iogen

What it is:

The iogen kernel module is a driver that opens up a range of I/O ports in order to make it possible to write software that uses ACCES ISA and PC/104 products in computers running Linux distributions with a 2.6 kernel.

Important files:

acceslib.h: Header file that provides functions that closely mimic DOS style inport/outport functions.

acceslib.c: Implementation of the functions declared in acceslib.h. All provided samples make use of these functions.

iogen.h: Header file for the kernel module. This must be included in any program that makes use of the iogen driver.

iogen.c: Implementation of the kernel driver. Do not attempt to compile this file into a program that makes use of the iogen driver.

Makefile: Makefile for kernel module.
http://www.freesoftwaremagazine.com/articles/drivers_linux

build: a script that runs the commands required to compile the kernel module. This file must be edited to replace "/usr/src/linux-source-2.6.15" with the location of the source tree on the machine the module is being built for.

iogen_load: A script that loads the module into the system.

iogen_unload: A script that removes the module from the system.

routines.c: A few functions that are used in almost all the samples provided with the iogen driver. None of these functions are required, but they can make life easier.

How it is used:

Sample programs are provided for most ACCES products that demonstrate how to use the iogen driver. The samples generally make use of the acceslib.c and acceslib.h files, but these are not required to use the driver.

Using Acceslib:

Making use of iogen via acceslib requires you to #include acceslib.h in your program. Even though you don't directly include iogen.h in your program it will have to be where your compiler can find it.

Acceslib provides the following six functions in order to communicate with I/O ports:

```
int outportb (int fd, unsigned int offset, __u8 data);
int outport (int fd, unsigned int offset, __u16 data);
int outportl (int fd, unsigned int offset, __u32 data);

int inportb(int fd, unsigned int offset, __u8 *data);
int inport(int fd, unsigned int offset, __u16 *data);
int inportl (int fd, unsigned int offset, __u32 *data);
```

All these functions read from or write to I/O ports. The return value is 0 on success and an error code on failure. The fd parameter is an open file descriptor that is associated with the iogen device file /dev/iogen. The offset parameter is the I/O address that is being read from or written to. The data parameter is a pointer to unsigned 8/16/32 bit value. The inport functions will place the data read into the parameter and the outport function will not alter the parameter.

In order to make use of these functions it is necessary to open the iogen device file. The routines.c file provides a function for just this purpose.

```
int open_dev_file();
```

The return value from this function will be the file descriptor associated with the device file. If the function fails to open the device file it will print an error message to stdout and terminate the program. It is strongly advised that this be one of the first things your program attempts to do as being unable to open the device file makes the iogen driver useless.

Using iogen.h:

The iogen.h file provides an alternate interface to the I/O ports, along with the ability to use IRQ driven I/O. Using the ioctls associated with reading from and writing to ports is more work for an application programmer, and the acceslib method can still be used while using iogen.h for IRQ driven I/O, but ioctls are the only method available through the iogen driver package for IRQ driven I/O.

An in-depth discussion of using ioctls for reading from and writing to I/O ports is not provided here as it is much more convenient to simply use the acceslib method. The acceslib.c file demonstrates how to use the ioctls, and the comments in iogen.h should answer any additional questions.

There are three ioctls provided with the iogen driver for using IRQs. Each one is

explained in depth here.

io_gen_irq_setup:

This ioctl must be the first of the IRQ associated ioctls to be called. First fill a variable of type io_gen_irq_struct with the required values. (See Figure 1.)

io_gen_irq_struct	
offset_to_clear	The address that must be written to or read from in order to clear a pending interrupt from a card/board. Information for this, and all other values to put in the structure can be found in the Product's manual.
irq_num	The IRQ that the card/board is set for.
write_to_clear	A boolean value for if the card/board is write-to-clear or read-to-clear for pending interrupts.
val_to_clear	If the card/board is write-to-clear this is the value that will be written to clear a pending interrupt.

Figure 1

io_gen_wait_for_irq_ioctl:

This ioctl is a blocking call that waits for an IRQ to be called. This means that the thread the ioctl is called from will wait for the ioctl to return before it can do anything else. The ioctl will return 0 if an IRQ occurred and nonzero for an error.

io_gen_cancel_wait_for_irq_ioctl:

This will cancel a wait for IRQ and cause the thread waiting on an IRQ to receive -ECANCELED for a return value.

ADDITIONAL INFORMATION:

For more information on writing device drivers and how a kernel module Makefile works try http://www.freesoftwaremagazine.com/articles/drivers_linux

The iogen driver simply opens up a wide range of I/O ports. Unfortunately, Linux does not appear to enforce the region requesting mechanism present in the kernel. It may do so for other architectures or I/O ranges. If you are working in an environment where this type of security is a priority then then irqgen driver may be more appropriate. The irqgen driver is similar to the iogen driver but only a specific range of I/O ports is opened so only a single product or products using contiguous address space can be accommodated.