ECR #: 36
Title: Multifunction A.G.P. Compliant Master
Release Date: 6/4/97
Impact: Clarification
Spec Version: A.G.P. 1.0

Summary: This ECR specifies how multiple functions (as defined by PCI) share an A.G.P. Compliant Master interface.

Background: Since A.G.P. is a point to point connection, the use of multifunction devices are needed whenever multiple functions are built on a single piece of silicon. PCI defines how each of these functions interact with system software. From software's point of view, each function is unique regardless of whether it resides with other functions or by itself on a piece of silicon. The current version of the A.G.P. interface specification is void in this area and needs to be augmented to explain how an A.G.P. compliant master should support multiple independent functions in a single silicon device.

Change Current Specification by adding the following:

6.3 Multifunction A.G.P. Compliant Master

A multifunction device consists of at least two independent functions integrated on a single piece of silicon. Any single function device can be composed of multiple sub-functions. The left block diagram in Figure 0-1 is a single function PCI agent that has two subfunctions x and y. The right hand block is a multifunction device that has functions A and B. Note that A or B could have multiple subfunctions as illustrated in the left block diagram. From a software point of view, the main difference between a single function device (that has multiple subfunctions) and a true multifunction device, is that a unique device driver is required for each function, but is not required for each subfunction. For a multifunction PCI device, the only logic shared between functions is the actual I/O buffers and an internal arbiter that determines when each function gains access to the bus. Beyond this difference, each function is completely independent of all other functions. A PCI example might be a Graphics and a Video Capture controllers were each was developed independently but are now integrated on the same silicon. Function A in the figure could be the Graphics controller and Function B the Video Capture controller.

Since only a single **REQ#**/**GNT#** pair is available at the device interface, the arbiter determines which interface logic may become a master on the bus. Each must do its own decode and assert **DEVSEL#** when selected. The existing device driver for each function can be used unmodified[1] with the new integrated device.

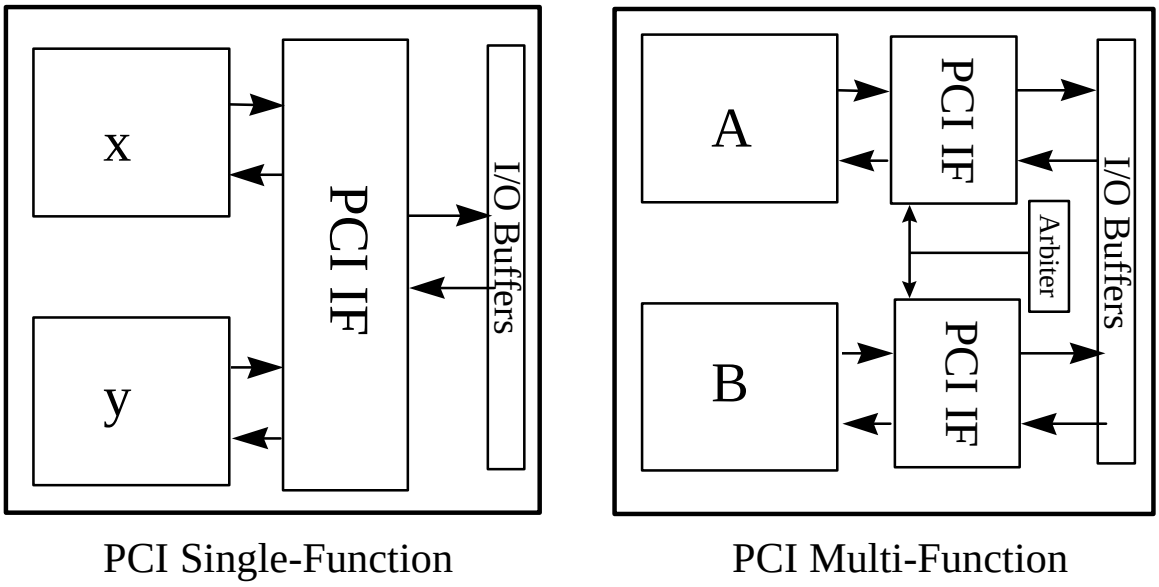

Figure 0-1 PCI Device Block

When implementing an A.G.P. compliant multifunction device, it is similar to a PCI compliant multifunction device, but with significant differences. Since the A.G.P. master sequencer can only request a single data transfer rate, all functions on the device must use the same rate. All devices must use either 1x or 2x and are not allowed to switch between transactions. The master must also indicate whether it will enqueue requests via the **AD** bus or the **SBA** Port. Simply integrating independent A.G.P. implementations does not result in a viable implementation since the master must present one choice per option. This requires all functions of the device to use the same modes.

[1] This means that the same driver is used independent of whether a single function card is inserted in the system, or multifunction card containing the same function is used.

On the other hand, the corelogic must support all basic modes of operation defined by this specification. However, the corelogic is not required to support all modes at the same time. For example, the corelogic can be programmed to accept requests on the **AD** bus or the **SBA** Port but is not required to support both at the same time.

No device driver changes are required for a PCI multifunction implementation. The same should be true for a multifunction A.G.P. implementation. However, care needs to be taken to ensure that this is true. If the driver used information on how the interface was implemented, the integration of multiple functions could require driver changes. For example, if the driver used the RQ_DEPTH value to optimize its operation, the device driver would require changes when the function is integrated with other A.G.P. functions. In the multifunction case, this value is shared between multiple functions and can not be used directly by the driver. On the other hand if this type of information is not used directly by the driver, then no driver changes are required.

Figure 0-2 shows how an A.G.P. compliant master would support multiple functions. Note: that unlike the PCI block diagram (Figure 0-1) where there is a difference between a multifunction device and a single function device with multiple subfunctions, A.G.P. has no differences. Because of this, function 0 represents the A.G.P. capabilities of the device regardless of how many functions use the A.G.P. interface.

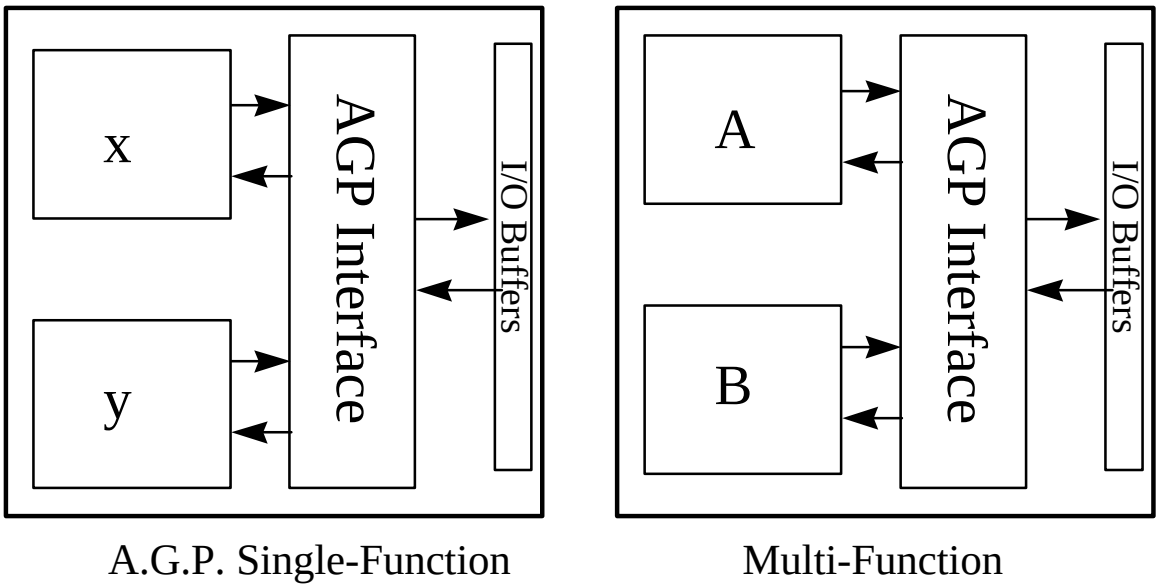

Figure 0-2 A.G.P. Block

A.G.P. Configuration registers

Each A.G.P. function is required to implement its own PCI target interface. This ensures that system resources (memory space, I/O space and an Interrupt line) that the device requires are assigned during configuration time.

Since A.G.P. capabilities are queried after the OS is running, Function 0 (single or multifunction device) of the A.G.P. compliant master presents its capabilities to the system in the A.G.P. status register. System software then programs the A.G.P. Command register as to how the A.G.P. interface will operate. A summary of operating options are found in Table 0-1 and the designer must ensure that all functions that use the interface make the same choices.

Table 0-1 A.G.P. Options

A.G.P. Attribute	Options

Rate	1x or 2x
4G Addressing	Support or Not
Requests	**SBA** Port or **PIPE#**
Outstanding Requests	1 to 256

Close attention needs to be paid to the RQ field of the status register and the RQ_DEPTH field of the command register. The RQ field must represent the sum of the maximum number of memory reference transactions that all functions on the device can have outstanding at any given time. For example, if function 0 can have 24 A.G.P. requests outstanding and function N[2] can have an additional 12 A.G.P. requests, then the RQ field of the master needs to indicate that up to 36 requests can be outstanding at any given time.

Careful attention needs to be paid when the RQ_DEPTH field is programmed to a value less than the RQ field. When this condition occurs the designer must choose how the available request slots are allocated. A simple implementation would be to do a first come first served approach. A more complicated, but fairer option would be to allocate a fixed ratio to each function. Any choice is acceptable as long as the A.G.P. master never allows more requests to be enqueued then the RQ_DEPTH field allows.

The AGP_ENABLE bit (9) in the command register controls all master functions that use the A.G.P. interface. When this bit is cleared (0), the interface is not allowed to make any new requests to the corelogic. This bit has the same meaning for both single or a multifunction implementations. For example, if function 0 is a 3D engine and function 4 is a Video Capture, both functions are disabled when the AGP_ENABLE bit (which resides in function 0) is cleared. If individual enables bits are desired, these are provided for in function specific space.

Internal Arbiter

In a PCI multifunction device the arbiter determines which PCI master interface logic gains access to the devices pins to initiate a request on the bus. For a multifunction A.G.P. implementation this same function does not exist. The arbiter in the A.G.P. compliant master sequencer determines which request is enqueued next, regardless of which function generated the request, but common logic presents the request to the system.

Deadlock Avoidance

When multiple A.G.P. functions are supported, the designer must ensure that no deadlocks or livelocks are possible. The designer must ensure that the master device appears to the corelogic as a single function agent. This requires that the A.G.P. compliant master make no assumptions as to the order in which requests are completed. Note that in a multifunction implementation, Function A and Function B may interfere with each others operation. For example, Function A has LP read requests 4, 5 and 9 outstanding while Function B has LP read requests 6, 7 and 8. If Function B asserts **RBF#** when data for request 7 is returned, it will cause the delay of Functions A data until request 8 completes. Function A may have room to accept the data but the corelogic is not allowed to return it until transaction 8 completes. Therefore care needs to be taken to balance how requests are enqueued and provide sufficient buffer to ensure that one function does not affect the other.

[2] Note that the device is not required to implement the second function as number 1, but can assign it any value between 1 and 7.