Computer Labs: The PC's Real Time Clock (RTC)
## 2º MIEIC

Pedro F. Souto (`pfs@fe.up.pt`)

October 19, 2011

The Real Time Clock (RTC)

- ▶ Integrated circuit that maintains:
  - ▶ The date and
  - ▶ The time of the day

  even when the PC is switched-off and unplugged

- ▶ In addition, it:
  - ▶ Includes alarm functionality and can generate interrupts at specified times of the day;
  - ▶ Can generate interrupts periodically
  - ▶ Includes at least 50 non-volatile one-byte registers, which are usually used by the BIOS to store PC's configuration

- ▶ Modern RTCs are self-contained subsystems, including:
  - ▶ A micro lithium battery that ensures over 10 years of operation in the absence of power (when the power is on, the RTC draws its power from the external power supply)
  - ▶ A quartz oscillator and support circuitry

The RTC's Internal Address Space
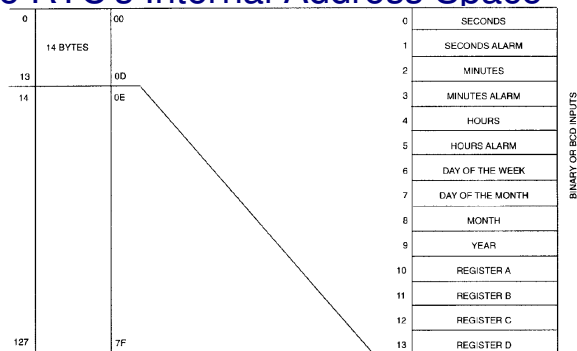
- ▶ ... is an array of at least 64 one-byte registers, whose content is non-volatile. Each register can be:
  - ▶ Addressed individually
  - ▶ Both read and written
- ▶ The first 10 registers are reserved for time-related functionality
- ▶ The following 4 registers are reserved for control of the RTC
- ▶ The remaining registers can be used for arbitrary purposes

Access to the RTC in the PC (1/2)

- The PC uses two ports to access the RTC's internal registers:

  RTC_ADDR_REG on port **0x70**, which must be loaded with the address of the RTC register to be accessed

  RTC_DATA_REG on port **0x71**, which is used to transfer the data to/from the RTC's accessed

- To read/write a register of the RTC requires always:
  1. writing the address of the register to the RTC_ADDR_REG
  2. reading/writing one byte from/to the RTC_DATA_REG

Access to the RTC in the PC (2/2)

Issue What if other code runs between the writing of
RTC_ADDR_REG and the access to the RTC_DATA_REG?

  ▶ And that code modifies the RTC_ADDR_REG?

Solution Disable interrupts **on the processor**. It prevents:

  ▶ IH code from running
  ▶ Other programs (processes) from running.

(Multi-Tasking)

- Modern OSs support the concurrent execution of multiple processes
- ...

Time of the Day, Alarm and Date Registers

ADDRESS LOCATION	FUNCTION	DECIMAL RANGE	DATA MODE RANGE	
			BINARY	BCD
0	Seconds	0–59	00–3B	00–59
1	Seconds Alarm	0–59	00–3B	00–59
2	Minutes	0–59	00–3B	00–59
3	Minutes Alarm	0–59	00–3B	00–59
4	Hours, 12-hour Mode	1–12	01–0C AM, 81–8C PM	01–12AM, 81–92PM
	Hours, 24-hour Mode	0–23	00–17	00–23
5	Hours Alarm, 12-hour	1–12	01–0C AM, 81–8C PM	01–12AM, 81–92PM
	Hours Alarm, 24-hourr	0–23	00–17	00–23
6	Day of the Week Sunday = 1	1–7	01–07	01–07
7	Date of the Month	1–31	01–1F	01–31
8	Month	1–12	01–0C	01–12
9	Year	0–99	00–63	00–99

- It is possible to program whether the data format is binary or BCD, but this applies to all registers
- It is also possible to program whether the hours range from 0 to 23 or 1 to 12 (plus AM and PM), but this applies both to the time and the alarm registers

Reading the Date or the Time of the Day (1/2)

Issue The registers with the date and the time of the day are updated **asynchronously** by the RTC every second

- ▶ These registers are just an image of non-accessible counters that are updated automatically as determined by the signal generated by the (internal) quartz oscilator

Problem What if there is an update while we are reading the time/date?

Question How big can the error be?

- ▶ Does it matter the order in which registers are read?

Reading the Date or the Time of the Day (2/2)

Solution The RTC offers 3 mechanisms to overcome this issue:

Update in progress flag (UIP) of the RTC

- ▶ The RTC sets the `UIP` of `REGISTER_A` 244 μs before starting the update and resets it once the update is done

Update-ended interrupt of the RTC

- ▶ If enabled, the RTC will interrupt at the end of the **update cycle**, the next cycle will occur at least 999 ms later
- ▶ `Register_C` should be read in the IH, to clear the IRQF

Periodic interrupt of the RTC

- ▶ Periodic interrupts are generated in such a way that updates occur sensibly in the middle of the period (actually, 244μs after)
  - ▶ For periodic interrupts whose period is long enough
  - ▶ Thus, after a periodic interrupt occurs, there are at least $P/2 + 244\mu$ seconds before the next update

Updating the Date or the Time of the Day

Problem Asynchronous updates can also make time/date updates inconsistent

Solution Set the SET bit of Register_B before updating

- ▶ It prevents the RTC from updating the time/date registers with the values of the date/time keeping counters
- ▶ At the end of the update the SET bit should be reset so that the RTC updates the counters with the values of the registers

Question Can we use the SET bit of REGISTER_B also for reading the date/time registers?

Alarm Registers

- ▶ The alarm registers allow to configure an alarm
- ▶ When the time of day registers match the corresponding alarm registers, the RTC alarm generates an alarm interrupt, if that interrupt is enabled at the RTC
    - ▶ Bit `AIE` (5) of `REGISTER_B`
- ▶ The RTC supports **don't care** values – values with the 2 MSB set – for alarm registers
    - ▶ These values match any value of the corresponding register of the time of day register set
    - ▶ This makes it possible to configure alarms for multiple times of the day, without changing the contents of the alarm registers
        - ▶ For example, if all 3 alarm registers are set to "don't care", then the RTC will generate an alarm every second

Interrupts

- ► The RTC can generate on 3 different events
  Alarm interrupts (AI)
  Update interrupts (UI)
  Periodic interrupts (PI) with a period between 122 μs and
    0.5 s, as determined by bits RS0-RS3 in REGISTER_A
- ► Each of the interrupts can be enabled/disabled individually,
  using bits AIE, UIE and PIE of REGISTER_B
- ► The RTC has only one IRQ line, which is connected to line
  IRQ0 of PIC2, i.e. IRQ8.
  - ► The source of the interrupt can be determined by checking
    the flags AF, UF and PF of REGISTER_C
  - ► Note that more than one of these flags may be set
    simultaneously
  - ► REGISTER_C must be read to clear these flags, even if
    there is only one enabled interrupt
- ► Flags AF, UF and PF of REGISTER_C are activated even if
  the corresponding interrupts are disabled
  - ► It is possible to use polling to check for the corresponding
    events

Control Registers

REGISTER_A

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
UIP	DV2	DV1	DV0	RS3	RS2	RS1	RS0

UIP If set to 1, update in progress. Do not access time/date
registers

  - ▶ More precisely, this bit is set to 1, $244\mu s$ before an
    update and reset immediately afterwards

DV2-DV0 Control the couting chain (not relevant)

RS3-RS0 Rate selector – for periodic interrupts and square
wave output

Control Registers

REGISTER_B

BIT 7	**BIT 6**	**BIT 5**	**BIT 4**	**BIT 3**	**BIT 2**	**BIT 1**	**BIT 0**
SET	PIE	AIE	UIE	SQWE	DM	24/12	DSE

SET Set to 1 to inhibit updates of time/date registers.

PIE, AIE, UIE Set to 1 to enable the corresponding interrupt source

SQWE Set to 1 to enable square-wave generation

DM Set to 1 to set time, alarm and date registers in binary. Set to 0, for BCD.

24/12 Set to 1 to set hours range from 0 to 23, and to 0 to range from 1 to 12

DSE Set to 1 to enable Daylight Savings Time, and to 0 to disable

  ▶ Useless: supports only old US DST ...

IMPORTANT Do not change **DM**, **24/12** or **DSE**, because it may interfere with the OS

  ▶ In any case, changes to **DM** or **24/12** require setting the registers affected by those changes

Control Registers

REGISTER_C

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
IRQF	PF	AF	UF	0	0	0	0

IRQF IRQ line active

PF Priority interrupt pending

AF Alarm interrupt pending

UE Update interrupt pending

REGISTER_D

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
VRT	0	0	0	0	0	0	0

VRT Valid RAM/time – set to 0 when the internal lithium battery runs out of energy – RTC readings are questionable

Example Code: Waiting for Valid Time/Date

```
void wait_valid_rtc(void) {
    int enabled;
    unsigned long regA = 0;

    do {
        enabled= disable(); // globally disable interrupts
        sys_outb(RTC_ADDR_REG, RTC_REG_A);
        sys_inb(RTC_DATA_REG, &regA);
        if( enabled )
            enable();
    } while ( regA & RTC_UIP);
}
```

- Assuming that functions `enable()` (`disable()`) enable (disable) processor interrupts
- May not be what you want!!!
  - What if code is preempted or interrupted?

Example Code: RTC IH in C

```c
void rtc_isr(void) {

    int cause;
    unsigned long regA;

    sys_outb(RTC_ADDR_REG, RTC_REG_C);
    cause = sys_inb(RTC_DATA_REG, &regA);

    if( cause & RTC_UF )
        handle_update_int();

    if( cause & RTC_AF )
        handle_alarm_int();

    if( cause & RTC_PF )
        handle_periodic_int();

}
```

Question Should we disable interrupts while reading
  REGISTER_C?

Minix 3 Notes

- In Minix 3, like in most OSs, each device is controlled by at most one device driver
- Minix 3 by default does not have any driver for the RTC
  - Need not worry about interference from such a driver
  - The issue raised in the last slide is void
- However, the possibility of the process being preempted or interrupted is not.

Minix 3 Notes: I/O In Assembly

Problem How can assembly code execute I/O operations?

- ▶ Minix 3 device drivers, and your programs, execute at user-level.

Solution Two possible solutions:

1. Use `sys_inX()`/`sys_outX()` kernel calls
2. Use the I/O privilege field in the EFLAGS register, via the `sys_iopenable()` kernel call

`sys_iopenable()`

> "Enable the CPU's I/O privilege level bits for the given
> process, so that it is allowed to directly perform I/O in
> user space."

I/O privilege level (IOPL) field (2 bits) in the EFLAGS register

- ▶ Specifies the privilege level of a process, so that it can
  perform the following operations
    - ▶ `IN/OUT`
    - ▶ `CLI` (disable interrupts)
    - ▶ `STI` (enable interrupts)

Note `sys_iopenable()` is a blunt mechanism

- ► The process is granted the permission to perform I/O on any I/O port
- ► With `sys_inX()`/`sys_outX()` the I/O operations are executed by the (micro)kernel and it is possible to grant permission to only a few selected I/O ports (as determined by `/etc/system.conf`)

Minix 3 Notes: `driver_receive()` is not Polling

`driver_receive()` is a blocking call. If ther process's "IPC queue" is empty:

- The OS will move it to the WAIT state
- The state will be changed to READY, only when a message (or notification) is sent to the process

```
5: while( 1 ) { /* You may want to use a different condition
6:     /* Get a request message. */
7:     if ( driver_receive(ANY, &msg, &ipc_status) != 0 ) {
8:         printf("driver_receive failed with: %d", r);
9:         continue;
10:    }
11:    if (is_ipc_notify(ipc_status)) { /* received notificat
12:        switch (_ENDPOINT_P(msg.m_source)) {
13:        case HARDWARE: /* hardware interrupt notification
14:            if (msg.NOTIFY_ARG & irq_set) { /* subscribed
15:                ... /* process it */
16:            }
17:            break;
18:        default:
19:            break; /* no other notifications expected: do
20:        }
```

Further Reading

- Folha de características dum RTC recente