ECR #: 19
Title: Arbitration Deadlock
Release Date: Feb. 17, 1997
Impact: Change
Spec Version: A.G.P. 1.0

Summary: The current definition of arbitration can cause a deadlock to occur. The arbiter needs the right to remove GNT# at anytime to resolve the deadlock condition. A deadlock can occur when the arbiter has asserted GNT# to the A.G.P. compliant master when the Processor makes an access to the A.G.P. compliant master (using PCI protocol) and disables the master by clearing either its PCI or A.G.P. compliant master enable bits. The current specification is written such that the arbiter is not allowed to remove GNT# until either FRAME# or PIPE# is sampled asserted. In this case, the arbiter is not allowed to remove GNT#, but the master never asserts FRAME# or PIPE#.

Background:

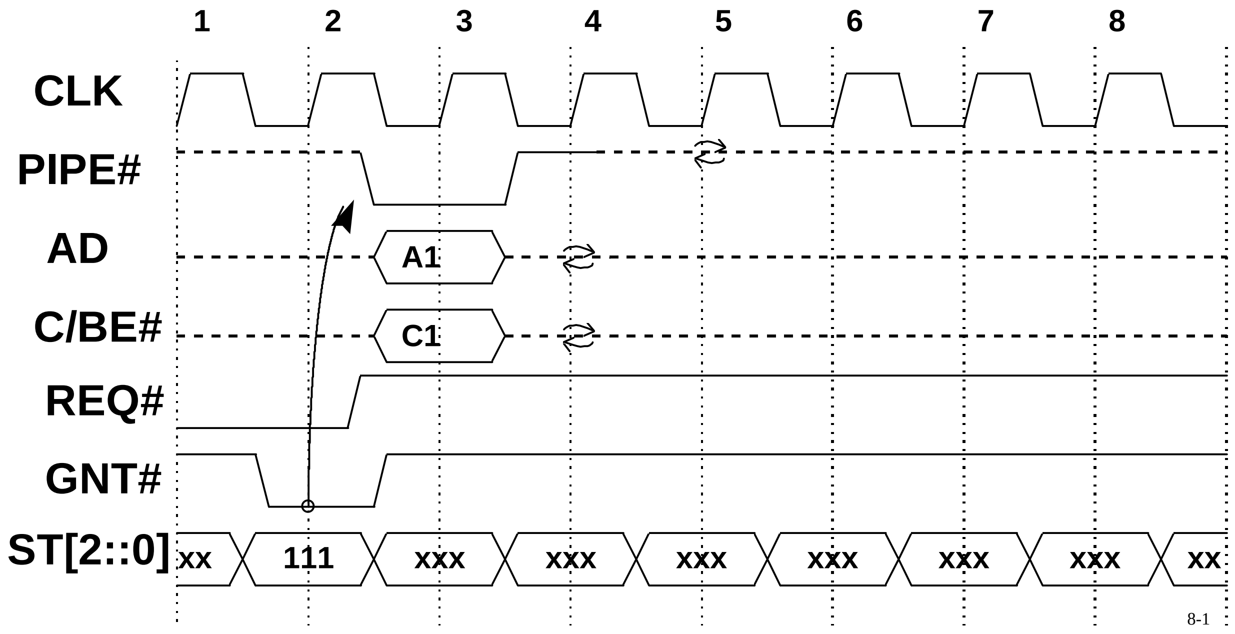
In the 1.0 interface specification the arbitration rules are not consistent with the current PCI specification and are different for A.G.P. transactions. This blending of rules (A.G.P. and PCI) has caused confusion and potential incompatibilities.

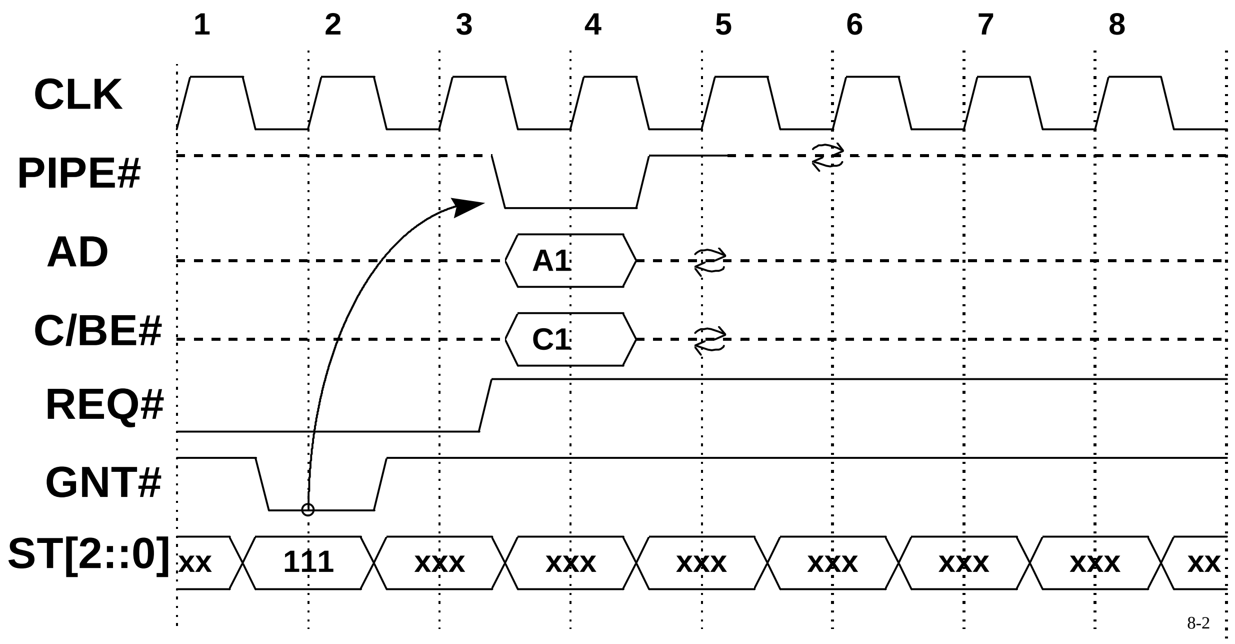
Change Current Specification as follows:

A.G.P. Compliant Master

A.G.P. Compliant Master initiating an A.G.P. request.

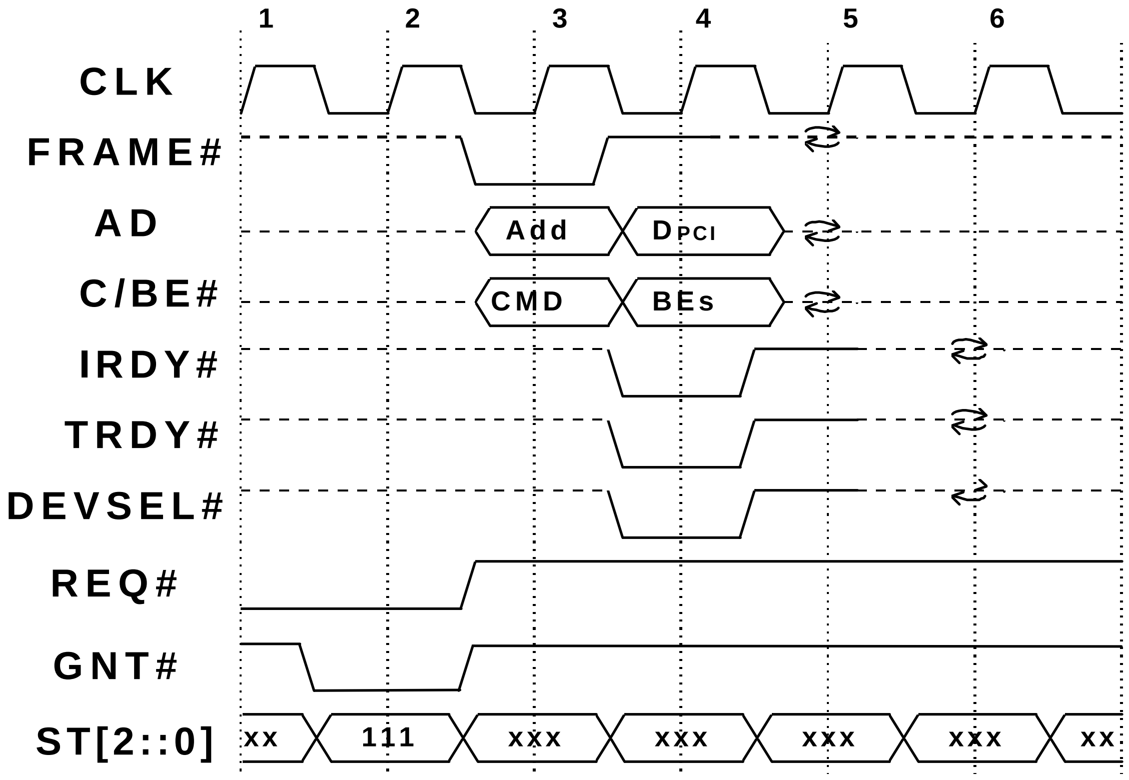
When the A.G.P. compliant master has **REQ#** asserted to request permission to use the **AD** bus to make an A.G.P. request, it must assert **PIPE#** within 2 clocks of sampling **GNT#** asserted and **ST[2::0]** = 111 and the bus is in a state in which the master can start. Figure 8-1 illustrates the earliest in which **PIPE#** can be sampled asserted by the arbiter. Figure 8-2 illustrates the latest it can be sampled asserted.




1
2
3
4
5
6
7
8
CLK
PIPE#
AD
C/BE#
REQ#
GNT#
ST[2::0]
A1
C1
xx
111
xxx
xxx
xxx
xxx
xxx
xxx
xx
8-2

A.G.P. Compliant Master initiating a PCI transaction

When the A.G.P. compliant master has **REQ#** asserted to request permission to use the **AD** bus to make an PCI transaction, it must follow the PCI 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 and **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. compliant master must follow the PCI 2.1 specification and 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 8-74 illustrates a PCI transaction . Since **GNT#** is asserted on clock 2 the A.G.P. compliant master is required to assert **FRAME#** so the arbiter samples it asserted on clock 3 otherwise the master may[1] 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 the master to initiate a PCI transaction but is allowed.



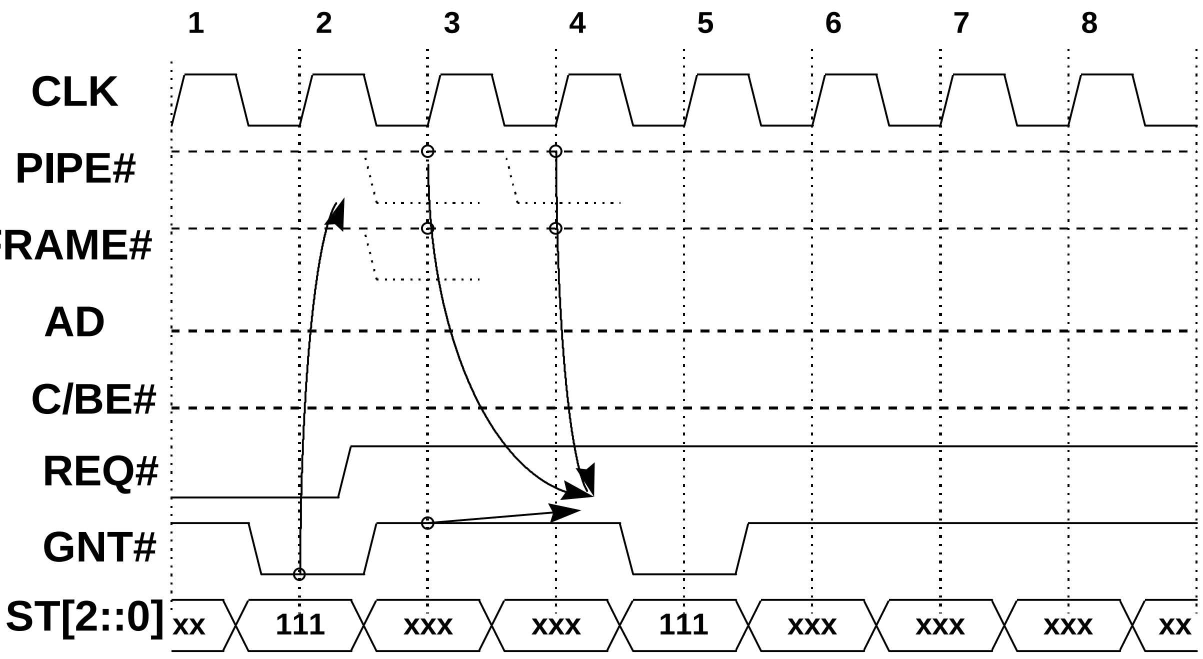
8-74

[1] If the arbiter keeps GNT# asserted then the master is allowed to assert **FRAME#** later, but it must always be asserted the next clock after GNT# was asserted. If GNT# is only active for a single clock and the PCI master requires 2 clocks to get going (i.e., address stepping), the master may never gain access to the bus.

A.G.P. Compliant Arbiter

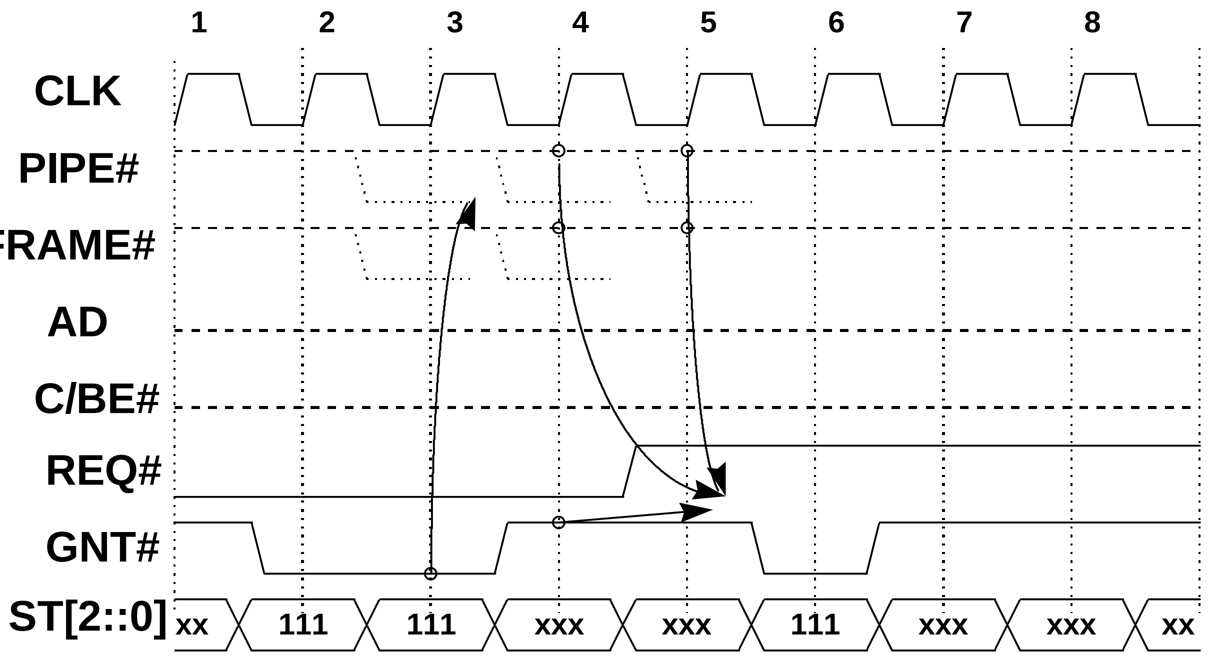
The arbiter is allowed to deassert (remove) **GNT#** at anytime. This allows the arbiter to prevent deadlocks from occurring.

When the arbiter removes a **GNT#** (deasserts it), the arbiter must not assert a new **GNT#** (to grant permission to initiate a request) until the original master has been allowed to initiate its transaction. For a PCI transaction this is one clock, while for A.G.P. the master is allowed 2 clocks to start. In figure A-1, 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.



a-1

Figure A-2 illustrates the arbiter keeping **GNT#** asserted on clock 3 instead of deasserting it, as in figure A-1. 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.



a-2

Pipelining GNT#s

The arbiter is allowed to pipeline grants when other bus transactions are in progress. This pipelining can be grouped into 4 difference conditions:

1. A Request transaction followed by a Request transaction.
   An A.G.P. Request followed by a PCI Read Transaction.
   An A.G.P. Request followed by a PCI Write Transaction.
   A PCI Transaction followed by a PCI Transaction.
   A PCI Read Transaction followed by an A.G.P. Request (**PIPE#**).
   A PCI Write Transaction followed by an A.G.P. Request (**PIPE#**).

   (Note: An A.G.P. Request can not 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.)

2. A Request transaction followed by a data transfer.
   An A.G.P. Request followed by an A.G.P. Read transaction.
   An A.G.P. Request followed by an A.G.P. Write transaction.
   A PCI Read Transaction followed by an A.G.P. Read Transaction.

A PCI Write Transaction followed by an A.G.P. Read Transaction.
A PCI Read Transaction followed by an A.G.P. Write transaction.
A PCI Write Transaction followed by an A.G.P. Write transaction.

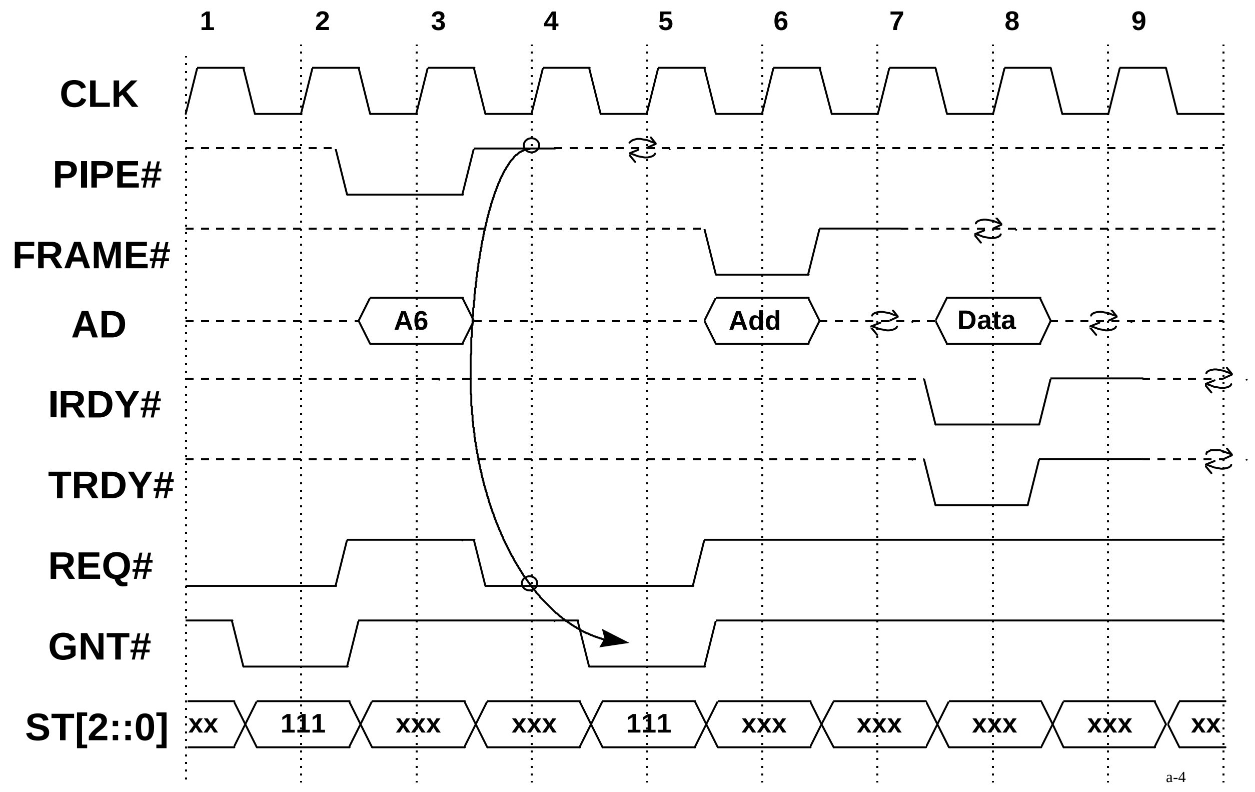
3. A Data transfer followed by a Request.
    An A.G.P. Read transaction followed by an A.G.P. Request.
    An A.G.P. Write transaction followed by an A.G.P. Request.
    An A.G.P. Read Transaction followed by a PCI Read Transaction.
    An A.G.P. Read Transaction followed by a PCI Write Transaction.
    An A.G.P. Write transaction followed by a PCI Read Transaction.
    An A.G.P. Write transaction followed by a PCI Write Transaction.

4. A data transfer followed by a data transfer.
    An A.G.P. Read transaction followed by an A.G.P. Read transaction.
    An A.G.P. Read transaction followed by an A.G.P. Write transaction.
    An A.G.P. Write transaction followed by an A.G.P. Read transaction.
    An A.G.P. Write transaction followed by an A.G.P. Write transaction.

A Request transaction followed by a Request transaction.

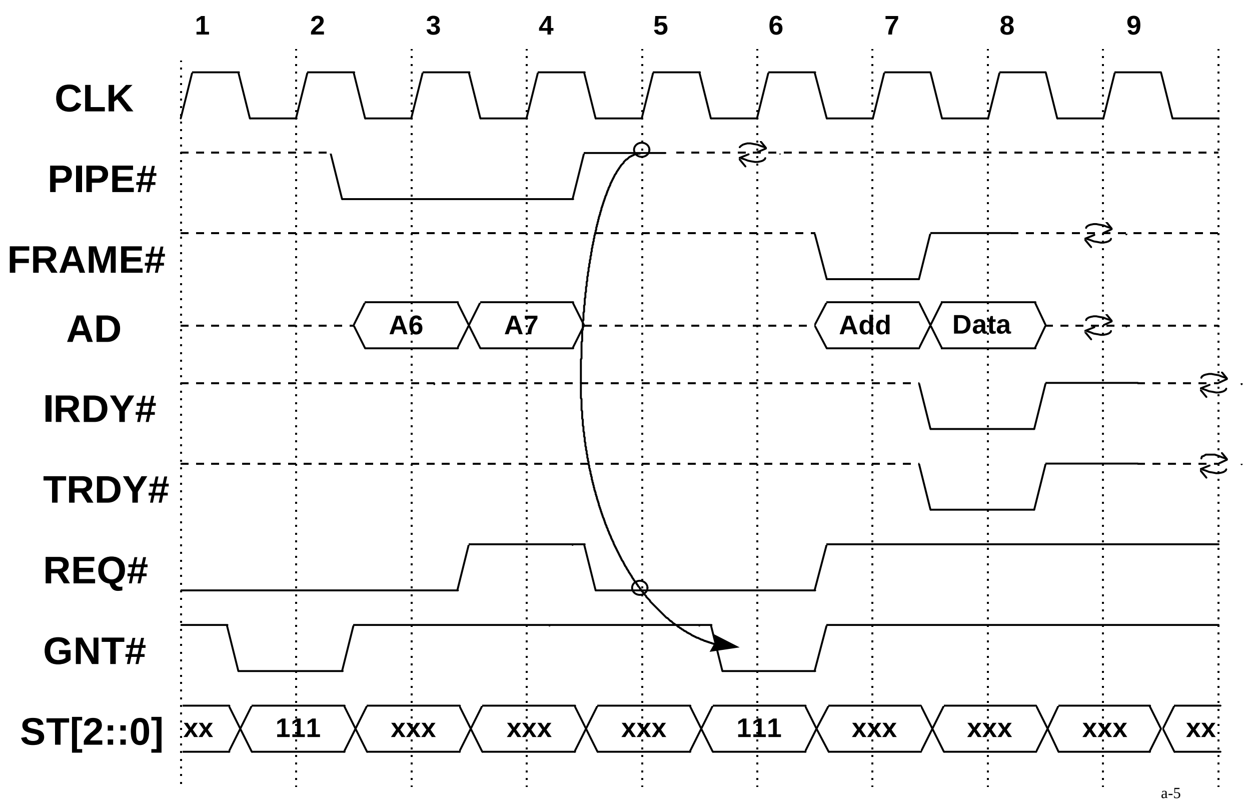
The arbiter is allowed to assert **GNT#** for a subsequent Request (PCI or A.G.P.) when **GNT#** has been deasserted for at least 2 clocks AND **FRAME#** or **PIPE#** is not asserted during those two clocks. Figure A-2 illustrates this condition. The arbiter is allowed to assert a new **GNT#** when **FRAME#** is asserted. The arbiter is not allowed to pipeline a second request while a A.G.P. request is currently active. (**REQ#** must be deasserted before it indicates that a "new" request is pending.)

An A.G.P. Request followed by a PCI Read Transaction. (A4)
An A.G.P. Request followed by a PCI Write Transaction. (A5)
A PCI Transaction followed by a PCI Transaction. (A6, BtoB A7)
A PCI Read Transaction followed by an A.G.P. Request (**PIPE#**). (A8)
A PCI Write Transaction followed by an A.G.P. Request (**PIPE#**). (A9)



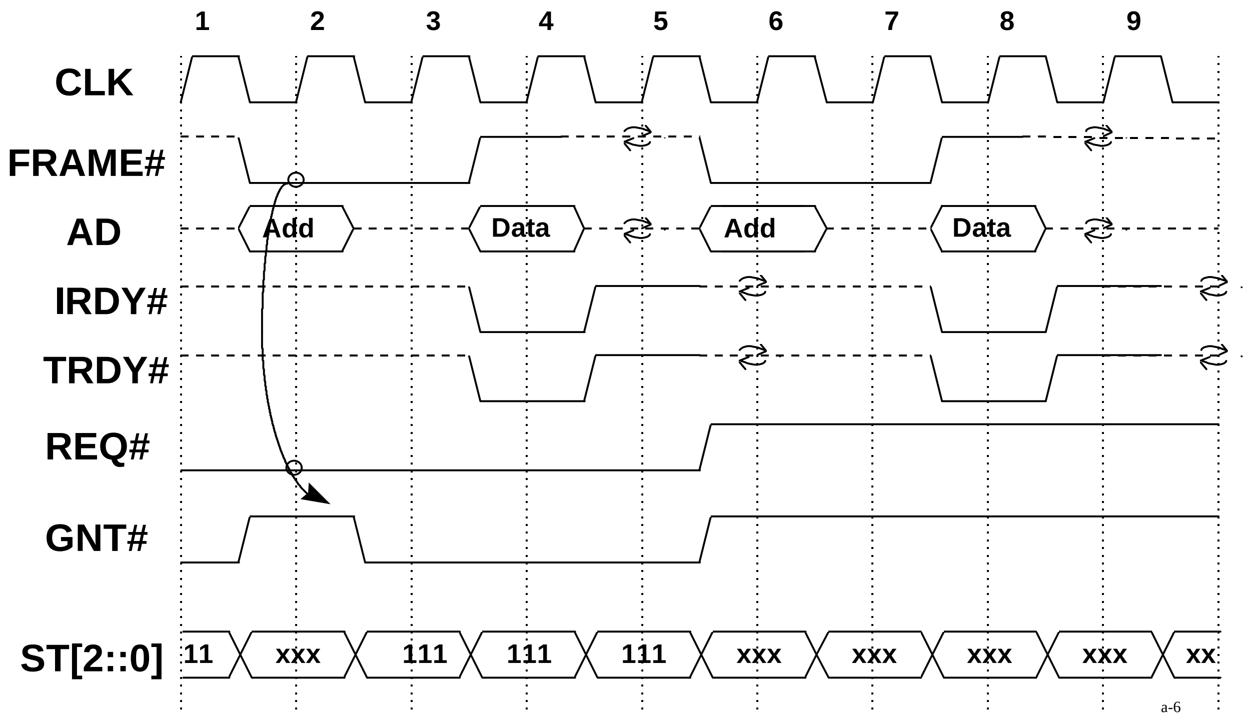
Single phase PIPE# followed by an AGP Master generated PCI Cycle
Governed by REQ# deassertion, no AD turnaround required

Figure A-4 illustrates the arbiter granting permission to start of 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 until **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 then **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.



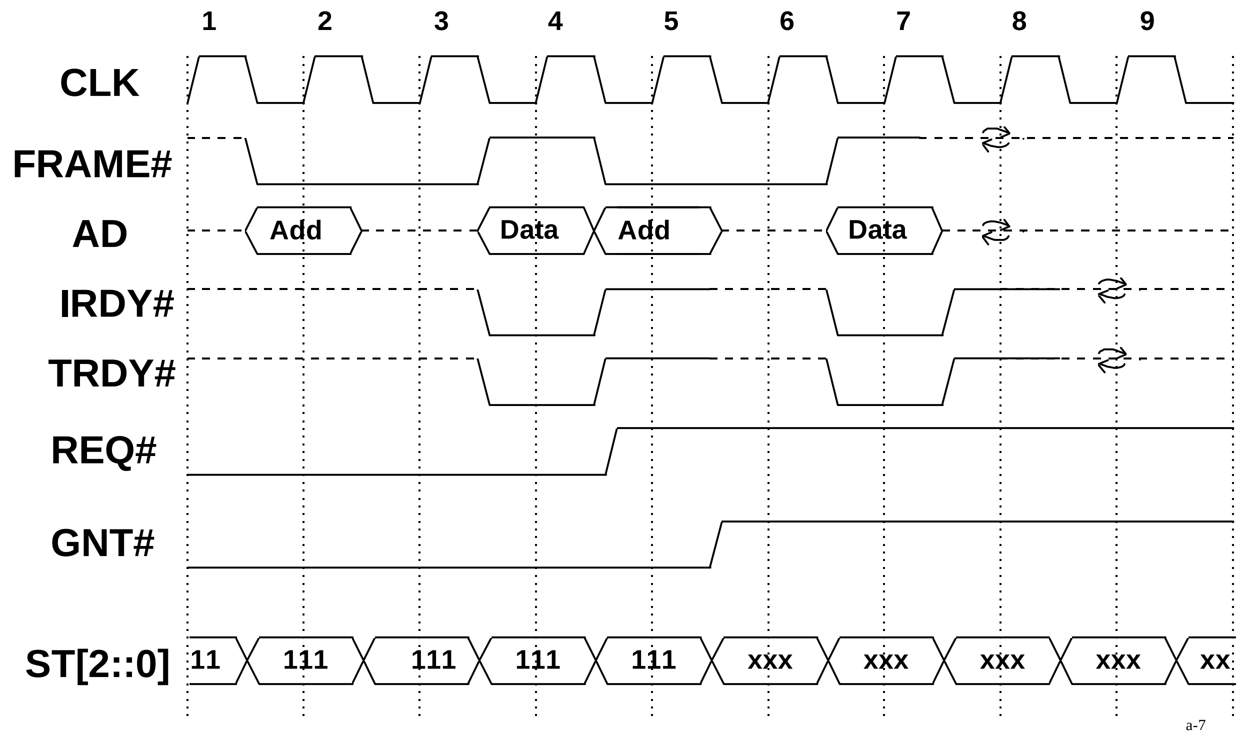
Double phase PIPE# followed by an AGP Master generated PCI Cycle
Governed by REQ# deassertion, no AD turnaround required

Figure A-5 is the same as A-4 except that multiple requests are enqueued. 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 then the **GNT#** would be delayed until clock 7.



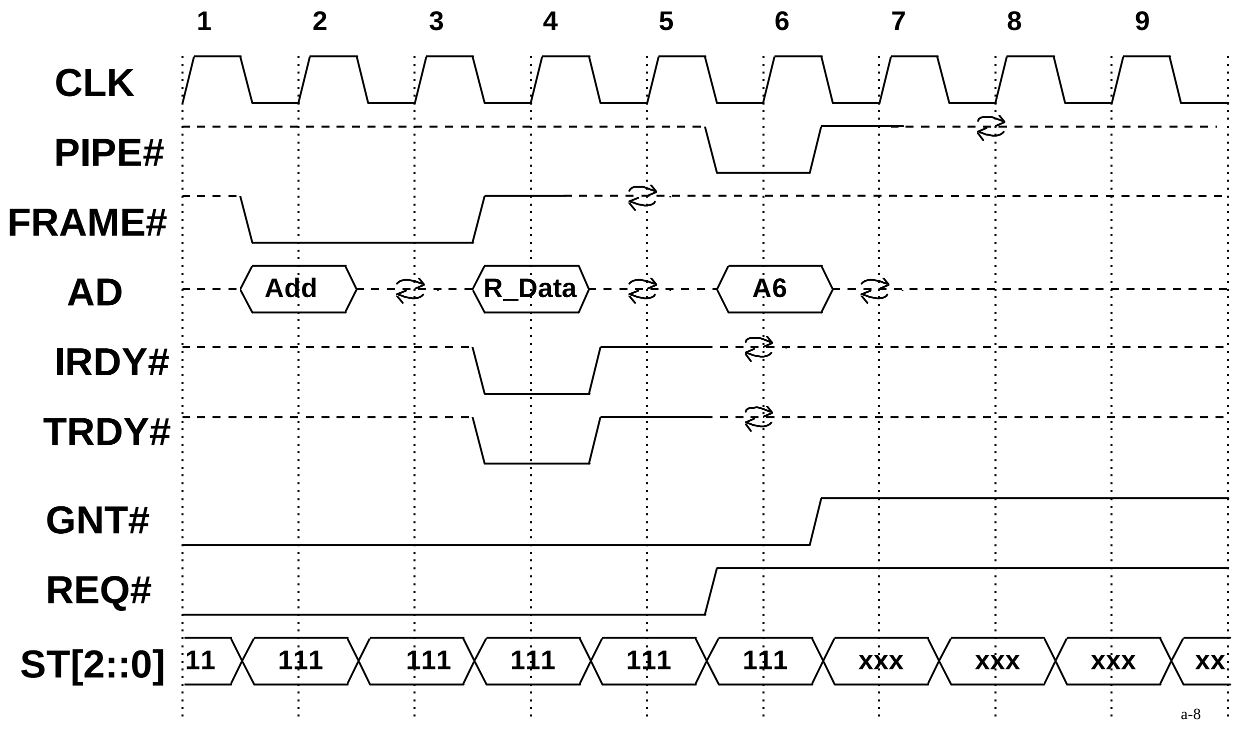
PCI Cycle followed by PCI Cycle
FRAME# asserted from Idle Bus

Figure A-6 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. compliant agent. 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.



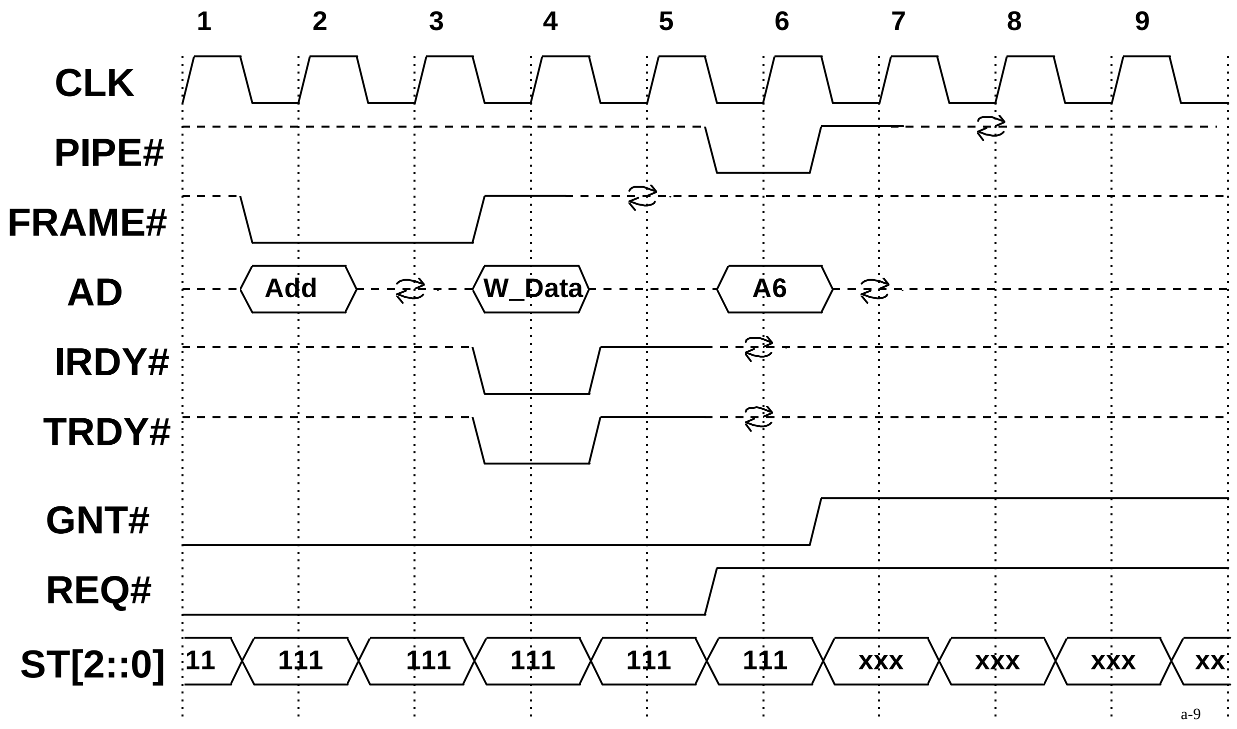
Fast back-to-back PCI Cycles

Figure A-7 is illustrates the same operation as A-6 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 A-6, but keeps it asserted indicating that the current master is allowed 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) indicated 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 (3) 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 all the arbiter to grant the bus to a different resource.



PCI Cycle followed by PIPE#
PIPE# asserted from Idle Bus

Figure A-8 illustrate 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 one 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. compliant 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. compliant 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. compliant master always uses two clock to get the transaction started and knows samples 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.



PCI Cycle followed by PIPE#
PIPE# asserted from Idle Bus

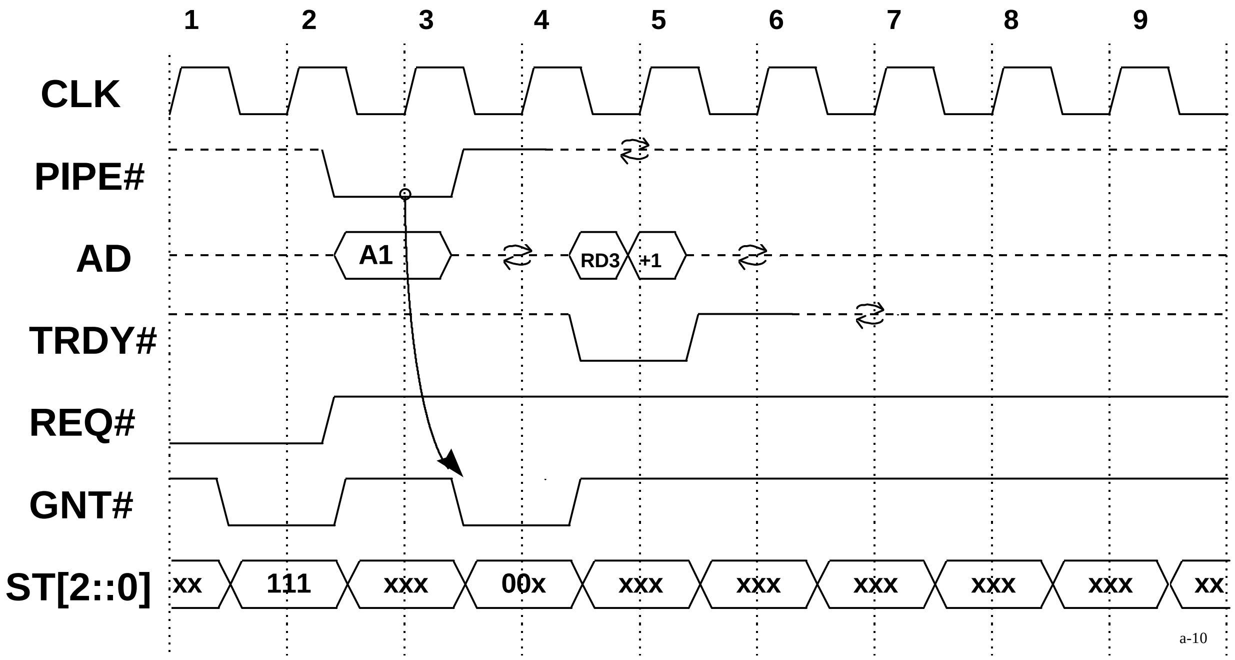
Figure A-9 is the same as A8 except that the PCI transaction is a write instead of a read. In this case a turn-around is not required between the data phase of the PCI transaction and the A.G.P. request, but a dead clock is required between any PCI transaction and A.G.P. transaction. The master of the 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 until clock 5 is acceptable, since the **GNT#** is to the same master and it knows when the current transaction completes and when the subsequent is allowed to start.

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 starting of the transaction may use latched signals and thereby causing the **GNT#** with **ST[2::0]** = 111 to remain after the Request has started. Since the A.G.P. compliant master ignores **GNT#** asserted when **ST[2::0]** = 111 except on the clock in which it desires to start and transaction and is allowed to do so. In all other cases, this condition on the interface is meaningless.

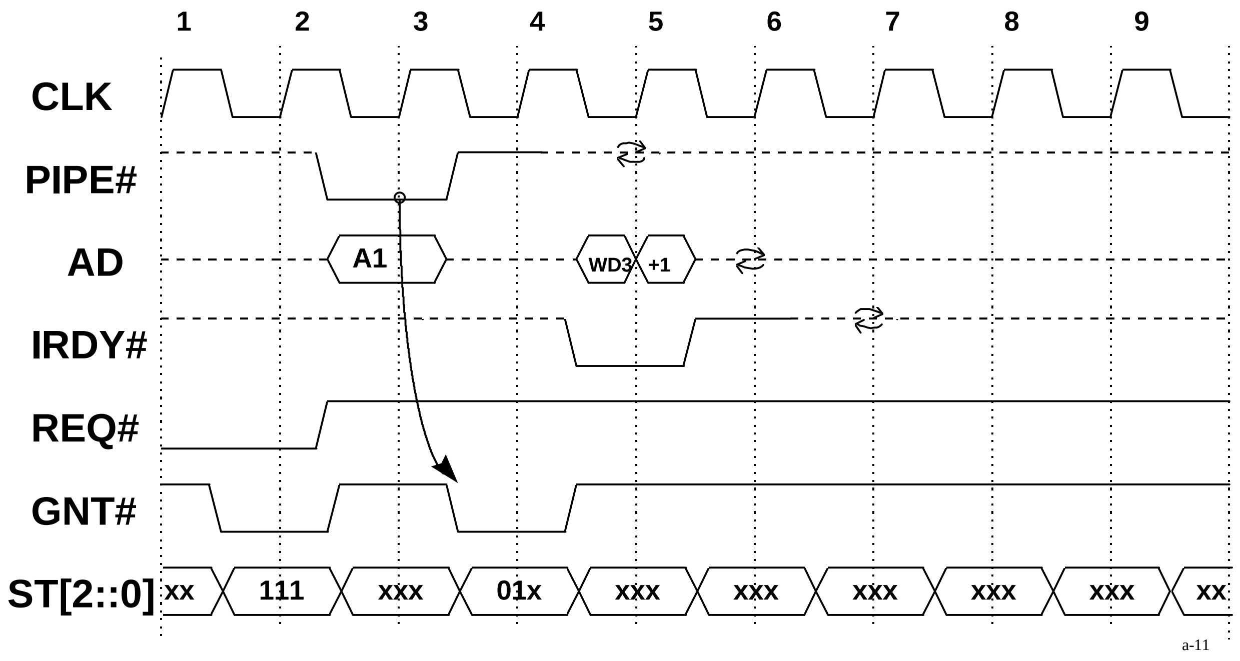
An A.G.P. Request followed by an A.G.P. Read transaction. (A10)
An A.G.P. Request followed by an A.G.P. Write transaction. (A11, A12)
A PCI Read Transaction followed by an A.G.P. Read Transaction. (A13 - dead cycle does not require a turn around)

A PCI Write Transaction followed by an A.G.P. Read Transaction. (A13 - only difference is the dead clock is a turn-around cycle)
A PCI Read Transaction followed by an A.G.P. Write transaction. (A14) Dead clock is a turn-around
A PCI Write Transaction followed by an A.G.P. Write transaction. (A14) Dead clock is not a turn-around



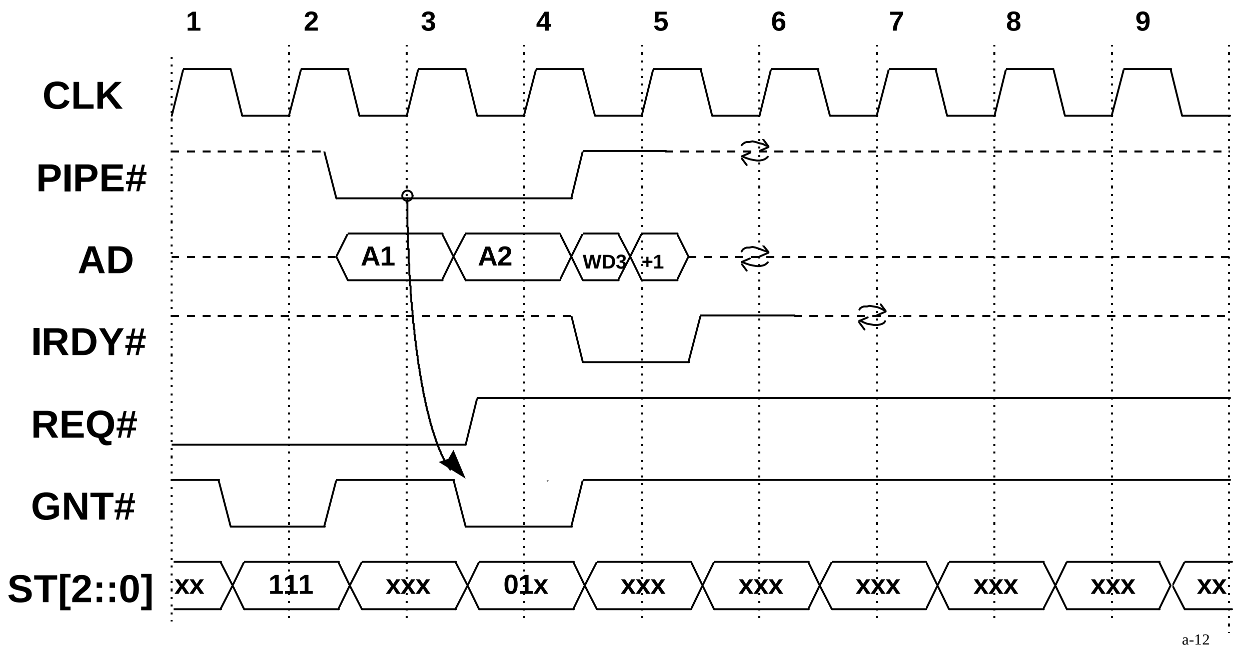
Single phase PIPE# followed by Read Data
Governed by read GNT#, AD turnaround required

Figure A-10 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.



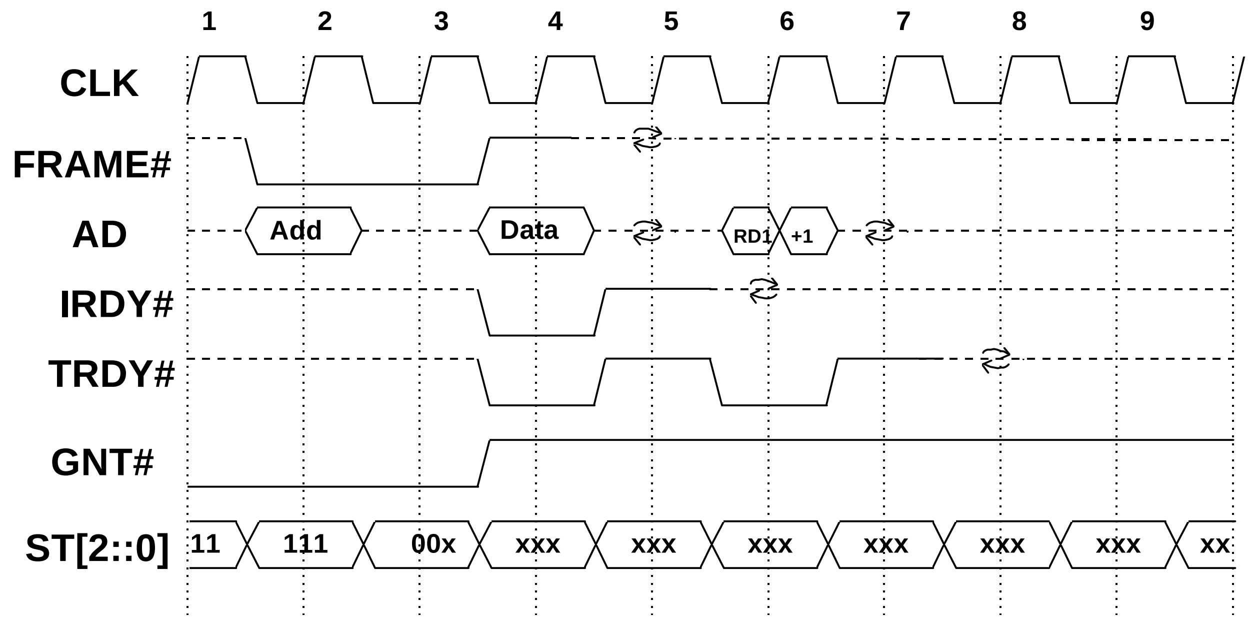
Single phase PIPE# followed by Write Data
Governed by write GNT#

Figure A-11 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 A-12 shows the case where **PIPE#** is asserted long enough for the next **GNT#** to be asserted 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 2 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 asserted on clock 4 with **ST[2::0]** = 111 then the write data transfer would have been delayed. This condition can occur when the arbiter uses a latched version of **PIPE#**.



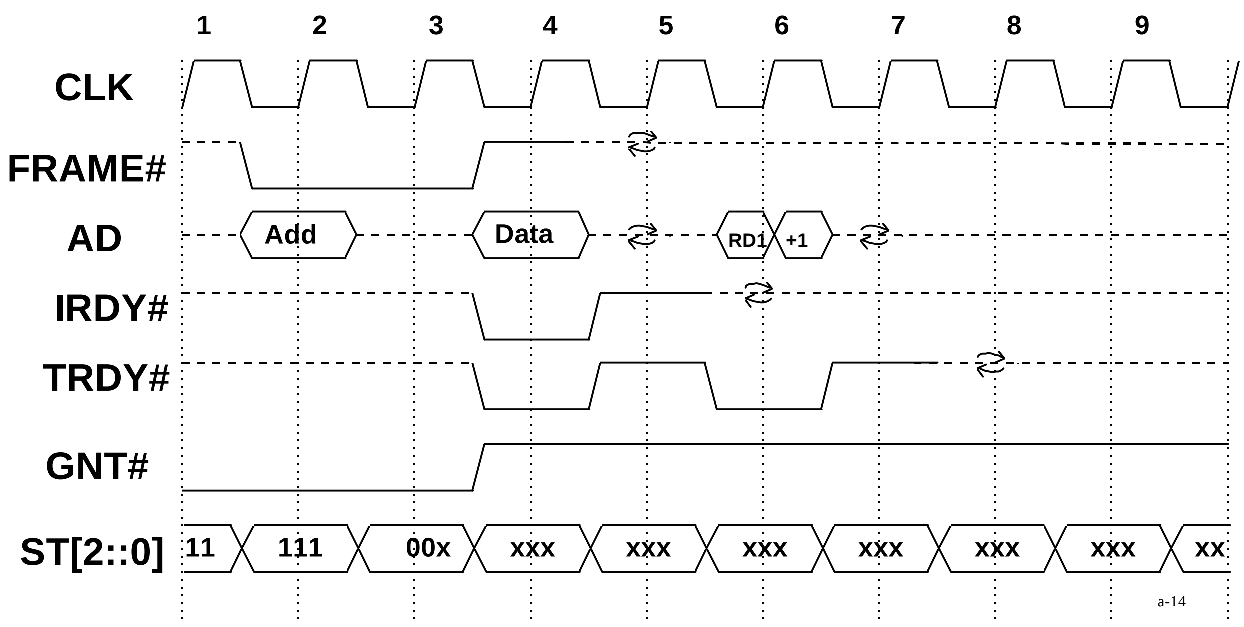
Double phase PIPE# followed by Write Data
No AD turnaround required

Figure A-12 is a write data following an A.G.P. Request. In this case, there is no turn-around cycle required since the same agent continues to own the **AD** bus for both transactions. The arbiter asserts **GNT#** indicating the A.G.P. compliant 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. compliant 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 then the write data transaction would have been delayed and bus bandwidth would have been wasted.



AGP Master generated PCI Cycle followed by Read Data a-13
No TRDY# turnaround required, TRDY# asserted from Idle Bus

Figure A-13 is a PCI transaction that is initiated by an A.G.P. compliant master followed by 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 assert 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 then 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. Although a dead cycle is required between PCI and A.G.P. transactions.



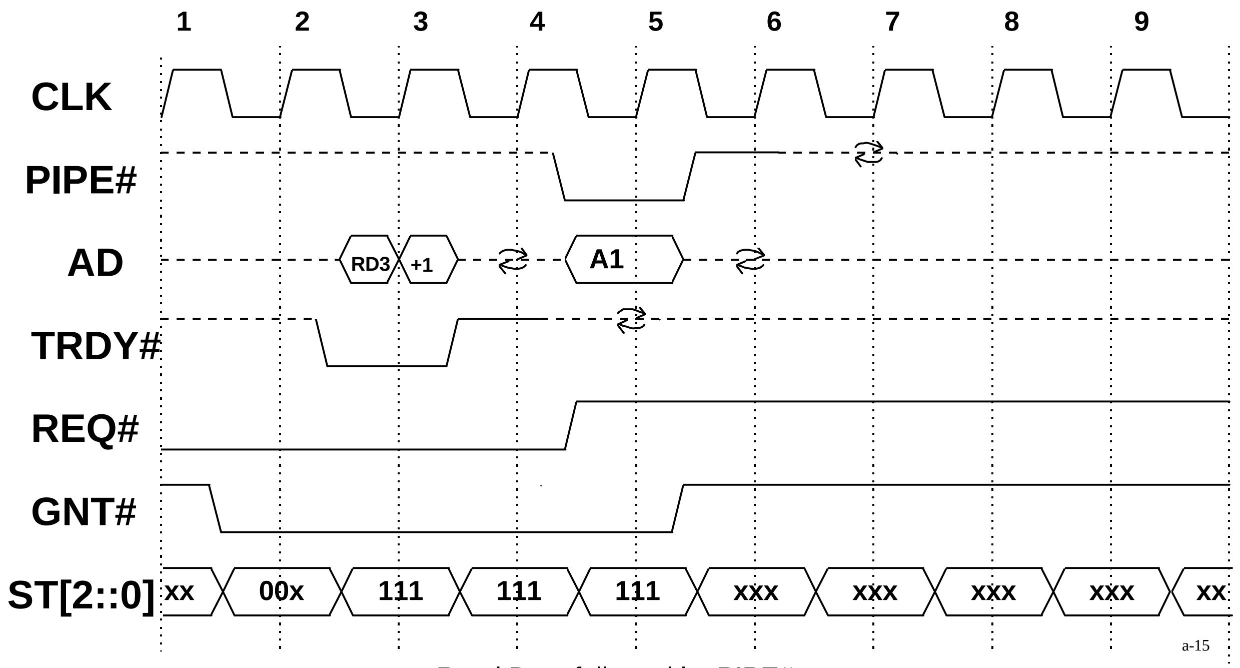
AGP Master generated PCI Cycle followed by Read Data
No TRDY# turnaround required, TRDY# asserted from Idle Bus

Figure A-14 is basically the same as A-13 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. compliant 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. compliant 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. compliant 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.

A Data transfer followed by a Request.

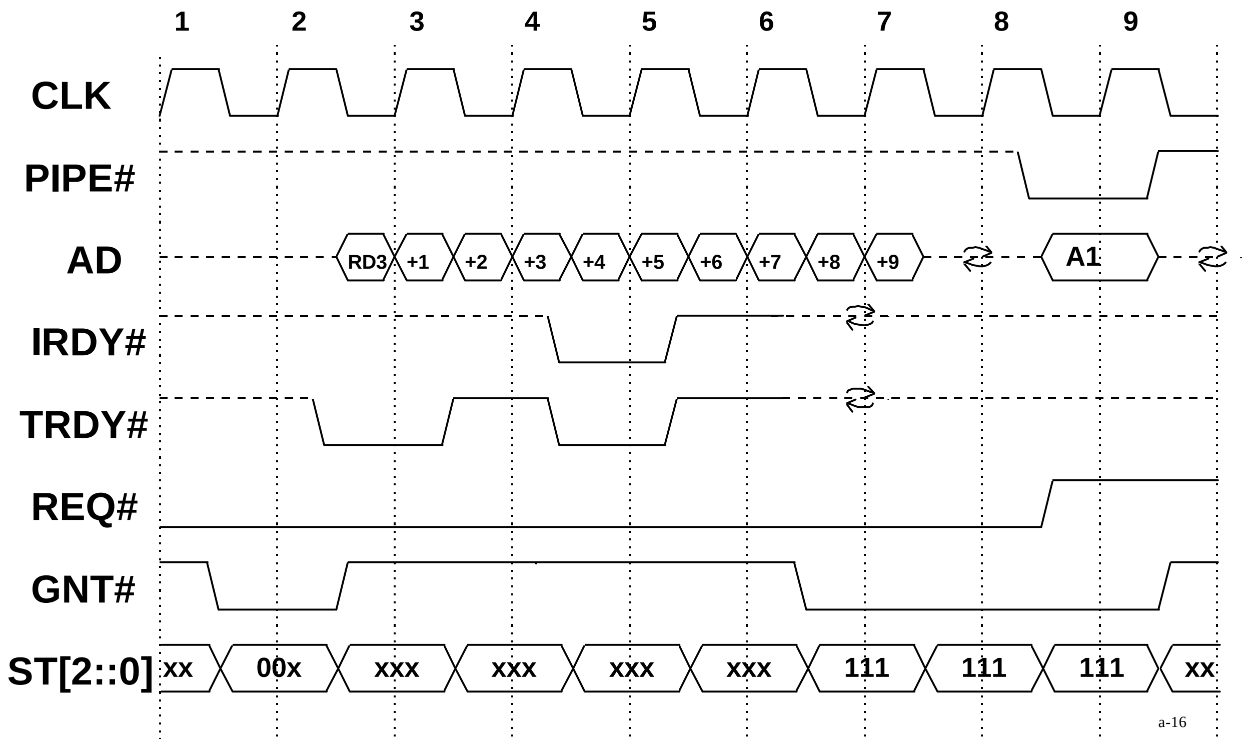
During an A.G.P. Read transaction the arbiter is not allowed to enqueue the next transaction until the read data transaction enters the last data phase.

An A.G.P. Read transaction followed by an A.G.P. Request. (A15, A16)
An A.G.P. Write transaction followed by an A.G.P. Request. (A17, A18)
An A.G.P. Read Transaction followed by a PCI (Read or Write) Transaction. (A19, A20)
An A.G.P. Write transaction followed by a PCI (Read or Write) Transaction. (A21, A22)



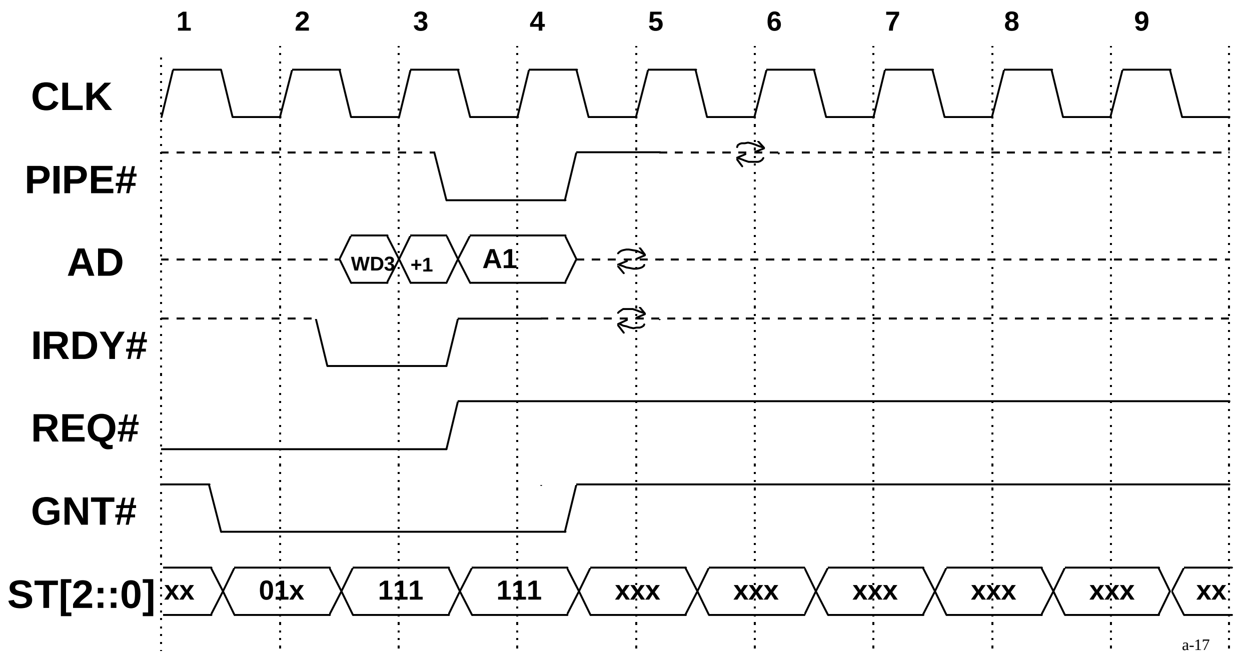
Read Data followed by PIPE#
AD turnaround required

Figure A-15 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. compliant 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.



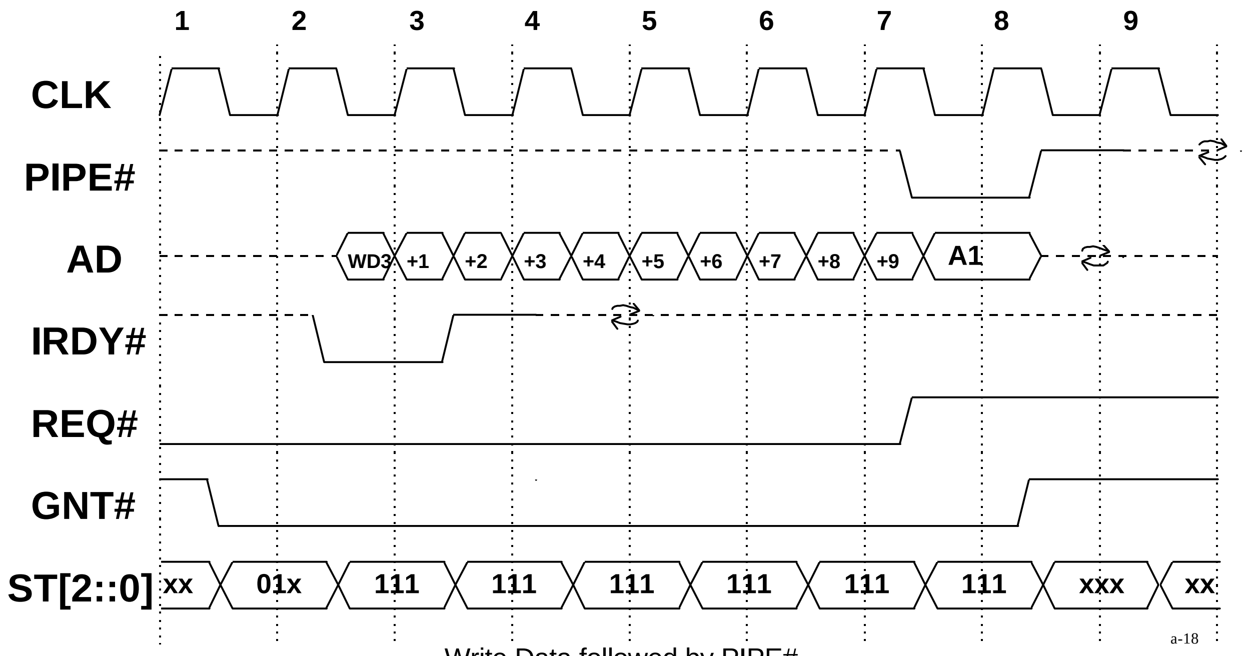
Read Data followed by PIPE#
AD turnaround required

Figure A-16 is the same as A-15 except that the data transfer last 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.



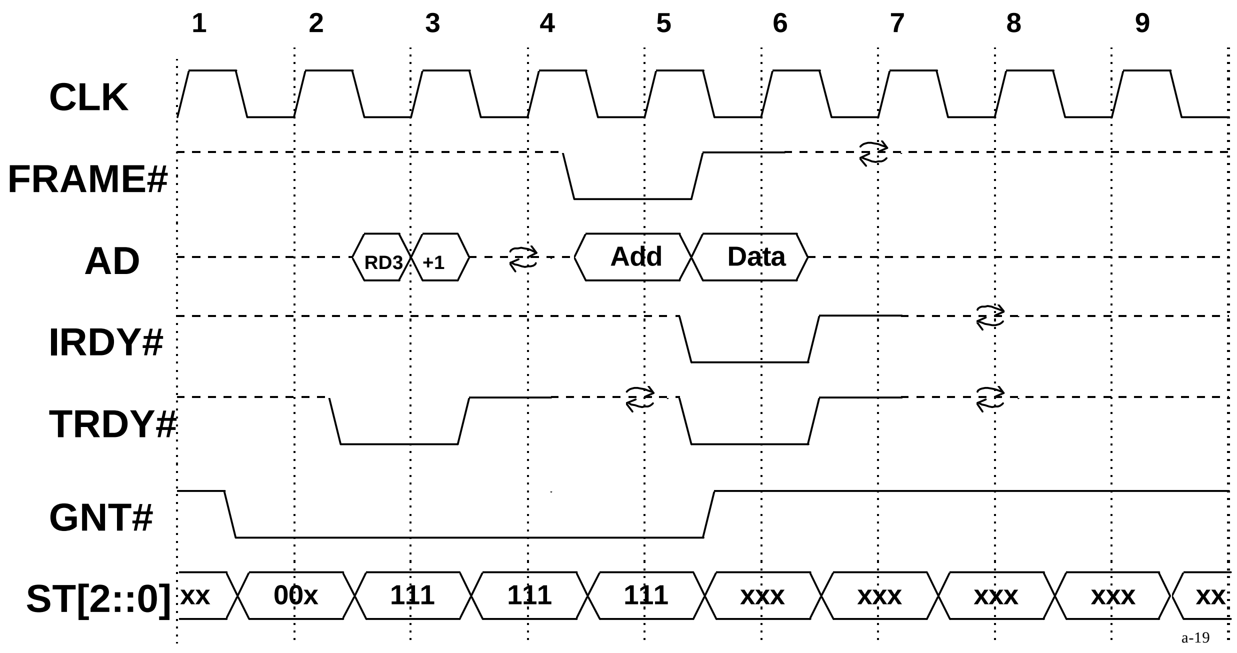
Write Data followed by PIPE#
No AD turnaround required

Figure A-17 is an A.G.P. write transaction followed by an A.G.P. Request. This figure is similar to figure A-15 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 up to a maximum of 4 transactions outstanding. In this case, there is only one outstanding since the subsequent one was not a write.



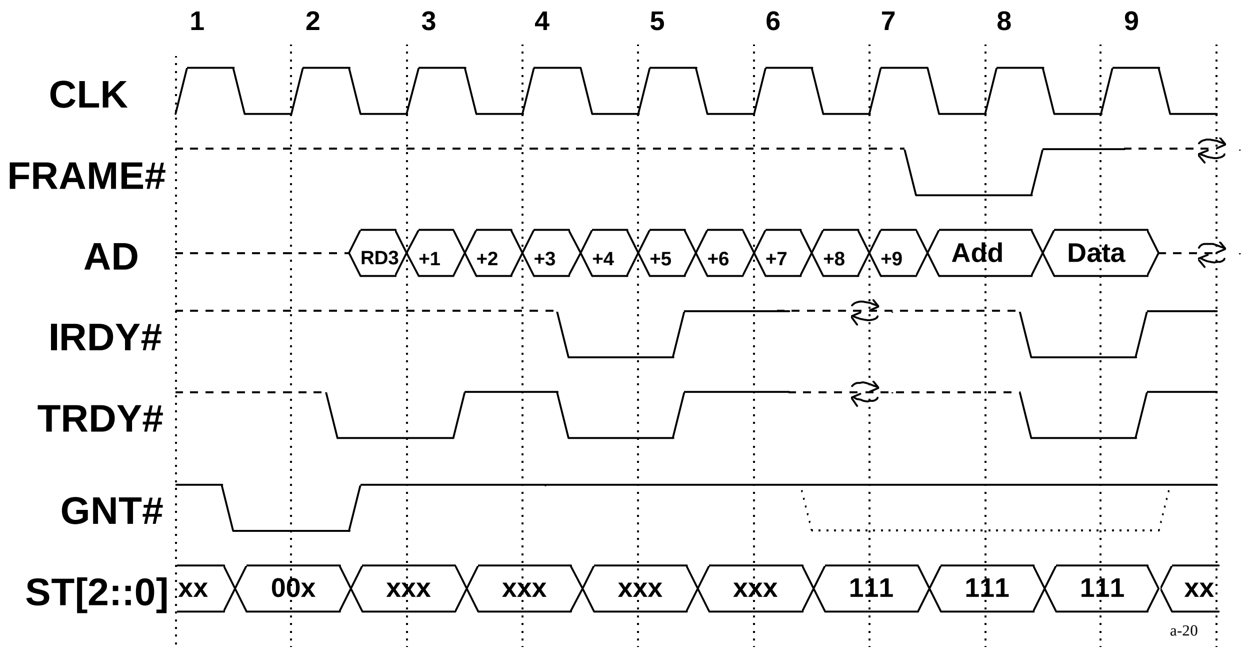
Write Data followed by PIPE#
No AD turnaround required

Figure A-18 is the same as A-17 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 anytime the master can not 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. compliant 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.



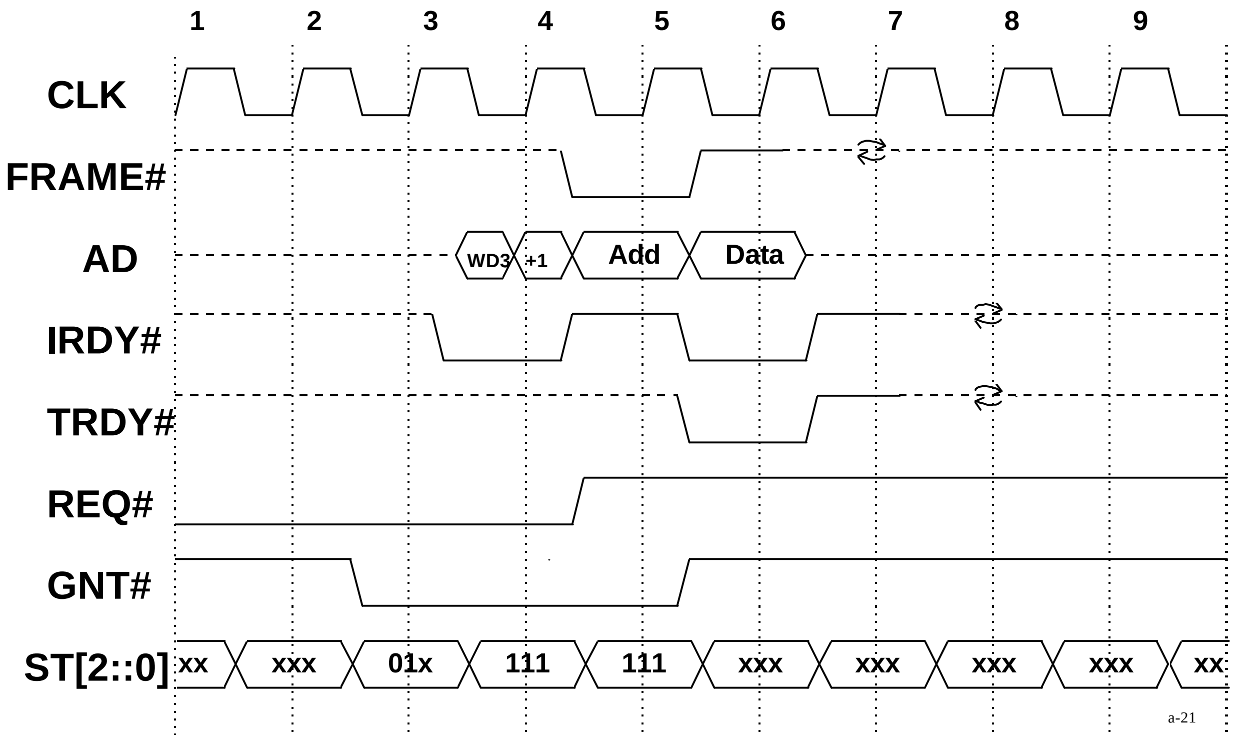
Read Data followed by an AGP generated PCI Cycle
TRDY# turnaround required

Figure A-19 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 **GNT#** on clock 4 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. compliant master initiating a PCI transaction) and **FRAME#** and **IRDY#** are deasserted. This condition occurs on clock 4 and the master asserts **FRAME#** and the address phase completes on clock 5.



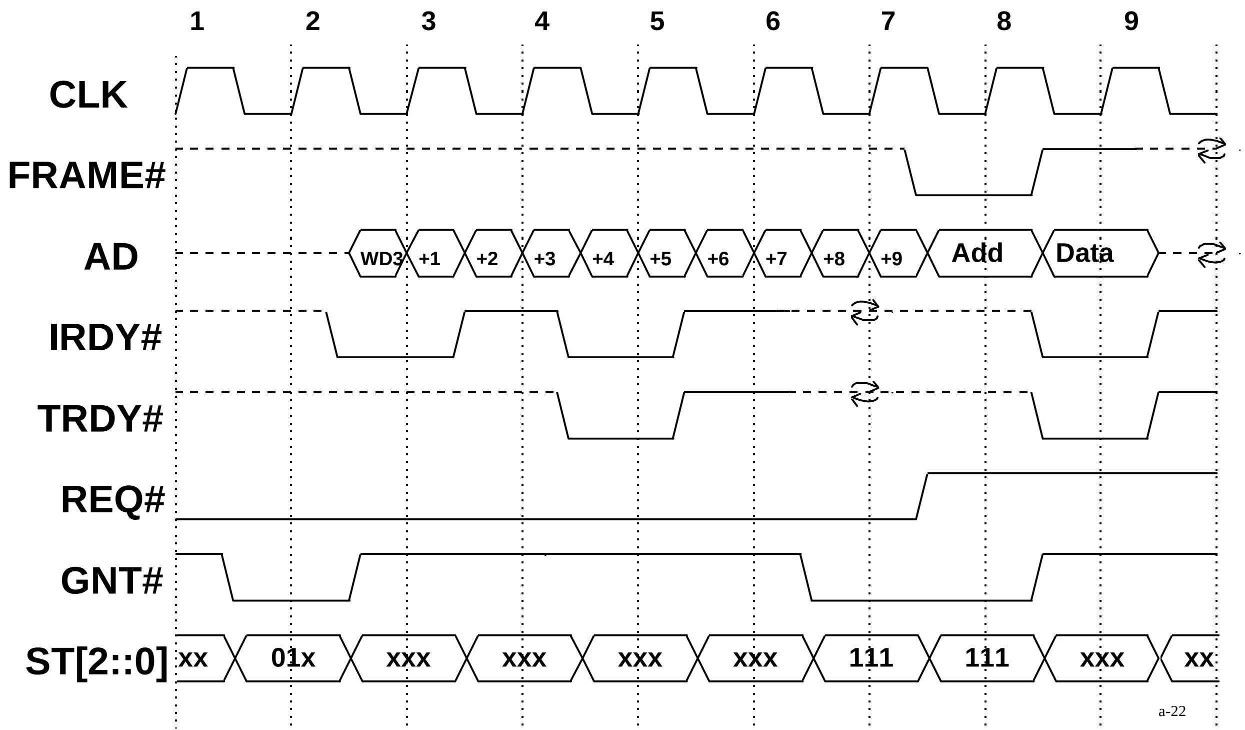
Read Data followed by an AGP generated PCI Cycle
No AD turnaround required

Figure A-20 is an example of an A.G.P. read followed by a PCI transaction. In this case the PCI master is allowed to start the request without a turn-around transaction because 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. 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 A-20 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.



Write Data followed by an AGP Master generated PCI Cycle
No AD, IRDY# turnaround required

Figure A-21 is an example where a turn-around cycle is not required between an A.G.P. and PCI transaction. In this case, ownership of the **AD** bus does not change. In this case, the A.G.P. compliant master is initiating a PCI transaction after providing data to the corelogic. In this case, 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. 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 it 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. compliant master doing a PCI transaction).



Write Data followed by an AGP Master generated PCI Cycle
No AD turnaround required

Figure A-22 is the same as A-21 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.

A data transfer followed by a data transfer.

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

Editorial Note: The above transactions are already covered in the specification and do not change. A summary of the rules needs to include the follow ideas.

1. The A.G.P. compliant master that has its **GNT#** asserted with **ST[2::0]** = 111, must not remember that **GNT#** was asserted. But must use the current version of **GNT#** to decide if it can initiate a transaction or not.

2. The A.G.P. compliant master when initiating a PCI transaction must follow the PCI 2.1 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. compliant master initiating a PCI

transaction is not allowed to start 1 or 2 clocks later as an A.G.P. compliant master is allowed making an A.G.P. request.

3. The A.G.P. compliant 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. compliant master is required to deassert **REQ#** for 2 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. compliant master when initiating an A.G.P. request is required to initiate the request within 2 clocks of when **GNT#** is asserted and **ST[2::0]** is 111. (This is the same as 1.0 and is not a change.)