#221: NuBus Interrupt Latency (I was a teenage DMA junkie)

See also: *Inside Macintosh*, Volume V, The Device Manager
Inside Macintosh, Volume V, The Vertical Retrace Manager
Macintosh Family Hardware Reference
Designing Cards and Drivers For Macintosh II and Macintosh SE

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

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

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 will speed your journey.

According to empirical evidence gathered by Apple engineering, typical NuBus to Macintosh transaction times fall in the 800 ns to 1 microsecond range. Although the NuBus specification points to faster accesses, you should consider these times realistic since there will always be 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 worst case, lock the bus, holding you off 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`) will run 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 700µSec - 900µSec 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.

  The following code will let you defer the cursor updating routine by having it run at the "pseudo-VBL time." This changes the cursor update to a lower priority interrupt, which then will allow slot interrupts to occur. It should be noted that there is a slightly visible flickering of the cursor as a result of using this technique.

Warning: The numbers cited above are based on our current implementations, and are not guaranteed to be the same in future machines.

```
*****************************************************************************
***
***     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 an RTS, and installs
***     a VBL task that calls jCrsrTask routine every VBL.
***
***     Since the normal VBLs are a level of interrupt lower than slot
***     interrupts, the cursor updating will not hold off slot interrupts.
***
***     This is a one way street. A real implementation should restore
***     the regular jCrsrTask when this scheme is no longer needed.
***
*****************************************************************************

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

********************************* Entry *************************************

Entry   MAIN
       ALIGN 2
       bra.s           Entry2
***************************** MyVBLTask ***************************
```

```
TaskBegin
MyVBLTask

       move.w      #1,vblCount(a0)
       move.l      TaskEnd,a0
       jmp         (a0)

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

MyjCrsrTask
       rts
TaskEnd

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

TaskSize    EQU     TaskEnd-TaskBegin
VBLEntry    EQU     TaskSize+4                  ;keep the old jCrsrTask pointer between the
                                               ; tasks and VBLTask record

Entry2
       move.l      #TaskSize+18,d0             ;18 bytes = 1 Pointer + VBLTask record
       _NewPtr     ,SYS,CLEAR                  ;make a block in the system heap
       bne.s       Abort
       move.l      a0,a2                       ;save the pointer
       move.l      a0,a1                       ;a0 = source, a1 = destination
       lea         MyVBLTask,a0                ;now copy the two tasks over
       move.w      #TaskSize,d0
       _BlockMove

       move.l      a2,a0                       ;restore the pointer
       move.l      jCrsrTask,TaskSize(a0)      ;put the system jCrsrTask pointer before
                                               ; the VBL record
       adda.l      #MyjCrsrTask-TaskBegin,a0   ;make a pointer to our jCrsrTask
       move.l      a0,jCrsrTask                ; and install it

       adda.l      #VBLEntry,a2                ;now point to the VBLTask Record
       move.w      #vType,qType(a2)            ;look for qType, etc. in AIncludes
       move.l      a1,vblAddr(a2)              ;point to our task
       move.w      #1,vblCount(a2)             ;do it once every 1/60th second
       move.w      #0,vblPhase(a2)             ;no phase
       move.l      a2,a0                       ;a0 must point to VBL task record
       _VInstall

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 will generally result in a sluggish user interface, something which is usually not well received by the end user.