High Precision Event Timer

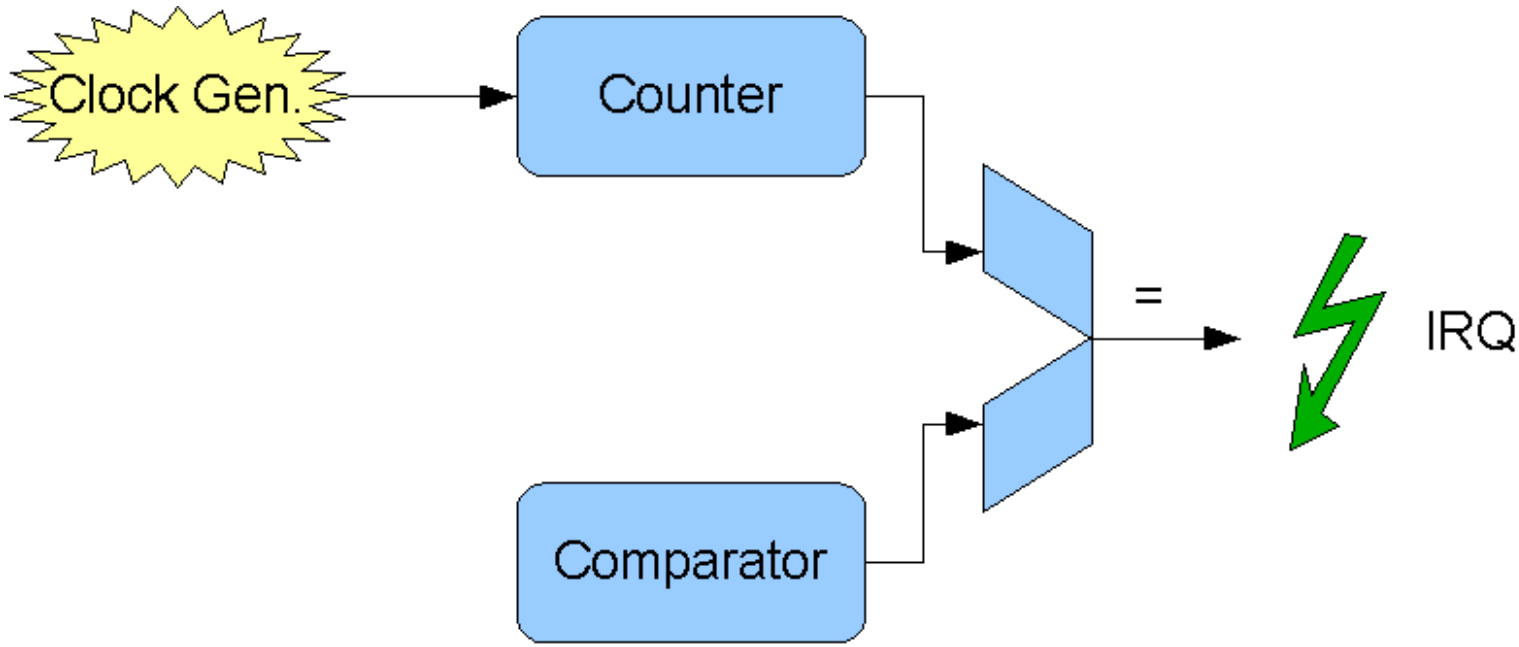
what it is (and what it can do for you)

Andrea Bastoni

Real-Time Lunch

What is a timer?

- Almost all PC timer devices have a similar block diagram



Time Hardware Devices /1

x86 Architecture supports *several* time hardware devices:

- *Real Time Clock (RTC)*: battery-energized clock that keeps the current date and time even when the computer is switched off (e.g., Motorola 146818 chip in PC's)
- *Time Stamp Counter (TSC)*: monotonically increasing counter, generally coupled with the memory bus clock signal (PowerPC has a similar counter: Time Base Register TB)
- *Programmable Interrupt Timer (PIT)*: device that generates periodic or one-shot interrupts (e.g., 8254 chip in PC's)

Time Hardware Devices /2

- *CPU Local Timer (LOC)*: circuit integrated in the CPU that raises local interrupts (e.g., APIC Local Timer in IA-32 CPU's, or Decrementer in PowerPC CPU's)
- *ACPI Power Management Timer (PMT)* (a.k.a. chipset timer): monotone counter included in all ACPI-compliant computers
- *High Precision Event Timer (HPET)*: device providing a monotonically increasing 64-bit counter and *several* timers

Time Hardware Devices /3

Dev.	Clock Src	Freq. range	IRQ	One Shot	Resol.	Accur.	
RTC	Own	2 - 8192 Hz	Yes	No	Low	Low	(1)
TSC	Bus	CPU freq.	No	No	High	High; Low	(2)
PIT	Own	$\approx$ 1.2 kHz	Yes	Yes	Low	Good	
LOC	Bus	BusFreq / 16	Yes	Yes	High	High; Low	(2)
PMT	Own	3.58 MHz	Yes	No	Good	High	(3)
HPET	Own	$\geq$ 10 MHz	Yes	Yes	High	High	

(1) generally not used after booting phase

(2) affected by CPU frequency / voltage scaling

(3) only generates *overflow* interrupts

TSC and LOC are located *on* each CPU (or on each core), the others on board

Some questions...

- Why are people still using RTC?
  - It is well understood and it is easy to program (OSes generally do not use it, so userspace is free to program it)
- PIT is *old*, is it still used?
  - LOC (LAPIC) is commonly used instead of PIT on recent PCs. However it is often used by kernels at boot time.
- If TSC cannot generate interrupts and is affected by scaling, what is it good for?
  - TSC has a low access time, offers a 64-bit counter, is easily readable from userspace. If frequency scaling is disabled, it is accurate (on UMA; NUMA machines need additional considerations)

Why you might want to use HPET

- Soft real-time processing of video / audio streams (“old” HPET name was *multimedia timer*)
  - synchronization of streams can exploit the 64-bit HPET up-counter
- Use different timers for various scheduling purposes
  - Tick based scheduling through periodic interrupt generation (cyclic schedulers. . .)
  - Program different one shot timers to be informed of different deadlines expiration
- Use the counter as a time-base reference (frequency scaling insensitive) on NUMA / Multiprocessor systems
- Substitute legacy specific-purpose timer hardware devices
- . . .

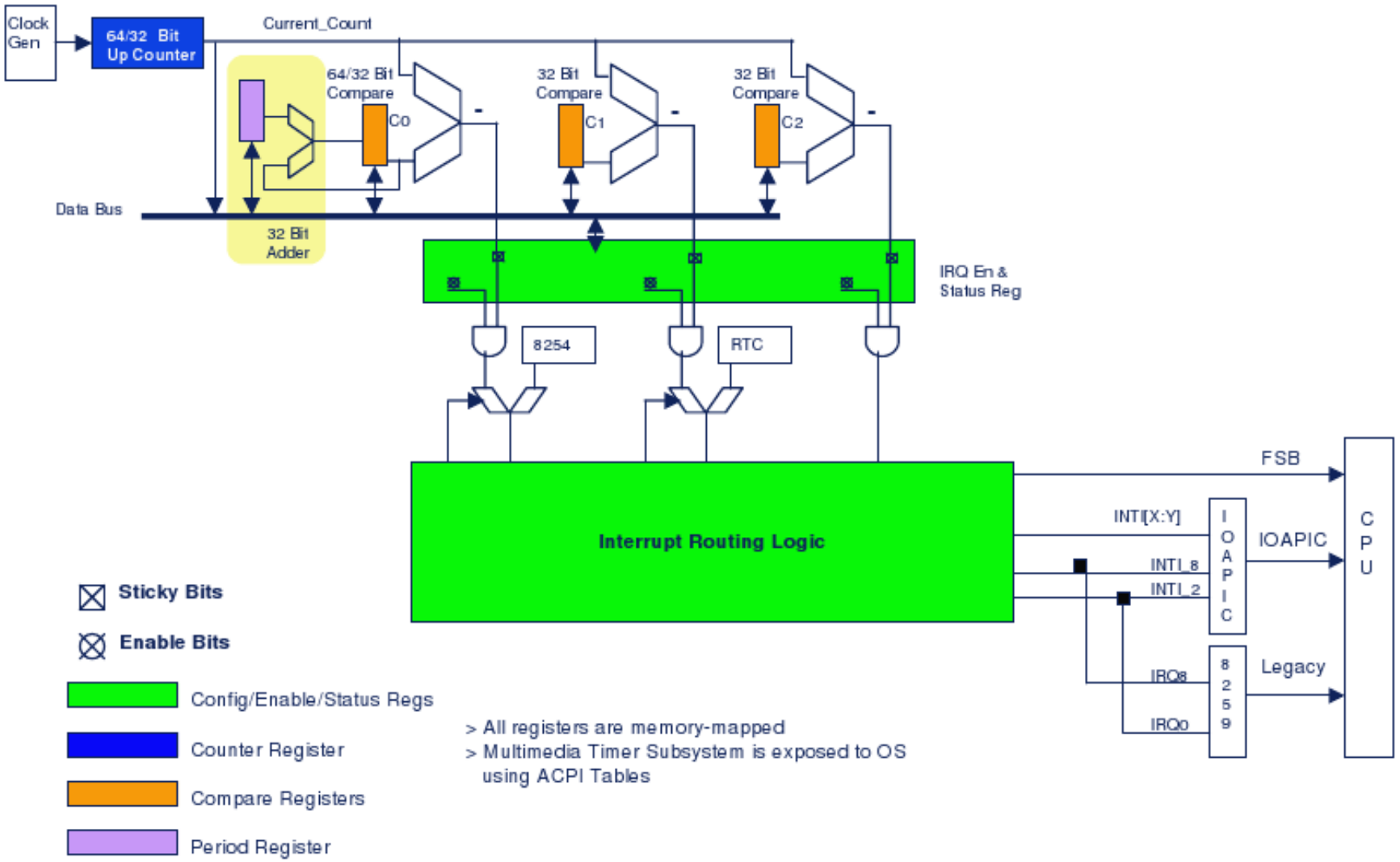
High Precision Event Timer /1

- The HPET architecture defines a set of timers that can be used by the OS
- Each timer can be configured to generate a separate interrupt
  - timers can be configured to generate *one-shot* or *periodic* interrupts
- Timers are implemented through *one monotonic up-counter* and a *set of comparators*
- Intel specification allows up to 32 comparators per timer block, with support for 8 blocks maximum (256 timers)
  - Current implementations only have a small subset of these timers

- HPET will gradually replace both PIT and RTC

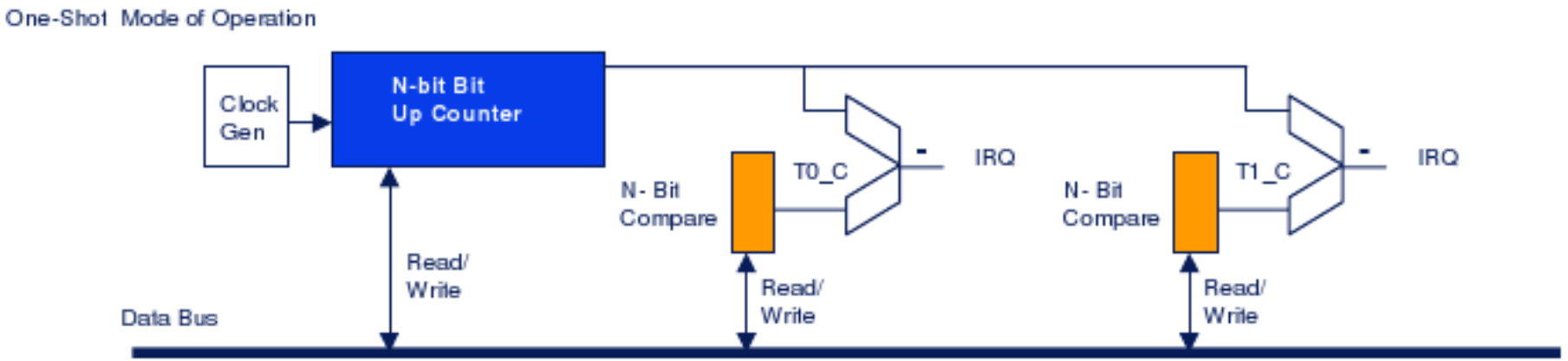
High Precision Event Timer /2

Block diagram:

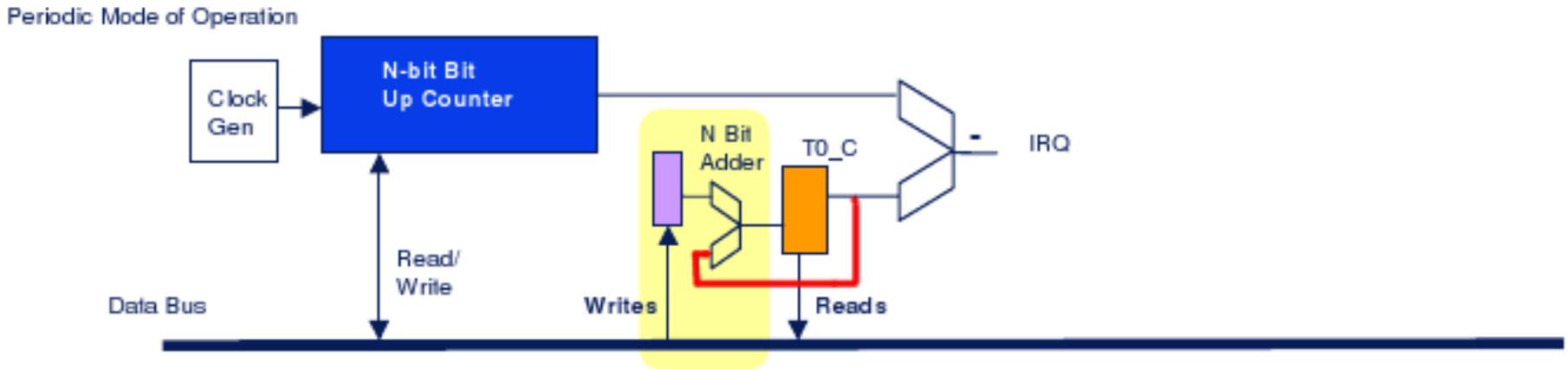


High Precision Event Timer /3

One-shot timer mode:



Periodic timer mode:



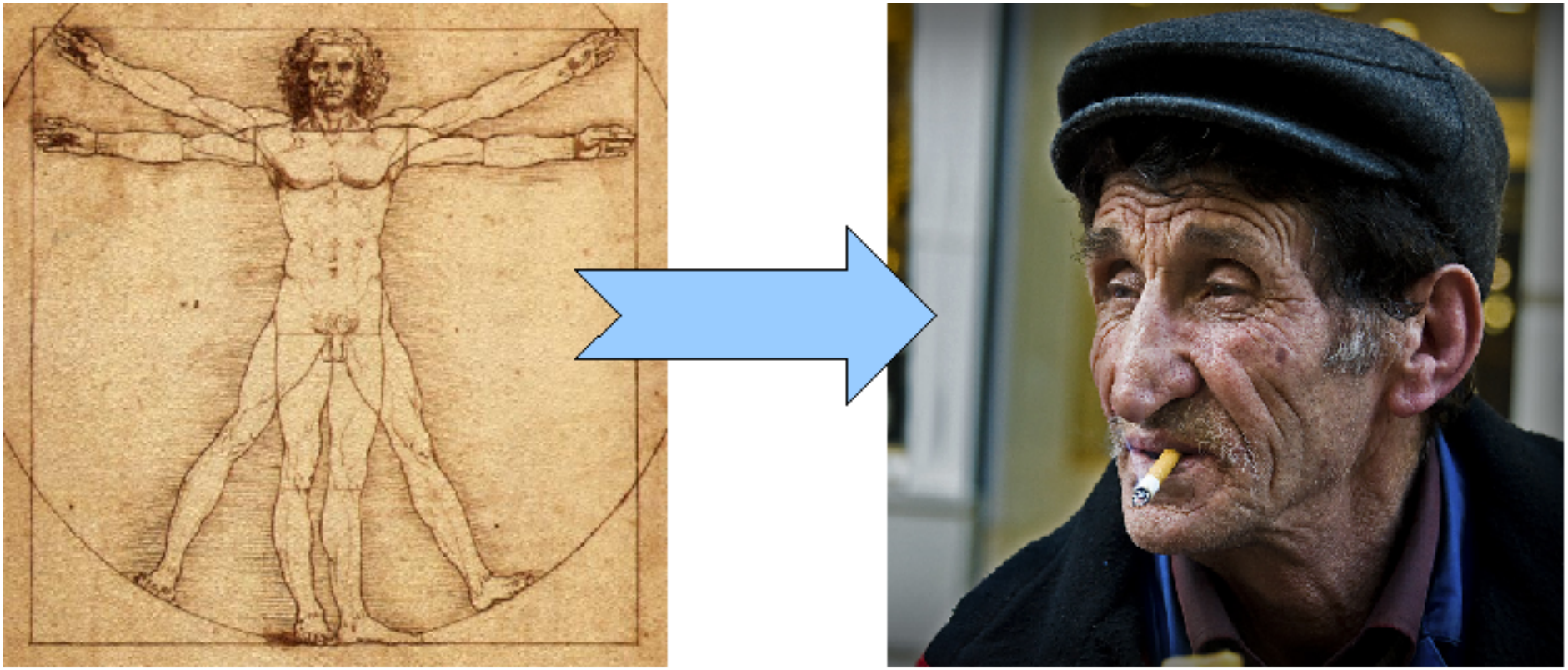
High Precision Event Timer /4

Minimal recommendations for HPET devices:

- Main counter *must* be a 64-bit up-counter
- Minimal clock frequency: *10 MHz*, typical frequencies are around 14 MHz (14.318180 MHz in most implementations)
  - main counter will wrap in $\approx$ 40853 years
- Clock drift:
  - 500 ppm (0.05%) for intervals greater than 1 *ms*
  - 2000 ppm (0.2%) for intervals less than 100 μs
- At least 3 comparators (32 bits width minimum)
  - a 32 bit comparator can only “count” up to $\approx$ 5 minutes
- 1 of 3 must be a periodic capable timer, all 3 timers must be one-shot capable

High Precision Event Timer

From specification to real implementations:



HPET real implementations /1

- Information on commonly available implementations of HPET can be found (for example) in Intel I/O Controller Hub specification (*ICHx*)
- Up to ICH9, HPET implementations provide:
  - one 64 bits up-counter @ 14.318180 MHz
  - one 64 bits comparator (periodic interrupts capable, can work in 32 bits mode)
  - three 32 bits comparators (one shot interrupts only)
- From ICH10 (Corporate Chipset version):
  - seven 32 bits comparators (one shot interrupts only)
  - 8 comparators in total

Reading HPET registers can be tricky:

- a read takes $1 - 2\mu s$
- can we read 64 bits at once?

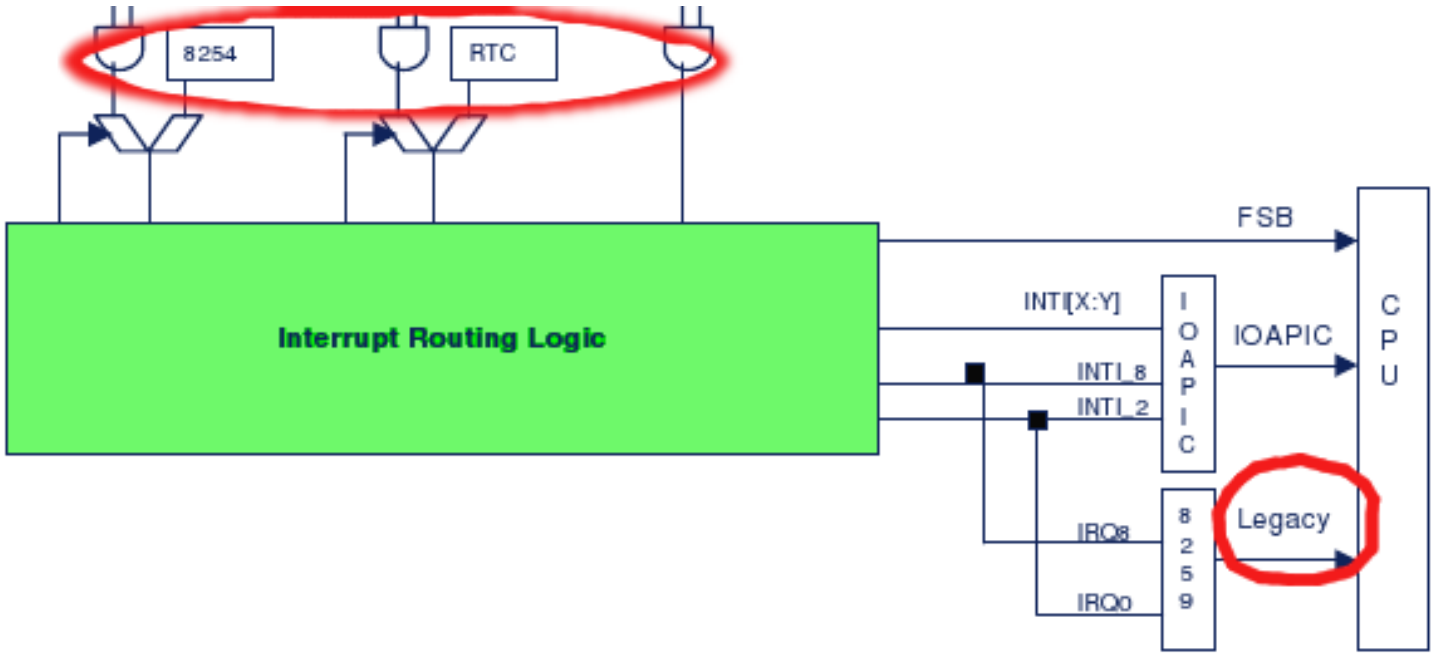
HPET real implementations /2

HPET implementations seem to match specification quite well!

. . . but:

- if HPET is available on your hardware, the OS is willing to use it
- unfortunately OS starts before our programs!

Linux typically wants to use *two* timers:



HPET real implementations /3

And it will take the best (and easier to manage) timers!

- Linux uses timer0 (the periodic interrupts capable timer) as a periodic clockevent (for example, for scheduling tick)
- Linux uses timer1 to emulate RTC device
  - Next expire time is set by software (timer1 is one-shot only)

Issues:

- Timer0 is the only timer which can fire periodic interrupts
- Timer1 rises interrupts on same RTC IRQ line

HPET real implementations /4

Userspace can only use the remaining timers:

- most of the times only 1,2 timers left (timer2, timer3)
- 32 bits timers
- we must share the interrupt line with other devices
- even if we have more timers (ICH10), we do not have enough interrupt lines (timers 4,5,6,7 deliver interrupts through FSB — MSI interrupt)

If we want to make use of timer0 and timer1:

- we need to disable kernel HPET support (it is subtle in x86-64)
- we need to patch the kernel (a new driver is needed; HPET memory addresses are allocated by BIOS. . .)

Userspace Linux support to HPET

Linux offers a two level driver to manage the HPET:

- a low level kernel driver:
  - used by the kernel to program the HPET
  - it cannot be used by userspace, but it is used by the high level driver (the design is not clean, low level kernel driver needs to call high level driver to proper initialize HPET)

- a high level driver which exposes to the userspace a character device (`/dev/hpet`) that can be used to control the HPET
  - supports normal file operations (open, close, read, *ioctl*)

Controlling HPET from userspace

Typical initialization code for the HPET:

```
1  int hpet_init(const char* hpet_path, unsigned int freq)
   {
3          int hpet_fd;

5          if((hpet_fd = open(hpet_path, O_RDONLY)) < 0) {
                   perror(hpet_path);
7                  return errno;
           }
9
           if (ioctl(hpet_fd, HPET_IRQFREQ, freq) < 0) {
11                 perror("Cannot set periodic IRQ");
                   goto out_err;
13         }

15         if (ioctl(hpet_fd, HPET_IE_ON, 0) < 0) {
                   perror("Enable periodic interrupts ioctl");
17                 goto out_err;
           }
19
           return hpet_fd;
21
   out_err:
23         close(hpet_fd);
           return -1;
25 }
```

Questions