1 Introduction

Purpose of this document is to provide an example using low power feature to help the first steps in application design. This feature is restricted to STA8090 product family. The original implementation illustrate periodic mode with or without standby, and way to synchronize or plan the different activities of system.

2 Contents

2.1 Index

2.2 List of Tables

2.3 List of Figures

3 Document Management

3.1 Revision History

Rev	Date	Author	Notes
1.0	13/06/2016	Vincent DELAUNAY	Creation

Table 1 Revision history

3.2 Acronyms

Keyword	Definition
GNSS	Global Navigation Satellite System – It can include any combination of different satellite constellations like GPS, GLONASS, SBAS etc.
UART	Universal Asynchronous Receiver Transmitter

Table 2 Acronyms

3.3 Reference Documents

Reference	Title	Author	Version
	STA8088/90_Firmware_Configuration	Andrea Di Girolamo	1.1
	STA8088/STA8090 SDK Usage	Fulvio Boggia	1.8

Table 3 References

3.4 Contact info

Keyword	Definition
J.Durand	jerome.durand@st.com
V.Delaunay	vincent.delaunay@st.com

Table 4 Contact name list

4 Low power application example

4.1 Functional behaviour of periodic mode

This example demonstrates the capacity of periodic mode to manage several activities, GNSS tasks and application task in parallel. This code creates and launches an application task which runs in parallel of other GNSS tasks.

The GNSS tasks are programmed to provide one fix every 25 seconds, and application task is to provide a simple time print out every 10 seconds.

Low power setup

At first start up (called **COLD STARTUP** in debug log), the low power interface configures the GNSS periodic mode. The activity task will define its own periodic activity to 10 seconds:

```
[POW_app] COLD STARTUP, setup periodic mode
[POW_app] Provide 1 fix every 25 seconds
[POW_app] Application activity plan every 10 seconds
```

System activity

The activity task is tagged by round counter and time in second:

```
[POW_app] Task is running
[POW_app] Task delayed during 0.00 second(s)
[POW_app] Task activity round 0 Time : 51.95 s
[POW_app] Task delayed during 9.96 second(s)
[POW_app] Task activity round 1 Time : 1.91 s
[POW_app] Task delayed during 10.00 second(s)
```

4.2 Functional behaviour with standby activated

In order to improve power consumption during idle phase, we run the same code with standby feature activated. Activate standby feature by declaring OPTION_POW_STANDBY into make `build.mk`:

```
EXT_CDEFS= EXAMPLE_POW OPTION_POW_STANDBY
```

The debug log output provides a details of timing and standby duration.

During standby phase, all peripherals are off, so the NMEA & debug stream are not present during this period.

Low power setup

At first start up (called **COLD STARTUP** in debug log), the low power interface configures the GNSS periodic mode. The activity task will define its own periodic activity to 10 seconds:

```
[POW_app] COLD STARTUP, setup periodic mode with standby
```

Restart after Standby

Leave standby and determinate source of restart is possible thanks to interface svc_pwr_StartupMode(). The standby wakeup is tagged with **STARTUP** into debug log, where duration appears.

```
[POW_app] STARTUP - LEAVE STANDBY after 10 second(s)-
```

After standby wake-up, the previous timer value, related to next activity, is no more relevant with the current timer unit. Interface svc_pwr_get_timer_adjustment, will deliver a projected previous os time value (VirtualPreviousOsTime), allowing to evaluate the time before the next activity. The previous timer event has to be stored in backup ram area to perform this operation.

```
// Update previous timer value from backup ram
    svc_pwr_get_timer_adjustment( &StandbyDuration_rtcbase,
&VirtualPreviousOsTime );
```

A second option, is to plan activity according to RTC and evaluate new RTC value at standby wake-up.

Waiting time before activity then appears in debug log:

```
 [POW_app] Task delayed during 9.97 second(s)
```

4.3 Set-up and execution

Note: *Pre-requisite: user needs to have Eclipse already installed and to have opened the `gnssapp_demo_freertos_gae` project or `gnssapp_demo_os20_rvct project` in it (refer to STA8088/STA8090 SDK Usage if needed).*

Main example files are located in `/apps/POW_app` directory: `POW_app.c` & `POW_app.h`. And a little piece of code is also located in `sta8090/gpsapp/main_demo[_fr or _os20].c`.

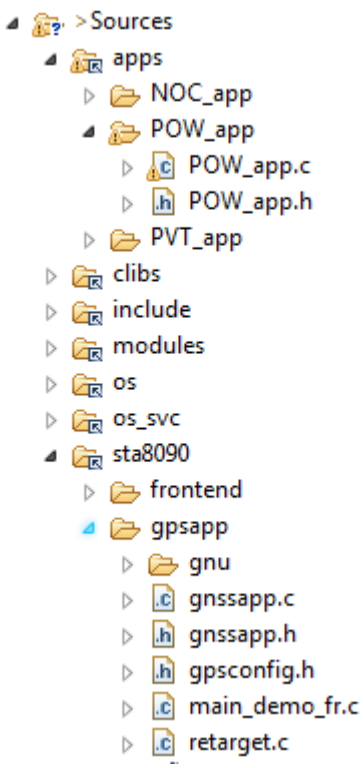

Figure 1: Screenshot of free rtos version files tree in Eclipse project

Implementation in `POW_app.c` mainly uses:

- GNSS libraries API
- Select your FREE RTOS API -gnssapp_demo_freertos_gae project- into `build.mk`. as bellow :

```
EXT_CDEFS= EXAMPLE_POW DEMO_USE_FREERTOS_API
```

- Or keep original operating system (gpOS & OS20) -gnssapp_demo_os20_rvct project:

```
EXT_CDEFS= EXAMPLE_POW
```

- In `main_demo[_fr or _os20].c` file: `POW_app_init()` is called in `main_idle_process()` function. This will launch the POW example task creation.
- Standby is destructive for variable, that's why persistent data are mapped into SRAM, and tagged **SRAM_STDBY_DATA:**

```
/* Backup area declaration */
#if defined(__STA8090__)
#pragma arm section zidata = "SRAM_STDBY_DATA"
#endif
SRAM_STDBY_DATA gpOS_clock_t pow_next_timer_activity;     /* Next
activity time */
```

```
SRAM_STDBY_DATA tUInt        pow_application_activity;    /*
Activity counter   */
#if defined(__STA8090__)
#pragma arm section zidata
#endif
```

- Backup data area must be indicated into scatter file according to tool chain :

sta8090_arm9gnssapp_template.scf

```
MEMORY
{
  …
  sram_stdby (rw)                  : org = 0x40007C00, len = 0x400
  …
}
```

sta8090_arm9gnssapp_template.map

```
SRAM_STDBY_AREA 0x40007C00  UNINIT 0x400
  {
    ……
    POW_app.o                    (SRAM_STDBY_DATA)
  }
```

Before starting building, make sure that `EXAMPLE_POW` compilation flag is enabled in `build.mk`.

Now:

- Build the software by selecting the wished target in Eclipse (`SOC SQI(SQI)`)
- Ensure your UART debug port is correctly configured as indicated in 6.1 Enable debug log. Data are sent to Debug Port using `GPS_DEBUG_MSG()` macro.
- Flash generated binary located in `/bin` directory, using either Trace32 or XLoader.
- Then execute the software and look at debug log. User should observe same kind of display than in 4.1 chapter.

5 Periodic mode behaviour

5.1 GNSS

Standby entry is conditioned by almanacs acquisition phase. So, do not expect standby before the end of this GNSS activity.

Periodic mode is also conditioned by ephemeris refresh, where standby entry is not possible. According to constellation selection, the GNSS will wake-up to acquire ephemeris and guarantee a better time to first fix.

5.2 Enhance consumption with synchronized application task

Application activity in this example, is not synchronised with GNSS low power period. This case requires to compute the next activity time after each standby wake-up.

In order to optimize power consumption, it's possible to synchronise application activity with GNSS tasks. After standby wake-up (we assume that wake-up is mapped on GNSS low power period), the application task will perform its action, and release wake lock for an infinite duration (waiting for next standby wake-up).Select this option by activating options OPTION_POW_STANDBY and OPTION_POW_SYNC into build.mk as bellow :

```
EXT_CDEFS= EXAMPLE_POW OPTION_POW_STANDBY OPTION_POW_SYNC
```

According to GNSS period, 25 seconds, the application activity will perform its task at the same period:

```
[POW_app] COLD STARTUP, setup periodic mode with standby
[POW_app] Provide 1 fix every 25 seconds
[POW_app] Application activity synchronized with GNSS
[POW_app] Task is running
[POW_app] Task activity round 0 Time : 14.65 s
[POW_app] STARTUP - LEAVE STANDBY after 23 second(s)-
[POW_app] Task is running
[POW_app] Task activity round 1 Time : 31.40 s
[POW_app] STARTUP - LEAVE STANDBY after 22 second(s)-
[POW_app] Task is running
[POW_app] Task activity round 2 Time : 57.40 s
```

5.3 Limitation

Standby wake-up mechanism is based on RTC alarm. In order to secure wake-up, the minimum duration of standby is limited to 2 seconds.

Debug log stream consumes a lot of time during initialization, and can affect the reactivity at wake-up. It's advised to disable it in final software to guarantee time accuracy.

6 Miscellaneous functionalities

6.1 Enable debug log

Referring to STA8088/90_Firmware_Configuration.pdf document, two parameters are used to enable the debug port:

- ID 100: Debug port number (where value 0...2 correspond to debug UART port number)
- ID 103: GPS Debug Mode (for which the value must be set to 0 to enable the Debug Mode)

So, as an example, if user wants to enable Debug Port on UART0, it needs to create the following configuration file:

`DebugCfg.txt`

```
100 -> 0
103 -> 0
```

And apply it to its binary file using this command line (generated binary file is located in `/bin` directory):

```
FWConfig.exe –f <generated *.bin file> -c DebugCfg.txt –o
<output_image_file>
```

Where `output_image_file` = the binary image which includes the new configuration

Another method to update the firmware configuration is to use the method described in STA8088/STA8090 SDK usage: write the parameters values you need to change in a customized dedicated `fwcfg.txt` file so that they will be taken into account during post build.

Important notice: the example above is only given as an example and user must refer to STA8088/90_Firmware_Configuration.pdf document to be certain of ID numbers and command line parameters to use.

6.2 Optimize the settings for periodic mode

The following feature can be disabled into Application ON/OFF configuration (ID = 200), to save power:

FEATURES	VALUES
PPS	0x1000000
WAS	0x4
GLONASS	0x20000

As depicted in chapter 6.1, use fwcgt.txt into SDK environment or application FWConfig.exe.

```
200 -> 18419640
```

7 Disclaimer

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2016 STMicroelectronics – All rights reserved