The io_mapping functions in linux/io-mapping.h provide an abstraction for efficiently mapping small regions of an I/O device to the CPU. The initial usage is to support the large graphics aperture on 32-bit processors where ioremap_wc cannot be used to statically map the entire aperture to the CPU as it would consume too much of the kernel address space.

A mapping object is created during driver initialization using

```
struct io_mapping *io_mapping_create_wc(unsigned long base,
                                        unsigned long size)
```

'base' is the bus address of the region to be made mappable, while 'size' indicates how large a mapping region to enable. Both are in bytes.

This _wc variant provides a mapping which may only be used with the io_mapping_map_atomic_wc or io_mapping_map_wc.

With this mapping object, individual pages can be mapped either atomically or not, depending on the necessary scheduling environment. Of course, atomic maps are more efficient:

```
void *io_mapping_map_atomic_wc(struct io_mapping *mapping,
                               unsigned long offset)
```

'offset' is the offset within the defined mapping region. Accessing addresses beyond the region specified in the creation function yields undefined results. Using an offset which is not page aligned yields an undefined result. The return value points to a single page in CPU address space.

This _wc variant returns a write-combining map to the page and may only be used with mappings created by io_mapping_create_wc

Note that the task may not sleep while holding this page mapped.

```
void io_mapping_unmap_atomic(void *vaddr)
```

'vaddr' must be the the value returned by the last io_mapping_map_atomic_wc call. This unmaps the specified page and allows the task to sleep once again.

If you need to sleep while holding the lock, you can use the non-atomic variant, although they may be significantly slower.

```
void *io_mapping_map_wc(struct io_mapping *mapping,
                        unsigned long offset)
```

This works like io_mapping_map_atomic_wc except it allows the task to sleep while holding the page mapped.

```
void io_mapping_unmap(void *vaddr)
```

This works like io_mapping_unmap_atomic, except it is used

for pages mapped with io_mapping_map_wc.

At driver close time, the io_mapping object must be freed:

```
void io_mapping_free(struct io_mapping *mapping)
```

Current Implementation:

The initial implementation of these functions uses existing mapping mechanisms and so provides only an abstraction layer and no new functionality.

On 64-bit processors, io_mapping_create_wc calls ioremap_wc for the whole range, creating a permanent kernel-visible mapping to the resource. The map_atomic and map functions add the requested offset to the base of the virtual address returned by ioremap_wc.

On 32-bit processors with HIGHMEM defined, io_mapping_map_atomic_wc uses kmap_atomic_pfn to map the specified page in an atomic fashion; kmap_atomic_pfn isn't really supposed to be used with device pages, but it provides an efficient mapping for this usage.

On 32-bit processors without HIGHMEM defined, io_mapping_map_atomic_wc and io_mapping_map_wc both use ioremap_wc, a terribly inefficient function which performs an IPI to inform all processors about the new mapping. This results in a significant performance penalty.