Macintosh
Technical Notes

Developer Technical Supp

NuBus Interrupt Latency
(I Was a Teenage DMA Junkie)

Hardware **M.HW.NuBusLatency**

Revised by: Cameron Birse October 1989
Written by: Cameron Birse, Mark Baumwell, & Rich Collyer December 1988

This Technical Note discusses NuBus™ interrupt latency, and why, contrary to popular belief, the Macintosh is **not** a real-time machine.

Changes since December 1988: Changed sample code to defer cursor rendering to a deferred task rather than a "pseudo-VBL" task.

The Macintosh is **not** a real-time machine. The Macintosh does **not** support DMA. There are many variables in the Macintosh that make it impossible to deterministically figure out exactly when things are going to happen. Despite these facts, there are those who must push the envelope. For these courageous adventurers, we provide the following information in the hope that it speeds your journey.

According to empirical evidence gathered by Apple engineering, typical NuBus to Macintosh transaction times fall in the 800 nanosecond to 1 microsecond range. Although the NuBus specification points to faster accesses, you should consider these times realistic since there is always some overhead. Synchronizing the NuBus and Macintosh clocks, for example, can cost a NuBus cycle.

One technique that can help optimize NuBus transfers is implementing bus locking. The bus can be locked for a small set of transactions (we recommend a maximum of four transfers), then unlocked for rearbitration. In order to allow fairness, it is important to lock the bus for as short a time as possible.

All processor interrupts and slot interrupts may be held off for various amounts of time by different parts of the system, so you must never count on instant interrupt response. To help deal with these delays, you should consider your data rate and include ample buffering on your card for your data. The following are just a few of the many system variables which affect interrupt latency:

- Floppy disk accesses turn off interrupts for "significant" (read milliseconds) amounts of time. For instance, some disk accesses (i.e., block reads) can disable interrupts for as much as 15 milliseconds. Inserting a blank floppy disk turns off interrupts for up to 25 milliseconds.

- Formatting a floppy disk turns off interrupts for up to 300 milliseconds.
- LocalTalk accesses can disable interrupts for up to 22 milliseconds.
- Assuming your interrupt handler is going to want to access your card immediately, there is also the arbitration for mastership of the bus, which could be in use at the time, and in the worst case, lock the bus, keeping you from accessing your card.

- All slot interrupts, including slot VBL interrupts, hold off other slot interrupts. This means another card's interrupt routine (installed via `_SIntInstall`) or a slot VBL interrupt routine (installed via `_SlotVInstall`) runs to completion with interrupts of the slot level and below disabled. VBL tasks may be of varying length, since applications, as well as drivers, can and do, install VBL tasks.
- Cursor updating (performed during slot VBL time) time ranges from around 700 µSec - 900 µSec for one-bit to eight-bit depth. Since this is done at slot VBL time, it holds off all other slot interrupts until it is finished.

Warning: The performance figures cited in this Note are based on current Macintosh models; they are not guaranteed to remain the same in future machines.

The following code lets you defer the cursor updating routine by having it run as a deferred task. This change means that the actual cursor rendering is performed with interrupts enabled, which allows the occurrence of other interrupts. It should be noted that there is a slightly visible flickering of the cursor as a result of using this technique.

```
;*****************************************************************
;***
;***     Defer Cursor
;***     This program defers the cursor updating that normally happens
;***     during slot VBL time. Since the cursor updating can take as
;***     long as 900µSec, and holds off other slot interrupts, it is
;***     handy to be able to defer the updating to a more civilized time.
;***     This program replaces the normal jCrsrTask with a routine that
;***     installs the real jCrsrTask routine as a deferred task.
;***
;***     Build commands:
;***
;***     asm DeferCrsr.a -lo DeferCrsr.a.lst -l
;***     link DeferCrsr.a.o -o DeferCrsr
;***
;*****************************************************************

                                STRING ASIS
                                PRINT  OFF
                                INCLUDE 'Traps.a'
                                INCLUDE 'SysEqu.a'
                                PRINT  ON

;------------------------------- Entry -------------------------------

Entry  MAIN

       bra.s        Entry2

;---------------------------- MyDefTask ----------------------------

TaskBegin
MyDefTask

             DC.L   0      ;qLink (handled by OS)
             DC.W   0      ;qType (equ 7, find this value in MPW AIncludes)
             DC.W   0      ;dtFlags (reserved, don't mess with 'em)
             DC.L   0      ;dtAddr (pointer to actual routine to be performed)
             DC.L   0      ;dtParm (optional parameter, this example doesn't use it)
```

```
                DC.L   0      ;dtReserved (should be zero, DC.L 0 takes care of that)

SysCrsrTask
                DC.L   0

***************************** MyjCrsrTask ****************************

MyjCrsrTask
       movem.l       a0/d0,-(sp)
       lea           MyDefTask,a0               ;point to our deferred task element
       move.l        SysCrsrTask,dtAddr(a0)     ;set up pointer to routine
       move.w        #dtQType,dtType(a0)        ;set queue type
       _DTInstall                               ;install the task
       movem.l       (sp)+,a0/d0
       rts
TaskEnd

****************************** Entry2 ******************************

TaskSize      EQU           TaskEnd-TaskBegin

Entry2
       move.l        #TaskSize,d0         ;TaskSize = Deferred task element, room for
                                          ; a pointer (to original jCrsrTask), and
                                          ;our jCrsrTask
       _NewPtr       ,SYS,CLEAR           ;make a block in the system heap
       bne.s         Abort                ;no room at the Inn, head for the manger
       move.l        a0,a2                ;got a good pointer, keep a copy
       move.l        a0,a1                ;a0 = source, a1 = destination for
                                          ; BlockMove
       lea           MyDEFTask,a0         ;copy the task, etc. into the system heap
       move.w        #TaskSize,d0
       _BlockMove

       lea           dtQElSize(a2),a0     ;move original jCrsrTask pointer into our
       move.l        jCrsrTask,(a0)       ; pointer holder
       lea           dtQElSize+4(a2),a0   ;replace jCrsrTask pointer with a pointer
       move.l        a0,jCrsrTask         ; to our jCrsrTask

abort  rts                                ;all's well that ends…

       END
```

- Note, as an aside, that while using MacsBug, interrupts are disabled.

In summary, you cannot depend on real-time performance when transferring data between NuBus and the Macintosh. It is important to provide sufficient buffering on the card to allow for the variance in interrupt latency. Driver calls can be used to determine the amount of data available to be transferred, and transfers can be made on a periodic basis.

Remember too, since the entire system is so heavily interrupt-driven, it is very unfriendly for anyone to disable interrupts and take over the machine for long periods of time. Doing so almost always results in a sluggish user interface, something which is usually not well received by the user.

Further Reference:

- *Inside Macintosh*, Volume V, The Device Manager
- *Inside Macintosh*, Volume V, The Vertical Retrace Manager
- *Macintosh Family Hardware Reference*
- *Designing Cards and Drivers for the Macintosh II and Macintosh SE*

NuBus is a trademark of Texas Instruments