GigaDevice Semiconductor Inc.

Arm® Cortex®-M3/4/23/33 32-bit MCU

Application Note
AN021

Table of Contents

List of Figures

List of Tables

1. Introduction

In the process of MCU development, there are some problems, such as measuring the execution time of some codes or algorithms, comparing the running time of codes before and after optimization, and whether the waiting period designed in the code is accurate enough. This application note is based on gd32f10x series, using gd32f103c-eval development board, and the development environment is keil 5.27, four methods to measure code execution time are provided.

- Measure with timer
- Measure with systick counter
- Measure with gpio flip
- Measure with Keil online simulation

2. Implementation of measuring code running time

2.1. Measurement with Timer

The working frequency of MCU system clock is 108MHz, the division factor of Timer1 is 108, and the counter value is 10000. Set timer update interrupt and enable Timer1 update event interrupt. The configuration of Timer1 is shown in ***Table 2-1. Timer1 configuration parameters***.

Table 2-1. Timer1 configuration parameters

```
/**
    \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);
}
```

Define the external variable cnt_ cycle. When the timer is updated with 1ms event, cnt_ cycle plus one in the update interrupt.

Table 2-2. Timer1 interrupt handle function

```
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;
    }

}
```

Write the code segment to be measured. In this project, 108 __nop () is written in the function delay_1us, that is, the code runs for 108 clock cycles for 1us. The code is shown in ***Table 2-3. Test code function***.

Table 2-3. Test code function

```
/*!
    \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();
    }

}
```

Time start function measure_ runtime_ start and measure_ runtime_end is shown in ***Table 2-4. Timer measures start and end functions***.

Table 2-4. Timer measures start and end functions

```
/*!
    \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;

}
```

Write code in the main function to measure the time of the code to be executed, and print it through the serial port. The code is shown in ***Table 2-5. Timer measures start and end functions***.

Table 2-5. Timer measures start and end functions

```
/*!
        \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){

        }
}
```

The measurement results are printed by the upper computer, and the results are shown in ***Figure 2-1. Timer 1 measures the running time of the code and prints the result***.

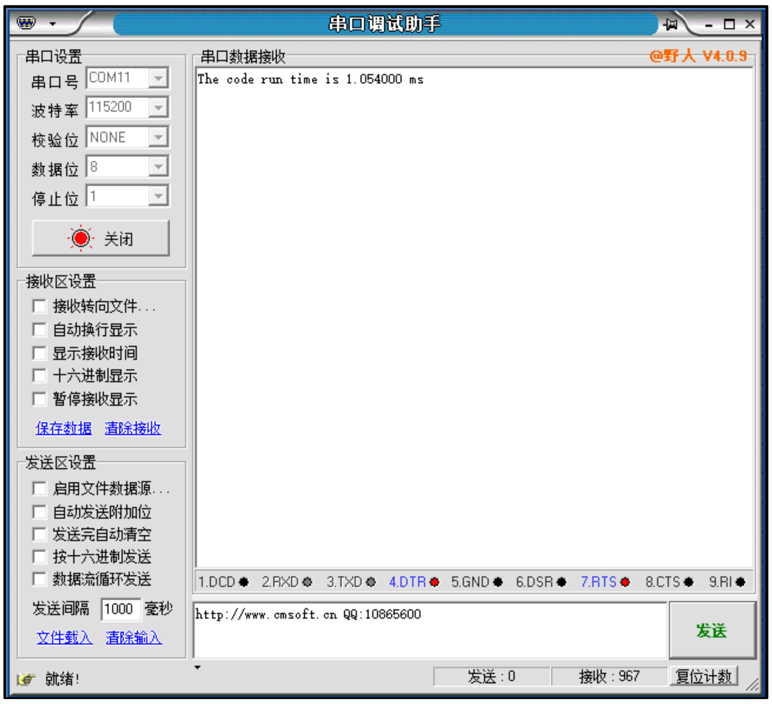


Figure 2-1. Timer 1 measures the running time of the code and prints the result

串口调试助手 @野人 V4.0.9

串口设置
- 串口号: COM11
- 波特率: 115200
- 校验位: NONE
- 数据位: 8
- 停止位: 1

串口数据接收

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

发送区:

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

发送间隔 1000 毫秒

就绪! 发送 : 0 接收 : 967

2.2. Measurement with systick counter

SysTick is a 24-bit countdown timer. When it counts to 0, it will automatically reload the initial value of the timing from the RELOAD register. In this section, systick uses the system clock as the clock input, and configures the systick interrupt to enter an interrupt every 1ms. Write the functions start_time, stop_time, and get_time to represent the start of the code timing, the end of the code timing and get the code running time. The function is shown in ***Table 2-6. Systick configuration function***.

Table 2-6. Systick configuration function

```
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;
}
```

The code in the main function is shown in ***Table 2-7. The main function of systick measuring code running time***.

Table 2-7. The main function of systick measuring code running time

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

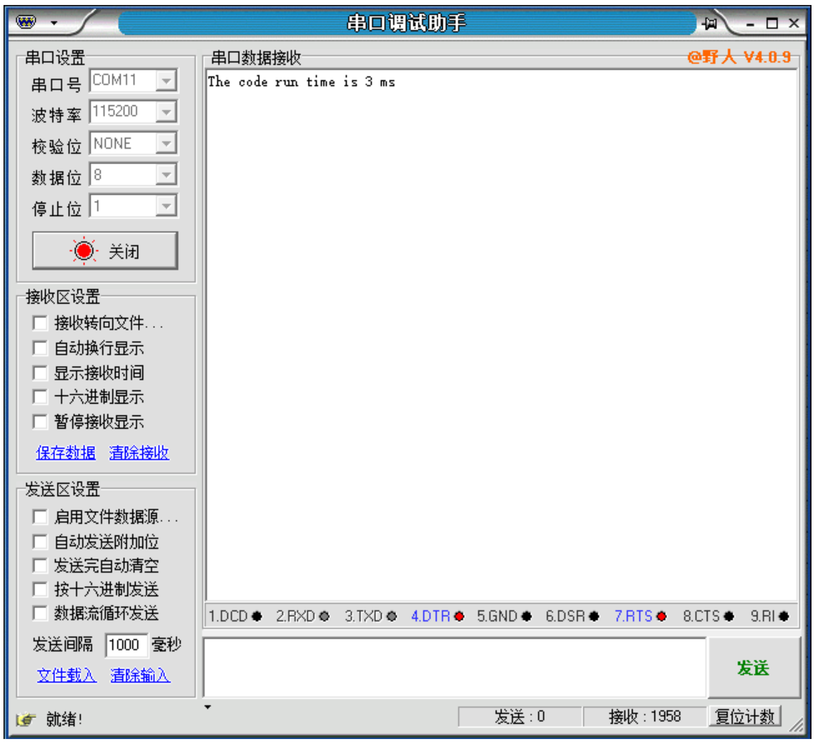
```
        \param[out] none
        \retval         none
*/
int main(void)
{
        uint32_t value;
        gd_eval_com_init(EVAL_COM0);

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

        }
}
```

The code running time measurement results are as follows:

Figure 2-2. Systick measurement code running time result printing



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
```

Note: By changing the parameter of the SysTick_Config() function to change the base time of the systick timer entering the interrupt.

2.3. Measure with gpio flip

Configure GPIO PA1 to push-pull output mode, the default output is low level, configure the PA1 pin is pulled high before the code segment to be measured, and the level is pulled high after the code is run. Use the logic analyzer or oscilloscope trigger mode to test the code running time. The code is shown in ***Table 2-8. GPIO flip configuration and main function***.

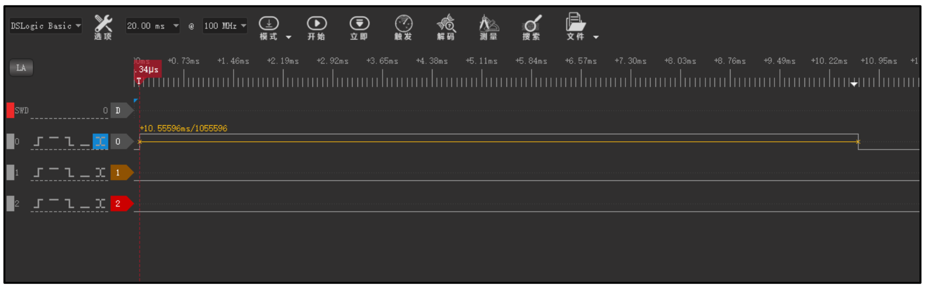
Table 2-8. GPIO flip configuration and main function

```
/*!
      \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){

      }
}
```

Use the logic analyzer to select the edge trigger, and the measurement results are as follows:

Figure 2-3. Use logic analyzer to measure GPIO flip time result

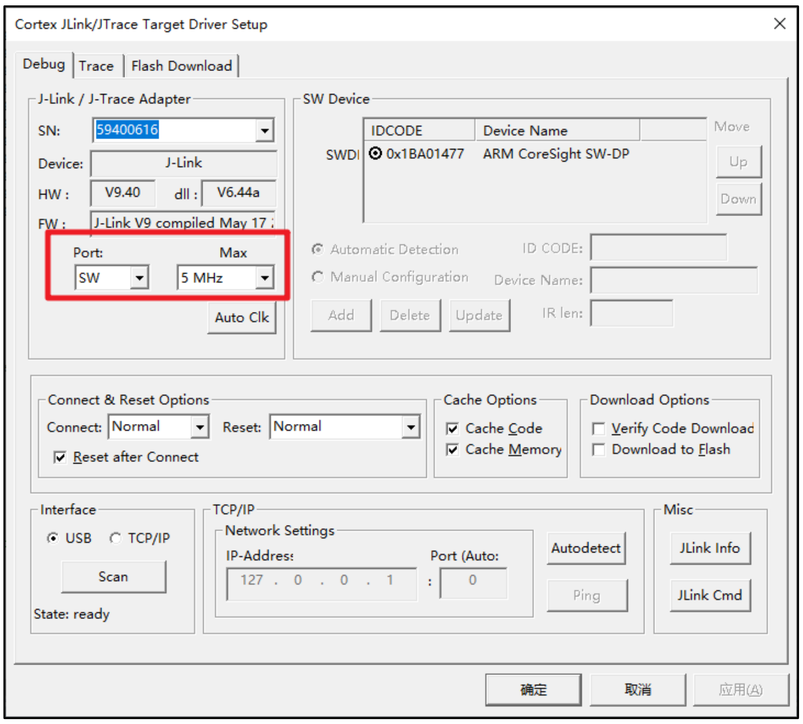


+10.55596ms/1055596

2.4. Measure with Keil online simulation

Use Jlink to connect to the target board, select SW Port in the Debug->setting tab, Enable Trace Setting in the Trace tab, configure Core clock as the system clock 108Mhz, click "OK". The configuration is shown in ***Figure 2-4. Configure SW port download mode***.

Figure 2-4. Configure SW port download mode



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
SWDI ⦿ IDCODE: 0x1BA01477 Device Name: 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

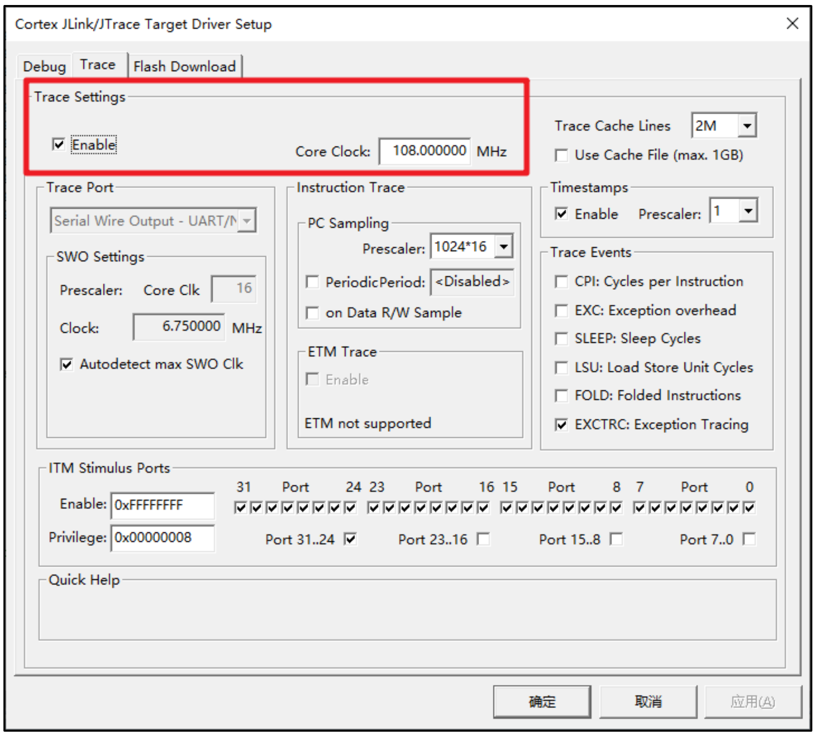
Interface
◉ USB ○ TCP/IP
Scan
State: ready

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)

Figure 2-5. Configure Trace interface



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/N
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 (Port 31..0 all ☑)
Privilege: 0x00000008
Port 31..24 ☑ Port 23..16 ☐ Port 15..8 ☐ Port 7..0 ☐

Quick Help

确定 取消 应用(A)

Enter the debugging interface, add breakpoints before and after the code to be measured, run to the breakpoint, select Reset Stop Watch(t1) in the lower right corner, clear the t1 running time, run at full speed, run to the end of the code, and view the code running time , The code running time result is shown in ***Figure 2-7. Code runtime measurement***:

Figure 2-6. Reset running time

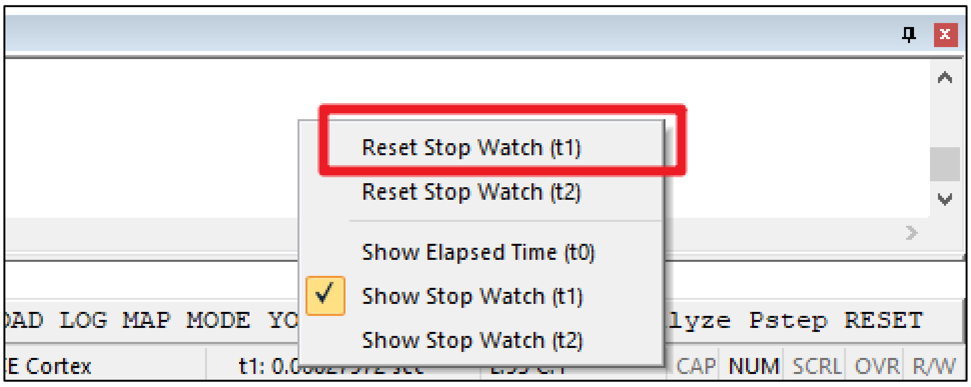
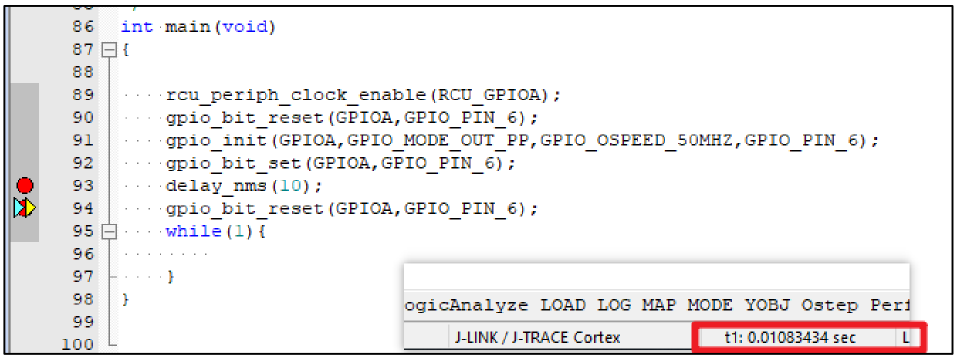


Figure 2-7. Code runtime measurement



```
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. Revision history

Table 3-1. Revision history

Revision No	Description	Date
1.0	Initial Release	Apr.30, 2021