
RT-THREAD ULOG 日志组件应用

笔记 - 基础篇

RT-THREAD 文档中心

上海睿赛德电子科技有限公司版权 ©2019



WWW.RT-THREAD.ORG

Tuesday 9th October, 2018

目录

目录	i
1 本文的目的和结构	1
1.1 本文的目的和背景	1
1.2 本文的结构	1
2 问题阐述	1
3 问题的解决	2
3.1 ulog 简介	2
3.2 ulog 框架总览	2
3.3 配置说明	3
3.4 日志标签	4
3.4.1. 使用标签保证日志模块化	4
3.4.2. 标签的定义方法	4
3.5 日志级别	5
3.5.1. 设定级别的分类	5
3.5.2. 日志输出 API 的使用方法	6
3.5.3. 输出 raw 日志	7
3.5.4. 在中断 ISR 中使用	7
3.6 断言	8
3.7 设置日志格式	8
3.8 hexdump	10
4 常见问题	11
5 参考	11
5.1 本文所有相关的 API	11

5.1.1.	API 列表	11
5.1.2.	核心 API 详解	11
5.1.3.	ulog 初始化	11
5.1.4.	ulog 反初始化	12
5.1.5.	LOG_X 日志输出 API	12
5.1.6.	ulog_x 日志输出 API	12
5.1.7.	输出 hex 格式日志	13

!!! abstract “摘要” 本应用笔记介绍了 RT-Thread ulog 组件的基本知识和 ulog 的基本使用方法，帮助开发者更好地上手、入门 RT-Thread ulog 组件。更多 ulog 组件的高级用法，详见《RT-Thread ulog 日志组件应用笔记 - 进阶篇》。

1 本文的目的和结构

1.1 本文的目的和背景

日志的定义：日志是将软件运行的状态、过程等信息，输出到不同的介质中（例如：文件、控制台、显示屏等），并进行显示和保存。为软件调试、维护过程中的问题追溯、性能分析、系统监控、故障预警等功能，提供参考依据。可以说，日志的使用，几乎占用的软件生命周期的至少 80% 的时间。

日志的重要性：对于操作系统而言，由于其软件的复杂度非常大，单步调试在一些场景下并不适合，所以日志组件在操作系统上几乎都是标配。完善的日志系统也能让操作系统的调试事半功倍。

ulog 的起源：RT-Thread 一直缺少小巧、实用的日志组件，而 ulog 的诞生补全了这块的短板。它将作为 RT-Thread 的基础组件被开源出来，让我们的开发者也能用上简洁易用的日志系统，提高开发效率。

1.2 本文的结构

本应用笔记将从以下几个方面来介绍 RT-Thread ulog 组件：

- ulog 组件简介、框架总览
- ulog 组件的配置
- ulog 组件基本功能的使用

2 问题阐述

本应用笔记将围绕下面几个问题来介绍 RT-Thread ulog 组件。

- ulog 组件的主要功能有哪些？
- 常用的日志接口有哪些？
- 如何使用 ulog？

想要解决这些问题，首先需要认识 RT-Thread ulog 组件基本功能，然后熟悉常用的日志 API，最后将在 qemu 平台上演示 ulog 的使用方法。

3 问题的解决

3.1 ulog 简介

ulog 是一个非常简洁、易用的 C/C++ 日志组件，第一个字母 u 代表 ，即微型的意思。它能做到最低 **ROM<1K, RAM<0.2K** 的资源占用。ulog 不仅有小巧体积，同样也有非常全面的功能，其设计理念参考的是另外一款 C/C++ 开源日志库：EasyLogger（简称 elog），并在功能和性能等方面做了非常多的改进。主要特性如下：

- 日志输出的 **后端多样化**，可支持例如：串口、网络、文件、闪存等后端形式；
- 日志输出被设计为 **线程安全**的方式，并支持 **异步输出模式**；
- 日志系统 **高可靠**，在中断 ISR、Hardfault 等复杂环境下依旧可用；
- 支持 **动态/静态**开关控制全局的日志输出级别；
- 各模块的日志支持 **动态/静态**设置输出级别；
- 日志内容支持按 **关键词及标签**方式进行全局过滤；
- API 和日志格式可兼容 **linux syslog** ；
- 支持以 hex 格式 dump 调试数据到日志中；
- 兼容 rtdbg（RTT 早期的日志头文件）及 EasyLogger 的日志输出 API。

3.2 ulog 框架总览

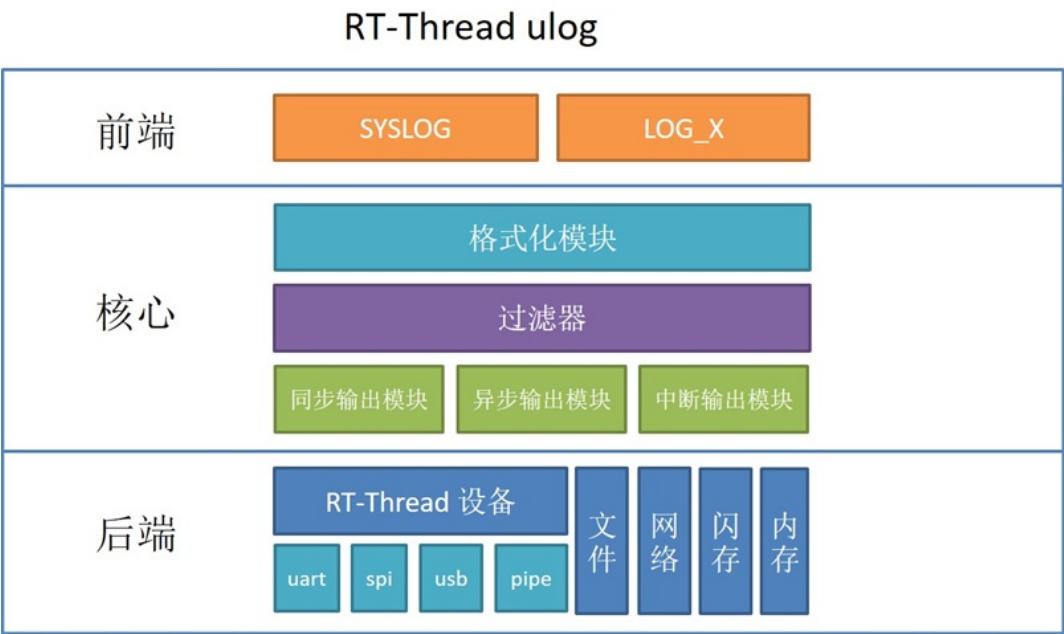


图 1: ulog 框架

上图为 ulog 的内部框架图，由此可见：

- **前端**：该层作为离应用最近的一层，给用户提供了 syslog 及 LOG_X 两类 API 接口，方便用户在不同的场景中使用；
- **核心**：中间核心层的主要工作是将上层传递过来的日志，按照不同的配置要求进行格式化与过滤然后生成日志帧，最终通过不同的输出模块，输出到最底层的后端设备上；
- **后端**：接收到核心层发来的日志帧后，将日志输出到已经注册的日志后端设备上。

3.3 配置说明

下载 RT-Thread 源码，使用 env 工具进入 `rt-thread\bsp\qemu-vexpress-a9` 文件夹，输入 `menuconfig` 打开配置菜单，在 `RT-Thread Components` → `Utilities` 下可以看到 ulog 的配置项，将其使能可以看到如下配置界面：

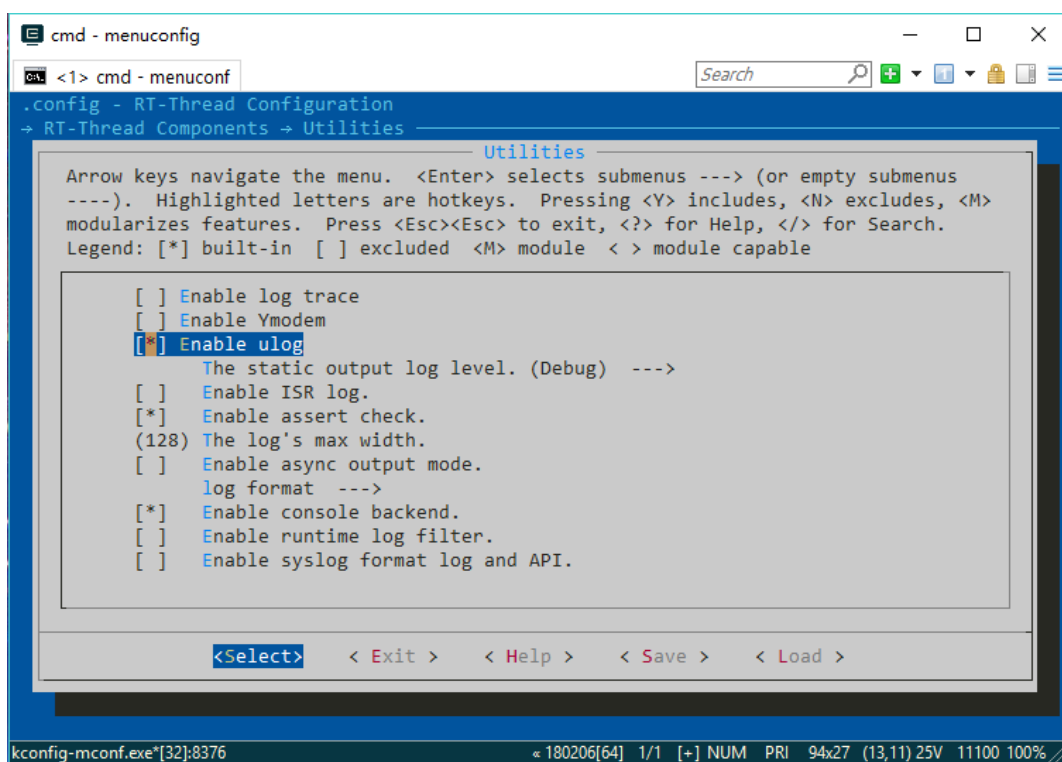


图 2: ulog 配置

每个选项配置说明如下：

- **The static output log level. (Debug)**：选择静态的日志输出级别。选择完成后，比设定级别低的日志（这里特指使用 LOG_X API 的日志）将不会被编译到 ROM 中。
- **Enable ISR log**：使能中断 ISR 日志，即在 ISR 中也可以使用日志输出 API。
- **Enable assert check**：使能断言检查。关闭后，断言的日志将不会被编译到 ROM 中。
- **The log's max width**：日志的最大长度。由于 ulog 的日志 API 按行作为单位，所以这个长度也代表一行日志的最大长度。
- **Enable async output mode**：使能异步日志输出模式。开启这个模式后，日志不会立刻输出到后端，而是先缓存起来，然后交给日志输出线程（例如：idle 线程）去输出。

该模式的好处有很多，将在《RT-Thread ulog 日志组件应用笔记 - 进阶篇》中详细介绍。

- **log format**：配置日志的格式，例如：时间信息，颜色信息，线程信息，是否支持浮点等等。
- **Enable console backend**：使能控制台作为后端。使能后日志可以输出到控制台串口上。建议保持开启。
- **Enable runtime log filter**：使能运行时的日志过滤器，即动态过滤。使能后，日志将支持按标签、关键词等方式，在系统运行时进行动态过滤。
- **Enable syslog format log and API**：使能 linux syslog API 及对应的日志格式。

使用默认配置即可，保存并退出 menuconfig。

3.4 日志标签

标签（tag）是一种常见的分类方式，ulog 也给每条 log 赋予了标签的属性，便于分类管理。

3.4.1. 使用标签保证日志模块化

由于日志输出量的不断增大，为了避免日志被杂乱无章的输出出来，就需要给每条日志按照 模块化方式，使用标签进行分类。例如：Wi-Fi 驱动模块的日志使用 `wifi.driver` 标签，Wi-Fi 设备管理模块日志使用 `wifi.mgmt` 作为标签。

每条日志的标签属性也可以被输出并显示出来，同时 ulog 还可以设置每个标签（模块）对应日志的输出级别，当前不重要模块的日志可以选择性关闭，不仅降低 ROM 资源，还能帮助开发者过滤无关日志。

各个标签（模块）对应日志的输出级别也支持在运行时动态调整，详见《RT-Thread ulog 日志组件应用笔记 - 进阶篇》。

3.4.2. 标签的定义方法

参见 `rt-thread\examples\ulog_example.c` ulog 例程文件，在文件顶部有定义 `LOG_TAG` 宏：

```
#define LOG_TAG          "example"          //该模块对应的标签。不定义时，
    默认：NO_TAG
#define LOG_LVL          LOG_LVL_DBG       //该模块对应的日志输出级别。不
    定义时，默认：调试级别
#include <ulog.h>                //必须在 LOG_TAG 与 LOG_LVL 下
    面
```

需要注意的，定义日志标签时，必须位于 `##include <ulog.h>` 的上方，否则会使用默认的 `NO_TAG`（不推荐定义在头文件中定义这些宏）。

日志标签的作用域是当前源码文件，项目源代码通常也会按照模块进行文件分类。所以在定义标签时，可以指定模块名、子模块名作为标签名称，这样不仅在日志输出显示时清晰直观，也能方便后续按标签方式动态调整级别或过滤。

3.5 日志级别

日志级别代表了日志的重要性，在 ulog 中 由高到低，有如下几个日志级别

级别	名称	描述
LOG_LVL_ASS	断言	发生无法处理、致命性的错误，以至于系统无法继续运行的断言日志
LOG_LVL_ERROR	错误	发生严重的、不可修复的错误时输出的日志属于错误级别日志
LOG_LVL_WARN	警告	出现一些不太重要的、具有可修复性的错误时，会输出这些警告日志
LOG_LVL_INFO	信息	给本模块上层使用人员查看的重要提示信息日志，例如：初始化成功，当前工作状态等。该级别日志一般在量产时依旧保留
LOG_LVL_DEBUG	调试	给本模块开发人员查看的调试日志，该级别日志一般在量产时关闭

3.5.1. 设定级别的分类

在 ulog 中，可分为如下几类日志级别

- **静态级别与动态级别：**是按照日志是否可以在 运行时修改 进行分类。比静态级别低的日志（这里特指使用 LOG_X API 的日志）将不会被编译到 ROM 中，最终也不会输出、显示出来。而动态级别可以管控的是高于或等于静态级别的日志。在 ulog 运行时，比动态级别低的日志会被过滤掉。
- **全局级别与模块级别：**是按照 作用域 进行的分类。在 ulog 中每个文件（模块）也可以设定独立的日志级别。全局级别作用域大于模块级别，也就是模块级别只能管控那些高于或等于全局级别的模块日志。

综合上面分类可以看出，在 ulog 可以通过以下 4 个方面来设定日志的输出级别

- **全局静态日志级别：**在 menuconfig 中配置，对应 ULOG_OUTPUT_LVL 宏
- **全局动态日志级别：**使用 void ulog_global_filter_lvl_set(rt_uint32_t level) 函数来设定
- **模块静态日志级别：**在模块（文件）内定义 LOG_LVL 宏，与日志标签宏 LOG_TAG 定义方式类似
- **模块动态日志级别：**使用 int ulog_tag_lvl_filter_set(const char *tag, rt_uint32_t level) 函数来设定

它们的作用范围关系如下：

全局静态 > 全局动态 > 模块静态 > 模块动态

日志输出 API 的使用

ulog 主要有两种日志输出 API

- `LOG_X("msg")` 宏 API：X 对应的是不同级别的第一个字母大写，API 为 `LOG_D`、`LOG_E` 等。这种方式是首选，一方面因为其 API 格式简单，入参只有一个即日志信息，再者还支持按模块静态日志级别过滤。
- `ulog_x("tag", "msg")` 宏 API：x 对应的是不同级别的简写，这个 API 适用于在一个文件中使用不同 tag 输出日志的情况。

3.5.2. 日志输出 API 的使用方法

下面将以 ulog 例程进行介绍，打开 `rt-thread\examples\ulog_example.c` 可以看到，顶部有定义该文件的标签及静态优先级

```
#define LOG_TAG          "example"
#define LOG_LVL          LOG_LVL_DBG
#include <ulog.h>
```

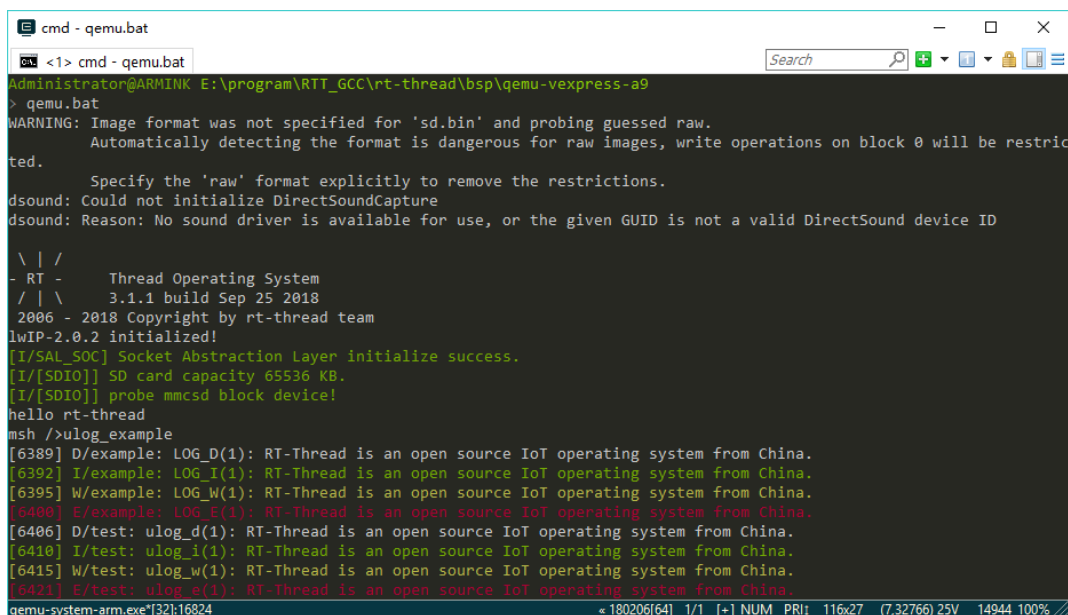
在 `void ulog_example(void)` 函数中有使用 `LOG_X` API，大致如下：

```
/* output different level log by LOG_X API */
LOG_D("LOG_D(%d): RT-Thread is an open source IoT operating system from
      China.", count);
LOG_I("LOG_I(%d): RT-Thread is an open source IoT operating system from
      China.", count);
LOG_W("LOG_W(%d): RT-Thread is an open source IoT operating system from
      China.", count);
LOG_E("LOG_E(%d): RT-Thread is an open source IoT operating system from
      China.", count);
```

这些日志输出 API 均支持 printf 格式，并且会在日志末尾自动换行。

下面将在 qemu 上展示下 ulog 例程的运行效果：

- 将 `rt-thread\examples\ulog_example.c` 拷贝至 `rt-thread\bsp\qemu-vexpress-a9\applications` 文件夹下
- 在 env 中进入 `rt-thread\bsp\qemu-vexpress-a9` 目录
- 确定之前已执行过 ulog 的配置后，执行 `scons` 命令并等待编译完成
- 运行 `qemu.bat` 来打开 RT-Thread 的 qemu 模拟器
- 输入 `ulog_example` 命令，即可看到 ulog 例程运行结果，大致效果如下图



```

cmd - qemu.bat
Administrator@ARMINK E:\program\RTT_GCC\rt-thread\bsp\qemu-vexpress-a9
> qemu.bat
WARNING: Image format was not specified for 'sd.bin' and probing guessed raw.
        Automatically detecting the format is dangerous for raw images, write operations on block 0 will be restric
ted.
        Specify the 'raw' format explicitly to remove the restrictions.
dsound: Could not initialize DirectSoundCapture
dsound: Reason: No sound driver is available for use, or the given GUID is not a valid DirectSound device ID

\ | /
- RT -   Thread Operating System
/ | \   3.1.1 build Sep 25 2018
2006 - 2018 Copyright by rt-thread team
lwIP-2.0.2 initialized!
[I/SAL_SOC] Socket Abstraction Layer initialize success.
[I/[SDIO]] SD card capacity 65536 KB.
[I/[SDIO]] probe mmcblk0 block device!
hello rt-thread
msh />ulog_example
[6389] D/example: LOG_D(1): RT-Thread is an open source IoT operating system from China.
[6392] I/example: LOG_I(1): RT-Thread is an open source IoT operating system from China.
[6395] W/example: LOG_W(1): RT-Thread is an open source IoT operating system from China.
[6400] E/example: LOG_E(1): RT-Thread is an open source IoT operating system from China.
[6400] D/test: ulog_d(1): RT-Thread is an open source IoT operating system from China.
[6410] I/test: ulog_i(1): RT-Thread is an open source IoT operating system from China.
[6415] W/test: ulog_w(1): RT-Thread is an open source IoT operating system from China.
[6421] E/test: ulog_e(1): RT-Thread is an open source IoT operating system from China.
qemu-system-arm.exe[32]:16824

```

图 3: ulog 例程

可以看到每条日志都是按行显示，不同级别日志也有着不同的颜色。在日志最前面有当前系统的 tick，中间有显示日志级别及标签，最后面是具体的日志内容。在本文后面也会重点介绍这些日志格式及配置说明。

3.5.3. 输出 raw 日志

LOG_X 及 ulog_x 这类 API 输出都是带格式日志，有些时候需要输出不带任何格式的日志时，可以使用 LOG_RAW 或 void ulog_raw(const char *format, ...) 函数。例如：

```

LOG_RAW("\r");
ulog_raw("\033[2A");

```

3.5.4. 在中断 ISR 中使用

很多时候需要在中断 ISR 中输出日志，但是中断 ISR 可能会打断正在进行日志输出的线程。要保证中断日志与线程日志互不干涉，就得针对于中断情况进行特殊处理。

ulog 已集成中断日志的功能，但是默认没有开启，使用时打开 Enable ISR log 选项即可，日志的 API 与线程中使用的方式一致，例如：

```

#define LOG_TAG          "driver.timer"
#define LOG_LVL          LOG_LVL_DBG
#include <ulog.h>

void Timer2_Handler(void)
{

```

```
/* enter interrupt */
rt_interrupt_enter();

LOG_D("I'm in timer2 ISR");

/* leave interrupt */
rt_interrupt_leave();
}
```

这里说明下中断日志在 ulog 处于同步模式与异步模式下的不同策略：

- **同步模式下：**如果线程此时正在输出日志时来了中断，此时如果中断里也有日志要输出，会直接输出到控制台上，不支持输出到其他后端；
- **异步模式下：**如果发生上面的情况，中断里的日志会先放入缓冲区中，最终和线程日志一起交给日志输出线程来处理。

3.6 断言

ulog 也提供里断言 API：`ASSERT(表达式)`，当断言触发时，系统会停止运行，内部也会执行 `ulog_flush()`，所有日志后端将执行 flush。如果开启了异步模式，缓冲区中所有的日志也将被 flush。断言的使用示例如下：

```
void show_string(const char *str)
{
    ASSERT(str);
    ...
}
```

3.7 设置日志格式

ulog 支持的日志格式可以在 menuconfig 中配置，位于 `RT-Thread Components` → `Utilities` → `ulog` → `log format`，具体配置如下：

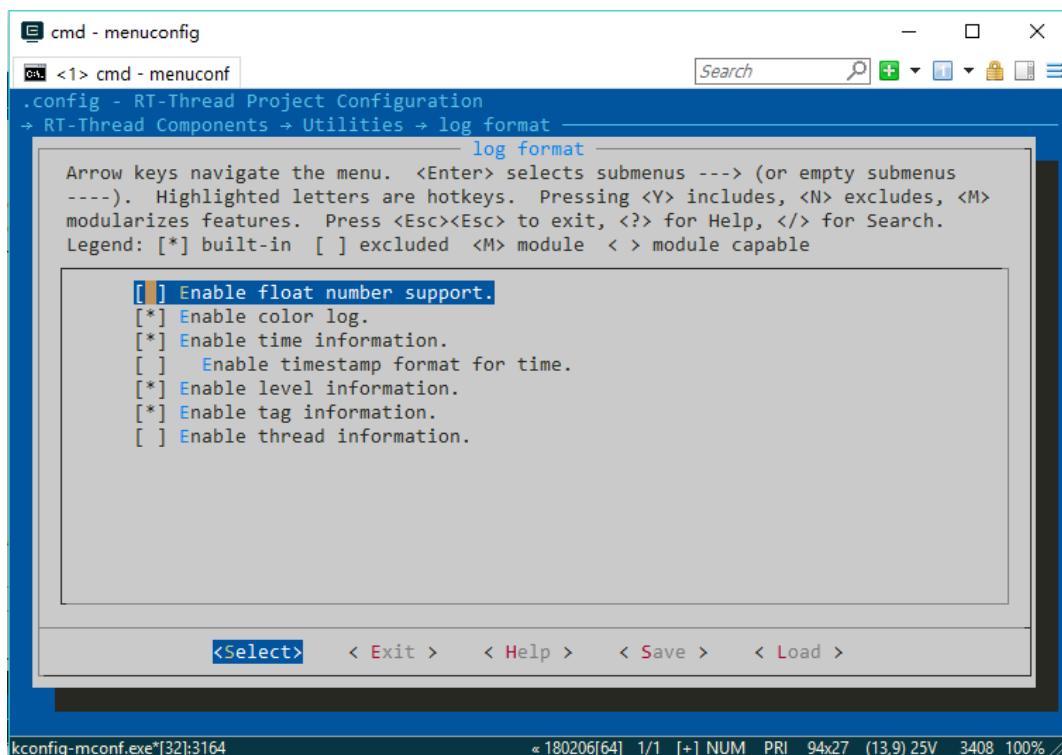


图 4: ulog 格式配置

分别可以配置：浮点型数字的支持（传统的 rtdbg/rt_kprintf 均不支持浮点数日志）、带颜色的日志、时间信息（包括时间戳）、级别信息、标签信息、线程信息。下面我们将这些选项 **全部选中**，保存后重新编译并在 qemu 中再次运行 ulog 例程，看下实际的效果：

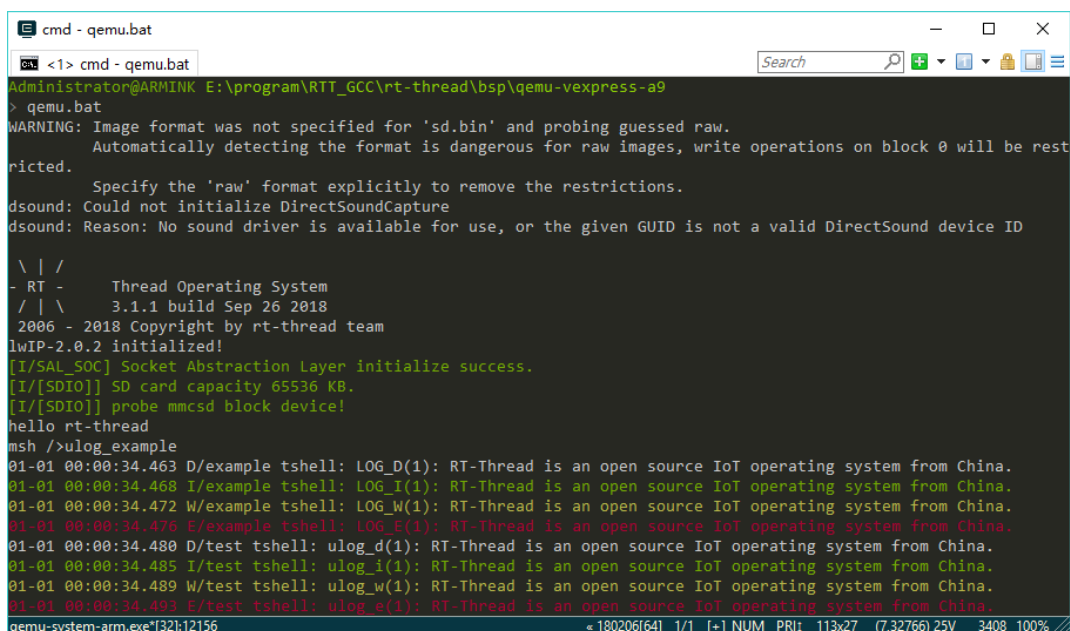


图 5: ulog 例程 (全部格式)

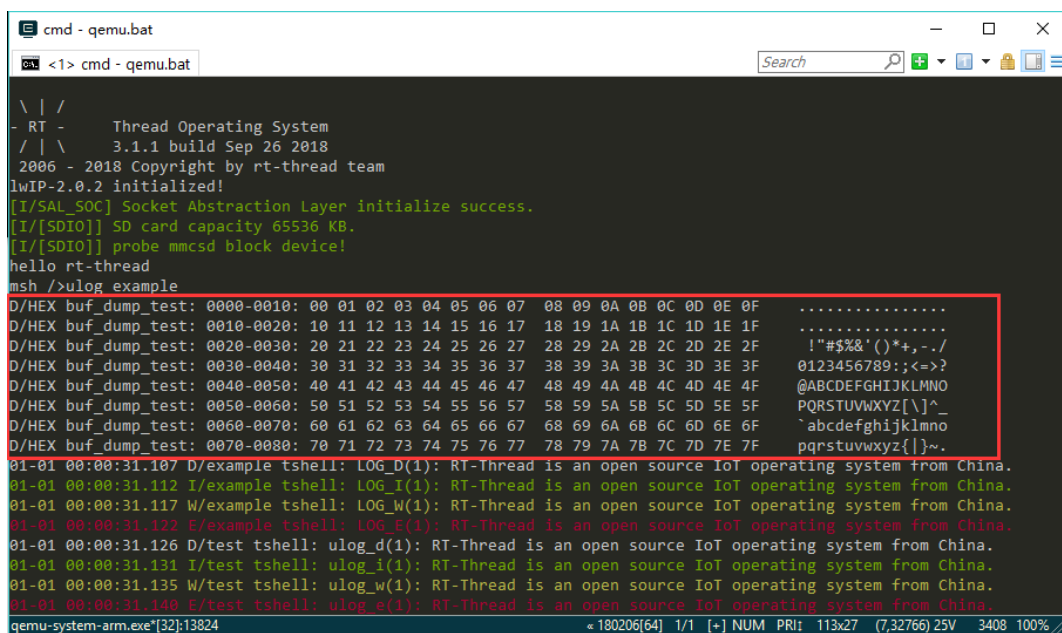
可以看出，相比第一次运行例程，新增的时间戳信息及线程信息已被输出出来。

3.8 hexdump

hexdump 也是日志输出时较为常用的功能，通过 hexdump 可以将一段数据以 hex 格式输出出来，对应的 API 为: `void ulog_hexdump(const char *name, rt_size_t width, rt_uint8_t *buf, rt_size_t size)`，下面看下具体的使用方法及运行效果：

```
/* 定义一个 128 个字节长度的数组 */
uint8_t i, buf[128];
/* 在数组内填充上数字 */
for (i = 0; i < sizeof(buf); i++)
{
    buf[i] = i;
}
/* 以 hex 格式 dump 数组内的数据，宽度为 16 */
ulog_hexdump("buf_dump_test", 16, buf, sizeof(buf));
```

可以将上面的代码拷贝到 ulog 例程中运行，然后再看下实际运行结果：



```
cmd - qemu.bat
<1> cmd - qemu.bat
\ | /
- RT -      Thread Operating System
/ | \      3.1.1 build Sep 26 2018
2006 - 2018 Copyright by rt-thread team
lwIP-2.0.2 initialized!
[I/SAL_SOC] Socket Abstraction Layer initialize success.
[I/[SDIO]] SD card capacity 65536 KB.
[I/[SDIO]] probe mmcblk0 block device!
hello rt-thread
msh />ulog example
D/HEX buf_dump_test: 0000-0010: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F .....
D/HEX buf_dump_test: 0010-0020: 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F .....
D/HEX buf_dump_test: 0020-0030: 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F !"#%&'()*+,-./
D/HEX buf_dump_test: 0030-0040: 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 0123456789;<=>?
D/HEX buf_dump_test: 0040-0050: 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F @ABCDEFGHIJKLMNO
D/HEX buf_dump_test: 0050-0060: 50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F PQRSTUVWXYZ[\]^_
D/HEX buf_dump_test: 0060-0070: 60 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F `abcdefghijklmnopqrstuvwxyz
D/HEX buf_dump_test: 0070-0080: 70 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D 7E 7F pqrstuvwxyz{|}~.
01-01 00:00:31.107 D/example tshell: LOG_D(1): RT-Thread is an open source IoT operating system from China.
01-01 00:00:31.112 I/example tshell: LOG_I(1): RT-Thread is an open source IoT operating system from China.
01-01 00:00:31.117 W/example tshell: LOG_W(1): RT-Thread is an open source IoT operating system from China.
01-01 00:00:31.122 E/example tshell: LOG_E(1): RT-Thread is an open source IoT operating system from China.
01-01 00:00:31.126 D/test tshell: ulog_d(1): RT-Thread is an open source IoT operating system from China.
01-01 00:00:31.131 I/test tshell: ulog_i(1): RT-Thread is an open source IoT operating system from China.
01-01 00:00:31.135 W/test tshell: ulog_w(1): RT-Thread is an open source IoT operating system from China.
01-01 00:00:31.140 E/test tshell: ulog_e(1): RT-Thread is an open source IoT operating system from China.
qemu-system-arm.exe[32]:13824 180206[64] 1/1 [+] NUM PRI: 113x27 (7,32766) 25V 3408 100%
```

图 6: ulog 例程 (hexdump)

可以看出，中部为 buf 数据的 16 进制 hex 信息，最右侧为各个数据对应的字符信息。

至此，关于 ulog 的入门基础已经介绍完了，想要了解更多关于 ulog 的进阶使用，可以继续查看《RT-Thread ulog 日志组件应用笔记 - 进阶篇》

4 常见问题

- 1、日志代码已执行，但是无输出
参考日志级别章节，了解日志级别分类，并检查日志过滤参数。还有种可能是不小心将控制台后端给关闭了，重新开启 `Enable console backend` 即可。
- 2、日志内容的末尾缺失
这是由于日志内容超出设定的日志的最大宽度。检查 `The log's max width` 选项，并增大其至合适的大小。
- 3、开启时间戳以后，为什么看不到毫秒级时间
这是因为 ulog 目前只支持在开启软件模拟 RTC 状态下，显示毫秒级时间戳。如需显示，只要开启 RT-Thread 软件模拟 RTC 功能即可。
- 4、每次 include ulog 头文件前，都要定义 `LOG_TAG` 及 `LOG_LVL`，可否简化
`LOG_TAG` 如果不定义，默认会使用 `NO_TAG` 标签，这样输出的日志会容易产生误解，所以标签的宏不建议省略。
`LOG_LVL` 如果不定义，默认会使用调试级别，如果该模块处于开发阶段这个过程可以省略，但是模块代码如果已经稳定，建议定义该宏，并修改级别为信息级别。

5 参考

5.1 本文所有相关的 API

5.1.1. API 列表

API	位置
<code>int ulog_init(void)</code>	<code>ulog.c</code>
<code>void ulog_deinit(void)</code>	<code>ulog.c</code>
<code>LOG_E(...)</code> / <code>LOG_W(...)</code> / <code>LOG_I(...)</code> / <code>LOG_D(...)</code> / <code>LOG_RAW(...)</code>	<code>ulog.h</code>
<code>ulog_e(TAG, ...)</code> / <code>ulog_w(TAG, ...)</code> / <code>ulog_i(TAG, ...)</code> / <code>ulog_d(TAG, ...)</code>	<code>ulog_def.h</code>
<code>void ulog_hexdump(const char name, rt_size_t width, rt_uint8_t buf, rt_size_t size)</code>	<code>ulog.c</code>

5.1.2. 核心 API 详解

5.1.3. ulog 初始化

```
int ulog_init(void)
```

在使用 ulog 前必须调用该函数完成 ulog 初始化。如果开启了组件自动初始化，API 也将被自动调用。

返回	描述
≥ 0	成功
-5	失败，内存不足

5.1.4. ulog 反初始化

```
void ulog_deinit(void)
```

当 ulog 不再使用时，可以执行该 deinit 释放资源。

5.1.5. LOG_X 日志输出 API

```
LOG_X(...)
```

该 API 是一个宏，**x** 对应的是不同级别的第一个字母大写。

注意：使用这个 API 前需在先在 ulog.h 头文件上方，定义 **LOG_TAG** 及 **LOG_LVL** 宏，详见：日志输出 API 的使用章节。

这个 API 使用就可以基于已定义好的标签，输出对应级别的日志。

参数	描述
...	日志内容，格式与 printf 一致

5.1.6. ulog_x 日志输出 API

```
ulog_x(TAG, ...)
```

该 API 是一个宏，**x** 对应的是不同级别的第一个字母小写。

注意：使用这个 API 前需在先在 ulog.h 头文件上方，定义 **LOG_LVL** 宏，详见：日志输出 API 的使用章节。

这个 API 可在输出对应级别的日志时指定标签，比 **LOG_X** 多一个入参，不推荐使用。

参数	描述
TAG	日志标签
...	日志内容，格式与 printf 一致

5.1.7. 输出 hex 格式日志

```
void ulog_hexdump(const char *name, rt_size_t width, rt_uint8_t *buf,
                  rt_size_t size)
```

以 16 进制 hex 格式 dump 数据到日志中，使用方法及效果详见 hexdump 章节

参数	描述
name	日志标签
width	一行 hex 内容的宽度（数量）
buf	待输出的数据内容
size	数据大小