ADAS ICV150 VME Analog Acquisition Board's device driver for VME-Linux/m68k

J. Gaulmin - IRTS 06 September 1999

ADAS's general purpose VME Analog Acquisition board **ICV150** has 64 inputs channels (expandable to 256 by coupler cards) which can be used either has 32 differential or 64 single resident inputs. There is a choice of various ICV150 boards with 12, 14 or 16 bits converters, acquisition rate up to 250000 measurements/s, software programmable gain up to 128, galvanic isolation... ICV150 is the VME-Linux/m68k device driver written to support up to 4 ICV150 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 ICV150 device driver and the programmer's C-interface library for it (note that ioctl() calls are the only way to access to ICV150 devices with this driver).

Contents

1 Introduction **1**

2 ICV150 kernel module **3**

2.1 Loading (ICV150_load) . . . 3

2.1.1 Mechanism . . . 3

2.1.2 Parameters . . . 3

2.1.3 Errors . . . 4

2.2 Unloading (ICV150_unload) . . . 4

2.2.1 Mechanism . . . 4

2.2.2 Errors . . . 4

3 Ioctl() calls **4**

3.1 Description . . . 4

3.2 Command parameter (cmd) . . . 5

3.3 Argument parameters (arg[256]) . . . 6

3.4 Returned value . . . 7

4 Miscellaneous **7**

4.1 Interrupts specifications . . . 7

4.2 Debug options . . . 8

4.3 Special files . . . 8

1 Introduction

ADAS's general purpose VME Analog Acquisition board ICV150 has 64 input channels (expandable to 256 by coupler cards) which can be used either has 32 differential or 64 single resident inputs. There is a choice

of various ICV150 boards with 12, 14 or 16 bits converters, acquisition rate up to 250000 measurements/s, software programmable gain up to 128, galvanic isolation... The card has also external and software triggers possibilities with or without sample and hold stage. End of triggered sequences can be signaled by interrupts (software trigger and interrupts are not used on the ICV150 device driver).

ICV150 (icv150.o) 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 ICV150 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 ICV150 device driver and the programmer's C-interface library for it.

Note that ioctl() calls are the only way to access to ICV150 devices with this driver.

The ICV150 device driver main features are :

- Reading the conversion result of one channel.

User can read any conversion result of a scanning channel without disturbing acquisition. The channel's number is specified in arg[0] and the acquisition value is read in arg[0].

Note that the channel must be scanning when you read the value otherwise the ioctl() returned value will be EAGAIN.

- Reading the conversion results of all the channels.

User can read all the conversion results of the scanning channels without disturbing acquisition. Only scanning channels are read.

Note that the channels must be scanning when you read the values otherwise the ioctl() returned value will be EAGAIN.

- Gain configuration.

If the board has a software programmable gain amplifier, user can program each channel individually to a different gain (one by one or all at the same time). When programming one channel gain, the channel's number is specified in arg[0] and the gain value in arg[1]. When programming all the gains, the gain values are specified in arg[0] for channel 0, arg[1] for channel 1...

The gain value is programmable as a multiple of 2. The value parameter and the real gain have the following correspondence :

value -> gain

0 1

1 2

2 4

3 8

4 16

5 32

6 64

7 128

8 256

9 512

10 1024

The maximum gain that can be set depends on the input module and may be smaller than 1024. This must be checked from the modules documentation as there is no means that the driver can detect this. Additionally, trying to program a gain higher than the maximum gain of the module may lead to an actual gain that is undetermined.

The gain values may also be stored to an EEPROM from which they are fetched during power-up or by user command.

Note that the gain can be read, set, stored or fetched only when the board is stopped otherwise the ioctl() returned value will be EBUSY.

- Number of scanning channels configuration.

As the board can be extended up to 256 channels and user doesn't always need all of them, the number of scanning channels is programmable from 1 to 256. The new number is specified in arg[0] and channels from 0 to number-1 will then be scanned.

Note that the number of scanning channels can only be set when the board is stopped otherwise the ioctl() returned value will be EBUSY.

- Use of external triggered or continuous scanning mode.

In continuous mode, the board scans the specified inputs continuously. In external triggered mode, the scan is triggered by an external rising edge.

2 ICV150 kernel module

2.1 Loading (ICV150_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 ICV150 device driver, all the work is done in one time by the ICV150_load shell script.

As the ICV150 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

One optional parameter can be specified when you load the ICV150 device driver with the ICV150_load script or with the insmod command. The loading command will looks like :

./ICV150_load io_adr=adr0,adr1,adr2,adr3

io_adr:

As the ICV150 device driver can handle up to 4 ICV150 boards at the same time and the ICV150 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 ICV150 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 ICV150.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.

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 (ICV150_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 ICV150_load.

For the ICV150 device driver, all the work is done by the ICV150_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/ICV150_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 (unsigned short arg[256]). The call then may look like err=ioctl(FD, cmd, arg) where err is an integer returned by 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 which action you want to make.

cmd=RD_CHANNEL:

read the value of one of the channels that are being scanned

cmd=RD_ALL_CHANNELS:

read the values of all the channels that are being scanned

cmd=RD_GAIN:

read the gain of one of the scanned channels

cmd=RD_ALL_GAINS:

read the gains of all the scanned channels

cmd=SET_NUMBER:

set the number of channels that are going to be scanned

cmd=SET_GAIN:

set the gain of one of the channels that are going to be scanned

cmd=SET_ALL_GAINS:

set the gains of all of the channels that are going to be scanned

cmd=START:

start continuous scanning

cmd=EXT_TRIG:

start external triggered scanning

cmd=STOP:

stop current scanning

cmd=STORE:

store all the 256 gain values in EEPROM

cmd=RECALL:

retrieve all the 256 gain values from EEPROM and start continuous scanning

cmd=DEBUG:

set the debug level

RD_CHANNEL, RD_ALL_CHANNELS... are unsigned char (u8) values declared on ICV150.h:

```
/*ICV150.h*/
/*ioctl() cmd constants*/
#define RD_CHANNEL 0x10
```

#define RD_ALL_CHANNELS 0x11

#define RD_GAIN 0x12

#define RD_ALL_GAINS 0x13

#define SET_NUMBER 0x14

#define SET_GAIN 0x15

#define SET_ALL_GAINS 0x16

#define START 0x17

#define EXT_TRIG 0x18

#define STOP 0x19

#define STORE 0x1A

#define RECALL 0x1B

#define DEBUG 0x1C

3.3 Argument parameters (arg[256])

This array of 256 unsigned shorts is used to pass input parameters such as channel's number or debug level and to retrieve output values such as acquisition values and gain values.

RD_CHANNEL:

arg[0] is used to specify the channel's number and to return the acquisition value

RD_ALL_CHANNELS:

arg[0] is used to return the acquisition value of channel 0, arg[1] for channel 1...

RD_GAIN:

arg[0] is used to specify the channel's number and to return the gain value

RD_ALL_GAINS:

arg[0] is used to return the gain value of channel 0, arg[1] for channel 1...

SET_NUMBER:

arg[0] is used to specify the number of channels to scan

SET_GAIN:

arg[0] is used to specify the channel's number and arg[1] to specify the gain value

SET_ALL_GAINS:

arg[0] is used to specify the gain value of channel 0, arg[1] for channel 1...

START:

don't care

EXT_TRIG:

don't care

STOP:

don't care

STORE:

don't care

RECALL:

don't care

DEBUG:

arg[0] is used to specify the debug level

MAX_CHANNELS, DEF_CHANNELS and MAX_GAIN are constants values declared on ICV150.h:

/*ICV150.h*/

#define MAX_CHANNELS 256

#define DEF_CHANNELS 32

#define MAX_GAIN 10

3.4 Returned value

The ioctl() call returns 0 on success and -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 :

EAGAIN:

try again (11)

ENOMEM:

out of memory (12)

EFAULT:

bad address (14)

EBUSY:

device or resource busy (16)

ENODEV:

no such device (19)

EINVAL:

invalid argument (22)

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

Interrupts are not handled by this device driver.

4.2 Debug options

User can dynamically specify the level of debug he wants to be displayed on the kernel log by a simple ioctl() call. There are 4 different debug levels on the ICV150 device driver :

- level 0 (=0) : nothing
- level 1 (>0) : causes of error
- level 2 (>1) : causes of error + actions
- level 3 (>2) : causes of error + actions + parameters and status

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

Note that debug messages 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/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's 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/icv150_[1-4]:

these files are the devices files associated with each board using the ICV150 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 types that can exist with their major and minor number.