Hi3510 watchdog

# Application Notes

| | |
|---|---|
| 文档版本 | 01 |
| 发布日期 | 2006-08-31 |
| BOM编码 | N/A |

深圳市海思半导体有限公司

深圳市海思半导体有限公司为客户提供全方位的技术支持，用户可与就近的海思办事处联系，也可直接与公司总部联系。


## 深圳市海思半导体有限公司

地址：　深圳市龙岗区坂田华为基地华为电气生产中心　邮编：518129

网址：　http://www.hisilicon.com

客户服务电话：　0755-28788858

客户服务传真：　0755-28788838

客户服务邮箱：　support@hisilicon.com




**商标声明**

 、Hisilicon、海思，均为深圳市海思半导体有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

**注意**

由于产品版本升级或其它原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

# 目 录

# 前 言

## 概述

本节介绍本文档的内容、对应的产品版本、适用的读者对象、行文表达约定、历史修订记录等。

## 产品版本

与本文档相对应的产品版本如下所示。

| 产品名称 | 产品版本 |
| --- | --- |
| Hi3510 芯片 | V100 |

## 读者对象

本文档主要适用于软件开发人员。

## 内容简介

本文档首先简要介绍 watchdog 的启动和功能，然后介绍 HiBoot 下 watchdog 的使用和移植，最后介绍 Linux 下 watchdog 的功能、规格、模块参数、编程接口和典型示例。

| 章节 | 内容 |
| --- | --- |
| 1 简介 | 简要介绍 watchdog 的启动和功能。 |
| 2 HiBoot watchdog | 详细介绍 HiBoot 下 watchdog 的使用和移植。 |
| 3 Linux watchdog | 详细介绍 Linux 下 watchdog 的监控模式、规格、监控模式参数、编程接口和应用示例。 |

# 约定

## 符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

| 符号 | 说明 |
| --- | --- |
| 危险 | 以本标志开始的文本表示有高度潜在危险，如果不能避免，会导致人员死亡或严重伤害。 |
| 警告 | 以本标志开始的文本表示有中度或低度潜在危险，如果不能避免，可能导致人员轻微或中等伤害。 |
| 注意 | 以本标志开始的文本表示有潜在风险，如果忽视这些文本，可能导致设备损坏、数据丢失、设备性能降低或不可预知的结果。 |
| 窍门 | 以本标志开始的文本能帮助您解决某个问题或节省您的时间。 |
| 说明 | 以本标志开始的文本是正文的附加信息，是对正文的强调和补充。 |

## 通用格式约定

| 格式 | 说明 |
| --- | --- |
| 宋体 | 正文采用宋体表示。 |
| **黑体** | 一级、二级、三级标题采用黑体。 |
| 楷体 | 警告、提示等内容一律用楷体，并且在内容前后增加线条与正文隔离。 |
| “Terminal Display”格式 | “Terminal Display”格式表示屏幕输出信息。此外，屏幕输出信息中夹杂的用户从终端输入的信息采用加粗字体表示。 |

# 修改记录

修订记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

## 文档版本 01 (2006-9-10)

第一次版本。

# 1 简介

## 关于本章

本章描述内容如下表所示。

| 标题 | 内容 |
| --- | --- |
| 1.1 介绍 | 简要介绍 watchdog 的性质。 |
| 1.2 时间特性和应用分析 | 描述从 HiBoot 到 Linux 的启动过程中 watchdog 的时间特性和典型的 watchdog 使用方法。 |

# 1.1 介绍

为保证系统运行的整个过程都有 watchdog 的监控，需要：

- HiBoot 启动时启动 watchdog

  为保证 HiBoot 的正常启动，watchdog 必须在系统被复位后的第一时间被使能。HiBoot 启动成功后，检测 HiBoot 的参数以确定是否继续使能 watchdog 和查看 watchdog 的超时时间。如果没有配置 watchdog 的相关参数或读取不到 watchdog 的相关参数，默认将使能 watchdog，并使用默认的 watchdog 超时时间。

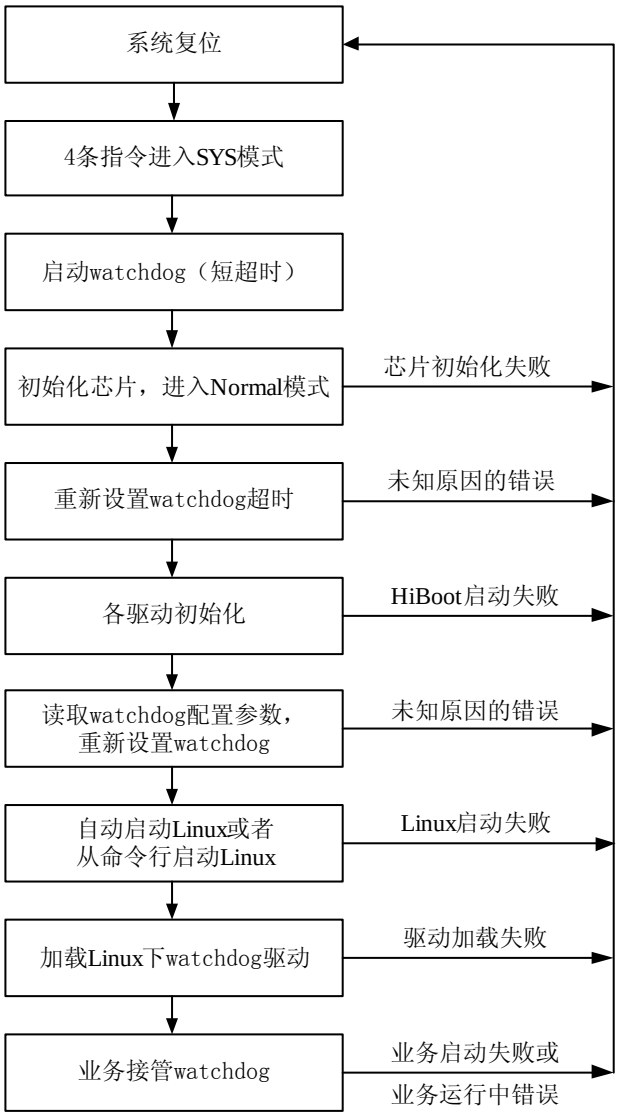
- Linux 启动时加载 watchdog

  在 watchdog 使能的情况下，可以直接加载内核启动 Linux，则在 Linux 的启动过程中 watchdog 是使能的。如果 Linux 启动失败，系统将被复位。Linux 启动成功，业务程序启动失败，没有在规定的时间内接管 watchdog，系统将被复位。

保证在 HiBoot 和 Linux 系统运行的阶段都有 watchdog 的监控，在发生任何意外的情况而导致业务不能正常运行时系统都能自动复位。

watchdog 在整个系统工作过程中的作用关系如图 1-1 所示。除了系统被复位后的最初阶段（最初的几条指令），其他阶段都处于 watchdog 的监控下。

图1-1 watchdog 与系统的关系




# 1.2 时间特性和应用分析

从 HiBoot 到 Linux 的启动过程中 watchdog 都是使能的，以下描述在启动过程中各环节的时间特性。

- HiBoot 启动到 Linux 时，watchdog 可以设置最长为 79 秒的超时，足够 Linux 启动并加载 watchdog 驱动。
- watchdog 驱动加载后会重新设置 watchdog 超时时间并初始化喂狗一次，又得到最长为 79 秒的超时，足够业务程序启动并接管 watchdog（不一定是业务启动完成才能接管）。
- 业务是否正常工作可以由接管 watchdog 的任务来监控，如果接管 watchdog 的任务发生故障，watchdog 超时后复位系统。

要有效使用 watchdog 的推荐做法是：业务启动一个监控任务，该任务负责喂狗并监控其他业务相关的任务。如果其他任务发生故障，则由监控任务负责处理（监控任务也可以简单的停止喂狗来重启系统）；如果监控任务故障，则 watchdog 复位系统。

如果系统只有一个业务在运行，可以在该任务的主循环里面喂狗。

# 2 HiBoot watchdog

## 关于本章

本章描述内容如下表所示。

| 标题 | 内容 |
| --- | --- |
| 2.1 watchdog 的使用 | 描述 HiBoot 中使用 watchdog。 |
| 2.2 watchdog 的移植 | 描述 watchdog 在不同单板间的移植。 |

# 2.1 watchdog 的使用

HiBoot 下可以通过其环境变量来配置 watchdog，示例如下：

```
setenv watchdog 45 ;     设置HiBoot下watchdog超时时间为45秒
setenv watchdog off;     禁止HiBoot下watchdog运行
```

saveenv 用来保存环境变量的设置，请参见《Hi3510 Linux 开发环境用户指南》。

HiBoot 下 watchdog 运行时，如果配置了 LED 指示灯，可以看到 LED 指示灯闪烁。

HiBoot 下 watchdog 的超时时间范围限制为 10 秒≤watchdog≤79 秒，范围限制原因如下：

- 上限为 79 秒，因为在默认总线时钟频率下，watchdog 定时器能达到的最大超时时间是 79 秒。
- 下限为 10 秒，为防止 watchdog 超时时间设置得太短，导致在启动 Linux 的过程中意外复位。

从系统的可靠性方面考虑，HiBoot 在自身启动过程中默认开启 watchdog，并使用默认的超时时间；HiBoot 启动成功后根据环境变量的设置来决定 watchdog 的配置，环境变量 setenv 的设置在设置后立即生效。如果读取不到 watchdog 配置参数或配置参数无效，则查看宏 CONFIG_HISILICON_WDT 的定义。如果 CONFIG_HISILICON_WDT 的定义是非零，默认打开 watchdog；如果定义为零，默认禁止 watchdog。

如果在 watchdog 禁止的情况下启动 Linux，则 Linux 启动过程中 watchdog 仍然是禁止的。

说明

建议将 CONFIG_HISILICON_WDT 定义为 1，这样在任何异常情况下 watchdog 都将被使能，默认设置为 0 只是为了保持 HiBoot 和以前版本的兼容。

# 2.2 watchdog 的移植

HiBoot 下 watchdog 的移植在 include/configs/hi3510-v100-p01-dvs01.h 文件中完成，只需要简单的选择需要的特性即可。

在该文件中找到/* Watchdog configs */定义：

```
/* Watchdog configs */
/* #define CONFIG_HISILICON_WDT 1 */
#define CONFIG_HISILICON_WDT      0     //是否使用海思芯片的watchdog
#define CONFIG_HISILICON_WDT_LED 1      //是否使用watchdog LED指示灯
#define HIDOG_BOOTUP_TIMEOUT        10          //Boot启动过程的默认超时时间
#define CONFIG_WDT_TIMEOUT          60          //Boot启动后的默认超时时间
#define HISILICON_WDT_LED_GPIO_GROUP   3        //watchdog LED使用的GPIO组
```

```
#define HISILICON_WDT_LED_GPIO_BITS    2        //watchdog LED使用的GPIO位
#define HISILICON_WDT_LED_ON_VAL       0        //点亮LED使用的电平
#define HISILICON_WDT_LED_PERIOD (CFG_HZ/2)        //LED闪烁的半周期
```

以上注释是在本文档中加入的，源代码中没有。如果不需要 watchdog 或 LED 指示灯，只需要将定义屏蔽掉即可。

如果要使用 watchdog LED 指示灯，则需要使能 WDT_LED 并配置 LED 使用的 GPIO。上面的配置中，指定 GPIO 3 的 BITS 2 来控制 LED，低电平时点亮 LED，亮灭时间各为 0.5 秒（CFG_HZ 的时间是 1 秒）。

watchdog 和 LED 的使能是完全独立的，可以只使能 watchdog 或 LED，也可以都使能。

# 3 Linux watchdog

## 关于本章

本章描述内容如下表所示。

| 标题 | 内容 |
| --- | --- |
| 3.1 监控模式 | 描述 Linux 下 watchdog 的监控模式。 |
| 3.2 规格 | 描述 Linux 下 watchdog 的规格。 |
| 3.3 监控模式参数 | 描述 Linux 下 watchdog 监控模式的参数。 |
| 3.4 编程接口 | 描述 Linux 下 watchdog 的编程接口函数。 |
| 3.5 应用示例 | 描述 Linux 下运行 watchdog 的应用示例。 |

## 3.1 监控模式

Linux 下 watchdog 有以下三种监控模式：默认监控模式用于监控 Linux 系统的正常工作；业务监控模式提供与业务关联的操作模式，以保证业务的正常运行；nowayout 模式适用于启动后就一直运行的业务，得到更高的可靠性。

- 默认监控模式

  模块启动时自动启动片内 watchdog 定时器，监控系统进程调度是否正常，如果系统任务调度发生异常（包括但不限于内核崩溃，硬件故障等）导致任务调度停止，watchdog 超时后复位系统。默认监控模式可以被禁用，以保证 watchdog 的使用完全由应用程序控制。

- 业务监控模式

  默认的监控模式只能保证 Linux 系统能正常工作，对业务进程没有监控能力。业务启动后可以接管 watchdog，由业务自己执行喂狗动作，以保证关键业务的正常运行，达到对业务进程的监控。

- nowayout 模式

  watchdog 模块加载时可指定启用 nowayout 模式，该模式一旦启用后就不能更改。该模式下 watchdog 一旦被业务进程接管后就无法停止，且无法返回默认监控模式，业务必须永远的进行喂狗动作。

## 3.2 规格

### 3.2.1 规范

设计上遵从 Linux 下的 watchdog 接口规范，可以兼容其他的开源应用程序，使应用程序的编写和移植更加容易。更多的信息，请参考 linux-2.6.14/Documentation/watchdog/ 下的相关文档。

### 3.2.2 超时范围

watchdog 超时范围为大于 0 秒小于 80 秒（watchdog 最大超时时间依赖于总线时钟频率，该处假定总线时钟频率为 108MHz）。

watchdog 超时时间以秒为单位，不支持更高精度的设置。这是由 Linux 操作系统本身的特性决定的。Linux 是非实时操作系统，任务的调度时间没有严格的保证，如果设置毫秒级别的 watchdog，正常情况下可能也会发生超时复位的情况，如果设置 5 秒以下的超时都可能发生“误杀”的情况，建议最短超时时间应该在 10 秒以上，以保证系统正常工作。

watchdog 最大超时时间依赖于总线时钟频率，总线时钟频率越高则可设置的 watchdog 最大超时时间越短。

### 3.2.3 复位方式

watchdog 触发芯片复位，单板其他硬件的复位可由芯片发起。

### 3.2.4 总线时钟频率自适应

由于 watchdog 时钟超时时间设置依赖于总线时钟频率，总线时钟频率的更改将引起 watchdog 超时时间的变化，因此将 watchdog 驱动连接到系统的时钟管理模块，在系统时钟频率发生变化时，系统将自动调整当前的 watchdog 超时时间，以达到总线时钟频率改变后能自动校准 watchdog 超时时间。可以通过运行 cat /proc/clocks 命令看到当前的 watchdog 时钟信息。

说明

在较低总线时钟频率下设置的超时值，当系统动态切换到一个更高的总线时钟频率后，可能会发生当前的超时值超过 watchdog 定时器能达到的最大超时时间的情况。这时当前的超时时间将被自动截短为允许的最大值。比如，在 50MHz 的总线时钟频率下可以设置 90 秒的超时，那么当总线时钟频率切换回 108MHz 的时候，超时时间将自动截短为 79 秒。

## 3.3 监控模式参数

watchdog 以模块的方式加载，模块加载后将创建 watchdog 设备文件 /dev/misc/watchdog。监控模式参数分别为 default_margin、nowayout 和 nodeamon。

- default_margin 用于设置默认监控模式下的超时时间，范围请参见“3.2.2 超时范围”，默认值为 60 秒。
- nowayout 用于设置 watchdog 是否启用 nowayout 模式，默认值为 0，不启用 nowayout 模式。如果需要，设置 nowayout 参数为 1，启用 nowayout 模式。
- 设置 nodeamon 参数为 1，可以禁用内核态的默认监控模式，这要求必须有应用程序一直接管 watchdog。

注意

在 nodeamon 参数为 1 的情况下，模块仍然可以被卸载，卸载后将自动关闭 watchdog 运行；而在 nowayout=1，nodeamon=1 的情况下，则 watchdog 必须有应用程序控制，并且 watchdog 一旦被使用模块就不可卸载（也就是说必须有应用程序的全程喂狗），这是最强的条件。

示例：

```
insmod hidog.ko ;使用默认参数插入模块
insmod hidog.ko default_margin=30  ;            设置默认监控超时为30秒
insmod hidog.ko default_margin=30 nowayout=1  ;     启用nowayout模式
insmod hidog.ko nowayout=1  ;                   启用nowayout模式，使用默认超时
insmod hidog.ko default_margin=30 nowayout=1 nodeamon=1 ;禁用内核态的默认监控模式
```

说明

在发布的 rootbox 里面，默认系统启动时将加载 watchdog 驱动，加载操作在/etc/init.d/下的配置文件中，该配置文件名为命令 uname -r 输出的名称，可更改该文件中加载 hi3510_wdt 模块的操作来改变驱动的默认行为。

# 3.4 编程接口

## 3.4.1 API 描述

所有对 watchdog 的操作都使用 Linux 文件操作系统调用完成，函数的原型如下：

```
int open(const char *pathname, int flags);
ssize_t write(int fd, const void *buf, size_t count);
int ioctl(int d, int cmd, unsigned long arg);
int close(int fd);
```

上述函数的返回值均遵从 Linux 下约定的返回值，即返回 0 表示成功，负数为失败。

要使用 watchdog，需要包含以下头文件：

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <linux/watchdog.h>
```

## 3.4.2 接管 watchdog

应用程序调用 open 函数，打开/dev/misc/watchdog 文件，应用程序接管 watchdog。

注意

系统运行过程中，watchdog 设备文件只能被一个应用程序打开一次。

应该使用 O_WRONLY 为标志打开文件。watchdog 一旦被接管，应用程序必须及时进行喂狗，否则系统会被复位。open 操作后默认的 watchdog 超时值为 default_margin。

示例：

```
int fd = open("/dev/misc/watchdog",O_WRONLY);
if (fd == -1)
{
    perror("watchdog");
```

```
    exit(1);
}
```

## 3.4.3 定时喂狗

定时喂狗有 ioctl 和 write 两种方式，效果是完全相同的。

- write 方式

  最简单的喂狗方式是往打开的 watchdog 设备文件写入一个空字符串（"\0"）。

  示例：

```
while (1)
{
    write(fd, "\0", 1);
    sleep(10);
}
```

- ioctl 方式

  使用 ioctl 方式可以用命令的方式显式的喂狗，这需要调用 ioctl，并在 cmd 参数中传入 WDIOC_KEEPALIVE。

  示例：

```
while (1)
{
    ioctl(fd, WDIOC_KEEPALIVE, 0);
    sleep(10);
}
```

## 3.4.4 设置超时时间

通过 ioctl 可以设置超时时间。调用 ioctl 并在 cmd 参数中传入命令 WDIOC_SETTIMEOUT，arg 参数中传入一个 int 型的指针，该指针指向的整型数就是要设置的超时值。

示例：

```
int timeout = 45;
ioctl(fd, WDIOC_SETTIMEOUT, &timeout);
printf("The timeout was set to %d seconds\n", timeout);
```

## 3.4.5 获取超时时间

和设置超时时间类似，调用 ioctl 并在 cmd 参数中传入命令 WDIOC_GETTIMEOUT，arg 参数中传入一个 int 型的指针，该指针指向的整型数就是获取的超时时间。

示例：

```
int timeout;
ioctl(fd, WDIOC_GETTIMEOUT, &timeout);
printf("The timeout was is %d seconds\n", timeout);
```

## 3.4.6 获取 watchdog 支持特性

调用 ioctl 并在 cmd 参数中传入命令 WDIOC_GETSUPPORT。可获得 watchdog 支持的特性，在 arg 参数中传入一个 struct watchdog_info 的指针。该指针指向的结构值就是 watchdog 支持的特性。

struct watchdog_info 结构的定义如下：

```
struct watchdog_info {
    __u32 options;              /* Options the card/driver supports */
    __u32 firmware_version;     /* Firmware version of the card */
    __u8  identity[32];         /* Identity of the board */
};
```

options 指出了当前 watchdog 支持的特性。Hi3510 watchdog 支持以下三个特性：

- WDIOF_SETTIMEOUT
- WDIOF_KEEPALIVEPING
- WDIOF_MAGICCLOSE

WDIOF_SETTIMEOUT 特性与 WDIOF_KEEPALIVEPING 特性介绍请参见“3.4.3 定时喂狗”和“3.4.4 设置超时时间”，WDIOF_MAGICCLOSE 的特性请参见“3.4.8 Magic close”。

示例：

```
struct watchdog_info ident;
ioctl(fd, WDIOC_GETSUPPORT, &ident);
```

## 3.4.7 使能和禁止 watchdog

应用程序可以禁止 watchdog 运行或重新使能 watchdog。一般情况下不需要这样做，因为如果应用程序在禁止 watchdog 的期间发生意外，整个系统将得不到 watchdog 的监控。

调用 ioctl 并在 cmd 参数中传入命令 WDIOC_SETOPTIONS，arg 参数中传入 WDIOS_ENABLECARD 或 WDIOS_DISABLECARD 将使能或禁止 watchdog 运行。

调用 ioctl 并在 cmd 参数中传入命令 WDIOC_GETSTATUS，arg 参数中传入一个 int 型的指针来获得当前 watchdog 被使能或禁止的状态。

说明

如果 watchdog 被禁止运行，即使应用程序意外崩溃，watchdog 也不会自动恢复运行（不会返回到默认监控模式）。

示例：

```
/* 获取当前watchdog的使能/禁止状态 */
int flags;
ioctl(fd, WDIOC_GETSTATUS, &flags);


/* 设置当前watchdog禁止运行 */
```

```
int options = WDIOS_DISABLECARD;
ioctl(fd, WDIOC_SETOPTIONS, & options);

/* 设置当前watchdog使能运行 */
int options = WDIOS_ENABLECARD;
ioctl(fd, WDIOC_SETOPTIONS, & options);
```

### 3.4.8 Magic close 防止 watchdog 设备文件意外关闭机制

应用程序打开 watchdog 设备文件来接管 watchdog，关闭 watchdog 设备文件时，watchdog 驱动会尝试收回喂狗的权利，以默认监控模式运行。如果应用程序意外崩溃，系统会自动为应用程序打开的所有文件调用 close 操作。因此，为了防止应用程序意外的关闭打开的 watchdog 设备文件，引入 Magic close 机制。

引入 Magic close 机制后，应用程序在正常退出时，必须先对 watchdog 设备文件写入一个"V"字符串，才能正常的关闭 watchdog 设备文件，这时 watchdog 驱动收回喂狗的控制权，进入默认监控模式；否则，如果应用程序意外的关闭了 watchdog 设备文件，watchdog 驱动认为应用程序出现异常，不会收回喂狗的控制权。如果在超时时间内应用程序没有重新接管喂狗的控制权，系统将被复位。

示例：

```
write(fd, "V", 1);
```

### 3.4.9 结束使用 watchdog

应用程序正常结束后，关闭 watchdog 设备文件，watchdog 驱动收回喂狗的控制权，进入默认监控模式。

说明

必须执行完 Magic close 操作后才能正常的关闭 watchdog，否则 watchdog 驱动不会进入默认监控模式，并且 watchdog 驱动模块不能移除。watchdog 超时后系统被复位。

示例：

```
write(fd, "V", 1);
close(fd);
```

## 3.5 应用示例

```
#include <stdio.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <linux/watchdog.h>
#include <stdlib.h>
```

```
int main(int argc, char *argv[])
{
    int i = 0;
    int timeout = 10;
    int fd = open("/dev/misc/watchdog",O_WRONLY);

    if (fd == -1)
    {
        perror("watchdog");
        exit(1);
    }

    ioctl(fd, WDIOC_SETTIMEOUT, &timeout);
    ioctl(fd, WDIOC_GETTIMEOUT, &timeout);
    printf("The timeout was set to %d seconds\n", timeout);

    while(i++ < 10)
    {
        write(fd, "\0", 1);
        sleep(5);
    }

    write(fd, "V", 1);
    close(fd);
    printf("Application exit normal.\n");
    exit(0);
}
```

加载 watchdog 驱动并运行应用程序，50 秒后应用程序正常退出，watchdog 驱动收回喂狗的控制权，进入默认监控模式；如果中间按 Ctrl+C 结束应用程序，应用程序意外终止，watchdog 驱动不会收回控制权，watchdog 超时后系统被复位。