ADAS ICV196 VME I/O Board's device driver for VME-Linux/m68k

J. Gaulmin - IRTS 10 August 1999

ADAS's general purpose VME I/O board **ICV196** has 96 input/output channels which can be programmed either as input lines or output lines by group of 8. The 16 first input lines can be used as edge or level sensitive interrupt sources. ICV196 is the VME-Linux/m68k device driver written to support up to 4 ICV196 boards. The driver has been implemented as a Linux loadable module for kernels 2.0.x and 2.2.x. This document explains the functionalities of the ICV196 device driver and the programmer's C-interface library for it (note that ioctl() calls are the only way to access to ICV196 devices with this driver).

Contents

1 Introduction

ADAS's general purpose VME I/O board ICV196 has 96 input/output channels which can be programmed either as input lines or output lines by group of 8. The 16 first input lines can be used as edge or level sensitive interrupt sources. The card has also three 16 bits timers available to generate interrupt (these timers are not used on the ICV196 device driver).

ICV196 is the VME-Linux/m68k device driver written by Integrated Real Time Systems (IRTS) for the European Synchrotron Radiation Facility (ESRF) to support up to 4 ICV196 boards. The driver has been implemented as a Linux loadable module for kernels 2.0.x and 2.2.x. This document explains the functionalities of the ICV196 device driver and the programmer's C-interface library for it.

Note that ioctl() calls are the only way to access to ICV196 devices with this driver.

The ICV196 device driver main features are :

- Configuration of the ICV196 I/O board.

User can configure the 96 input/output channels either as input lines or output lines by group of 8. These channels are divided in 12 groups of 8.

group 0 : channels 0 .. 7

group 1 : channels 8 .. 15

group 2 : channels 16 .. 23

group 3 : channels 24 .. 31

group 4 : channels 32 .. 39

group 5 : channels 40 .. 47

group 6 : channels 48 .. 55

group 7 : channels 56 .. 63

group 8 : channels 64 .. 71

group 9 : channels 72 .. 79

group 10 : channels 80 .. 87

group 11 : channels 88 .. 95

All the groups directions are configured/read to INPUT (0) or OUTPUT (1) at the same time. The 12 groups directions are specified/read in the bytes (unsigned chars) arg.val[0] for groups 0 to 7 and arg.val[1] for groups 8 to 11.

Note that : the LSB of the byte is always associated with the lowest group number; all the channels in a group are configured in the same way; all the groups are positioned on input at driver's initialization.

- Bit-wise (1 bit) input and output on any channel.

User can set/read any output/input to LOW (0) or HIGH (1) state without disturbing the state of the other channels. The bit's number is specified in arg.num and the bit's value is specified/read in arg.val[0].

Note that the value must be 0 or 1.

- Byte-wise (8 bits) input and output on any group.

User can set/read any output/input of the group to LOW (0) or HIGH (1) state without disturbing the state of the other channels and groups. The byte number (group number) is specified in arg.num and the byte's value is specified/read in the byte (unsigned char) arg.val[0].

Note that the LSB of the byte is always associated with the lowest channel of the group.

- Word-wise (16 bits) input and output on any pair of following groups.

User can set/read any output/input of the 2 consecutive groups to LOW (0) or HIGH (1) state without disturbing the state of the other channels and groups. The low byte's number (group number) is specified in arg.num and the high byte's number is taken as arg.num+1 (next group). The 2 bytes' values are specified/read in the bytes (unsigned chars) arg.val[0] for the low group and arg.val[1] for the high group (in fact the following one).

Note that the LSB of the byte is always associated with the lowest channel of the group.

- All-wise (96 bits) input and output on all the channels.

User can set/read all the outputs/inputs (all the 12 groups) to LOW (0) or HIGH (1) state. The 12 bytes' values are specified/read in the bytes (unsigned chars) arg.val[0] for group 0 .. arg.val[11] for group 11.

Note that the LSB of the byte is always associated with the lowest channel of the group.

- External event occurrence wait.

User can make processes/threads wait, in a sleeping state with or without timeout, for a specific edge or level on one of the 16 interrupt input channels. The interrupt channels are the first 16 channels (groups 0 and 1) of the board and the corresponding groups must be set as input to generate interrupts. The waiting channel's number is specified in arg.num. The waiting event and timeout are specified in arg.op and arg.it_timeout (0 for none).

Note that severals processes/threads can wait for the same event on the same interrupt channel.

2 ICV196 kernel module

2.1 Loading (ICV196_load)

2.1.1 Mechanism

Linux kernel modules are specially made to be pieces of kernel that can be loaded and unloaded dynamically, while the kernel is running. These appear as object files (modname.o) and are loaded with the command insmod modname.o [arguments]. This operation runs the initialization of the device(s) and gives a major number to the device driver. This one can be found in the /proc/devices file. The command can receive many arguments specific to the module.

After being loaded, the device driver module must be associated with devices files which will be used by user programs. This is made with the command : mknod devname c major minor.

For the ICV196 device driver, all the work is done in one time by the ICV196_load shell script.

As the ICV196 device driver use a probe call to insure that a board is really responding at the specified address(es), the probe module which contains the probe function must be installed first on system. This one appear as a probe.o object file and can be installed with the insmod -f probe.o command.

2.1.2 Parameters

Three optional parameters can be specified when you load the ICV196 device driver with the ICV196_load script or with the insmod command. The loading command will looks like :

./ICV196_load io_adr=adr0,adr1,adr2,adr3 base_vector=bv0,bv1,bv2,bv3 vme_irq=x

io_adr:

As the ICV196 device driver can handle up to 4 ICV196 boards at the same time and the ICV196 board(s) do(es) not always have the same address, user can specify board(s) address by adding io_adr=adr0,adr1,... at the end of the loading command. Only the 24 most significant bits of the board's address must be specified in io_adr and these must be presented in a hex format (e.g. : io_adr=0xFF7C04,0xFF7C05 for two ICV196 boards with base addresses 0xFF7C0400 and 0xFF7C0500). In case of no io_adr parameter, only one board with the default address IO_BOARD_ADR (specified in the ICV196.h header file) will try to be loaded. To see which I/O space is already use by devices you can look at the /proc/ioports file.

base_adr:

As each board requires 16 IRQ vectors (one for each of its interrupt channels), user can specify board(s) IRQ base vector by adding base_vector=bv0,bv1,... at the end of the loading command. If user tries to load more boards than IRQ base vectors, the missing base_vector(s) will be the base_vector of the previous board + 0x10. Thus, if you try to load 4 boards with only one IRQ base vector specified, the driver will require IRQs from kernel from base_vector to base_vector + 0x3F. In case of no base_vector parameter, bv0 will take the default value BASE_VECTOR (specified in the ICV196.h header file). On Linux/m68k for VME, user can require IRQ vectors from 64 to 255. To see the IRQ vectors that are already required you can look at the /proc/interrupts file (only used one appear in this file). The 16 vectors of each board will be required from kernel like this :

- channel 0 : base_vector + 0
- channel 1 : base_vector + 2
- .....
- channel 7 : base_vector + 0xE
- channel 8 : base_vector + 1
- .....
- channel 15 : base_vector + 0xF

vme_irq:

As the interrupts are classified in 7 IRQ priority levels on the VME bus, user can specify ICV196 board(s) VME IRQ level by adding vme_irq=x at the end of the loading command where 'x' is an integer from 1 to 7. Of course this only specifies the priority level of the interrupt request that will be generated by the ICV196 board(s) and it supposes that the VME controller is well configured to handle this level and to transmit the interrupt to the processor. In case of no vme_irq parameter, the device driver will use the default value VME_IRQ (specified in the ICV196.h header file) to configure the board(s).

In addition of these parameters you can have 2 bonus ones if you compile the device driver with the VMEchip2 features management. These are:

chip2_adr:

This parameter enables user to specify a different address for the VMEchip2 controller. In case of no chip2_adr, the device driver will use the default value VMECHIP2_ADR (specified in the VMEchip2lib.h header file) to map the chip.

adr_mod:

This parameter enables user to specify which address modifier the device driver will look for in the VMEchip2's master windows. In case of no adr_mod, the device driver will use the default value ADR_MODIFIER (specified in the VMEchip2lib.h header file).

2.1.3 Errors

Error can appear during the module loading. This error may be caused by invalid loading parameters or by the fact that module is already running on system.

A probe function call insures that a board is really responding at the specified address(es) and avoid system crash if not. An error message will appear on the kernel log file if a bus error occurs.

In case of error, you should check if the module is not already running and if all the required resources are free (see the Special files section). You can also use the dmesg command to see debug or error messages (see the Debug options section).

2.2 Unloading (ICV196_unload)

2.2.1 Mechanism

As you can dynamically load your kernel module, you can also unload it when you want using the command rmmod modname. You also have to remove the devices files that you made with ICV196_load.

For the ICV196 device driver, all the work is done by the ICV196_unload shell script.

2.2.2 Errors

Nevertheless, the module will be unloaded only if all the processes/threads have been closed before. This is done with the driver's release() function call which is called by the generic close() function. Finally, the call may look like close(FD). If some processes/threads are still using the device driver when you try to unload it, the kernel will display a 'busy device or resource' message on console.

3 Ioctl() calls

3.1 Description

Before making an ioctl() call to a special file (device driver description file in our case), the device must have been opened by the user, using the driver's open() function call which may look like FD=open("/dev/ICV196_0", O_RDWR, 0x666).

Then to make any ioctl() call user has to indicate the file descriptor (int FD) that has been returned by the open() function, a command parameter (unsigned char cmd) and an argument parameter (ICV196_arg arg). The call then may look like err=ioctl(FD, cmd, &arg) where err is an integer returned by the function. In the ICV196 device driver, arg is a complex structure thus, the ioctl() function waits for a ICV196_arg* argument. That's why we provide arg's address to the function.

This section explains the specifications of cmd and arg parameters and the returned values of the ioctl() function.

3.2 Command parameter (cmd)

This unsigned char parameter is used to indicate to the driver that you want to read (RD) or to write (WR) on the ICV196 board. You can also indicate that you want your process to go to sleep (SLEEP) waiting for an interrupt from one of the 16 interrupt channels of the board.

cmd=RD:

read on channel(s) of the board

cmd=WR:

write on channel(s) of the board

cmd=SLEEP:

go to sleep waiting for an interrupt from one of the 16 interrupt channels of the board

RD, WR and SLEEP are unsigned char (u8) values declared on ICV196.h:

```
/*ICV196.h*/
/*ioctl() cmd constants*/
#define RD 0x11
#define WR 0x12
#define SLEEP 0x13
```

3.3 Argument parameters (arg)

The arg parameter is a ICV196_arg structure defined in the ICV196.h header file and used in complement of the cmd parameter.

```
/*ICV196.h*/
#define MAX_PORTS = 12 /*number of ports on ICV196 board*/
struct icv196arg
{
u8 op;
u8 num;
u8 val[MAX_PORTS];
u32 it_timeout;
};
typedef struct icv196arg ICV196_arg;
```

3.3.1 Operation argument (arg.op)

In case of a RD or WR command, this unsigned char takes value DIO_BIT when you want to RD or WR a single channel on the board, DIO_BYTE for a byte (8 channels, 1 group), DIO_WORD for two consecutive

bytes (16 channels, 2 groups) or DIO_ALL for all the bits at the same time (96 channels, 12 groups). You can also RD or WR groups directions with IO_DIR value.

In case of a SLEEP command, it is used to determine which event the process/thread has to wait for. It can wait for a rising edge (low to high transition - RISE), a falling edge (high to low transition - FALL), any edge (any transition - ANY), a high level (HIGH) or a low level (LOW).

RD/WR:

- **arg.op = DIO_BIT. Single channel read or write**
- **arg.op = DIO_BYTE. Single group (8 channels) read or write**
- **arg.op = DIO_WORD. Two consecutive groups (16 channels) read or write**
- **arg.op = DIO_ALL. All the channels (12 groups) at the same time read or write**
- **arg.op = IO_DIR. Groups direction read or write**

SLEEP:

- **arg.op = RISE. Rising edge wait**
- **arg.op = FALL. Falling edge wait**
- **arg.op = ANY. Both rising and falling edge wait**
- **arg.op = HIGH. High level wait**
- **arg.op = LOW. Low level wait**

DIO_BIT, DIO_BYTE, DIO_WORD, DIO_ALL, DIO_DIR, RISE, FALL, ANY, HIGH and LOW are unsigned char (u8) values declared in ICV196.h:

/*ICV196.h*/

/*ioctl() arg.op constants*/

#define DIO_BIT 0x21

#define DIO_BYTE 0x22

#define DIO_WORD 0x23

#define DIO_ALL 0x24

#define IO_DIR 0x25

#define RISE 0x26

#define FALL 0x27

#define ANY 0x28

#define HIGH 0x29

#define LOW 0x2A

3.3.2 Number argument (arg.num)

This unsigned char is used to indicate the channel number (RD/WR | DIO_BIT or SLEEP), the group number (RD/WR | DIO_BYTE) or the word's low group number (RD/WR | DIO_WORD). It is not used with RD/WR | DIO_ALL and RD/WR | IO_DIR, so the value will be ignored in that cases.

RD/WR:

- **DIO_BIT : arg.num = [0-95]. This is the channel number**
- **DIO_BYTE : arg.num = [0-11]. This is the group number**
- **DIO_WORD : arg.num = [0-10]. This is the word's low group number. The word's high group number will be arg.num+1**
- **DIO_ALL : arg.num = don't care**
- **IO_DIR : arg.num = don't care**

SLEEP:

- **arg.num = [0-15]. This is the interruptible channel number. Only the 16 first channels can generate interrupts**

3.3.3 Value argument (arg.val[]) - only with RD or WR commands

This array of 12 unsigned char contains the bit/byte/word/all value in case of a RD or a WR command. This value may be written with WR command and read with RD command. It is not used with SLEEP command and will be ignored in that case.

RD/WR:

- **DIO_BIT : arg.val[0] = [0 or 1]. This is the bit value**
- **DIO_BYTE : arg.val[0] = [0x00-0xFF]. This is the byte value**
- **DIO_WORD : arg.val[0] = [0x00-0xFF]. This is the low group value. arg.val[1] = [0x00-0xFF]. This is the high group value**
- **DIO_ALL : arg.val[0-11] = [0x00-0xFF]. These are the groups values (arg.val[0] -> group 0 .. arg.val[11] -> group 11)**
- **IO_DIR : arg.val[0] = [0x00-0xFF] and arg.val[1] = [0x00-0x0F]. This is the direction mask value. Use 1 for outputs and 0 for inputs (at initialization of the device driver, all the groups are inputs)**

SLEEP:

- **arg.val[0-11] = don't care**

e.g. : cmd = WR | arg.op = DIO_WORD | arg.num = 4 | arg.val[0] = 0x81 & arg.val[1] = 0x18 makes a write command on group 4 (channels 32 to 39) with arg.val[0] and group 5 (channels 40 to 47) with arg.val[1]. Thus channels 32, 39, 43 and 44 will be high and others channels of groups 4 and 5 will be low.

e.g. : cmd = RD | arg.op = IO_DIR makes a read command of the direction word. If after the ioctl() call arg.val[0] = 0x81 and arg.val[1] = 0x02, it indicates that groups 0, 7 and 9 are outputs and groups 1, 2, 3, 4, 5, 6, 8, 10 and 11 are inputs.

3.3.4 Waiting timeout argument (arg.it_timeout) - only with SLEEP command

This unsigned long (u32) is only used with SLEEP command and will be ignored with others. It is used to indicate the maximum amount of time that the process/thread should wait before being woken up by the kernel. The timeout value is indicated in jiffies (usually 1/100 s on Linux systems) and a zero value indicates that you don't want any timeout.

RD/WR:

- **arg.it_timeout = don't care**

SLEEP:

- **arg.it_timeout = 0. The process will only be woken up by the interrupt (or kill signal)**
- **arg.it_timeout = [0x00000001-0xFFFFFFFF]. Timeout value in jiffies (usually 1/100 s on Linux systems)**

3.4 Returned value

The ioctl() call returns 0 on success or -1 on fail. In case of fail, errno values are standardized by the include file <asm/errno.h> so that you can know what kind of problem has occurred. The following errno constants are used in the ICV150 device driver :

ENOMEM:

out of memory (12)

EFAULT:

bad address (14)

EBUSY:

device or resource busy (16)

ENODEV:

no such device (19)

EINVAL:

invalid argument (22)

ERESTARTSYS:

interrupted system call, should be restarted (85)

If the driver has the required debug level, you can also use the command dmesg to see in details where and why the ioctl() call has failed.

4 Miscellaneous

4.1 Interrupts specifications

On ICV196 board, the interrupts are serviced by the Z8536 chip. This one can deal with 16 interrupt channels divided in 2 groups called Port A (channels 0 to 7 of the board) and Port B (channels 8 to 15 of the

board). Unfortunately, this chip is unable to handle multiple interrupts on same port at the same time when the waited event is an edge. To avoid errors, user should only use one channel of each port when waiting for edges than can occur at the same time. If the user needs more simultaneous interrupt channels, he should use level events for all channels. Even if these ones can occur all at the same time, they will be serviced one by one with priority (see 2.1.2 for vectors priority)

If the user asks for a waiting level that is already reached by the channel, the command will immediately returns from sleeping because the state has been reached.

4.2 Debug options

At the compilation of the driver, user can specify which level of debug he wants to be displayed on the kernel log. This is done by uncommenting #define DEBUG for debug level 1 or both #define DEBUG and #define DEBUG2 for debug level 2 in the ICV196.c file. In general, debug level 1 displays actions and probable causes of command faults and debug level 2 add the state of important global and local variables at that time so that you can determine what was wrong.

You can also display all the kernel messages by using dmesg command.

Note that debug options slow down the device driver.

4.3 Special files

Kernel uses special files to save all the systems parameters. Some of those can be very useful to get informations about the device driver :

/proc/devices:

this file indexes all the devices drivers installed on the system with their major number and their type (char or block).

/proc/interrupts:

this file indexes all the interrupts that have already appear on the system with their interrupt vector, their frequency and the name of the device driver which owns them.

/proc/ioports:

this file indexes all the I/O regions that have been taken by devices drivers. The name of the device driver that owns the region is also displayed.

/proc/ksyms:

this file indexes all the kernel entry points with their address and the name of the function. You can display these informations with the ksyms command.

/proc/modules:

this file registers all the loaded modules with their memory occupation and the number of processes/threads that have opened it. You can display these informations with the lsmod command.

/proc/version:

this file contains the current running kernel version. It is usefull to see if your module version is compatible with the current kernel but you can force the module even if the versions are incompatible with insmod -f.

/dev/ICV196_[0-3]:

these files are the devices files associated with each board using the ICV196 device driver. You can see major and minor number of each of these files with ls -l command.

/var/log/messages:

these file contains all the messages sent by kernel with printk() calls. You can display these messages with the dmesg command.

/etc/devinfo:

this file indexes all the different device drivers type that can exist with their major and minor number.