OpenCPU Watchdog Application Note

GSM/GPRS Module Series
Rev. OpenCPU_Watchdog_Application_Note_V1.0
Date: 2014-11-14
Our aim is to provide customers with timely and comprehensive service. For any assistance, please contact our company headquarters:

Quectel Wireless Solutions Co., Ltd.
Office 501, Building 13, No.99, Tianzhou Road, Shanghai, China, 200233
Tel: +86 21 5108 6236
Mail: info@quectel.com

Or our local office, for more information, please visit:
http://www.quectel.com/support/salesupport.aspx

For technical support, to report documentation errors, please visit:
http://www.quectel.com/support/techsupport.aspx

GENERAL NOTES
QUECTEL OFFERS THIS INFORMATION AS A SERVICE TO ITS CUSTOMERS. THE INFORMATION PROVIDED IS BASED UPON CUSTOMERS’ REQUIREMENTS. QUECTEL MAKES EVERY EFFORT TO ENSURE THE QUALITY OF THE INFORMATION IT MAKES AVAILABLE. QUECTEL DOES NOT MAKE ANY WARRANTY AS TO THE INFORMATION CONTAINED HEREIN, AND DOES NOT ACCEPT ANY LIABILITY FOR ANY INJURY, LOSS OR DAMAGE OF ANY KIND INCURRED BY USE OF OR RELIANCE UPON THE INFORMATION. ALL INFORMATION SUPPLIED HEREIN IS SUBJECT TO CHANGE WITHOUT PRIOR NOTICE.

COPYRIGHT
THIS INFORMATION CONTAINED HERE IS PROPRIETARY TECHNICAL INFORMATION OF QUECTEL CO., LTD. TRANSMITTABLE, REPRODUCTION, DISSEMINATION AND EDITING OF THIS DOCUMENT AS WELL AS UTILIZATION OF THIS CONTENTS ARE FORBIDDEN WITHOUT PERMISSION. OFFENDERS WILL BE HELD LIABLE FOR PAYMENT OF DAMAGES. ALL RIGHTS ARE RESERVED IN THE EVENT OF A PATENT GRANT OR REGISTRATION OF A UTILITY MODEL OR DESIGN.

Copyright © Quectel Wireless Solutions Co., Ltd. 2014. All rights reserved.
About the Document

History

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Author</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>2014-11-14</td>
<td>Stanley YONG</td>
<td>Initial</td>
</tr>
</tbody>
</table>
Contents

About the Document .......................................................................................................................... 2
Contents .............................................................................................................................................. 3
Table Index ....................................................................................................................................... 4
Figure Index ................................................................................................................................... 5

1 Introduction ..................................................................................................................................... 6

2 Principle of OpenCPU Watchdog .................................................................................................. 7
  2.1. Hardware Reference Design ..................................................................................................... 7
  2.2. Software Principle ................................................................................................................... 10
  2.3. Duty Time and Feed Watchdog ............................................................................................... 11

3 API Functions ................................................................................................................................ 12
  3.1. QI_WTD_Init ............................................................................................................................ 12
  3.2. QI_WTD_Start .......................................................................................................................... 13
  3.3. QI_WTD_Feed .......................................................................................................................... 13
  3.4. QI_WTD_Stop ......................................................................................................................... 14

4 Program Watchdog ......................................................................................................................... 15
  4.1. For Single Task App .................................................................................................................. 15
  4.2. For Multitasks App .................................................................................................................... 16

5 Appendix A Reference .................................................................................................................. 17
## Table Index

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE 1</td>
<td>RECOMMENDED COMPONENTS</td>
<td>8</td>
</tr>
<tr>
<td>TABLE 2</td>
<td>OWNER FOR FEEDING EXTERNAL WATCHDOG</td>
<td>11</td>
</tr>
<tr>
<td>TABLE 3</td>
<td>TERMS AND ABBREVIATIONS</td>
<td>17</td>
</tr>
</tbody>
</table>
Figure Index

FIGURE 1: HARDWARE DESIGN SKETCH FOR WATCHDOG ................................................................. 7
FIGURE 2: HARDWARE REFERENCE DESIGN FOR WATCHDOG .................................................... 8
FIGURE 3: WATCHDOG TIMING DIAGRAM .................................................................................... 9
FIGURE 4: SOFTWARE PRINCIPLE FOR WATCHDOG SOLUTION ............................................... 10
1 Introduction

This document mainly introduces the OpenCPU watchdog solution, API functions and how to program the watchdog as well.
2 Principle of OpenCPU Watchdog

OpenCPU supports multitasking. Developer may design multitasks to implement the application. To totally prevent each task of the application from getting stuck, OpenCPU has designed an effective watchdog solution that can monitor all user tasks.

2.1. Hardware Reference Design

The following sketch map demonstrates the hardware principle.

![Figure 1: Hardware Design Sketch for Watchdog](image-url)
The following schematic shows the reference design of hardware.

![Schematic of Hardware Reference Design for Watchdog](image)

**Figure 2: Hardware Reference Design for Watchdog**

In the schematic, the watchdog chip “TPS3823-33DBVR” is used. The watchdog chip must have the timeout of at least 1.6 seconds. The following table lists the recommended components.

**Table 1: Recommended Components**

<table>
<thead>
<tr>
<th>Location</th>
<th>Part No.</th>
<th>Descriptions</th>
<th>Vendor</th>
</tr>
</thead>
<tbody>
<tr>
<td>U2</td>
<td>TPS3823-33DBVR</td>
<td>IC PROCESSOR SUPERVISORY CIRCUITS</td>
<td>TI</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SOT23-5 RO</td>
<td></td>
</tr>
<tr>
<td>Q1</td>
<td>S1233CDS-T1-GE3</td>
<td>PMOSFET -12V 4A 45mOHM @VGS=-2.5V</td>
<td>VISHAY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SOT23</td>
<td></td>
</tr>
<tr>
<td>Q4</td>
<td>SE2306</td>
<td>MOSFET N-Channel Vds=20V Id=6A SOT-23</td>
<td>WILLAS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RO</td>
<td></td>
</tr>
<tr>
<td>Q2, Q3</td>
<td>DTC143ZEBTL</td>
<td>NPN 50V 100mA R1=4.7K R2=47K EMT3F RO</td>
<td>Rohm</td>
</tr>
<tr>
<td>D2</td>
<td>RB160M-30TR</td>
<td>DIO Schottky 30V 1.0A SOD123 RO</td>
<td>Rohm</td>
</tr>
</tbody>
</table>
Combined with Figure 2, when 4V2 is powered on, after VDD becomes higher than $V_{IT}$ and a delay of $t_d$ time, RESET will be pulled high. Then VBAT will supply the module and the module is turned on by startup circuit. Thereafter, the module will output pulse to feed TPS3823-33DBVR and keep RESET high all along, and the TPS3823-33DBVR will monitor VDD all the time. Meanwhile, when VDD drops below the threshold voltage $V_{IT}$, or TPS3823-33DBVR does not receive pulse during $t_{(out)}$, RESET will be pulled down.

**NOTES**

1. The watchdog chip must have the timeout of at least 1.6 seconds.
2. The jumper “J1” will be shorted out when downloading firmware to module on the production line.
3. Pay attention to the position of decoupling capacitor C6, it must be put in front of MOSFET switch Q1, otherwise, it will cause a big dropout in VBAT that triggers watchdog reset.
4. It would be better to feed watchdog in the bootloader, because the time between power on and Operating System start is long and the watchdog will be reset during this time.
2.2. Software Principle

The following sketch map demonstrates the software principle.

![Watchdog Solution Diagram](image)

**Figure 4: Software Principle for Watchdog Solution**

The watchdog solution consists of three primary elements:

- **External Physical Watchdog**: An external watchdog chip, monitors whether the OpenCPU App works normally or not, and it will reset the module when some App task gets stuck. App needs to call `Ql_WTD_Init()` to initialize the external watchdog in any task.

- **Logic Watchdog**: One or more software watchdogs, serves for App tasks. Each task may create a logic watchdog using the API function `Ql_WTD_Start()`.

- **Watchdog Manager**: The software agent of external watchdog, which manages and monitors one or more logic watchdogs and feeds the external watchdog through a GPIO.

The watchdog solution has the following features:

- An external watchdog may monitor all App tasks.
- Customer App does not have to feed the external watchdog, and just needs to simply specify the I/O pin. The watchdog manager automatically feeds the external watchdog. But the customer App tasks have to periodically report its status to watchdog manager by calling `Ql_WTD_Feed()` in each task, in which a logic watchdog has started calling `Ql_WTD_Start()`.
- Each App task may specify the individual overflow time. Developer may specify the different overflow time according to the load of different task.
- When any logic watchdog overflows, the watchdog manager will not feed the external watchdog.
1. The interval for feeding external watchdog should be two thirds or less of the overflow-time of the watchdog chip. E.g.: If the overflow-time of the external watchdog is 1.6s, then it would be better when the feeding interval is set to 1s; if the overflow-time of the external watchdog is 3s, then it would be better that the feeding interval is set to 2s.

2. The interval for feeding logic watchdog should be at least twice as long as the interval for feeding external watchdog. The interval for feeding logic watchdog is not that sensitive. You can set it to a long time, such as 30s, 1min, etc. This interval value should be decided according to the load of the task.

2.3. Duty Time and Feed Watchdog

Besides the time that App runs, the external watchdog works in the boot course, App FOTA upgrade course and the production course as well. So the external watchdog has to be fed in whole working period.

The following table lists the working time of the external watchdog and the feeding owner.

<table>
<thead>
<tr>
<th>Period</th>
<th>Owner for Feeding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Booting</td>
<td>Core system</td>
</tr>
<tr>
<td>App Running</td>
<td>App</td>
</tr>
<tr>
<td>Upgrading App By FOTA</td>
<td>Core system</td>
</tr>
</tbody>
</table>

For the “Booting” and “Upgrading App by FOTA” course, developer just needs to specify the GPIO in “custom_sys_cfg.c” which is designed to connect to the external watchdog.

```c
static const ST_ExtWatchdogCfg wtdCfg = {
  PINNAME_GPIO0,  //Specify a pin which connects to the external watchdog, other GPIO is ok.
  PINNAME_END     //Specify another pin for watchdog if needed
};
```
3 API Functions

Developer may call `Ql_WTD_Init()` to initialize the external watchdog in any task, including specifying the I/O pin and feeding interval for external watchdog. The API function `Ql_WTD_Start()` is used to start a logic watchdog, and `Ql_WTD_Feed()` may feed the logic watchdog.

Once App program calls `Ql_WTD_Init()`, it must also call `Ql_WTD_Start()`, or the external watchdog will overflow.

3.1. QI_WTD_Init

This function initializes watchdog manager, which is responsible for feeding the external watchdog. It can specify the I/O pin and feeding interval.

The interval for feeding external watchdog should be two thirds or less of the overflow-time of the watchdog chip.

- **Prototype**

  ```c
  s32 QI_WTD_Init(s32 resetMode, Enum_PinName wtdPin, u32 interval);
  ```

- **Parameters**

  - `resetMode`:
    - `[in]` Must be zero.
  
  - `wtdPin`:
    - `[in]` I/O pin that connects to the WDI pin of external watchdog chip.
  
  - `interval`:

- **Return Value**

  - `QL_RET_OK`, this function succeeds.
  
  - `QL_RET_ERR_PARAM`, invalid parameter.
3.2. QI_WTD_Start

The function starts a logic watchdog with the specified interval. If needed, every task may call this function to start a logic watchdog service.

The interval for feeding logic watchdog should be at least twice as long as the interval for feeding external watchdog. The interval for feeding logic watchdog is not that sensitive. You can set it to a long time, such as 30s, 1min, and etc. And the interval value should be decided according to the load of the task.

- **Prototype**

  ```c
  s32 QI_WTD_Start(u32 interval);
  ```

- **Parameters**

  `interval`:
  
  - [in] The interval for feeding the logic watchdog. Unit: ms.

- **Return Value**

  This function returns a watchdog ID if succeeds.
  
  QL_RET_ERR_PARAM, invalid parameter.

3.3. QI_WTD_Feed

This function feeds the logic watchdog which is started by QI_WTD_Start().

- **Prototype**

  ```c
  void QI_WTD_Feed(s32 wtdID);
  ```

- **Parameters**

  `wtdID`:
  
  - [in] Watchdog ID, which is returned by QI_WTD_Start().

- **Return Value**

  None.
3.4. QI_WTD_Stop

This function stops the specified logic watchdog.

● Prototype

```c
void QI_WTD_Stop(s32 wtdID);
```

● Parameters

`wtdID:`

[in] Watchdog ID, which is returned by `QI_WTD_Start()`.

● Return Value

None.
4 Program Watchdog

The following codes show how to program watchdog in one task and multitasks.

4.1. For Single Task App

```c
void proc_main_task(s32 TaskId)
{
    ST_MSG msg;
    s32 wtdId;

    // Init watchdog, GPIO0, 1s interval for external watchdog (suppose that the overflow-timer of external
    // watchdog is 1.6s).
    QL_WTD_Init(1, PINNAME_GPIO0, 1000);

    // Start a logic watchdog service in main task, the max. interval is 5s
    wtdId = QL_WTD_Start(5*1000);

    // Register & start a timer to feed the logic watchdog.
    QL_Timer_Register(TIMER_ID_WTD1, callback_onTimer, &wtdId);
    QL_Timer_Start(TIMER_ID_WTD1, 3000, TRUE);  // The real feeding interval is 3s.

    while (TRUE)
    {
        QL_memset(&msg, 0x0, sizeof(ST_MSG));
        QL_OS_GetMessage(&msg);
        switch(msg.message)
        {
            // ...
            default:
                break;
        }
    }
}

// Feed the logic watchdog in timer callback
void callback_onTimer(u32 timerId, void* param)
{
    // ...
}
```
### 4.2. For Multitasks App

Based on the case “one task”, developer just needs to start a new logic watchdog service in multitask, and start a timer to feed the watchdog periodically.

```c
void proc_subtask1(s32 TaskId)
{
    ST_MSG msg;
    s32 wtdId;

    // Start a logic watchdog service in subtask, the timeout is 10s.
    wtdId = QL_WTD_Start(10 * 1000);

    // Register & start a timer to feed the logic watchdog
    QL_Timer_Register(TIMER_ID_WTD2, callback_onTimer, &wtdId);
    QL_Timer_Start(TIMER_ID_WTD2, 8 * 1000, TRUE);    // The real interval of feeding watchdog is 8s.

    while (TRUE)
    {
        Ql_memset(&msg, 0x0, sizeof(ST_MSG));
        QL_OS_GetMessage(&msg);
        switch(msg.message)
        {
            // ...
            default:
                break;
        }
    }
}

// Feed the logic watchdog in timer callback
void callback_onTimer(u32 timerId, void* param)
{
    QL_WTD_Feed(*((s32*)param));
}
```
# Appendix A Reference

Table 3: Terms and Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>App</td>
<td>OpenCPU Application</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>FOTA</td>
<td>Firmware Over The Air</td>
</tr>
<tr>
<td>GPIO</td>
<td>General Purpose Input Output</td>
</tr>
<tr>
<td>External Watchdog</td>
<td>The external hardware watchdog chip</td>
</tr>
<tr>
<td>Watchdog Manager</td>
<td>The software agent of external watchdog, which manages one or more logic</td>
</tr>
<tr>
<td></td>
<td>watchdogs.</td>
</tr>
<tr>
<td>Logic Watchdog</td>
<td>The software watchdog that serves for App task.</td>
</tr>
</tbody>
</table>