Apple II Technical Notes

Developer Technical Support

Apple IIGS
#69: The Ins and Outs of Slot Arbitration

Revised by: Matt Deatherage May 1990
Written by: Matt Deatherage September 1989

This Technical Note discusses the concept of a 14-slot Apple IIGS system through dynamic software slot arbitration. It presents concepts of which all IIGS programmers should be aware for full compatibility.
Changes since September 1989: Removed the section which stated that this Note showed how to switch slots in a way that does not interfere with slot arbitration and replaced it with the proper description, which is how to search a 14-slot system for peripherals and their identification bytes.

History

The Apple II has always had seven slots. In some cases (e.g., IIe), one of the slots was handled specially by the hardware, or (e.g., IIc) there was no hardware present for peripheral cards at all. But there have always been seven "slots" with firmware at location $Cn00 (where n is the slot number). If there was no firmware, there was no peripheral connected.

With the introduction of the Apple IIGS, the Apple II family saw its first 14-slot system. Seven hardware slots are provided for peripheral cards (like on the IIe), and seven internal "ports" with connectors on the back panel are provided by the system (like on the IIc). Since $C800 and above cannot be used for additional slots (that space is shared between all interface cards), each of the seven internal ports is matched with one of the slots, and either the port or the slot is enabled at any given time. The IIGS hardware allows switching between the two, so all fourteen slots could be used more or less simultaneously.

This situation posed a problem—the Apple II had only a disk operating system, not an overall operating system. Access to non-disk devices (i.e., character devices, like a serial card) was not arbitrated by the system in any way. The world was used to seven, and only seven, slots. Attempting to use more in a shared system such as the IIGS resulted in somebody jumping to slot firmware that somebody else had switched out. This tended to crash the system.

Then came GS/OS. With its centralized mechanism for dispatching to **all** devices connected to a system, GS/OS provides hope (for the first time) that a central routing mechanism can dynamically arbitrate between slots and ports, allowing the use of all 14 at one time. This is called dynamic slot arbitration, and is handled by a portion of GS/OS referred to as the Slot Arbiter.

Although the Slot Arbiter does not function in System Software 5.0 or earlier, it may function in the future. A skeleton is present in version 5.0 and later that accepts Slot Arbiter calls, but the skeleton does not actually switch any slots. This Note details the Slot Arbiter functionality and shows how to search a 14-slot system for peripherals and their identification bytes.

Note: The Slot Arbiter must **not** be used unless GS/OS is the current operating system.

The Slot Arbiter

The Slot Arbiter is accessed through the GS/OS system service call vector DYN_SLOT_ARBITER ($01FCBC). On ROM 03 and later, the vector is duplicated at $E10208. Entry to the Slot Arbiter is via a JSL instruction, and exit is via RTL. The parameters are as follows:

```
Entry:
   A = Slot to be selected (defined below)
   X = Undefined (or Bit Encoded Slot Configuration)
   Y = Undefined
   B = Undefined
   D = Undefined
   P = N V M X D I Z C  E
       x x 0 0 0 x x x  0

Exit:
   A = Error Code
   X = Bit Encoded Slot Configuration
   Y = Undefined
   B = Unchanged
   D = Undefined
   P = N V M X D I Z C  E
       x x 0 0 0 x x 0  0        If A = $0000 (no error)
       x x 0 0 0 x x 1  0        If A = $0010 (slot not available)
```

The slot number in the A register tells the Slot Arbiter what you are requesting. Bits 0-2 are the slot number in the range 0 through 7. Bit 3 is set if you are requesting an external slot and clear if you are requesting an internal port. Taken together, bits 0-3 give slot numbers of $0-$7 for internal ports and $9-$F for external slots. This is the same way that slot numbers are returned by the GS/OS DInfo command.

Bits 8 and 9 of the slot number indicate the action you wish the Slot Arbiter to take. A value in these two bits of 00 asks the Slot Arbiter to switch in the slot identified in bits 0 through 3. If both bits are set to 11, the Slot Arbiter restores all the slots to match the Bit Encoded Slot Configuration present in the X register. Bit Encoded Slot Configurations are discussed in the next section of this Note. Values other than 00 or 11 in bits 8 and 9 are **reserved** and must **not** be used by applications.

Bit 15 of the slot number is set if the slot selection has no slot dependencies. When the Slot Arbiter is asked to switch in a slot with no slot dependencies, it does no actual switching, although it returns a Bit Encoded Slot Configuration in the X register. The slot number and the definitions of the individual bits are illustrated in Figure 1.

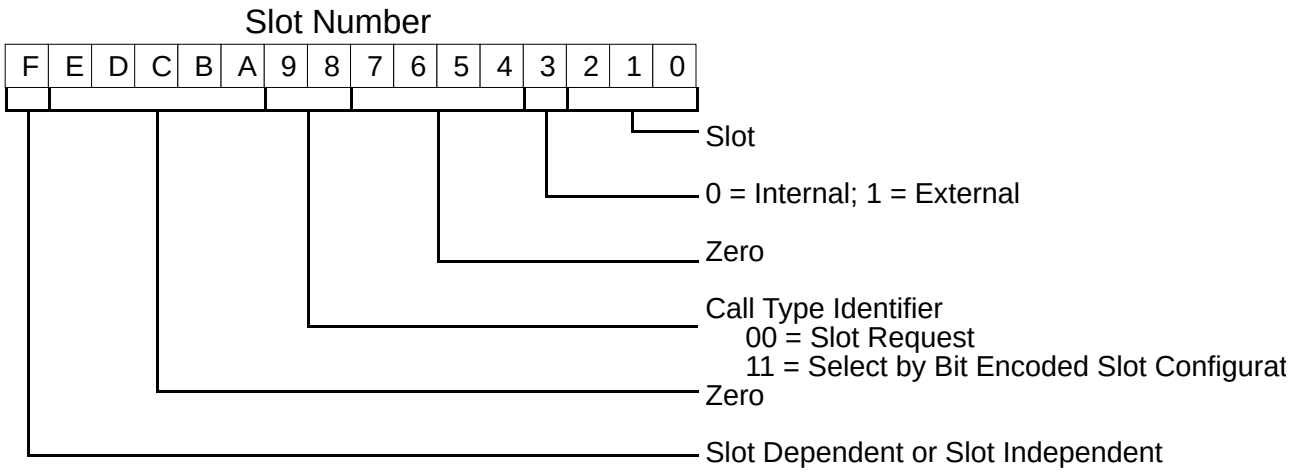

Figure 1–Slot Number and Bit Definitions

Bit Encoded Slot Configurations

Every call to the Slot Arbiter returns (on exit) a miniature picture of the slot configuration in the X register (as it was on entry). This picture has one bit set for each of the 14 slots; if the bit is set, then the corresponding slot is switched in. Bits 0 and 8 are reserved and are always clear. This picture is called a Bit Encoded Slot Configuration.

Since each external slot has the same number as an internal port (with bit 3 set), and since such pairs share the same address space, it follows that both of them may not be enabled at the same time. For example, port 5 and slot 5 ($D) both may not be enabled. This makes the high byte of the Bit Encoded Slot Configuration the eXclusive-OR of the low byte (excluding bits 0 and 8, which are always clear). Figure 2 illustrates the Bit Encoded Slot Configuration.

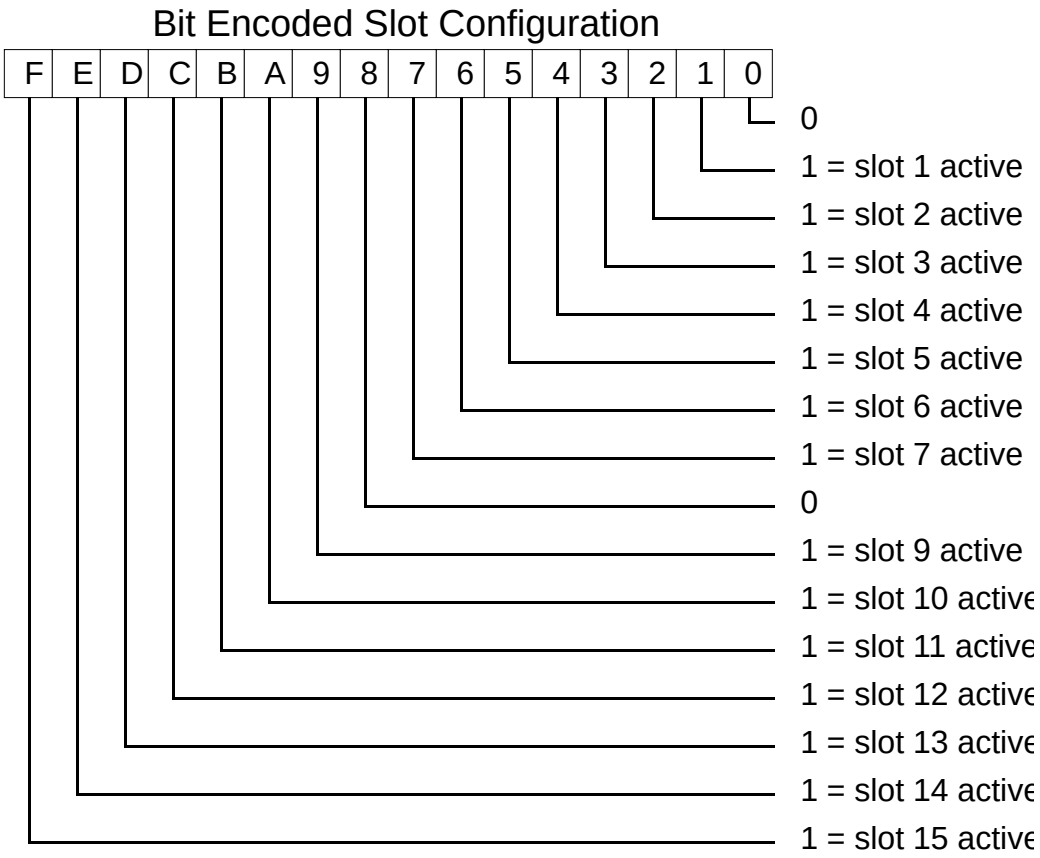

Figure 2–Bit Encoded Slot Configuration

By fully using the slot number parameter, the Slot Arbiter returns any aspect of the current slot configuration. Following are a few examples:

Slot number	Action Taken by Slot Arbiter
$8000	Returns current Bit Encoded Slot Configuration in the X register. This number asks the Slot Arbiter to switch in with no slot dependencies (no switching), so it just returns the Bit Encoded Slot Configuration.
$0300	Restore from Bit Encoded Slot Configuration. This command, when paired with the one above, can be used to save and restore a slot environment.
$0005	Asks the Slot Arbiter for internal port 5.

The Impact on Applications and Drivers

Applications which correctly do all input and output through GS/OS are affected by slot arbitration, except that they find more devices available. GS/OS uses the slot number parameter in the Device Information Block to call the Slot Arbiter, making sure the slot is available for the device before it gets control. However, there are some applications (such as peripheral card configuration programs) which go directly to firmware or hardware, not using GS/OS. Perhaps the card has no ROM, so there is no generated driver, or perhaps there is no loaded driver and the generated driver does not control certain aspects of the hardware. In any case, such applications are directly impacted by slot arbitration.

Slot Searching

The first problem is finding the hardware. In a 14-slot system, it's not suitable to just look for ID bytes between $C100 and $C700—two peripherals may be sharing each of those pages of slot ROM space. Drivers must examine all 14 slots, with the aid of the Slot Arbiter. The following sample code demonstrates this technique:

```
find_slot       lda    #$8000            ; request current Bit Encoded Slot Configuration
                jsl    slot_arbiter
                phx                      ; save it on the stack

                lda    #$000F            ; start with slot 15
                sta    slot_number       ; be sure of the data bank when doing this!

slot_search     lda    slot_number       ; get the slot number to examine
                jsl    slot_arbiter      ; and ask for it
                bcs    continue_search   ; if an error, then don't look here
                jsr    check_for_hw      ; this routine looks for your hardware
                bcc    found_my_hw       ; if found it, we're done searching
continue_search dec    slot_number       ; try the next lower slot
                bpl    slot_search       ; (if there are any left, of course)

found_my_hw     plx                      ; get Bit Encoded Slot Configuration from stack
                lda    #$0300            ; and tell the Slot Arbiter to restore from it
                jsl    slot_arbiter

; We're done.  Our slot number is in the location slot_number.
```

Note: You **must** restore the previous slot configuration when searching for a slot. This is vital to device drivers during the `Drvr_Startup` call, and failure to do so at other times may break older, seven-slot applications.

The Slot Arbiter attempts to maintain a static seven-slot system for applications as reflected by the user's Control Panel settings. This system allows older applications to continue to work, as something they find in an older, seven-slot scan is still present. Newer applications may wish to consider implementing a 14-slot scan, but any slot not present in the static seven-slot environment requires a call to the Slot Arbiter before and after **every** access to that device. The overhead in such instances may be intolerable. Apple recommends that if an application requires hardware that cannot be found in a seven-slot scan, it request the user to set the Control Panel to make the hardware available and restart the system.

Using Slot-Dependent Hardware

Applications which have slot dependencies must call the Slot Arbiter before each use of the slot in question. Since Slot Arbitration changes the environment to which Apple IIGS programs have become accustomed, everyone has a better chance of working by sticking to the general Apple IIGS rule of "put back what you use when you're done with it." Ask for the slot, use it, then restore the previous Bit Encoded Slot Configuration. (If you use multiple slots, you might wish to get the Bit Encoded Slot Configuration, save a copy, modify it to reflect the slots you want, and restore from the modified version.)

Note: Peripherals accessed through GS/OS do not have to call the Slot Arbiter; GS/OS handles this task automatically.

There are certain applications with more specialized needs, such as high-speed, single character input or output. In such cases, the Slot Arbiter may be a bottleneck. When a slot is not switched, the Slot Arbiter returns quickly, but when a slot must be switched, it takes a significant amount of time. Doubling that significant time for switching in and restoring gives a substantial overhead for each hardware access, which may be too much for some applications.

Note: It is **far** better to write a GS/OS driver to deal with hardware than to write a slot-dependent application to control it. A slot-dependent application must deal with the Slot Arbiter, and the user must quit the current application to run your application just to change some aspect of the hardware. Writing a GS/OS driver lets any application, desk accessory, or CDev control your hardware with regular GS/OS calls.

Problems with Slot-Dependent Tools

Code designed before the Slot Arbiter may have slot-dependencies that cause unexpected problems when dynamic slot arbitration is fully implemented. This list includes some of the Apple IIGS System Software. Specifically, the Text Tools and the `FWEntry` call in the Miscellaneous Tools present problems with dynamic slot arbitration.

Text Tools

When using the Text Tools to specify a device for input, output, or error, the value specified (a four-byte parameter) is assumed to be a slot number if it is in the range 0-7. The Text Tools were not designed to use Slot Arbiter-style slot numbers, and this causes a compatibility problem.

The Text Tools were modified in System Software 5.0 to recognize Slot Arbiter-style slot numbers where possible. The trick is that it's not possible as often as we'd like. External slots are specified by using slot numbers 9 through 15; if such a slot number is used as input to a Text Tools call, the appropriate Slot Arbiter call is made and that external slot is used if it can be made available. However, internal port numbers are in the range 1-7—the same range used by the old Text Tools to indicate which of two peripherals was switched in for a particular slot. The Text Tools cannot assume that you are requesting an internal slot when using a slot number between one and seven.

For example, your old application might do a seven-slot search and find a parallel printer card in slot 1 (where the Control Panel setting for that slot is "Your Card"). If the Text Tools assumed all slot numbers in the range one through seven meant internal ports, your application would actually access the internal port 1 firmware every time it tried to access the parallel card it found in slot 1; this problem occurs since old applications don't know and don't care about internal or external slots.

The Text Tools may be used to access any external slot (if available), but they may only be used to access internal ports that are set to internal in the Control Panel. The Text Tools slot numbers zero through seven always match the Control Panel settings.

Apple strongly recommends that the Text Tools not be used. GS/OS character-based drivers are preferable for standard character input and output. The Text Tools may be used for specialized

purposes; however, you cannot access some internal ports and other components of the system that are not well-behaved. Doing so could cause your application to trash memory or media. You must assume these risks when using the Text Tools.

FWEntry

The Miscellaneous Tools call `FWEntry` should not be used to access entry points on a peripheral card (entry points in the $Cxxx range). As discussed, a poorly-behaved routine could switch the slot from one you've identified to something else between the time you identify the slot and issue the `FWEntry` call. Furthermore, the space between $C800 through $CFFF cannot be identified as belonging to any given slot, and the Slot Arbiter more or less guarantees that it won't be what you expect. Accesses to peripheral card ROM space ($Cxxx) should **only** be made by GS/OS drivers. `FWEntry` must **not** be used to access $Cxxx addresses.

`FWEntry` is still safe to use for addresses in the $D000-$FFFF range.

Further Reference

- *Apple IIGS Toolbox Reference*, Volume 2
- *Apple IIGS Firmware Reference*
- *Apple IIGS Hardware Reference*
- *GS/OS Reference*