# GPIO 软件防抖

NuMicro® 32 位系列微控制器范例代码介绍

## 文件信息

| | |
|---|---|
| 代码简述 | 使用定时器周期模式，每 1 毫秒产生一次定时器中断，来完成软件按键防抖 |
| BSP 版本 | M031_Series_BSP_CMSIS_V3.02.000 |
| 开发平台 | NuMaker-M032SE V1.3 |

# 1 功能介绍

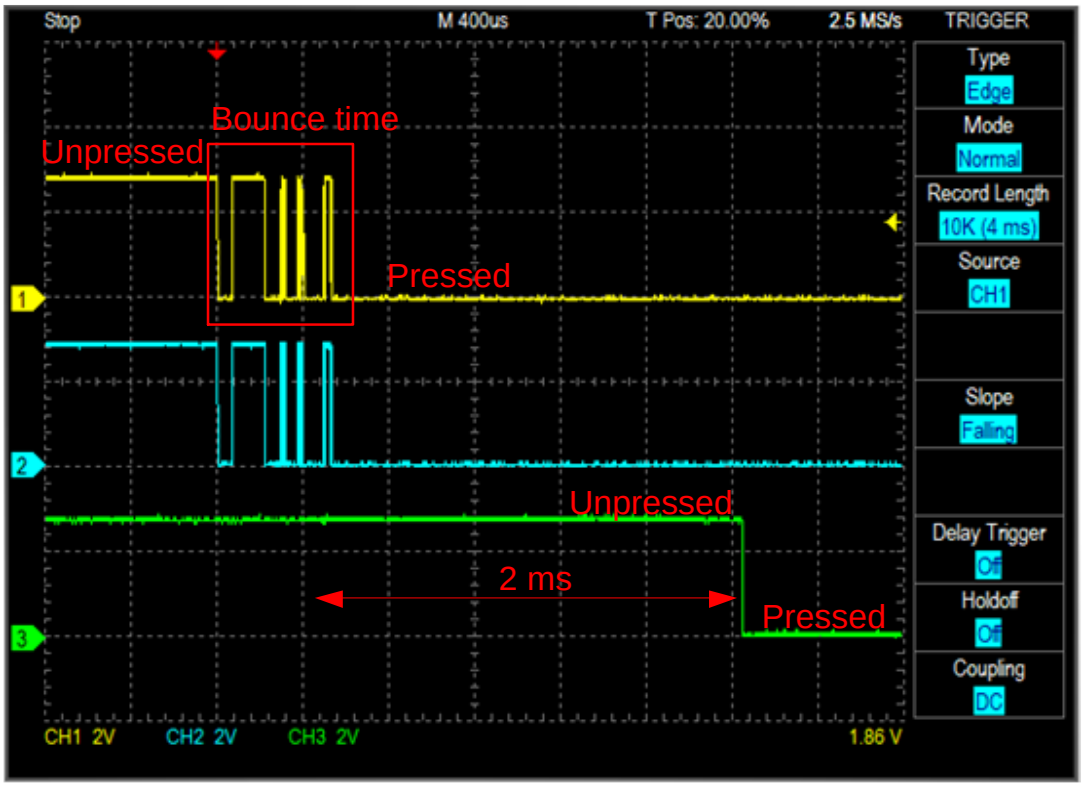
## 1.1 简介

弹跳是由电子设备中触点的闭合或断开产生的多个脉冲信号。 最常见的是机械按钮，当按下按钮一次时，会产生多个脉冲信号， MCU会认为是多次按下并放开按键，而导致误动作，因此需要防抖，以确保获得正确的信号。 本文主要使用定时器和GPIO进行软件防抖实验。

## 1.2 原理

此软件防抖是使用Timer每1ms产生一个中断并将1ms旗标设置为1，然后根据此旗标每1ms确认按钮的状态。 在按钮状态一段时间未改变之后，即可以确定按钮已被稳定地按下或放开。 然后MCU可以将其判断为正确的信号，并将该信号输出到LED。 另外，可以根据不同按钮的稳定时间设置防抖时间。

## 1.3 执行结果




**CH1:** 击键 (PC.14),
**CH2:** 没有防抖的讯号输出 (PB.15),
**CH3:** LED, 有防抖的讯号输出(PB.14)

注: 这个波型的防抖时间设为 2 ms.

# 2 代码介绍

```
/*---------------------------------------------------------------------------*/
/* Define                                                                    */
/*---------------------------------------------------------------------------*/
#define DEBOUNCE_TIME 50            //Press and release button settling time ( ms )
#define BUTTON1 PC14                //Button input port
#define LED PB14                    //Indicator light used to indicate that a button is
                                      pressed(debounced)
#define TEST_PIN PB15               //Used to measure button state without debouncing

/*---------------------------------------------------------------------------*/
/* Global variables                                                          */
/*---------------------------------------------------------------------------*/
uint8_t bTimer_1ms;                 //Timer 1ms flag
uint8_t g_ButtonState;              //BUTTON1's current state
uint8_t g_ButtonStateOld;           //BUTTON1's previous state
uint8_t g_ButtonStateDebounced = 1; //BUTTON1's de-bounced state
uint32_t g_ButtonDebounceCnt = 0;

/*---------------------------------------------------------------------------*/
/* Functions                                                                 */
/*---------------------------------------------------------------------------*/
uint8_t GPIO_Debounce(void);

void TMR0_IRQHandler(void)
{
    /* Set 1ms flag */
    bTimer_1ms = 1;
    /* clear timer interrupt flag */
    TIMER_ClearIntFlag(TIMER0);
}

void SYS_Init(void)
{
    /* Unlock protected registers */
    SYS_UnlockReg();

    /* Enable HIRC */
    CLK_EnableXtalRC(CLK_PWRCTL_HIRCEN_Msk);
```

```
    /* Waiting for HIRC clock ready */
    CLK_WaitClockReady(CLK_STATUS_HIRCSTB_Msk);

    /* Switch HCLK clock source to HIRC */
    CLK_SetHCLK(CLK_CLKSEL0_HCLKSEL_HIRC, CLK_CLKDIV0_HCLK(1));

    /* Set both PCLK0 and PCLK1 as HCLK/2 */
    CLK->PCLKDIV = (CLK_PCLKDIV_APB0DIV_DIV2 | CLK_PCLKDIV_APB1DIV_DIV2);

    /* Switch UART0 clock source to HIRC */
    CLK_SetModuleClock(UART0_MODULE, CLK_CLKSEL1_UART0SEL_HIRC, CLK_CLKDIV0_UART0(1));

    /* Enable UART peripheral clock */
    CLK_EnableModuleClock(UART0_MODULE);

    /* Enable IP clock */
    CLK_EnableModuleClock(TMR0_MODULE);

    /* Select IP clock source */
    CLK_SetModuleClock(TMR0_MODULE, CLK_CLKSEL1_TMR0SEL_HIRC, 0);

    /* Update System Core Clock */
    /* User can use SystemCoreClockUpdate() to calculate PllClock, SystemCoreClock and
       CycylesPerUs automatically. */
    SystemCoreClockUpdate();

    /*---------------------------------------------------------------------*/
    /* Init I/O Multi-function                                             */
    /*---------------------------------------------------------------------*/
    /* Set GPB multi-function pins for UART0 RXD and TXD */
    SYS->GPB_MFPH = (SYS->GPB_MFPH & ~(SYS_GPB_MFPH_PB12MFP_Msk |
                     SYS_GPB_MFPH_PB13MFP_Msk)) |
                    (SYS_GPB_MFPH_PB12MFP_UART0_RXD | SYS_GPB_MFPH_PB13MFP_UART0_TXD);

    /* Lock protected registers */
    SYS_LockReg();
}
```

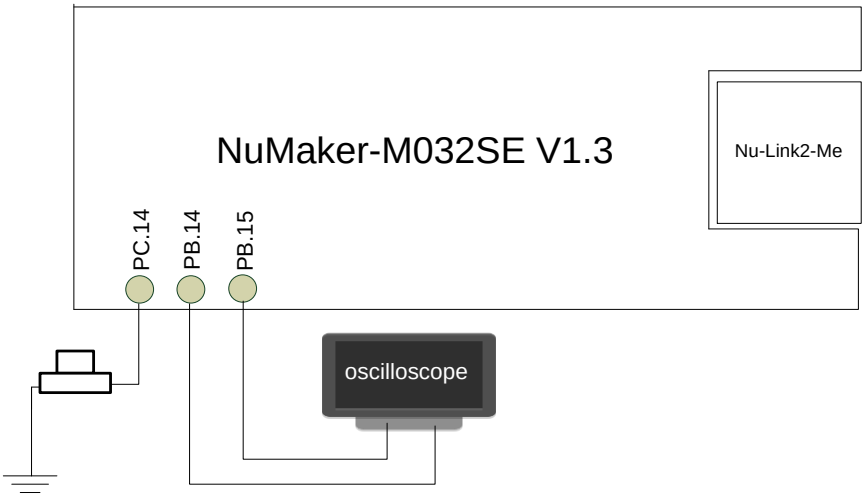
```
/*---------------------------------------------------------------------*/
/* Init UART0                                                          */
/*---------------------------------------------------------------------*/
void UART0_Init(void)
{
    /* Reset UART0 */
    SYS_ResetModule(UART0_RST);

    /* Configure UART0 and set UART0 baud rate */
    UART_Open(UART0, 115200);
}

int main(void)
{
    /* Init System, IP clock and multi-function I/O. */
    SYS_Init();

    /* Init UART0 for printf */
    UART0_Init();

    printf("\nThis sample code use timer to generate interrupt every 1 ms for software
              debouncing test \n");

    /* Set timer frequency to 1000 Hz */
    TIMER_Open(TIMER0, TIMER_PERIODIC_MODE, 1000);

    /* Enable timer interrupt */
    TIMER_EnableInt(TIMER0);
    NVIC_EnableIRQ(TMR0_IRQn);

    /* Start Timer 0 */
    TIMER_Start(TIMER0);

    /* Set PC4 to QUASI mode for BUTTON1 debouncing test*/
    GPIO_SetMode(PC, BIT14, GPIO_MODE_QUASI);

    /* Set PB14 and PB15 to output mode for BUTTON1's debouncing test*/
    GPIO_SetMode(PB, BIT14, GPIO_MODE_OUTPUT);//LED
    GPIO_SetMode(PB, BIT15, GPIO_MODE_OUTPUT);//TEST_PIN



    while (1)
```

```
    {
        /* Output BUTTON1 state to TESE_PIN */
        TEST_PIN = BUTTON1;

        /* Check the BUTTON1 state every 1ms */
        if (bTimer_1ms)
        {
            /* Clear 1ms flag */
            bTimer_1ms = 0;
            /* Output debounced BUTTON1 state to LED */
            LED = GPIO_Debounce();
        }
    }
}


uint8_t GPIO_Debounce(void)
{
    /* Read the BUTTON1 state */
    g_ButtonState = BUTTON1;

    /* Increase g_ButtonState if the previous state is the same as the current state */
    if (g_ButtonStateOld == g_ButtonState)
    {
        g_ButtonDebounceCnt++;

        /*The state is held longer than the button settling time,
          so this state can be treated as the correct state of the button. */
        if (g_ButtonDebounceCnt >= DEBOUNCE_TIME)
        {
            g_ButtonStateDebounced = g_ButtonState;
        }

        /* Clear the count to zero if the current state of the button is different from
           the previous state */
    }
    else
    {
        g_ButtonDebounceCnt = 0;
    }

    g_ButtonStateOld = g_ButtonState;
```

```
    /* Return the correct state of the button */
    return  g_ButtonStateDebounced;
}
```

# 3 软件与硬件环境

- 软件环境
  - BSP 版本
    - M031_Series_BSP_CMSIS_V3.02.000
  - IDE 版本
    - Keil uVersion 5.28
- 硬件环境
  - 电路组件
    - NuMaker-M032SE V1.3
  - 示意图



# 4 目录信息

- EC_M031_GPIO_Software_Debounce_V1.00
  - Library — Sample code header and source files
    - CMSIS — Cortex® Microcontroller Software Interface Standard (CMSIS) by Arm® Corp.
    - Device — CMSIS compliant device header file
    - StdDriver — All peripheral driver header and source files
  - SampleCode
    - ExampleCode — Source file of example code

# 5 如何执行范例程序

1. 根据目录信息章节进入 ExampleCode 路径中的 KEIL 文件夹，双击 EC_M031_GPIO_Software_Debounce.uvproj。
2. 进入编译模式接口
   a. 编译
   b. 下载代码至内存
   c. 进入 / 离开除错模式
3. 进入除错模式接口
   a. 执行代码

# 6 修订纪录

| Date | Revision | Description |
|---|---|---|
| Nov. 01, 2019 | 1.00 | 1. 初始发布. |

## Important Notice

**Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".**

**Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.**

**All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.**