Local APIC Timerを用いて時間計測機能を実装

内園

目次

1. はじめに
2. 事前知識
3. ACPI PM Timerについて
4. Local APIC Timerについて
5. Local APIC Timerの周波数を計測
6. 実際に動かしてみる
7. やってみて
8. 参考文献

1. はじめに

1.1 自己紹介

1.2 概要

1.3 注意点

1.4 構成

1.1 自己紹介

内園潤也

年齢: 23歳

出身: 大阪府茨木市

趣味: ゲーム

1.2 概要

・やったこと詳細

→ ACPI PM Timerを使ってLocal APIC Timerの周波数を計測し、それを利用して1秒間隔で割り込みを発生させる

・勉強した目的

→ OSがどうやって時間計測しているか知りたかった

・リポジトリ

→ https://github.com/junyaU/time-measurement-using-Local-APIC

1.3 注意点

・x86-64環境での動作しか想定していないので他のマシンアーキテクチャでは動きません

・今回の主眼はLocal APIC TimerとACPI PM Timerを用いた時間計測の実装であり、下記の項目を解説すると発表が終わらなくなるので、軽く触れる程度にしておきます

- 割り込みとその設定方法について
- UEFI BIOS
- ACPI PM Timerの制御レジスタの取得方法
- Local APIC Timerの細かい設定

1.4 構成

- qemu-system-x86_64 7.2.0
- C++17
- Clang 15.0.7
- LLD 15.0.7
- EDKⅡ CLANGPDB
- GDB 12.1

2. 事前知識

2.1 周波数

2.2 割り込み

2.3 UEFI BIOS

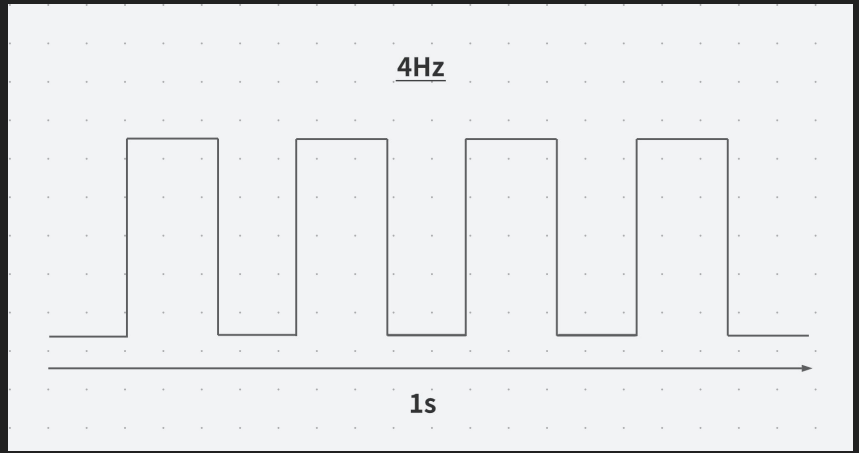
2.1 周波数とは

・1秒あたりに繰り返す信号の数（Hz）

・コンピュータの中ではクロックと呼ばれる電気的な信号が周期的に刻まれている（クロック周波数）

・周波数と信号の数を計測することでかかった秒数を測定することができる

↓信号が4回発生しているので4Hz



2.1 周波数とは

・秒数は 信号数 / 周波数 によって求めることができる

・例えば周波数が4Hz、信号の数が16とすると4秒経過していることがわかる

・つまりタイマーとは信号の数を数える装置であるといえる

・周波数が4Hzの場合0.25秒単位で時間を計算することができる

・周波数が大きくなればなるほど細かい時間を測れるようになり、この測定の細かさを分解能という

→1GHzになると1*10^-9秒ととんでもないぐらい細かくなる

2.2 割り込み

・実行中のプログラムを一時中断させて、実行させたいプログラムを割り込ませて実行させる機構のこと

・例えば、アプリの実行途中にマウスが動かせるのもアプリの実行に割り込んで、マウスを動かすという動作のプログラムが実行されるため

・ゼロ除算などの致命的なエラーに対しても割り込みで例外処理を走らせる

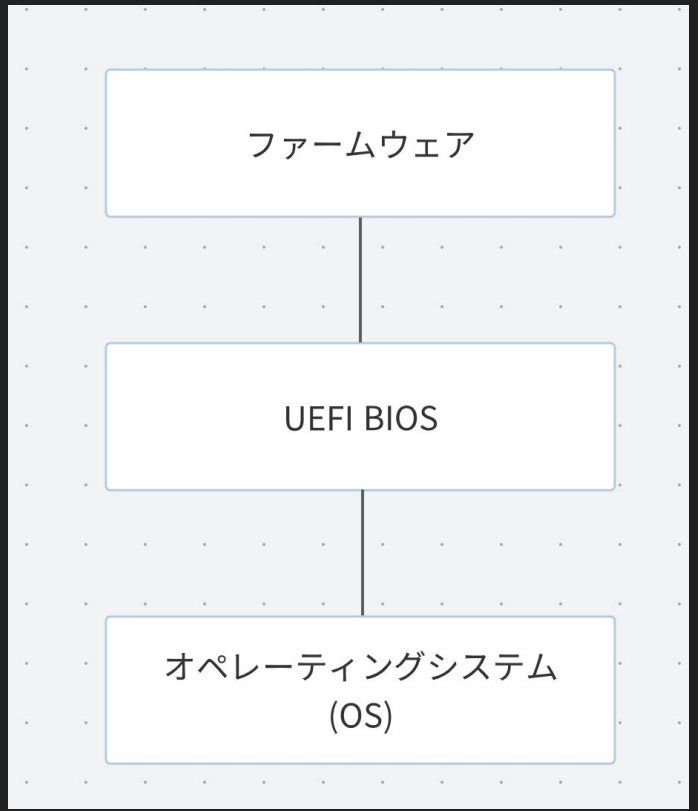
・x86-64環境ではIDTと呼ばれるテーブルに割り込みの設定を登録する

2.3 UEFI BIOS

・PCに入っているファームウェアとの通信仕様を定めたインターフェース

・PC起動後に実行されて、PCの初期化やOSを実行するためのboot loaderの機能を提供してくれる

・UEFIはレガシーBIOSの後継

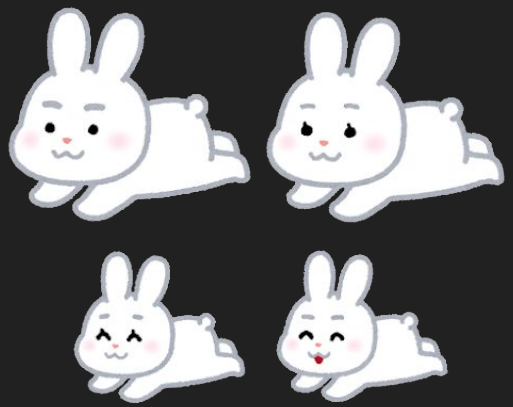


3. Local APIC Timerについて

3.1 Local APIC Timerとは

3.2 使い方について

3.3 周波数



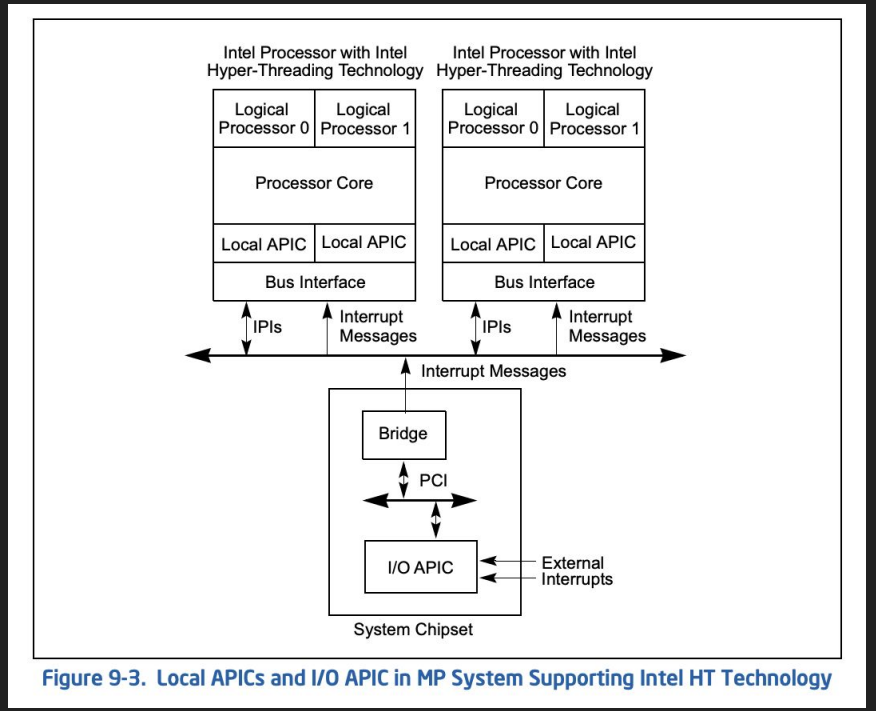


Figure 9-3. Local APICs and I/O APIC in MP System Supporting Intel HT Technology

3.1 Local APIC Timerとは

・Local APIC TimerはCPUのコアそれぞれに内蔵されている、Local APICに実装されているタイマーのこと

・Local APICはマルチプロセッサ環境での割り込みを制御するための処理装置

・APICは Advanced Programmable Interrupt Controllerの略

・CPUのコアに内蔵されているため、低いオーバーヘッドでの割り込みやCPUコア内のレジスタを通じて高速な読み書きなどが可能になる

・OSのスケジューリング機能などに使われる

・今回は1秒経過したら割り込みで文字出力をさせたいのでこれを使う

3.2 使い方について

・これらのレジスタに値を設定することで制御できる（FEE0から始まるアドレスはレジスタ）

・LVT Timer　割りこみや動作モード（単発、周期）の設定を行います

・Initial Count　カウンタの初期値を設定する

・Current Count　カウンタの現在値、Initial Countに設定した値がデクリメントされていき、0になったら割り込みが発生する

・Divide Config　クロック周波数を 1/Nに変換（分周比）することでより長い時間カウントできる

→1秒間にカウントされる数が 1/Nになるため

アドレス	レジスタ名	概要
FEE0 0320H	LVT Timer	割り込み、動作モードの設定を行う
FEE0 0380H	Initial Count	カウンタの初期値
FEE0 0390H	Current Count	カウンタの現在値
FEE0 03E0H	Divide Configuration	カウンタの分周比

3.3 周波数

・Local APIC Timerのクロック周波数が分かれば、1秒を計測して割り込みを発生させることができる

→しかしこのタイマーのクロック周波数はBIOSの設定、CPUの種類やバージョン、システムのモデル…など様々な要因によって決まるので、OS起動時に、　クロック周波数を計測する必要がある

このままでは1秒を測ることができないので、クロック周波数がわかっている別のタイマーを使ってLocal APIC Timerのクロック周波数を計測する

4. ACPI PM Timer

4.1 ACPI PM Timerとは

4.2 使い方について

4.3 時間計測機能を作る

4.1 ACPI PM Timerとは

・ACPI(Advanced Configration and Power Interface)仕様に準拠したタイマー

・ACPI規格はPCの電源の管理などの標準規格

・PMはPower Managementのことで、このタイマーはパソコンのアイドル時間を計測してスリープにするかどうかとかを判断したりする

・このタイマーはクロック周波数は決められているので、これを使ってLocal APIC Timerのクロック周波数を求めることができる

・Local APIC Timerみたいにアドレスが固定されてないので、動的にレジスタのアドレスを取得する必要あり

4.2 使い方について

・このタイマは24or32ビットのカウンタでありクロック周波数は3.579545MHz

→つまり、1秒間にタイマーが3579545回カウントアップされることになる

・ 2^24 * 1/3579545 = 16777216 * 0.000000279 = 4.680843264

→24ビットの場合だと4.7秒ぐらいまでの時間を計測することができる

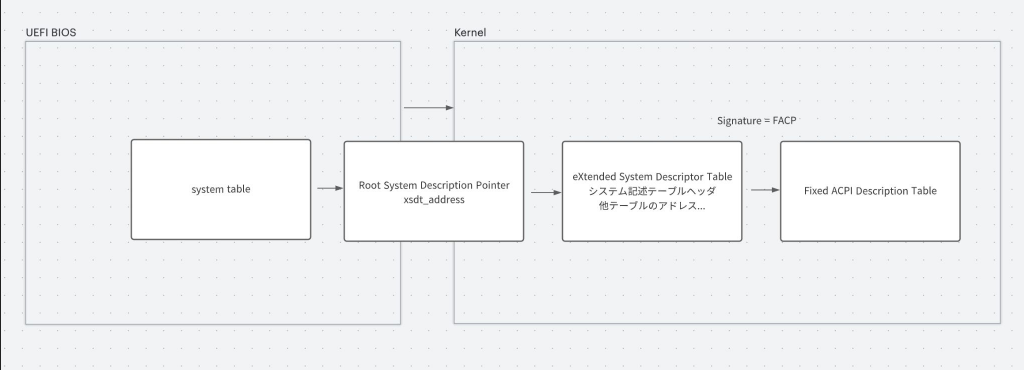
・カウント値はPM_TMR_BLKレジスタの値を読むことで取得可能

→まずはPM_TMR_BLKレジスタのアドレスを取得する

4.2 使い方について

・UEFI BIOSのsystem tableからRSDP取得して、RSDPからXSDTを取得して、XSDTのエントリの中から取得したFADTテーブルの要素にPM_TMR_BLKがある

→ややこしい&長くなるので気になる方はリポジトリを覗いてみてください💦



4.3 時間計測機能を作る（ACPI PM Timer）

① タイマーの現在値の取得とNミリ秒後時点での値を取得

・start_count → タイマーの現在のカウント値を取得している

・end_count → タイマーの終了値。現在のタイマーのカウント値にN * 周波数を足したものがN秒後のタイマーの値になる

④ 現在のタイマーのカウント値がend_countを超えるまで待機し終了する

※③④は本質から逸れるので省略します

これで任意のミリ秒待機することができる

```
void wait(unsigned long milliSec) {
    // ①
    const uint32_t start_count = receiveFromIO(fadt->pm_tmr_blk);
    uint32_t end_count = start_count + kPMTimerFrequency * milliSec / 1000;

    // ②
    const bool pm_timer_32 = (fadt->flags >> 8) & 1;
    if (!pm_timer_32) {
        end_count &= 0x00ffffffu;
    }

    // ③
    bool isOverFlow = end_count < start_count;
    if (isOverFlow) {
        while (receiveFromIO(fadt->pm_tmr_blk) >= start_count)
            ;
    }

    // ④
    while (receiveFromIO(fadt->pm_tmr_blk) < end_count)
        ;
}
```

5. Local APIC Timerの周波数を計測する

5.1 レジスタを準備する

5.2 周波数を測る

5.3 周波数をもとに1秒でタイムアウトするように設定

5. Local APIC Timerの周波数を計測する

・Local APIC Timerを初期化しているコード

・先ほど実装したACPI PM Timerを用いたwaitで周波数の測定を行っている

・ここからこのコードの解説をしていく

```
void initializeLocalAPICTimer() {
    printj("Initializing local APIC timer...\n");

    // 1:1
    divide_conf = 0b1011;
    lvt_timer = (0b001 << 16);

    startTimer();

    acpi::wait(100);
    const auto elapsed_time = kCountMax - current_count;

    stopTimer();

    auto local_apic_timer_frequency = static_cast<uint64_t>(elapsed_time * 10);

    lvt_timer = (0b010 << 16) | InterruptVector::kLocalAPICTimer;
    initial_count = local_apic_timer_frequency;
    printj("Initialized local APIC timer.\n");

    printj("frequency of Local APIC Timer: %llu Hz\n",
           local_apic_timer_frequency);
}
```

5.1 レジスタを準備する

軽くもう一度説明しておくと、

・lvt_timer：動作モードや割り込みの設定

・initial_count：カウンタの初期値を設定する このレジスタに初期値を設定した時点でカウントが始まる

・current_count：現在のカウンタの値が設定されている。initial_countの値をコピーして、その値をデクリメントしていき0になったら割り込みを発生させる

・divide_conf：分周比を設定する。1/Nまでクロック周波数を下げることができるが、今回は 1秒だけなので1:1で十分

```
// register for setting operating mode and interrupts
volatile uint32_t& lvt_timer = *reinterpret_cast<uint32_t*>(0xfee00320);

// register to set the initial value of the counter
volatile uint32_t& initial_count = *reinterpret_cast<uint32_t*>(0xfee00380);

// register in which the current value of the counter is stored
volatile uint32_t& current_count = *reinterpret_cast<uint32_t*>(0xfee00390);
      junya uchizono, 3 days ago • impl local apic timer
// register for divider ratio setting
volatile uint32_t& divide_conf = *reinterpret_cast<uint32_t*>(0xfee003e0);
```

5.1 レジスタを準備する

・divide_conf = 0b1011;

→ 分周比を1:1にしている

→ 1:1なので周波数は変化しません

・lvt_timer = (0b001 << 16);

→ 動作モードを単発、割り込みは無しに設定

→ とりあえず周波数を測るためだけに動かすので単発実行でOK

```
void initializeLocalAPICTimer() {
    printj("Initializing Local APIC timer...\n");

    // 1:1
    divide_conf = 0b1011;
    lvt_timer = (0b001 << 16);

    start();

    acpi::wait(100);
    const auto elapsed_time = kCountMax - current_count;

    stopTimer();


    auto local_apic_timer_frequency = static_cast<uint64_t>(elapsed_time * 10);

    lvt_timer = (0b010 << 16) | InterruptVector::kLocalAPICTimer;
    initial_count = local_apic_timer_frequency;
    printj("Initialized local APIC timer.\n");

    printj("frequency of Local APIC Timer: %llu Hz\n",
           local_apic_timer_frequency);
}
```

5.2 周波数を測る

・startTimer();

→タイマーの計測スタート。initial_countに計測できる最大値（kCountMax）を入れている

・acpi::wait(100);

→ 先ほど実装したwaitで100ミリ秒待機する

・elapsed_time = kCountMax - current_count;

→ 最大値(タイマーに設定した初期値)から現在の値を引くことで、0.1秒の間にカウントした数を取得

・stopTimer(); → タイマーのストップ

```
void initializeLocalAPICTimer() {
    printj("Initializing local APIC timer...\n");

    // 1:1
    divide_conf = 0b1011;
    lvt_time

    startTimer();

    acpi::wait(100);
    const auto elapsed_time = kCountMax - current_count;

    stopTimer();


    auto local_apic_timer_frequency = static_cast<uint64_t>(elapsed_time * 10);

    lvt_timer = (0b010 << 16) | InterruptVector::kLocalAPICTimer;
    initial_count = local_apic_timer_frequency;
    printj("Initialized local APIC timer.\n");

    printj("frequency of Local APIC Timer: %llu Hz\n",
           local_apic_timer_frequency);
}
```

5.3 周波数をもとに1秒でタイムアウトするように設定

・local_apic_timer_frequency = elapsed_time * 10;

→0.1秒の間にカウントした数を10倍することで1秒あたりのカウント数=周波数となる

・lvt_timer = (0b010 << 16) | kLocalAPICTimer;

→タイマーを周期モードに設定し、割り込みが発生した時に文字列が出力されるようにする

・initial_count = local_apic_timer_frequency;

→初期値にこのタイマーのクロック周波数を設定することで、ちょうど1秒で割り込みが発生するようになる

```
void initializeLocalAPICTimer() {
    printj("Initializing local APIC timer...\n");

    // 1:1
    divide_conf = 0b1011;
    lvt_timer = (0b001 << 16);

    startTimer();

    acpi::wait(100);
    const auto elapsed_time = kCountMax - current_count;

    stopT    ();

    auto local_apic_timer_frequency = static_cast<uint64_t>(elapsed_time * 10);

    lvt_timer = (0b010 << 16) | InterruptVector::kLocalAPICTimer;
    initial_count = local_apic_timer_frequency;
    printj("Initialized local APIC timer.\n");

    printj("frequency    l APIC Timer: %llu Hz\n"
           local_apic_timer_frequency);
}
```

6. 実際に動かしてみる

- UEFI BIOS側からフレームバッファを受け取って画面描画の初期化
- 実装した文字の出力機能を初期化
- 割り込み設定の初期化
- ACPI PM Timerの初期化
- Local APIC Timerの初期化
- 省電力モードにして割り込みを待つ

```
extern "C" void main(const FrameBufferConfig& frame_buffer_config,
                     const MemoryMap& memory_map, const acpi::RSDP& rsdp) {
    initializeScreenDrawer(frame_buffer_config);

    drawConsoleScreen();

    initializeConsole();

    printj("Booting the operating system...\n");

    initializeInterruptConfig();

    acpi::initialize(rsdp);

    initializeLocalAPICTimer();

    while (1) {
        __asm__("hlt");
    };
}       You, 先週 • add kernel file
```

6. 実際に動かしてみる

・このマシンではLocal APIC Timerのクロック周波数は1000710000Hzであることがわかる

→ 1秒間に1000710000回カウントしている

・タイマーが1秒ごとにタイムアウトして、割り込みが発生して文字列が出力されているから計測成功

※厳密にはピッタリ1秒ではないけど

7. やってみて

・時間計測の仕組みをほんのちょっとだけ理解した気になれました！

・準備に60時間ぐらいかかったので普通にしんどかった！

・低レイヤー面白い！！

・次はメモリ管理をもっと勉強したい

8. 参考文献

・Intel® 64 and IA-32 Architectures Software Developer’s Manual

・Noam Nisan コンピュータシステムの理論と実装 ―モダンなコンピュータの作り方 オライリージャパン 2015

・内田 公太 ゼロからのOS自作入門 マイナビ出版 2021

・内田 公太 自作エミュレータで学ぶx86アーキテクチャ マイナビ出版 2015

・Hello uchan world, Local APICタイマー入門（PDF）

・Microsoft ACPI system description tables

・EDKⅡ UEFI Driver Writer’s Guide

・https://github.com/osdev-jp/osdev-jp.github.io/wiki

・https://wiki.osdev.org/ACPI_Timer

・https://wiki.osdev.org/APIC