GigaDevice Semiconductor Inc.

Arm® Cortex®-M3/4/23/33 32-bit MCU

应用笔记

AN021

目录

图索引

表索引

1. 简介

单片机在开发过程中会遇到需要测量部分代码或者算法所执行的时间、优化前和优化后代码运行时间的对比以及在代码中所设计的等待周期是否足够精确等问题，本应用手册基于GD32F10x 系列，采用 GD32F103C-EVAL 开发板，开发环境为 KEIL 5.27，提供四种测量代码执行时间的方法：

■使用 TIMER 定时器测量
■使用 Systick 计数器测量
■使用 GPIO 翻转测量
■使用 KEIL 在线仿真测量

2. 测量代码运行时间的实现

2.1. 使用 TIMER 定时器测量

本工程MCU系统时钟工作频率为108Mhz，使用定时器1分频系数为108，计数器值为10000，设置定时器更新中断并使能 TIMER1 更新事件中断，TIMER1 配置如*表 2-1. 定时器 TIMER1 配置*所示：

表 2-1. 定时器 TIMER1 配置

```
/**
    \brief       configure the TIMER peripheral
    \param[in]   none
    \param[out] none
    \retval      none
*/
void timer_config(void)
{
    /* -----------------------------------------------------------------------
    TIMER1 Configuration:
    TIMER1CLK = SystemCoreClock/108 = 1MHz, the period is 1ms (10000/1000000 = 1ms).
    ----------------------------------------------------------------------- */
    timer_parameter_struct timer_initpara;
    rcu_periph_clock_enable(RCU_TIMER1);
    timer_deinit(TIMER1);
    /* initialize TIMER init parameter struct */
    timer_struct_para_init(&timer_initpara);
    /* TIMER1 configuration */
    timer_initpara.prescaler            = 107;
    timer_initpara.alignedmode          = TIMER_COUNTER_EDGE;
    timer_initpara.counterdirection     = TIMER_COUNTER_UP;
    timer_initpara.period               = 9999;
    timer_initpara.clockdivision        = TIMER_CKDIV_DIV1;
    timer_init(TIMER1, &timer_initpara);
    timer_interrupt_flag_clear(TIMER1, TIMER_INT_FLAG_UP);
    timer_interrupt_enable(TIMER1, TIMER_INT_UP);
}
/**
    \brief       configure the TIMER1 interrupt
    \param[in]   none
    \param[out] none
    \retval      none
*/
```

```
void nvic_config(void)
{
    nvic_priority_group_set(NVIC_PRIGROUP_PRE1_SUB3);
    nvic_irq_enable(TIMER1_IRQn, 1, 1);
}
```

定义外部变量 cnt_cycle，当定时器发生 1ms 事件更新时，在更新中断中让 cnt_cycle 加 1。

表 2-2. 定时器 TIMER1 中断服务函数

```
extern uint32_t cnt_cycle;
/*!
    \brief      this function handles TIMER1 interrupt request.
    \param[in]  none
    \param[out] none
    \retval     none
*/
void TIMER1_IRQHandler(void)
{
    if(SET == timer_interrupt_flag_get(TIMER1, TIMER_INT_FLAG_UP)){
        /* clear the update event interrupt flag bit */
        timer_interrupt_flag_clear(TIMER1, TIMER_INT_FLAG_UP);
        ++cnt_cycle;
    }
}
```

编写要测量的代码段，本工程中通过在函数 delay_1us 编写 108 个__nop()，即代码运行 108 个时钟周期为 1us。代码编写如*表 2-3. 测试代码函数*所示：

表 2-3. 测试代码函数

```
/*!
    \brief      delay 1us
    \param[in]  none
    \param[out] none
    \retval     none
*/
#define delay_1us()\
do{\
    __nop(); __nop(); __nop(); __nop(); __nop(); __nop(); __nop(); __nop(); __nop();
__nop(); \
    __nop(); __nop(); __nop(); __nop(); __nop(); __nop(); __nop(); __nop(); __nop();
__nop();  \
    __nop(); __nop(); __nop(); __nop(); __nop(); __nop(); __nop(); __nop(); __nop();
__nop();  \
    __nop(); __nop(); __nop(); __nop(); __nop(); __nop(); __nop(); __nop(); __nop();
__nop();  \
    __nop(); __nop(); __nop(); __nop(); __nop(); __nop(); __nop(); __nop(); __nop();
__nop();   \
        __nop(); __nop(); __nop(); __nop(); __nop(); __nop(); __nop(); __nop(); __nop(); 
__nop();   \
        __nop(); __nop(); __nop(); __nop(); __nop(); __nop(); __nop(); __nop(); __nop(); 
__nop();   \
        __nop(); __nop(); __nop(); __nop(); __nop(); __nop(); __nop(); __nop(); __nop(); 
__nop();   \
        __nop(); __nop(); __nop(); __nop(); __nop(); __nop(); __nop(); __nop(); __nop(); 
__nop();   \
        __nop(); __nop(); __nop(); __nop(); __nop(); __nop(); __nop(); __nop(); __nop(); 
__nop();   \
        __nop(); __nop(); __nop(); __nop(); __nop(); __nop(); __nop(); __nop(); \
}while(0)

/*!
        \brief          delay nms
        \param[in]    nms: delay nms ms
        \param[out] none
        \retval         none
*/
void delay_nms(uint32_t nms)
{
        uint32_t nus=nms*1000;
        while(--nus)
        {
                delay_1us();
        }
}
```

编写计时开始函数 measure_runtime_start 和计时结束函数 measure_runtime_end，如*表 2-4. 定时器测量启动和结束函数*所示：

表 2-4. 定时器测量启动和结束函数

```
/*!
        \brief          start measure
        \param[in]    none
        \param[out] none
        \retval         none
*/
void measure_runtime_start()
{
        cnt_cycle = 0;
        timer_enable(TIMER1);
}
```

```
/*!
    \brief      end measurement
    \param[in]  none
    \param[out] none
    \retval     measure time
*/
float measure_runtime_end()
{
    float work_time;
    uint32_t cnt_num,tmp;
    cnt_num = TIMER_CNT(TIMER1);
    timer_disable(TIMER1);
    nvic_irq_disable(TIMER1_IRQn);
    tmp = cnt_cycle*10000 + cnt_num;
    work_time = (float)tmp/1000.0;
    return work_time;
}
```

在主函数中编写代码测量待执行的代码段时间，并通过串口打印，代码如*表 2-5. 定时器测量代码运行时间主函数*所示：

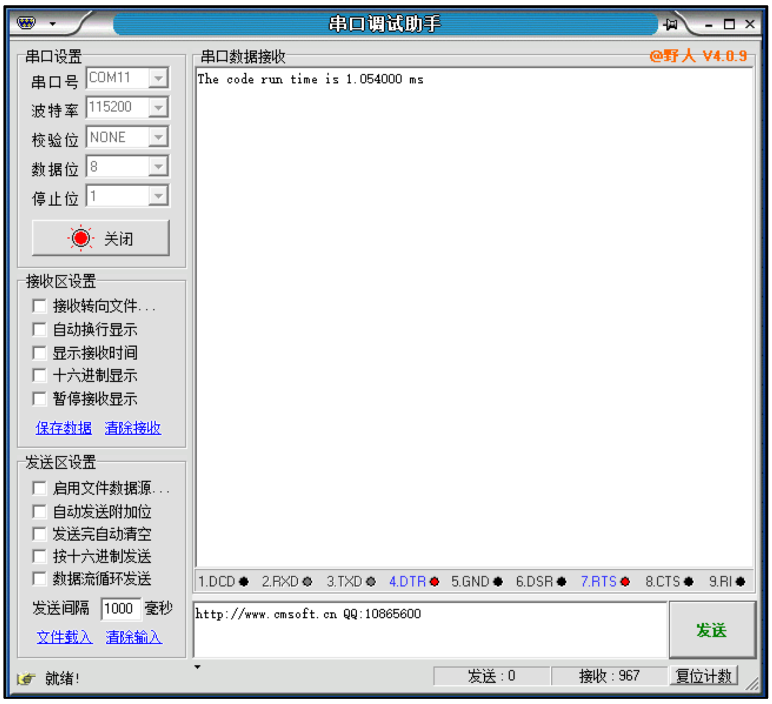
表 2-5. 定时器测量代码运行时间主函数

```
/*!
    \brief      main function
    \param[in]  none
    \param[out] none
    \retval     none
*/
int main(void)
{
    gd_eval_com_init(EVAL_COM0);
    dbg_periph_enable(DBG_TIMER1_HOLD);
    timer_config();
    nvic_config();
    measure_runtime_start();
    delay_nms(1);
    rtimevla = measure_runtime_end();
    printf("The code run time is %f ms",rtimevla);
    while(1){

    }
}
```

测量结果通过上位机打印结果如*图 2-1. 定时器 Timer1 测量代码运行时间结果打印*所示：

图 2-1. 定时器 Timer1 测量代码运行时间结果打印



串口号 COM11
波特率 115200
校验位 NONE
数据位 8
停止位 1
发送间隔 1000 毫秒

```
The code run time is 1.054000 ms
```

```
http://www.cmsoft.cn QQ:10865600
```

发送：0 接收：967

注意：代码在执行过程中跳转会造成延迟误差，可以通过减少__nop()个数，来调整自定义延迟 1ms 函数的精确度。

2.2. 使用 Systick 计数器测量

SysTick 是一个 24 位的倒计数定时器，当计到 0 时，将从 RELOAD 寄存器中自动重装载定时初值，本节 Systick 采用系统时钟时钟输入，配置 Systick 中断每 1ms 进入一次中断，让变量 tick 加 1，编写函数 start_time、stop_time 和 get_time 分别代表代码计时开始，代码计时结束和获取代码运行时间。函数实现如***表 2-6. 系统定时器 Systick 配置***所示：

表 2-6. 系统定时器 Systick 配置

```
uint32_t tick;
/*!
    \brief          strat time
    \param[in]      none
    \param[out] none
    \retval         none
*/
void start_time(void) {
    tick = 0;
    /* setup systick timer for 1000Hz interrupts */
    if (SysTick_Config(SystemCoreClock / 1000)){
        /* capture error */
        while (1){
        }
    }
    /* configure the systick handler priority */
    NVIC_SetPriority(SysTick_IRQn, 0x00U);
}

/*!
    \brief      stop time
    \param[in]  none
    \param[out] none
    \retval     none
*/
void stop_time(void) {
    SysTick->CTRL &= SysTick_Counter_Disable;
    SysTick->VAL = SysTick_Counter_Clear;
}

/*!
    \brief      get time
    \param[in]  none
    \param[out] none
    \retval     none
*/
uint32_t get_time(void) {
    uint32_t elapsed = (uint32_t)tick;
    return elapsed;
}
/*!
    \brief      this function handles SysTick exception
    \param[in]  none
    \param[out] none
    \retval     none
*/
void SysTick_Handler(void)
{
    ++tick;
}
```

在 main 函数中代码如*表 2-7. Systick 测量代码运行时间主函数*所示：

表 2-7. Systick 测量代码运行时间主函数

```
/*!
    \brief      main function
    \param[in]  none
    \param[out] none
    \retval     none
```

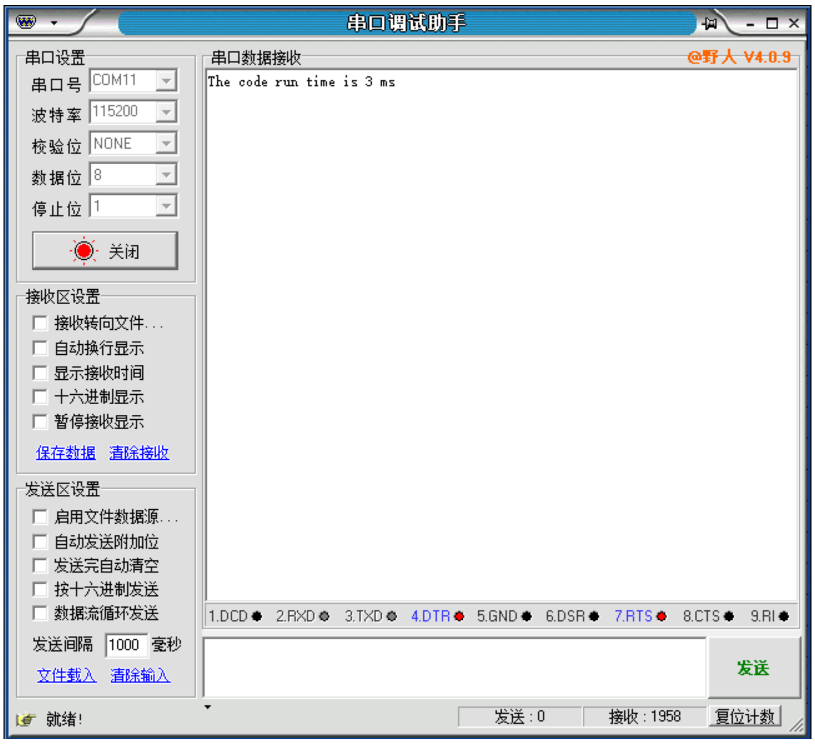
```
*/
int main(void)
{
    uint32_t value;
    gd_eval_com_init(EVAL_COM0);

    start_time();
    delay_nms(3);
    stop_time();
    value=get_time();
    printf("The code run time is %d ms",value);
    while(1){

    }
}
```

代码运行时间测量结果如*图 2-2. Systick 测量代码运行时间结果打印*所示：

图 2-2. Systick 测量代码运行时间结果打印



串口号 COM11
波特率 115200
校验位 NONE
数据位 8
停止位 1
发送间隔 1000 毫秒
发送 : 0
接收 : 1958

```
The code run time is 3 ms
```

注意：可以通过改变 SysTick_Config()函数传参来改变 Systick 定时器进入中断的基准时间。

2.3. 使用 GPIO 翻转测量

配置 GPIO PA1 为推挽输出模式，默认输出低电平，在待测量的代码段前拉高 PA1 引脚，在代码运行完成后拉高电平，使用逻辑分析仪或者示波器触发模式，测试代码运行时间。代码如*表 2-8. GPIO 翻转配置及主函数*所示：

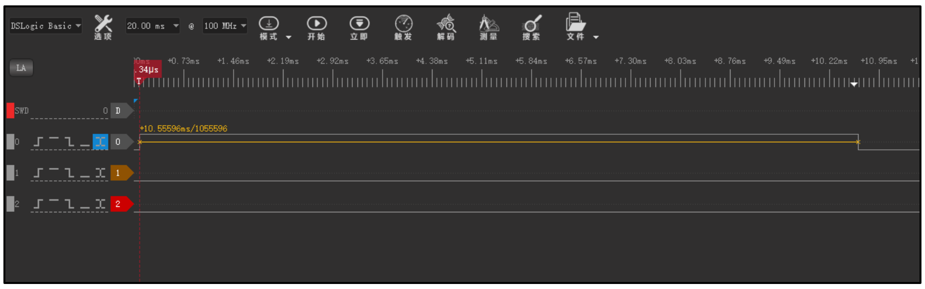
表 2-8. GPIO 翻转配置及主函数

```
/*!
    \brief      main function
    \param[in]  none
    \param[out] none
    \retval     none
*/
int main(void)
{
    rcu_periph_clock_enable(RCU_GPIOA);
    gpio_bit_reset(GPIOA,GPIO_PIN_6);
     gpio_init(GPIOA,GPIO_MODE_OUT_PP,GPIO_OSPEED_50MHZ,GPIO_PIN_6);
    gpio_bit_set(GPIOA,GPIO_PIN_6);
    delay_nms(10);
    gpio_bit_reset(GPIOA,GPIO_PIN_6);
    while(1){

    }
}
```

使用逻辑分析仪选择边沿触发，测量结果如*图2-3. 使用逻辑分析仪测量GPIO 翻转时间*所示：

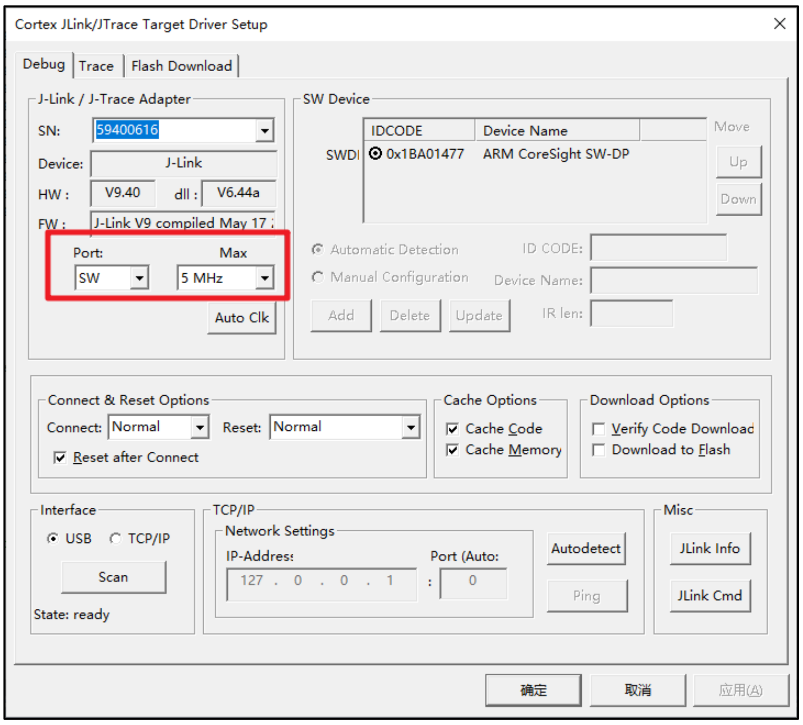
图 2-3. 使用逻辑分析仪测量 GPIO 翻转时间



2.4. 使用 KEIL 在线仿真测量

使用 Jlink 连接目标板，在 Debug->setting 选项卡中选择 SW Port，在 Trace 选项卡中 Enable Trace Setting，配置 Core clock 为系统时钟，工程使用 108Mhz,点击确定。配置如*图2-4. 配置 SW 口下载模式*所示：

图 2-4. 配置 SW 口下载模式



Cortex JLink/JTrace Target Driver Setup

Debug | Trace | Flash Download

J-Link / J-Trace Adapter
SN: 59400616
Device: J-Link
HW: V9.40 dll: V6.44a
FW: J-Link V9 compiled May 17
Port: SW Max: 5 MHz
Auto Clk

SW Device
	IDCODE	Device Name
SWDI	⊙ 0x1BA01477	ARM CoreSight SW-DP

Move Up Down
Automatic Detection ID CODE:
Manual Configuration Device Name:
Add Delete Update IR len:

Connect & Reset Options
Connect: Normal Reset: Normal
☑ Reset after Connect

Cache Options
☑ Cache Code
☑ Cache Memory

Download Options
☐ Verify Code Download
☐ Download to Flash

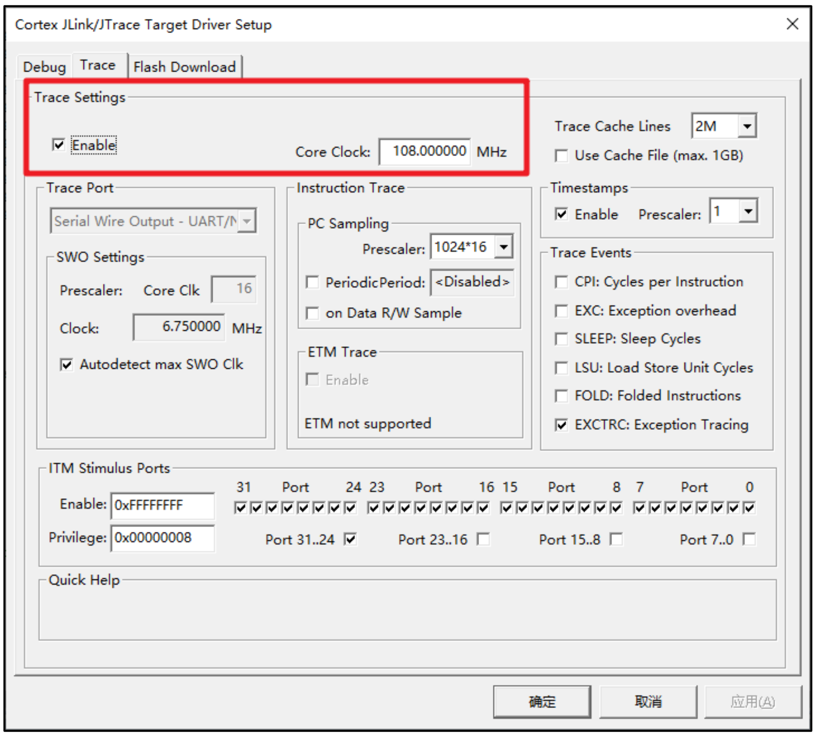
Interface
● USB ○ TCP/IP
Scan
State: ready

TCP/IP
Network Settings
IP-Address: 127 . 0 . 0 . 1 Port (Auto: 0
Autodetect
Ping

Misc
JLink Info
JLink Cmd

确定 取消 应用(A)

图 2-5. 配置 Trace 界面



Cortex JLink/JTrace Target Driver Setup

Debug | Trace | Flash Download

Trace Settings
☑ Enable Core Clock: 108.000000 MHz

Trace Cache Lines 2M
☐ Use Cache File (max. 1GB)

Trace Port
Serial Wire Output - UART/NRZ
SWO Settings
Prescaler: Core Clk 16
Clock: 6.750000 MHz
☑ Autodetect max SWO Clk

Instruction Trace
PC Sampling
Prescaler: 1024*16
☐ PeriodicPeriod: <Disabled>
☐ on Data R/W Sample
ETM Trace
☐ Enable
ETM not supported

Timestamps
☑ Enable Prescaler: 1

Trace Events
☐ CPI: Cycles per Instruction
☐ EXC: Exception overhead
☐ SLEEP: Sleep Cycles
☐ LSU: Load Store Unit Cycles
☐ FOLD: Folded Instructions
☑ EXCTRC: Exception Tracing

ITM Stimulus Ports
Enable: 0xFFFFFFFF
Privilege: 0x00000008
Port 31..24 ☑ Port 23..16 ☐ Port 15..8 ☐ Port 7..0 ☐

Quick Help

确定 取消 应用(A)

进入调试界面，在要测量代码前后分别加入断点，运行到断点处，在右下角选择 Reset Stop Watch(t1)，清零 t1 运行时间，全速运行，运行到代码结束处，查看代码运行时间，代码运行时间结果如*图 2-7. 代码运行时间测量*所示：

图 2-6. 复位运行时间

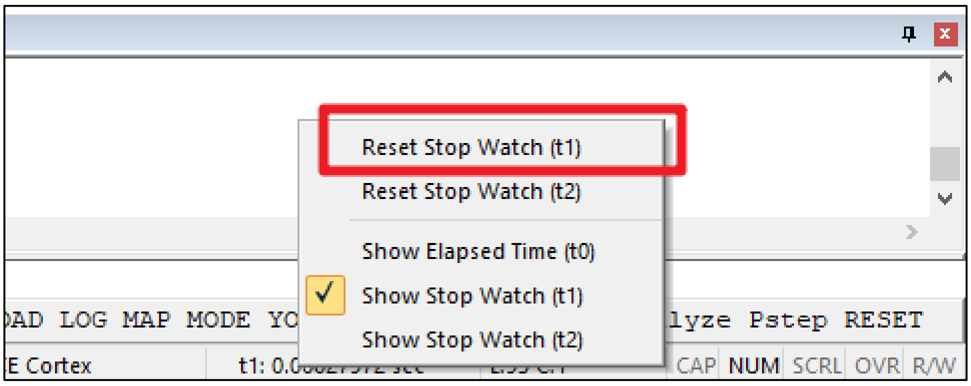
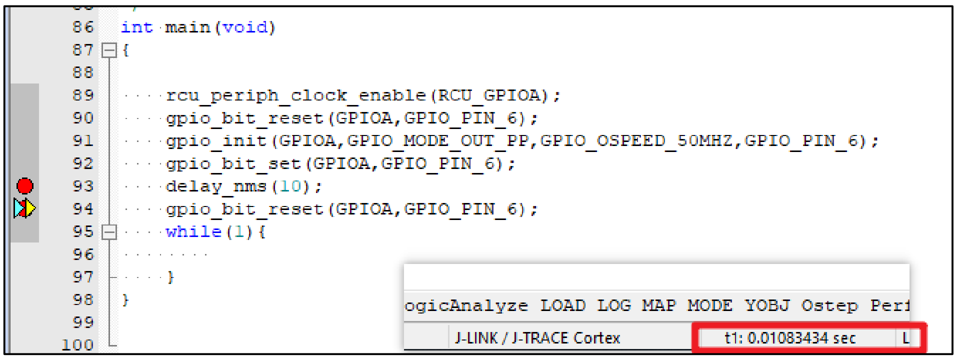


图 2-7. 代码运行时间测量



```
int main(void)
{

    rcu_periph_clock_enable(RCU_GPIOA);
    gpio_bit_reset(GPIOA,GPIO_PIN_6);
    gpio_init(GPIOA,GPIO_MODE_OUT_PP,GPIO_OSPEED_50MHZ,GPIO_PIN_6);
    gpio_bit_set(GPIOA,GPIO_PIN_6);
    delay_nms(10);
    gpio_bit_reset(GPIOA,GPIO_PIN_6);
    while(1){

    }
}
```

3. 版本历史

表 3-1. 版本历史

版本号.	说明	日期
1.0	首次发布	2021 年 4 月 30 日