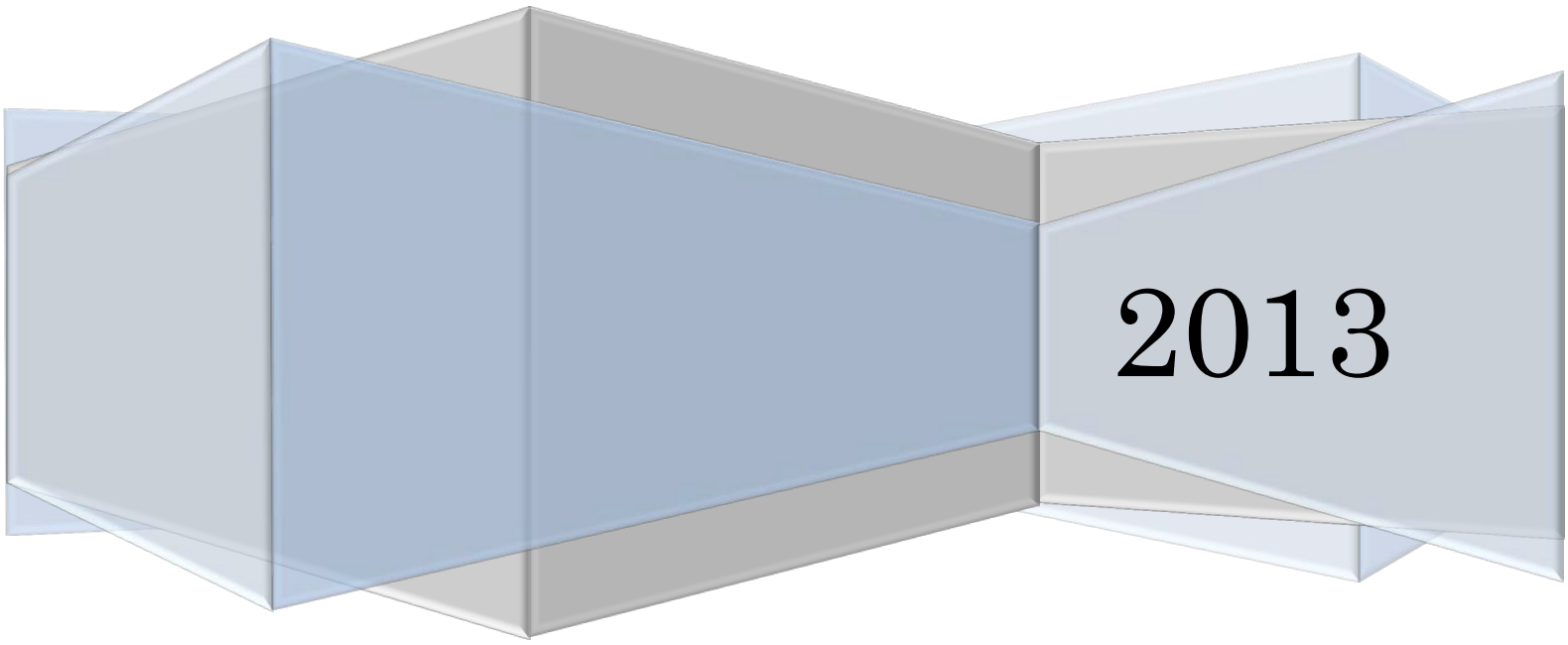
コンカレント日本株式会社

RCIM ETI 使用法

ETI:Edge-Triggered Interrupts

プロフェッショナルサービス部



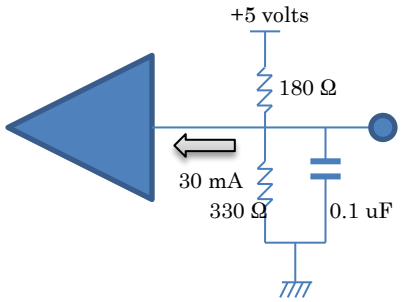
RCIM には、『外部からトリガを受信し、それに同期をさせてソフトに割り込みをする』目的のため、外部から信号を入力できる ETI（エッジトリガーインタラプト）が用意されています。

RCIM III and RCIM II support twelve ETIs (0-11);
RCIM I supports four ETIs (0-3).

RedHawk 側のデバイス名は「/dev/rcim/eti*X*」となります

詳細は、下記 URL の" Real-Time Clock and Interrupt Module (RCIM) Userís Guide"
http://redhawk.ccur.com/docs/root/1redhawk/Hardware/0898007-610.pdf
を見ていただきたいのですが、下記のような TTL 入力仕様になっています。

The external interrupt input signals are 5 volt TTL levels. The external interrupt outputs (labeled EXT_PIG[0-11]) are driven using a 74ABT16240 line driver. The external interrupt inputs are terminated with 180 ohms to +5 volts, 330 ohms and 0.1 uf to ground. To drive this input requires a line driver that can sink at least 30 ma. The input termination limits the speed of the external interrupt signals and helps prevent noise from causing spurious interrupts. Since most line drivers can sink more current than they can source, the falling edge of the signal will be faster.



この入力端子に、信号を入力していただきますと、レベル、立ち上がり、または、立下りのエッジトリガーで非同期に SIGNAL 処理することが出来ます。
この入力仕様はユーザで変更可能ですので、cat /proc/driver/rcim/config でご確認ください。

```
$ cat /proc/driver/rcim/config
h/Not_Configured, sync/ptr
pig0|out0, pig1|out1, pig2|out2, pig3|out3, pig4|out4, pig5|out5
pig6|out6, pig7|out7, pig8|out8, pig9|out9, pig10|out10, pig11|out11
none|di0/f, none|di1/f, none|di2/f, none|di3/f, none|di4/f, none|di5/f
```

```
none|di6/f, none|di7/f, none|di8/f, none|di9/f, none|di10/f, none|di11/f
eti0/f, eti1/f, eti2/f, eti3/f, eti4/f, eti5/f
eti6/f, eti7/f, eti8/f, eti9/f, eti10/f, eti11/f

$ cat /proc/driver/rcim/eti0
eti0: armed, enabled, not pending, vecnum 0, signo 34 sigpid 6379,
nopens 1, keepalive NO.
```

設定例

Falling(立下りエッジ)　デフォルト

```
$ echo eti0/f > /proc/driver/rcim/config
```

Rising(立ち上がりエッジ)

```
$ echo eti0/r > /proc/driver/rcim/config
```

Low(Low レベル)

```
$ echo eti0/l > /proc/driver/rcim/config
```

High(High レベル)

```
$ echo eti0/h > /proc/driver/rcim/config
```

大まかには下記のような手順で利用していただければ、入力した信号のタイミングで、SIGRTMIN が発生いたします。

```
{
        int fd;
        if((fd = open("/dev/rcim/eti1", O_RDONLY)) == -1)
        {
                return(-1);
        }
        if(ioctl(fd, ETI_ATTACH_SIGNAL, SIGRTMIN) == -1)
        {
                 return(-1);
        }
        if(ioctl(fd, ETI_ARM) == -1)
        {
                return(-1);
        }
        if(ioctl(fd, ETI_ENABLE) == -1)
        {
                return(-1);
        }
        return 0;
}
```

また、この ETI を利用する際のオンラインマニュアルおよび動作試験プログラムを、以下に示します。

試験プログラムｈ,コンパイルした後、下記のようにテストをお願い致します。

例)　./eti /dev/rcim/eti1 1

テスト実施時、外部からの信号は不要です。

```
/*
 * ETI special file names under PowerMAX have the form /dev/reti/eti##.
 * There are two numeric digits in the name, i.e. eti0 is /dev/reti/eti00.
 * There are 4 ETIs per CPU board.
 */

#include <stdio.h>
#include <fcntl.h>
#include <sys/file.h>
#include <sys/signal.h>
#include <linux/rcim_eti.h>
//MODIFY    #include <sys/eti.h>
#include <string.h>
#include <errno.h>
#include <stdlib.h>
#include <signal.h>

#define ERRHDR "** Edge Triggered Interrupt Test ¥n"

#define ERR_NULL  0
#define ERR_OPEN  1
#define ERR_CLOSE 2
#define ERR_IOCTL 3

char * case_msg = NULL;

#define DELAY_COUNT (100000)
unsigned long intr_wait = 99*DELAY_COUNT; /* wait for expected edge int */
unsigned long spur_wait = DELAY_COUNT;  /* wait for spurious edge int */

unsigned long delay;
#define DELAY(WAIT) ¥
   for ( delay=0; delay < WAIT && !interrupt_received; delay++ );

#define MAXDEVPATHSZ    64
static char devname[MAXDEVPATHSZ];

#define DEFAULT_COUNT   100
int loop = DEFAULT_COUNT;

#define TRUE  1
#define FALSE 0

volatile int interrupt_received = FALSE;
int debug = FALSE;

int interrupt_signal();
extern int errno;

/*****************************************************************************
*
*  ETI main test
*
*****************************************************************************/
main(argc, argv)
int argc;
char **argv;
{
   int fd;
   int count;

   if ( argc < 2 )
   {
      printf( "No ETI device pathname argument.¥n" );
      exit(1);
```

```
}

if ( argc > 2 )
   debug = TRUE;

strcpy( devname, argv[1] );

printf( "ETI test: %s¥n", devname );


/* Open the specified ETI device */

if ( (fd = open( devname, O_RDWR )) == -1 )
{
   etitst_err( ERR_OPEN, ERRHDR );
   exit(1);
}
eti_attach_signal( fd, SIGUSR1, interrupt_signal );

for ( count = 0; count < loop; count++ )
{
   /* case 1 */
   case_msg = "Case: clear, arm, enable, request";
   eti_clear(fd);
   eti_arm(fd);
   eti_enable(fd);
   eti_request(fd);
   DELAY(intr_wait);         /* wait for signal */
   if ( !interrupt_received )
   {
      etitst_err( ERR_NULL, ERRHDR );
      printf( "** ERROR: Interrupt NOT received on: %s¥n", devname );
      exit(1);
   }

   /* case 2 */
   case_msg = "Case: clear, arm, disable, request";
   eti_clear(fd);
   eti_arm(fd);
   eti_disable(fd);
   eti_request(fd);
   DELAY(spur_wait);         /* wait for signal */
   if ( interrupt_received )
   {
      etitst_err( ERR_NULL, ERRHDR );
      printf( "** ERROR: Unexpected interrupt received on: %s¥n", devname );
      exit(1);
   }

   /*
    *  case 3
    */
   case_msg = "Case: clear, arm, disable, request, enable";
   eti_enable(fd);
   DELAY(intr_wait);         /* wait for signal */
   if ( !interrupt_received )
   {
      etitst_err( ERR_NULL, ERRHDR );
      printf( "** ERROR: Interrupt NOT received on: %s¥n", devname );
      exit(1);
   }
}                            /* end loop  */

/*
 *  clean up and exit
 */
eti_clear(fd);
eti_disarm(fd);
eti_disable(fd);

close(fd);

exit(0);
```

```
}


/*****************************************************************************
*
*  ioctl error
*
*****************************************************************************/
ioctl_err( cmd )
char *cmd;
{
   etitst_err( ERR_IOCTL, ERRHDR );
   printf( "error in performing an %s ioctl¥n", cmd );
   exit(1);
}

/*****************************************************************************
*
*  arm ETI
*
*****************************************************************************/
eti_arm( fd )
int fd;
{
   if ( debug )
      printf("ETI_ARM¥n");

   if (ioctl( fd, ETI_ARM, 0 ) == -1)
      ioctl_err("ETI_ARM");
}

/*****************************************************************************
*
*  disarm ETI
*
*****************************************************************************/
eti_disarm( fd )
int fd;
{
   if ( debug )
      printf("ETI_DISARM¥n");

   if (ioctl( fd, ETI_DISARM, 0 ) == -1)
      ioctl_err("ETI_DISARM");
}

/*****************************************************************************
*
*  enable ETI
*
*****************************************************************************/
eti_enable( fd )
int fd;
{
   if ( debug )
      printf("ETI_ENABLE¥n");

   if (ioctl( fd, ETI_ENABLE, 0 ) == -1)
      ioctl_err("ETI_ENABLE");
}

/*****************************************************************************
*
*  disable ETI
*
*****************************************************************************/
eti_disable( fd )
int fd;
{
   if ( debug )
      printf("ETI_DISABLE¥n");

   if (ioctl( fd, ETI_DISABLE, 0 ) == -1)
```

```
      ioctl_err("ETI_DISABLE");
}

/*****************************************************************************
*
*  set ETI request
*
*****************************************************************************/
eti_request( fd )
int fd;
{
   if ( debug )
      printf("ETI_REQUEST¥n");

   if (ioctl( fd, ETI_REQUEST, 0 ) == -1)
      ioctl_err("ETI_REQUEST");
}

/*****************************************************************************
*
*  Attach handler to signal
*
*****************************************************************************/
eti_attach_signal( fd, sig, func )
int fd;
int sig;
void (*func)();
{
   struct sigaction sigs;               /* sigaction(2) */

   if ( debug )
      printf("ETI_ATTACH_SIGNAL¥n");

   sigaction(sig, (struct sigaction*)NULL, &sigs);
   sigs.sa_handler = (void (*)())func;
   sigaction(sig, (struct sigaction*)(&sigs), (struct sigaction*)NULL);

   if (ioctl( fd, ETI_ATTACH_SIGNAL, (char*)sig ) == -1)
      ioctl_err("ETI_ATTACH_SIGNAL");
}

/*****************************************************************************
*
*  Clear any pending interrupts
*
*****************************************************************************/
eti_clear( fd )
int fd;
{
   if ( debug )
      printf("Clear ETI¥n");

   eti_arm(fd);
   eti_enable(fd);
   DELAY(spur_wait);            /* wait for signal pending, in case */

   interrupt_received = FALSE;
}

/*****************************************************************************
*
*  signal handler
*
*****************************************************************************/
interrupt_signal()
{
   interrupt_received = TRUE;

   signal( SIGUSR1, (void*)interrupt_signal );

   if ( debug )
      printf("*** Interrupt ***¥n");
}
```

```
/*****************************************************************************
*
*  error reporting
*
*****************************************************************************/
etitst_err( tst, err )
int tst;
char *err;
{
   printf( err );
   printf( "** ETI device: %s¥n", devname );

   if ( case_msg )
      printf( "** %s¥n", case_msg );

   if ( tst == ERR_OPEN )
   {
      printf( "open() failed(), (errno = %d)¥n", errno );
   }
   else if ( tst == ERR_IOCTL )
   {
      printf( "ioctl() failed(), (errno = %d)¥n", errno );
   }
}
```

NAME

rcim - Real-Time Clock and Interrupt Module

DESCRIPTION

The Real-Time Clock and Interrupt Module (RCIM) is a standard PCI-based card that provides synchronized clocks, edge-triggered interrupts, real-time clocks, and programmable interrupts.

If RCIM boards of various systems are chained together, an interrupt can be distributed to all connected RCIMs, and from there to all the associated host systems.

This man page provides a summary overview of all versions of the RCIM board and driver. "RCIM" refers to common functionality among the versions. A specific RCIM board version is designated in full; e.g., "RCIM II". For more details on various subsystems, see rcim_rtc(4), rcim_eti(4), rcim_pig(4), rcim_distrib_intr(4), rcim_sync_clocks(4).

Types of RCIM's

RCIM-I is the earliest design. It fitted into a PCI slot and used custom-built analog cables to chain the RCIM-I's of various systems together. This variant also sports either 4 or 8 of each interrupt type (RTCs, PIGs, ETIs, ETOs, and DIs). Also on board were two time-of-day counters: a simple tick counter and a Posix time counter, both incremented every 400 nsecs. When cabled together, all the slave (downstream) RCIM-I's would tick in unison with that of the master (head-of-chain) RCIM. However, even though hardware made these counters tick in unison, there was no hardware support to get the Posix time-of-day initial values to agree, and only limited (simultaneous zeroing) support available to get the tick counter values to agree.

The RCIM-II improved on this by converting the form factor from PCI to PCI-X, by chaining RCIM's together via standard Ethernet cables rather than via custom-built cables, by using digital (serialized packet) rather than analog signaling down the cables, by expanding all interrupt types from 8 to 12 examples of each, by providing the operating system two methods to finely control the operational frequency of the RCIM, and, if cabled together, to transmit this modified operational frequency from master to all the slaves, so that they too could operate in unison with the master. Various high precision crystals are available as options, and in addition, an optional GPS module is available which the provided system software will periodically query to determine how much the RCIM must be sped up or slowed down by to match world atomic time.

The RCIM-III, the latest and best of the pack, improved on the RCIM-II by going to the PCIe form factor, by supporting MSI-X interrupts, by going to optical cabling as the default, and by the Master RCIM sending down the cable, once per second, its notion of Posix time. Each of the slave systems can now, through provided software, use this broadcasted timestamp to make its slaved RCIM's posix time exactly match up to that of the master's (see rcimdate(8)). In this model, only one RCIM .. the master .. needs a GPS, all the slave RCIMs need only to have their Posix time track the master's in order to get the same benefits.

RCIM-III's come in two versions, an earlier version which only supports isolated mode, and a later version that supports cabling as described above.

Connection Modes

An RCIM module may be used in one of four modes, depending on how it is cabled:

Isolated Mode There are no connections to any other RCIM.

Master Mode The RCIM is at the head of a chain of RCIMs. There is no

cable connection going into the RCIM, only a cable connection going out.

Pass-through Slave Mode
The RCIM is connected to two other RCIMs--there is an input cable connection coming from the previous RCIM in the chain, and an output cable connection going to the next RCIM in the chain.

Final Slave Mode
The RCIM is connected to exactly one other RCIM. There is an input cable connection going into a final slave RCIM, but no output cable connection coming out of it.

The RCIM board has hardware to sense the combination of connected cables and automatically determines the operating mode.

Note that all RCIMs connected in a chain must be of the same model--e.g., all RCIM IIs.

Synchronized Clocks

An RCIM board provides two synchronized clocks: the tick clock and the POSIX clock. The tick clock is a 64-bit non-interrupting counter that counts by one on each tick of the common 400ns clock signal. The tick clock cannot be set to a specific time, it can only increment or be set to zero. Hence the tick clock cannot be adjusted on the fly to approximate the current time of day, as would be required of a true time-of-day clock.

The POSIX clock is a 64-bit non-interrupting counter encoded in POSIX 1003.1 format. The upper 32 bits contain seconds and the lower 32 contain nanoseconds. This clock is incremented on each tick of the common 400ns clock signal. This clock can be set to a specific time.

When the RCIM board is part of an RCIM chain, then all clocks on all the RCIM boards in the chain are incremented in unison, as they are all driven by a common 400ns clock signal emanating from the master RCIM.

When an RCIM board is part of an RCIM chain, the tick clocks on all slave RCIMs become read-only. They are incremented and cleared in synchronization with whatever incrementing and clearing is being done to the tick clock located on the master RCIM. However, the POSIX clocks each remain read-write, and a value written to the POSIX clock of one RCIM board does not change the value of the POSIX clocks of any of the other RCIM boards in the chain.

The tick clock can be read on any system, master or slave, using direct reads when the device file /dev/rcim/sclk is mapped into the address space of a program. Information on how to do this may be found in rcim_sync_clocks(4).

Tick clock initialization (zeroing) can be invoked using the rcim_clocksync(1) command. The rcim_clocksync(1) command can only be performed on an isolated or a master RCIM. Tick clock synchronization can also occur automatically whenever the RCIM master boots, but this is disabled by default due to its potentially disruptive nature on the slave systems. See the rcim_clocksync(1) man page for more details.

The POSIX clock is accessed in a manner similar to the tick clock in that the same utilities and device files are used.

The RCIM-I and II boards have no hardware support for setting the POSIX clocks of all boards to a consistent value. It is possible to pause the operation of the RCIM board chain, and then synchronize in software the POSIX clocks during the pause, using the rcim_clocksync(1) utility. This requires setup and an operational TCP/IP network between the systems of the chain.

The RCIM-III board periodically transmits its Posix time-of-day value from the master to all the slaves. This is done once per second on the second mark. The slaves, in turn, snapshot this and snapshot their own Posix time-of-day values into special read-only registers on the RCIM.

With these snapshots system software, if so desired, can keep the Posix time of each slave exactly in sync with that of the master. See rcim-date(8) for more details.

On an RCIM system equipped with the optional GPS module and ntpd running, the POSIX clock on the RCIM containing the GPS module is synchronized to GPS time.

Edge-Triggered Interrupts

Each RCIM board has incoming external interrupt lines, called ETIs or Edge-Triggered Interrupts, so named after their most common mode of operation. These lines permit users to provide their own interrupt sources. The RCIM processes and delivers these interrupts to the host system and, if configured to do so, routes and delivers them to all other RCIMs in the chain as distributed interrupts. RCIMs II and III support twelve ETIs (0-11); RCIM I supports four (0-3).

Each ETI can be configured independently of the others. An ETI may treat the incoming signal as an edge or level sensitive interrupt. If edge sensitive, it may raise an interrupt on either the rising or the falling edge. If level sensitive, it may raise interrupts for either the high or the low signal value. Configuration parameters are selected when the RCIM board and driver are initialized, and remain constant for the life of the driver, or until changed through the /proc/driver/rcim/config interface. See CONFIGURATION below for details.

Applications in turn arm or disarm, enable or disable each ETI on each system on the fly, as appropriate to the needs of those applications.

In addition to delivering its ETI interrupts to the attached host system, each RCIM board can be configured to route the output of any or all of its ETIs to some set of the distributed interrupt lines. In this way an ETI can be broadcast and delivered to all interested hosts that are in the RCIM chain. ETIs may also be routed to the RCIM board's external output interrupt lines.

For more information on how to configure and use ETIs, see the CONFIGURATION section of this man page, rcim_eti(4), or the ETI description in the Real-Time Clock and Interrupt Module (RCIM) User's Guide, pub number 0898007.

Distributed Interrupts

The real heart and power of the RCIM board lies in its distributed interrupt (DI) system. These are signal lines that are shared among all RCIM boards in an RCIM chain. For each distributed interrupt line, one RCIM board is configured to drive that line with one of its internal signal sources (RTCs, ETIs, PIGs). All boards, including the board that is driving the line, are capable of listening in on the line and sending an interrupt to the attached host system, if so configured. RCIMs II and III support twelve DIs (0-11); RCIM I supports eight (0-7).

Each RCIM board must configure each incoming distributed interrupt signal line as either an edge-triggered interrupt that triggers on the rising or falling edge, or as a level triggered interrupt that triggers on the high or the low value.

Applications in turn arm or disarm, enable or disable each distributed interrupt on each system on the fly, as appropriate to the needs of those applications.

For more details on how to configure and use distributed interrupts, see the CONFIGURATION section of this man page, rcim_distrib_intr(4), or the Distributed Interrupt description in the Real-Time Clock and Interrupt Module (RCIM) User's Guide.

External Output Interrupts

Each RCIM board has a connector which provides output interrupt lines. Equipment can be attached to and be controlled by those lines. Any signal source available within the RCIM, including the distributed interrupt lines, can be configured to drive these external output

interrupt lines. RCIMs II and III support twelve external output interrupts (0-11); RCIM I supports four (0-3).

For more information on external output interrupts, see the CONFIGURATION section of this man page or the External Output Interrupts description in the Real-Time Clock and Interrupt Module (RCIM) User's Guide.

Real-Time Clocks

Real-time clocks (RTCs) are decrementing counters that generate an interrupt each time they reach zero. They may be one-shot or periodic; if periodic, the original load value is automatically reloaded into the counter each time zero is reached. The resolution of each RTC is programmable, ranging from 1 microsecond to 10 milliseconds per tick. RCIMs II and III support eight RTCs (0-7); RCIM I supports four (0-3).

In addition to delivering its RTC interrupts to the attached host system, each RCIM board can be configured to attach the output of any or all of its RTCs to some set of the distributed interrupt lines. In this way an RTC interrupt can be broadcast and delivered to all interested hosts that are in the RCIM chain. RTC interrupts can also be used as a source for an RCIM board's external output interrupts.

For more details on how to configure and use RTCs, see the CONFIGURATION section of this man page, rcim_rtc(4), or the Real-Time Clocks section of the Real-Time Clock and Interrupt Module (RCIM) User's Guide.

Programmable Interrupts

A programmable interrupt generator (PIG) is an output signal generator capable of switching between a high and low signal value under direct control of user software. This can be used as an interrupt source to drive any combination of distributed interrupt lines or external interrupt output lines. RCIMs II and III support twelve PIGs (0-11); RCIM I supports four (0-3).

For more details on how to configure and use PIGs, see the CONFIGURATION section of this man page, rcim_pig(4), or the Programmable Interrupt Generator section of the Real-Time Clock and Interrupt Module (RCIM) User's Guide.

Proc Filesystem Interface

The RCIM driver makes its internal state viewable through various proc filesystem files. Unless otherwise stated these files are read-only. They may be found in directory /proc/driver/rcim and are:

config — The RCIM configuration, displayed in a form that can be cut and pasted as the value for an RCIM configuration variable.

A write of a suitably formatted string to this file will change the configuration. See CONFIGURATION below for details.

interrupts — A count of all incoming RCIM ETI, distributed, and RTC interrupts, in total and per cpu.

status — Various RCIM board status values not displayed in any of the other procfs files.

rawregs — A commented hexadecimal display of all readable RCIM board registers.

rtcN — The status of each of the real-time clocks: whether each is running, what their current countdown values are, etc.

etiN — The status of each of the ETIs: whether each is armed, enabled, etc.

diN — The status of each of the distributed interrupt lines: whether each is armed, enabled, etc.

Device Interface

The RCIM driver makes its services available to applications via a

character device driver interface. The following special files in /dev/rcim are used:

```
rcim    master rcim board status/control
sclk    access to the RCIM tick and POSIX clocks
rtcN    real-time clocks 0 through 7
etiN    external interrupts 0 through 11
diN     distributed interrupts 0 through 11
pigN    programmable interrupt generators 0 through 11
```

The initialization script /etc/init.d/rcim creates the RCIM special files automatically on each system boot. If the driver is loaded at other times, then the user should take care to run this script at those times also.

PROGRAM INTERFACE

An application controls an RCIM by invoking open(2) close(2), ioctl(2), and mmap(2) calls against special file /dev/rcim/rcim. Other special files exist and are owned and documented by their respective sub-drivers:

- rcim_rtc(4)
- rcim_eti(4)
- rcim_pig(4)
- rcim_distrib_intr(4)
- rcim_sync_clocks(4)

The following ioctl operations are available for /dev/rcim/rcim:

RCIM_GET_INFO

```
#include <sys/types.h>
#include <rcim.h>
ioctl (fildes, command, arg)
int fildes, command;
struct rcim_ginfo *arg;
```

Returns information about the RCIM, including version numbers, connection modes and tunable values. The layout of struct rcim_ginfo is in /usr/include/rcim.h.

RCIM_GET_ADDR

```
#include <sys/types.h>
#include <rcim.h>
ioctl (fildes, command, arg)
int fildes, command;
struct rcim_gaddr *arg;
```

Returns the virtual and physical address of the RCIM control registers. The layout of struct rcim_gaddr may be found in /usr/include/rcim.h.

The mmap(2) system call may be used to map in some or all of the device registers of the RCIM board. For the register layout, see the data structure rcim_mem_t in /usr/include/linux/rcim_ctl.h.

CONFIGURATION

The RCIM driver can be configured to be either a module or be statically linked into the kernel. This is selected at the time the kernel is built from source, via the RCIM configuration option accessible through the Character Devices selection of the Kernel Configuration GUI.

When the RCIM driver initializes, it looks for two possible configuration options. For a statically linked RCIM driver, these tunables can be specified on the LILO or Grub command line. For an RCIM driver in module form, tunables can be specified on the insmod(8) command line, or be placed in modprobe.conf(5), where the modprobe(8) invocation in the startup script /etc/init.d/rcim will find them.

The rest of this section discusses the rcim tunable option. See the following MSI INTERRUPT CONFIGURATION section for information about MSI

interrupts.

The rcim tunable option (rcim=RCIMoptions) may be used to set various rcim configuration options. For a statically linked RCIM driver, this tunable can be specified on the LILO or Grub command line. For an RCIM driver in module form, this tunable can be specified on the insmod(8) command line, or be placed in modprobe.conf(5), where the modprobe(8) invocation in the startup script /etc/init.d/rcim will find it.

The rcim tunable accepts a comma-separated list of option names. For example:

```
$insmod rcim rcim=' eti1/rising, di3/high, eti0|di6'
```

In this example, eti #1 is configured to trigger on the rising edge, distributed interrupt #3 triggers on a high value, and distributed interrupt #6 is to be driven by eti #0.

A write of a suitably formatted string to /proc/driver/rcim/config will change the configuration. Example: echo eti2/r >/proc/driver/rcim/config changes ETI #2 to trigger on a rising edge. It is recommended that changes be made only when the RCIM is not in use.

Available Configuration Options

Options are comma separated and are processed from left to right. If the same option is specified more than once, the rightmost option is the one in effect when the RCIM driver finishes its configuration.

Any option state not specified is left at its default value.

Available options include:

diN/[rising|falling|high|low]

Configures one of the distributed interrupts to trigger on the rising edge, the falling edge, on a high signal value or on a low signal value.

The flags (rising, falling, etc.) can be reduced to a single character and are case insensitive.

Examples:

```
di0/high
di5/L
di7/Rising
```

Default: falling

etiN/[rising|falling|high|low]

Configures one of the ETIs to trigger on the rising or falling edge of its input signal, or on the high or low signal value.

The flag words (rising, falling, etc.) can be reduced to a single character and are case insensitive.

Examples:

```
eti0/falling
eti1/r
eti2/h
eti3/LOW
```

Default: falling

source|diN

Configures one of the distributed interrupt lines to be driven by one of the signal sources within the RCIM. Available sources include:

- pigN - one of the PIGs.
- rtcN - one of the RCIM RTCs.
- etiN - one of the RCIM ETIs.
- none - This RCIM is not to drive this distributed interrupt.

Examples:

none|di0
pig1|di1
rtc3|di3

Default: none

source|outN
Configures one of the RCIM external output lines to be driven by the given source. Available sources include:

rtcN - drive the output line with this RTC.
pigN - drive the output line with this PIG.
etiN - drive the output line with this ETI.
diN - drive the output line with this distributed interrupt.
none - let the interrupt output line float.

Examples:
rtc3|out0
di5|out2

Defaults:
pig0|out0
pig1|out1
pig2|out2
pig3|out3
etc.

host/hostname
Specifies the Internet name of the system to which the master RCIM board is attached. Though not used by the RCIM driver, this value is supplied to applications that ask for it.

Default: Not_Configured

sync | nosync (RCIM I)
sync | nosync [r|t|p] (RCIM II,III)
Specifies if this RCIM is to use its local clock (nosync) or use the cable clock driven by the master RCIM's local clock (sync).

On RCIM IIs and III, the timing sources can be configured separately. The default is all.
r real-time clock
t tick clock
p POSIX clock

Default: sync

Examples:
nosync
sync/tp
nosync/r

clock | noclock
Specifies if this RCIM is to be registered as a clocksource for the system. The default is "clock". Note that once the RCIM is registered as a clocksource, it cannot be "unregistered." Also, if the RCIM is configured as a module and registered as a clocksource, it is locked in (rmmod rcim will fail).

MSI INTERRUPT CONFIGURATION
The latest version of RCIM III supports the use of MSI interrupts and by default, the rcim kernel driver will initialize the hardware to use MSI interrupts instead of PCI INTA interrupts whenever possible. By using MSI interrupts, the RCIM III is guaranteed of having its own non-shared interrupt, thus providing more reliable interrupt response times.

In addition to the rcim tunable option, the disable MSI option (nomsi=1) may be used to disable MSI interrupts on RCIM III cards. When this option is specified, the rcim driver will fallback to using the PCI INTA interrupt method. For performance reasons, this option should only be used if a problem with MSI interrupts is suspected.

FILES

/dev/rcim/rcim, /usr/include/rcim.h, /usr/include/linux/rcim_ctl.h, /dev/rcim/sclk, /proc/driver/rcim, /etc/sysconfig/rcim.

SEE ALSO

rcim_rtc(4), rcim_eti(4), rcim_pig(4), rcim_distrib_intr(4), rcim_sync_clocks(4), modprobe.conf(5), insmod(8), modprobe(8), rcim_clocksync(1), rcimdate(8).

COPYRIGHT

NAME

rcim_eti - RCIM edge-triggered interrupt character device driver

DESCRIPTION

The RCIM edge-triggered interrupt (ETI) device driver provides a software interface to the external input interrupt lines available on an RCIM board. A special device file is associated with each possible RCIM ETI device.

ETI Hardware Features

An ETI is an external interrupt input line. A board connector makes twelve such lines available on each RCIM II and III board (0-11); four (0-3) on each RCIM I board. The intent is for users to attach appropriately designed signal generating equipment to these lines. The RCIM will continuously examine the signal values on these lines, generating interrupts whenever the input signal value meets certain (preconfigured) conditions.

One requirement the RCIM imposes on attached equipment is that the signal being fed into an RCIM ETI must hold any low or high value for at least 1.5 microseconds before changing to the next state. If this requirement is not met, the ETI may not generate all the interrupts the user would otherwise expect.

How the incoming signal is converted to interrupt requests is part of the configuration of each ETI, performed at board initialization time and remains constant for the life of the driver, or until changed through the /proc/driver/rcim/config interface. See rcim(4) for details. The possible configuration states are:

edge-triggered, rising edge
: Each rising edge is treated as an interrupt request.

edge-triggered, falling edge
: Each falling edge is treated as an interrupt request.

level-triggered, high level
: The interrupt is repeatedly requested as long as the signal is being held high.

level-triggered, low level
: The interrupt is repeatedly requested as long as the signal is being held low.

These interrupt requests are fed into an interrupt engine maintained for each ETI by the RCIM board. This engine operates as follows:

Because interrupt requests are really attempts to set the request bit of the ETI, these attempts are not allowed to succeed unless the ETI has its armed bit set.

When the armed bit is not set, the request bit is summarily cleared on each tick of the RCIM clock. This in effect turns off the ETI.

As long as the request bit remains set, the RCIM periodically tries to move this bit to the pending bit and clear the request bit at the same time.

This operation is not allowed to proceed as long as the pending bit remains set from some previous transfer. It also will not proceed unless the enable bit is set.

The enable bit may be thought of as granting the RCIM board permission to deliver any interrupt requests that it has accepted. When the ETI is disabled, the ETI accepts interrupt requests, but delays delivering the interrupt until it is re-enabled.

The value of the pending bit is what the ETI outputs. When the

pending bit is set the ETI is outputting an interrupt to whatever other subsystems that ETI has been routed to. When the pending bit is clear the ETI is not trying to output an interrupt. The host computer to which the RCIM board is attached always has the ETI pending bit routed to it. Other possibilities include RCIM boards and the RCIM external output interrupt lines. These routing possibilities are documented in rcim(4).

The ETI interrupt handler on the attached host computer must clear the pending bit each time it finishes processing a delivered interrupt. This prepares the way for the RCIM board to immediately process a fresh instance of this ETI interrupt, should one have been requested while the driver was busy processing the previous instance. If a request is not outstanding, prepares the way for an interrupt to be generated the next time an interrupt request is made.

From an operational point of view, an ETI is manipulated as follows:

1. An ETI must be armed for anything to happen. If it is not armed, the ETI summarily clears any unprocessed interrupt request and does not accept any new interrupt requests, even those generated by software. This state is changeable with the ioctl(2) commands ETI_ARM and ETI_DISARM.

2. An ETI must be enabled for successfully requested interrupts to be delivered. If the ETI is not enabled, interrupt requests are accepted but they are not passed on. This state is changeable with the ioctl(2) commands ETI_ENABLE and ETI_DISABLE.

3. An ETI input signal line is not the only way an interrupt request can be generated. The RCIM permits software to set the request bit directly using the ioctl(2) command ETI_REQUEST.

Accessing the ETI Files

Each ETI is referenced through its own special device file:

/dev/rcim/etiN

where N is the ID of the ETI. The files are created automatically on system boot by the /etc/init.d/rcim initialization script.

User Interface

An ETI is controlled by open(2), close(2), and ioctl(2) system calls. The read(2), write(2) and mmap(2) commands are not used by this driver.

The open(2) call assigns a file descriptor to one ETI.

A close(2) call frees the file descriptor and, if it is the last close, disarms the ETI if the IOCTLKEEPALIVE state is not set.

All device manipulation is done using ioctl(2). The #defines used below can be found in /usr/include/rcim.h.

Note that all of the ioctl(2) commands documented below can also be applied to distributed interrupt devices, as documented in rcim_distrib_intr(4). Thus these ioctl's can be used when one is writing generic code that doesn't particularly care whether the device is an ETI or a DI.

The following ioctl(2) calls have the form:

```
#include <sys/types.h>
#include <sys/ioctl.h>
#include <rcim.h>

ioctl(fildes, command, 0)
int fildes, command;
```

The commands are:

ETI_ARM Arms the edge-triggered interrupt.

ETI_DISARM Disarms the edge-triggered interrupt.

ETI_ENABLE Enables the edge-triggered interrupt.

ETI_DISABLE Disables the edge-triggered interrupt.

ETI_REQUEST Generates the edge-triggered interrupt using software. This sets the same ETI request bit that the incoming signal line sets when it detects an interrupt condition.

ETI_WAIT Causes the process to sleep until the next ETI interrupt from this device, or until an ETI_WAKEUP is performed on this device by another task, or until the task is interrupted by a signal.

ETI_WAKEUP Wakes up all processes currently sleeping in ETI_WAIT on this device.

The following ioctl(2) calls have the form:

```
#include <sys/types.h>
#include <sys/ioctl.h>
#include <rcim.h>

ioctl(fildes, command, arg)
int fildes, command, arg;
```

The commands are:

ETI_VECTOR
Associates the Linux interrupt handlers at IRQ arg with this ETI. Whenever this ETI fires, the interrupt handlers at the associated IRQ are also invoked. An arg of zero clears a previously established association. An ETI will not accept a new IRQ association until a previously established association has been removed.

This service allows other Linux drivers to register their interrupt handlers, which themselves have no hardware interrupt, with the RCIM's RTC interrupt handler. The Frequency Based Scheduler (FBS) is one such user. This service is not intended to be called by application code.

Other drivers can find a safe, unused IRQ for their handlers using the kernel function find_unused_irq(). They can then attach their handlers to this IRQ using request_irq() and finally they can attach that driver to some ETI via this ioctl(2) command.

IOCTLVECNUM
The generic ioctl command IOCTLVECNUM is identical to ETI_VECTOR and may be used in those situations where one does not care what device type (be it ETI, RTC, DI, or whatever) to which the additional IRQ is being attached.

ETI_KEEPALIVE
A non-zero value for arg sets the keepalive state for this ETI. If arg is zero, the keepalive state for this ETI is cleared.

When the keepalive state is set for an ETI then the ETI does not shut down when the final close to the associated ETI device is made. When the keepalive state is reset, the ETI device is automatically shut down on final close.

IOCTLKEEPALIVE
A generic version of ETI_KEEPALIVE that may be used in those situations where one does not care what type of device (be it ETI, RTC, DI, or whatever) is being ordered to remain alive after close.

ETI_ATTACH_SIGNAL
If arg is non-zero then it is a signal number, and each time this

ETI generates an interrupt this signal will be sent to the process thread group which made the call. If arg is zero then signal sending capability for this ETI, if it was enabled, is disabled.

IOCTLSIGATTACH
A generic version of ETI_ATTACH_SIGNAL that may be used in those situations where one does not care what type of device (be it ETI, RTC, DI, or whatever) is being ordered to send signals.

The following ioctl(2) commands expect arg to be a pointer to an integer value:

ETI_GETICNT
Returns the number of times this ETI has fired.

IOCTLGETICNT
A generic version of ETI_GETICNT. Potentially supportable by any driver that would like to return interrupt counts to its users.

ETI_INFO
Places information about the edge-triggered interrupt in the integer pointed to by arg. The defined bitfields are:

ETI_TYPE_RCIM
RCIM edge-triggered interrupt. This is set if the device is actually an RCIM based ETI and is reset if it is some other ETI device.

ETI_TYPE_RCIM_MASTER
This ETI is located on the master RCIM.

ETI_TYPE_RCIM_DISTRIB
This ETI drives some RCIM distributed interrupt.

CONFIGURATION

The tunable 'rcim=' controls all configurable features of the RCIM driver, including the setup of ETIs. See rcim(4) for details.

The file /proc/driver/rcim/config provides information on the configuration of the ETIs on the local RCIM board.

FILES

/dev/rcim/etiN, /proc/driver/rcim/etiN, /usr/include/rcim.h

ERRORS

On failure, open(2) and ioctl(2) return -1 and set errno to one of the following error codes.

ENODEV The edge-triggered interrupt device does not exist.

ENXIO The edge-triggered interrupt device is not properly configured.

EBUSY The edge-triggered interrupt is in use by another device driver.

EBADF The values specified for fildes is not a valid open file descriptor.

EINVAL Request or argument is not valid.

EINTR ETI_WAIT was interrupted by a signal.

EPERM Command is IOCTLVECNUM or ETI_VECTOR and the specified IRQ is not within the allowed range of values.

EFAULT Command is IOCTLGETICNT or ETI_GETICNT and the specified integer pointer value for arg is not valid.

EBUSY Command is IOCTLVECNUM or ETI_VECTOR and the device already has an IRQ association.

SEE ALSO
rcim(4), rcim_distrib_intr(4), open(2), close(2), ioctl(2).