Seamless Power Management on IoT Devices — Lessons from an HVAC Use Case using RIOT

Jürgen Fitschen - SSV Software Systems GmbH

Slides available at:
https://ssv-embedded.github.io/RIOTSummit2020/
Who am I?

Jürgen Fitschen (jue89 on GitHub)
Systems Engineer at SSV Software Systems, Germany
Using RIOT since 2018

What we'll cover ...

Why does good power management matter?
How does it work?
It's all about timers!
Why does Good Power Management *Matter* to us?
Retrofit Systems are our Passion.

- Goal: enhance efficiency and value of existing systems and environments
- Sensors and actuators must be deployed within already existing systems
- Some retrofit systems require 100+ sensors

➡️ Battery-powered sensors and actuators are required!
Example Retrofit Setup: Battery-powered Sensors

- Task: Send notifications when the windows should be opened
- Measure CO$_2$ concentration in every room
- Send the sensor reading to gateway
- Notify occupier upon high sensor readings via e-mail
The CO₂ Sensor's Application Sequence

<table>
<thead>
<tr>
<th>Phase</th>
<th>Duration</th>
<th>Current</th>
<th>Charge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Measure</td>
<td>100 ms</td>
<td>1,000 μA</td>
<td>0.1 mC</td>
</tr>
<tr>
<td>TX</td>
<td>50 ms</td>
<td>13,500 μA</td>
<td>0.7 mC</td>
</tr>
<tr>
<td>RX</td>
<td>200 ms</td>
<td>8,200 μA</td>
<td>1.6 mC</td>
</tr>
<tr>
<td>Sleep</td>
<td>299,650 ms</td>
<td>10 μA</td>
<td>3.0 mC</td>
</tr>
<tr>
<td>Sum:</td>
<td>300,000 ms</td>
<td></td>
<td>5.4 mC</td>
</tr>
</tbody>
</table>

Battery Charge: [1] 2,400 mAh = 8,640,000 mC
Battery Self-discharge Current: [2] 1.9 μA
Number of Cycles: 1,444,454
Livetime: 13.74 years

→ Reduce power consumption during sleep phase!
How does Power Management Work?
The Internals of the Microcontroller \textit{SAM R30}[3]

Set the \textit{SLEEPCFG} register to "STANDBY" and the \textit{RF Network Interface} to "SLEEP" during sleep phase!
RIOT has a Driver for Power Management

<table>
<thead>
<tr>
<th>Power Mode</th>
<th>Blocker</th>
<th>Lowest Mode?</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDLE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STANDBY</td>
<td>$\text{pm_unblock(STANDBY)}$</td>
<td>1</td>
</tr>
<tr>
<td>BACKUP</td>
<td>$\text{pm_unblock(BACKUP)}$</td>
<td>1</td>
</tr>
</tbody>
</table>

- $\text{pm\_layered}$ keeps track of which power mode can be entered
- The idle thread enters the lowest mode
- Someone must tell $\text{pm\_layered}$ which modes are allowed

⇒ For a seamless user experience, drivers must interact with $\text{pm\_layered}$
It's all about *Timers*!
RIOT's Current Default Timer System: xtimer

```
#include "xtimer.h"
#include "timex.h"

static void callback (void * arg) {
  puts((char*) arg);
}

int main (void) {
  /* 1. Run a callback after 3s */
  static xtimer_t cb_timer = {.callback = callback, .arg = "Hello World"};
  xtimer_set(&cb_timer, 3 * US_PER_SEC);

  /* 2. Sleep the current thread for 60s */
  xtimer_sleep(60);
}
```

- **xtimer** requires the High Speed Timer to run all the time
- **STANDBY** mode must not be entered at any time
There's an Alternative for the Rescue: ztimer

```c
#include "ztimer.h"
#include "timex.h"

static void callback (void * arg) {
    puts((char*) arg);
}

int main (void) {
    /* 1. Run a callback after 3s */
    static ztimer_t cb_timer = {.callback = callback, .arg = "Hello World"};
    ztimer_set(ZTIMER_USEC, &cb_timer, 3 * US_PER_SEC);

    /* 2. Sleep the current thread for 60s */
    ztimer_sleep(ZTIMER_MSEC, 60 * MS_PER_SEC);
}
```

App's Makefile:

```makefile
USEMODULE += ztimer ztimer_usec ztimer_msec ztimer_periph_rtt
USEMODULE += pm_layered
```

samr30-based-board/include/board.h:

```c
#define CONFIG_ZTIMER_USEC_REQUIRED_PM_MODE PM_SLEEPCFG_SLEPMMODE
#define PM_BLOCKER_INITIAL 0x0001
```

⇒ ztimer unblocks STANDBY mode if no ztimer_t requires ZTIMER_USEC to run
xtimer & ztimer can be Friends and Coexist!

USEMODULE += ztimer ztimer_usec xtimer xtimer_on_ztimer evtimer

**High Speed Timer**

- ZTIMER_USEC
  - ztimer_t
  - ztimer_t
  - ztimer_t

**xtimer**

- xtimer_t
  - xtimer_t
  - xtimer_t

**evtimer**

- evtimer_t
  - evtimer_t
  - evtimer_t

⇨ xtimer_on_ztimer blocks STANDBY mode all the time

USEMODULE += ztimer ztimer_usec ztimer_msec ztimer_periph_rtt ztimer_xtimer_compat evtimer evtimer_on_ztimer

# evtimer_on_ztimer hasn't been merged, yet. See Pull Request #13661

**High Speed Timer**

- ZTIMER_USEC
  - ztimer_t
  - xtimer_t
  - xtimer_t

**Low Speed Timer**

- ZTIMER_MSEC
  - ztimer_t
  - ztimer_t

**evtimer**

- evtimer_t
  - evtimer_t
  - evtimer_t

⇨ ztimer_xtimer_compat doesn't implement xtimer_*64() methods

⇨ ztimer is utilized in our use case!
RIOT & Power Management: Status Quo?
Conclusion

RIOT has all important parts for PM inside ...

... but by default they aren't configured for reasonable power saving.

RIOT has three different timer systems ...

... but the *RIOT Developer Memo* could lead to one standard system.

(cf. [#12970](#))

RIOT is heading in the right direction for seamless power management!
References

1. Tadiran Batteries GmbH - Datasheet: SL-860
2. Dittrich, Menachem, Yamin, Adamas - Lithiumbatterien für Funksensornetzwerke
3. Microchip Technology Inc. - SAM R30 Microcontroller