Apple II Technical Notes

Developer Technical Support

Apple IIe
#2: Hardware Protocol for Doing DMA

Revised by: Glenn A. Baxter & Rob Moore
November 1988
Written by: Peter Baum
January 1983

This Technical Note explains the hardware protocol for doing direct memory access (DMA) on the Apple IIe and Apple][and is meant as a guideline for developing peripherals which do DMA on these machines, not as a specification for future Apple products or revisions.

This Note covers the timing differences between the Apple][and IIe and also gives tips on how to design a peripheral card that will work in both systems. The reader should be very familiar with either the Apple][+ or the Apple IIe, especially the timing on the data and address buses in relation to the 6502.

DMA is used by peripheral cards in the Apple II family to transfer data directly into memory without benefit of the processor. Transfer of data from a peripheral device into RAM can normally be handled one byte at a time under control of the processor. By using DMA, you can achieve greater data transfer rates than the 6502 can handle in software. This transfer rate can approach the full-cycle time of the memory. This technique can also be used to transfer single data bytes into memory without requiring the CPU to process an interrupt, which can be very time consuming.

The DMA process entails five steps: turn the processor off, gain access to the R/W* line and both address and data buses, complete the data transfer, release the data and address buses, and finally, allow the microprocessor to restart. This Note covers each of these steps in detail.

At this point, I should caution the prospective developer that DMA on an Apple][+ or Apple IIe can only be done under certain circumstances. Because DMA turns off the processor, any program with a software timing loop will not work properly. These programs assume that each instruction will take a fixed amount of time, which is not true when the processor stops in the middle of an instruction. This assumption means that the Apple II disk drives will not work since they require a timing loop to read a disk. (Co-processor cards work with DMA because they initiate the disk access and know that DMA cannot be used until the disk is finished).

Another problem is that because of the mapping scheme used on the Apple IIe extended 80-column (64K) card, a peripheral card cannot tell which memory bank is being used without a complicated detection scheme. This problem means that if a DMA device writes to a certain memory space, it might not be able to read the same data back.

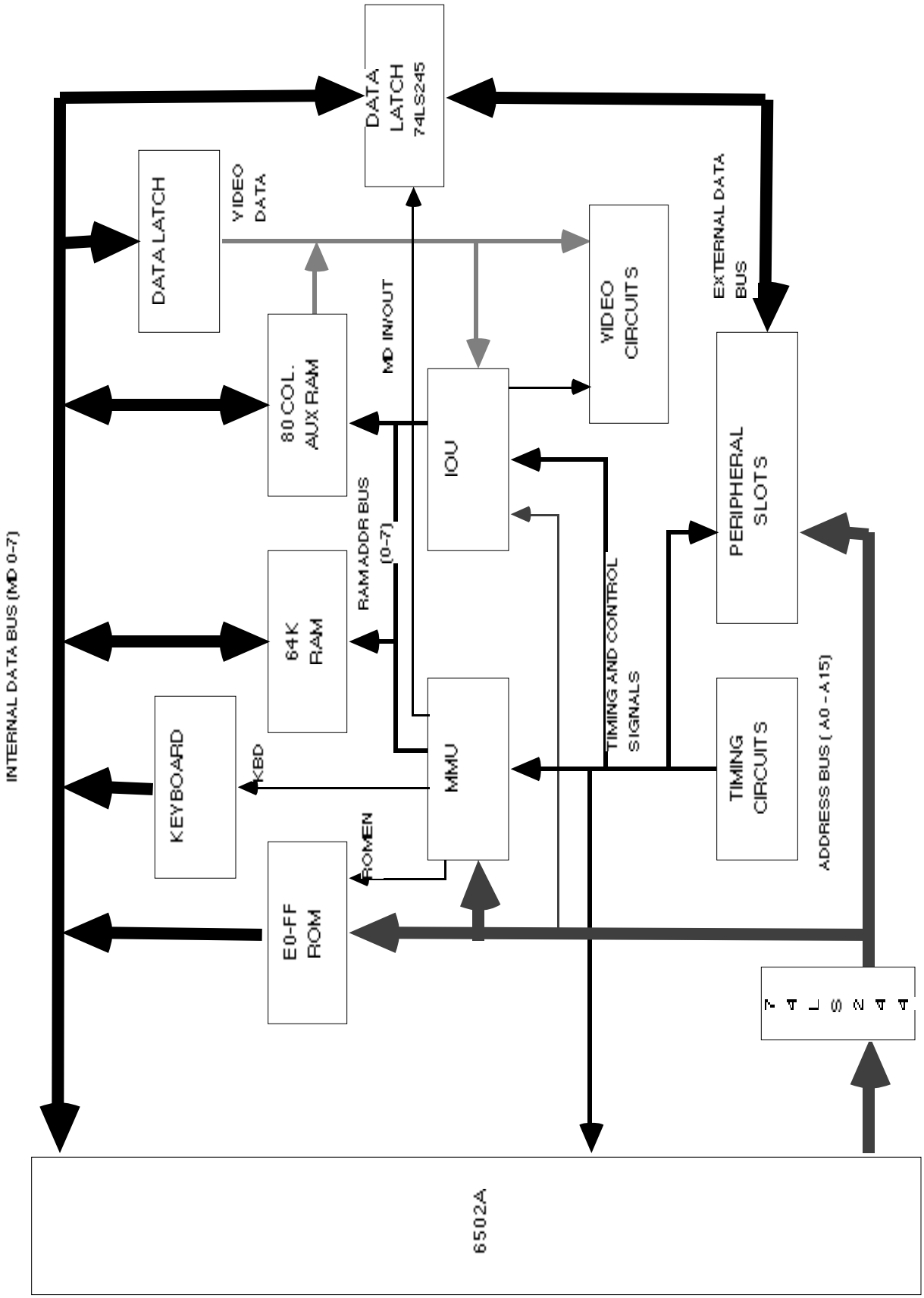

Figure 1–Apple IIe Functional Block Diagram

Though the differences between the Apple IIe and Apple][+ architecture appear to be significant to a device which uses DMA, this should not affect the design in most cases. A good rule of thumb is that if a device is designed to work on the Apple IIe, then it will be backward compatible and also run on the Apple][+. The converse is not true; cards that use DMA on the Apple][+ might not work on the Apple IIe, hence, most of the descriptions in this Note refer to the Apple IIe with occasional references to the Apple][+. For example, the timing specifications listed are calculated from the Apple IIe timing paths unless otherwise noted.

Occasionally the descriptions refer to a chip on the motherboard of the Apple IIe, so a set of Apple IIe schematics should be nearby. The corresponding parts on the Apple][+ will be specified when applicable.

The following paragraphs describe and define some of the terms that are used throughout this Note. The Apple IIe block diagram on the previous page may be helpful when reading about the buses.

01 (phase one) time — The time when the 01 system timing signal is high. During this time the data bus, address bus, and RAM are used for video refreshing. This time is also called the video cycle or video phase.

0o (phase zero) time — The time when 0o clock is high. 0o is the inverse of 01. During this time the microprocessor uses the data and address buses. This time is also known as the CPU cycle or CPU phase.

IOU and MMU — Two MOS custom chips inside the Apple IIe. See chapter 7 of the *Apple IIe Technical Reference Manual* for more details on the custom chips.

Data bus — The microprocessor, ROM, and RAM are connected to this bus. On the IIe this bus generally has MOS components connected to it rather than TTL and is sometimes called the MOS data bus. A 74LS245 bidirectional bus transceiver

(location B2 on the original motherboard) connects this internal bus to an external bus that the outside world sees through the peripheral slots. The data bus connected to the peripheral slots is called the external data bus. The Apple][does not have these two data buses. Instead, the peripheral slots are connected to the ROM, CPU data buffers, and RAM data inputs. The RAM data outputs are multiplexed with the keyboard data onto this bus.

Address bus

There are three different sections to the address bus on the Apple IIe. The first section consists of the addresses from the 6502A into a pair of 74LS244s (locations B1,B3). Part two connects the other side of the '244 to the address bus that the peripheral slots see. Also connected on this bus are the MMU, the ROM, and the chips that decode `I/O SELECT`, `DEVICE SELECT`, and `I/O STROBE`. The third address bus is generated by the custom chips and is only used to access the RAM. The MMU and IOU automatically multiplex this bus with the high byte and low byte of an address during any RAM access, whether it be for video refresh or for a microprocessor instruction fetch. This third bus is called the RAM address bus. The Apple][also has these three buses, but uses 8T97s and discrete logic instead of the 74LS244 and custom chips.

6502 microprocessor

In the Apple IIe a 6502A, a 2 MHz part is used instead of the 1 MHz 6502 used in the Apple][+. Since the custom chips in the Apple IIe are MOS and slower than the TTL in the Apple][+, the faster 6502A was used to guarantee better margins. For example, the 6502A sets up the address bus faster on the Apple IIe than the 6502 does in the Apple][+.

On the IIe, all the timing signals are generated by the PAL timing chip, except for the 7 M signal which is generated from an 74S109 or 74109 (early versions of the IIe). Although both the PAL and the 74S019 use the 14 M signal for a clock, there will be some skew between edges of the 7 M clock and the timing signals from the PAL, such as the edges of 0o or 01. This skew means the 7 M clock edge may rise as much as 20 ns before, or 5 ns after, the 0o falling edge. The clock signals of the Apple] [+ should be tighter than this (probably within 5 ns of each other) since 7 M, 0o, and 01 are all generated from the same chip, a 74S175. Take this skew into account whenever using the 7 M signal in a design.

Getting on the Bus (Exact Change Only)

1. Pull DMA low during 01 time.

   On the Apple IIe, the DMA line controls the direction of the 74LS245, which enables the internal data bus outwards to the peripheral slots or enables external data onto the internal bus. Changing the state of the DMA line during 0o could cause the '245 to change directions, forcing the internal data bus to go tri-state during a microprocessor read. The 6502 would read garbage and the computer might go belly up by jumping to a random memory location.

   On the Apple][, pulling the DMA line always forces the CPU data bus buffer to point inward and drive toward the 6502. Pulling the DMA line low during 0o of a write cycle would result in garbage being written to memory, since the data bus to the RAM would suddenly go tri-state.

2. Wait 30 ns, then assert address bus and R/W* line.

   Before driving the address bus and R/W* line, the system must process the transition on the DMA line and release the bus. This requires:

25 ns	'LS244 output disable from low level
5 ns	'S02 low to high level output transition
30 ns	delay from DMA negative edge before driving address bus

   The 30 ns wait will also work on the Apple][, since it only needs 27 ns ('LS04 and 8T97).

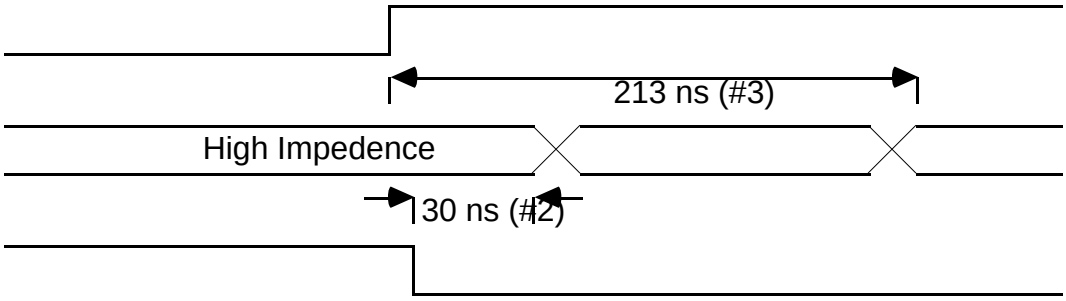

Figure 2–Getting On The Bus

3. Address and R/W* line must be valid within 213 ns of 01 positive edge.

   This constraint is needed to meet the setup requirements of the IOU, MMU, and RAM. This time can be derived from the 6502A (2 MHz) setup requirements. The

Apple][can wait for 300 ns before data must be valid, because it uses the 1 MHz 6502 which has a longer setup time.

Warning: This specification (the address setup time) is the major cause of failure for cards which use DMA in the Apple IIe. Many DMA cards which were originally designed for the Apple][+ do **not** meet this specification.

During DMA (Keep Your Hands Inside the Bus at all Times)

1. Don't drive the data bus during 01 time.

   On an Apple][+, it is safe to drive the data bus 35 ns after asserting the R/W* line low, regardless of the point in the timing cycle. When the R/W* line goes low, the 74LS257s at locations B6 and B7 tri-state the data bus, even in the middle of 0o or 01. This action prevents a bus fight from occurring between a DMA device and the system.

   At first glance of the Apple IIe logic schematics, it appears that a bus fight cannot occur on the data bus. During the 01 half of a write cycle, the 74LS245 tri-states the data bus within 30 ns of the R/W* line being pulled low. While this does preclude a fight from occurring on the data bus during 01, it doesn't prevent a bus crash from occurring at the beginning of 0o. At the beginning of 0o, the 74LS245 is enabled and will drive the external data bus. If the peripheral card also drives the data bus, there could be a horrendous bus fight, since the 74LS245 can source 15 ma and sink 24 ma per line. This might cause a spike on the ground plane, which could cause a processor to reset on a co-processor card.

   Let us take a look at the problem by stepping thru Figure 3, a timing diagram.

   The diagram starts with the video cycle of a read operation. During the video cycle, the video refresh data is read from the RAM and put on the data bus. This video data will appear on the peripheral slot (external) data bus because the 74LS245, as can be seen from Table 1, drives outward during 01 of a read cycle.

   Typically, the address bus and R/W* line would be setup by the 6502A during 01 for the next CPU cycle, but instead, a peripheral card pulls the DMA line low. As explained earlier, the peripheral device should wait at least 30 ns before driving the address bus and R/W* line. In this first DMA cycle, the peripheral card wants to read a byte from RAM, so it keeps the R/W* line high.

   At this point we must switch over and use the Apple][+ to explain the timing

required to read the data from RAM. The rule of thumb, that designing a DMA card for the Apple IIe will be backward compatible and run on the Apple][+, will not hold here. On the Apple][+ data is valid on the peripheral connector a minimum of 468 ns from the 0o rising edge and holds to at least the falling edge of 0o at 490 ns. The hold time is actually the minimum propagation delay from the falling edge of 0o thru the following chips: 74LS257 at J1, 74LS139 at F2, 74LS20 at D2, 74LS00 at A2, and finally to the enable of the 74LS257s at B5 and B6. On the Apple IIe a byte from RAM becomes valid at most 345 ns after the rising edge of 0o and stays valid until the 0o falling edge.

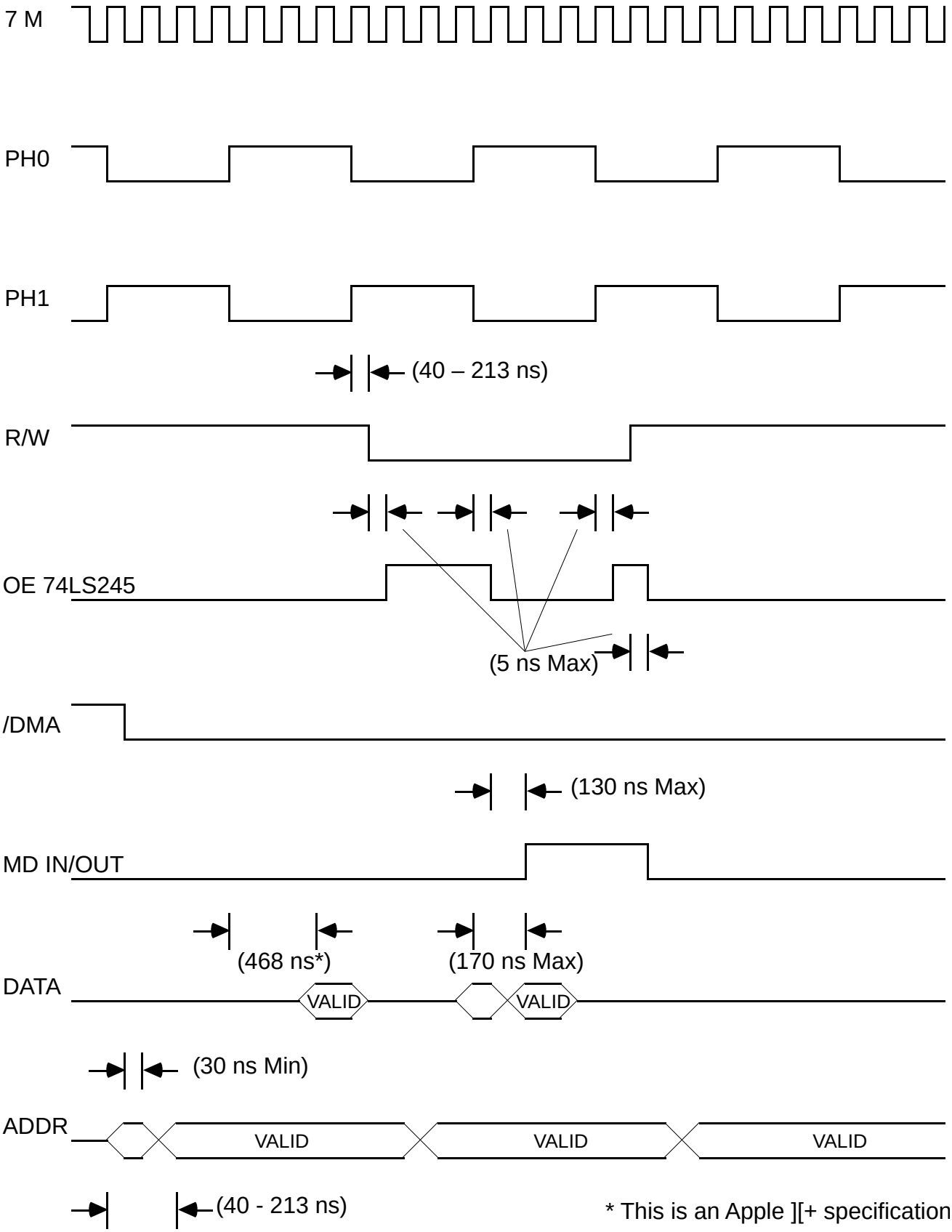

Figure 3–Timing Diagram

In the second DMA cycle, the timing diagram shows the peripheral card writing a byte to memory. In the first phase of the cycle, the video phase, the address bus and R/W* line are setup by the peripheral card within the timing specifications described earlier, 213 ns. Though the direction of the 74LS245 still points toward the slots, the ’245 is disabled when the R/W* line is pulled low by the peripheral device (see Table 1). This will tri-state the external data bus. All the signals stay unchanged through the rest of the video phase, until the CPU cycle starts with the rising edge of 0o.

Most bus fights occur at the beginning of the CPU cycle. The CPU cycle begins with address bus and R/W* line setup already and the data bus tri-stated. The signal `MD IN/OUT`, which drives the 74LS245 direction control, is generated by the MMU and is always low during 01, so the 74LS245 drives toward the slots. `MD IN/OUT` uses the 0o rising edge to clock itself high during a DMA write cycle, but because the MMU is a MOS chip the delay before `MD IN/OUT` finally rises can be as long as 130 ns from the 0o rising edge. Hence, at the beginning of 0o the 74LS245 is in tri-state mode, but with the direction set to drive toward the peripheral slots.

PHO	R/W		Stable State of 74LS245
1	0	(Write to RAM)	High impedance
1	1	(Read from RAM)	Outward driving external data bus (slots)
0	0	(Write to RAM)	Inward driving into RAM
0	1	(Read from RAM)	Outward driving external data bus (slots)

Table 1–Stable State of 74LS245

Within 5 ns after 0o goes high, the chip enable to the 74LS245 goes low, enabling data onto the external data bus. The 74LS245 specification guarantees that the data will be valid within 40 ns from the chip enable. If the peripheral device was also driving the bus, there would be a bus crash. To prevent this bus crash, the data bus cannot be driven during 01, unless the data is pulled off the bus before 0o goes high. This means that the rising edge of 0o cannot be used to gate data on and off the bus. The bus fight will occur before the peripheral card can tri-state the data bus.

Data can only be enabled onto the bus after the 74LS245 has changed directions and is driving the internal data bus. The DMA device must allow 130 ns for the `MD IN/OUT` line to change, plus the delay for the 74LS245 to change directions which takes 25 ns, for a total of 155 ns.

After this 155 ns, the data must be valid on the bus within 55 ns, because the RAM requires data be setup at the CAS falling edge, which occurs 210 ns into 0o. This does not leave any time to spare, since, for example, a 74LS245 has a 40 ns enable time. This timing criteria will also work for the Apple][+ since the setup time for 16K RAM is the same as the 64K RAM, and CAS also falls at 210 ns. The data hold time of 55 ns after CAS falls is also the same for both the Apple IIe and the Apple][+.

Here is a scenario for a DMA write. Set up the address bus and R/W* line within the required 213 ns, then wait for the first 7 M (pin 36 on slot) falling edge after 0o goes high before enabling your buffer onto the data bus. This edge will occur at 140 ns into 0o, and when the gate delay is added, should guarantee the buffer will

not be driving the bus in the first 155 ns. I don't advocate depending on a minimum gate delay as standard design practice (my college professor thinks public whipping would be a justifiable punishment) but this is the real world (I'm not getting graded anyway). The data bus is valid by the time CAS falls, and should be stable for at least another 55 ns or until 0o falls.

2. The processor can be held off for a total of 10 µs. (10 0o clock cycles).

   This is true if a Rockwell 6502 is being used. (A Synertek part can be held off for as long as 40 µsec.) This time is the maximum cycle time of the 6502 and if there are no clock transitions within this time, it could result in internal registers (A,X,Y) losing data. This maximum time varies from manufacturer to manufacturer of the 6502.

3. MMU and IOU multiplex address bus

   The custom chips automatically handle the multiplexing required of the RAM address bus. The external device doing DMA must set up the address bus and R/W* line within 213 ns of the rising edge of 01 just like the 6502A does. The custom chips will automatically generate the addresses to the RAM for the video refresh cycle during 01, but use the addresses from the address bus to set up for the next instruction cycle. Hence, the only consideration on the address bus during DMA is to meet the 213 ns setup time requirement.

   The 213 ns setup time will also work with the Apple][since it can take as long as 300 ns to set up the address bus and R/W* line.

Getting Off the Bus

1. Don't release DMA during 0o.

   This is analogous to step 1 of Getting on the Bus. If the DMA line is released during 0o the microprocessor will try to execute a cycle during this time without the data or address bus set up properly. This random instruction fetch will probably cause the system to crash.

2. Tri-state address bus drivers on peripheral slots

   The DMA line is holding off the addresses from the 6502 onto the internal address bus by tri-stating the two 74LS244s on the Apple IIe bus and the 8T97s on the Apple][+ bus. The address bus and R/W* line from the external device in the

peripheral slots should be tri-stated before releasing the DMA line or a bus fight will occur between the internal bus buffers and the peripheral slot drivers.

3. Release DMA line

These last two steps are the opposite of the first two steps required to get on the bus. Both of these steps, releasing the address and R/W* lines then the DMA line, should be done within 178 ns of 01 going high. This allows time for the 6502A to set up the address and R/W* lines properly for the next instruction cycle.

213 ns	address set up requirements
5 ns	’S02 output high-to-low transition
–30 ns	’LS244 out enable time
178 ns	to release DMA line and allow 6502 to set up address bus

Again, the Apple][can take longer, up to 260 ns, before releasing the DMA line.

Further Reference

- *Apple IIe Technical Reference Manual*