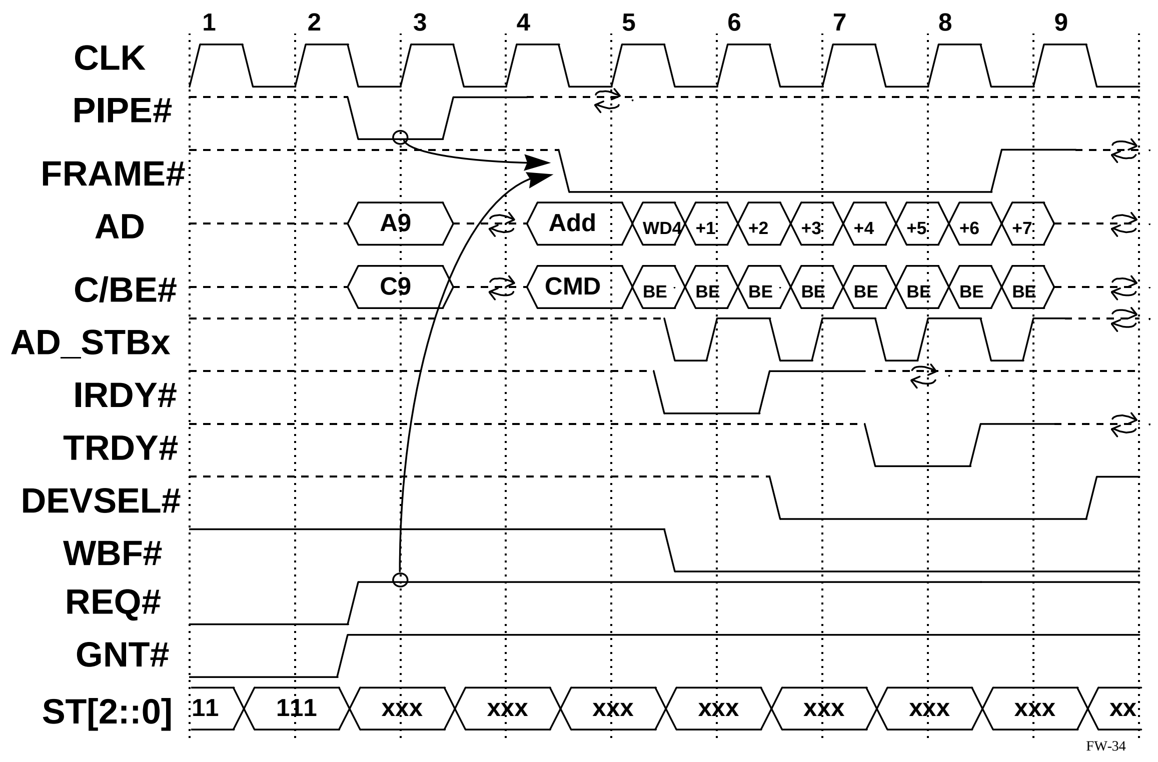


Figure 3-68: Request Enqueued Followed by an FW Transaction

Figure 3-68 is an FW transaction following an A.G.P. Request (single request). In this case, the **AD** and **C/BE#** buses must be turned around before the FW transaction can be initiated. This can be accomplished with a single turn-around access since the arbiter knows that the request transaction will be a single clock because **REQ#** is deasserted on the clock in which **PIPE#** is asserted. This indicates that a single request is being enqueued. Since the corelogic is the arbiter and the master of an FW transaction, the corelogic does not need an external **GNT#** and, therefore, can know in advance that it can start on the clock after **PIPE#** is deasserted.

3.5.3.5.3.13.1 PIPE# Followed by FW with Delay

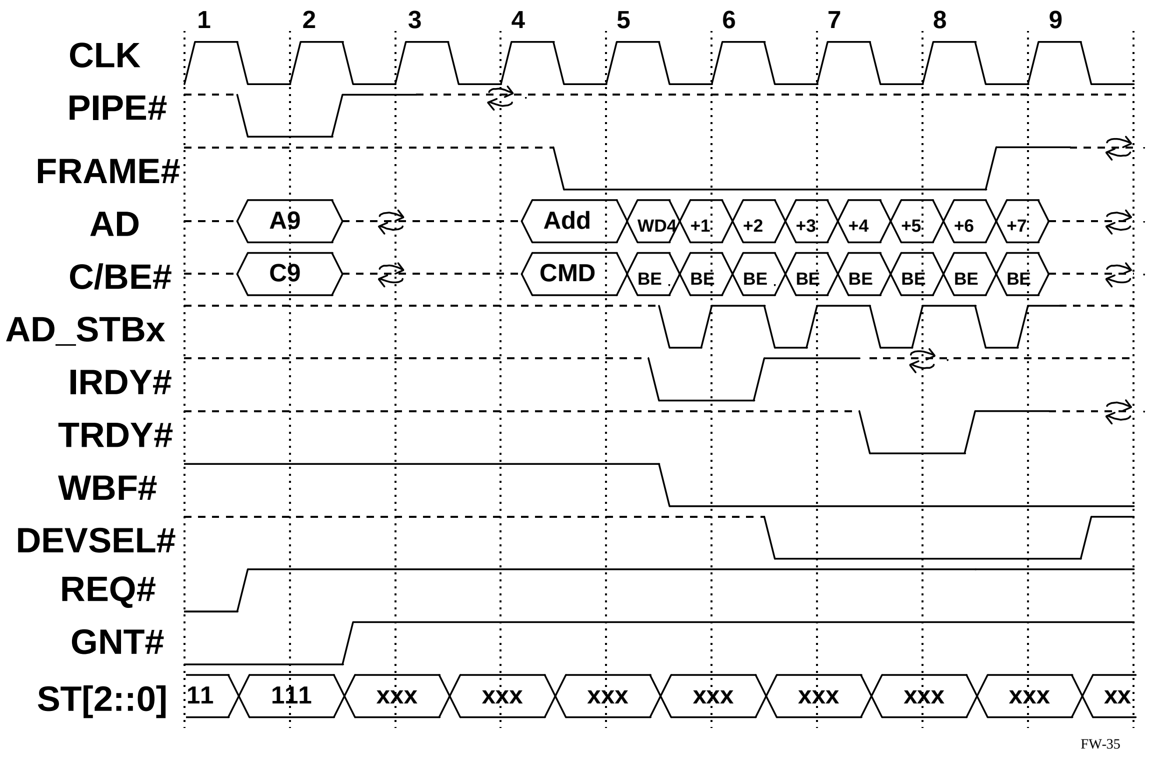


Figure 3-69: Request Followed by an FW Transaction

Figure 3-69 is the same as Figure 3-68 except that the corelogic takes an extra clock to start the FW transaction. In this case, the arbiter was slow in giving the internal **GNT#** or the FW interface took an extra clock to get started.

3.6 Arbitration

3.6.1 Introduction

This section describes the rules that the A.G.P. master's **REQ#** signal and the A.G.P. 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. 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 and 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 since that is a chipset specific implementation issue.

3.6.2 Master's REQ#

The A.G.P. master asserts its **REQ#** signal when it wants to either initiate a PCI cycle or enqueue a request using **PIPE#**. When the master deasserts **REQ#** depends on whether the current request is an A.G.P. or PCI request. When enqueueing A.G.P. Requests over the **AD** bus using **PIPE#**, the master must keep its corresponding **REQ#** asserted until one clock prior to deasserting **PIPE#**. When **PIPE#** is asserted and **REQ#** is deasserted (on the same clock), it indicates that the current request is the last to be enqueued for this transaction and **PIPE#** is required to be deasserted on the next clock returning the bus to an Idle condition. When the current bus operation is the enqueueing of requests on the **AD** bus, the master is not allowed to reassert **REQ#** to continue enqueueing more requests once **REQ#** has been deasserted. Therefore when **REQ#** is reasserted, it indicates that the A.G.P. master has a new request to initiate on the interface. This new request could be a new transaction to enqueue more A.G.P. Requests or to initiate a PCI transaction. The arbiter may utilize this information 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 A.G.P. transactions that enqueue requests. The master should concatenate as many requests into a single **PIPE#** transaction as possible to minimize the turn-around and idle bus cycles.

When an A.G.P. or PCI master issues a PCI transaction using **FRAME#** (and no other access requests are pending) it will deassert **REQ#** when it asserts **FRAME#**. If another access request is pending, the master will keep **REQ#** asserted. These rules are summarized in Table 3-20.

Table 3-20: A.G.P. Arbitration Rules

Current Access	**Next Access**		
	PCI	**A.G.P.**	**None**
PCI	Keep **REQ#** asserted.	Keep **REQ#** asserted.	Deassert **REQ#** when asserting **FRAME#.**
A.G.P.	Deassert **REQ#** one clock prior to deasserting **PIPE#**.	Deassert **REQ#** one clock prior to deasserting **PIPE#.**	Deassert **REQ#** one clock prior to deasserting **PIPE#**.

Figure 3-10 shows an access request using **PIPE#**. The master deasserts **REQ#** one clock prior to deasserting **PIPE#**. Simultaneously enqueueing requests on the **SBA** Port and the **AD** bus is *not* allowed. An A.G.P. master that is enabled to issue commands over the **SBA** Port is not allowed to generate commands with **PIPE#** over the **AD** bus.

3.6.3 GNT# and ST[2::0]

The A.G.P. arbiter will assert **GNT#** to allow the A.G.P. master 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 3-3.

3.6.4 A.G.P. Master

3.6.4.1 A.G.P. Master Initiating an A.G.P. Request.

When the A.G.P. master has **REQ#** asserted to request permission to use the **AD** bus to make an A.G.P. Request, it must assert **PIPE#** within two clocks of sampling **GNT#** asserted and **ST[2::0]** = 111 and the bus is in a state in which the master can start. Figure 3-70 illustrates the earliest in which **PIPE#** can be sampled asserted by the arbiter. Figure 3-71 illustrates the latest it can be sampled asserted.

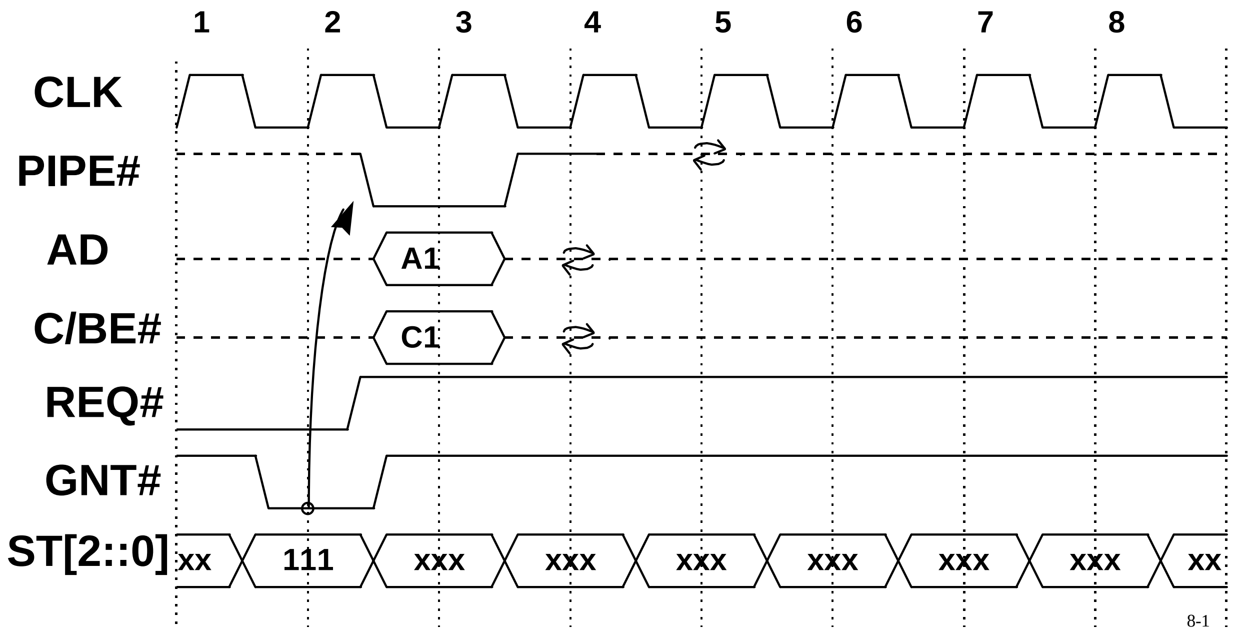


Figure 3-70: Single Address - No Delay by Master

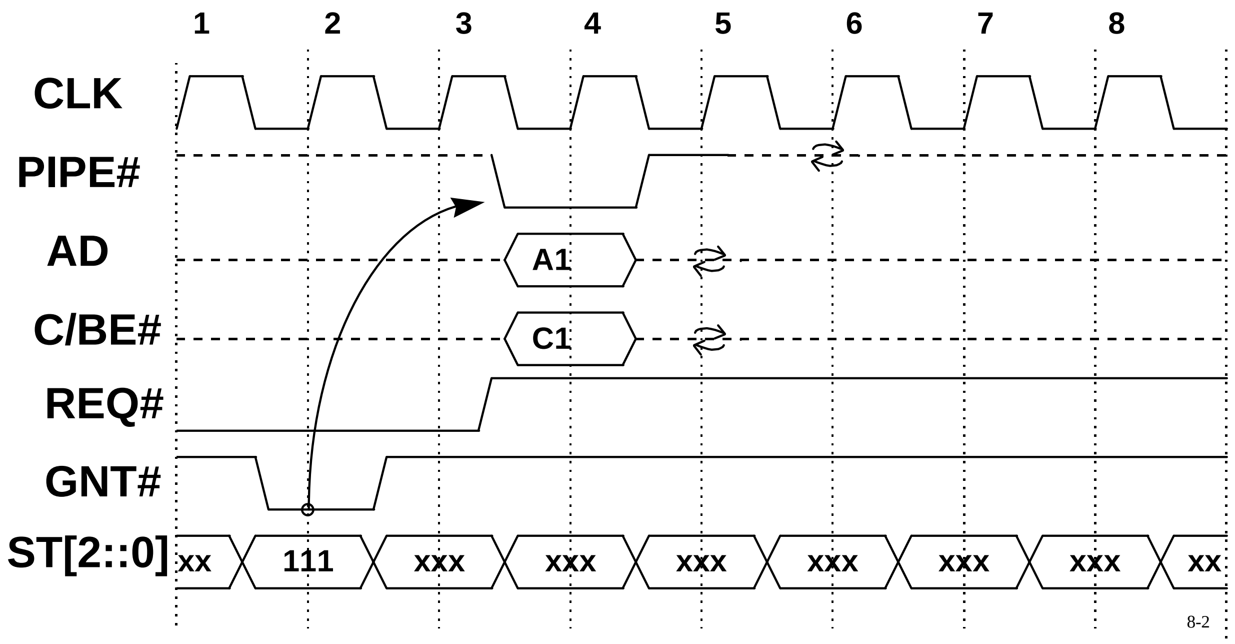


Figure 3-71: Single Address - Maximum Delay by Master

3.6.4.2 A.G.P. Master Initiating a PCI Transaction

When the A.G.P. master has **REQ#** asserted to request permission to use the **AD** bus to initiate a PCI transaction, it must follow the *PCI Local Bus Specification* in initiating a transaction by asserting **FRAME#**. This requires the master to assert **FRAME#** from the clock in which **GNT#** is sampled asserted, **ST[2::0]** = 111, and the bus is in a condition in which the master can start a transaction. The master does not get the option of taking one or two clocks to start. If the master delays starting a transaction, it runs the risk of having **GNT#** removed and losing its turn to use the bus. The A.G.P. master must follow the *PCI Local Bus Specification*, not the A.G.P. rules for this type of transaction. This means that the master is only allowed to initiate a PCI transaction when **GNT#** is asserted (**ST[2::0]** = 111) and the bus is in the Idle condition. Figure 3-72 illustrates a PCI transaction on the A.G.P. interface. Since **GNT#** is asserted on clock 2, the A.G.P. master is required to assert **FRAME#** so the arbiter samples it asserted on clock 3 otherwise, the master may[26] lose its opportunity to initiate the transaction. Since **GNT#** is deasserted on clock 3, the master is not allowed to assert **FRAME#** (for the address phase) on clock 4. If **GNT#** is deasserted on clock 3 and if **FRAME#** is not asserted on clock 3, the master is not allowed to start a PCI transaction. The arbiter is not required to keep **GNT#** asserted for multiple clocks to allow the master to initiate a PCI transaction.

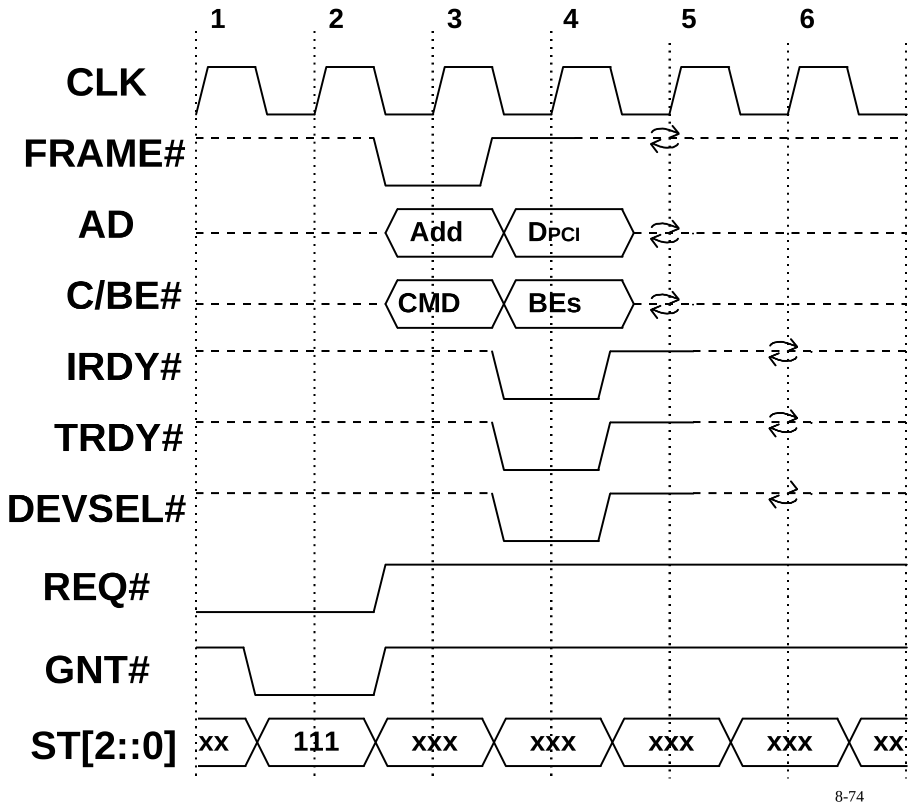


Figure 3-72: PCI Write Transaction on the A.G.P. Interface

[26] If the arbiter keeps **GNT#** asserted, the master is allowed to assert **FRAME#** later, but it must always be asserted the next clock after **GNT#** was asserted or it may lose its opportunity to start a transaction. If **GNT#** is only active for a single clock and the PCI master requires two clocks to get going (i.e., address stepping), the master may never gain access to the bus.

3.6.5 A.G.P. Arbiter

The arbiter is allowed to deassert (remove) **GNT#** at any time which allows it to prevent deadlocks from occurring.

When the arbiter removes a **GNT#** (deasserts it), the arbiter must not assert a new **GNT#** (to grant permission to initiate a request) to a different agent until the original master has been allowed sufficient time to initiate its transaction. For a PCI transaction, this is one clock while for an A.G.P. transaction, the master is allowed two clocks to start. In Figure 3-73, the arbiter grants the bus by asserting **GNT#** with **ST[2::0]** = 111 on clock 2. In this example, no master initiates a request by asserting **PIPE#** on clocks 3 or 4, or **FRAME#** on clock 3. In this example, **GNT#** is deasserted on clock 3. The earliest the arbiter can assert a new **GNT#** with **ST[2::0]** = 111 is clock 5 since **PIPE#** could be asserted on clock 3 or 4 and **FRAME#** could be asserted on clock 3.

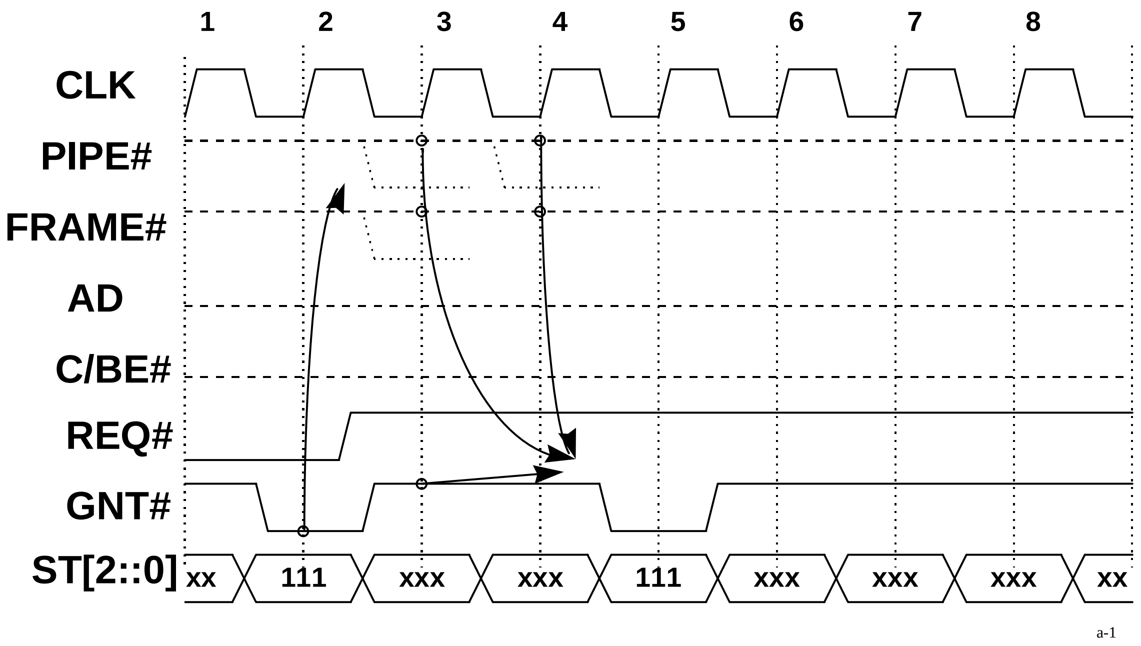


Figure 3-73: Arbiter Removes Grant - Master Does Not Start

Figure 3-74 illustrates the arbiter keeping **GNT#** asserted on clock 3 instead of deasserting it, as in Figure 3-73. In this case, the arbiter is required to delay the assertion of a new **GNT#** until clock 6. In this figure, the master is allowed to assert **PIPE#** on clocks 3, 4, and 5 or **FRAME#** on clocks 3 and 4. The assertion of **GNT#** on clock 3 allows **FRAME#** to be asserted on clock 4 and **PIPE#** to be asserted on clocks 4 and 5.

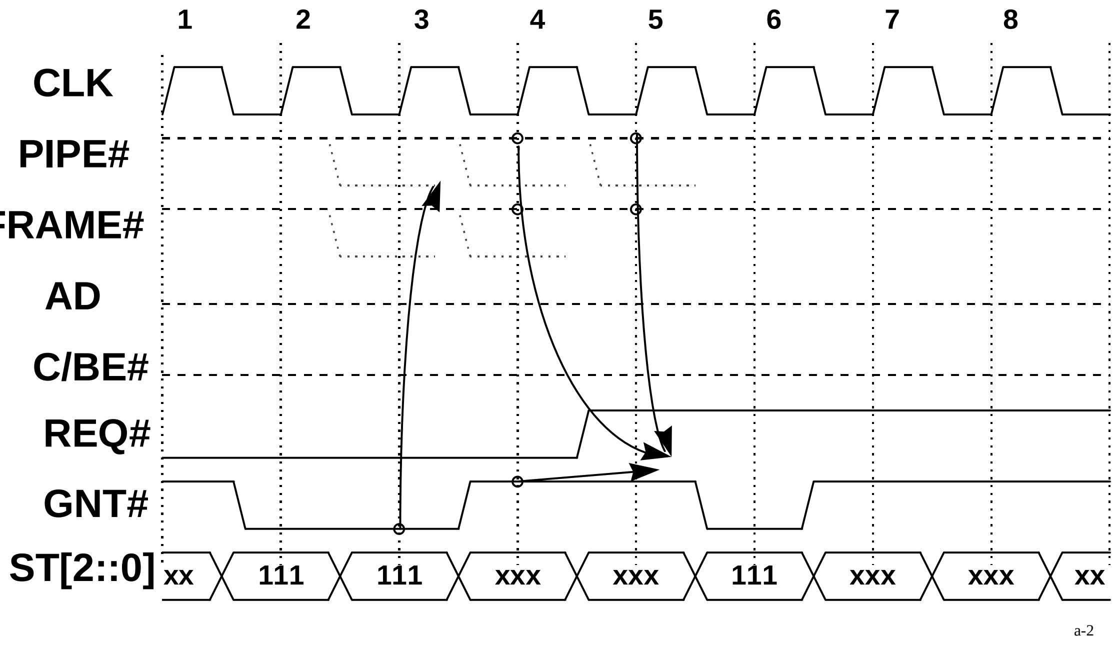


Figure 3-74: Arbiter Removes Grant Later - Master Still Does Not Start

3.6.6 GNT# Pipelining

In order to 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 as the arbiter can issue. 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 turn around, 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. When the current transaction is returning 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. The last data phase is defined as the last rising 1x clock of the read data transaction. This rule (along with proper use of **RBF#**) minimizes the amount of low priority read data buffering required by 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 since, by definition, every 1x clock is the last data phase of a transaction.

If the current transaction is providing write data, **GNT#** for the next transaction can be asserted on the clock immediately following the **GNT#** for the current write data transfer while there are less than four **GNT#**s already queued to request write data. The arbiter tracks the number of outstanding **GNT#**s for write data and can only assert a **GNT#** for a subsequent write transaction if there are less than four outstanding. The arbiter increments its **GNT#** for write data counter when it asserts **GNT#** for the master to provide write data and decrements the counter when the master asserts **IRDY#**[27]. The *master must be able to handle five pipelined* **GNT#**s (this assumes that a master does not 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 does not indicate another request) until one clock prior to the deassertion of **PIPE#**, it is impossible to pipeline a **GNT#** for another request transaction (PCI or **PIPE#**). 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 PCI, **GNT#** for the next transaction can be asserted immediately following **GNT#** for the current PCI transaction. 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 3-21 entries refer to the earliest clock off which the arbiter can drive **GNT#** asserted for the next cycle.

[27] The count is four 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 master is required to handle four outstanding pipelined transactions.

Table 3-21: Current/Next AD Activity

Current AD Transaction	**Next AD Transaction**		
	PCI or A.G.P. Request	**A.G.P. Read Data**	**A.G.P. Write Data**
PCI	**FRAME#** of current transaction sampled asserted.	**FRAME#** of current transaction sampled asserted and **RBF#** is deasserted for LP Read.	**FRAME#** of current transaction sampled asserted.
A.G.P. Command	**REQ#** sampled asserted after being deasserted.	**PIPE#** of current transaction sampled asserted and **RBF#** is deasserted for LP Read.	**PIPE#** of current transaction sampled asserted.
A.G.P. Read Data	Next to last data phase of current transaction.	Next to last data phase of current transaction and **RBF#** is deasserted for LP Read.	Next to last data phase of current transaction.
A.G.P. Write Data	Immediately following **GNT#** for current write while less than four outstanding **GNT#**s.	Immediately following **GNT#** for current write and **RBF#** is deasserted for LP Read, while less than four outstanding **GNT#**s.	Immediately following **GNT#** for current write while less than four outstanding **GNT#**s.

3.6.6.1 Pipelining GNT#s

The arbiter is allowed to pipeline grants when other bus transactions are in progress. This pipelining can be grouped into four different conditions:

1. A Request transaction followed by a Request transaction.
2. A Request transaction followed by a Data transfer.
3. A Data transfer followed by a Request.
4. A Data transfer followed by a Data transfer.

3.6.6.2 A Request Transaction Followed by a Request Transaction.

The arbiter is allowed to assert **GNT#** for a subsequent Request (PCI or A.G.P.) when **GNT#** has been deasserted for at least two clocks and **FRAME#** or **PIPE#** is not asserted during those two clocks which is illustrated in Figure 3-74. The arbiter is allowed to assert a new **GNT#** when **FRAME#** is sampled asserted. The arbiter is not allowed to pipeline a second request while an A.G.P. Request is currently active. (**REQ#** must be deasserted for at least one clock before it can be reasserted to indicate that a "new" request is pending.) There are six different combinations where a request is followed by another request. Each will be discussed in the following paragraphs.

An A.G.P. Request followed by a PCI Read transaction. (See Figure 3-75.)

An A.G.P. Request followed by a PCI Write transaction. (See Figure 3-76.)

A PCI (Read or Write) transaction followed by a PCI (Read or Write) transaction. (See Figure 3-77 and Figure 3-78.)

A PCI Read transaction followed by an A.G.P. Request (**PIPE#**). (See Figure 3-79.)

A PCI Write transaction followed by an A.G.P. Request (**PIPE#**). (See Figure 3-80.)

(Note: An A.G.P. Request cannot be followed by an A.G.P. Request since **REQ#** is required to be deasserted indicating the last request is being enqueued during the current transaction. The arbiter does not know a subsequent transaction is needed until the next clock.)

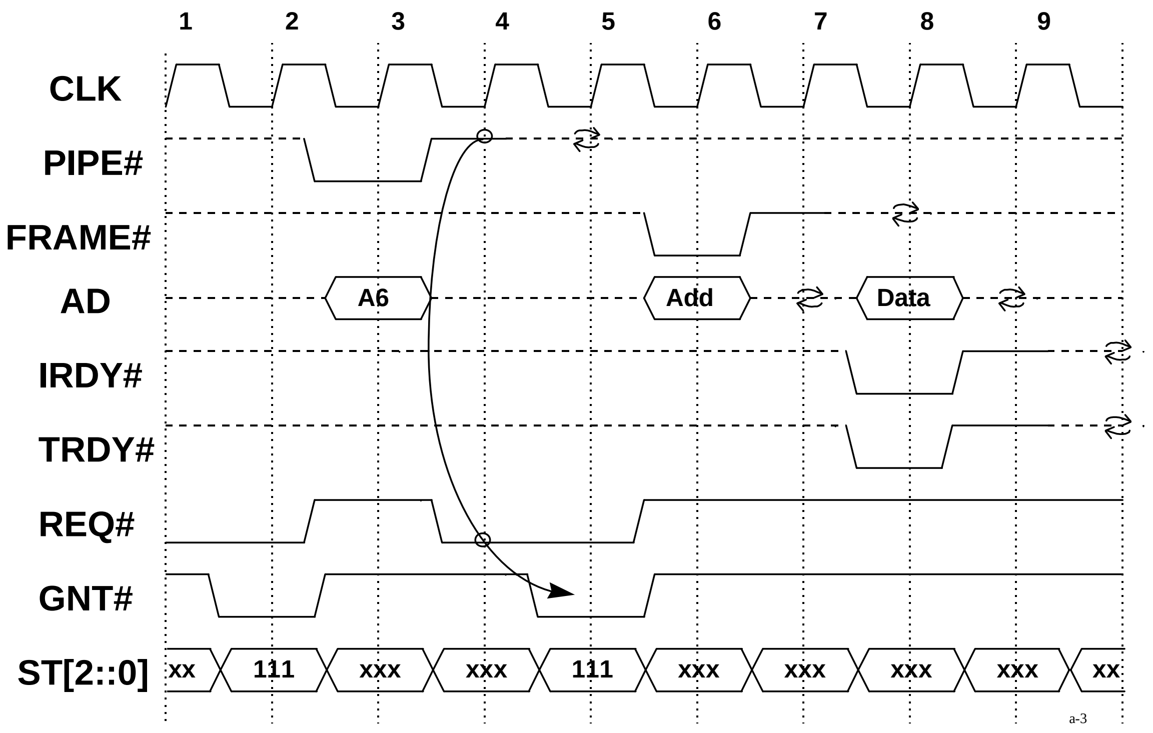


Figure 3-75: PIPE# Followed by an A.G.P. Master's PCI Read Transaction

Figure 3-75 illustrates the arbiter granting permission to start a request following an A.G.P. transaction. In this case, the arbiter does not know the master desires to use the bus until clock 4 when **PIPE#** is deasserted and **REQ#** is asserted. For the arbiter to assert **GNT#** on clock 5, it must use real time versions of **PIPE#** and **REQ#**. If latched versions are used, **GNT#** would be delayed until clock 6. The limiting condition for getting a subsequent **GNT#** asserted in this example is the assertion of **REQ#**. **REQ#** is used to indicate when the last request is present on the **AD** bus.

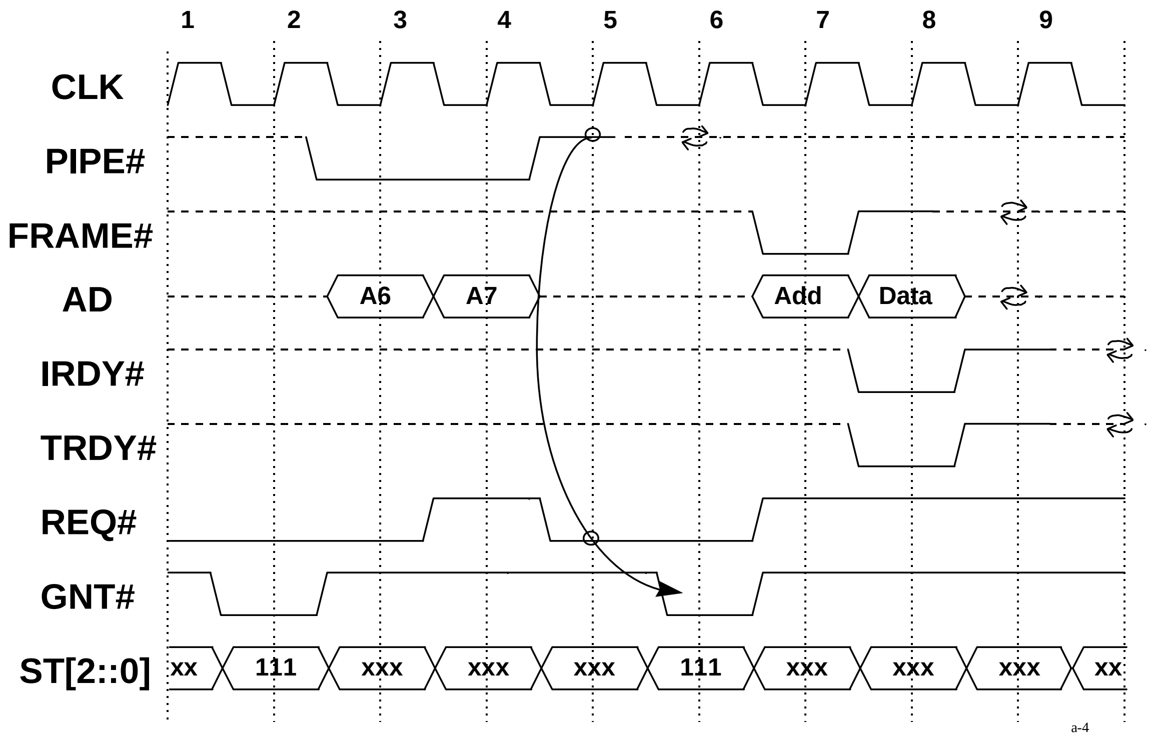


Figure 3-76: Double Phase PIPE# Followed by an A.G.P. Master PCI Write Transaction

Figure 3-76 is the same as Figure 3-75 except that multiple requests are enqueued by the master. Again, **REQ#** is the gating condition to get **GNT#** asserted. In this case, **REQ#** is not reasserted until clock 5, thus delaying **GNT#** until clock 6. If latched versions are used, the **GNT#** would be delayed until clock 7.

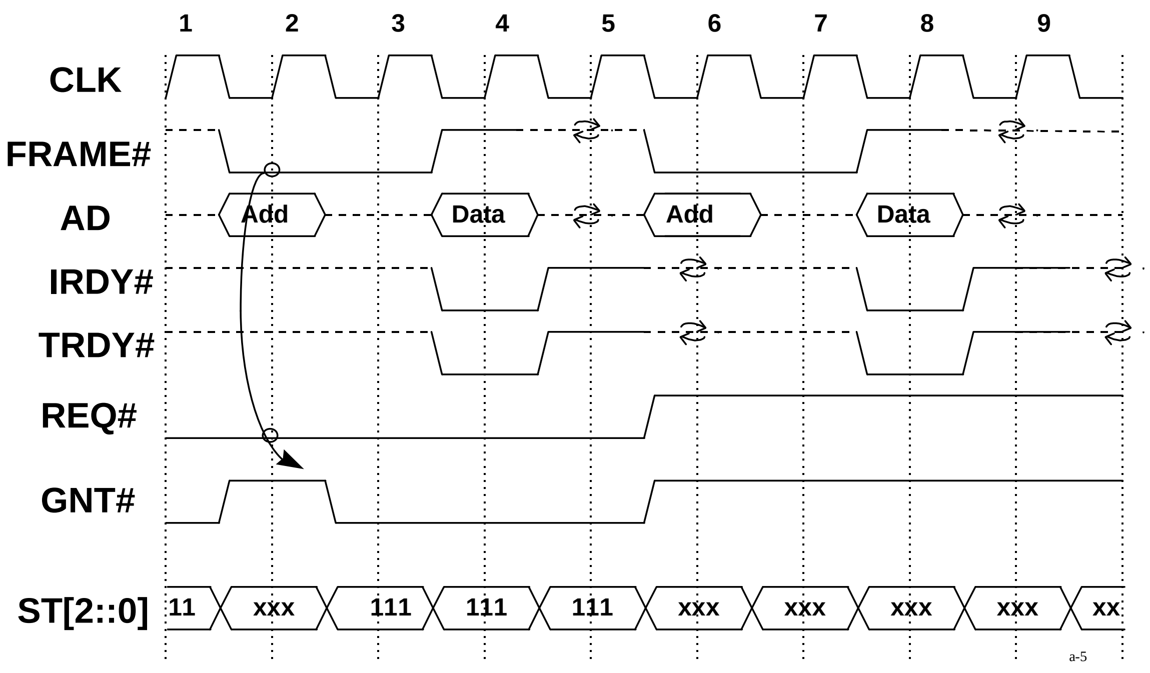


Figure 3-77: Back-to-Back PCI Read Transactions

Figure 3-77 illustrates that the arbiter can assert **GNT#** immediately following the assertion of **FRAME#** to grant permission to the "next" agent to use the **AD** bus to initiate a request. The next agent may be the current agent or the other A.G.P. agent. (When the agent is the corelogic, **GNT#** is an internal signal.) The arbiter could have allowed **GNT#** to remain asserted on clock 2 when the first and second transactions are by the same master. In this case, the master will not initiate the next transaction until clock 6 because a turn-around cycle is required between the read data and the address of the next transaction. The arbiter is allowed to deassert **GNT#** at any time. This is why the arbiter deasserts **GNT#** at the same time **FRAME#** is asserted which occurs on clock 6. If the corelogic had delayed asserting **FRAME#** beyond clock 6, it would have lost its opportunity to start a transaction.

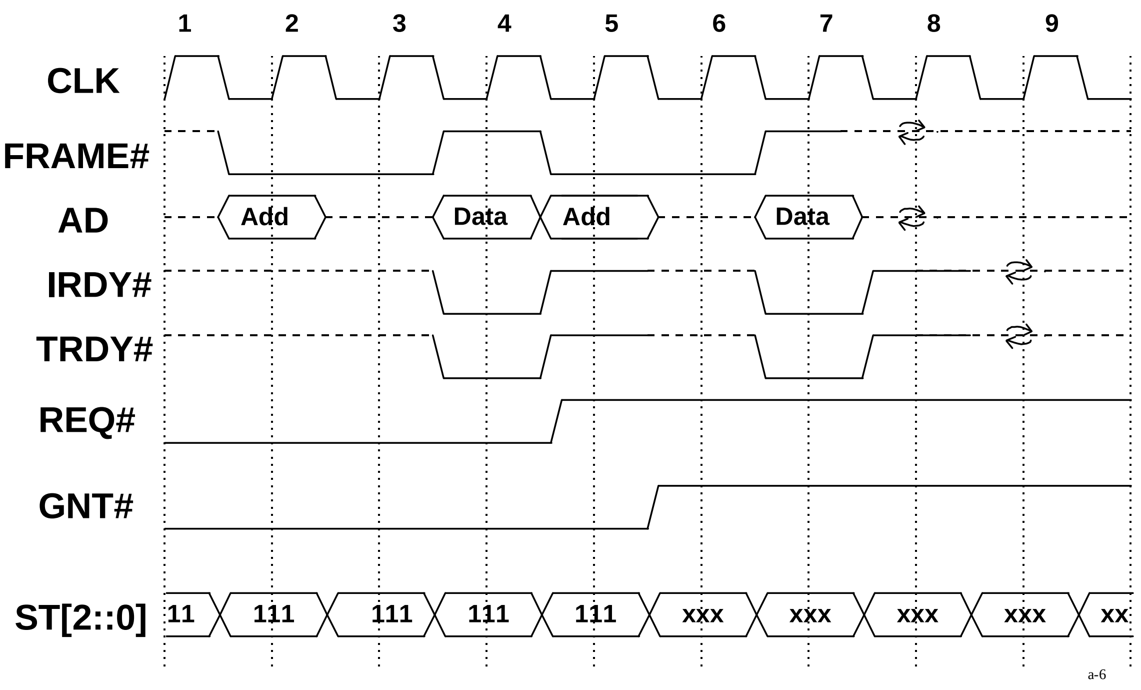


Figure 3-78: Fast Back-to-Back PCI Write Transactions

Figure 3-78 illustrates the same operation as Figure 3-77 except the first transaction is a Write and a turn-around between the first data and the second address phase is not required. In this case, the arbiter does not remove **GNT#** on clock 2 as it did in Figure 3-77, but keeps it asserted indicating that the current master is requesting to do multiple transactions. The master samples **GNT#** asserted on clock 4 which allows it to start the next transaction of clock 5 as long as the first transaction was a write. Otherwise, a turn-around cycle would be required on clock 5. If the first transaction was terminated with Retry, the arbiter does not have sufficient time to remove **GNT#** before the master could initiate the second transaction. If the corelogic (target machine) indicates to the arbiter that it will assert **STOP#** on clock 4, it could cause the arbiter to remove **GNT#** for clock 4. Otherwise, the master will initiate the second transaction (which may be a repeat of the current transaction) and most likely will complete the same as the first transaction. At a minimum, the arbiter must remove **GNT#** on the subsequent (third) transaction. The master is required to deassert **REQ#** on clock 5 (when the bus would have gone Idle) and one clock after in this case. This would allow the arbiter to grant the bus to a different resource.

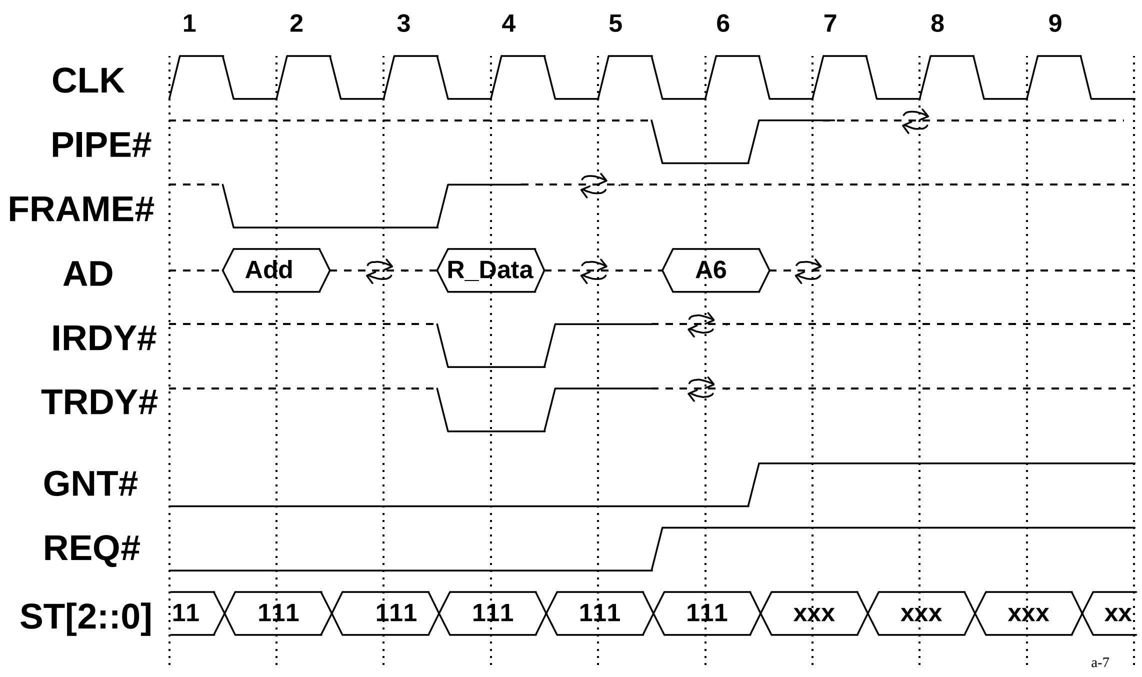


Figure 3-79: PCI Read Transaction Followed by PIPE#

Figure 3-79 illustrates a PCI read transaction followed by a **PIPE#** transaction. In this case, the **AD** bus requires a turn-around cycle between the PCI address phase and the Read data phase. A second turn-around cycle is then required between the read data and the first request of the A.G.P. transaction. The arbiter does not deassert **GNT#** between the transactions and, thereby allows the A.G.P. master to initiate the request as soon as possible. (The arbiter does not need to know if the current transaction is a read or write.) In this case, the A.G.P. master is not allowed to start the transaction until clock 6 even though the **GNT#** was asserted on clock 3. If the A.G.P. master always uses two clocks to get the transaction started and knows that the PCI transaction completed on clock 4, it uses the turn-around cycle to pipeline its request, thereby saving itself one dead clock on the bus.

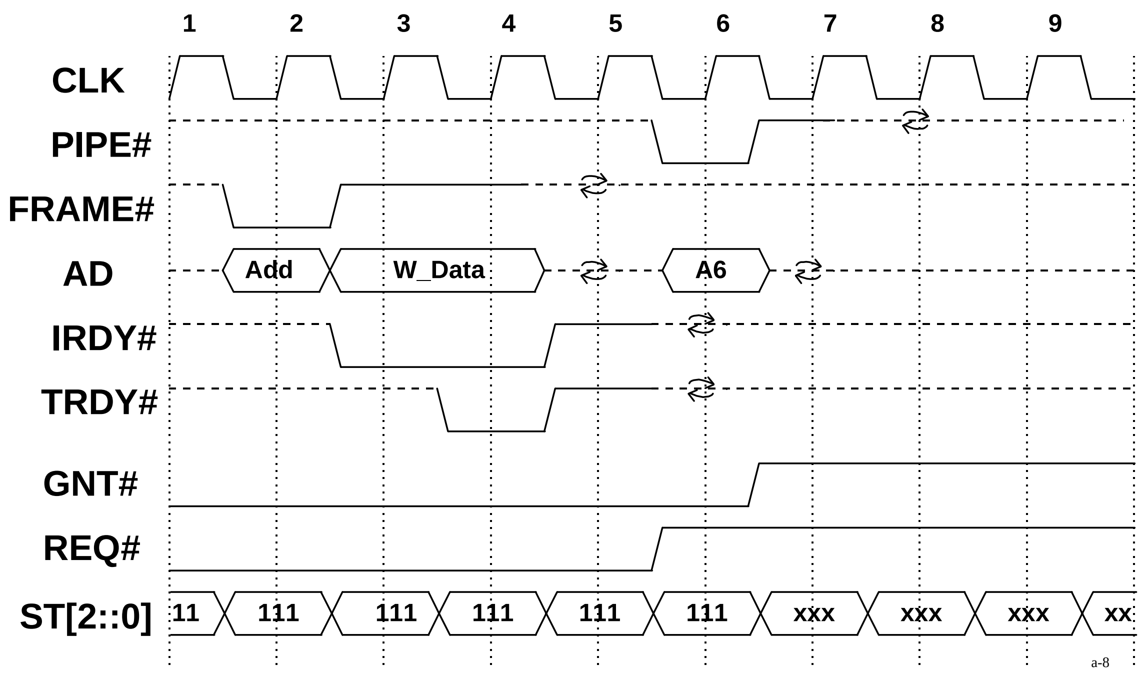


Figure 3-80: PCI Write Transaction Followed by PIPE#

Figure 3-80 is the same as Figure 3-79 except that the PCI transaction is a write instead of a read. In this case, a turn-around is not required between the data phase of the PCI transaction and the A.G.P. Request, but a dead clock is required between any PCI transaction and A.G.P. transaction. The master of the A.G.P. transaction (in this example) is required to delay the assertion of **PIPE#** until after the dead clock. Again the fact that **GNT#** is asserted on clock 3 and held until clock 6 is acceptable since the **GNT#** is to the same master and it knows when the current transaction completes and when the subsequent transaction is allowed to start.

3.6.6.3 A Request Transaction Followed by a Data Transfer.

The arbiter is allowed to assert **GNT#** to pipeline data transfers as soon as the Request has been started. Note that the logic starting the transaction may use latched signals, thereby causing the **GNT#** with **ST[2::0]** = 111 to remain after the Request has started. The A.G.P. master is required to ignore **GNT#** asserted when **ST[2::0]** = 111 except on the clock in which it desires to start a transaction and is allowed to do so. In all other cases, this condition on the interface is meaningless.

An A.G.P. Request followed by an A.G.P. Read transaction. (See Figure 3-81.)

An A.G.P. Request followed by an A.G.P. Write transaction. (See Figure 3-82 and Figure 3-83.)

A PCI Read transaction followed by an A.G.P. Read transaction. (See Figure 3-84.)

A PCI Read transaction followed by an A.G.P. Write transaction. (See Figure 3-85.)

A PCI Write transaction followed by an A.G.P. Read transaction. (See Figure 3-86 and Figure 3-87.)

A PCI Write transaction followed by an A.G.P. Write transaction. (See Figure 3-88 and Figure 3-89.)

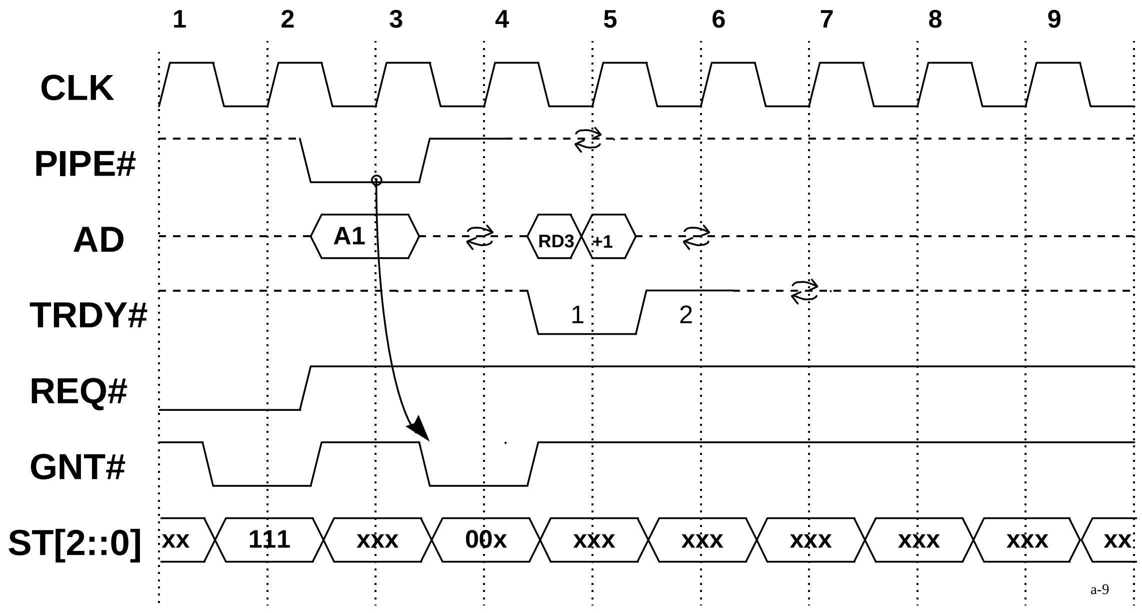


Figure 3-81: PIPE# Followed by Read Data

Figure 3-81 is an example of an A.G.P. Request followed by an A.G.P. read data transfer. In this example, the arbiter deasserts **GNT#** for clock 3 and when it samples **PIPE#** asserted on clock 3, it asserts **GNT#** indicating that previously requested read data will be returned when the current transaction completes. The earliest the arbiter can assert **GNT#** with **ST[2::0]** = 00x is clock 4. The turn-around cycle is required on clock 4 since ownership of the **AD** bus is changing.

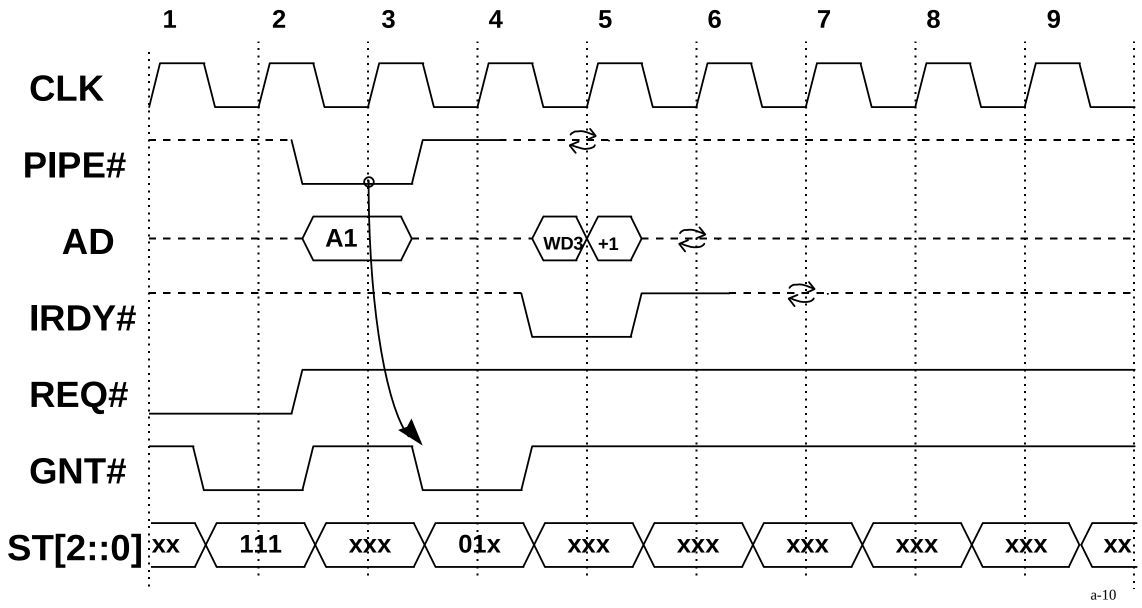


Figure 3-82: PIPE# Followed by Write Data

Figure 3-82 is an A.G.P. Request followed by an A.G.P. Write transaction. In this example, no turn-around is required between the request and the write data transfer. The dead clock on clock 4 is caused because the arbiter is not allowed to enqueue a data transaction until **PIPE#** is sampled asserted. Figure 3-83 shows the case where **PIPE#** is asserted long enough for the next **GNT#** to be pipelined, thereby allowing no dead clock between the two transactions. The arbiter is not allowed to enqueue an A.G.P. data transfer until **PIPE#** or **FRAME#** is asserted or until **GNT#** with **ST[2::0]** = 111 has been deasserted for two clocks. In this example, **PIPE#** was sampled asserted on clock 3 and, therefore, the arbiter is allowed to assert **GNT#** with **ST[2::0]** = 01x indicating that when the current transaction completes, the write data is to be transferred across the bus. **GNT#** could have remained asserted on clock 3 with **ST[2::0]** = 111. If **GNT#** remained deasserted on clock 4 with **ST[2::0]** = 111, the write data transfer would have been delayed. This condition can occur when the arbiter uses a latched version of **PIPE#**.

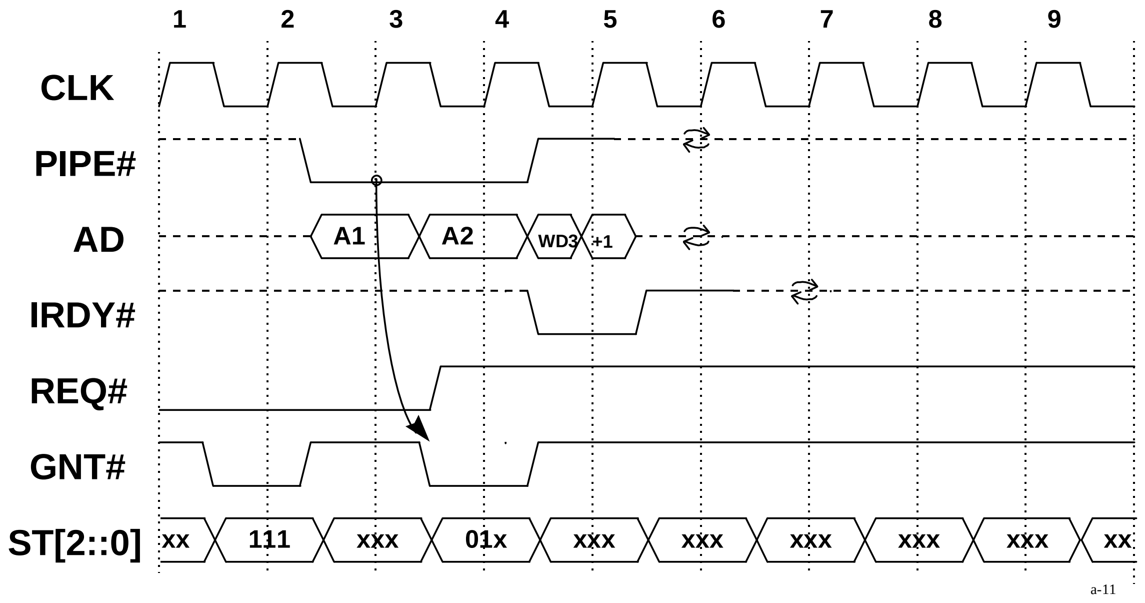


Figure 3-83: Double Phase PIPE# Followed by Write Data

Figure 3-83 is a write data transfer following an A.G.P. Request. In this case, a turn-around cycle is not required since the same agent owns the **AD** bus for both transactions. The arbiter asserts **GNT#** indicating the A.G.P. master is to provide write data after the current transaction completes once it detects that the current transaction has been initiated. The arbiter in this figure removed the **GNT#** on clock 3 since the A.G.P. master is required to assert **PIPE#** on either clock 3 or 4. However, the arbiter could have left **GNT#** asserted with **ST** = 111 on clock 3 or longer. If the latter had occurred, the write data transaction would have been delayed and bus bandwidth would have been wasted. When this bus operation is supported by the master, careful review of the setup and hold times of the 1x and 2x timing parameters is warranted. See note 1 under Table 4-7 for details.

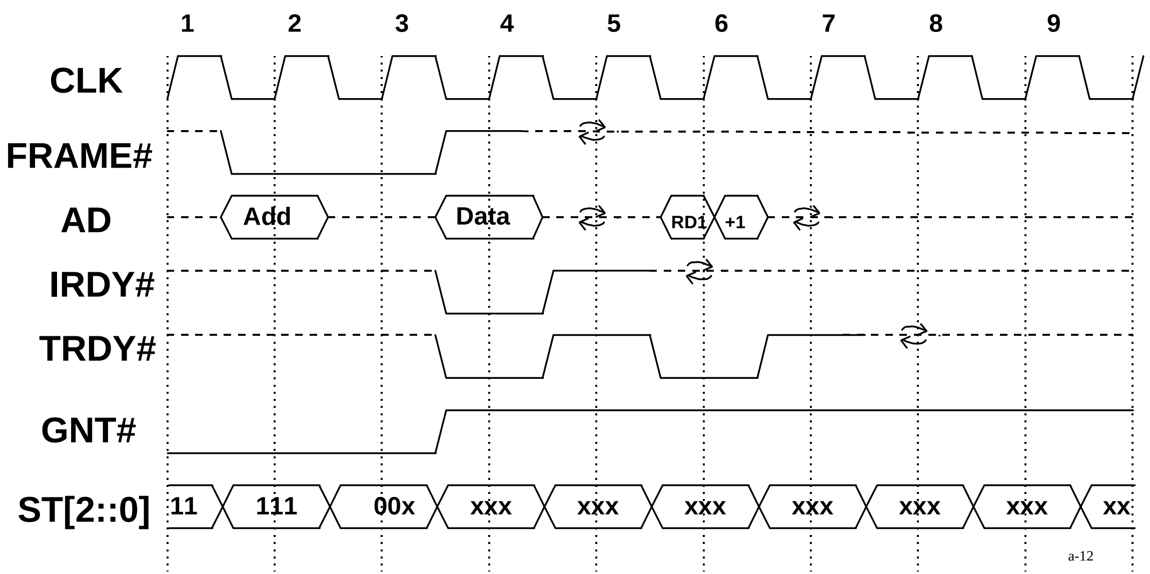


Figure 3-84: Master's PCI Read Transaction Followed by Read Data

Figure 3-84 is a PCI transaction that is initiated by an A.G.P. master followed by an A.G.P. read data being returned to the master. A turn-around cycle is required between the PCI transaction and the A.G.P. transaction even though ownership of the **AD** bus does not change. In this case, the arbiter samples **FRAME#** asserted on clock 2 and asserts a new **GNT#** to the master indicating that previously requested read data is being returned to the master. In this case, the arbiter uses the state of **RBF#** to determine if the assertion of **GNT#** is allowed for clock 3. If **RBF#** is asserted, the arbiter is not allowed to return the read data if it is low priority. Since the PCI transaction is in process, the return of the read data is not allowed to start until the PCI transaction completes. In this example, the ownership of the **TRDY#** signal does not change hands and, therefore, a turn-around is not required. A dead cycle, however, is required between PCI and A.G.P. transactions.

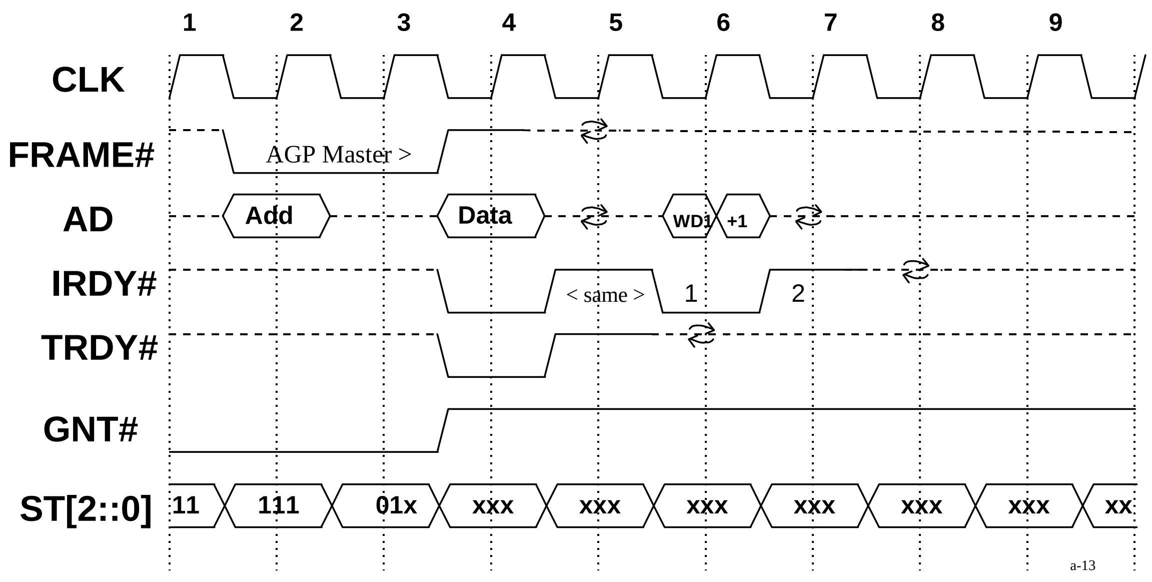


Figure 3-85: A.G.P. Master's PCI Read Transaction Followed by Write Data

Figure 3-85 is basically the same as Figure 3-84 except that write data is being provided by the master. In this case, a turn-around cycle is required since ownership of the **AD** bus is changing. In this case, the dead clock is required because of a PCI to A.G.P. transition. The arbiter in this example asserts the **GNT#** for the data movement at the earliest possible time. The A.G.P. master is required to monitor the bus until the current transaction completes and then must provide the write data. In this case, the arbiter could have waited until clock 4 to assert **GNT#** and **ST** = 01x without causing additional delays. When the A.G.P. master does not require an additional clock to start the write data transfer, the assertion of **GNT#** on clock 5 would not cause a delay. However, if the A.G.P. master uses two clocks to initiate the transfer, a dead clock would appear on the bus. Therefore, it is recommended that the arbiter enqueue the next **GNT#** at the earliest time possible to give the agent providing the data as much notification as possible to minimize the dead clocks on the interface.

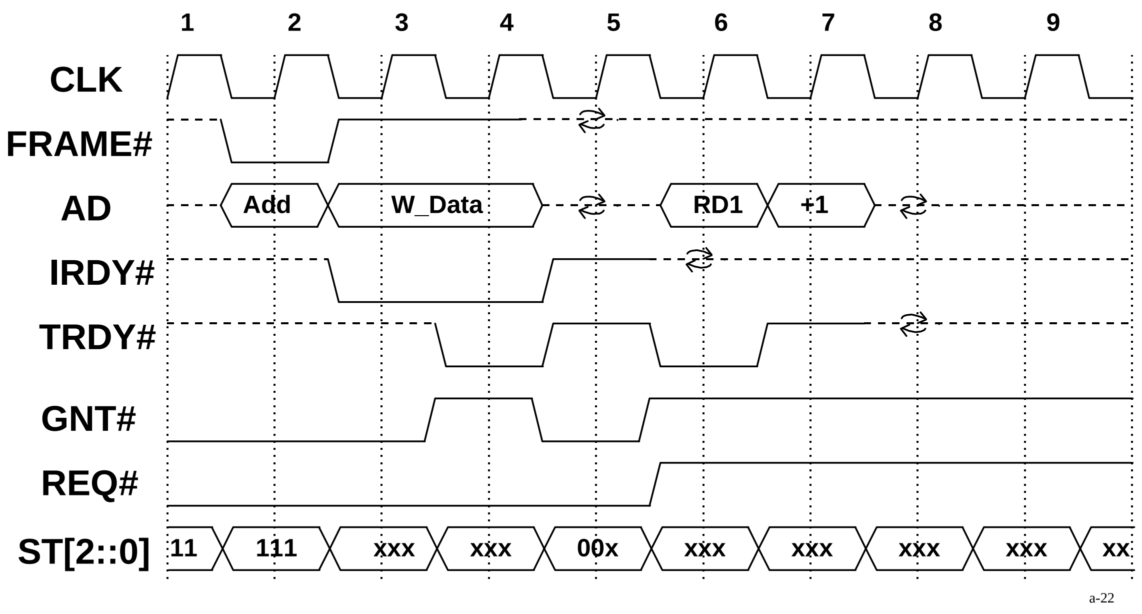


Figure 3-86: A.G.P. Master's PCI Write Followed by Read Data

Figure 3-86 is a PCI transaction initiated by the A.G.P. master and is followed by the return of read data. Since ownership of the **AD** bus changes, a turn-around cycle is required. Note that ownership of **IRDY#** and **TRDY#** does not change and, therefore, no turn around is required for them.

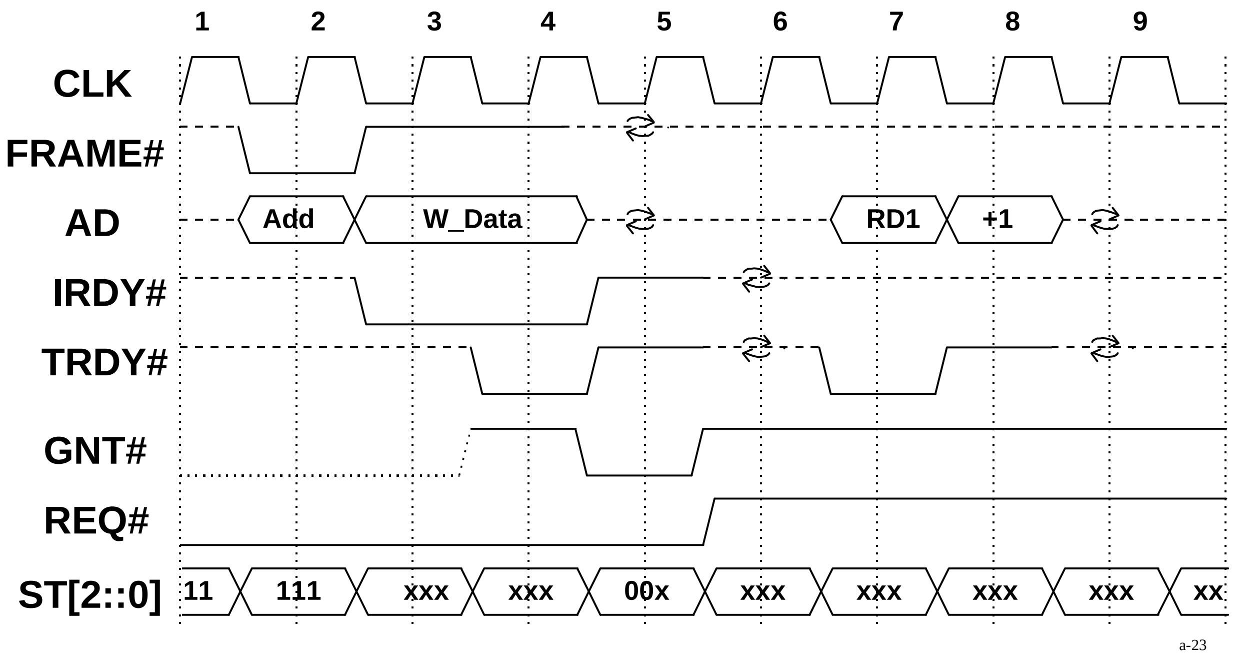


Figure 3-87: Corelogic's PCI Write Followed by Read Data

Figure 3-87 is the same as Figure 3-86 except that the corelogic is the agent initiating the PCI write transaction. In this case, ownership of the control signals changes but ownership of the **AD** bus does not change. Therefore, the return of the read data is delayed an additional clock due to the turn-around on **TRDY#** which cannot occur until clock 6.

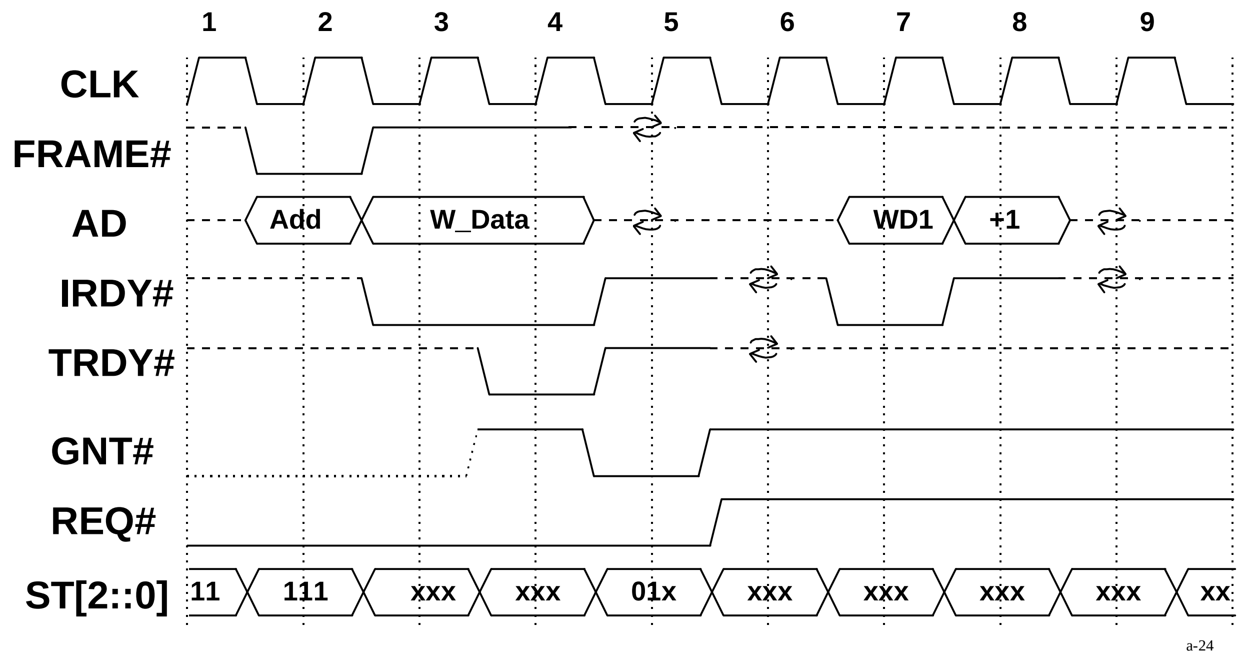


Figure 3-88: Corelogic's PCI Write Transaction Followed by Write Data

Figure 3-88 is the same as Figure 3-87 except that write data is being requested instead of read data being returned. In this case, both the AD bus and the control signals require turn-around cycles.

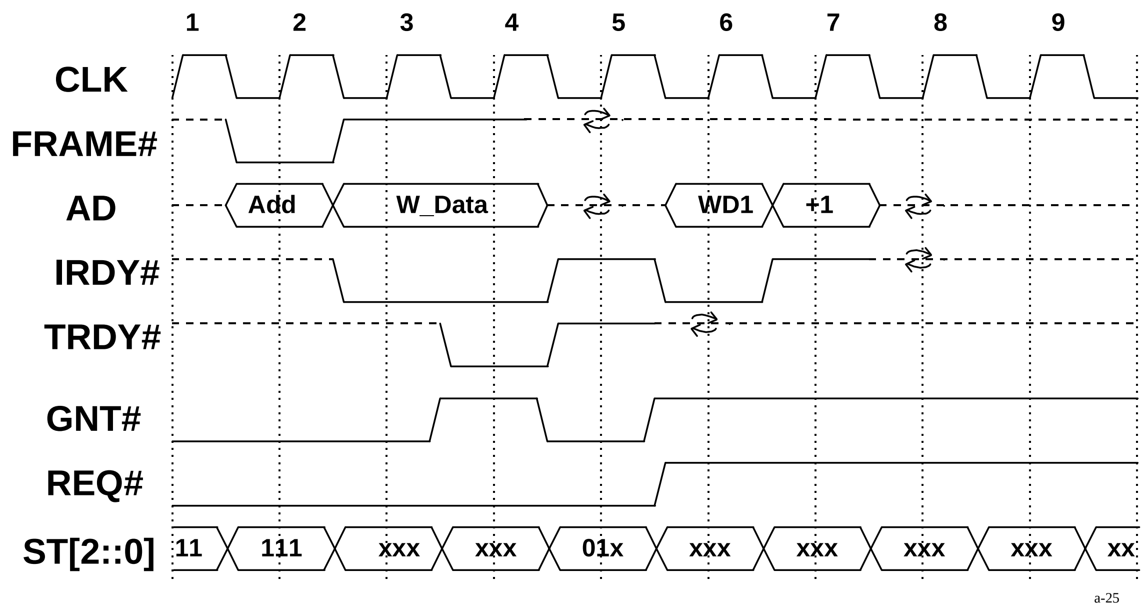


Figure 3-89: A.G.P. Master's PCI Write Transaction Followed by Write Data

Figure 3-89 is a the same as Figure 3-86 except that write data is also provided by the A.G.P. master. In this case, neither the AD bus nor the control signals change ownership, therefore, no turn-around cycles are required. The dead clock between the transactions is required when the bus is changing from one protocol to another. In this case, the change is from a PCI transaction to an A.G.P. transaction.

3.6.6.4 A Data Transfer Followed by a Request

During an A.G.P. Read transaction, the arbiter is not allowed to enqueue the next transaction until the read data transaction enters the last data phase.

An A.G.P. Read transaction followed by an A.G.P. Request. (See Figure 3-90 and Figure 3-91.)

An A.G.P. Write transaction followed by an A.G.P. Request. (See Figure 3-92 and Figure 3-93.)

An A.G.P. Read transaction followed by a PCI (Read or Write) transaction. (See Figure 3-94 and Figure 3-95.)

An A.G.P. Write transaction followed by a PCI (Read or Write) transaction. (See Figure 3-96 and Figure 3-97.)

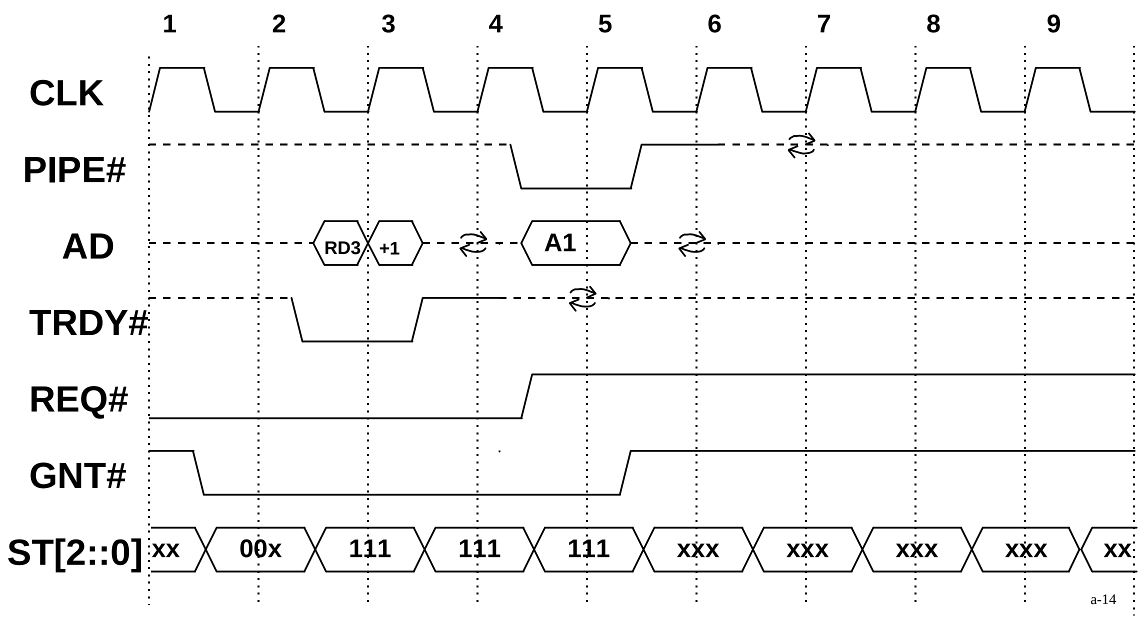


Figure 3-90: Single Read Data Followed by PIPE#

Figure 3-90 is an A.G.P. Read transaction followed by an A.G.P. Request. Ownership of the **AD** bus changes; therefore, a turn-around cycle is required. In this case, the arbiter indicated to the master that read data is being returned by asserting **GNT#** on clock 2 with **ST** = 00x. Since grants for A.G.P. data transfers are latched and remembered by the A.G.P. master, they only last a single clock. The following clock, the arbiter is allowed to pipeline another **GNT#** since the read transaction has entered the last data phase.

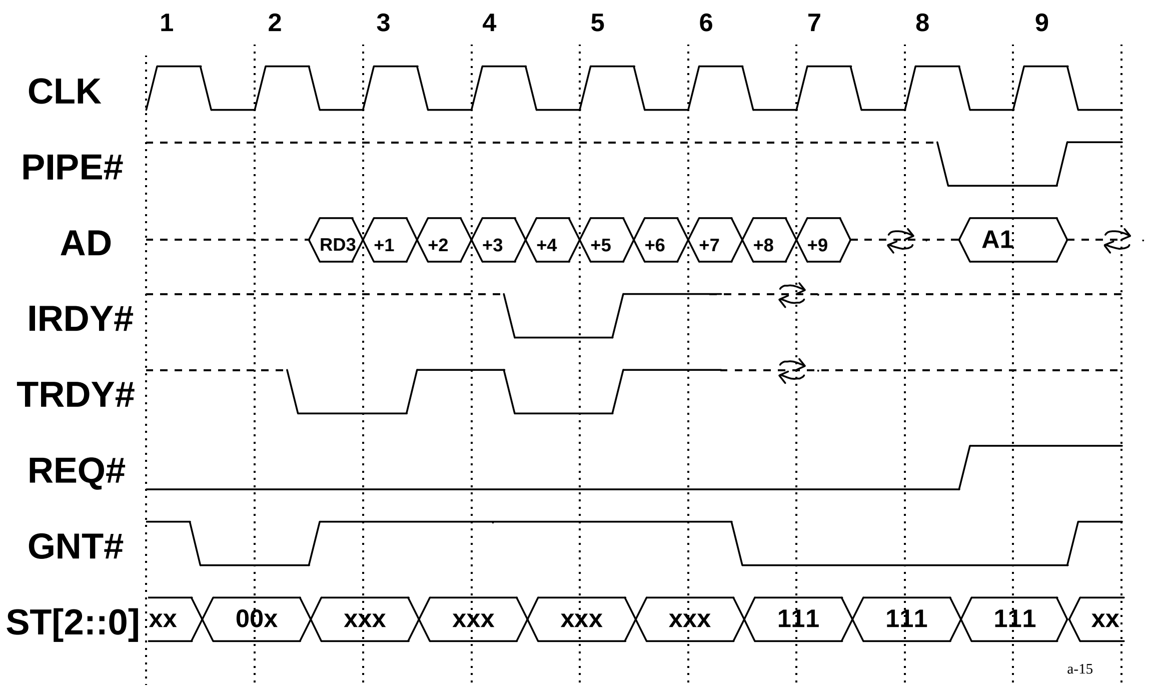


Figure 3-91: Burst Read Data Followed by PIPE#

Figure 3-91 is the same as Figure 3-90 except that the data transfer lasts longer, and, therefore, the **GNT#** to start a Request is delayed until the read data transfer enters the last data phase which occurs on clock 7. The arbiter is not allowed to assert **GNT#** on clocks 3-6. The arbiter could delay the assertion of **GNT#** beyond clock 7 but bus bandwidth would be wasted.

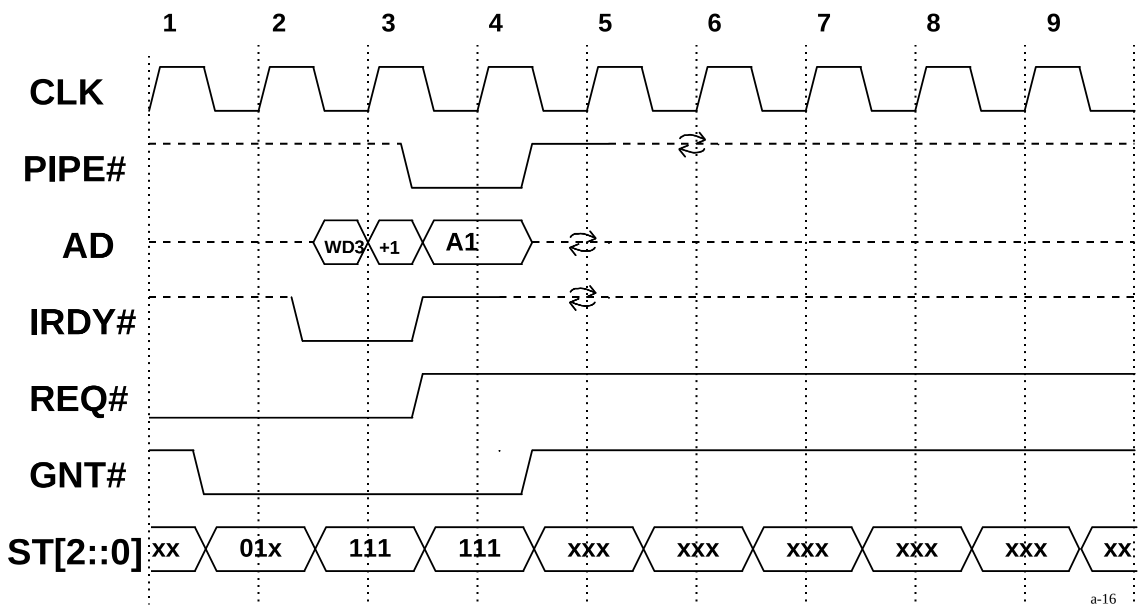


Figure 3-92: Single Write Data Followed by PIPE#

Figure 3-92 is an A.G.P. write transaction followed by an A.G.P. Request. This figure is similar to Figure 3-90 except the data transfer is a write instead of a read. In this example, a turn-around cycle is not required between the data transfer and the request since ownership of the **AD** bus does not change. The data transfer is indicated on clock 2. The arbiter is allowed to enqueue the next transaction immediately on a write transaction with up to a maximum of four transactions outstanding. In this case, there is only one outstanding transaction since the subsequent transaction was not a write.

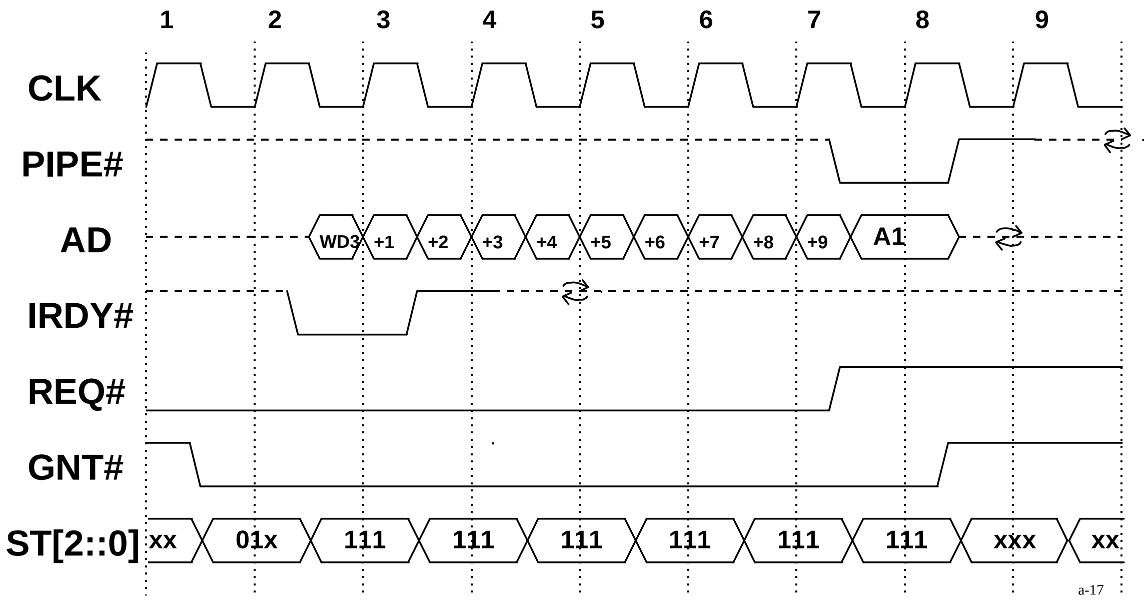


Figure 3-93: Burst Write Data Followed by PIPE#

Figure 3-93 is the same as Figure 3-92 except the write transaction is longer and illustrates how the **GNT#** to start the Request is pending for a number of clocks. Since the arbiter is allowed to remove a grant at any time, the master cannot assume that it can start at the end of the write transaction until **GNT#** is sampled asserted (with **ST** = 111) on clock 6 or 7. If **GNT#** was asserted on clocks 3-5 and deasserted on clock 6, the master would not be allowed to start the Request. The A.G.P. master is not allowed to “remember” that it had been granted access to the bus but must only check **GNT#** (**ST** = 111) when it is allowed to actually start the transaction.

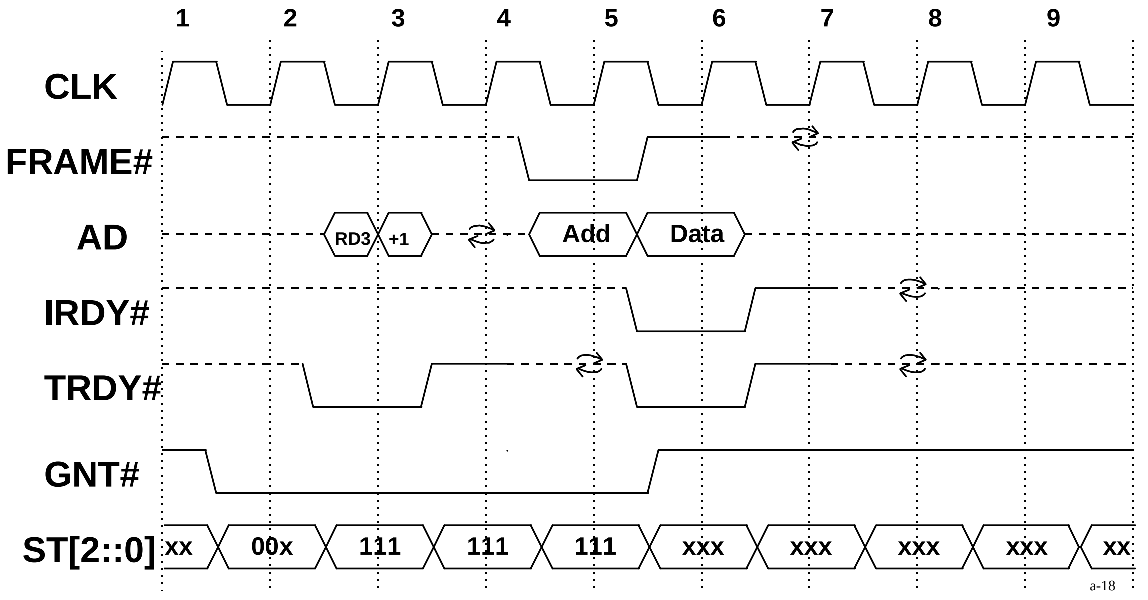


Figure 3-94: Single Read Data Followed by A.G.P. Master’s PCI Transaction

Figure 3-94 is an A.G.P. read transaction followed by a PCI request. In this case, **TRDY#** is required to have a turn-around cycle. Therefore, a dead clock is required between the A.G.P. and PCI transactions. In this example, the arbiter is allowed to pipeline a **GNT#** on clock 3 since the Read transaction is in the last data phase. If the arbiter caused **GNT#** to be deasserted on clock 4, the PCI master would not be allowed to start the request. The PCI master is only allowed to initiate a request when **GNT#** is asserted (**ST** = 111 for an A.G.P. master initiating a PCI transaction) and **FRAME#, IRDY#**, and **TRDY#** are deasserted. This condition occurs on clock 4, the master asserts **FRAME#**, and the address phase completes on clock 5. When supporting both PCI and A.G.P. interfaces, each state machine needs to know when the other type of transaction is occurring. For example, the PCI state machine needs to know when the A.G.P. interface is active.

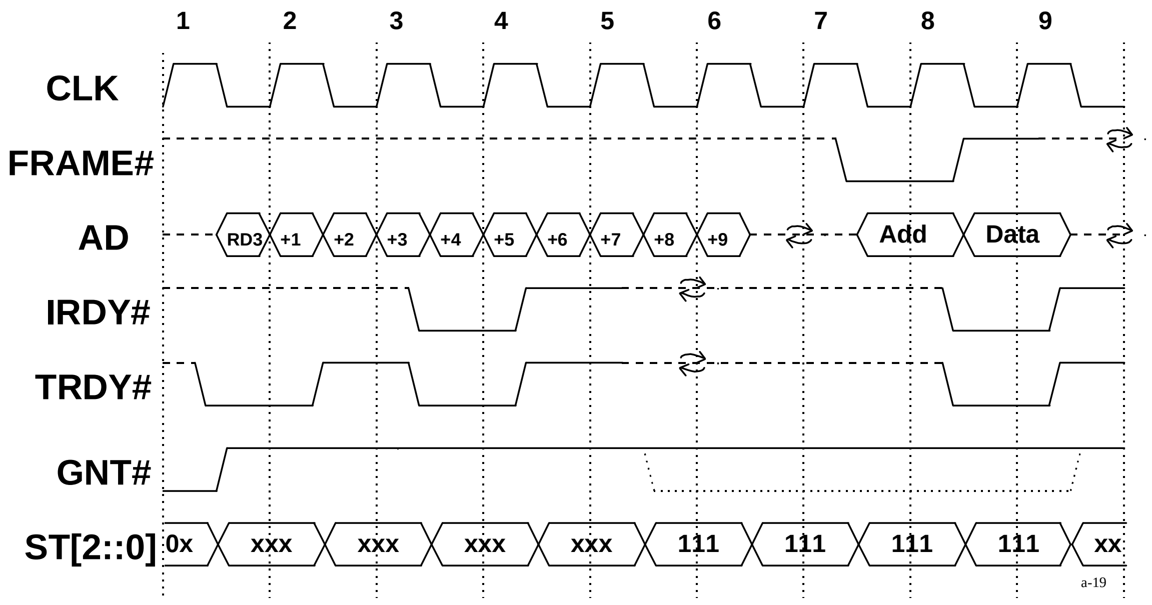


Figure 3-95: Burst Read Data Followed by A.G.P. Corelogic's PCI Transaction

Figure 3-95 is an example of an A.G.P. read followed by a PCI transaction. In this case, the PCI master is required to put a turn-around cycle in because the bus protocol changes even though there is no contention on **IRDY#** or **TRDY#** and ownership of the **AD** bus does not change. This sequence can only occur when the PCI master is the corelogic. Otherwise, a turn-around cycle would be required between the data transfer and the PCI request because ownership of signals changes. The PCI master is allowed to initiate the transaction on clock 8 since an internal **GNT#** is asserted on clock 7 (which is indicated in Figure 3-95 with a dotted line) and **FRAME#** and **IRDY#** are deasserted on clock 7. Notice on clock 9 that **GNT#** is still asserted. In this case, the master could do a fast back to back transaction. The arbiter can cause this not to occur by deasserting **GNT#** when **FRAME#** is sampled asserted. Note: if the arbiter uses a latched version of **FRAME#**, it may lose its ability to prevent a fast back-to-back transaction from occurring when the first transaction is terminated with Retry or Disconnect. The arbiter is required to remember that a previous transaction was terminated with the assertion of **STOP#**, and the bus must be given to a different agent, otherwise, a livelock condition can occur.

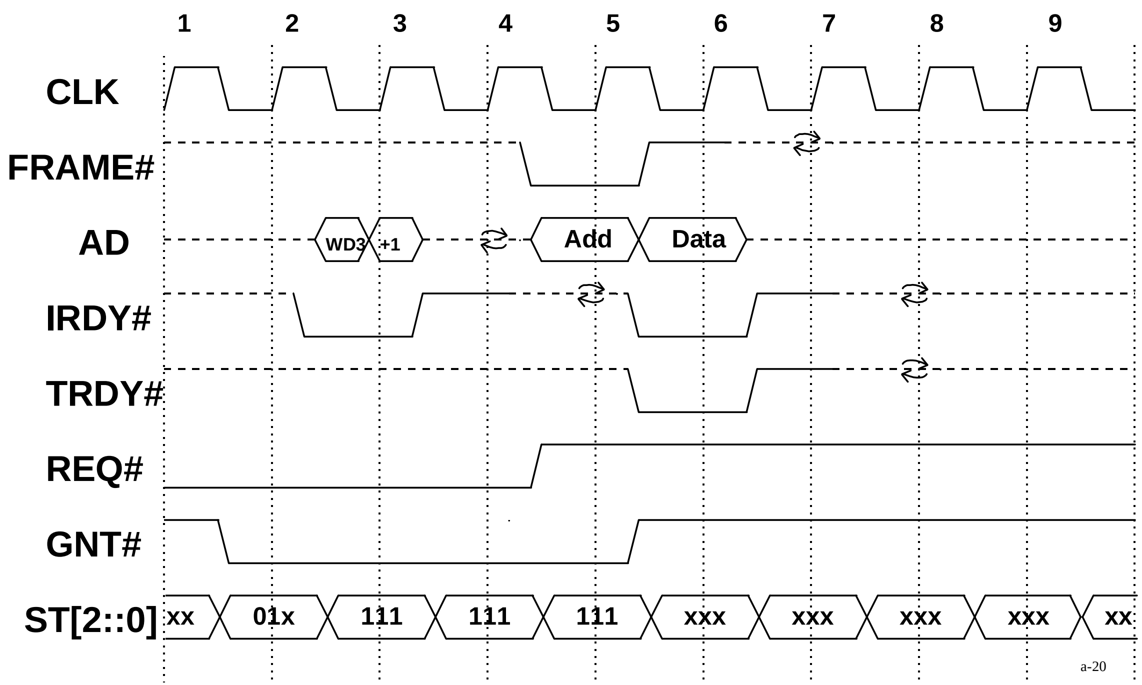


Figure 3-96: Single Write Data Followed by A.G.P. Master's PCI Transaction

Figure 3-96 is an example where a turn-around cycle is required between an A.G.P. and PCI transaction. In this case, ownership of the **AD** bus does not change and the A.G.P. master is initiating a PCI transaction after providing data to the corelogic. The arbiter is allowed to enqueue a new **GNT#** as soon as **GNT#** is asserted for the Write transaction. However, the PCI master is not allowed to start the transaction until the A.G.P. data transfer completes and then causes a turn-around cycle to occur. A dead clock can be inserted by the master on clock 5 if the master desires. However, if the arbiter removes **GNT#** on clock 5, then the master has lost its turn using the bus. The fact that **GNT#** was asserted for several clocks before is meaningless. **GNT#** only has meaning when **ST** = 111 and the master is allowed to start. This condition is when the master is ready to start, the bus is in the correct state, and **GNT#** is asserted (**ST** = 111 for an A.G.P. master doing a PCI transaction).

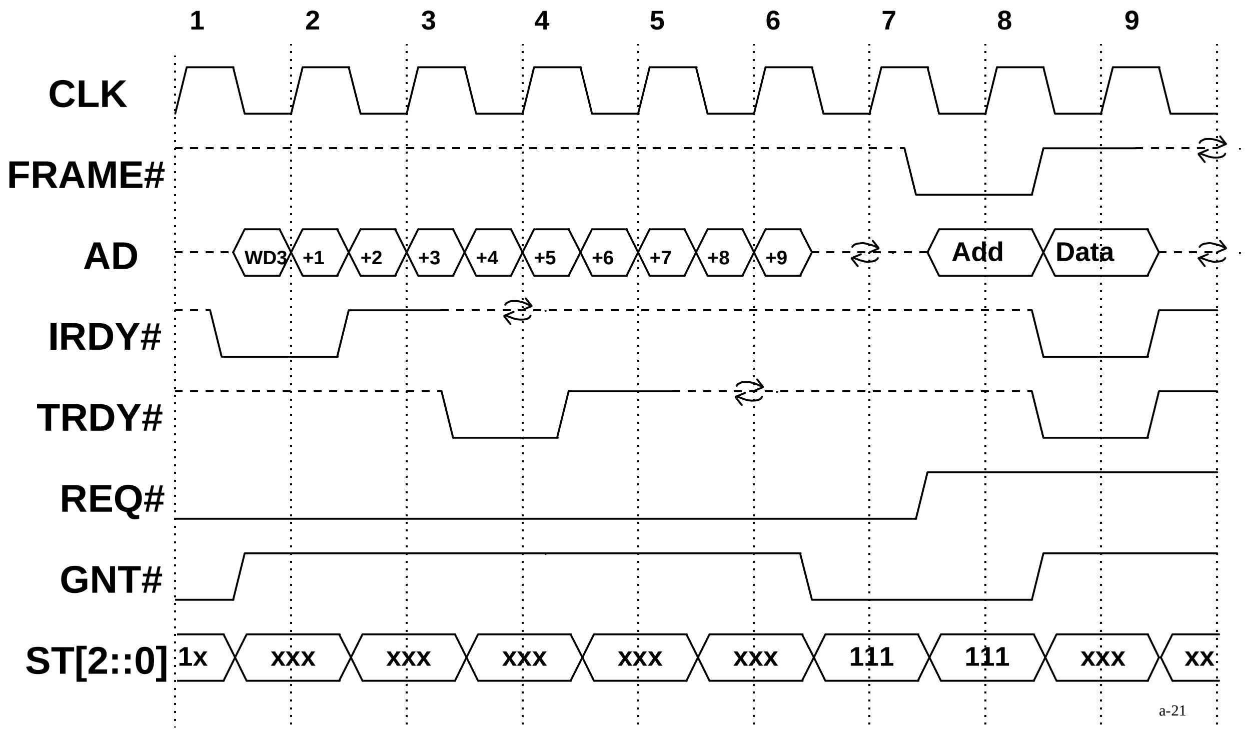


Figure 3-97: Burst Write Data Followed by an A.G.P. Master's PCI Transaction

Figure 3-97 is the same as Figure 3-96 except that the write data transfer is longer. In this figure, the arbiter delays the assertion of **GNT#** (**ST** = 111) until clock 7. The arbiter could assert **GNT#** earlier but no improvement in performance occurs. If **GNT#** was asserted earlier, the master is not allowed to start the transaction even though **FRAME#** and **IRDY#** are both deasserted. In this case, the agent initiating the PCI request is the same agent that is providing the write data. Therefore, the PCI master interface knows when the bus is free to start the PCI transaction.

3.6.6.5 A Data Transfer Followed by a Data Transfer.

The arbiter is allowed to pipeline data transfers to maximize the throughput of the A.G.P. interface. The following figures will illustrate how the arbiter can schedule the movement of both read and write data. In some cases, turn-around cycles are required and in some cases they are not.

An A.G.P. Read transaction followed by an A.G.P. Read transaction.

(See Figure 3-98 through Figure 3-101.)

An A.G.P. Write transaction followed by an A.G.P. Write transaction.

(See Figure 3-102 through Figure 3-104.)

An A.G.P. Read transaction followed by an A.G.P. Write transaction. (See Figure 3-105.)

An A.G.P. Write transaction followed by an A.G.P. Read transaction. (See Figure 3-106.)

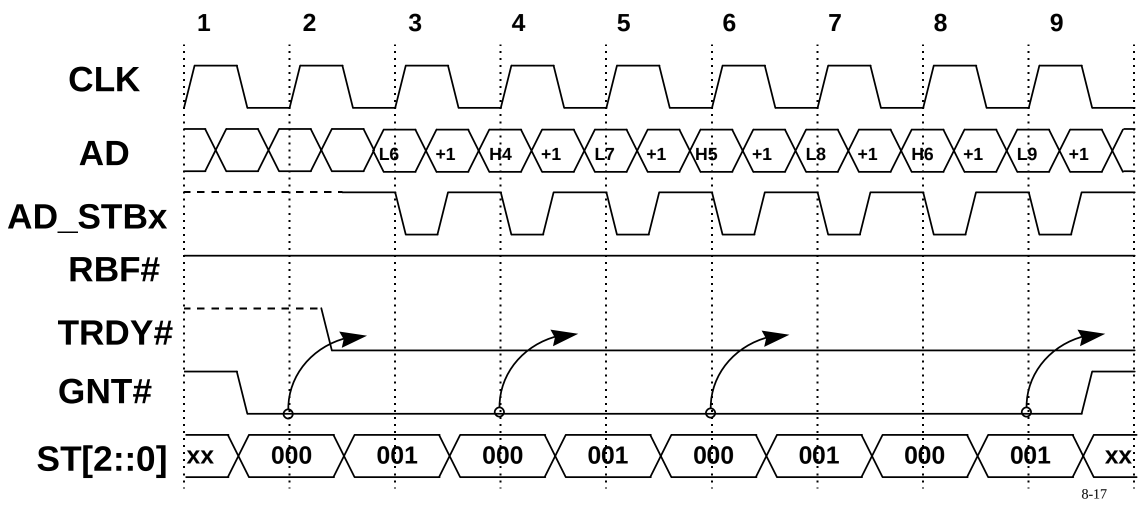


Figure 3-98: Back-to-Back Read Data

Figure 3-98 shows a sequence of back-to-back 8 byte read data transactions in 2x data transfer mode. The target samples **GNT#** asserted on clock 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 3 since that is the last data phase of the L6 read data transaction. **GNT#** is asserted on every clock so that an 8 byte read data transaction can occur on every clock.

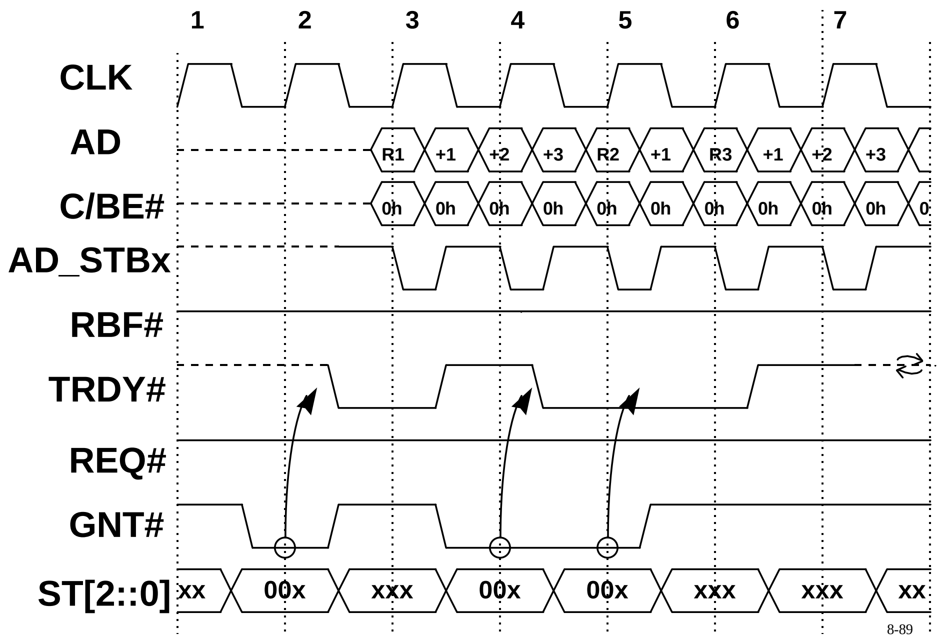


Figure 3-99: GNT# Assertion for 16, 8, and then 16 Byte Read Transfers

Figure 3-99 shows a sequence of 2x read data transactions. **GNT#** for the second read transaction (R2) is asserted on the clock 4 which is the last data phase of the R1 read transaction. **GNT#** for the third read transaction (R3) is asserted on clock 5 which is the last data phase of the R2 read transaction.

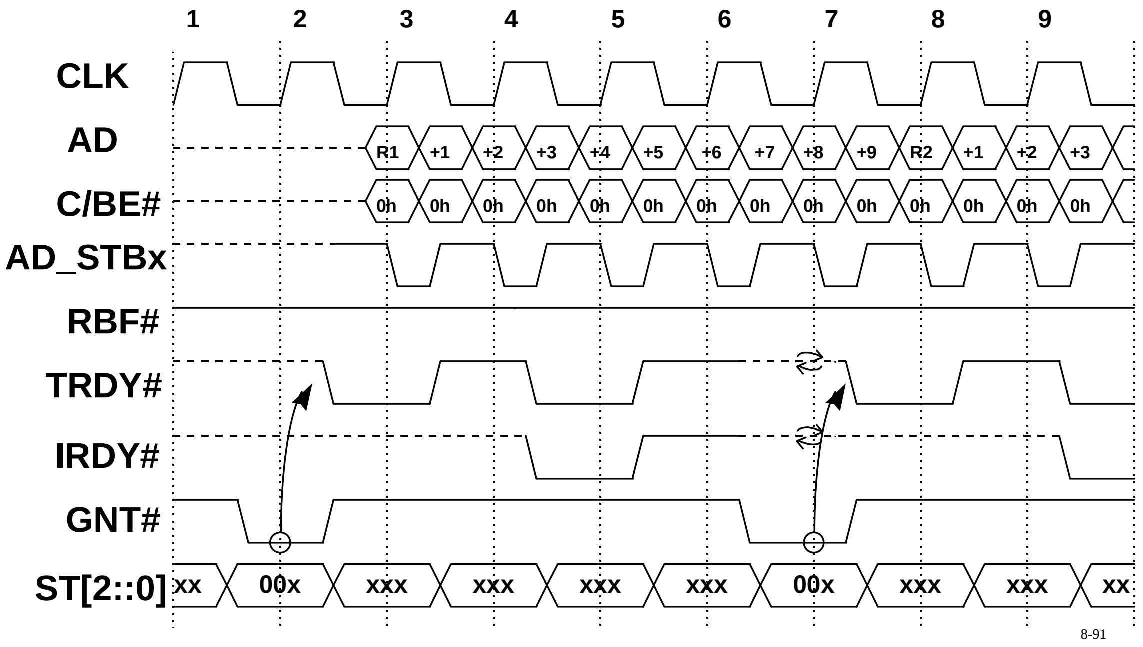

1
2
3
4
5
6
7
8
9
CLK
AD
R1 +1 +2 +3 +4 +5 +6 +7 +8 +9 R2 +1 +2 +3
C/BE#
0h 0h 0h 0h 0h 0h 0h 0h 0h 0h 0h 0h 0h 0h
AD_STBx
RBF#
TRDY#
IRDY#
GNT#
ST[2::0]
xx 00x xxx xxx xxx xxx 00x xxx xxx xx
8-91

Figure 3-100: GNT# Assertion for Next Read Data After Long Data Transfer

Figure 3-100 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 7 which is the last data phase of the R1 read transaction.

GNT# Interaction with RBF#

The A.G.P. 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 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.

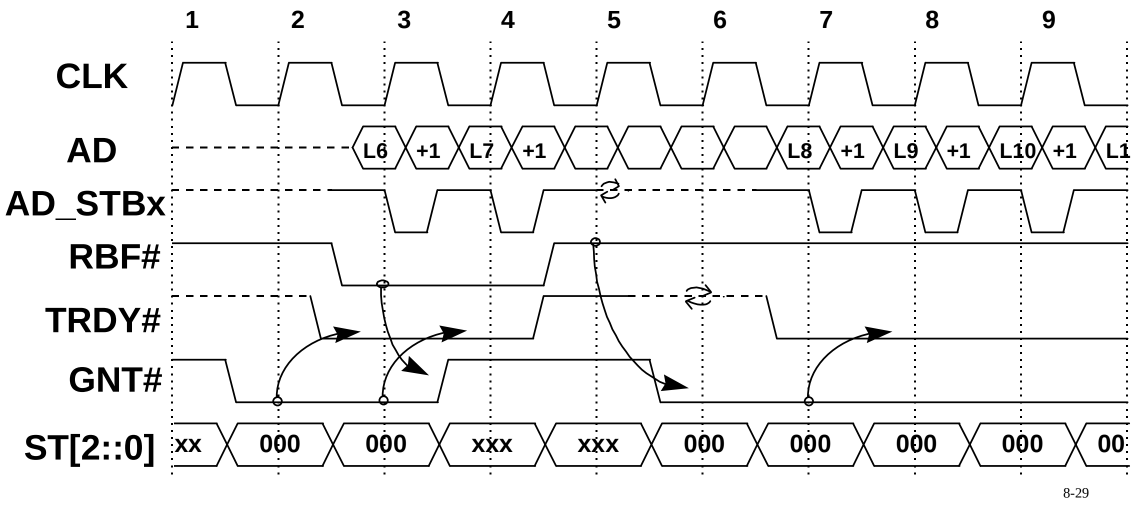

1
2
3
4
5
6
7
8
9
CLK
AD
L6 +1 L7 +1 L8 +1 L9 +1 L10 +1 L1
AD_STBx
RBF#
TRDY#
GNT#
ST[2::0]
xx 000 000 xxx xxx 000 000 000 000 00
8-29

Figure 3-101: LP GNT# Pipelining Stopped While RBF# is Asserted

Figure 3-101 shows the master asserting **RBF#** indicating that it cannot accept further low priority read data. The master samples **GNT#** asserted on clock 2 with **ST[2::0]** indicating a low priority read data transaction. The master asserts **RBF#** on clock 3 because it does not have sufficient buffer space to take the next low priority read transaction. The arbiter has already asserted **GNT#** on clock 3 which is the last data phase of L6. The master must accept the **GNT#** on clock 3 for read data transaction L7. The arbiter samples **RBF#** asserted on clock 3 and deasserts **GNT#** until it samples **RBF#** deasserted on clock 5. Note that if the arbiter did not deassert **GNT#** immediately upon sampling **RBF#** asserted on clock 3, then **GNT#** would be asserted on clock 4. This would increase the minimum amount of low priority read data buffering required in the master.

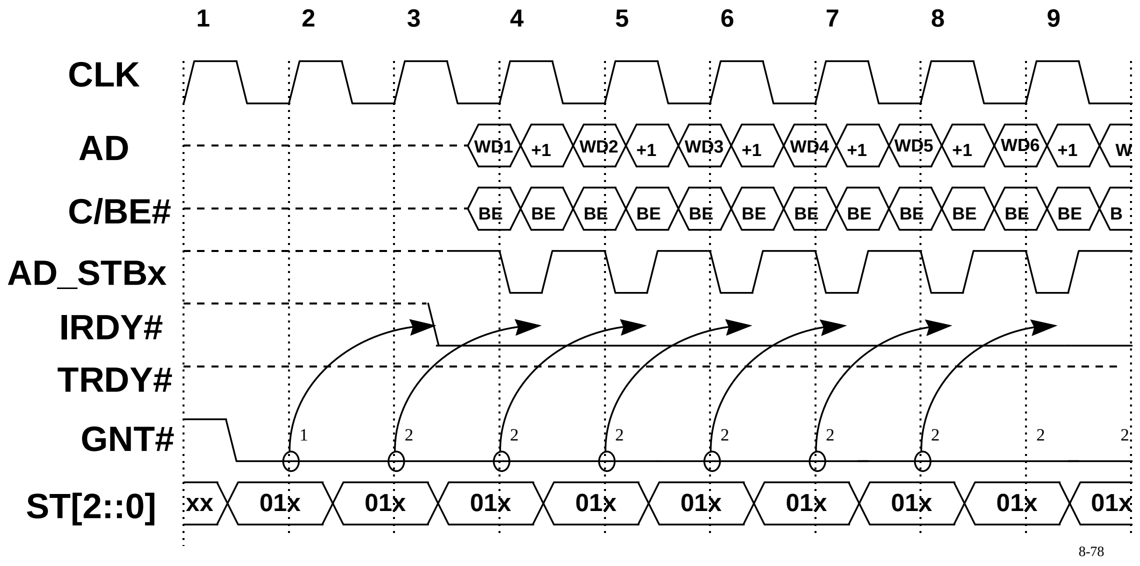


Figure 3-102: GNT# Assertion for Back-to-Back Write Data Transfers

Figure 3-102 shows back-to-back 8 byte write data transactions in 2x data transfer mode. The following figures show that a maximum of three transactions are outstanding and will transfer data. The reason that it is only three and not four is that these diagrams assume that the arbiter is not using the latched version of **IRDY#**. When the latched version is used, all the counts of grants outstanding (beyond 3) are increased by one, since the arbiter delays the decrement. However, the arbiter can actually have four outstanding transactions; otherwise, dead clocks could occur on the bus. Note: the arbiter can use a latched version of the control signals but this may require additional dead clocks. However, the A.G.P. target is not allowed to use a latched version of the control signals.

The master samples **GNT#** asserted on clock 2 and asserts **IRDY#** and drives write data W1 two clocks after sampling (clock 4). On clock 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 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 4. Even though **GNT#** is asserted on clock 4, the write **GNT#** counter does not increment since **IRDY#** for W1 is sampled asserted on clock 4. The arbiter continues asserting **GNT#** on every clock, 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.

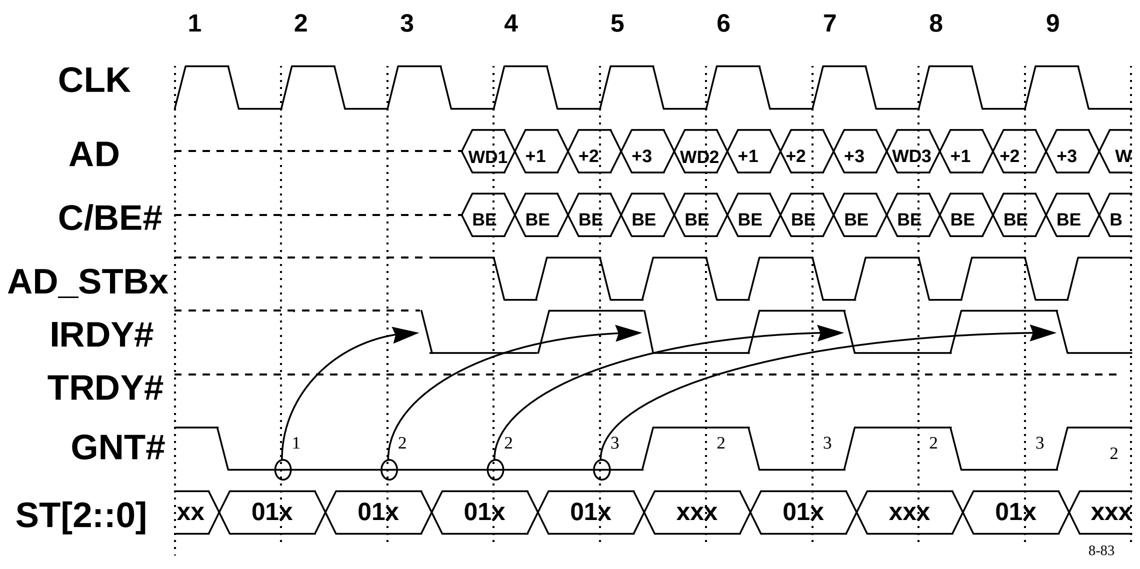


Figure 3-103: Back-to-Back GNT# with Delay on Initial Transfer

Figure 3-103 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 2. On clock 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 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 4. Even though **GNT#** is asserted on clock 4, the write **GNT#** counter does not increment since **IRDY#** for W1 is sampled asserted on clock 4. Since the write **GNT#** counter is still less than three, the arbiter asserts **GNT#** for write data W4 on clock 5. Since there is no **IRDY#** asserted on clock 5, the write **GNT#** counter increments to three and the arbiter is prohibited from asserting **GNT#** for W5 on clock 6. **IRDY#** for W2 is asserted on clock 6 decrementing the write **GNT#** counter to two. This allows the arbiter to assert **GNT#** for W5 on clock 7. This again increments the write **GNT#** counter to three and prohibits **GNT#** assertion for W6 on clock 8. Note that on clock 5, four **GNT#**s 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.

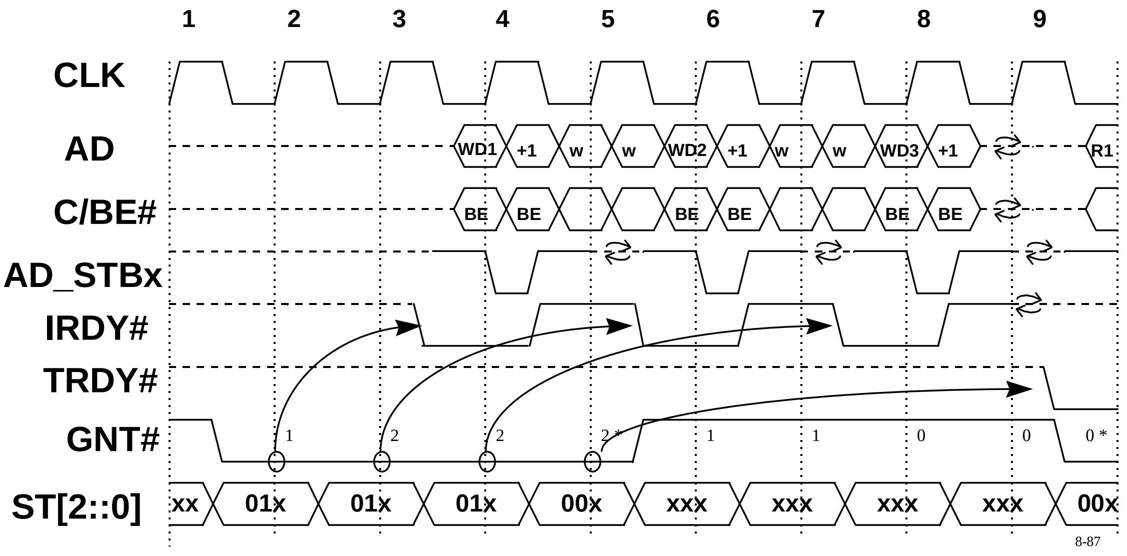


Figure 3-104: Pipelined GNT#s - Read and Writes (Part 1)

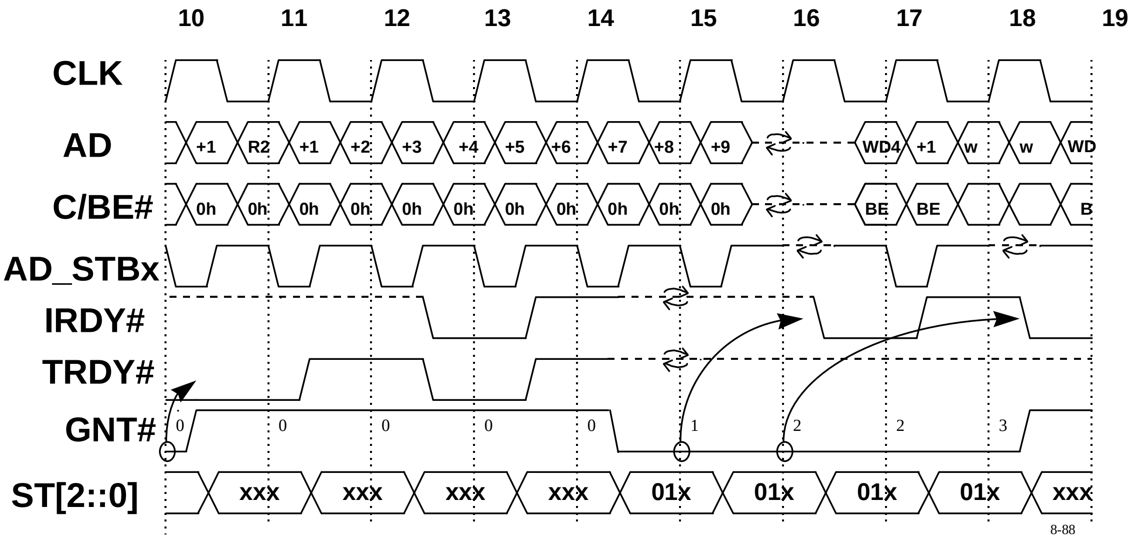


Figure 3-105: Pipelined GNT#s - Read and Writes (Part 2)

Figure 3-104 and Figure 3-105 show back-to-back write data followed by back-to-back read data. These figures should be viewed as a single figure for the following discussion. The first three **GNT#**s are for write data transactions. The master inserts a waitstate between the write data transactions. The **GNT#** asserted on clock 5 is for read data transaction R1. Note that the **GNT#** for R1 on clock 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 deasserts **GNT#** on clock 6 and waits to assert **GNT#** for read data R2 on clock 10 which is the last data phase of read data transaction R1. Note that by this time, the write **GNT#** counter has decremented to zero by sampling **IRDY#** asserted on clock 6 and 8. Note, also, that the write **GNT#** counter does not increment on clock 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 15 which is the last data phase of R2. The **GNT#** on clock 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 turn around between transactions R2 and W4. The arbiter asserts **GNT#** for W5, W6, and W7 on clock 16, 17, and 18, respectively. The arbiter is prohibited from asserting **GNT#** on clock 19 for another transaction since the write **GNT#** counter is at three.

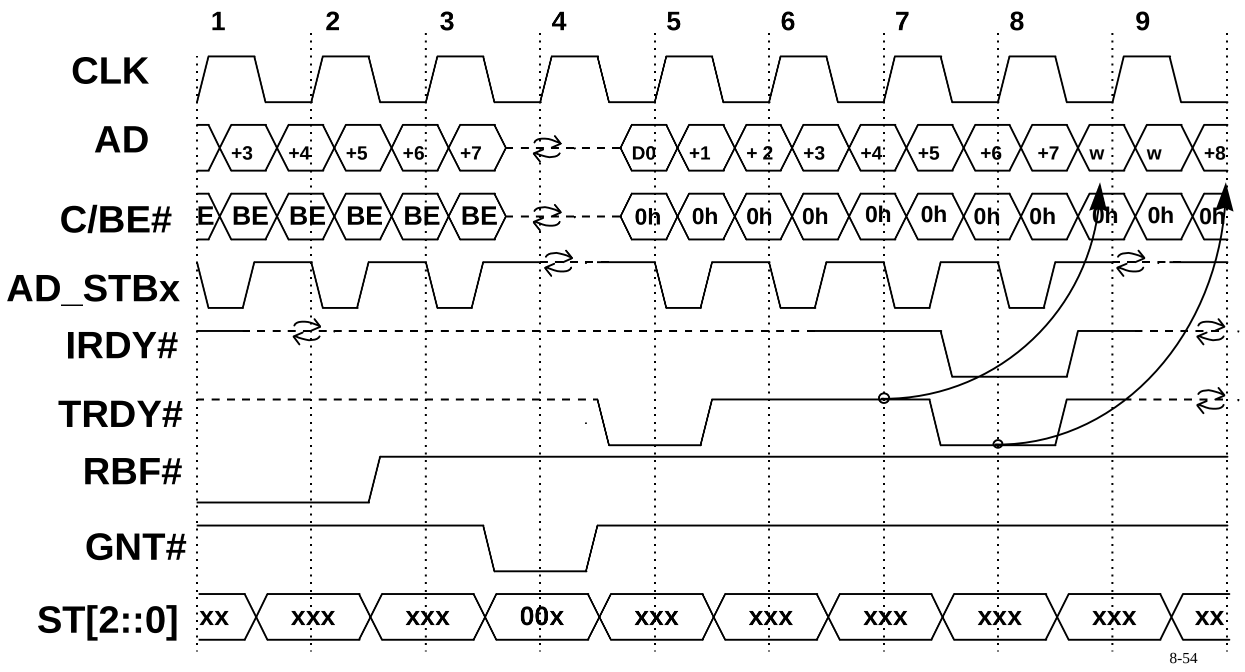


Figure 3-106: A.G.P.Write Followed by a Read

Figure 3-106 is an example of an A.G.P. write transaction followed by an A.G.P. read transaction. Since ownership of the **AD** bus is required, then a turn-around cycle is inserted between the end of the first transaction and the start of the second. But since ownership of the control signals do not change, no turn around cycle is required.

Summary of Arbitration Rules

1. The A.G.P. master that has its **GNT#** asserted with **ST[2::0]** = 111 must not remember that **GNT#** was asserted. It must use the current version of **GNT#** to decide if it can initiate a transaction or not.
2. The A.G.P. master when initiating a PCI transaction must follow the *PCI Local Bus Specification*. This requires the master to assert **FRAME#** from the same clock in which **GNT#** is sampled asserted and **ST[2:0]** = 111. The A.G.P. master initiating a PCI transaction is not allowed to start one or two clocks later as it is when initiating an A.G.P. Request.
3. The A.G.P. master is required to deassert **REQ#** when it initiates the last transaction. (The assertion of **REQ#** indicates a true need to gain access to the **AD** bus.) The A.G.P. master is required to deassert **REQ#** for two clocks when the transaction is terminated with **STOP#** asserted. An exception is granted when **STOP#** is first asserted during the last data phase.
4. The A.G.P. master, when initiating an A.G.P. Request, is required to initiate the request within two clocks of when **GNT#** is asserted, **ST[2::0]** is 111, and the bus is idle.

3.7 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 this interface specification for detection or reporting of any bus errors.

3.8 Special Design Considerations

Potential Deadlock With Misaligned Read Access to an A.G.P. Master

When data generated by the A.G.P. master is being written into main memory, a potential deadlock can occur when the A.G.P. master requires the data to be flushed before completing a read initiated by the corelogic (PCI transaction). The read must be a misaligned memory access that straddles an odd DWORD boundary.

The deadlock occurs when the CPU initiates a read reference that the corelogic must split into two accesses on the A.G.P. interface. After the initial read completes, but before the second access is attempted, the A.G.P. master posts write data into the interface. When the corelogic attempts the second part of the read, the A.G.P. master terminates with Retry forcing the access to stall in the corelogic. Since the first half of the access has completed, the corelogic is not allowed to discard the data since a side affect may occur. The corelogic may not allow the write to occur until the read can complete. This is caused by the condition where the write may be required to be coherent with main memory and the CPUs cache. Since the processor bus is stalled pending the read request, the sequence ends in a livelock or deadlock condition. In either case, no agent can make forward progress and neither can be backed up to allow the other to progress.

There are two proposed solutions that can be implemented in the PCI target interface of an A.G.P. master to prevent this from occurring.

1. The A.G.P. master can require its driver[28] to never access its internal register with a read operation that is misaligned when it has posted write data.
2. Not require that posted write data be flushed before completing a PCI read transaction to the PCI target interface of an A.G.P. master.

Option 1 is a reasonable choice if the device never writes data to main memory or posts data internally. Posting write data internally occurs when one state machine believes the access has completed at the final destination and changes status or causes a new process to occur that depends on the write being at the final destination. This sequence assumes that when a read of the Status register occurs that the data will be pushed or pulled to the final destination.

Option 2 is reasonable when the read of Status register has no effect on the posted write operation. In this case, the Status register does not get updated until the write data leaves the A.G.P. master.

[28] This may not be possible when support of legacy software is required.

4. Electrical Specification

4.1 Overview

4.1.1 Introduction

As with the protocol enhancements, most of the A.G.P. electrical interface requirements are similar to the PCI specification, however, with extensions to allow the higher transfer rate.

The A.G.P. physical interface is optimized for a point-to-point topology using either 1.5 volt or 3.3 volt signaling. The baseline performance level utilizes a 66 MHz clock[29] to provide a peak bandwidth of 266 MB/s.

A.G.P. includes two options for higher performance levels. The first option provides peak bandwidths of 533 MB/s. This 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*), 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 second mode provides an option for even higher performance levels providing peak bandwidths of up to 1066 MB/s. This mode uses a quad-clocked data transfer technique to transfer four times the data per each 66 MHz clock. This mode, referred to as *4X transfer mode*, requires differential interface timing strobes and different signal timings from the 66 MHz baseline and 2X mode A.G.P. requirements. The 4X clock mode requirements are a superset of the baseline A.G.P. 66 MHz timings and the 2X clock mode.

The section establishes a conceptual framework by which to understand these transfer modes.

4.1.2 Transfer Mode Operation

4.1.2.1 Transfer Mode Signaling Levels

The A.G.P interface allows signaling at either 3.3 volts or 1.5 volts. The signaling level is determined through the value of Vddq I/O interface voltage. A.G.P. add-in cards operate at one of the two signaling levels and can only be plugged into a motherboard capable of supporting that particular signaling level. The signaling level of the motherboard is indicated by the position of the key in the A.G.P connector. In the case of the universal connector where no key exists, the **TYPEDET#** pin on the add-in card indicates to the platform the signaling level for that card. Only core logic devices that support the universal connector need to have interfaces capable of operating at

[29] The A.G.P. base frequency is actually $66^2/_3$ MHz (15.0 ns period) and is the basis for the 2X and 4X transfer rates and bandwidths described in this section.

either value of Vddq. All other core logic and all graphics devices operate at just one value of Vddq. Note: The interface signaling level is determined by the level of Vddq independent of transfer rate or whether the interface protocol is PCI or A.G.P. Section 4.3.4 gives more information on all pins in the A.G.P. interface. See Chapter 5 for more information on connectors.

Table 4-1 shows the allowed signaling levels for each data transfer mode. The 1X and 2X modes can operate at either signaling level. The 4X mode is restricted to the 1.5 volt signaling level because of signal integrity limitations. As mentioned above, signaling level is determined by the connector on the motherboard or by the **TYPEDET#** signal on universal connectors. The signaling rate is determined by software negotiation based on the capabilities of the A.G.P. master and target.

Table 4-1: Allowed Transfer Mode Signaling Levels

Signaling Level	1X Mode	2X Mode	4X Mode
3.3 volt	✔	✔	no
1.5 volt	✔	✔	✔

4.1.2.2 1X Transfer Mode Operation

The 1X mode, 66 MHz A.G.P. interface can be designed using standard CMOS 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.

4.1.2.3 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[30]. With 2X transfer, Qword transfers only require one clock cycle, and sideband commands only require one clock per 16-bit command.

4.1.2.4 4X Transfer Mode Operation

With the 4X mode, the A.G.P. protocol provides double Qword access granularity and can transfer a double Qword and two 16-bit commands on the sideband address lines in one 66 MHz clock cycle. The 4X transfer mode provides a mechanism for quadrupling the data transfer rate of the **AD**, **C/BE#**, and **SBA** signals[30] relative to the 1X mode.

4.1.2.5 2X/4X Timing Model

The 2X and 4X transfer modes are 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

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

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.

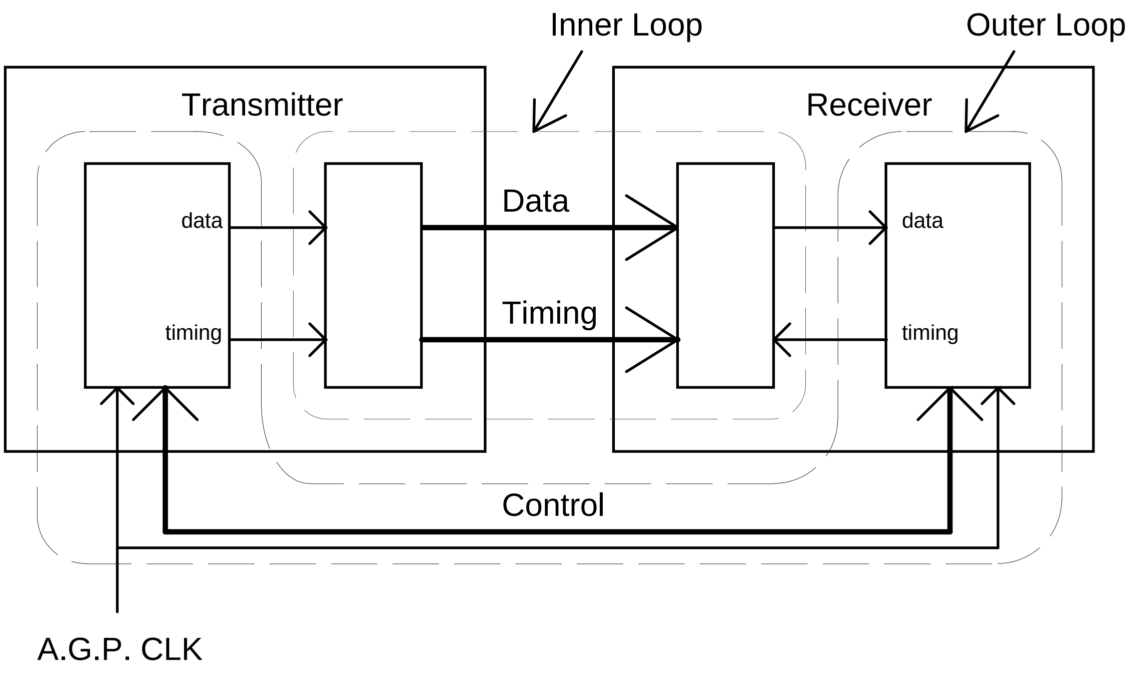


Figure 4-1: 2X/4X Mode Time Domains

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 compensated by the strobe delays 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.

In 2X mode, two single-ended strobes **AD_STB[1::0]** are used to capture the address-data lines and another strobe **SB_STB** is used for the sideband signals. In 4X mode, differential strobe pairs **AD_STB[1::0]**, **AD_STB[1::0]#** and **SB_STB**, **SB_STB#** are used to capture data. In the following descriptions, the 4X transfers will reference the **AD_STB[1::0]** signals to show the common timings between 4X mode and 2X mode. However, 2X timings are referenced at a particular level on the rising or falling strobe edge while 4X timings are referenced to the crossover point of the differential strobes. The crossover is targeted to be at 0.5Vddq.

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

4.1.2.6 Transmit/Receive Outer Loop

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

4.1.2.7 Transmit to Receive Inner loop

Transfer of data[31] between the transmit and receive inner loop circuits is accomplished using timing strobes sent from the transmitter to the receiver, with a set of simple timing relationships between the data and strobes. Both edges of the strobes 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.

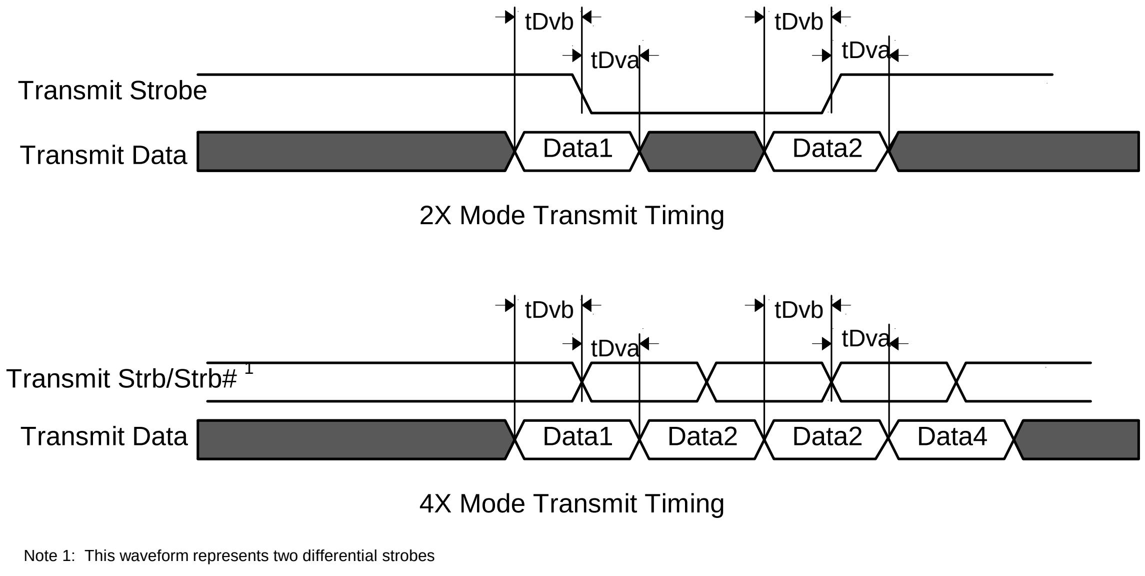


Figure 4-2: Mode Transmit Strobe/Data Timings

[31] Data refers to any of the 2X/4X capable signal groups: **AD[31:00]**, **C/BE[3:0]#**, or **SBA[7:0]**.

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.

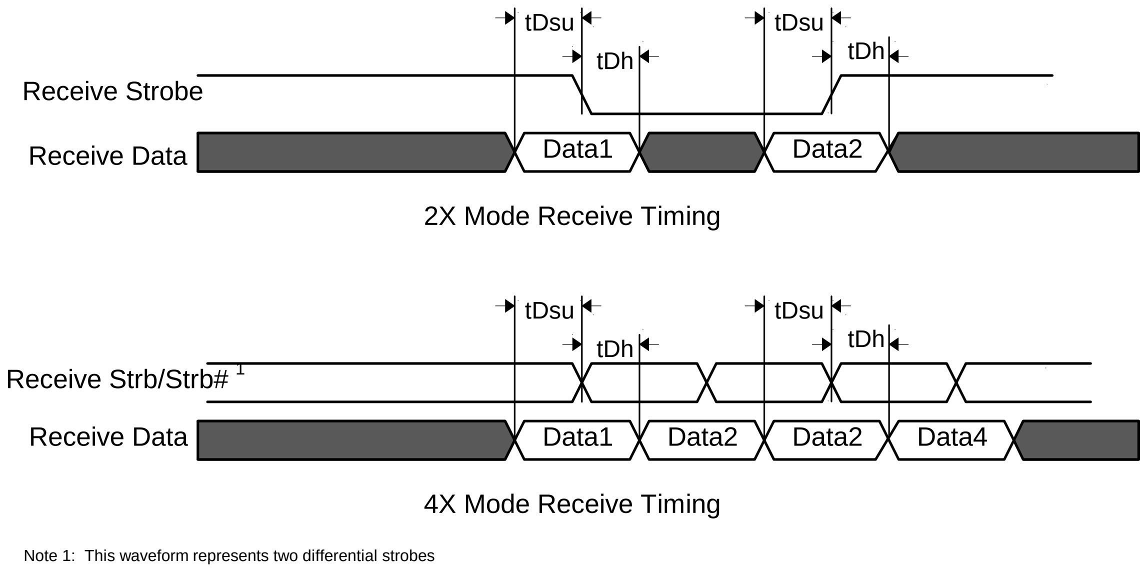


Figure 4-3: Receive Strobe/Data Timings

4.1.2.8 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 specifications 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.

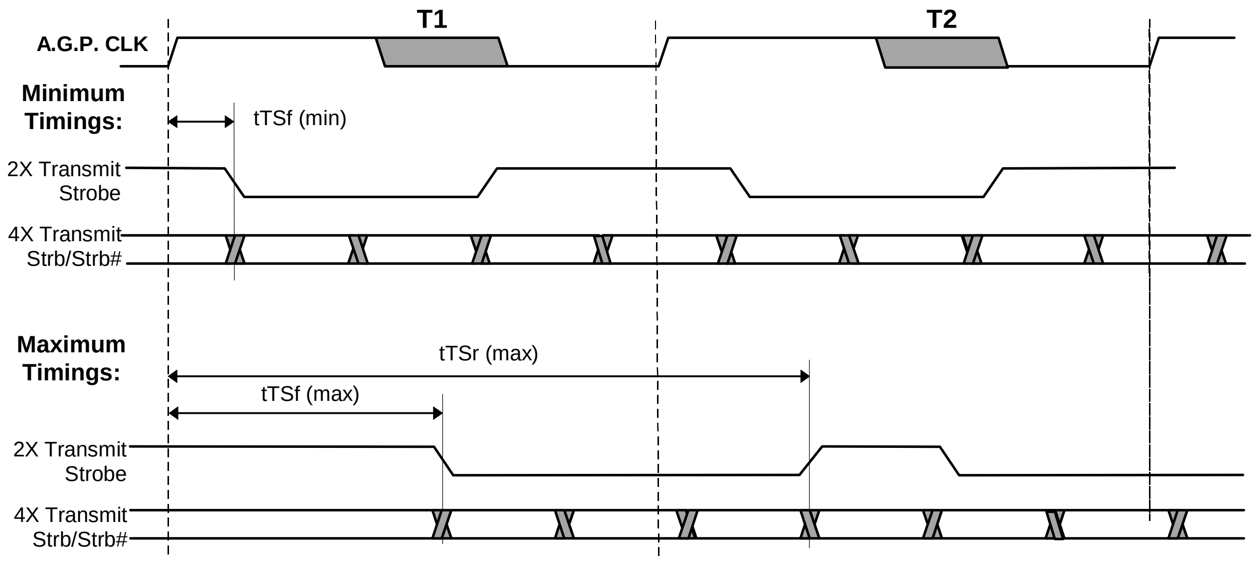


Figure 4-4: Transmit Strobe/Clock Timings

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

Figure 4-5 and Figure 4-6 show the composite inner and outer loop timing relationships for the transmitter in 2X and 4X transfer modes respectively.

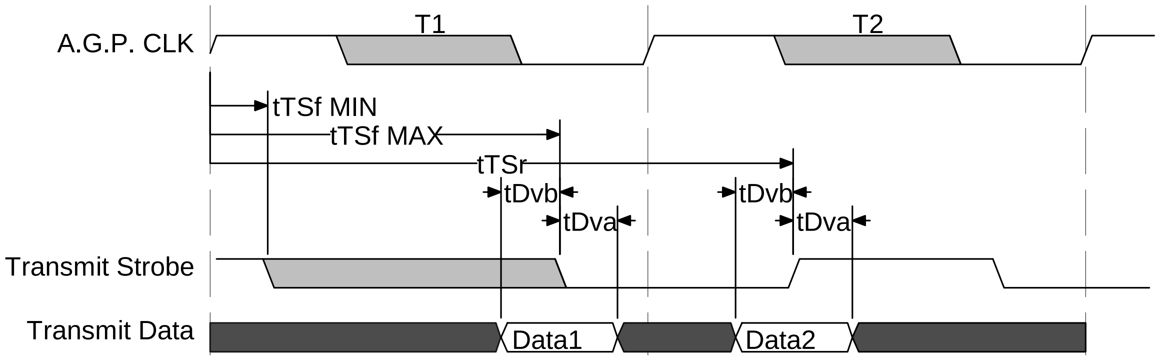


Figure 4-5: Composite 2X Mode Transmit Timings

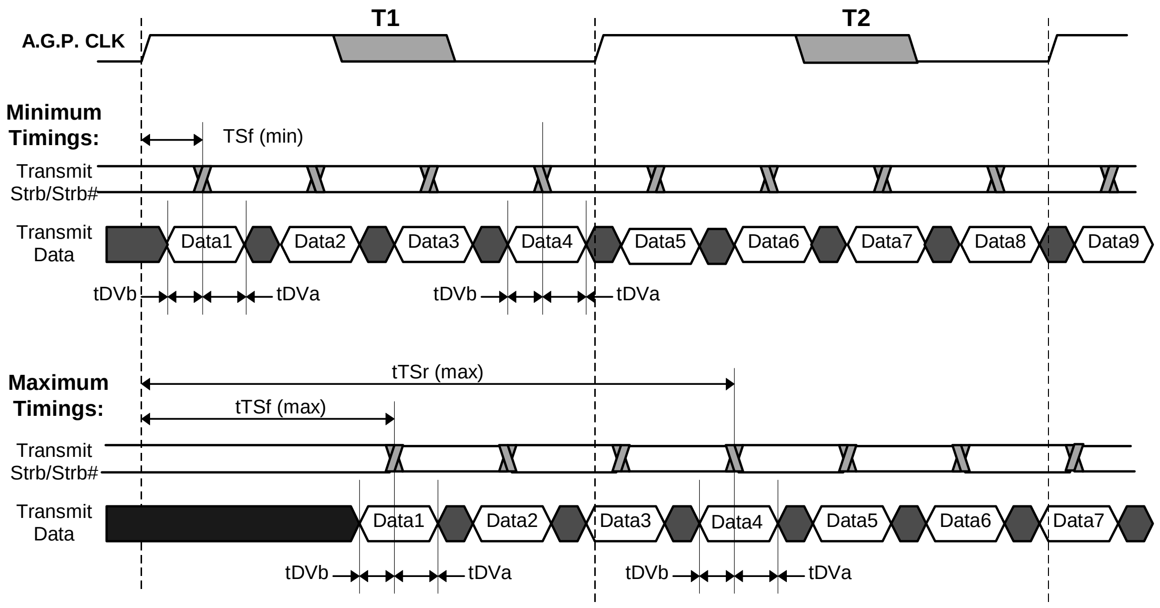


Figure 4-6: Composite 4X Mode Transmit Timings

4.1.2.9 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 4-7 and Figure 4-8.

For 2X transfer mode, after the rising edge of the receive strobe, a Qword of valid data will be available from the **AD** bus interface. Data will be transferred from the inner loop to the outer loop based on an A.G.P. clock event. The requirement is to define a circuit to reliably affect this transfer for all system conditions.

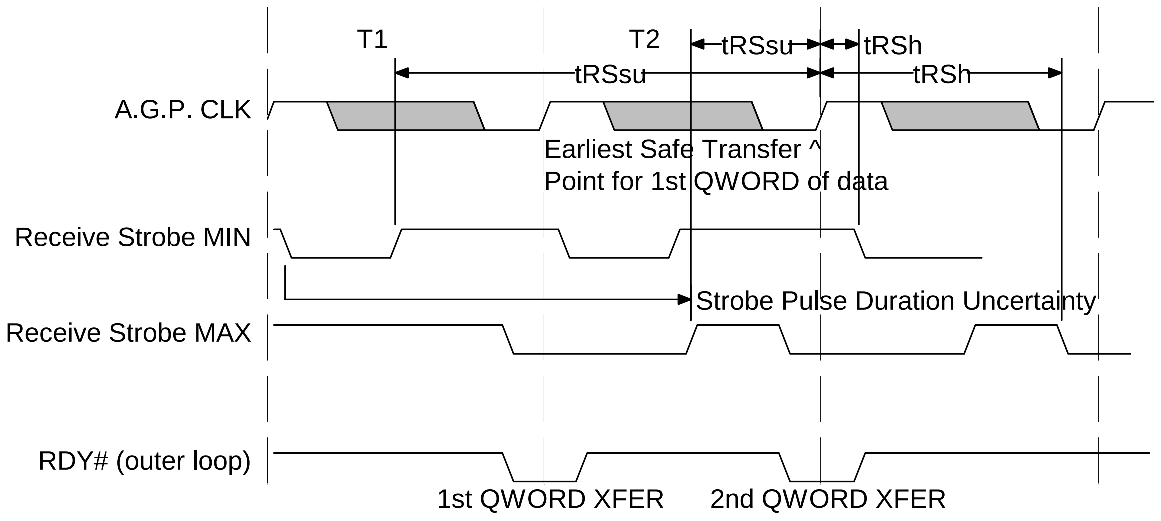


Figure 4-7: 2X Mode Receiver Inner to Outer Loop Transfer Timing

Figure 4-7 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.

The same problem exists in 4X mode and is shown in Figure 4-8. After the fourth strobe pair transition, a double Qword is available, but because of the uncertainty of the strobe timing, this can occur in T1 or T2.

Due to this uncertainty window, the earliest *safe* transfer point from the inner loop to an A.G.P. clock edge in the outer loop occurs at the end of T2. Notice that in the minimum case, a second set of strobes occurs in T2, since a second Qword (2X) or double Qword (4X) of data is being transferred. Therefore, to prevent 2X transfer mode data from being overwritten before the safe transfer point, at least *two* stages of *Qword-wide* edge-triggered latches must exist in the inner loop input circuitry. For 4X mode, at least *two* stages of *double Qword-wide* edge-triggered latches are required.

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 specification 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 specification 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 *end of the previous set of strobes*, not the immediately prior one.

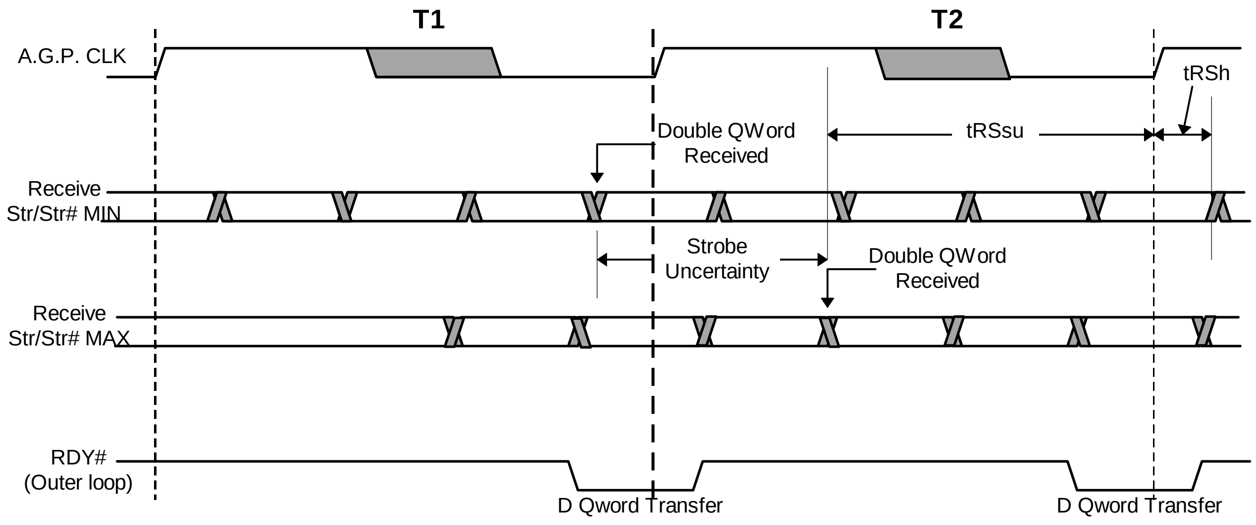


Figure 4-8: 4X Mode Receiver Inner to Outer Loop Transfer Timing

The receive strobe specification 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 4-9 and Figure 4-10.

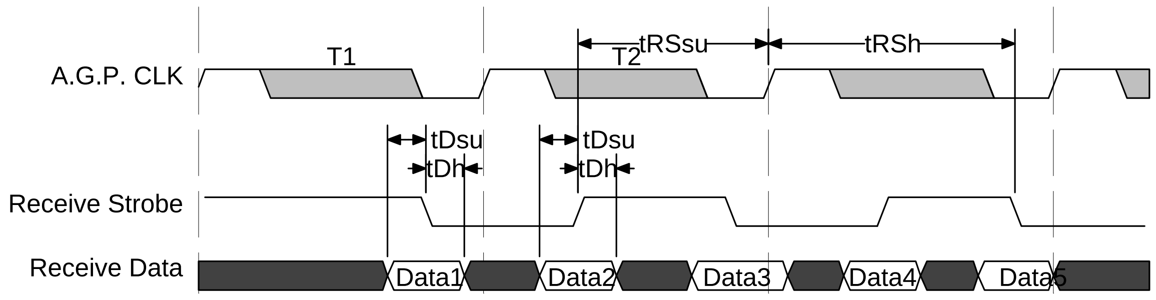


Figure 4-9: 2X Composite Receive Timings

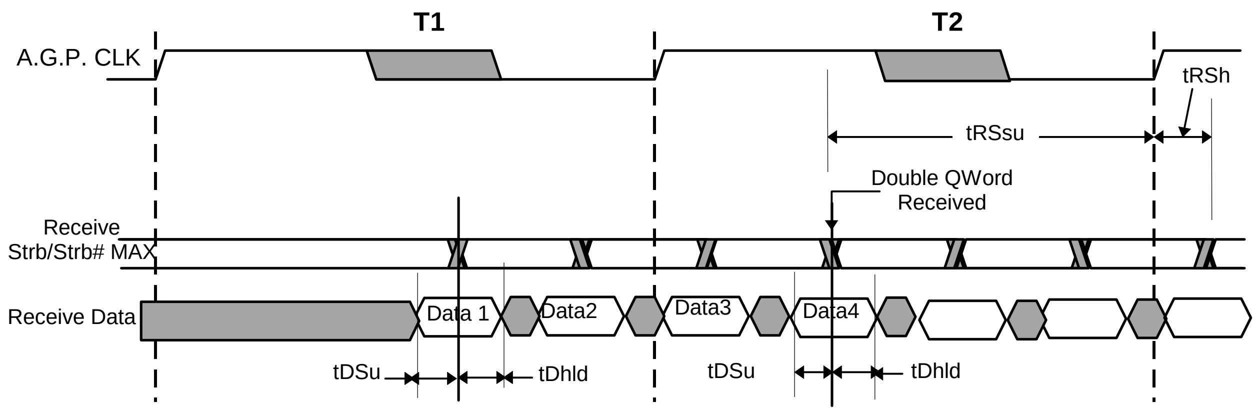


Figure 4-10: 4X Composite Receive Timings

4.1.2.10 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.[32] 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. 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 or 4X command strobe point. This first valid 2X or 4X command strobe point occurs *exactly* two clocks after the sync cycle event is sampled by the target (cycle T1 in Figure 4-11), at which point 2X or 4X timing operation has commenced. The first valid command[33] occurs on this first T1 cycle.

The 2X or 4X timing operation must continue, unless **SB_STB** is stopped. Prior to stopping the **SB_STB** signal, a minimum of *four* A.G.P. clock cycles of NOPs must be transmitted. (Note this is equivalent to four 16-bit NOP commands in 2X mode or eight NOP commands in 4X mode). If stopped, **SB_STB** must be driven high and either held high or tri-stated. Once stopped, **SB_STB** must remain stopped for a *minimum* of *eight* A.G.P. clock cycles prior to any new sync event. The **SB_STB** synchronization protocol for 2X mode is shown in Figure 4-11.

[32] The strobe to clock AC timings specifications 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.

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

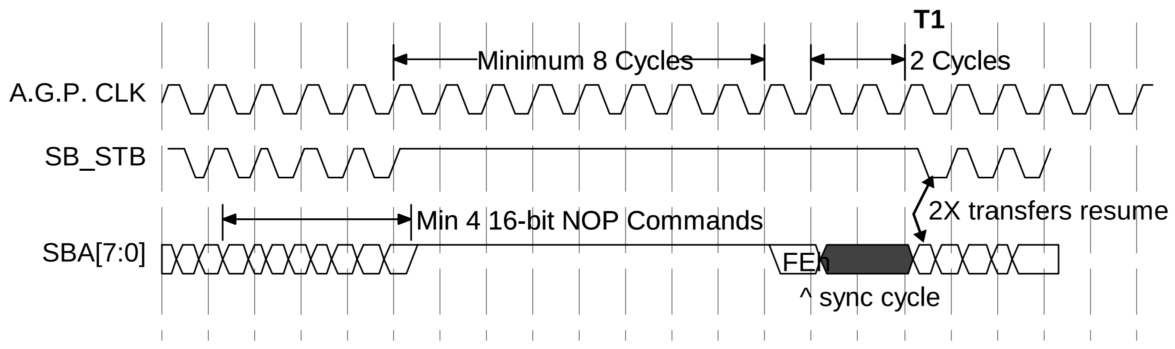


Figure 4-11: SB_STB Synchronization Protocol

4.2 Component Specification

This section 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 Local Bus Specification*.

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

4.2.1 DC Specifications

The A.G.P. interface has two voltage levels of operation. The first is optimized for a 3.3 volt operating environment and is based on PCI 3.3 volt signaling, with changes to allow higher data transfer rates. DC parameters differing from PCI 66 MHz specifications are highlighted in bold in Table 4-2. The second signaling level is 1.5 volts. It uses the same signaling conventions, scaled to 1.5 volts. The DC parameters for 1.5 volt signaling is shown in Table 4-3.

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

Table 4-2: DC Specifications for A.G.P. 1X Signaling at 3.3 Volts

Symbol	Parameter	Condition	Min	Max	Units	Notes
Vddq	**I/O Supply Voltage**		**3.15**	**3.45**	**V**	**1**
V_{ih}	Input High Voltage		0.5Vddq	Vddq+0.5	V	
V_{il}	Input Low Voltage		-0.5	0.3Vddq	V	
I_{il}	Input Leakage Current	$0 < V_{in} <$ Vddq		±10	µA	
V_{oh}	Output High Voltage	I_{out} = -500 µA	.9Vddq		V	
V_{ol}	Output Low Voltage	I_{out} = 1500 µA		.1Vddq	V	
C_{in}	**Input Pin Capacitance**			**8**	**pF**	**2**
C_{clk}	CLK Pin Capacitance		5	12	pF	
V_{ol}	Output Low on **OVRCNT#**	I_{out} = ±20 µA		0.4	V	
V_{oh}	Output High on **OVRCNT#**	I_{out} = ±20 µA	2.4	3.6	V	

Table 4-3: DC Specifications for A.G.P. 1X Signaling at 1.5 volts.

Symbol	Parameter	Condition	Min	Max	Units	Notes
Vddq	**I/O Supply Voltage**		**1.425**	**1.575**	**V**	**1**
V_{ih}	Input High Voltage		0.6Vddq	Vddq+0.5	V	
V_{il}	Input Low Voltage		-0.5	0.4Vddq	V	
I_{il}	Input Leakage Current	$0 < V_{in} <$ Vddq		±10	µA	
V_{oh}	Output High Voltage	I_{out} = -200 µA	.85Vddq		V	
V_{ol}	Output Low Voltage	I_{out} = 1000 µA		.15Vddq	V	
C_{in}	**Input Pin Capacitance**			**8**	**pF**	**2**
C_{clk}	CLK Pin Capacitance		5	12	pF	
V_{ol}	Output Low on **OVRCNT#**	I_{out} = ±20 µA		0.4	V	
V_{oh}	Output High on **OVRCNT#**	I_{out} = ±20 µA	2.4	3.6	V	

Notes:

1. Vddq only specifies the voltage at the A.G.P. interface; component supply voltages are independent of Vddq. Vddq for both A.G.P. master and A.G.P. target must be driven from the same supply line.
2. Absolute maximum pin capacitance for an A.G.P. compliant component 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 2X and 4X Mode DC Specification

The parameters in Table 4-4 and Table 4-5 are the incremental requirements for the 2X and 4X mode interface for their respective signaling levels. Note that the primary change is the addition of a voltage reference, which allows for a differential input buffer with a 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 4-4: DC Specifications for 2X Mode Only at 3.3 volts 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	µA	2
Cin	Input Pin Capacitance			8	pF	3
Δ Cin	Strobe to data Pin Capacitance delta		-1	2	pF	3,4

Table 4-5: DC Specifications for 2X or 4X Mode at 1.5 volts Signaling

Symbol	**Parameter**	**Condition**	**Min**	**Max**	**Units**	**Notes**
Vref	Input reference voltage		0.48Vddq	0.52Vddq	V	1,2
Iref	Vref pin input current	$0 < V_{in} < Vddq$		±5	µA	2
Cin	Input Pin Capacitance			8	pF	3
Δ Cin	Strobe to data Pin Capacitance delta	2X Mode 4X Mode	-1 -1	2 1	pF pF	3,4

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 for 3.3 volt signaling and 0.5Vddq for 1.5 volt signaling. Vref can be designed with 2% resistors to achieve the specified minimum and maximum values. The value of Vref is intended to specify the center point of the VIL/VIH range. For the 3.3 volt signaling case, at nominal Vddq (3.3 V), Vref is 1.32 V +/- 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. Although a differential input buffer is not a required implementation, it is recommended especially at higher data transfer rates where there is less timing margin. All designs regardless of implementation style must meet all other specifications. Component designs requiring a reference are required to adhere to the Vref and Iref specifications and 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. Capacitance specifications refer only to pin capacitance on the A.G.P. compliant components used on the A.G.P. interface.
4. 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_STB1# =>AD[31::16]** and **C/BE[3::2]#**; **AD_STB0**, **AD_STB0# =>AD[15::00]** and **C/BE[1::0]#**; **SB_STB**, **SB_STB# =>SBA[7::0]**. (Complementary strobes apply to 4X mode only.)

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 (see Table 4-6) and the second for the optional A.G.P. 2X transfer mode operation (see Table 4-7). The A.G.P. 2X specifications are in *addition* to the A.G.P. 1X specifications. The A.G.P. 1X specifications still apply to all outer loop control signals during A.G.P. 2X operation. See Figure 4-12 for the 1X timing diagram and Figure 4-13 and Figure 4-14 for the 2X timing diagram.

4.2.2.1 A.G.P. 1X Timing Parameters

Table 4-6: A.G.P. 1X AC Timing Parameters

Symbol	Parameter	Min	Max	Units	Notes
Clock:					
t_{CYC}	CLK cycle time	15	30	ns	2
t_{HIGH}	CLK high time	6		ns	
t_{LOW}	CLK low time	6		ns	
-	CLK slew rate	1.5	4	V/ns	3
t_{LOCK}	PLL Lock Time		1000	µs	4
Transmitter Output Signals:					
t_{VALC}	**CLK to control signal valid delay**	**1.0**	**5.5**	**ns**	**1,5**
t_{VALD}	**CLK to data valid delay**	**1.0**	**6.0**	**ns**	**1,5**
t_{ON}	**Float to Active Delay**	**1.0**	**6**	**ns**	**1**
t_{OFF}	**Active to Float Delay**	**1**	**14**	**ns**	
	Output slew rate	1.5	4	V/ns	
Receiver Input Signals:					
t_{SUC}	**Control signals setup time to CLK**	**6.0**		**ns**	**5**
t_{SUD}	**Data setup time to CLK**	**5.5**		**ns**	**5**
t_{H}	**Control signals hold time to CLK**	**0.0**		**ns**	**5**
Reset Signal:					
t_{RST}	Reset active time after power stable	1		ms	
$t_{RST\text{-}CLK}$	Reset active time after **CLK** stable	100		µs	
$t_{RST\text{-}OFF}$	Reset active to output float delay		40	ns	
---	**RST#** Slew Rate	50	n/a	mV/ns	6

Notes:

1. Output delays into a capacitive load of 10 pF.
2. In general, A.G.P. devices must work with any stable clock frequency from 33 MHz 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 *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 signals are floated by the component and pulled-up to Vddq by the central resource).
3. 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 4-17.
4. 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. devices must allow for the clock to stop without any loss of data or configuration information. When the clock is restarted, a bus master cannot issue a new transaction or bus request until the tLOCK time period has been met.
5. In A.G.P. 2X mode, the tVALD, tSUD, and tH values for the **AD**, **C/BE#**, and **SBA** signals are *superseded* by the A.G.P.-2X parameters tDvb, tDva, tDsu, and tDh.
6. 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.

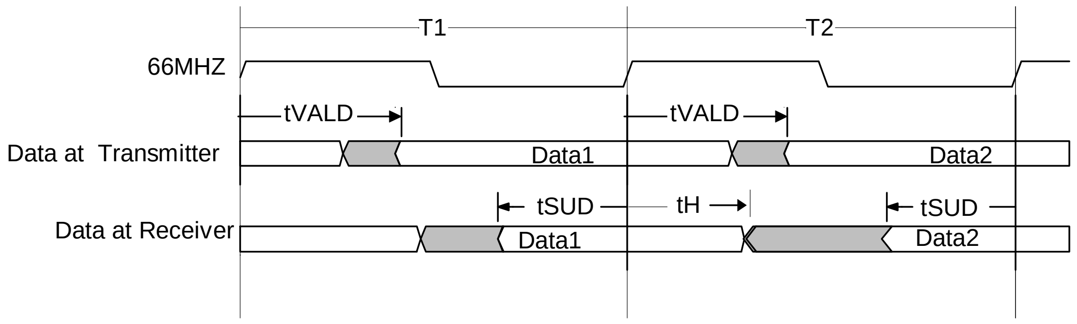


Figure 4-12: A.G.P. 1X Timing Diagram

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 specifications below *replace* the corresponding data specifications from the A.G.P. 1X table for these signals. The A.G.P. 1X parameters apply to all other signal operation.

Table 4-7: 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	1
t_{TSr}	**CLK** to transmit strobe rising		20	ns	
t_{Slow}	Strobe width low	5.0		ns	
t_{SHigh}	Strobe width high	5.0		ns	
t_{Dvb}	Data valid before strobe	1.7		ns	
t_{Dva}	Data valid after strobe	1.9		ns	
$\mathbf{t_{ONd}}$	**Float to Active Delay**	**-1**	**9**	**ns**	
$\mathbf{t_{OFFd}}$	**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	2
t_{RSh}	Receive strobe hold time hold time from **CLK**	1			2
t_{Dsu}	Data to strobe setup time	1		ns	
t_{Dh}	Strobe to data hold time	1		ns	

Notes:

1. When Write data immediately follows the enqueuing of requests (1x) (Figure 3-83), the minimum time of 2 ns causes a violation of the 1x hold time of the request. When this type of operation is supported by the master, the tTSf minimum time is required to be greater than 2 ns.
2. These specifications refer to the setup and hold times for the strobe set started in the previous cycle. See Figure 4-13 and Section 4.1.2.9 for more details.
3. All the parameters in this table only apply to A.G.P. clock of 66 MHz (15.0 ns period). Parameters like tTSf(max), tTSr(max), tONS, and tOFFS are dependent on the period of the A.G.P. clock.

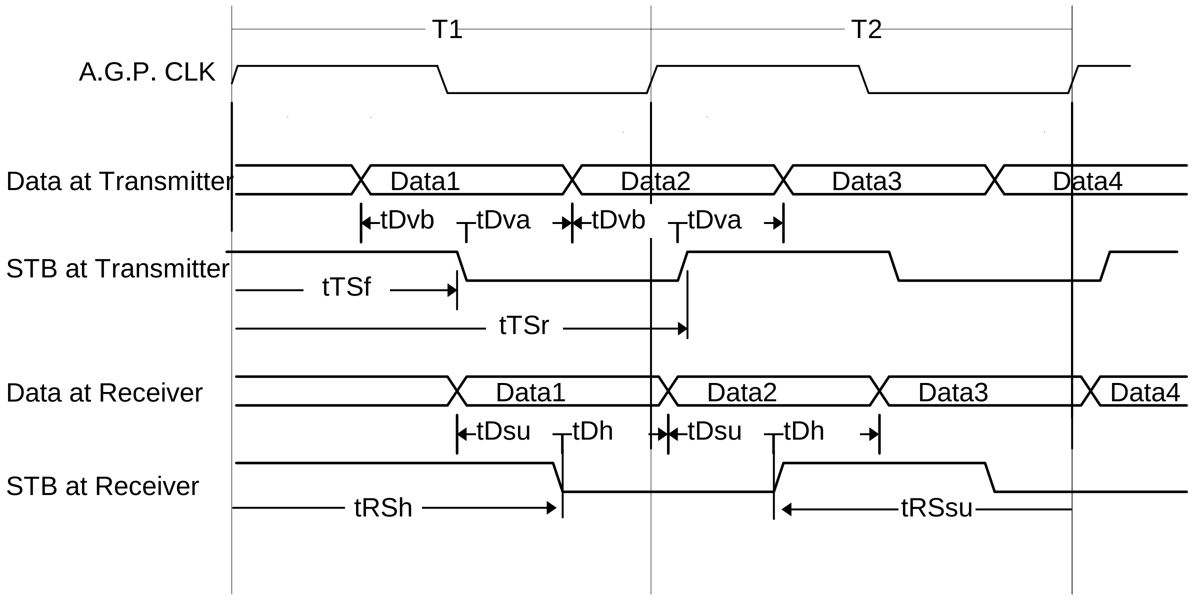


Figure 4-13: A.G.P. 133 Timing Diagram

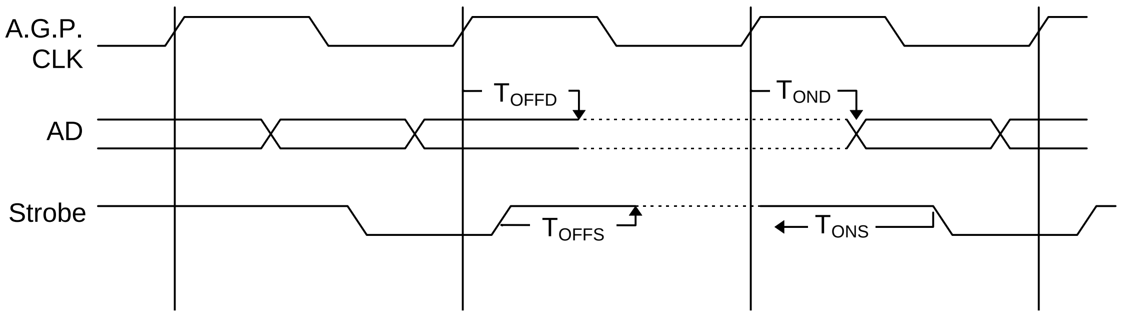


Figure 4-14: Strobe/Data Turnaround Timings

4.2.2.3 4X AC Timing Parameters

The parameters below apply only to the inner loop 4X clock mode signals **(AD**, **C/BE#**, **SBA)** during 4X operation. A pair of Strb/Strb# for each group of 16 bits data is required for 4X operation. The data specifications below *replace* the corresponding data specifications from the A.G.P. 1X mode and A.G.P. 2X mode tables for these signals. The A.G.P. 1X mode parameters apply to all other signal operation.

Note: Information in this section particularly is still under investigation and development by Intel and is not complete. Therefore, this information is quite likely to evolve and change significantly before the release of the final Revision 2.0 A.G.P. Interface Specification.

Table 4-8: A.G.P. 4X AC Timing Parameters

Symbol	Parameter	Min	Max	Units	Notes
Transmitter Output Signals:					
t_{TSf}	**CLK** to first transmit strobe transition	**1.9**	**8**	ns	1
t_{TSr}	**CLK** to 4th transmit strobe transition		**20**	ns	1
t_{Dvb}	Data valid before strobe	**1.0**		ns	
t_{Dva}	Data valid after strobe	**1.2**		ns	
t_{ONd}	**Float to Active Delay**	**-1**	**7**	**ns**	
t_{OFFd}	**Active to Float Delay**	**1**	**14**	**ns**	
t_{ONS}	Strobe active to first strobe edge setup	**4**	**9**	ns	
t_{OFFS}	Last strobe edge to strobe float delay	**9**	**-4**	ns	
Receiver Input Signals:					
t_{RSsu}	Receive strobe setup time to **CLK**	**6**		ns	2
t_{RSh}	Receive strobe hold time from **CLK**	**0.5**			2
t_{Dsu}	Data to strobe setup time	**0.45**		ns	
t_{Dh}	Strobe to data hold time	**0.65**		ns	

Notes:

1. See Figure 4-15 and Section 4.1.2.8 for more details.
2. These specifications refer to the setup and hold times for the strobe set started in the previous cycle. See Figure 4-15 and Section 4.1.2.9 for more details.

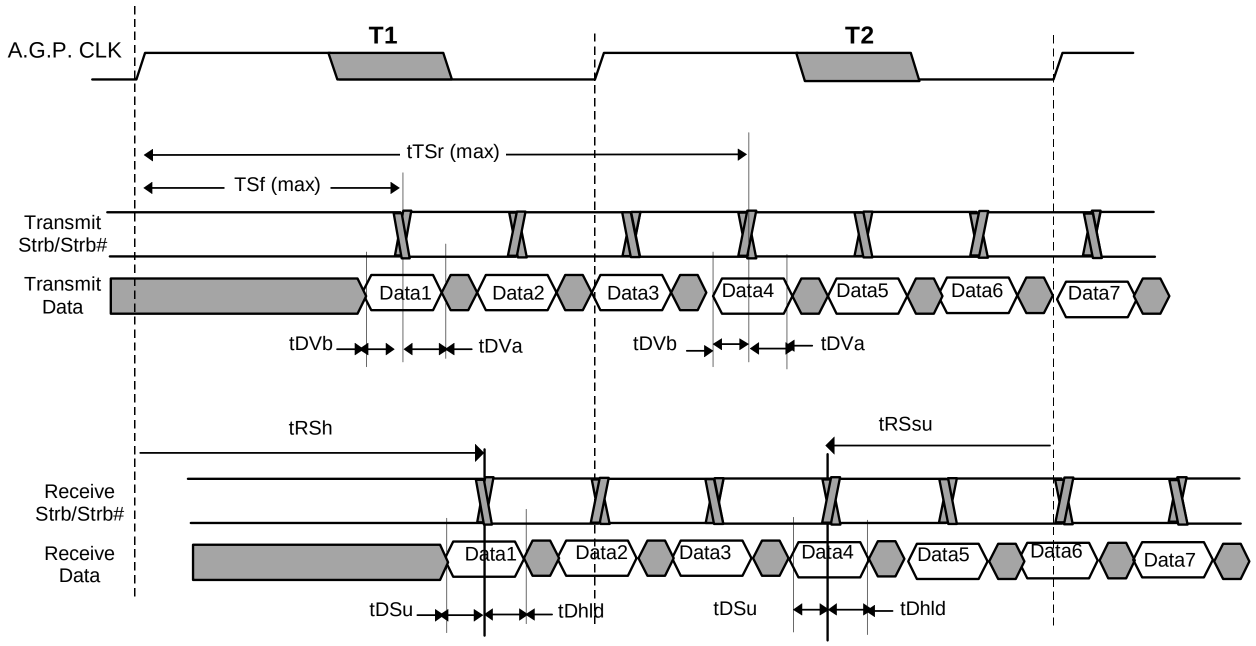


Figure 4-15: 4X Mode Timing Diagram

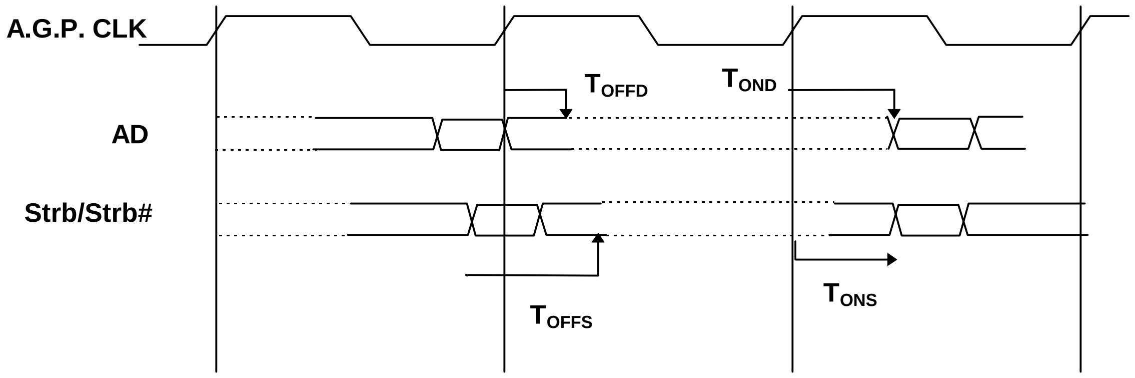


Figure 4-16: 4X Mode Strobe/Data Turnaround Timings

4.2.3 Measurement and Test Conditions

This section gives the measurement and test conditions for design. Production test systems and boards have different capacitive loads than specified below. It is the responsibility of the manufacturer to correlate those systems to the reference points given in this section. The reference point for all AC timing measurements is 0.4Vddq for 3.3 volt signaling and 0.5Vddq for 1.5 volt signaling. The output capacitive loading (C_L) for all maximum timings is 10 pF and for all minimum timings is 0 pF.

Table 4-9: Measurement and Test Condition Parameters

Symbol	3.3 V Signaling Levels	1.5 V Signaling Levels	Units	Notes
V_{th}	0.6Vddq	0.7Vddq	V	1
V_{tl}	0.2Vddq	0.3Vddq	V	1
V_{test}	0.4Vddq	0.5Vddq	V	
V_{max}	0.4Vddq	0.4Vddq	V	1
Input Signal Slew Rate	1.5-4.0	0.9-2.3	V/ns	2

Notes:

1. The test is done with 0.1*Vddq of overdrive. V_{max} specifies the maximum peak-to-peak waveform allowed for testing input timing.
2. Outputs will be characterized and measured at the package pin with the load shown in Figure 4-20. Input signal slew rate will be measured between V_{tl} and V_{th}.

4.2.3.1 1X Mode Measurements

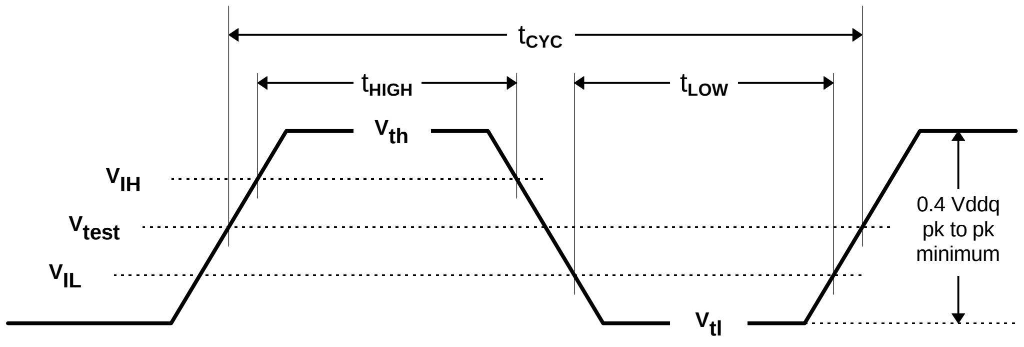


Figure 4-17: Clock Input Measurement Conditions

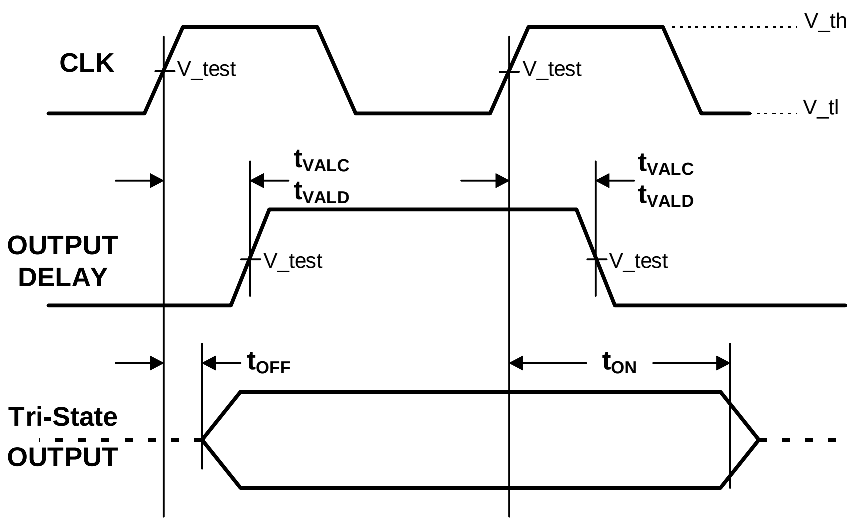


Figure 4-18: Output Timing Measurement Conditions

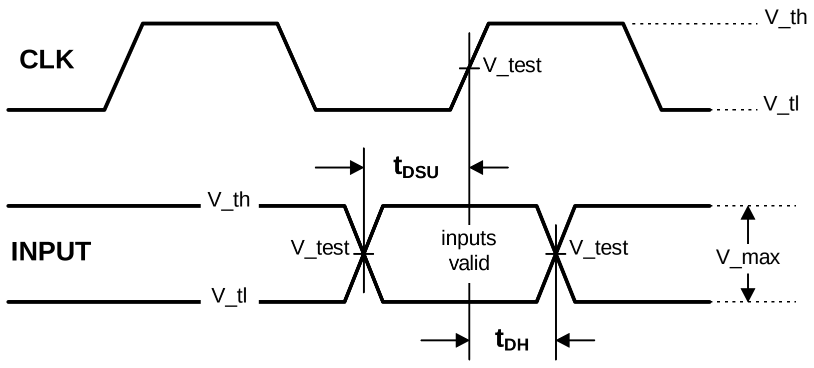


Figure 4-19: Input Timing Measurement Conditions

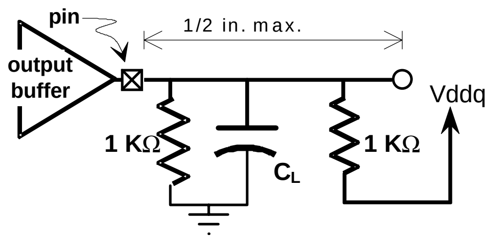


Figure 4-20: Load for Testing Output Timings

4.2.3.2 2X Mode Measurements

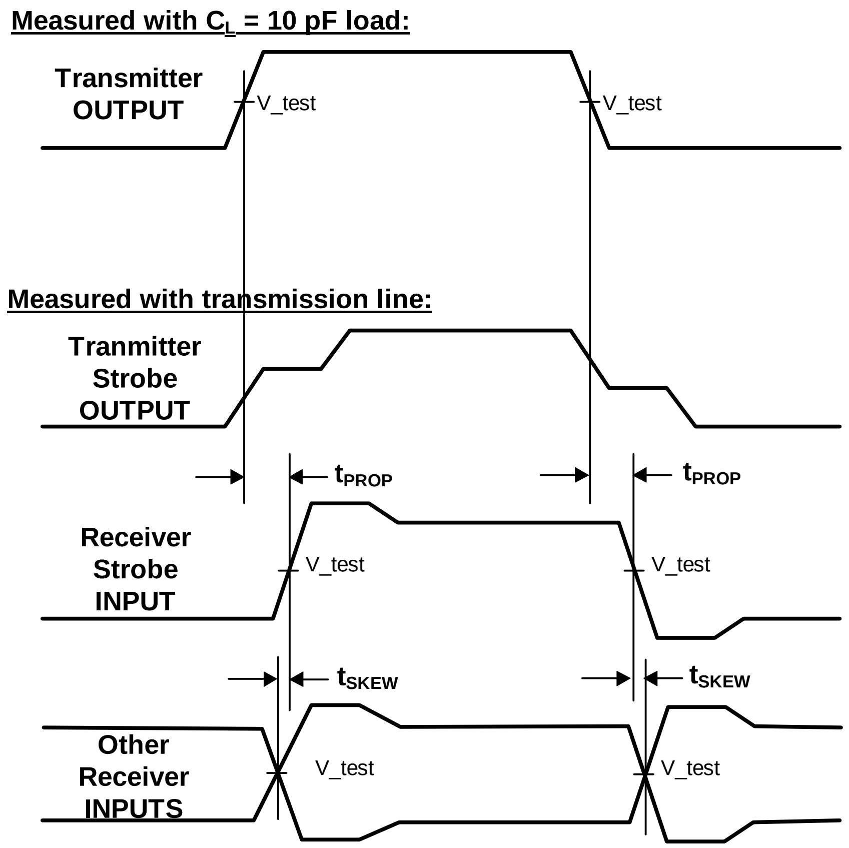


Figure 4-21: Flight Time and Skew Measurement

4.2.3.3 4X Mode Measurements

TBD

4.3 General System Specifications

4.3.1 Physical Requirements

The AC timings and electrical loading on the A.G.P. interface are optimized for one active host component on the motherboard and one active A.G.P. agent either on the motherboard or through a connector. The interface is a logical point-to-point[34] network. The board routing should use layout design rules consistent with high speed digital design. Due to the high speed nature of the A.G.P. bus, any design should be thoroughly simulated and every effort should be made to reduce signal skew and improve signal quality. If a bus with more than two loads and/or branching in the topology is implemented, it is the system designers responsibility to ensure compliance to this interface specification. This can be accomplished by thoroughly simulating the design to ensure proper signal quality and that the timings are met. These topologies are not shown or discussed in this interface specification due to the difficulty in designing them, especially at the higher transfer rates, and it is recommended that a physical point-to-point topology be used.

4.3.2 Clock Skew

The maximum total system clock skew is 1 ns for all data transfer modes. This 1 ns 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 4-10 and Figure 4-22.[35] This is measured between the pins of the two A.G.P. components (not at the connector).

The total skew and jitter is allocated so that 0.1 ns originates from the add-in card routing, and 0.9 ns originates from the motherboard routing and clock synthesizer (the motherboard designer shall determine how the 0.9 ns 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 4.4.6.1.

Table 4-10: Clock Skew Parameters, All Transfer Rates

Symbol	1.5 Volt Signaling	3.3 Volt Signaling	Units
V_{test}	0.5Vddq	0.4Vddq	V
T_{skew}	±1 (max)	±1 (max)	ns

[34] See footnote 3 on page 25.

[35] 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.

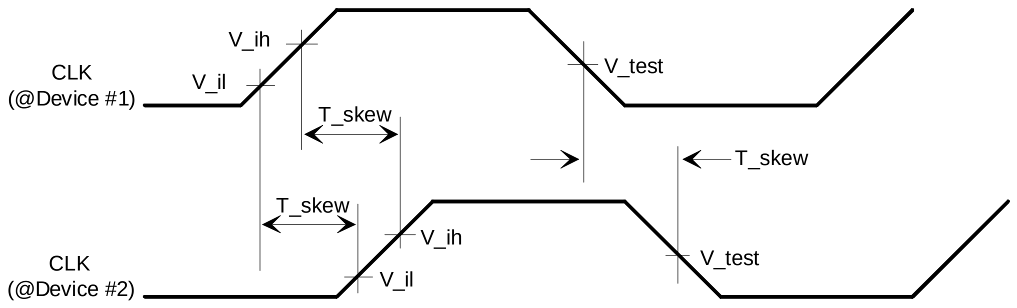


Figure 4-22: Clock Skew Diagram

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 inTable 4-6. The specification does not preclude the implementation of a synchronous **RST#**, if desired. The timing parameters for reset are contained in Table 4-6, 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 specification. 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 5 V rail falling below the 3.3 V 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 Interface Signaling

A.G.P. interface signals (not including power supplies) fall into three categories: Vddq signaling, 3.3 volt signaling and special signaling. The signals in the Vddq group are **REQ#**, **GNT#**, **ST[2::0]**, **FRAME#**, **TRDY#**, **IRDY#**, **DEVSEL#**, **STOP#**, **SERR#**, **PERR#**, **PAR**, **RBF#**, **PIPE#**, **AD[31::00]**, **CB/E[3::0]#**, **SBA[7::0]**, **AD_STB[1::0]**, and **SB_STB**. These signals scale with and reference to the Vddq power supply. **WBF#**, **STB[1::0]#**, **AD_STB[1::0]#,** and **SB_STB#** are also Vddq signals for A.G.P. 4X signaling.

The 3.3 volt signals are **CLK**, **RST#**, **PME#**, **INTA#**, and **INTB#**. Their signal levels are referenced to the **VCC3.3** power supply. These signals may require special handling for a device where the logic is powered by a supply of less than 3.3 volts. **CLK** and **RST#** may require special input circuitry on the controller or dividers have to be provided to prevent overvoltage on the pin and distortion of the source signal (see the *A.G.P. Design Guide*). The A.G.P. Master outputs **PME#**, **INTA#**, and **INTB#** need to be 3.3 volt tolerant.

The interrupt signals from the A.G.P. bus must interface to the PCI bus interrupt and power management controllers. These controllers and the PCI devices may be +5 V 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 pull up the

PCI interrupts to 3.3 V 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 5 V environment from the A.G.P. bus.

In the special interface group, **TYPEDET#** indicates whether the interface is 1.5 volt or 3.3 volt. If **TYPEDET#** is open (not connected to any power rail or signal), the interface is 3.3 volt signaling. If **TYPEDET#** is shorted to ground, the interface is 1.5 volt. Note that only a system with a universal connector will respond to the condition of the **TYPEDET#** signal. The remaining signals in the special interface group are the Universal Serial Bus signals **USB+**, **USB-**, and **OVRCNT#**. More information on these signals can be found in the "USB Design Considerations" section and in the *Universal Serial Bus Specification*.

4.3.5 Motherboard / Add-in Card Interoperability

The A.G.P. interface can be classified by its signaling voltage level and maximum transfer rate capability. Interoperability of a particular add-in card with a particular motherboard is dependent only on the signaling levels. Signaling level is determined by the value of the Vddq I/O interface voltage. If the add-in card and the motherboard have the same Vddq or if the motherboard has a universal connector, the add-in card will work with the motherboard. A key in the A.G.P. connector prevents add-in cards from being plugged into an incompatible motherboard. Universal connectors have no key and will accept either 1.5 or 3.3 volt Vddq add-in cards. The **TYPEDET#** pin on the add-in card tells the universal motherboard which value of Vddq is to be supplied to the interfaces as described above.

A given part will also have a maximum transfer rate capability. However, every A.G.P. agent must be capable of operating at all lower transfer rates as well (i.e., a 4X mode part must also work at 2X and 1X rates). The transfer rate actually used is determined at configuration time and is likely to be the largest common rate between the A.G.P. master and target. Table 4-11 shows the matrix of compatible motherboards and add-in cards.

Table 4-11: Motherboard / Add-in Card Interoperability

	1.5 Volt Add-in Card			3.3 Volt Add-in Card	
Motherboard	**1X Capable**	**2X Capable**	**4X Capable**	**1X Capable**	**2X Capable**
1.5 Volt - 2X Mode Capable	✔	✔	(1)	▨	▨
1.5 Volt - 4X Mode Capable	✔	✔	✔	▨	▨
3.3 Volt - 2X Mode Capable	▨	▨	▨	✔	✔
Universal - 2X Mode Capable	✔	✔	(1)	✔	✔
Universal - 4X Mode Capable	✔	✔	✔	✔	✔

▨ Indicates that this combination is precluded by position of key in A.G.P. connector

Notes:

1. A 4X transfer mode capable add-in card can be plugged into a 2X capable motherboard, but it will run only at 2X transfer rate.

This interoperability requires that all A.G.P. components and boards have a superset of the electrical characteristics of all the signaling levels and transfer rates it supports. This includes I/O timings, buffer drive characteristics, input signal clamping, and board layout requirements. In general, the higher transfer rate electrical interface and board requirements are backward compatible to the lower rates for a particular signaling level. The A.G.P. interface for a core logic component on a universal motherboard must meet all electrical requirements up to its maximum capable transfer rate. (Note: The 4X mode characteristics are not final at this time. Inherent backward compatibility is not guaranteed.)

4.3.6 Pull-ups

A.G.P. control signals require pull-ups to Vddq on the motherboard (or, optionally, integrated on the 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#**, **RBF#**, **INTA#**, **INTB#**[36], **PIPE#**, **AD_STB[1::0]**, and **SB_STB**. The core logic (A.G.P. target or motherboard) may require weak pull-ups on **REQ#** and **SBA[7::0]** to insure that these signals do not float when there is no add-in card in the connector. Values for this pull-up shall be specified by the core logic vendor.

Pull-ups are allowed on any A.G.P. pin. Care should be taken when attaching pull-ups to the **AD**, **SBA**, **AD_STB[1::0]**, **SB_STB**, or **C/BE#** signals. The trace stub to the pull-up on these signals should be kept to less than 0.1 inch to avoid signal reflections from the stub.

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

Table 4-12: Pull-up Resistor Values

Rmin	Rtypical	Rmax	Notes
4 KΩ	8.2 KΩ @ 10%	16 KΩ	

4.3.7 Maximum AC Ratings and Device Protection

All A.G.P. input, bidirectional, and tri-state output buffers should be capable of withstanding continuous exposure to the waveform shown in Figure 4-23. 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 4-23, which is based on the expected worst case overshoot and undershoot expected in actual A.G.P. buses. 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.

[36] **INTA#** and **INTB#** are special cases. The motherboard should ensure that these signals cannot float to other motherboard inputs, while meeting the interface signaling requirements in Section 4.3.4. If **INTA#** or **INTB#** are connected to inputs on the add-in card, the add-in card design must ensure those inputs cannot float.

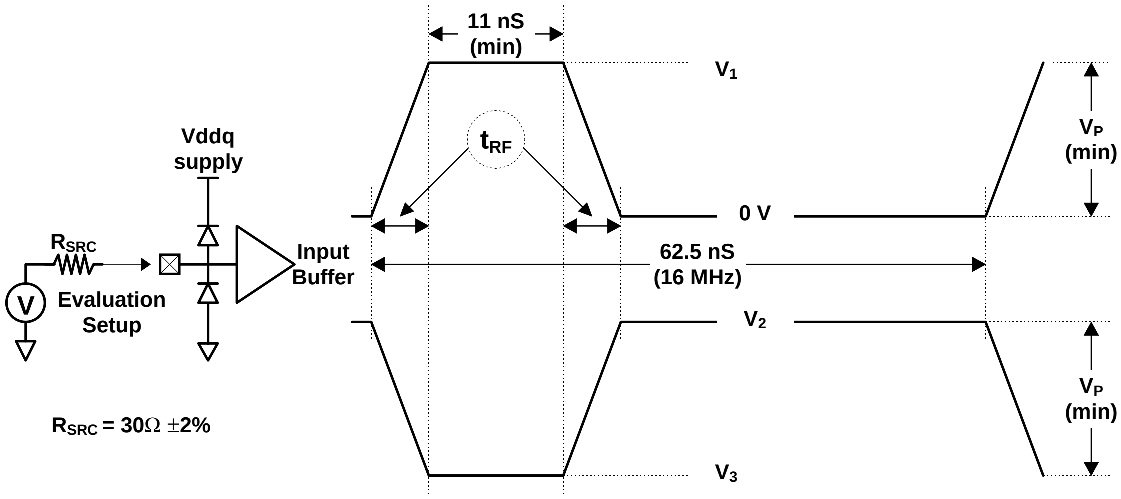


Figure 4-23: Maximum AC Waveforms for A.G.P. Signaling

Table 4-13: Parameters for Maximum AC A.G.P. Signaling Waveforms

Symbol	Parameter	3.3 Volt Signaling		1.5 Volt Signaling		Units
		Min	Max	Min	Max	
V_1	Overshoot voltage	7.1		3.1		V
V_2	Undershoot initial voltage	3.3	3.6	1.5	1.6	V
V_3	Undershoot voltage		-3.5		-1.5	V
V_P	Waveform peak-to-peak	7.1		3.1		V
t_{RF}	Rise fall time	1.5	4.0	0.9	2.3	ns

4.3.8 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. The VCC3.3 and Vddq power supplies must be sequenced such that the Vddq voltage level is never more than 0.5 volts above the level of VCC3.3. In systems with a universal connector and programmable Vddq, the Vddq cannot exceed the maximum limits of the supply value selected by **TYPEDET#** at any time.

The motherboard must connect all power supply pins on the connector as shown in Chapter 5 to guarantee proper current delivery and to provide proper AC signal return paths. The add-in card should also attach all connector power pins to appropriate power planes on the card for good power delivery and signal returns. Add-in cards must use all ground pins, and any power pins not used must be bypassed to ground on the card with a good quality, low inductance 0.01 µF or larger capacitor.

Table 4-14 lists the voltage ranges for the supplies to the add-in card and the maximum currents that can be supplied via the connector.

Table 4-14: Add-in Card Power Supply Limits

Symbol	Parameter	Condition	Min	Max	Units	Notes
Vddq1.5	I/O Supply Voltage	I_{MAX} = 8.0 A	1.425	1.575	V	1
Vddq3.3	I/O Supply Voltage	I_{MAX} = 8.0 A	3.15	3.45	V	1
VCC3.3	3.3 V Power Supply	I_{MAX} = 6.0 A	3.15	3.45	V	
VCC5	5 V Power Supply	I_{MAX} = 2.0 A	4.75	5.25	V	
VCC12	12 V Power Supply	I_{MAX} = 1.0 A	11.4	12.6	V	

Note:

1. The Vddq current is due only to the AC switching transients of the A.G.P. I/O buffers. This level should not be seen in practice, but represents the current carrying capability of the connector. Actual currents will be less than 1.5 A.

4.3.9 USB Design Considerations

When USB is included on the add-in card, there are two key issues that must be addressed: signaling and power delivery. The USB signal lines on the add-in card need to be designed to 45Ω ±15% to match the impedance of the USB drivers and cable in order to preserve the signal integrity. Care should also be taken to avoid coupling to any high frequency signal as this can cause EMI radiation problems when a cable is attached. The power lines should be properly bypassed to decouple noise. If USB is provided through a standard USB connector, it has to support hot attach. The power lines to the connector have to have sufficient bulk capacitance to filter the surge currents. Refer to the *Universal Serial Bus Specification* for more details.

Any system that delivers power to a cable (i.e., for I^2C and USB) will need to provide overcurrent protection on the card to comply with regulatory safety requirements (UL, CSA, etc.). This can take the form of active current limiting circuits or a simple self-resetting fuse. The overcurrent protection should be set to limit the current available to the cable at 2 A. Additionally, USB requires that this condition be reported on the **OVRCNT#** pin. The overcurrent indication can be taken from the active limiting circuit or by a simple monitoring of the power condition on the cable side of the fuse. In the latter case, the cable power supply voltage must be reduced to be within the Voh specification range. A simple resistive divider, taking account of load current, is sufficient. If the card does not provide power to the cable, then the **OVRCNT#** pin should be tied to 3.3 volts via a pullup resistor.

A USB capable motherboard must be careful with the USB signal lines. They must be designed to 45 Ω ±15% to match the driver and cable impedance and carefully routed to avoid picking up high frequency noise that will radiate on the cable. Also, the 15 KΩ ±5% pull-down resistors required by the USB specification must be located on the motherboard. Lastly, the **OVRCNT#** signal should be tied to a 3.3 volt supply through a high value resistor. The **OVRCNT#** signal should be above the Voh level and the total current from the pullup and USB host controller component is less than the 20 µA leakage limit.

4.4 1X and 2X Transfer Mode Specifications

4.4.1 Signal Integrity Requirement

Table 4-15: Signal Integrity Requirements[1]

		A.G.P. 1X				A.G.P. 2X					
		3.3 Volt Signaling		1.5 Volt Signaling		3.3 Volt Signaling		1.5 Volt Signaling			
Symbol	**Parameter**	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**	**Units**	**Notes**
	Output Slew Rate	1.5	4.0	0.9	2.3	1.5	4.0	0.9	2.3	V/ns	2,3
Tset	Output settling time to ±20% of rail						7.0		7.0	ns	4

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 65 Ω ± 15 Ω; and 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 2.5 V/ns.
4. Settling time is a recommendation, not a requirement. All delay simulations and measurements (flight times, flight time skews) should be performed such that they include the impact settling time may have on interconnect delay, as described in the *A.G.P. Design Guide*.

4.4.2 1X and 2X Mode Driver Characteristics

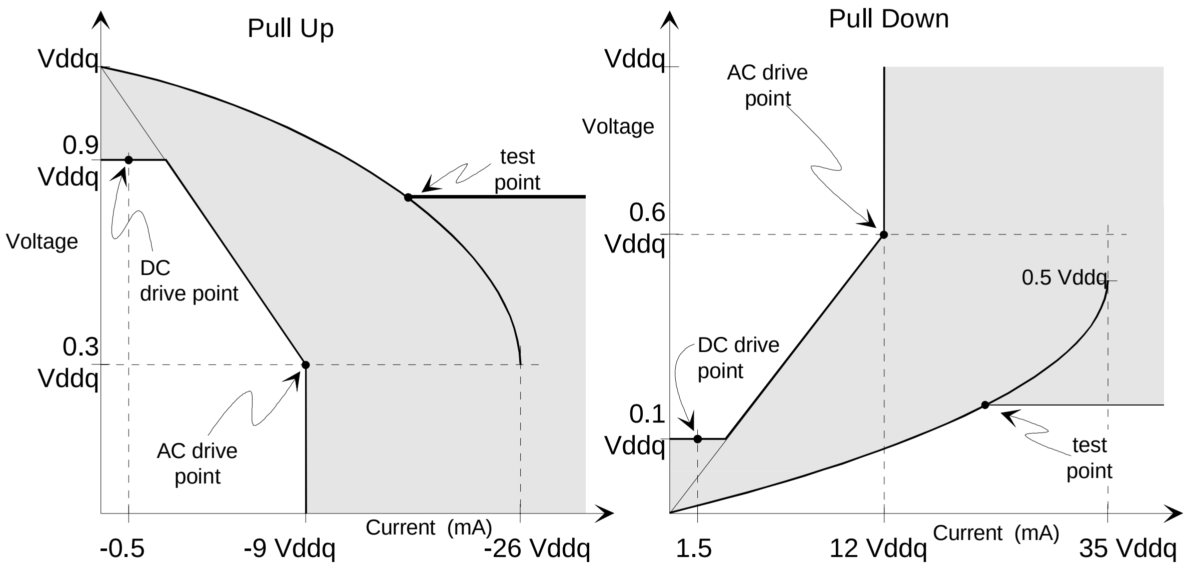
To provide optimal performance in a point-to-point environment, A.G.P. requires a driver that is roughly *half* the strength of the PCI buffer. The output driver must be able to deliver an initial voltage swing of at least the VIL/VIH value to the receiver through the bus with a known characteristic 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 (see Figure 4-24 and Figure 4-25). 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 maximum 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 50 Ω 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 A:

$I_{oh} = (54.0/Vddq)*(V_{out}-Vddq)*(V_{out}+0.4Vddq)$

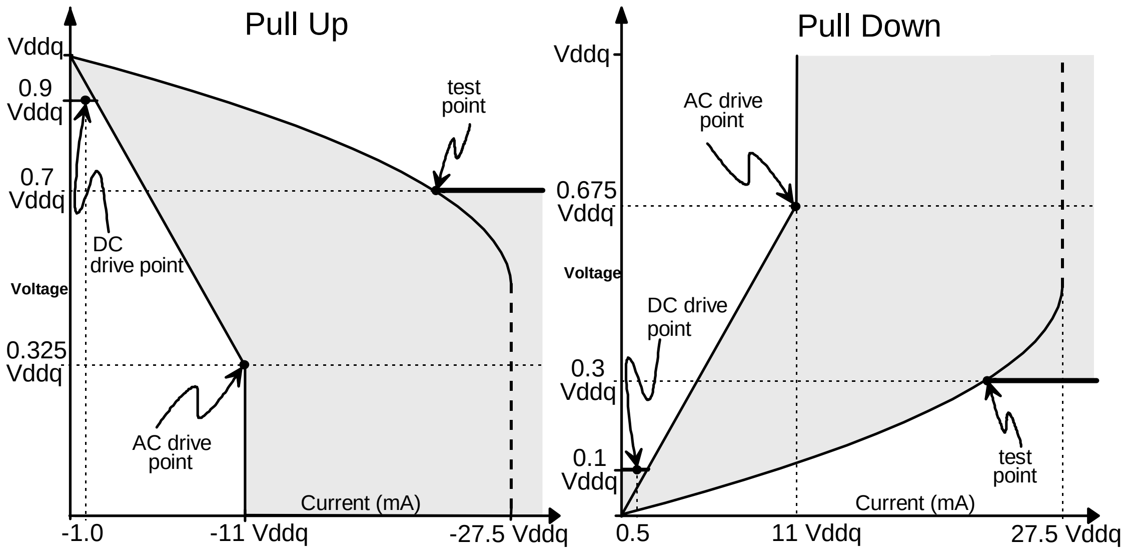
for $Vddq > V_{out} > 0.7\ Vddq$

Equation B:

$I_{ol} = (141/Vddq)*V_{out}*(Vddq-V_{out})$

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

Figure 4-24: V/I Curves for 3.3 Volt Signaling



Equation C:

I_{oh} = (111.0/Vddq)(V_{out}-Vddq)*V_{out}*

for Vddq > V_{out} > 06757 Vddq

Equation D:

I_{ol} = (111/Vddq)V_{out}*(Vddq-V_{out})*

for 0v < V_{out} < 0.325 Vddq

Figure 4-25: V/I Curves for 1.5 Volt Signaling

Inputs are required to be clamped to both ground and Vddq 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 specification momentarily. Diode clamps to a power rail, as well as output pullup devices, must be able to withstand short circuit current until drivers can be tri-stated. The clamp diode characteristics are listed in Table 4-16 and Table 4-17.

Table 4-16: AC Specifications for A.G.P. 3.3 Volt Signaling

Symbol	Parameter	Condition	Min	Max	Units	Notes
$I_{oh(AC)}$	Switching	$0<V_{out}\le 0.3V_{ddq}$	$-9V_{ddq}$		mA	1
	Current High	$0.3V_{ddq}<V_{out}<0.9V_{ddq}$	$-13(V_{ddq}-V_{out})$	Eqt'n A	mA	1,2
	(Test Point)	$V_{out} = 0.7V_{ddq}$		$-18V_{ddq}$	mA	2
$I_{ol(AC)}$	Switching	$V_{ddq}>V_{out}\ge 0.6V_{ddq}$	$12V_{ddq}$		mA	1
	Current Low	$0.6V_{ddq}>V_{out}>0.1V_{ddq}$	$20V_{out}$	Eqt'n B	mA	1,2
	(Test Point)	$V_{out} = 0.18V_{ddq}$		$21V_{ddq}$	mA	2
I_{cl}	Low Clamp Current	$-3<V_{in} \le -1$	$-25+(V_{in}+1)/0.015$		mA	
I_{ch}	High Clamp Current	$V_{ddq}+4>V_{in} \ge V_{ddq}+1$	$25+(V_{in}-V_{ddq}-1)/0.015$		mA	
$slew_r$	Output Rise Slew Rate	$0.2V_{ddq}$ - $0.6V_{ddq}$ I load	1.5	4	V/ns	3
$slew_f$	Output Fall Slew Rate	$0.6V_{ddq}$ - $0.2V_{ddq}$ I load	1.5	4	V/ns	3

Table 4-17: AC Specifications for A.G.P. 1.5 Volt Signaling

Symbol	Parameter	Condition	Min	Max	Units	Notes
$I_{oh(AC)}$	Switching	$0<V_{out}\le 0.325V_{ddq}$	$-11V_{ddq}$		mA	1
	Current High	$0.325V_{ddq}<V_{out}<0.9V_{ddq}$	$-16.5(V_{ddq}-V_{out})$	Eqt'n C	mA	1,2
	(Test Point)	$V_{out} = 0.7V_{ddq}$		$-23.3V_{ddq}$	mA	2
$I_{ol(AC)}$	Switching	$V_{ddq}>V_{out}\ge 0.675V_{ddq}$	$11V_{ddq}$		mA	1
	Current Low	$0.675V_{ddq}>V_{out}>0.1V_{ddq}$	$16.5V_{out}$	Eqt'n D	mA	1,2
	(Test Point)	$V_{out} = 0.3V_{ddq}$		$23.3V_{ddq}$	mA	2
I_{cl}	Low Clamp Current	$-3<V_{in} \le -1$	$-25+(V_{in}+1)/0.015$		mA	
I_{ch}	High Clamp Current	$V_{ddq}+4>V_{in} \ge V_{ddq}+1$	$25+(V_{in}-V_{ddq}-1)/0.015$		mA	
$slew_r$	Output Rise Slew Rate	$0.3V_{ddq}$ - $0.7V_{ddq}$ I load	0.9	2.3	V/ns	3
$slew_f$	Output Fall Slew Rate	$0.7V_{ddq}$ - $0.3V_{ddq}$ I load	0.9	2.3	V/ns	3

Notes:

1. Refer to the V/I curves in Figure 4-24 and Figure 4-25.

2. Equations A and B are provided with the respective diagrams in Figure 4-24 and equations C and D are provided with the respective diagrams in Figure 4-25. The equation defined maxima should be met by design. In order to facilitate component testing, a maximum current test point is defined for each side of the output driver.
3. This parameter is to be interpreted as the cumulative edge rate across the specified range, rather than the instantaneous rate at any point within the transition range. The specified load (Figure 4-20) is optional; i.e., the designer may elect to meet this parameter with an unloaded output. However, adherence to both maximum and minimum parameters is required (the maximum is not simply a guideline).

4.4.3 1X and 2X Mode Receiver Characteristics

A differential input receiver is recommended for the 1X and 2X operation. The voltage reference for 3.3 volt signaling is specified at 0.4Vddq. This reference voltage is compatible with the PCI 66 MHz VIL/VIH specification. For 1.5 volt signaling, the voltage reference is 0.5Vddq. This gives the best noise margin at the reduced signal swing. A differential input receiver is not a strict requirement, as long as an implementation can meet all other AC/DC input specifications.

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 5.0 to 10 µA. The differential input buffer must be designed to have sufficient gain to convert an input differential voltage (~100 mV) to a full internal CMOS voltage swing, without introducing additional skews.

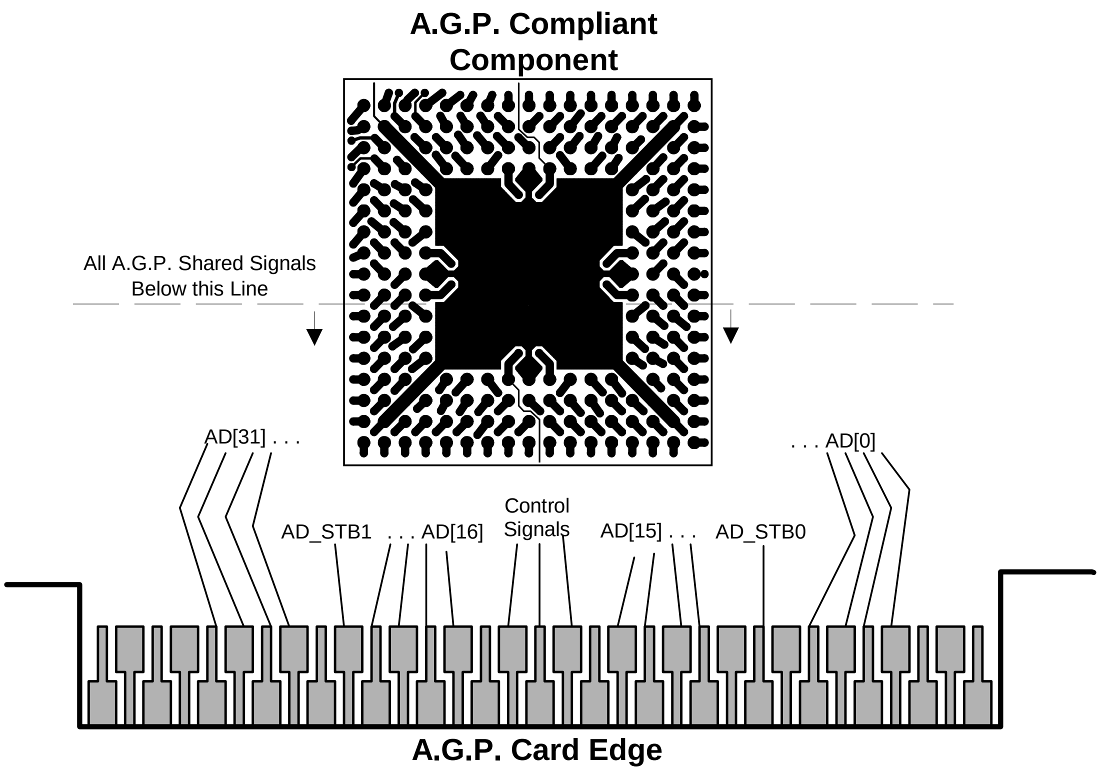
4.4.4 Component Pinout Recommendations

All A.G.P. signals on the graphics chip should be located to facilitate meeting the add-in card requirements as specified in Section 4.4.6. The component pinout should be ordered to match the connector pinout and component side of the add-in card as defined in Chapter 5. This alignment will minimize signal crossing, minimize overall trace lengths, and aid in matching the trace lengths within the groups. See Figure 4-26 for the recommended component pinout.

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

- **AD_STB0** with **AD[15::00]** 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.



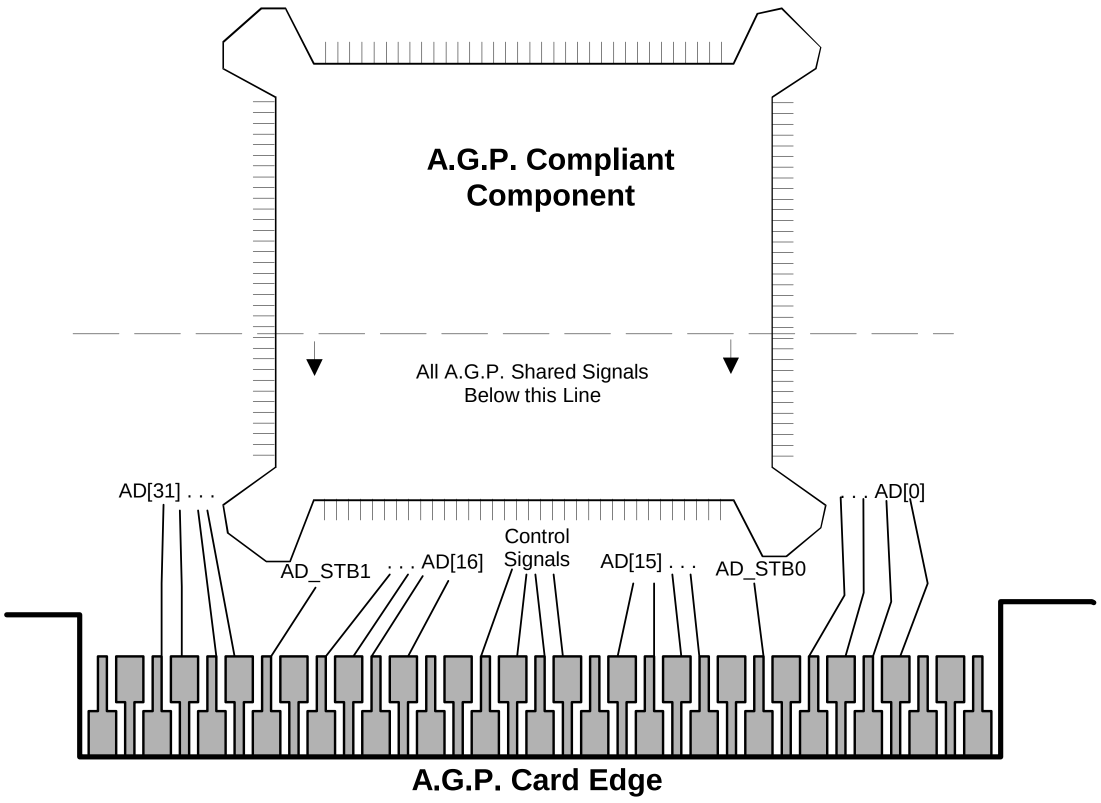


Figure 4-26: Recommended Component Pinout

4.4.5 1X and 2X Mode Motherboard Specifications

4.4.5.1 System Timing Budget

Table 4-18 summarizes the system timing parameters for 1X mode signals.

Table 4-18: 1X / 2X Mode System Timing Summary

Timing Element	Parameter	Max	Units	Notes
T_{cyc}	Cycle Time	15	ns	
T_{val}	Valid Delay	5.5/6.0	ns	1
T_{prop}	Prop Delay	2.5	ns	
T_{su}	Input Setup	6.0/5.0	ns	1
$T_{skew\text{-}total}$	Total Skew	1	ns	2
$T_{skew\text{-}mb}$	Motherboard Skew	.9	ns	
$T_{skew\text{-}add\text{-}in}$	Add-in Card Skew	.1	ns	

Notes:

1. Control signal / Data line specification
2. Tskew is the sum of all skews (motherboard and add-in).

Table 4-19 summarizes all system interconnect delays. Note that the individual motherboard and add-in card components are repeated later in their respective sections.

Table 4-19: 1X / 2X Mode Interconnect Delay Summary

Symbol	Parameter	Max[1]	Units	Notes
t_{PROP}	Signal propagation	**2.5**	ns	2
$t_{PROP-MB}$	Signal propagation, motherboard	**1.65**	ns	5,9
$t_{PROP-CONN}$	Signal propagation, connector	**.15**	ns	9
$t_{PROP--CARD}$	Signal propagation, add-in card	**.7**	ns	6,9
$t_{TRMATCH}$	Total Trace mismatch between data and strobe	**.7**	ns	3,4,7,8,9
$t_{TRMATCH-MB}$	Trace mismatch, motherboard	**.5**	ns	4,7,9
$t_{TRMATCH-CARD}$	Trace mismatch, card	**.2**	ns	4,8,9

Notes:

1. Signal propagation delays are measured as the difference between the driver driving a 10 pF lumped load vs. the driver driving an 80 Ω transmission line terminated by a 10 pF 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]** and **C/BE[3::2]#**; **AD_STB0=>AD[15::00]** and **C/BE[1::0]#**; **SB_STB=>SBA[7::0]**. The trace mismatch specification only applies between the strobe and data signals within a group, not between data signals within a group or between groups.
5. Recommended motherboard trace lengths: 1.0 - 9 (inches) depending on trace spacing.
6. Recommended add-in card trace lengths: 0.0 - 3.0 (inches).
7. Recommended motherboard matching between any data trace and its associated **STB#** trace:
   L_data - L_stb = -0.5 to 0.0 (inches).
8. Recommended add-in card matching between any data trace and its associated **STB#** trace:
   L_data - L_stb = -0.5 to +0.5 (inches).
9. Trace length and trace length matching are recommendations based on interconnect simulations including a wide variety of transmission line and loading effects. Designers must ensure through simulation or other techniques that the interconnect timing requirements will still be met.

4.4.5.2 Interconnect Delay

Table 4-20: 1X / 2X Mode Motherboard Interconnect Delays

Symbol	Parameter	Max[1]	Units	Notes
$t_{PROP-MB}$	Signal propagation, motherboard	**1.65**	ns	3,5
$t_{PROP-CONN}$	Signal propagation, connector	**.15**	ns	5
$t_{TRMATCH-MB}$	Trace mismatch, motherboard	**.5**	ns	2,4,5

Notes:

1. Signal propagation delays are measured as the difference between the driver driving a 10 pF lumped load vs. the driver driving an 80 ohm transmission line terminated by a 10 pF lumped load.
2. Trace mismatch applies between signal groups and their associated strobes: **AD_STB1=>AD[31::16]** and **C/BE[3::2]#**; **AD_STB0=>AD[15::00]** and **C/BE[1::0]#**; **SB_STB=>SBA[7::0]**. The trace mismatch specification only applies between the strobe and data signals within a group, not between data signals within a group or between groups.
3. Recommended motherboard trace lengths: 1.0 - 9 (inches) depending on trace spacing.
4. Recommended motherboard matching between any data trace and its associated **STB#** trace:
   L_data - L_stb = -0.5 to 0.0 (inches).
5. Trace length and trace length matching are recommendations based on interconnect simulations including a wide variety of transmission line and loading effects. Designers must ensure through simulation or other techniques that the interconnect timing requirements will still be met.

4.4.5.3 Physical Requirements

The AC timings and electrical loading on the A.G.P. interface are optimized for one active host component on the motherboard and one active A.G.P. agent either on the motherboard or through a connector. The interface is a logical point-to-point network, with a maximum electrical length of 2.5 ns. The board routing should use layout design rules consistent with high speed digital design. Due to the high speed nature of the A.G.P. bus, any design should be thoroughly simulated and every effort should be made to reduce signal skew and improve signal quality. If a bus with more than two loads and/or branching in the topology is implemented, it is the system designers responsibility to ensure compliance to this interface specification. This can be accomplished by thoroughly simulating the design to ensure proper signal quality and that the timings are met. These topologies are not shown or discussed in this interface specification due to the difficulty in designing them and it is recommended that a physical point-to-point topology be used. The following paragraphs summarize the board layout restrictions on the 2X transfer mode interface.

4.4.5.4 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 board routing to verify that the specifications are met for flight times and skews due to trace mismatch and crosstalk.

The total flight time allowed for the A.G.P. bus is 2.5 ns. The timing budget for the components of the flight path is identified in Table 4-18. The motherboard prop delay budget of 1.65 ns restricts the total trace length on the motherboard to 9 inches or less, depending on trace spacing. See the *A.G.P. Design Guide* for more details.

The trace lengths for signals within a group must be matched to meet the total mismatch requirement given in Table 4-18, of 0.5 ns. This means that the traces within a group on the motherboard must be matched to their respective strobe trace, so that the length of each data trace is either equal in length to the strobe trace or up to 0.5 inches shorter than the strobe trace. (The strobe should always be the longest motherboard trace in each group.)

4.4.5.5 Crosstalk Consideration

For 66 and 133 MT/s modes, noise due to crosstalk must be carefully controlled to a minimum. Crosstalk is the key cause of timing skews and is the largest part of the tRMATCH skew parameter. Refer to the *A.G.P. Design Guide* for typical values.

4.4.5.6 Impedances

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

4.4.5.7 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.

4.4.5.8 Line Termination

Line termination mechanisms are not specified for the A.G.P. interface. Internal or external termination circuits may be used to meet signal integrity requirements as long as these elements do not inhibit either sending or receiving agents from meeting their performance specifications. Active clamping devices and slew rate controlled output buffers achieve acceptable signal integrity by controlling signal reflection, over/undershoot, and ringback. Some form of receiver termination is more likely with 1.5 volt signaling than with 3.3 volt signaling due to the tighter settling times and smaller ringback margins. Note that the receiver must allow for any drive strength and board impedance characteristic within the specified ranges. See the *A.G.P. Design Guide* for more details.

4.4.6 1X and 2X Mode Add-in Card Specifications

4.4.6.1 Clock Skew

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

4.4.6.2 Interconnect Delay

Table 4-21: 1X / 2X Mode Add-in Card Interconnect Delays

Symbol	Parameter	Max[1]	Units	Notes
$t_{PROP\text{--}CARD}$	Signal propagation, add-in card	**.7**	ns	3,5
$t_{TRMATCH\text{-}CARD}$	Trace mismatch, add-in card	**.2**	ns	2,4,5

Notes:

1. Signal propagation delays are measured as the difference between the driver driving a 10 pF lumped load vs. the driver driving an 80 Ω transmission line terminated by a 10 pF lumped load.
2. Trace mismatch applies between signal groups and their associated strobes: **AD_STB1=>AD[31::16]** and **C/BE[3::2]#**; **AD_STB0=>AD[15::00]** and **C/BE[1::0]#**; **SB_STB=>SBA[7::0]**. The trace mismatch specification only applies between the strobe and data signals within a group, not between data signal within a group or between groups.
3. Recommended add-in card trace lengths: 0.0 - 3.0 (inches).
4. Recommended add-in card matching between any data trace and its associated **STB#** trace:
   L_data - L_stb = -0.5 to +0.5 (inches).
5. Trace length and trace length matching are recommendations based on interconnect simulations including a wide variety of transmission line and loading effects. Designers must ensure through simulation or other techniques that the interconnect timing requirements will still be met.

4.4.6.3 Physical Requirements

4.4.6.4 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. component's I/O buffers must be powered from these special pins only — not from the other power pins.[37]

4.4.6.5 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. It is recommended that the board designer simulate the board routing to verify that the specifications are met for flight times and skews due to trace mismatch and crosstalk.

The total flight time allowed for the A.G.P. bus is 2.5 ns. The timing budget for the components of the flight path is identified in Table 4-19. The add-in card prop delay budget of 0.7 ns 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 4-21, of 0.2 ns. This means that the traces within a group on the add-in card must be matched to their respective strobe trace, so that the length of each data trace in the group is within ± 0.5 inch of the strobe length. Refer to Figure 4-26 for add-in card component placement recommendations.

4.4.6.6 Impedances

The add-in card impedances should be controlled to minimize the impact of any mismatch between the motherboard and the add-in card. An impedance of 65 Ω $\underline{+}$ 15 Ω is strongly recommended; otherwise, signal integrity requirements may be violated.

4.4.6.7 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.

[37] Any clamp diodes on A.G.P. signal pins must only connect to the Vddq rail and not to any other component power supply rails.

4.5 4X Mode Specifications

This section will be added in the final version of the A.G.P. Interface Specification, Rev. 2.0 and will parallel Section 4.4 in content.

5. Mechanical Specification

5.1 Introduction

This chapter defines an A.G.P. 1X/2X connector and expansion card 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 in ATX chassis implementations and has been incorporated in the *NLX Motherboard Specification* (available at http://www.teleport.com/~nlx/). In the ATX case, the PCI connector that was previously utilized for graphics is now replaced by the A.G.P. connector which provides higher performance. The A.G.P. expansion card for ATX systems 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 the *PCI Local Bus Specification* for the ISA bracket and retainer). The bracket shall be supplied with the card so that the card can be easily installed in the system.

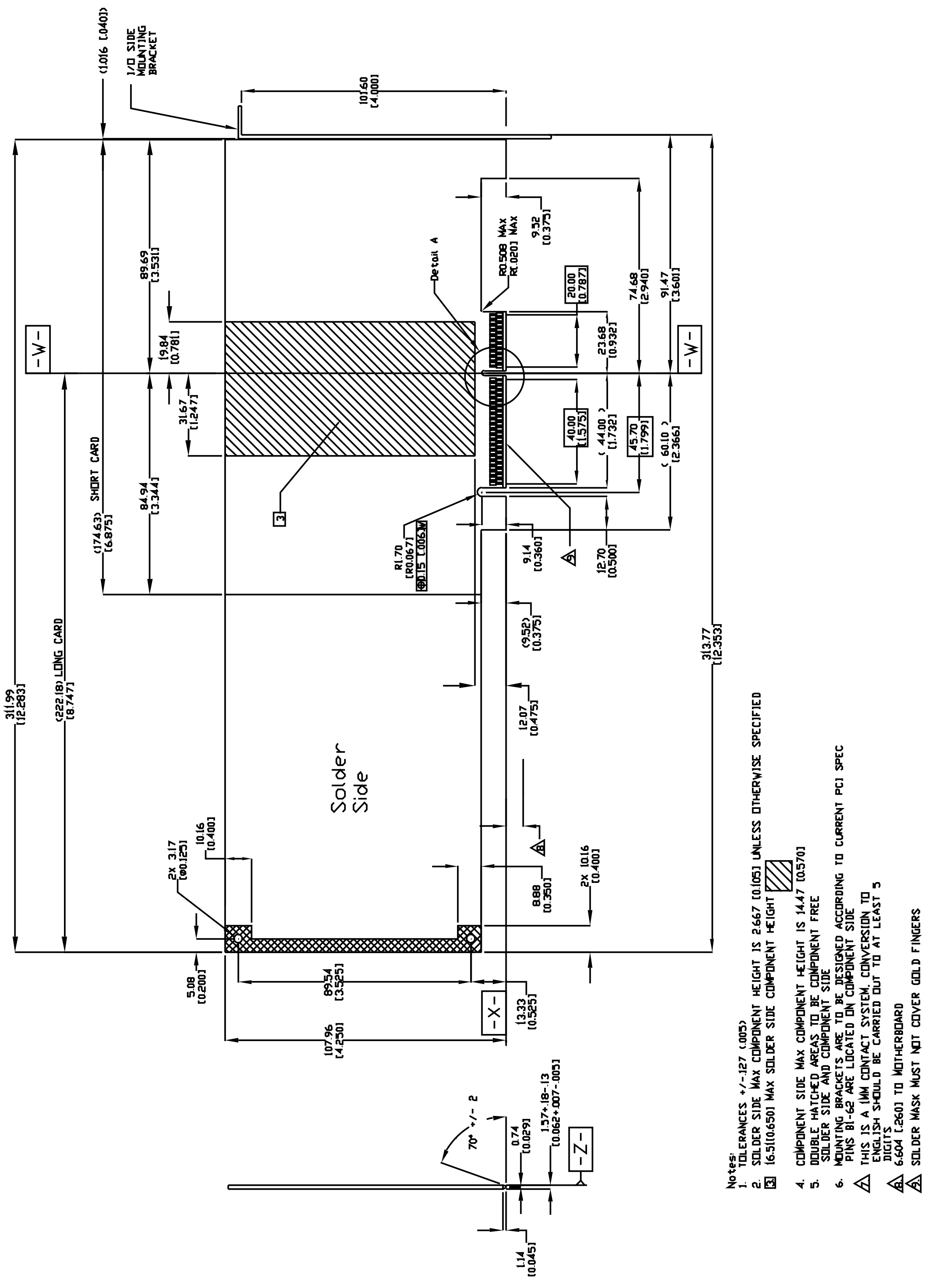
The A.G.P expansion card dimensions are also defined for the new NLX chassis. The NLX card dimensions are a subset of the ATX card dimensions. It is possible to design an NLX card that can be used in both NLX and ATX systems, but the mounting bracket must be changed.

Appendix C describes the high-end A.G.P 4X 110 W connector designed specifically for the server and workstation market segments. The connector provides the extra power necessary to support high-end A.G.P 4X graphics cards. This connector is a superset of the existing A.G.P connectors and supports A.G.P 1X, 2X, and 4X mode graphics cards with 1.5 volt or 3.3 volt signaling.

5.2 Expansion Card Description

5.2.1 Physical Dimensions and Tolerances

The A.G.P. card, like the PCI card, is designed to fit most existing ATX chassis. (The NLX version of the A.G.P card has reduced dimensions.) The maximum component height on the primary side of the A.G.P. ATX expansion card is not to exceed 14.47 mm (0.570 in.). The maximum component height on the backside of the card is not to exceed 2.667 mm (0.105 in.) unless otherwise specified. The maximum component height on the backside of the card has been increased in some areas to allow thermal enhancements (such as heatsinks or heatpipes) to be attached. Datum W on the illustrations is used to locate the A.G.P. card to the planar and to the chassis frame interfaces, which could include the back of the chassis frame and the card guide. Datum W is carried through the locating key on the card edge and the locating key on the connector. See Figures 5-1, 5-2, and 5-8 for A.G.P ATX form factor add-in card physical dimensions. See Figure 5-3 through Figure 5-8 for A.G.P NLX form factor add-in card physical dimensions.



Notes:
1. TOLERANCES +/-.127 (.005)
2. SOLDER SIDE MAX COMPONENT HEIGHT IS 2.667 [0.105] UNLESS OTHERWISE SPECIFIED
3. [6.51[0.650] MAX SOLDER SIDE COMPONENT HEIGHT ▨
4. COMPONENT SIDE MAX COMPONENT HEIGHT IS 14.47 [0.570]
5. DOUBLE HATCHED AREAS TO BE COMPONENT FREE SOLDER SIDE AND COMPONENT SIDE
6. MOUNTING BRACKETS ARE TO BE DESIGNED ACCORDING TO CURRENT PCI SPEC PINS B1-62 ARE LOCATED ON COMPONENT SIDE
7. THIS IS A 1MM CONTACT SYSTEM, CONVERSION TO ENGLISH SHOULD BE CARRIED OUT TO AT LEAST 5 DIGITS
8. 6.604 [.260] TO MOTHERBOARD
9. SOLDER MASK MUST NOT COVER GOLD FINGERS

Figure 5-1: A.G.P. ATX Form Factor Add-in Card

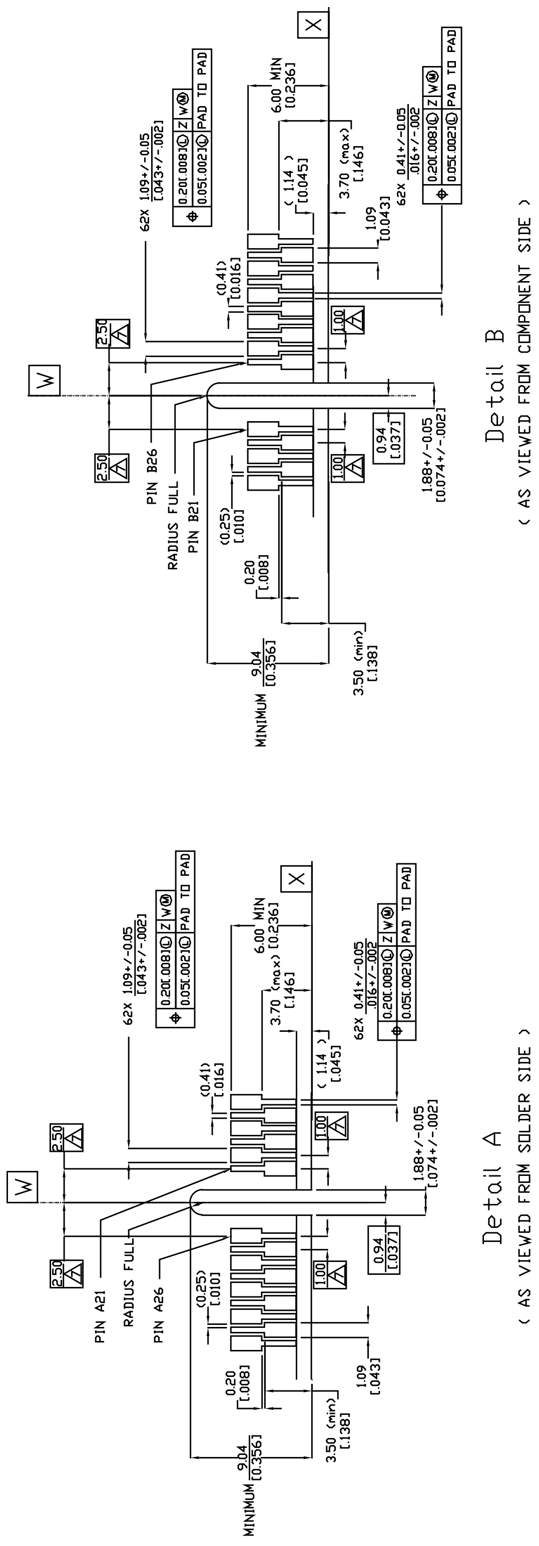


Figure 5-2: Detail A and B: A.G.P. Card Edge Finger Layout

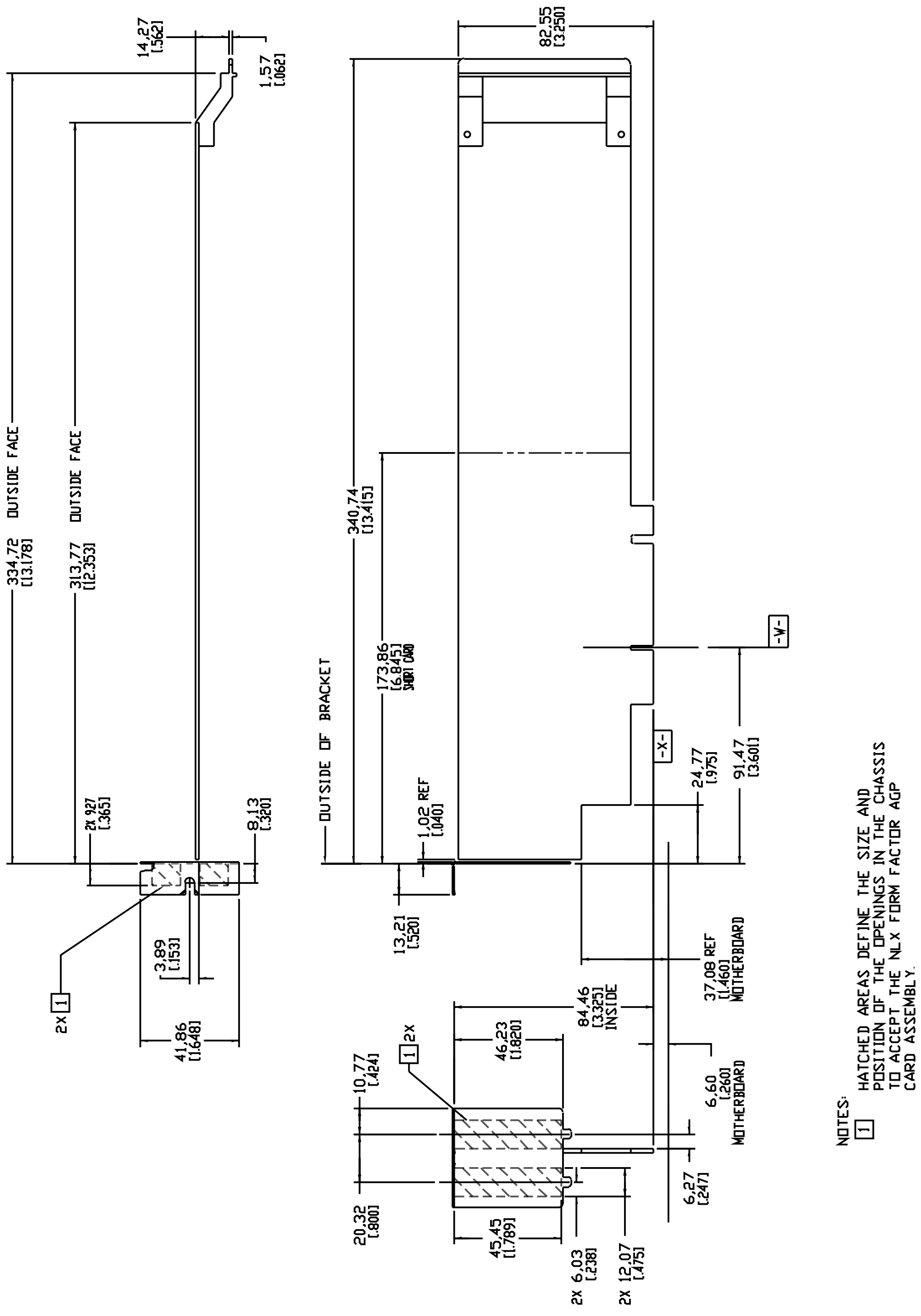


Figure 5-3: A.G.P. NLX Form Factor Add-in Card

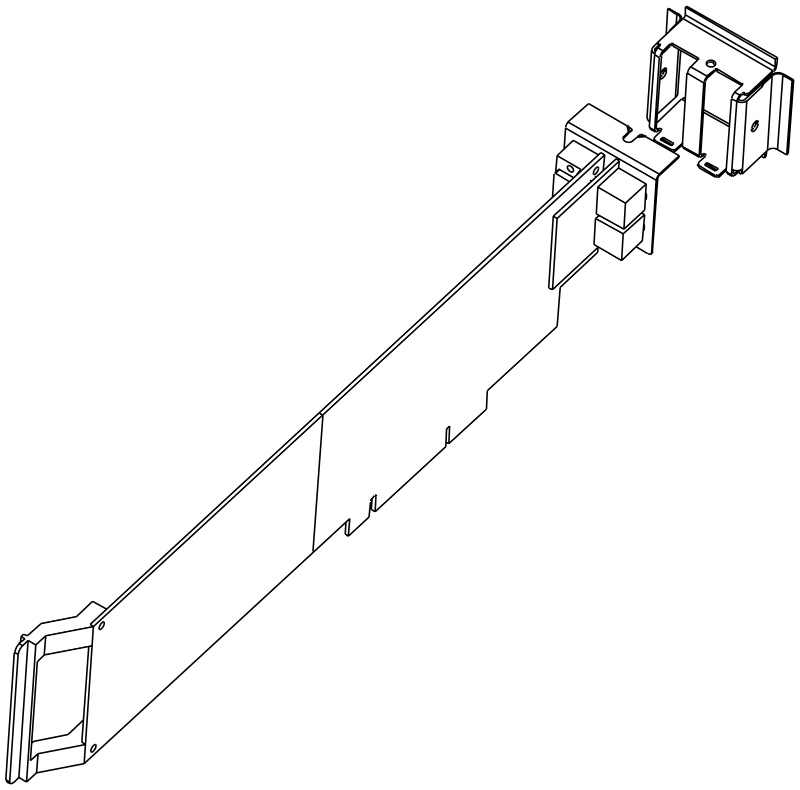
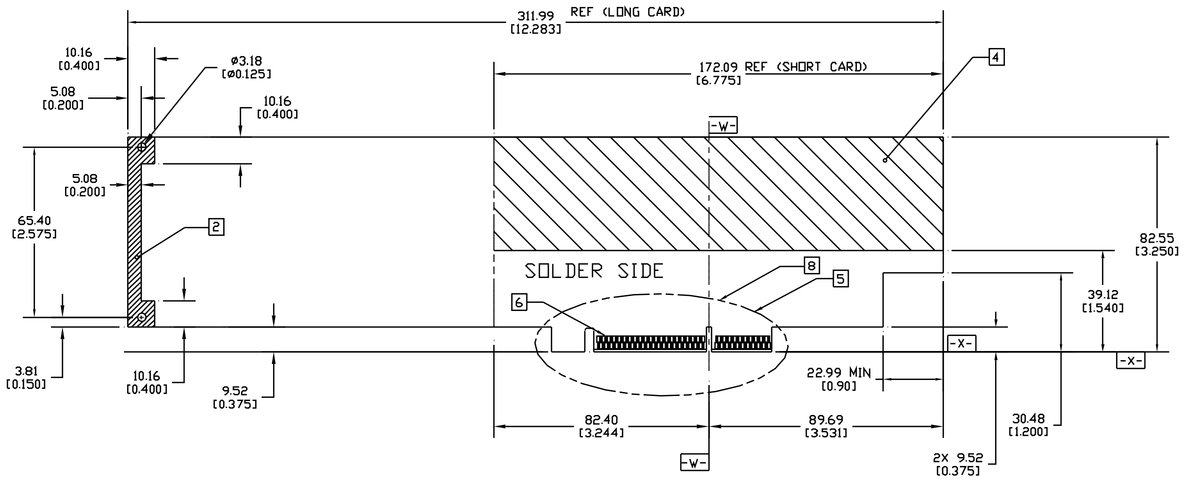
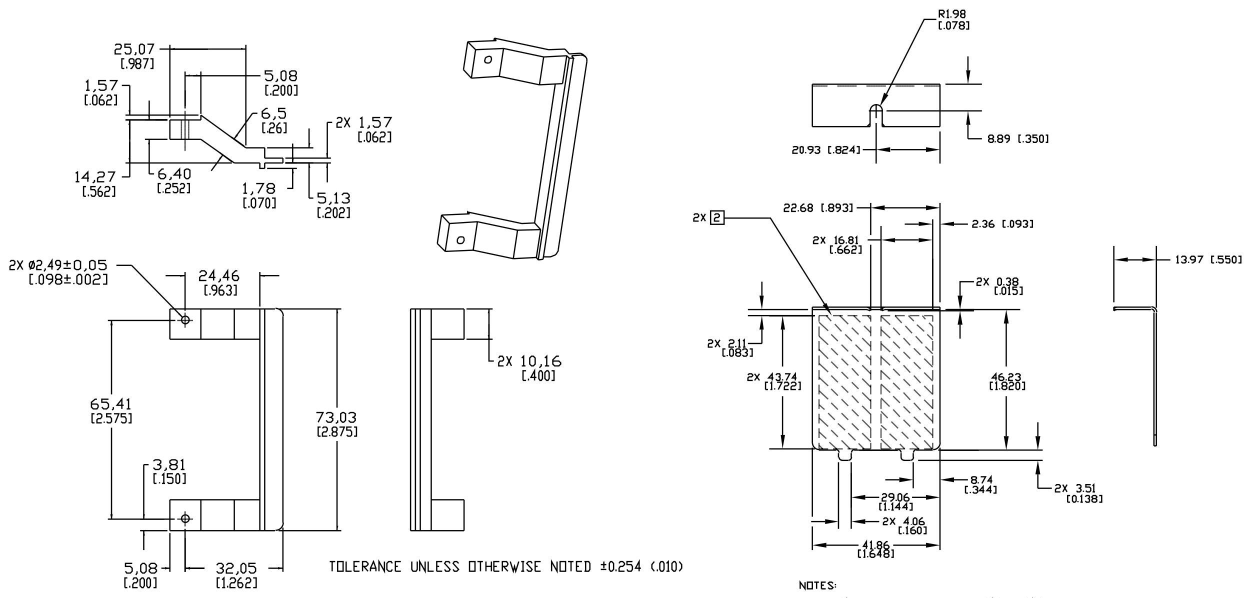


Figure 5-4: A.G.P. NLX Factor Add-in Card Comprehensive



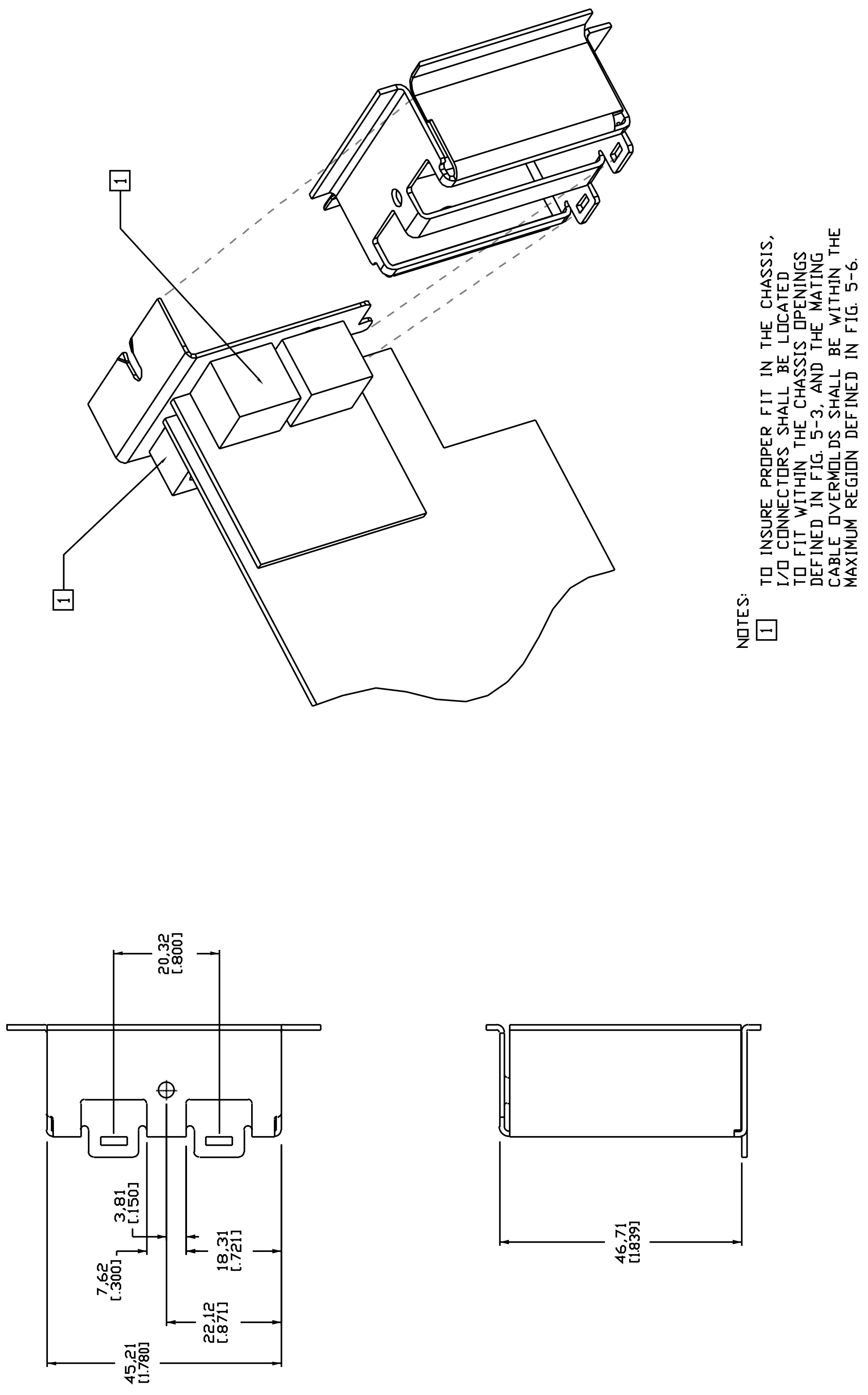
NOTES:
1. TOLERANCES +/-.127 [+/-.005]
2. THIS AREA TO BE COMPONENT FREE BOTH SIDES.
3. MAXIMUM ALLOWABLE HEIGHT ON SOLDER SIDE IS 2.667[.105] UNLESS OTHERWISE SPECIFIED.
4. SOLDER SIDE COMPONENTS IN THIS AREA RESTRICTED TO 22.35 [.880].
5. GOLD FINGERS AND TABS SAME AS DETAILED IN ATX AGP DESIGN.
6. SOLDER MASK MUST NOT COVER GOLD FINGERS BOTH SIDES
7. COMPONENT SIDE MAX COMPONENT HEIGHT IS 14.47 [.570]
8. THIS IS A 1 MM CONTACT SYSTEM, CONVERSION TO ENGLISH SHOULD BE CARRIED OUT TO AT LEAST 5 DIGITS

Figure 5-5: A.G.P. NLX Form Factor Card Detail



NOTES:
1. MATERIAL THICKNESS: 0.8MM ±0.1MM (.030 ±.004)
2. HATCHED AREAS DEFINE THE MAXIMUM SIZE AND LOCATION OF OVERMOLDS USED ON MATING CABLES. I/O CONNECTORS SHALL BE LOCATED ON THE BRACKET SUCH THAT THE MATING OVERMOLDS ARE WITHIN THESE REGIONS AND THE I/O CONNECTORS FIT THE CHASSIS OPENINGS DEFINED IN FIG. 5-3.

Figure 5-6: A.G.P. NLX Form Factor Add-in Card Reference Bracket Details



NOTES:

1. TO INSURE PROPER FIT IN THE CHASSIS, I/O CONNECTORS SHALL BE LOCATED TO FIT WITHIN THE CHASSIS OPENINGS DEFINED IN FIG. 5-3, AND THE MATING CABLE OVERMOLDS SHALL BE WITHIN THE MAXIMUM REGION DEFINED IN FIG. 5-6.

Figure 5-7: A.G.P. NLX Form Factor I/O Bracket and Chassis Interaction

5.2.2 Contact Design

A requirement of 100 microinches (min) tin lead over nickel underplate on solder tails, for all plating alternatives. The following two plating alternatives are listed to allow flexibility to connector suppliers. The absolute requirement is that the plating must meet the qualification requirements listed in Section 5.10.

Alternative 1: High strength copper alloy; 15 microinches (min) gold plating over 50 microinches minimum nickel plating.

Alternative 2: High strength copper alloy, 2 microinches (min) gold flash over 15 microinches palladium-nickel over 50 microinches (min) nickel underplate in critical contact areas.

5.3 Thermal Specification

A.G.P. add-in card designers are responsible for supplying a thermal solution for their add-in card that meets the requirements listed in Table 5-1. An add-in card targeted for an OEM market segment (and not as a generic add-in card) is exempt from meeting the requirements in Table 5-1. This exemption is granted because the OEM will define its own thermal requirements for the add-in card that may be more or less restrictive depending on the capabilities and features supported by the OEM.

For example, the designer of an add-in card must ensure that the card, which is targeted for a retail market segment, will operated under the thermal conditions shown in Table 5-1. The OEM that allows a generic add-in card to be inserted into its system must ensure that the specified thermal environment is maintained. Another example is where the included add-in card does not meet Table 5-1 but the OEM can accommodate the additional thermal load. However, a generic add-in card cannot rely on the OEM to manage a thermal load greater than what is specified in Table 5-1.

Please refer to the *A.G.P. Design Guide* for further details on thermal design for A.G.P. add-in cards.

Table 5-1: Thermal Specification for Add-in Cards

Symbol	Parameter	Condition	Min	Max	Units	Notes
Ta	Ambient Temperature	Within 0.5 inch of card PCB	0	55	°C	
Af	Air flow			0	m/s	1

Notes:
1. Natural convection airflow only with any card orientation (vertical, horizontal-components up, horizontal-components down) and air flow constrained by adjacent card and system modules.

5.4 A.G.P Add-in Card Connector Physical Description

5.4.1 Add-in Card Edge Dimensions

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 (Figure 5-8). Connector key width dimension of 1.78 mm (.070 in.) is measured prior to draft.

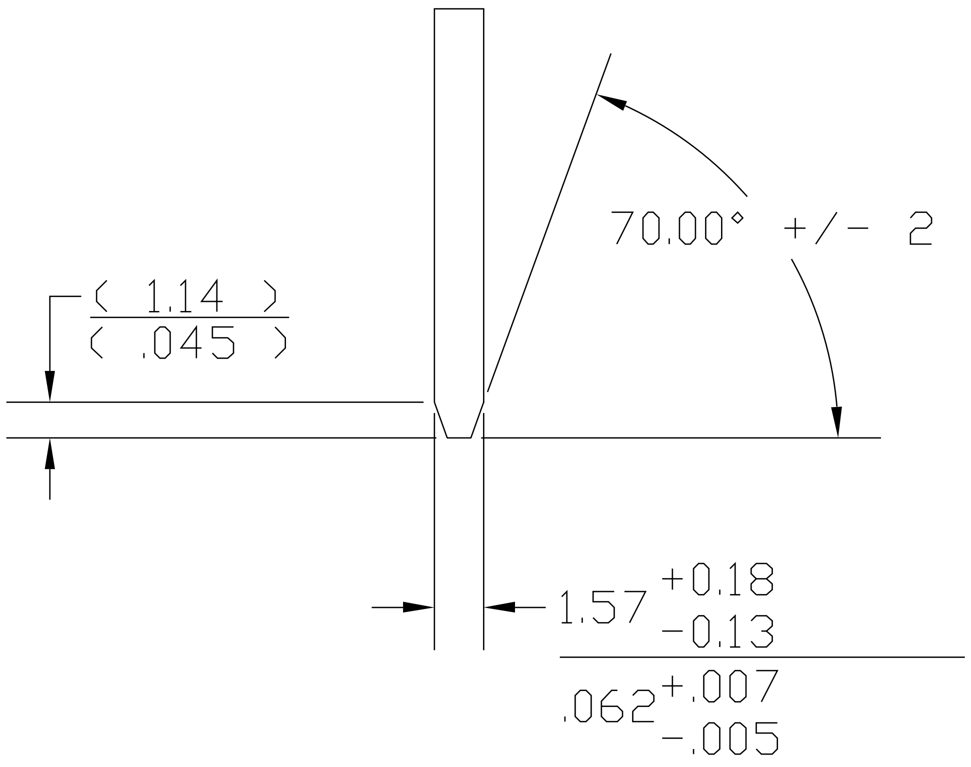


Figure 5-8: A.G.P. Card Edge Connector Bevel

5.4.2 Insertion/Extraction Force

Insertion force of PCB into A.G.P connector: 12.8 lbf (max)

Extraction force of PCB from A.G.P connector: 7.7 lbf (min)

Connector shall withstand a minimum of 50 insertion/extraction cycles with an A.G.P add-in card.

5.4.3 Assembly Requirements to Motherboard

5.4.3.1 Pre-Solder Attachment

A method of fastening the connector to the motherboard is required to assist in the manufacturing process. The fastening method shall be consistent with low-cost, high volume printed circuit board assembly lines. The recommended approach is to use three snap-in clips, that require a total insertion force of 3 lb to 15 lb to install the A.G.P add-in connector into the motherboard. The A.G.P Universal Connector has two snap-in clips. The clip length can be longer that the solder tail pins for the purposes of alignment, but must not interfere with or misdirect the alignment and penetration of the solder tails into the motherboard during the assembly process.

5.4.3.2 Solder Tail Design and Alignment:

The connector solder tails must be designed and aligned such that the end of the solder tails must enter a virtual condition hole that is 0.48 mm (.019 in.) in diameter. The virtual condition solder tail holes result from the true positional tolerance dimensions specified in the motherboard layout shown in Figures 5-9B, 5-10B, and 5-11B. The solder tails and retention clips have a 0.25 mm (.010 in.) perpendicularity requirement with respect to Datum V, see Figures 5-9A, 5-10A, and 5-11A.

5.4.3.3 Contact Backout Wipe

The minimum contact backout wipe within the connector for the upper and lower contacts on the gold finger pads, as shown in Figure 5-2, is .99 mm (0.039 in.).

5.4.4 A.G.P Add-in Card 3.3 Volt Connector

This section provides a physical description of the 124 pin 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 5-9 for connector dimensions and layout recommendations.

The connector specification detail and connector supplier information is available from the A.G.P. Implementer's Forum homepage (http://www.agpforum.org).

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.

Note: All 3.3 volt cards leave **TYPEDET#** open.

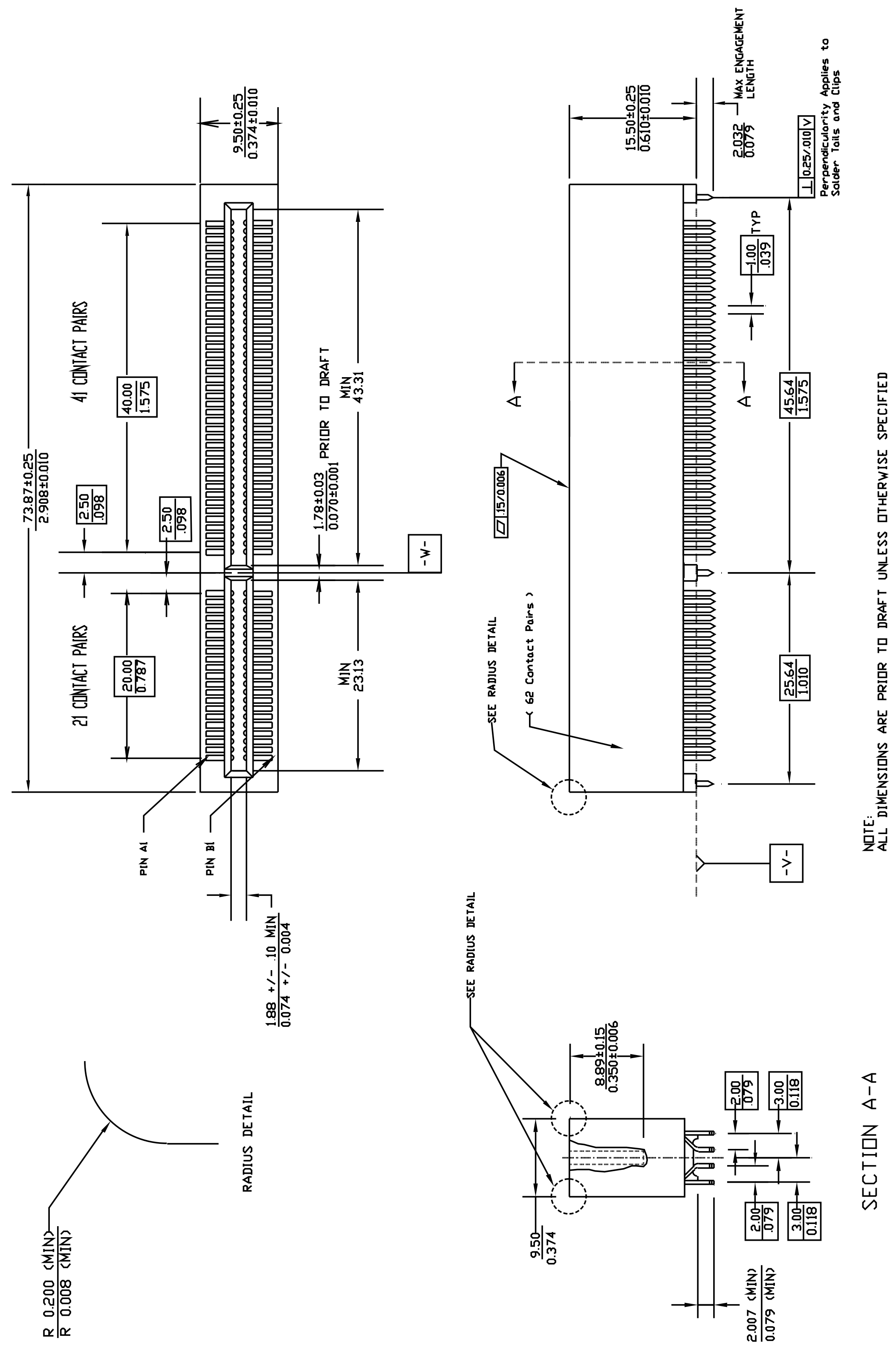


Figure 5-9A: A.G.P. 3.3 Volt Connector Footprint and Layout Dimensions

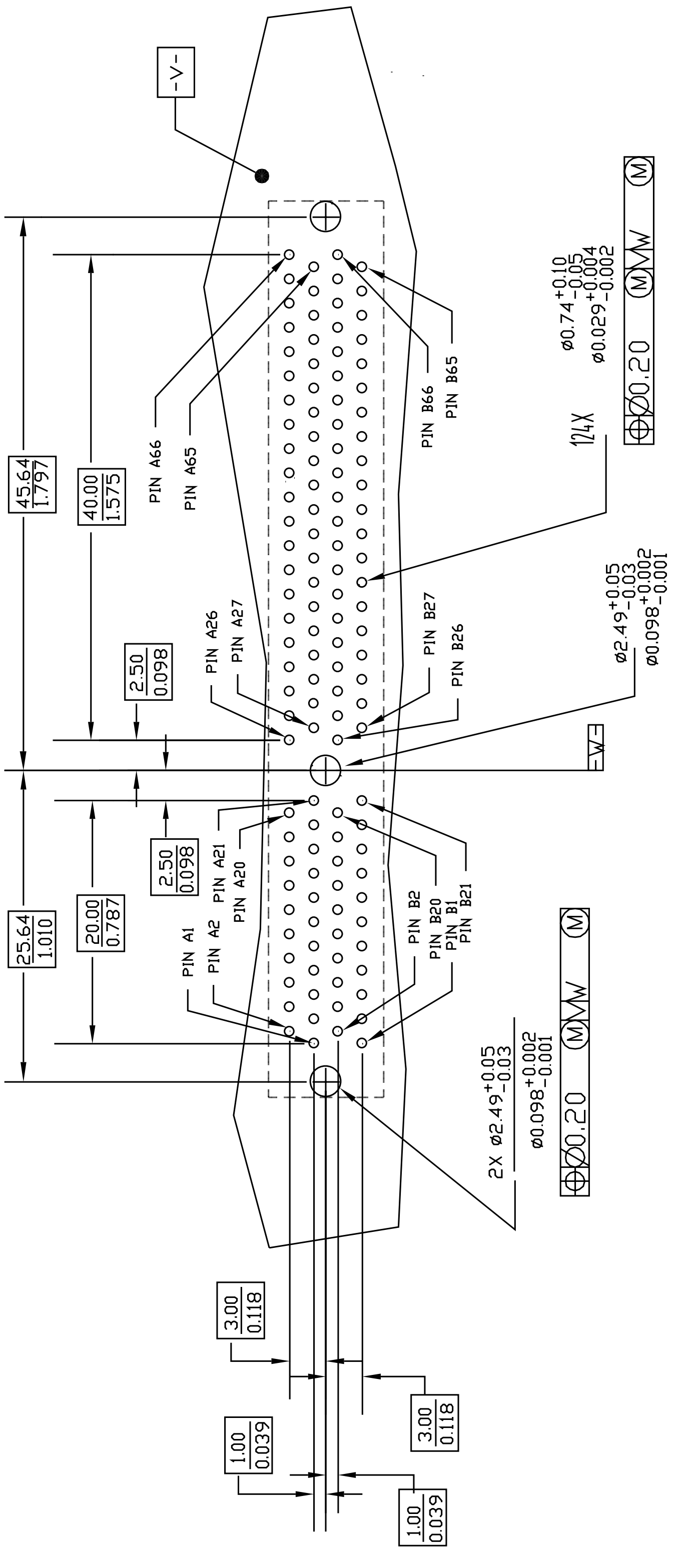


Figure 5-9B: A.G.P. 3.3 Volt Connector Footprint and Layout Dimensions

5.4.5 A.G.P Add-in Card Universal Connector

The A.G.P. 132 pin symmetrical universal connector, see Figure 5-10, provides for the transition from A.G.P. 3.3 volt signaling connector to A.G.P. 1.5 volt signaling connector. The Universal connector will accommodate 3.3 volt signaling for A.G.P. or 1.5 volt signaling for A.G.P. The motherboard or planar will need to detect and supply the correct voltage for the signaling interface based on **TYPEDET#**. For A.G.P. 3.3 volt signaling add-in cards, **TYPEDET#** is left open. On an A.G.P. 1.5 volt signaling add-in card, **TYPEDET#** is hardwired to ground. Note that the A.G.P. clock and reset will be driven at a 3.3 volt signaling level regardless of what I/O signaling levels (3.3 volt/1.5 volt) is selected by the add-in card. Components that are not 3.3 volt tolerant must divide the 3.3 volt A.G.P. clock and reset signaling levels down to avoid possible damage to the inputs of these devices.

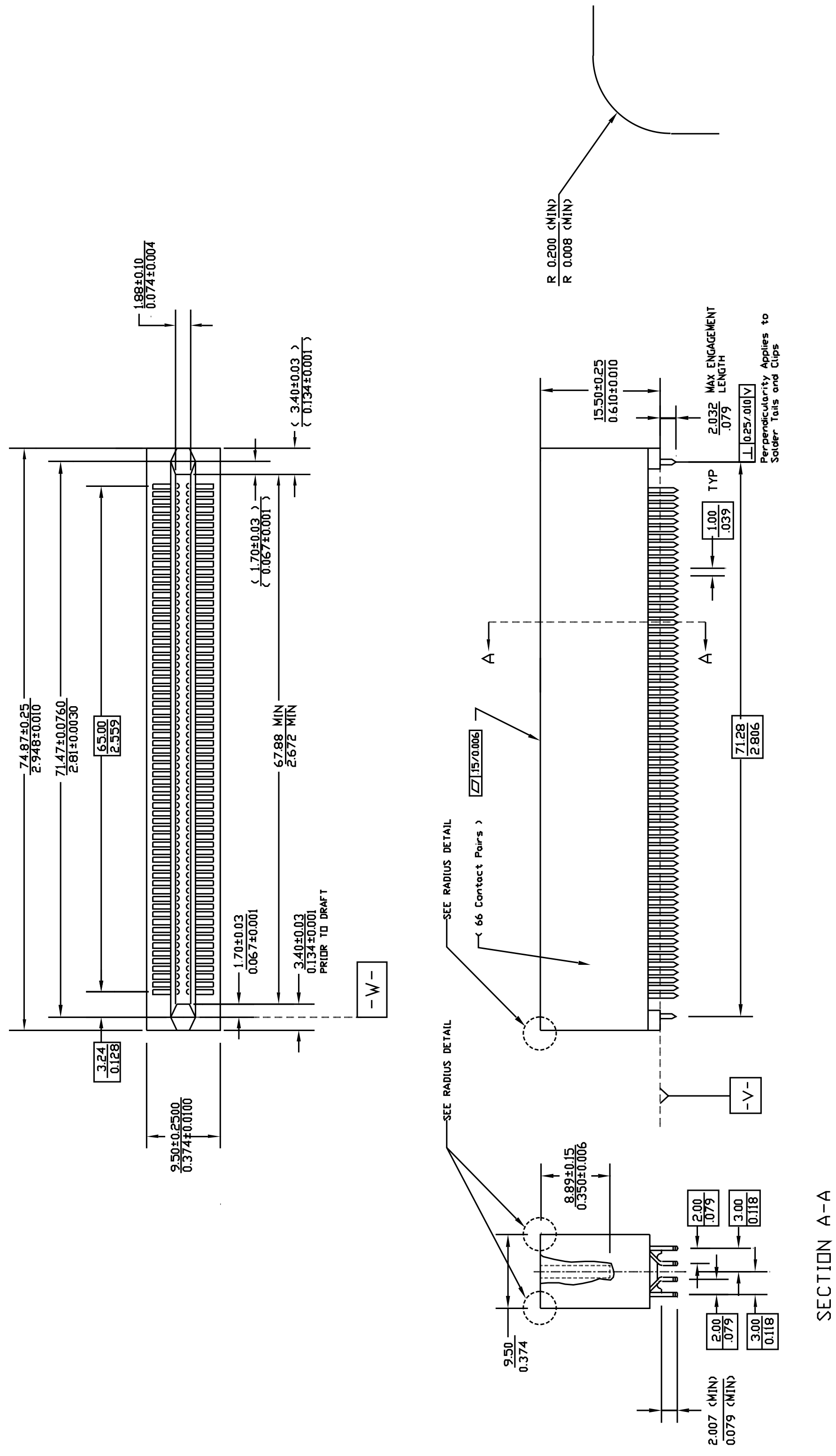


Figure 5-10A: A.G.P. Universal Connector Footprint and Layout Dimensions

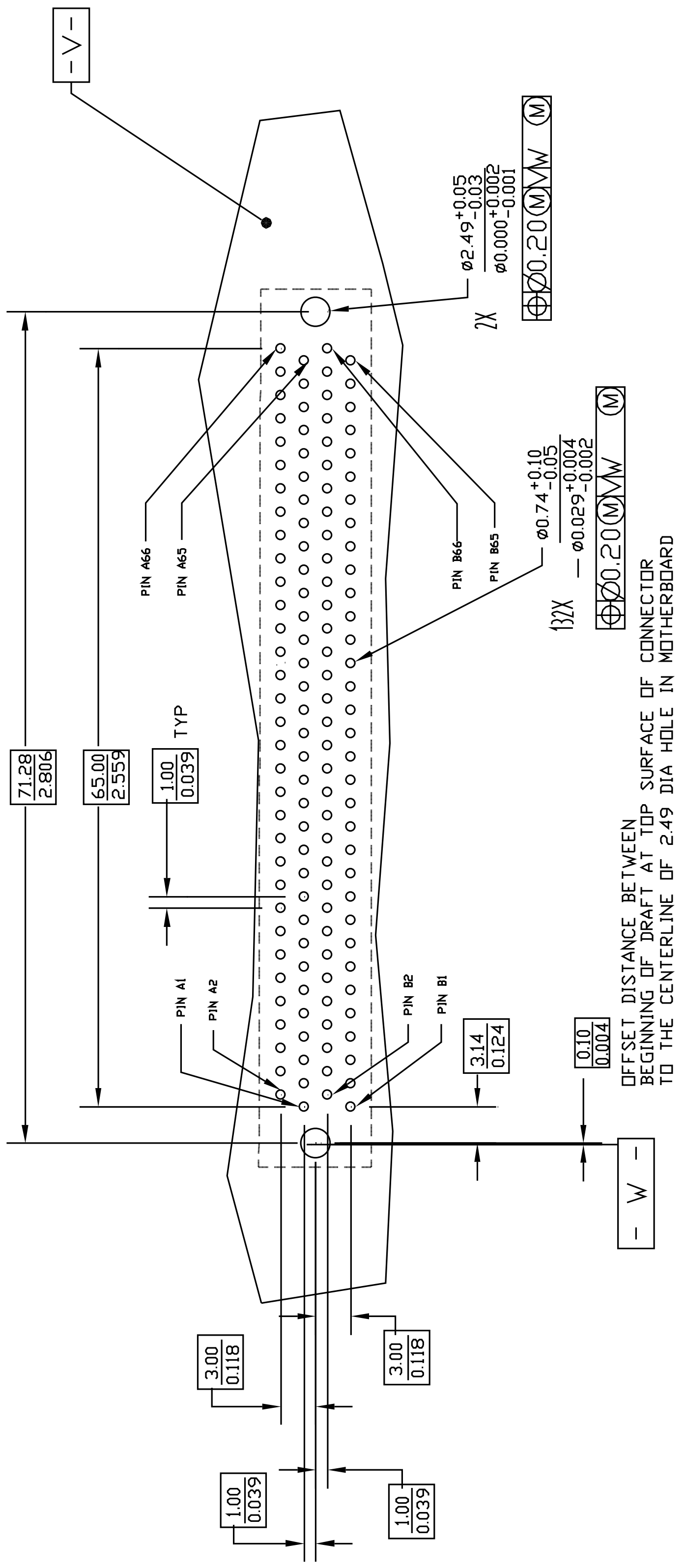


Figure 5-10B: A.G.P. Universal Connector Footprint and Layout Dimensions

5.4.6 A.G.P Add-in Card 1.5 Volt Connector

The 124 pin A.G.P. 1.5 volt signaling connector, see Figure 5-11, will be implemented with the A.G.P 3.3 volt connector by rotating the connector 180 degrees on the planar (motherboard). Therefore, the key of the connector will move to the opposite side of the planar away from the I/O panel and will not allow A.G.P. 3.3 volt cards to be inserted into the connector. **TYPEDET#** must be provided by 1.5 volt signaling add-in cards. Support for detecting **TYPEDET#** on motherboards containing the A.G.P. 1.5 volt signaling connector is optional since only 1.5 volt signaling add-in cards are supported. The signaling voltage is simply hard wired for 1.5 volt signaling levels in this case. Note that for the A.G.P. 1.5 volt connector, the A.G.P. clock and reset will be at 3.3 volt signaling levels. The same concerns apply here that apply for the universal connector in 1.5 volt signaling mode.

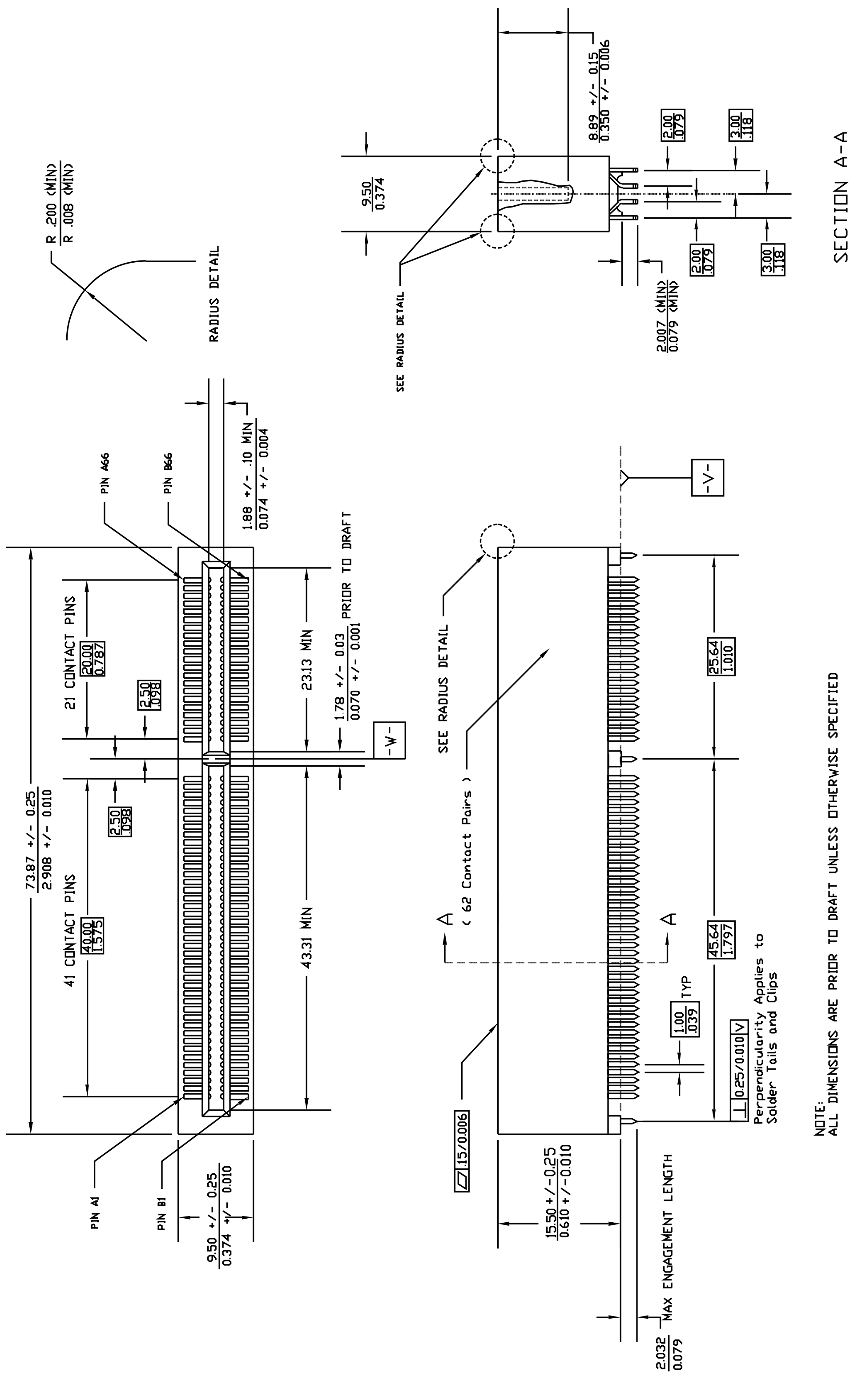


Figure 5-11A: A.G.P. Add-in Card 1.5 Volt Connector Footprint and Layout Dimensions

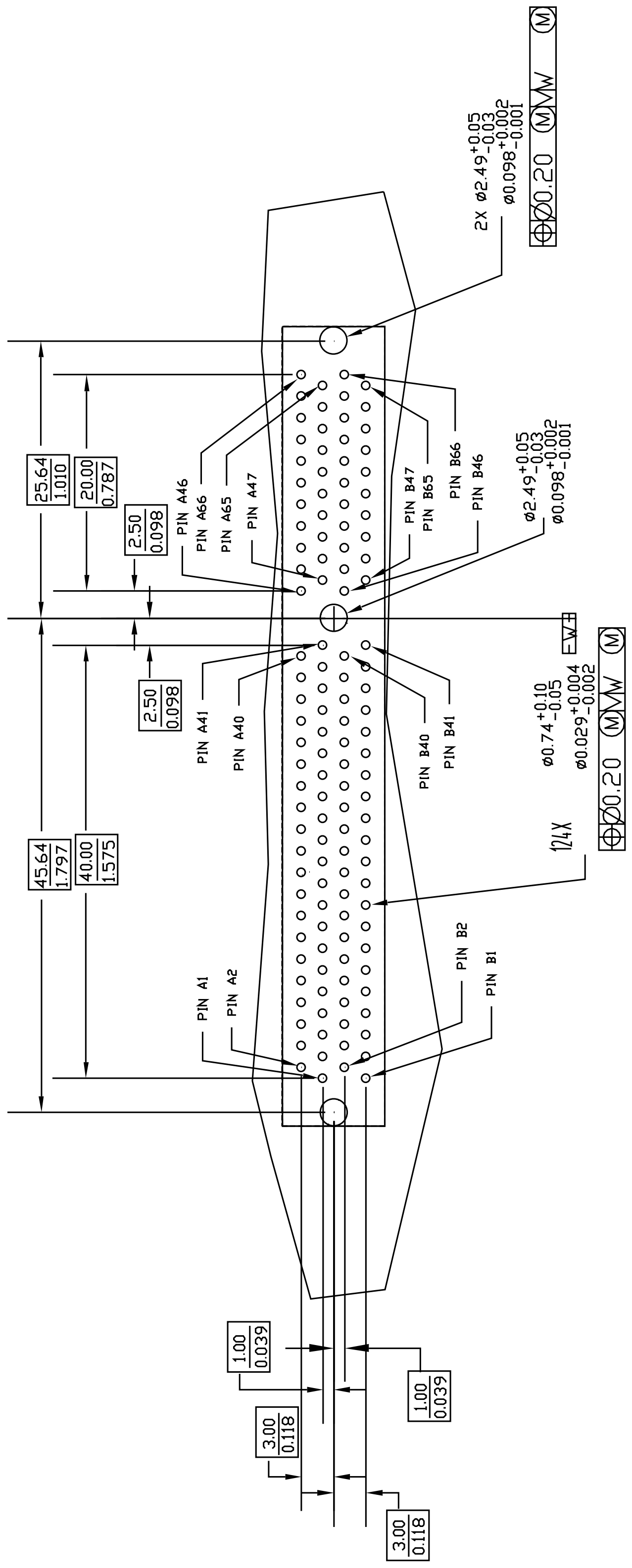


Figure 5-11B: A.G.P. Add-in Card 1.5 Volt Connector Footprint and Layout Dimensions

5.4.7 A.G.P Add-in Card Planar Implementation

5.4.7.1 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 5-12. 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.

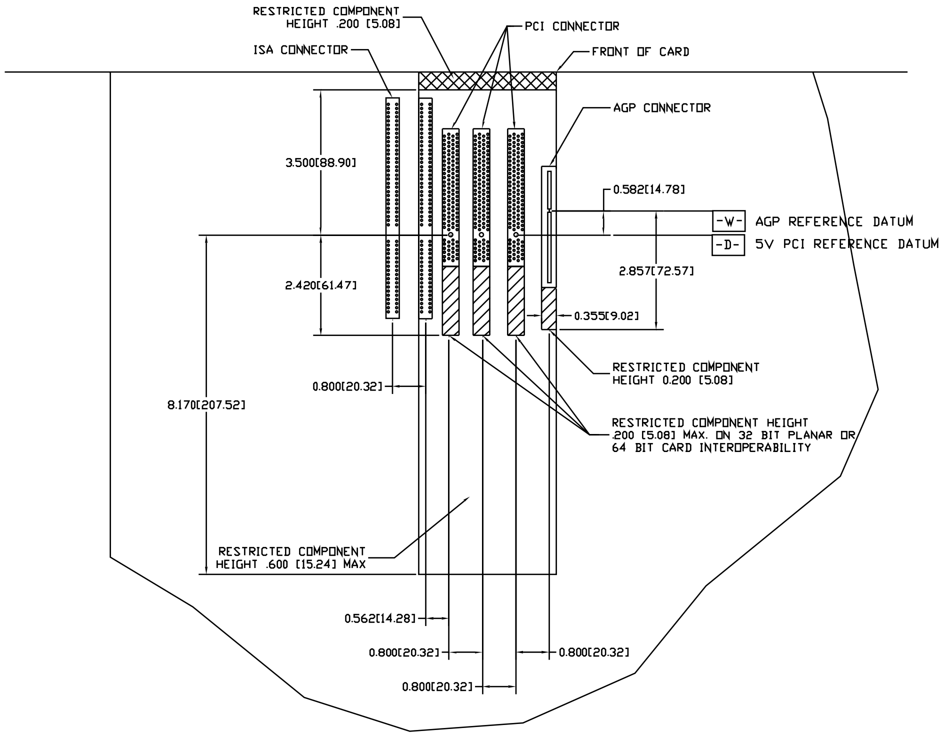

RESTRICTED COMPONENT HEIGHT .200 [5.08]
PCI CONNECTOR
ISA CONNECTOR
FRONT OF CARD
AGP CONNECTOR
3.500[88.90]
0.582[14.78]
-W- AGP REFERENCE DATUM
-D- 5V PCI REFERENCE DATUM
2.857[72.57]
2.420[61.47]
0.355[9.02]
RESTRICTED COMPONENT HEIGHT 0.200 [5.08]
0.800[20.32]
8.170[207.52]
RESTRICTED COMPONENT HEIGHT .200 [5.08] MAX. ON 32 BIT PLANAR OR 64 BIT CARD INTEROPERABILITY
RESTRICTED COMPONENT HEIGHT .600 [15.24] MAX
0.562[14.28]
0.800[20.32]
0.800[20.32]
0.800[20.32]

Figure 5-12 Typical ATX 3.3 Volt Connector Implementation

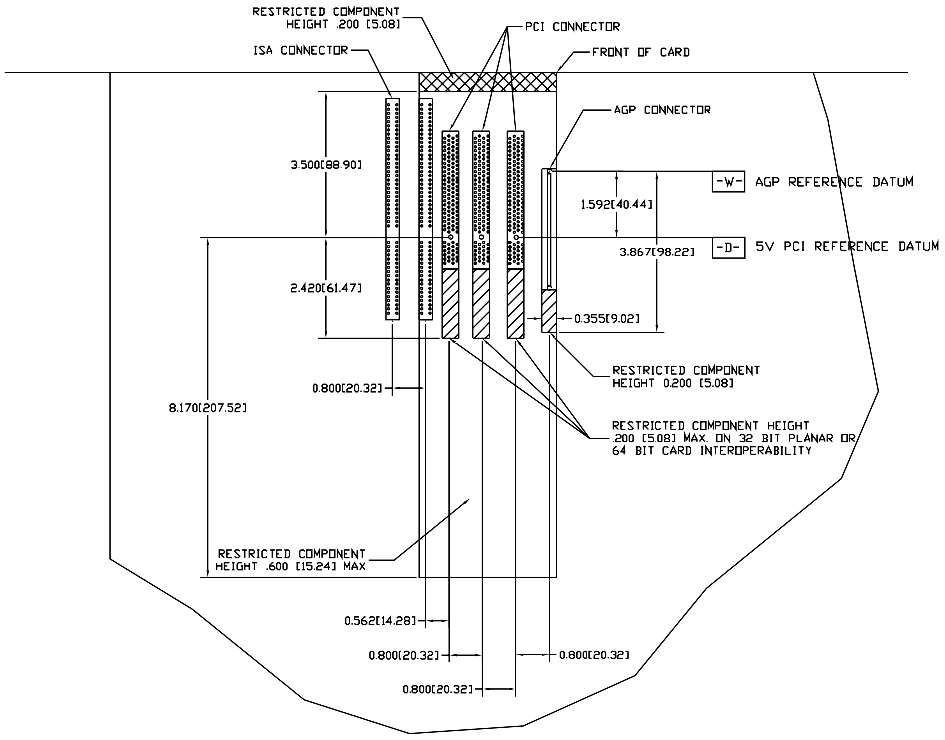

RESTRICTED COMPONENT HEIGHT .200 [5.08]
PCI CONNECTOR
ISA CONNECTOR
FRONT OF CARD
AGP CONNECTOR
3.500[88.90]
-W- AGP REFERENCE DATUM
1.592[40.44]
-D- 5V PCI REFERENCE DATUM
3.867[98.22]
2.420[61.47]
0.355[9.02]
RESTRICTED COMPONENT HEIGHT 0.200 [5.08]
0.800[20.32]
8.170[207.52]
RESTRICTED COMPONENT HEIGHT .200 [5.08] MAX. ON 32 BIT PLANAR OR 64 BIT CARD INTEROPERABILITY
RESTRICTED COMPONENT HEIGHT .600 [15.24] MAX
0.562[14.28]
0.800[20.32]
0.800[20.32]
0.800[20.32]

Figure 5-13: Typical ATX Universal Connector Implementation

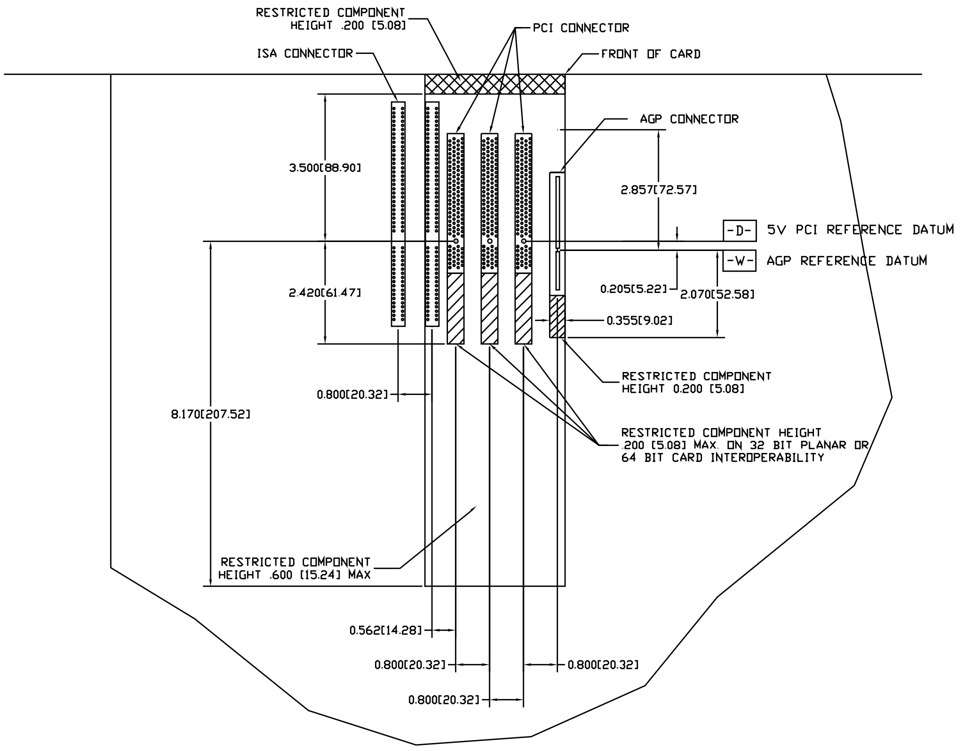

RESTRICTED COMPONENT HEIGHT .200 [5.08]
PCI CONNECTOR
ISA CONNECTOR
FRONT OF CARD
AGP CONNECTOR
3.500[88.90]
2.857[72.57]
-D- 5V PCI REFERENCE DATUM
-W- AGP REFERENCE DATUM
2.420[61.47]
0.205[5.22]
2.070[52.58]
0.355[9.02]
RESTRICTED COMPONENT HEIGHT 0.200 [5.08]
0.800[20.32]
8.170[207.52]
RESTRICTED COMPONENT HEIGHT .200 [5.08] MAX. ON 32 BIT PLANAR OR 64 BIT CARD INTEROPERABILITY
RESTRICTED COMPONENT HEIGHT .600 [15.24] MAX
0.562[14.28]
0.800[20.32]
0.800[20.32]
0.800[20.32]

Figure 5-14: Typical ATX 1.5 Volt Connector Implementation

5.4.7.2 NLX Planar Implementation

An A.G.P connector may be implemented on NLX planars.

5.5 Connector Pinout

Table 5-2: A.G.P. Motherboard Connector Pinout

	3.3 Volt		Universal		1.5 Volt	
Pin#	**B**	**A**	**B**	**A**	**B**	**A**
1	OVRCNT#	12V	OVRCNT#	12V	OVRCNT#	12V
2	5.0V	TYPEDET#	5.0V	TYPEDET#	5.0V	TYPEDET#
3	5.0V	Reserved	5.0V	Reserved	5.0V	Reserved
4	USB+	USB-	USB+	USB-	USB+	USB-
5	GND	GND	GND	GND	GND	GND
6	INTB#	INTA#	INTB#	INTA#	INTB#	INTA#
7	CLK	RST#	CLK	RST#	CLK	RST#
8	REQ#	GNT#	REQ#	GNT#	REQ#	GNT#
9	VCC3.3	VCC3.3	VCC3.3	VCC3.3	VCC3.3	VCC3.3
10	ST0	ST1	ST0	ST1	ST0	ST1
11	ST2	Reserved	ST2	Reserved	ST2	Reserved
12	RBF#	PIPE#	RBF#	PIPE#	RBF#	PIPE#
13	GND	GND	GND	GND	GND	GND
14	Reserved	Reserved	Reserved	WBF#	Reserved	WBF#
15	SBA0	SBA1	SBA0	SBA1	SBA0	SBA1
16	VCC3.3	VCC3.3	VCC3.3	VCC3.3	VCC3.3	VCC3.3
17	SBA2	SBA3	SBA2	SBA3	SBA2	SBA3
18	SB_STB	Reserved	SB_STB	SB_STB#	SB_STB	SB_STB#
19	GND	GND	GND	GND	GND	GND
20	SBA4	SBA5	SBA4	SBA5	SBA4	SBA5
21	SBA6	SBA7	SBA6	SBA7	SBA6	SBA7
22	KEY	KEY	Reserved	Reserved	Reserved	Reserved
23	KEY	KEY	GND	GND	GND	GND

Table 5-2: A.G.P. Motherboard Connector Pinout (continued)

	3.3 Volt		Universal		1.5 Volt	
Pin#	**B**	**A**	**B**	**A**	**B**	**A**
24	KEY	KEY	Reserved	Reserved	Reserved	Reserved
25	KEY	KEY	VCC3.3	VCC3.3	VCC3.3	VCC3.3
26	AD31	AD30	AD31	AD30	AD31	AD30
27	AD29	AD28	AD29	AD28	AD29	AD28
28	VCC3.3	VCC3.3	VCC3.3	VCC3.3	VCC3.3	VCC3.3
29	AD27	AD26	AD27	AD26	AD27	AD26
30	AD25	AD24	AD25	AD24	AD25	AD24
31	GND	GND	GND	GND	GND	GND
32	AD_STB1	Reserved	AD_STB1	AD_STB1#	AD_STB1	AD_STB1#
33	AD23	C/BE3#	AD23	C/BE3#	AD23	C/BE3#
34	Vddq3.3	Vddq3.3	Vddq	Vddq	Vddq1.5	Vddq1.5
35	AD21	AD22	AD21	AD22	AD21	AD22
36	AD19	AD20	AD19	AD20	AD19	AD20
37	GND	GND	GND	GND	GND	GND
38	AD17	AD18	AD17	AD18	AD17	AD18
39	C/BE2#	AD16	C/BE2#	AD16	C/BE2#	AD16
40	Vddq3.3	Vddq3.3	Vddq	Vddq	Vddq1.5	Vddq1.5
41	IRDY#	FRAME#	IRDY#	FRAME#	IRDY#	FRAME#
42	Reserved	Reserved	Reserved	Reserved	KEY	KEY
43	GND	GND	GND	GND	KEY	KEY
44	Reserved	Reserved	Reserved	Reserved	KEY	KEY
45	VCC3.3	VCC3.3	VCC3.3	VCC3.3	KEY	KEY
46	DEVSEL#	TRDY#	DEVSEL#	TRDY#	DEVSEL#	TRDY#
47	Vddq3.3	STOP#	Vddq	STOP#	Vddq1.5	STOP#
48	PERR#	PME#	PERR#	PME#	PERR#	PME#
49	GND	GND	GND	GND	GND	GND

Table 5-2: A.G.P. Motherboard Connector Pinout (continued)

	3.3 Volt		Universal		1.5 Volt	
Pin#	**B**	**A**	**B**	**A**	**B**	**A**
50	SERR#	PAR	SERR#	PAR	SERR#	PAR
51	C/BE1#	AD15	C/BE1#	AD15	C/BE1#	AD15
52	Vddq3.3	Vddq3.3	Vddq	Vddq	Vddq1.5	Vddq1.5
53	AD14	AD13	AD14	AD13	AD14	AD13
54	AD12	AD11	AD12	AD11	AD12	AD11
55	GND	GND	GND	GND	GND	GND
56	AD10	AD9	AD10	AD9	AD10	AD9
57	AD8	C/BE0#	AD8	C/BE0#	AD8	C/BE0#
58	Vddq3.3	Vddq3.3	Vddq	Vddq	Vddq1.5	Vddq1.5
59	AD_STB0	Reserved	AD_STB0	AD_STB0#	AD_STB0	AD_STB0#
60	AD7	AD6	AD7	AD6	AD7	AD6
61	GND	GND	GND	GND	GND	GND
62	AD5	AD4	AD5	AD4	AD5	AD4
63	AD3	AD2	AD3	AD2	AD3	AD2
64	Vddq3.3	Vddq3.3	Vddq	Vddq	Vddq1.5	Vddq1.5
65	AD1	AD0	AD1	AD0	AD1	AD0
66	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved

1. Reserved pins are only for future use by the A.G.P interface specification.
2. **IDSEL#** is not a pin on the A.G.P. connector. A.G.P. graphics components should connect the **AD16** signal to the 3. 3 volt **IDSEL#** function internal to the component.
3. All 3.3 volt cards leave the **TYPEDET** signal open. All 1.5 volt cards tie the signal hard to ground.

5.6 A.G.P Connector Electrical Requirements

Note: Information in this section particularly is still under investigation and development by Intel and is not complete. Therefore, this information is quite likely to evolve and change significantly before the release of the final Revision 2.0 A.G.P. Interface Specification.

5.6.1 Determination of Averaged Contact Resistance

- Measure Total Contact Resistance $\mathbf{R_{AB}}$ for each of the 124 contacts.
- Measure Bulk Contact Resistance $\mathbf{R}_{\mathbf{A}\mathrm{C}}$ for each of the 62 lower contacts.
- Determine Contact Resistance for each of the 124 contacts using either
  $\mathbf{R_{CB}} = \mathbf{R_{AB}}$ for upper contacts or
  $\mathbf{R_{CB}} = \mathbf{R_{AB}} - \mathbf{R_{AC}}$ for lower contacts.

5.6.1.1 Bulk Resistance

It is the resistance of the thin pad connecting a common bus and the lower contact pad on the mating substrate. Bulk Resistance is measured between point A on a common bus and point C at the junction of the thin pad and the contact pad as shown in Figure 5-15.

5.6.1.2 Initial Contact Resistance

10 mΩ (max) before testing for any power and ground contact as specified in Table 5-2.

5.6.1.3 Final Contact Resistance

5 mΩ (max) increase through testing, for a total of 15 mΩ (max) after testing, for any power and ground contact as specified in Table 5-2.

5.6.1.4 Test Voltage and Current Rating

Test to be performed at 1.0 A per contact. Voltage can vary to a maximum of 5.0 V during test to attain 1.0 A.

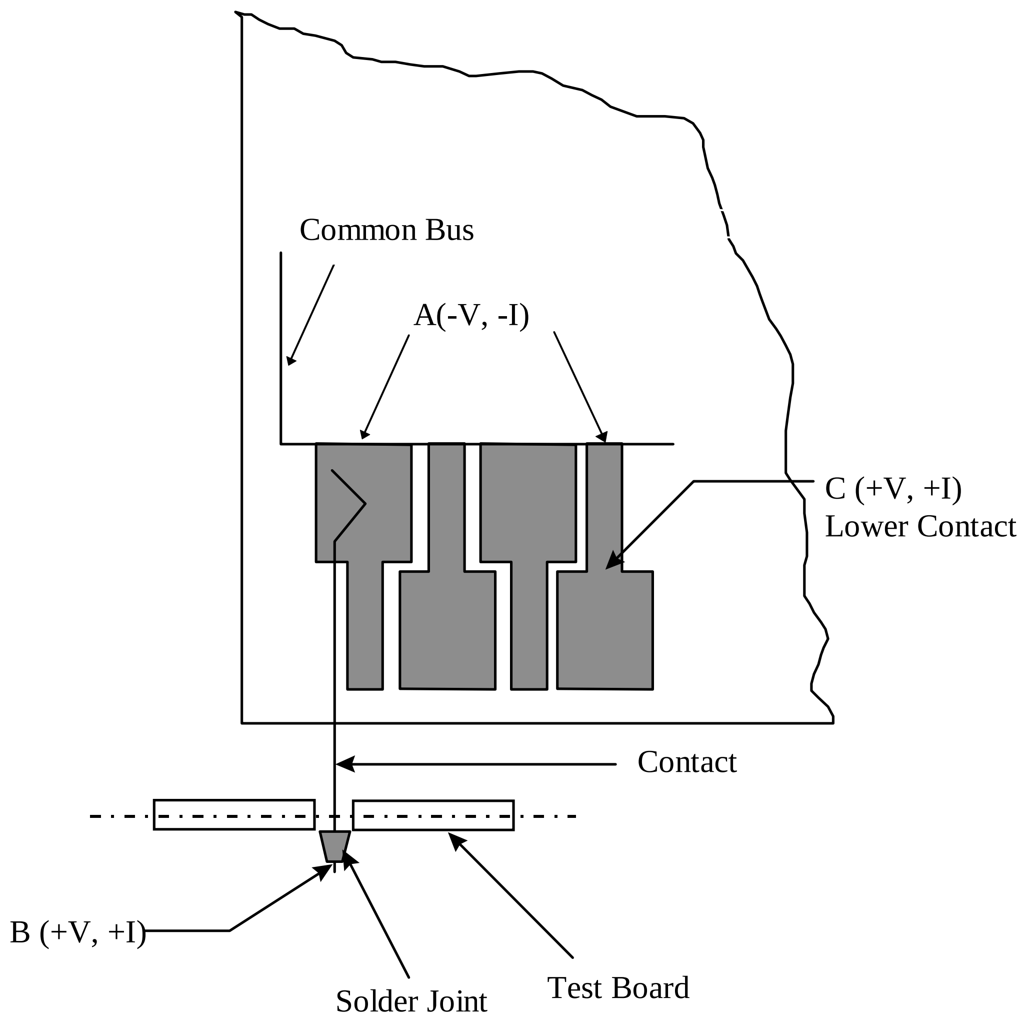


Figure 5-15: Bulk Resistance Measurement

5.6.2 Contact Current Rating

1.0 Amps / contact.

5.6.3 Effective Inductance

10.5 nH (max). Measured between two adjacent pins which have their solder tails shorted together (using a shorting bar or other mechanism), probed at a location within 0.25 mm (.010 in) from the top of the contact area.

5.6.4 Pin-to Pin-Capacitance

2 pF (max) at 1 MHz. Measured between two adjacent pins which are not shorted together, probed at a location within 0.25 mm (0.010 in) from the top of the contact area.

5.6.5 Pin-to-Pin Insulation Resistance

800 MΩ (min), as measured per EIA 364, Test Procedure 21.

5.6.6 Dielectric Withstand Voltage

400 VAC per MIL-STD-1344 Method D3001.1 Condition 1 or, EIA 364, Test Procedure 20.

5.6.7 Characteristic Impedance, Propagation, and Crosstalk Coupling

35<Zo<80 Ω, Delay < 150 ps, Crosstalk Coupling < 10%, using the TDR measurements described in Sections 5.6.7.1 and 5.6.7.2. The assumed rise time is 1.0 ns.

5.6.7.1 Connector Impedance, Propagation Delay, and Crosstalk Measurements

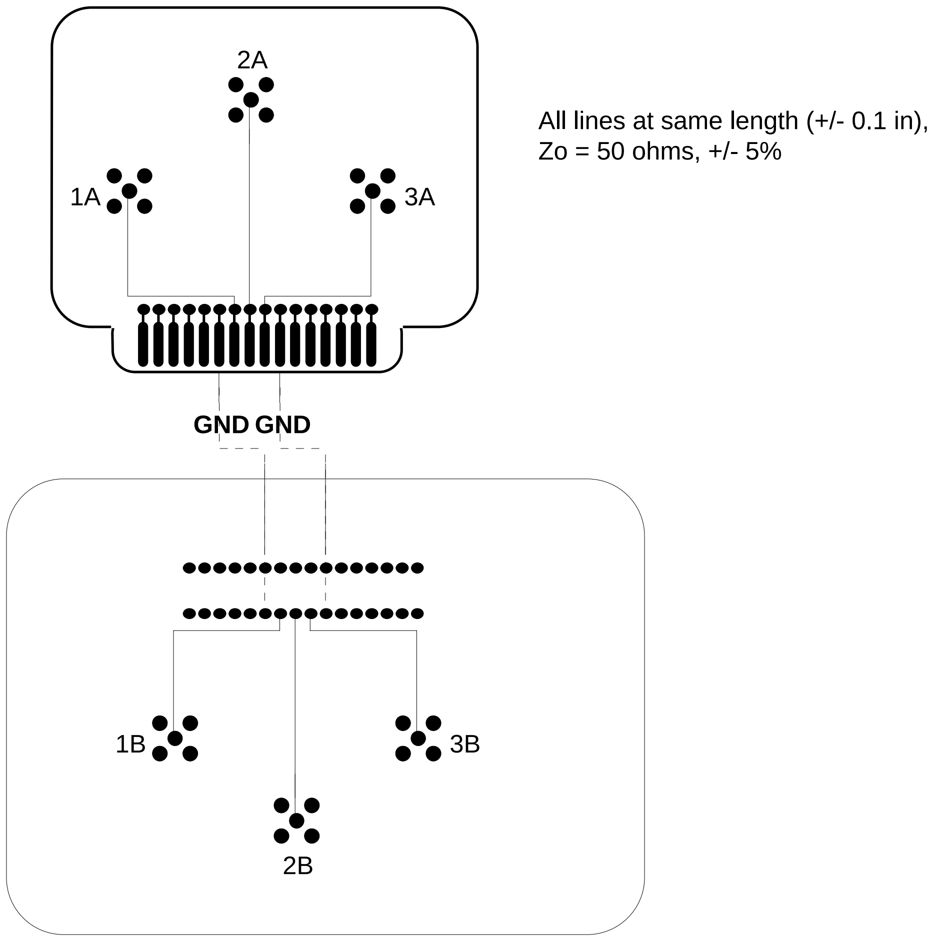


Figure 5-16. Test Structure for Impedance, Propagation Delay, and Crosstalk Measurements

5.6.7.2 Impedance and Propagation Delay

Measurement Conditions (see Section 5.6.7.1):

- TDR/TDT measurement (or equivalent technique)
- Signal:ground ratio = 3:1
- Test board Zo = 50 Ω ± 5%
- Connector delay measured from substrate via to motherboard via (substrate via to be within 0.25 mm (.010 in) of the top of the edge finger).
- Measurements made on each line while others floating

Zo: 35 <= Zo <= 80 Ω **Delay: Tpd < 150 ps**

5.6.7.3 Crosstalk

Measurement Conditions (see Section 5.6.7.1):

- TDR/TDT measurement (or equivalent technique)
- Signal driven into line 1, response measured on line 2

Magnitude:

- Coupled voltage <= 10 % of input voltage

5.7 A.G.P Connector Environmental Requirements

Design, including materials, shall be consistent with the manufacture of units which meet the following environmental standards.

5.7.1 Temperature Range

5.7.1.1 Operating

0 ºC to +85 ºC.

5.7.1.2 Shipping and Storage

-40 ºC to +105 ºC.

5.7.1.3 Temperature Life

The connectors shall withstand a minimum of 500 hours at the maximum operating temperature plus the maximum recommended temperature rise for the contact due to electrical heating. Normally, this is 85 ºC plus 20 ºC rise = 105 ºC.

5.7.2 Visual Inspection

Connector must meet mechanical requirements as specified in Section 5.4.

5.7.3 Vibration, Random

Tested per MIL-STD-1344A, Method 2005.1, Condition 5 or, EIA 364, Test Procedure 28, Test Condition 5, Letter B: 50 Hz to 2000 Hz, 3.1 G_{rms}, 45 min/axis, tested in each of three perpendicular axes. No electrical discontinuities > 1.0 µs. The connector must be mated with a mechanical sample outlined in Section 5.9 and be retained via the I/O bracket which is attached with a metal screw into the chassis.

5.7.4 Shock

Tested per MIL-STD-1344A, Method 2004.1, Test G, or EIA 364, Test Procedure 27: Test Condition A: 100 G, 6 ms duration, Sawtooth waveform. Three shocks applied in each of three perpendicular axes (18 total). No electrical discontinuities > 1.0 µs. The connector must be mated with the mechanical sample outlined in Section 5.9 and be retained via the I/O bracket which is attached with a metal screw into the chassis.

5.7.5 Durability

Mate and unmate samples for 49 cycles at a rate of 500 cycles per hour (max), using the same add-in card. On the 50th cycle use a new add-in card.

5.7.6 Mating Force

MIL-STD-1344, Method 2013.1 or, EIA 364, Test Procedure 13: Force necessary to mate steel gage to samples at a maximum rate of 0.5 in. per minute. 3.3 oz average initial insertion force per opposing pair of contacts (performed over a minimum of 10 opposing pair contacts) using steel gauge, per MIL-STD-C-21097, except 1.58 mm (0.062 in.) thick with a 70 degree chamfer angle (see Figure 5-8 for side view of substrate). Total force < 12.9 lbs, per Section 5.4.2.

5.7.7 Unmating Force

Force necessary to unmate substrate from samples at a maximum rate of 0.5 in. per minute. Test performed similar to 4.6, except steel gage is extracted. 1.98 oz average initial extraction force per opposing pair of contacts (performed over a minimum of 10 opposing pair contacts) using steel gauge, per MIL-STD-C-21097, except 1.58 mm (0.062 in.) thick with a 70 degree chamfer angle (see Figure 5-8 for side view of the substrate). Total extraction force < 7.6 lb., per Section 5.4.2.

5.7.8 Thermal Shock

-55 ºC to +85 ºC, 5 cycles per MIL-STD-1344A, Method 1003.1, Test Condition A or, EIA 364, Test Procedure 32: Test Group 4 (see Table 5-3) unmated and unmounted for this test. Test group 5 mated and mounted for this test.

5.7.9 Humidity Temperature Cycling

25 ºC to 65 ºC at 90 % to 95 % RH (non-condensing) for 10 days per MIL STD 202 Method 106 or EIA 364, Test Procedure 31, Method 3.

5.7.10 Temperature Life

Mated samples exposed to 105 ºC air temperature for 500 hr per EIA 364, Test Procedure 17 . See Section 5.7.1.3. Precondition samples with three insertion/extractions (min).

5.7.11 Mixed Flowing Gas:

Testing shall be performed in accordance with EIA 364, Test Procedure 65, Class 2A. The test is split into two parts: (1) Half of the samples (4) are exposed unmated to the mixed flowing gas for seven days, and then mated for the remaining seven days of the test. (2) The other half of the samples (4) are mated while exposed to the mixed flowing gas conditions for the entire 14 days. Precondition samples with three insertion and extractions (minimum) using a new substrate that has not gone through the mixed flowing gas test.

5.7.12 Withstand Temperature

Ramp temperature at rate of 1 ºC to 3 ºC/s to 145 ºC for two minutes, then ramp to 225 ºC for 40 s. Test per EIA 364, Test Procedure 56, Procedure 5.

5.7.13 Porosity

For plating alternative 2.1.3 (1) EIA 364, Test Procedure 53, Nitric Acid Test. For plating alternative 2.1.3 (2) EIA 364, Test Procedure 60, Procedure 1.1.2 Sulfur Dioxide Test.

5.7.14 Plating Thickness

Record thickness of plating on contact surface per EIA 364, Test Procedure 48, Method C.

5.7.15 Solvent Resistance

EIA 364, Test Procedure 11.

Requirement: No damage to ink markings if applicable.

5.7.16 Normal Force

EIA 364, Test Procedure 4.

Requirement: Calculate normal force using nominal thickness add-in card.

5.7.17 Solderability

EIA 364, Test Procedure 52, Class 2, Category 3.

Requirement: 95 % coverage.

5.7.18 Contact Retention

EIA 364, Test Procedure 29, 300 gm (min) load per individual contact.

Requirement: No movement > 0.38 mm (.015 in).

5.7.19 Maximum Force on Connector

No physical damage to connector after application of 150 lbf to fully seated add-in card. Time duration for the test is 30 seconds. Force applied to top surface of add-in card, perpendicular to the motherboard, after the add-in card is fully mated and bottomed out in the connector. No movement > 0.076 mm (.003 in.). Test performed within a 15 °C to 35 °C temperature range.

5.7.20 Contact Backout Wipe

There shall be no discontinuities or improper connections after withdrawing an add-in card 0.99 mm (.039 in.), after the add-in card is first bottomed out in the connector.

5.8 Safety Requirements

Design, including materials, shall be consistent with the manufacture of units which meet the following safety standards:

UL Recognition

CSA Certified

5.9 Add-in Card Mechanical Sample

A representative generic A.G.P. add-in card with a yet-to-be determined size and weight.

5.10 Connector Qualification

5.10.1 Sample Size Per Group

Test samples are to be taken from two different lots. For example, if eight samples are required, four samples will be used from two different lots.

5.10.2 Test Sequence

The numeric values in each column of Table 5-3 represents the order in which specific tests from the list are to be conducted. The following is an example of how the test sequence works: In Test Group 8, the first test is (1) visual inspection, followed by test (2) contact retention, followed by test (3) solvent resistance.

Table 5-3: Qualification Test

Test Description Sequence	**Test Group**								
(Reference Section Location)	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Visual Inspection (5.7.2)	1,9	1,5	1,5	1,8	1	1,5	1	1	1
Termination Resistance (5.6.1)	3,7	2,4	2,4		2,4,6				
Adjacent Pair Inductance (5.6.3)									2
Pin to Pin Capacitance (5.6.4)									3
Insulation Resistance (5.6.5)				2,6					
Dielectric Withstand Voltage (5.6.6)				3,7					
Characteristic Impedance, Propagation, and Crosstalk Coupling (5.6.7)									4
Vibration (5.7.3)	6								
Shock (5.7.4)	5								
Durability (5.7.5)	4								
Mating Force (5.7.6)	2								
Unmating Force (5.7.7)	8								
Thermal Shock (5.7.8)				4	3				
Humidity Temperature Cycling (5.7.9)				5	5				
Temperature Life (5.7.10)		3							
Mixed Flowing Gas (5.7.11)			3						
Resistance to Solder Heat (5.7.12)						2			
Porosity (5.7.13)							2		

Table 5-3: Qualification Test (continued)

Test Description Sequence	**Test Group**								
(Reference Section Location)	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Plating Thickness (5.7.14)							3		
Solvent Resistance (5.7.15)								3	
Normal Force (5.7.16)							4		
Solderability (5.7.17)							5		
Contact Retention (5.7.18)								2	
Maximum Force on Connector (5.7.19)						3			
Contact Backout Wipe (5.7.20)						4			
Sample Size per Test Group	8	8	8	8	8	4	4	4	4

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

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

1. Power-on self test (POST) code allocates resources to all devices in the system. (BIOS)
2. The operating system activates A.G.P. features. (not BIOS)
3. Microsoft's DirectDraw* carries out the final runtime memory management activity.

The first two of these operations are described in this chapter. Refer to Microsoft documentation for details on the third operation.

6.1 POST-time Initialization

Conventional bus enumeration software in the POST code identifies all system devices (including A.G.P. devices), creates a consistent system address map, and allocates system resources to each device. An A.G.P. 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 the *PCI Local Bus Specification* for more detail.) Supporting the PCI header allows conventional bus enumeration software to function correctly while being completely unaware of A.G.P. features.

6.1.1 A.G.P. Master Devices

A.G.P. 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 its own base address register. See the *PCI Local Bus Specification* for a description of PCI base address registers.

6.1.2 A.G.P. Target Devices

Figure 6-1 illustrates a typical host bus bridge (also known as corelogic) implementation that supports the A.G.P. interface along with other typical interfaces (or ports). Ports, other than the A.G.P. port, of the corelogic are included for illustrative purposes only and are not required by this interface specification. The example corelogic provides ports to the processor, system memory, the PCI bus, and to A.G.P. The shaded area represents the A.G.P. target. The blocks inside the dashed line represent different functions that the host bus bridge usually provides. Blocks that are not part of the A.G.P. target (shaded area) are included only for discussion purposes and are not required by this interface specification. The following paragraphs describe which accesses are typically supported at each port; whether an access is supported or not may depend on the destination of the request. For example, arrows 1, 2 and 3 are typically implemented by a host bus bridge that supports the PCI bus, but are not required if supporting an A.G.P. interface. Arrows 4 through 8 are associated with the A.G.P. port. The arrows in the figure describe paths in which transactions are routed inside the corelogic. Each port will be discussed and the associated paths (arrows) will be described.

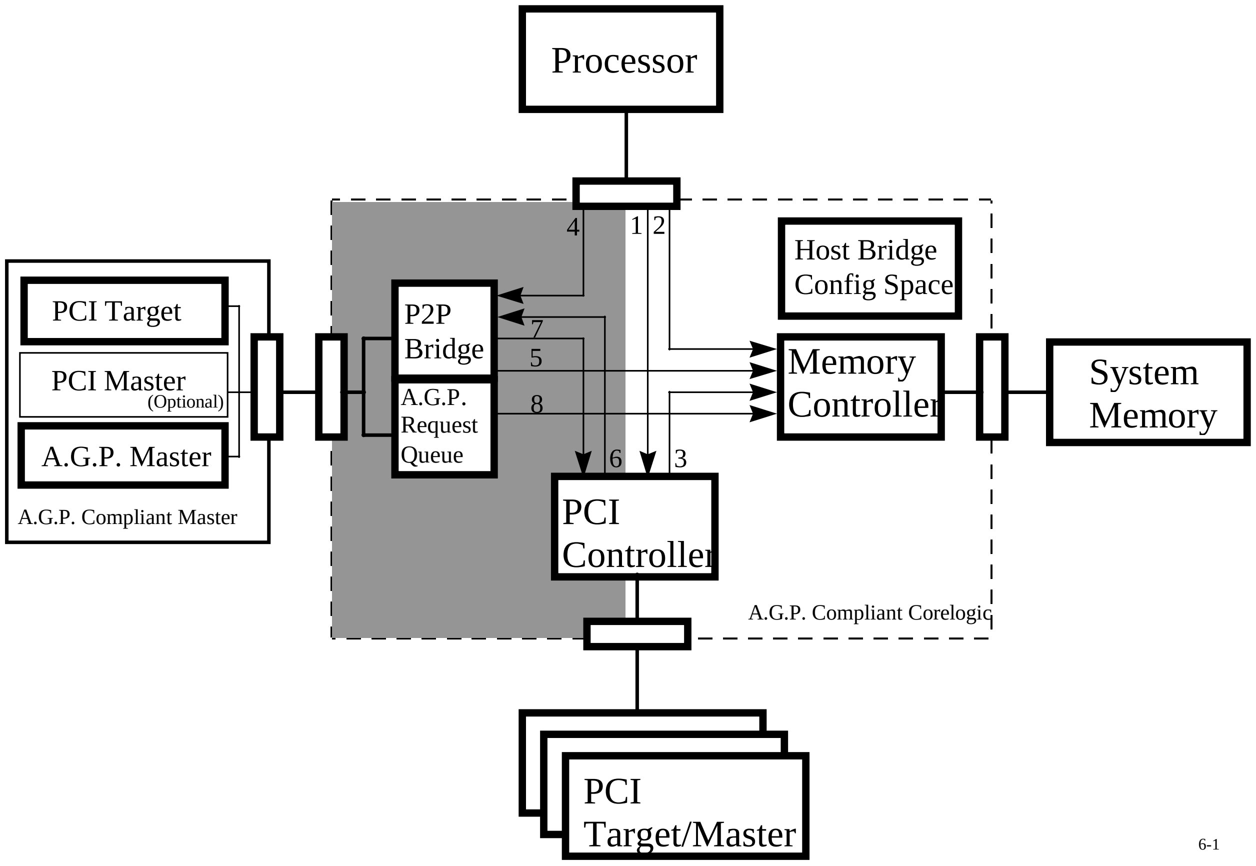


Figure 6-1: Configuration View of an A.G.P. Target

6.1.3 Corelogic Ports

6.1.3.1 Processor Port

The processor port is not required when supporting an A.G.P. interface, but is typically supported by a host bus bridge. The processor port provides a means for the processor to generate accesses to the PCI controller (path 1), to the P2P (PCI to PCI bridge) (path 4), and to the memory controller (path 2). The corelogic determines to which port these accesses are routed by using information stored in the Host Bus Bridge Config Space block. This information is provided during the initialization process.

6.1.3.2 System Memory Port

This port provides a means to connect system memory to the corelogic. The memory controller is responsible for converting accesses that are initiated on other ports (processor, A.G.P., and PCI) into memory commands.

6.1.3.3 PCI Port

The PCI controller converts processor accesses into PCI transactions. Since the processor has no Configuration commands, the PCI controller generates PCI Configuration commands as described in the *PCI Local Bus Specification*. The PCI controller takes memory commands that address system memory and forwards them to the Memory controller (path 3). The PCI controller also provides a means for PCI masters to access the PCI target that resides on the A.G.P. port and is represented by path 6 in the shaded area. This path is limited in its functionality and the corelogic is not required to provide full PCI to PCI bridge functionality. The corelogic provides support for PCI write[38] commands as described in the *PCI Local Bus Specification*, while support of other PCI commands is optional.

6.1.3.4 A.G.P. Port

When the corelogic supports an A.G.P. port, it requires new logic that has not been incorporated in previous chipsets, however no new functionality is required to boot the system. This new logic is shown in Figure 6-1 as a shaded area. To enable the use of existing enumeration code (unmodified) to handle A.G.P. devices the corelogic will use functionality already defined by the *PCI Local Bus Specification*. The P2P bridge block facilitates the configuration of the second I/O port (A.G.P.) of the corelogic using enumeration code and follows the *PCI to PCI Bridge Architecture Specification*. The P2P bridge makes it possible to configure the PCI target interface in an A.G.P. master device. This information is also used to route memory and I/O addresses to the PCI target of an A.G.P. master from the processor. The P2P bridge block is not required to be a fully functioning PCI to PCI bridge. The corelogic is only required to support PCI write commands from the PCI to the A.G.P. port. The corelogic may optionally support other PCI commands between PCI and A.G.P. or A.G.P. and PCI but there is no requirement. The processor can initiate transactions to a PCI target on the A.G.P. port by path 4. Path 7 optionally provides a path for the A.G.P. master using PCI protocol to access a target on PCI. The A.G.P. master can initiate PCI commands to the memory controller by path 5. Path 5 and path 3 have the same capabilities. The new functionality provided by the A.G.P. port is represented by the A.G.P. Request queue block. The request queue accepts A.G.P. commands from the A.G.P. master. Once A.G.P. commands are accepted by the request queue, it is implementation specific as to how these requests are presented to the memory controller (see path 8). The A.G.P. master issuing

[38] Memory Write and Memory Write and Invalidate commands.

A.G.P. commands can only access system memory. Support to any other port is not required or supported by this interface specification.

The A.G.P. master (solid line around the A.G.P. master, PCI master and PCI target blocks also identified as an A.G.P. master) is allowed to initiate any A.G.P. commands described in Section 3.3. The PCI master is allowed to initiate any PCI command specified in the *PCI Local Bus Specification*. Which commands the corelogic is required to support and which commands may be optionally supported are listed in Table 6-1. The manner in which the corelogic behaves when an unsupported command is used is not defined by this interface specification[39].

Table 6-1: Commands Supported by Each Port

Path	Typically Supported by Corelogic	Commands Required to be Supported by Corelogic	Commands Optionally Supported by Corelogic
1	Processor memory read and write, I/O read and write. The corelogic generates PCI Configuration Read and Write commands from Processor I/O read and write commands per the *PCI Local Bus Specification.*	None	Interrupt Acknowledge and CPU Special Cycles.
2	Processor memory read and write commands.	None	N/A
3	Memory Read, Memory Read Line, Memory Read Multiple, Memory Write, and Memory Write and Invalidate.	None	I/O (read and write) and Configuration (read and write).
4	Same as 1.	None	N/A
5	Same as 3.	None	N/A
6	N/A	Memory Write and Memory Write and Invalidate.	I/O (read and write), Configuration (read and write), and memory read (Read, Read Line, and Read Multiple).
7	N/A	None.	I/O (read and write), Configuration (read and write), memory read (Read, Read Line, and Read Multiple) and memory write (Write and Invalidate and Write).
8	N/A	A.G.P. commands.	N/A

The Host Bridge Config Space block contains configuration registers used to specify parameters associated with the GART and circuitry in the A.G.P. interface. The corelogic uses a PCI base address register to request a naturally

[39] The chipset can ignore the request and allow it to be terminated with Master-Abort or claim the access and return FFFF FFFFh on a read and drop write data by asserting **TRDY#**.

aligned block of memory address space in which to locate the GART address range. Initiation code determines the size requested and allocates the resource. The bridge also uses this information to route requests initiated by the processor to either the memory controller, PCI controller, or P2P bridge.

6.1.4 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 code selects one VGA device by first searching the ISA bus, then PCI add-in card slots *(includes A.G.P. connector)*, and then motherboard devices *(includes motherboard A.G.P. devices).* Note: boot-time bus enumeration software is unaware of A.G.P. features.

6.1.5 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. target's address remapping hardware.
3. Set the A.G.P. 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 in the *PCI Local Bus Specification*. 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.

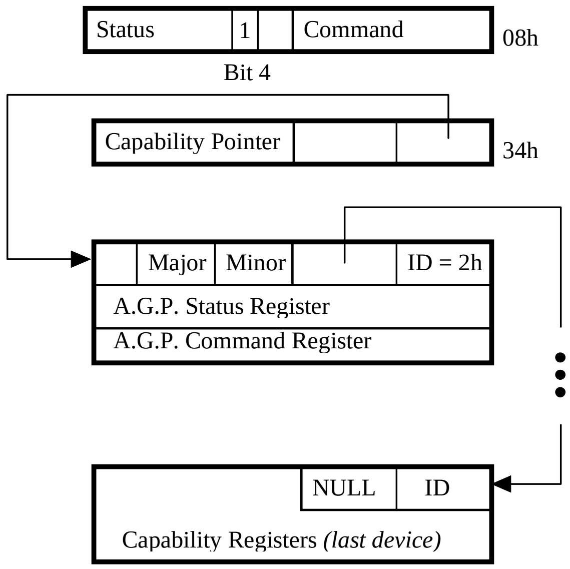


Figure 6-2: Location of A.G.P. Capabilities

Configuration registers are used by the operating system to initialize A.G.P. features. These features must be supported by both A.G.P. 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.

6.1.6 PCI Status Register

Bit	Field	Description
31:5		See the *PCI Local Bus 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*.
3:0		See the *PCI Local Bus Specification*.

6.1.7 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. 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.

6.1.8 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.

6.1.9 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. “0” means a depth of 1 entry, while FFh means a depth of 256 entries.
23:10	Reserved	Always returns 0 when read; write operations have no effect.
9	**SBA**	If set, this device supports sideband addressing.
8:6	Reserved	Always returns 0 when read; write operations have no effect.
5	**4G**	If set, this device supports addresses greater than 4 GB.
4	**FW**	When the bit is set, the device supports FW transfers.
3	Reserved	Always returns 0 when read; write operations have no effect.
2:0	**RATE**	The **RATE** field indicates the data transfer rates supported by this device. A.G.P. devices must report all that apply. **Bit Set** **Transfer Rate** 0 1X 1 2X 2 4X 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.

6.1.10 Command Register - (Offset CAP_PTR + 8)

Bits	Field	Description
31:24	**RQ_DEPTH**	**Master:** The **RQ_DEPTH** field must be programmed with the maximum number of pipelined operations the master is allowed to enqueue in the target. The value set in this field must be equal to or less than the value reported in the **RQ** field of target's Status register. "0" means a depth of one entry, while FFh means a depth of 256 entries. **Target:** 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 sideband address mechanism is enabled in this device.
8	**AGP_ENABLE**	**Master:** Setting the **AGP_ENABLE** bit allows the master to initiate A.G.P. operations. When cleared[40], the master cannot initiate A.G.P. operations. **Target:** 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 completely configured and enabled before the master is programmed. 2. A device can make no assumptions as to the sequence of command field programming, except that **AGP_ENABLE** is the last bit set. Concurrently setting all command fields and the **AGP_ENABLE** bit using a single 32-bit write is also permitted. The **AGP_ENABLE** bit is cleared by **AGP_RESET**.
7:6	Reserved	Always returns 0 when read; write operations have no effect
5	**4G**	**Master:** Setting the **4G** bit allows the master to initiate A.G.P. Requests to addresses above the 4 GB address boundary. When cleared, the master is only allowed to access addresses in the low 4 GB of the address space. **Target:** Setting the **4G** bit enables the target to accept A.G.P. protocol DAC[41] commands when bit 9 is cleared. Setting the **4G** bit enables the target to accept a Type 4 command and to utilize **A[35::32]** for a Type 3 command when bit 9 is set.

[40] When this bit is cleared, the A.G.P. master is allowed to quit driving the SBA port; however, if bits 1 or 2 are set in the Rate register, the master is required to perform a re-synch cycle before initiating a new request. (See Section 4.1.2.10 for details.)

[41] This bit has no affect on whether PCI protocol DAC is supported or not.

4	**FW_Enable**	When this bit is set, the device must do Memory Write transactions from the corelogic to the A.G.P. master following FW[42] protocol. When this bit is cleared memory write transactions from the corelogic to the A.G.P. master follow standard PCI protocol.
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. The same bit must be set on both master and target. **Bit Set** **Transfer Rate** 0 1X 1 2X 2 4X 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 remaining unaffected when written. All bits in the A.G.P. Command register are initialized to zero at reset.

6.2 A.G.P. Master MDA Resource Use Restrictions

A.G.P. platforms may support multiple graphics adapters in the system, including monochrome device adapters (MDA) off the PCI or standard expansion buses. A.G.P. graphics controllers which support coexistence with an MDA controller are required to not use any of the following standard MDA resources, whenever an MDA is present in the system:

Memory addresses:	0B0000h - 0B7FFFh
I/O Addresses:	3B4h, 3B5h, 3B8h, 3B9h, 3BAh, and 3BFh,
	(including ISA address aliases, **A[15::10]** are not used in decode)

A.G.P. graphics controllers may utilize the MDA resources if an MDA is not present in the platform. A.G.P. graphics controllers that do not support coexistence with MDA are not required to follow these MDA resource use restrictions; however, such controllers will cause platform failures if MDA devices are present.

During system initialization, software needs to determine if an MDA is present in the system or not. Before the primary graphic device is enabled, software will generate a write access to an MDA memory range (B0000h-B7FFFh), then read it back to determine if MDA is present. If an MDA device is present, software may configure the corelogic to pass MDA references to the primary PCI bus (which can then be claimed by the expansion bus bridge). How the corelogic chooses to provide this functionality (if supported) is specific to the implementation of the corelogic and is beyond the scope of this interface specification. Software completes the initialization process by allocating resources requested by the different agents and enabling the devices to operate.

[42] Fast Write protocol is not defined nor supported for accesses from the A.G.P. master to the corelogic. This type of transaction follows standard PCI protocol.

6.3 Multifunction A.G.P. Master

A multifunction device consists of at least two independent functions integrated on a single piece of silicon. Any single function device can be composed of multiple sub-functions. The left block diagram in Figure 6-3 is a single function PCI agent that has two subfunctions x and y. The right hand block is a multifunction device that has functions A and B. Note that A or B could have multiple subfunctions as illustrated in the left block diagram. From a software point of view, the main difference between a single function device (that has multiple subfunctions) and a true multifunction device, is that a unique device driver is required for each function, but is not required for each subfunction. For a multifunction PCI device, the only logic shared between functions is the actual I/O buffers and an internal arbiter that determines when each function gains access to the bus. Beyond this difference, each function is completely independent of all other functions. A PCI example might be a LAN and a SCSI controller each of which was developed independently but are now integrated on the same silicon. Function A in the figure could be the LAN controller and function B could be the SCSI controller.

Since only a single **REQ#**/**GNT#** pair is available at the device interface, the arbiter determines which interface logic may become a master on the bus. Each must do its own decode and assert **DEVSEL#** when selected. The existing device driver for each function can be used unmodified[43] with the new integrated device.

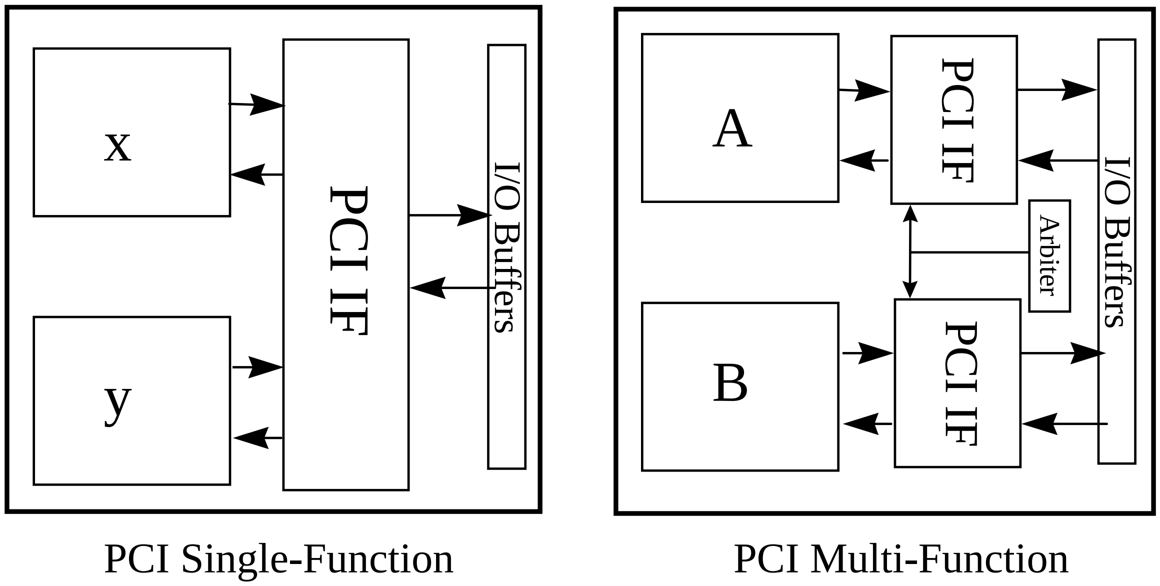


Figure 6-3: PCI Device Block

When implementing an A.G.P. multifunction device, it is similar to a PCI multifunction device, but with significant differences. Since the A.G.P. master sequencer can only request a single data transfer rate, all functions on the device must use the same rate. All devices must use either 1x, 2x, or 4x and are not allowed to switch data rates between transactions. The master must also indicate whether it will enqueue requests via the **AD** bus or the **SBA** port. Simply integrating independent A.G.P. implementations does not result in a viable implementation since the master must present one choice per option. This requires all functions of the device to use the same modes.

[43] This means that the same driver is used independent of whether a single function card is inserted in the system, or a multifunction card containing the same function is used.

On the other hand, the corelogic must support all basic modes of operation defined by this interface specification. It may optionally support the enhanced modes (4x and FWs transactions). However, the corelogic is not required to support all modes at the same time. For example, the corelogic can be programmed to accept requests on the **AD** bus or the **SBA** port but is not required to support both at the same time.

No device driver changes are required for a PCI multifunction implementation. The same should be true for a multifunction A.G.P. implementation. However, care needs to be taken to ensure that this is true. If the driver used information on how the interface was implemented, the integration of multiple functions could require driver changes. For example, if the driver used the RQ_DEPTH value to optimize its operation, the device driver would require changes when the function is integrated with other A.G.P. functions. In the multifunction case, this value is shared between multiple functions and cannot be used directly by the driver. On the other hand, if this type of information is not used directly by the driver, no driver changes are required.

Figure 6-4 shows how an A.G.P. master would support multiple functions. Note that unlike the PCI block diagram (Figure 6-3) where there are differences between a multifunction device and a single function device with multiple subfunctions, A.G.P. has no differences. Consequently, function 0 represents the A.G.P. capabilities of the device regardless of how many functions use the A.G.P. interface.

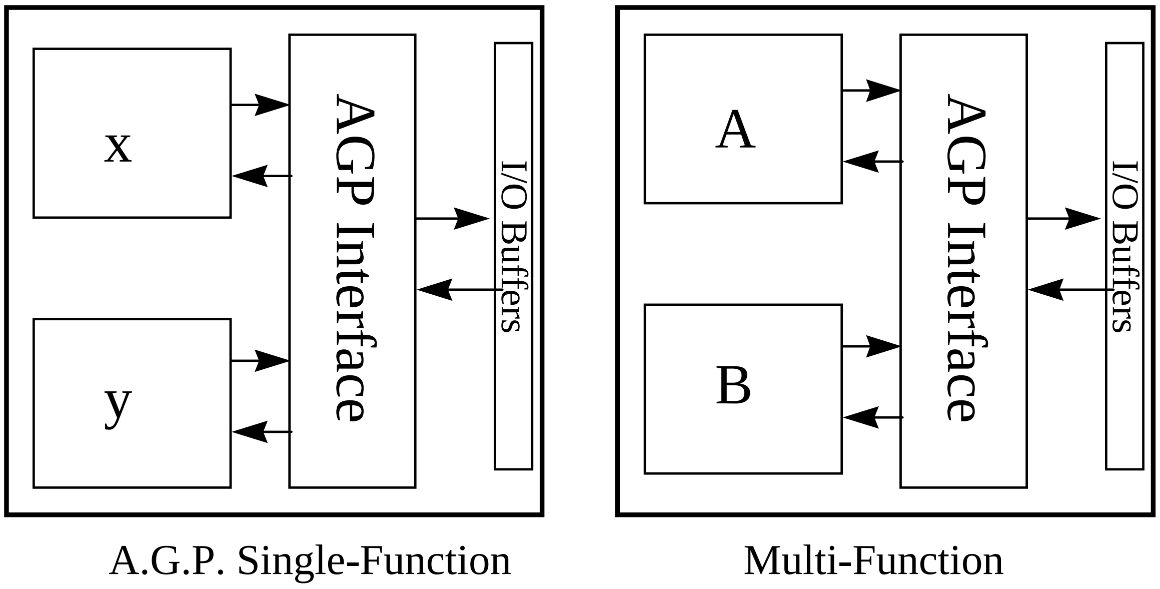


Figure 6-4: A.G.P. Block

6.3.1 A.G.P. Configuration Registers

Each A.G.P. function is required to implement its own PCI target interface. This ensures that system resources (memory space, I/O space, and an interrupt line) that the device requires are assigned during configuration time.

Since A.G.P. capabilities are queried after the operating system is running, function 0 (single or multifunction device) of the A.G.P. master presents its capabilities to the system in the A.G.P. Status register. System software then programs the A.G.P. Command register as to how the A.G.P. interface will operate. A summary of operating options is presented in Table 6-2. The designer must ensure that all functions that use the interface make the same choices.

Table 6-2: A.G.P. Options

A.G.P. Attribute	Options
Rate	1x, 2x, or 4x
FW	Support or not
4 GB Addressing	Support or not
Requests	**SBA** port or **PIPE#**
Outstanding requests	1 to 256

Close attention should be paid to the RQ field of the Status register and the RQ_DEPTH field of the Command register. The RQ field must represent the sum of the maximum number of memory reference transactions that all functions on the device can have outstanding at any given time. For example, if function 0 can have 24 A.G.P. Requests outstanding and function N[44] can have an additional 12 A.G.P. Requests, then the RQ field of the master needs to indicate that up to 36 requests can be outstanding at any given time.

Careful attention needs to be paid when the RQ_DEPTH field is programmed to a value less than the RQ field. When this condition occurs, the designer must choose how the available request slots are allocated. A simple implementation would be to do a first come first served approach. A more complicated, but fairer option would be to allocate a fixed ratio to each function. Any choice is acceptable as long as the A.G.P. master never allows more requests to be enqueued then the RQ_DEPTH field allows.

The AGP_ENABLE bit (9) in the Command register controls all master functions that use the A.G.P. interface. When this bit is cleared (0), the interface is not allowed to make any new requests to the corelogic. This bit has the same meaning for both single or a multifunction implementations. For example, if function 0 is a 3D engine and function 4 is a video capture device, both functions are disabled when the AGP_ENABLE bit (which resides in function 0) is cleared. If individual enable bits are desired, these are provided for in function specific space.

[44] The device is not required to implement the second function as number 1, but can assign it any value between 1 and 7.

6.3.2 Internal Arbiter

In a PCI multifunction device the arbiter determines which PCI master interface logic gains access to the device's pins to initiate a request on the bus. This same functionality does not exist in a multifunction A.G.P. implementation. The arbiter in the A.G.P. master sequencer determines which request is enqueued next, regardless of which function generated the request, but common logic presents the request to the system.

6.3.3 Deadlock Avoidance

When multiple A.G.P. functions are supported, the designer must ensure that no deadlocks or livelocks are possible. The designer must ensure that the master device appears to the corelogic as a single function agent. This requires that the A.G.P. master make no assumptions as to the order in which requests are completed. Note that in a multifunction implementation, function A and function B may interfere with each other's operation. For example, function A has LP Read requests 4, 5, and 9 outstanding while function B has LP Read requests 6, 7, and 8 outstanding. If function B asserts **RBF#** when data for request 7 is returned, it will cause the delay of function A's data until request 8 completes. Function A may have room to accept the data but the corelogic is not allowed to return it until transaction 8 completes. Therefore, care needs to be taken to balance how requests are enqueued and provide sufficient buffer to ensure that one function does not affect the other.

A. Terminology

A.G.P. bus	Abbreviation for PCI/A.G.P. bus.
A.G.P. master	An A.G.P. compliant master interface that is capable of generating A.G.P. pipelined read/write operations per this interface specification.
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.
A.G.P. port	Connection point on a chipset where an A.G.P. compliant master may be attached.
API	Application Programming Interface.
Bus 0	Compatibility PCI bus *(where ISA bridge resides).*
Bus *n*	PCI/A.G.P. bus. *n* = 1 when no PCI to PCI Bridges present on Bus 0.
Chipset	Motherboard chipset that provides connections to: Host Bus, Compatibility PCI bus, and A.G.P. interface.
DirectX	Microsoft API's for accessing graphics and audio hardware.
DirectDraw	Microsoft graphics API.
Display surface	Memory area containing graphics data object.
Fence	Means of synchronizing A.G.P. write operations with subsequent A.G.P. read operations.
Flush	Operation that makes an A.G.P. compliant target's accesses to system graphics memory visible to other parts of the system.
A.G.P. 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.
Local graphics memory	Memory local to the graphics controller.
Non-prefetchable memory	PCI registers that have side effects.
Prefetchable Memory	PCI memory and memory mapped registers that are free from side-effects.
Uncacheable Memory	Host caching protocol used on I/O operations, non-prefetchable regions or prefetchable regions not supported by hardware coherency.
Ordering rules	Rules specifying when the effect of a read or write operation can be observed by another operation.
Paging	Movement of data between disk and other memory levels of the virtual memory system.
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.
POST or POST Code	Initialization software executed out of the startup ROM.
SPI	System Programming Interface.

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.
System graphics memory	System memory accessible via the A.G.P. port by the graphics controller.
WriteBack	A type of Host cache coherency used for application memory.

B. Supplemental Timing Diagrams

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.

TBD

C. A.G.P 4X 110 W Connector

This appendix describes the high-end A.G.P. 4X 110 W connector designed for the workstation and server market segments. It also defines a 110 W thermal envelope to support these cards in the system. The connector is designed to support A.G.P. 1X, 2X, and 4X mode graphic cards. The target market segments for this connector are servers and workstations which may have additional power consumption requirements above and beyond the power supplied by the connectors described in Chapter 5. This connector is not required for desktop applications.

A.G.P. 4X graphic cards for the server and workstation market segments are projected to consume 50-85 W of power. Future high-end designs may even go up to 110 W per A.G.P. card. This connector builds on the existing A.G.P. connector work (i.e. add any new features as a superset). The purpose is to deliver enough power to the cards as cleanly as possible and to define a thermal envelope which supports high-end A.G.P. 4X graphics cards in the system.