RCIM クロック同期

版	日付	記述			
	発行		作成	確認	承認
版	日付 2017/02/27	記述			
2	発行 プロフェッショナルサービス		作成 大島龍博	確認	承認
版	日付 2017/02/22	記述 新規発行			
1	発行 プロフェッショナルサービス		作成 大島龍博	確認	承認
表題 RCIM クロック同期			No PSG-20170222		版 2
					ﾍﾟｰｼﾞ 1/13

RCIM とは

RCIM(Realtime Clock Interrupt Module)は、コンカレント製リアルタイムクロックモジュールです。

RCIM の使用目的としては、以下の３つがあります。
システムクロックの精度向上
ハードウェアに関連付けた FBS(Frequency-Based Scheduler)
異なるシステム間のイベントやクロックを同期させる

コンカレントでは、すべての iHawk 製品が同じ精度のシステムになるように、この RCIM を同梱出荷しています。
RCIM は、400ns 解像度の 64bit カウンタを持っていますが、使用しない場合には、TSC と呼ばれる、CPU クロックを基準としたカウンタを利用します。

RCIM3 の場合には、温度校正された 400ns の±2.5PPM の高精度のクロックオシレーター FOX 294 を使用していますが、TSC の場合には、温度補正されないＰＣ毎に異なる周波数のクロックであり、精度も±100PPM 程度です。（FOX294,RCIM の仕様は表 1,2 を参照ください）

また RCIM には、8ch のリアルタイムクロック、12ch の汎用入出力クロックカウンタが実装されています。
RedHawk では、オプション製品として、このクロックカウンタを利用した FBS(Frequency-Based Scheduler)を、ご用意させて頂いております。

• PART NUMBER SELECTION Learn More - Internet Required

Part Number	Model Number	Frequency Stability	Operating Temperature(°C)	Frequency Range (MHz)
718-Frequency-xxxxx	FOX924B	See table	-30 ~ +85	10.000~30.000
719-Frequency-xxxxx	FOX924E	See table	-30 ~ +85	10.000~30.000

• ELECTRICAL CHARACTERISTICS

PARAMETERS	MAX (unless otherwise noted)
Frequency Range (FO)	10.000 ~ 30.000 MHz
Temperature Range	
Operating (TOPR)	-30ºC ~ +85ºC
Storage (TSTG)	-40ºC ~ +85ºC
Supply Voltage (VDD)	3.3V ± 5%
Input Current (IDD)	6.0mA
Initial Frequency Tolerance @ 25ºC (after 2 reflows)	±1.5PPM
Frequency Stability	
Over Temperature Range	±2.5PPM
Over Supply Voltage Change (3.3V ± 5%)	±0.3PPM
Output Voltage (HCMOS) (VOL)	0.5V
(VOH)	80% VDD Min
Output Load	15pF
Aging per year	±1.0PPM
Startup Time (Ts)	5mS Max
Pullability (Vc = 1.65 ± 1.5V)	±5.0 ~ ±15.0 PPM

All specifications subject to change without notice.

表 1 FOX294 の仕様

Feature	RCIM III	RCIM II	RCIM I
Clocks			
POSIX			
Length	64 bits (two 32-bit words)	64 bits (two 32-bit words)	64 bits (two 32-bit words)
Resolution	High-order 32 bits–1 second Low-order 32 bits–100 nsec	High-order 32 bits–1 second Low-order 32 bits–400 nsec	High-order 32 bits–1 second Low-order 32 bits–400 nsec
Oscillator stability	+/-2.5 PPM	+/-20 PPM	+/-100 PPM
Tick Timer			
Length	64 bits (two 32-bit words)	64 bits (two 32-bit words)	64 bits (two 32-bit words)
Resolution	64 bit counter of 400 ns ticks	64 bit counter of 400 ns ticks	64 bit counter of 400 ns ticks
Real-Time Clocks			
Number	8	8	4
Length	32 bits	32 bits	32 bits
Resolution	1 microsecond (larger values programmable)	1 microsecond (larger values programmable)	1 microsecond (larger values programmable)
Oscillator stability	+/-2.5 PPM	+/-20 PPM	+/-100 PPM
Local Interrupts			
External Edge-Triggered Interrupts	12	12	4
External Output Interrupts	12	12	4
Real-Time Clocks	8	8	4
Distributed Interrupts			
Input	12	12	8
Output	12	12	8
Interrupt Response Time			
Interrupt to user process	< 8 microseconds	< 8 microseconds	< 8 microseconds
Packaging			
Form Factor	PCIe	PCI	PCI
Maximum cable length (See Appendix B for calculations.)	TBS	32 ft.	32 ft.
External Connectors	Molex LFH-60	Molex LFH-60	16 position .1" Latching Header
PCI Performance	x1	66 MHz 64-bit	66 MHz 64-bit
Options	GPS Module	GPS Module	None
Environmental			
Operating Temperature	10° to 40° C	10° to 40° C	10° to 40° C
Storage Temperature	-40° to 65° C	-40° to 65° C	-40° to 65° C
Relative Humidity	10 to 90% (non-condensing)	10 to 90% (non-condensing)	10 to 80% (non-condensing)
Power			
Consumption	~5 watts	~5 watts	~5 watts

表 2 RCIM の仕様

(2a)精度が向上するメカニズム

CPU クロックを使用した場合、CPU の温度上昇により、クロックが温度ドリフトを起こすため、クロックが補正されません。

RCIM のクロックの場合、表 1 FOX294 の仕様の温度補正回路を内蔵したクロックを使用しています。

(2b)精度の性能値(RCIM ボードが有る場合と無い場合)

RCIM3 の場合、2.5ppm で、RCIM を使用しない場合およそ、100ppm です。

PPM は Parts Per Million の頭文字をとったもので、100 万分の 1 を表しています。
例えば、2.5 MHz(400ns) の発振器の周波数精度に±2.5 PPM と記載されている場合は、

2.5 MHz × - 2.5PPM = -6.25 Hz
2.5 MHz × +2.5PPM = +6.25 Hz

という計算結果から、2.5MHz の 2.5ppm は±6.25Hz(13Hz の範囲)でふらつく事を意味し、サーバの水晶発振器の性能が±6.48 秒/月(30 日×24 時間×3600 秒×2.5×10^{-6})と言うことになります。

linux は、x86 使用時、いくつかのクロックソースの選択肢のなかから、TSC クロックカウンタが使用可能であるとき、TSC をシステムクロックに設定するが、RedHawk の場合には,RCIM を選択します。(注：RedHawk6.5 の例)

```
:
tsc: Refining TSC calibration synchronously, will consume one second...
tsc: Refined TSC calibration by -52 ppm to 2596.991 MHz.
Switching to clocksource tsc
:
Masterclock system initialized.
Masterclock "none": registered, rating is 0.
Masterclock "none": activated.
:
Loading Multiboard RCIM driver, major is 250.
Initializing Multiboard RCIM UART sub-driver, major is 204.
rcim 0000:0c:04.0: card #0 found: RCIM-III, isolated, no gps.
pci 0000:0b:00.0: irq 99 for MSI/MSI-X
rcim 0000:0c:04.0: card 0: MSI interrupts enabled, new irq is 99.
UART on RCIM card 0 not initialized, not needed.
Masterclock "rcim:0": registered, rating is 1399.
Masterclock "none": deactivated.
Masterclock "rcim:0": activated.
```

CPU クロックが 2596.991 MHz の場合 TSC の解像度は 385ns で、RCIM の解像度は 400ns に設定されていて、クロックソースの設定は、下記ファイルで確認することが出来ます。

```
# cd /sys/devices/system/clocksource/clocksource0
# ls -l
合計 0
-r--r--r-- 1 root root 4096 12 月  9 12:04 available_clocksource
-rw------- 1 root root 4096 12 月  9 12:04 cache_clocksource
-rw-r--r-- 1 root root 4096 12 月  9 12:04 current_clocksource
```

現在利用可能なクロックソース一覧

```
# cat available_clocksource
rcim tsc hpet acpi_pm jiffies
```

現在利用されているクロックソース

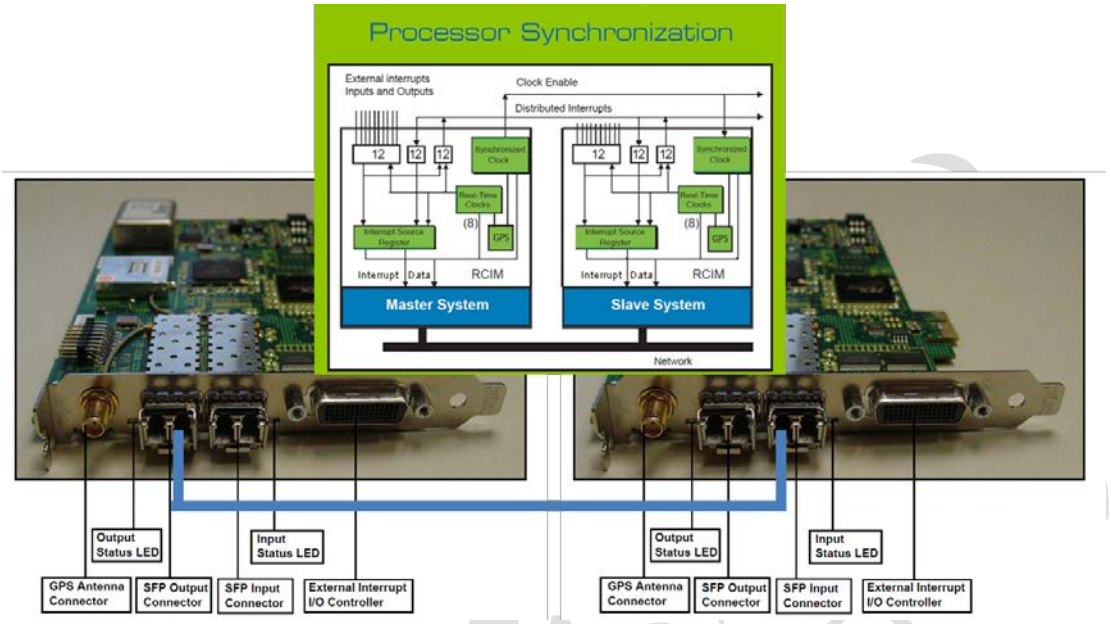
```
# cat current_clocksource
rcim
```

現在キャッシュとして使用されているクロックソース

```
# cat cache_clocksource
tsc
```

クロック同期ハードウェア接続

RCIM マスターの SFP 出力コネクタから RCIM スレーブの SFP 入力コネクタへ接続する。
この時 RCIM30m 光同期ケーブル（HS002-3CBL-30）を使用する。



カーネルのコンフィグレーション

下記の RedHawk Linux カーネル・パラメータは RCIM に関連しています。全てがカーネル構成 GUI の「Character Devices」選択項目を通して利用可能であり、全てのプレビルト RedHawk Linux カーネルでデフォルトで有効となっています。

- **RCIM**

本パラメータは RCIM ドライバをカーネル内に設定します。望むのであれば、これはモジュールとして構成することが可能です。

- **MULTI_RCIM**

コンパイルしてカーネルに組み込まれる RCIM ドライバをマルチ・ボードもしくはシングル・ボードに切り替えます。マルチ・ボード・ドライバがデフォルトで使用されます。

- **RCIM_MASTERCLOCK**

本パラメータは RCIM をシステム・クロックの監視や調整を行うマスター・クロックとして使用することを有効にし、結果、システムの時間管理が更に正確になります。

- **RCIM_PPS**

本パラメータは RCIM ティックと POSIX 時刻のレジスタをオプションの GPS 受信機のパルス/秒サポートに統一します。これは GPS システムで定義された公式な原始時間に更に厳密に調整された時間を提供します。本オプションは RCIM が GPS 機能を持っていない場合は効果がありません。

- **RCIM_IRQ_EXTENSIONS**

本パラメータは他のドライバが自分自身の割込みルーチンを RCIM ドライバに加える事を可能にします。Frequency-Based Scheduler (FBS)はこのサポートを必要とします。

カーネル調整パラメータの変更やカーネルの構築に関する詳細については、「RedHawk Linux User’s Guide」(出版番号 0898004)を参照してください。

ドライバ構成

本項では RCIM ボードの構成に関する 2 つの方法(動的：/proc ファイルへ文字列を書き込む、静的：ブート時またはモジュールをロードする時のオプション)について説明します。シングル・ボード RCIM ドライバは静的と動的をサポートしますが、マルチ・ボード・ドライバは動的な方法のみをサポートします。

- **静的構成**

システムでシングル・ボード RCIM ドライバを初期化する時、構成オプション用の単一調整パラメータ(rcim=RCIMoptions)の値を調べます。静的にリンクされた RCIM ドライバについては、この調整パラメータは GRUB ブート・ローダー・コマンド行で指定することが可能です。モジュール形式の RCIM については、この調整パラメータは insmod(8)コマンド行で指定する、またはスタートアップ・スクリプト/etc/init.d/rcim 内の modprobe(8)呼び出しが求める modprobe.conf(5)にセットする事も可能です。
次の機能は rcim 調整パラメータで定義されます

- 様々な割込みをトリガーとする方法：立上り/立下りエッジ、High/Low レベル
- 割込み、出力ライン、分配割込みライン間の関連性
- マスターRCIM がある RCIM チェーン内のシステムの名称
- ティック・クロックをローカルで実行する、またはチェーン内のマスターRCIM システムのクロックで同期させるかどうか
- RCIM をマスター・クロックとして登録するかどうか

rcim 調整パラメータはオプションのカンマ区切りリストを受け付けます。
例：
rcim='host/server1.ccur.com, eti1/rising, di3/high, rtc3|di6'

この RCIM スレーブ・システムに関する例では、RCIM マスターのシステム名は server1.ccur.com、エッジ・トリガ割込み(eti) #1 は立上りエッジで始動、分配割込み(di) #3 は High 値で始動、分配割込みライン(di) #6 はリアルタイム・クロック・タイマー(rtc) #3 で駆動されるよう構成されます。
sync/nosync オプションは RCIM チェーン全体のクロック同期に影響します。「sync」はマスターRCIM のローカル・クロックで駆動されたケーブル・クロックを RCIM が使うことを指定します。これが規定値となります。「nosync」は RCIM が自身のローカル・クロックを使う事を指定します。
clock/noclock オプションはシステムのクロックソースとして登録するかどうかを定義します。規定値は「clock」です。一旦 RCIM がクロックソースとして登録されると「未登録」にすることは出来ないことに注意してください。また、RCIM がモジュールとして構成され、かつクロックソースとして登録された場合、モジュールはロックされます(「rmmod rcim」は失敗します)
必要であれば、RCIM III と RCIM II のシステム、RTC/ティック/POSIX の各クロックへのタイミング・ソースはここに構成することが可能です。例えば、
rcim=nosync/r,sync/tp
は、RTC(r)に対してはローカル・クロック、ティック(t)と POSIX(p)の各クロックに対してはケーブル・クロックを選んでいます。sync および nosync に対して指定子が提供されない場合、3 つのクロック全てが有効なオプションに従って設定されます。

- **動的構成**

/proc/driver/rcimN/config(N はゼロから始まる RCIM カードの番号)へ設定用文字列(調整可能な rcim で使用される書式)を書くために echo(1)を使って構成を動的に変更する事が可能です。

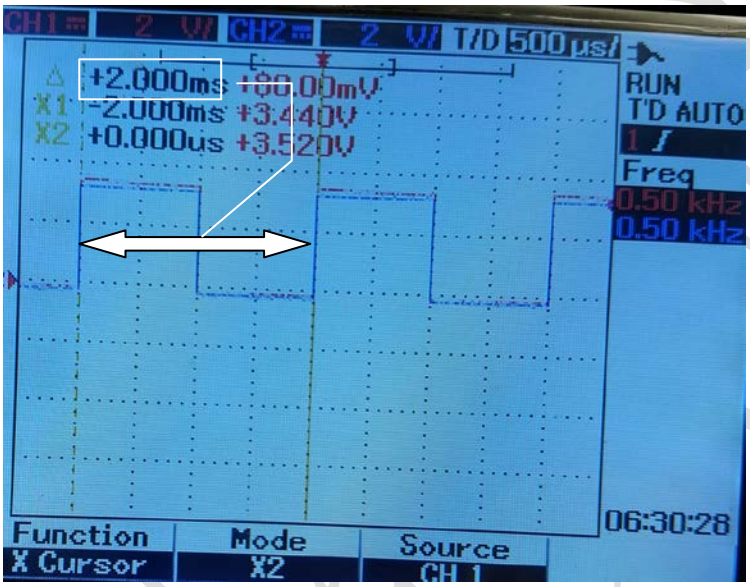
例：
echo eti1/f > /proc/driver/rcim0/config
は、eti#1 を立下りエッジで検知するように変更します。引用符は垂直バーを含む構成要求を囲むために使用する必要があります。例：
echo “rtc0|di1” > /proc/driver/rcim2/config
この方法で行われた構成変更はシステムが再起動された場合は保持されません。これらの変更を行うにはファイルの書込み権限を必要とし RCIM が使用中ではない時にのみ行う必要があります。
デフォルトの構成や各割込みのタイプの選択に関する詳細な情報については、3 章または rcim(4)の man ページを参照してください。

RCIM システムを構成する際は、以下に留意してください：
· RCIM チェーンに関して、マスター内で増加し続けるクロック・シグナルは全てのスレーブにブロードキャストされるため、全てのスレーブ RCIM のティック・クロックおよび POSIX クロックはマスターで同期されます。一旦、全ての RCIM のクロックが最初に同期されると同期されたままとなります。
ティック・クロックを同期するには、全てのシステム間で動作している TCP/IP 接続が必要となります。更に各スレーブ RCIM のホスト名の構成にマスターRCIM を設定する必要があり、各スレーブはブート時に一度 init スクリプトで rcim_clocksync を実行する設定する必要があります。これはアプリケーションが同期用にティック・クロックを使う場合のみ必要となります。
ntp は POSIX クロックを同期するために使用されますが、RCIM III に関してはより優れたメカニズム rcimdate(8)があります。RCIM-III マスターは 1 秒毎に 1 回自身の POSIX クロックを RCIM のケーブルにブロードキャストします(rcimdate はスレーブの POSIX クロックを正確にマスターに一致させるためにこれを利用)。これは ntp よりもいくつもの優位性を持っています(システム間の TCP/IP 接続を必要としない、同期化がより高速、同期化が非常に正確)。
· 割込み(ローカルで動作または RCIM チェーン全体に配信)は各システムで構成された値に従い処理されます。もし設定された規定値とは異なる方法で機能させたい場合、必要な構成オプションを指定する必要があります。
· RCIM チェーンのシステム全体に割込みを分配する場合、全てのシステムが分配割込みラインに対して対応する構成を行う必要があります。
MSI 割込み構成
RCIM III の最新バージョン(リビジョン 9 以上)は MSI (Message Signaled Interrupts)をサポートします。デフォルトで、RCIM のカーネル・ドライバは PCI INTA 割込みの代わりに MSI 割込みを使用するために可能な時にハードウェアを初期化します。MSI 割込みを使用することにより、RCIM III は特有の非共有割込み有する事が保証され、従って更に信頼のある割り込み応答時間を提供します。
RCIM ドライバは前述の rcim=構成オプションとは関係のない rcim.nomsi=1 オプションを持っています。全てのドライバのバージョンがこのオプションを持っています。指定した場合、MSI 機能はそれをサポートする全ての RCIM ボードが無効となります。ボードの MSI 無効の有無を選ぶためのメカニズムはありません。本オプションが指定された場合、RCIM ドライバは PCI INTA 割込み方式の使用に逆戻りします。性能上の理由により、本オプションは MSI 割込みの問題が発生する場合にのみ使用する必要があります。
静的にリンクされた RCIM ドライバについては、この調整は GRUB ブート・ローダのコマンド行(rcim.nomsi=1)で指定することが可能です。モジュール形式の RCIM については、この調整は「options rcim.nomsi=1」として/etc/modprobe.conf に設定する事が可能です。
本オプションは RCIM II または RCIM I のシステムには効果がありません。

マスタースレーブ間の時刻差計測

● [プログラム説明]

RCIM マスタの RTC0 に対して 2ms 毎に割り込みを発生させる設定を登録。
上記割り込みを RCIM/DI0 に分配し、5 mの光ケーブルを介してスレーブ側の RCIM/DI0 に割り込み信号を送信。
マスタおよびスレーブ共に DI0 の割り込みに対応したハンドラを登録し、当該ハンドラ内で /dev/rcim/sclk および clock_gettime()により POSIX クロックをそれぞれ取得して画面に表示。
合わせて、RTC0 を外部ピン OUT0 に出力し、マスター-スレーブ間の OUT0 出力をオシロスコープで観測しました。

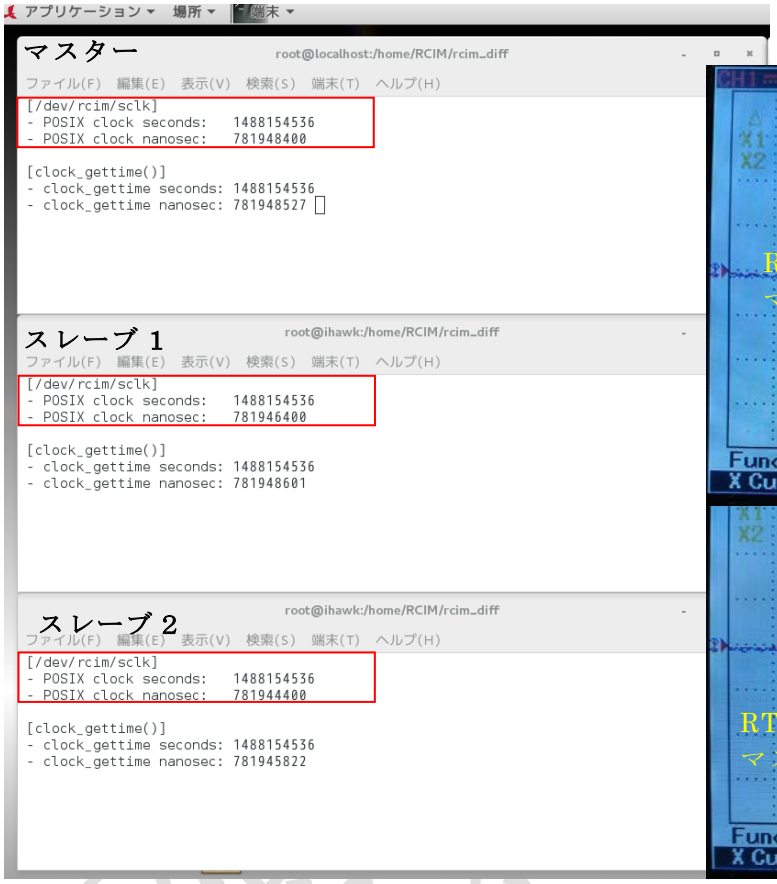


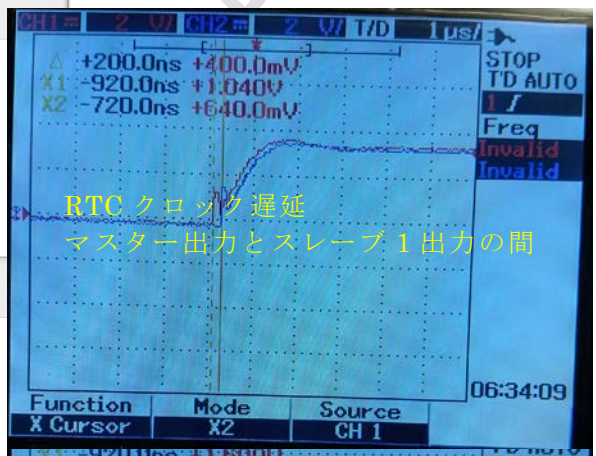

```
if ((fd_rtc = open("/dev/rcim/rtc0", O_RDWR)) < 0)
{
        return (-1);
}
rtc1.r_count  = 2;
/*
This field  holds the initial value the clock counter is to be
set to.
*/
rtc1.r_repeat = 2;
/*
This field specifies the value to be reloaded when the count
reaches zero.  If a value of zero is  requested, a single non-
repeating operation is performed. On RCIM, a repeating counter
must specify the same value for r_count and r_repeat.
*/
rtc1.r_res    = MSEC;
ioctl(fd_rtc, RTCIOCSETL, &rtc1);
ioctl(fd_rtc, RTCIOCSTART);
```

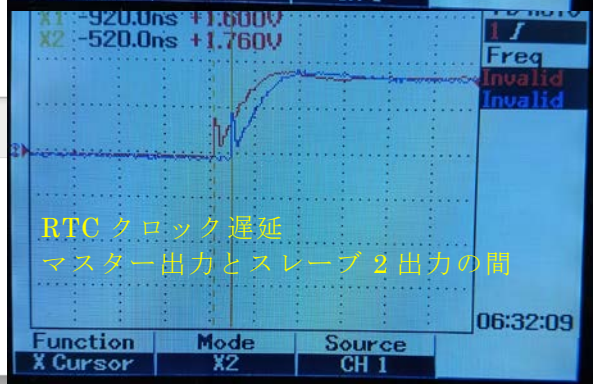
2ms=1ms×2 回

[プログラム起動方法]
マスタ側： ./rcim_diff -m
スレーブ側： ./rcim_diff -s
※ プログラム内で shield コマンドにより CPU1 をシールドし、その CPU に対して当該プログラムをバインドするように実行しています。
※ スレーブ側では起動時に rcimdate コマンドを実行しています。
テストで使用しました計算機のスペックを以下に示します。
マスタ： Xeon E5-2699v4 2.20GHz
スレーブ 1： Xeon E5-2690v4 2.60GHz
スレーブ 2： Xeon E5-1620 3.50GHz

● [結果]







● X 表示

	POSIX CLOCK	差分
マスター	1488154536 781948400	基準
スレーブ 1	1488154536 781946400	-2μ 秒
スレーブ 2	1488154536 781944400	-4μ 秒

	Clock_gettime()	差分 1	差分 2
マスター	1488154536 781948527	基準	-127ns
スレーブ 1	1488154536 781948601	-74ns	-2201ns
スレーブ 2	1488154536 781945822	2705ns	-1422ns

注：X 表示の差分は、RCIM 割込みジッターが原因のため、最大 15μ 秒遅延します。
差分 1 ：clock_gettime()のマスターを基準
差分 2 ：各 POSIX CLOCK を基準

● オシロスコープ
マスタースレーブ 1 間の「遅延」は 200ns
マスタースレーブ 2 間の「遅延」は 400ns

プログラム

注：実行前に RCIM の smp_affinity を専用に割り付ける事。本プログラム実行後、RCIM の割込みラインを変更してしまうため、rcim_clocksync は正常に動作しない。

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/shm.h>
#include <sys/socket.h>
#include <fcntl.h>
#include <netdb.h>
#include <pthread.h>
#include <signal.h>
#include <sched.h>
#include <time.h>
#include <unistd.h>
#include <sys/io.h>
#include <curses.h>
#include <poll.h>
#include <rcim.h>
#include <cpuset.h>
#include <mpadvise.h>

static volatile int    loop_stop_flg = 0;
static volatile unsigned int *sclk_ptr = NULL;
static volatile unsigned int posix_sec = 0;
static volatile unsigned int posix_nsec = 0;
static volatile unsigned int func_sec = 0;
static volatile unsigned int func_nsec = 0;

static int                      fd_sclk = 0;
static int                      fd_rtc  = 0;
static int                      fd_di   = 0;

static void sigint_handler(int sig)
{
        loop_stop_flg = 1;
}

static void sigrtmin_handler(int sig)
{
        struct timespec      now;

        posix_sec  = (volatile unsigned int)*(sclk_ptr + 0x100 / 4);
        posix_nsec = (volatile unsigned int)*(sclk_ptr + 0x108 / 4);

//      clock_gettime(CLOCK_MONOTONIC, &now);
        clock_gettime(CLOCK_REALTIME, &now);

        func_nsec  = (unsigned int)now.tv_nsec;
        func_sec   = (unsigned int)now.tv_sec;
}

static void do_disp(void)
{
        move( 0,   0); printw(" [/dev/rcim/sclk]");
        move( 1,   0); printw(" - POSIX clock seconds:     %9d ", posix_sec);
        move( 2,   0); printw(" - POSIX clock nanosec:     %09d ", posix_nsec);

        move( 4,   0); printw(" [clock_gettime()]");
        move( 5,   0); printw(" - clock_gettime seconds: %9d ", func_sec);
        move( 6,   0); printw(" - clock_gettime nanosec: %09d ", func_nsec);
}

static int rcim_open(int flg)
{
        struct sigaction     sigact;
        struct rtcl                     rtcl;

        if ((fd_sclk = open("/dev/rcim/sclk", O_RDWR)) < 0)
        {
```

```
                return (-1);
        }
        sclk_ptr = mmap(0, 0x1000, PROT_READ, MAP_SHARED, fd_sclk, 0);
        if (sclk_ptr == MAP_FAILED)
        {
                return (-1);
        }

        if (flg > 0)
        {/* 注意：RCIM 割込みは 1ms の表示より 2 倍以上遅くしないと、データの更新が見えない */
                if ((fd_rtc = open("/dev/rcim/rtc0", O_RDWR)) < 0)
                {
                        return (-1);
                }
                rtcl.r_count    = 2;
                /*
                This field    holds the initial value the clock counter is to be
                set to.
                */
                rtcl.r_repeat = 2;
                /*
                This field specifies the value to be reloaded when the count
                reaches zero.    If a value of zero is    requested, a single non-
                repeating operation is performed. On RCIM, a repeating counter
                must specify the same value for r_count and r_repeat.
                */
                rtcl.r_res      = MSEC;
                ioctl(fd_rtc, RTCIOCSETL, &rtcl);
                ioctl(fd_rtc, RTCIOCSTART);
                /*
                For more information about rtcl, see the man page of "rcim_rtc".
                */
        }

        if ((fd_di = open("/dev/rcim/di0", O_RDWR)) < 0)
        {
                return (-1);
        }

        sigemptyset(&sigact.sa_mask);
        sigaddset(&sigact.sa_mask, SIGRTMIN);
        sigact.sa_flags     = SA_RESTART;
        sigact.sa_handler = sigrtmin_handler;
        sigaction(SIGRTMIN, &sigact, NULL);
        ioctl(fd_di, DISTRIB_INTR_ATTACH_SIGNAL, SIGRTMIN);
        ioctl(fd_di, DISTRIB_INTR_ARM);
        ioctl(fd_di, DISTRIB_INTR_ENABLE);
        /*
        For more information about DI, see the man page of "rcim_distrib_intr".
        */

        return 0;
}

static void rcim_close(void)
{
        if (fd_sclk > 0)
        {
                close(fd_sclk);
        }
        if (fd_rtc > 0)
        {
                ioctl(fd_rtc, RTCIOCSTOP);
                close(fd_rtc);
        }
        if (fd_di > 0)
        {
                ioctl(fd_di, DISTRIB_INTR_DISARM);
                ioctl(fd_di, DISTRIB_INTR_DISABLE);
                close(fd_di);
        }
}

void do_main(void)
{
        int                     key, nfds = 1;
```

```
        int                     disp_flg = 1;
        struct pollfd           fdarray[1];
        struct timespec         w1us = {0, 1 * 1000};

        initscr();
        cbreak();
        noecho();
        nonl();
        timeout(1);
        printf("¥x1b[?25l");

        fdarray[0].fd = STDIN_FILENO;
        fdarray[0].events = POLLIN | POLLPRI;

        while (loop_stop_flg == 0)
        {
                if (poll(fdarray, nfds, 1) < 0)
                {
                        nanosleep(&w1us, NULL);
                }
                else
                {
                        key = getch();
                        if (key == 'q' || key == 'Q')
                        {
                                break;
                        }
                }
                do_disp();
                refresh();
        }

        printf("¥x1b[?25h");
        clear();
        refresh();
        echo();
        nl();
        endwin();
}

int main(int argc, char *argv[])
{
        extern char             *optarg;
        int                     c;
        int                     master = 0;
        struct sigaction        sigact;
        struct sched_param sparam;
        cpuset_t* setp;

        while((c = getopt(argc, argv, "ms")) != EOF)
        {
                if (c == 'm')
                {
                        master = 1;
                }
        }

        mlockall(MCL_CURRENT | MCL_FUTURE);

        // Execute some commands
        system("/usr/bin/run -b0 -g0");
        system("/usr/bin/run -b0 -u0");
        system("/usr/bin/shield -a1");
        if (master > 0)
        {
                system("/bin/echo ¥"rtc0|di0¥" > /proc/driver/rcim/config");
                system("/bin/echo ¥"rtc0|out0¥" > /proc/driver/rcim/config");
        }
        else
        {
                system("/usr/sbin/rcimdate -d");
                system("/bin/echo ¥"di0|out0¥" > /proc/driver/rcim/config");
        }

        // Change process priority
        sched_getparam(0, &sparam);
```

```
        sparam.sched_priority = 10;
        sched_setscheduler(0, SCHED_FIFO, (const struct sched_param*)&sparam);

        setp = cpuset_alloc();
        cpuset_set_string(setp, "2");
        mpadvise(MPA_PRC_SETBIAS,MPA_PID,0,setp);
        cpuset_free(setp);

        // Register a signal handler
        sigemptyset(&sigact.sa_mask);
        sigaddset(&sigact.sa_mask, SIGINT);
        sigact.sa_flags     = SA_RESTART;
        sigact.sa_handler = sigint_handler;
        if (sigaction(SIGINT, &sigact, NULL) < 0)
        {
                exit(1);
        }

        // Open the rcim device
        if (rcim_open(master) < 0) goto ERROR;

        do_main();

ERROR:
        rcim_close();
        munlockall();

        exit(0);
}
```