摘 要

本文档介绍和说明芯海科技旗下 CS32 系列 MCU 的自动波特率检测功能，并为不具备硬件自动波特率检测的 MCU 产品提供替代软件方法 。

本文档适用产品型号说明

类型	产品系列
MCU	CS32F0 系列，CS32F1 系列

版 本

历史版本	修改内容	日期
V1.0	初版生成	2022-10-9

目 录

1. 硬件自动波特率检测

硬件自动波特率检测（ABR）使 USART（通用同步异步收发器）可以根据接收的数据帧，自动检测通讯的波特率，使 USART 可以和外部设备正常通讯，无需事先建立数据速率。

芯海科技通用 MCU CS32 系列产品中，部分 USART 能够支持硬件自动波特率检测功能。如下表 1-1，说明 CS32 系列产品硬件自动波特率检测方式。

表 1　CS32 系列产品 USART 硬件自动波特率检测方式

产品	支持 ABR
CS32F0	是
CS32F1	否

1.1. 自动波特率检测模式

自动波特率检测（ABR）是指接收设备通过第一个特定的字符来确定通讯速率的过程。CS32 系列产品的自动波特率检测功能有内置不同的字符模式：

模式 0：字符以“1”开始。测量起始位下降沿到上升沿的持续时间。

模式 1：字符以 10xx 开始。USART 测量起始位下降沿到第一个数据位下降沿的持续时间。

在使能自动波特率检测之前，必须配置 USARTx_CTR2 寄存器 ABRSEL[1:0]位选择自动波特率检测的模式。

1.2. ABR 误差计算

该功能下，USART 的时钟源（fCK）决定通讯速率的范围。CS32 系列产品在该功能下仅支持 16 倍过采样。USART 的波特率介于 fCK/65535 与 fCK/8 之间。

波特率的误差取决于 USART 时钟源、过采样方法和 ABR 模式。

$$误差（\%）= \left| \frac{预期波特率 - 实际波特率}{预期波特率} \right|$$

注：

- 预期波特率取决于发送设备。
- 实际波特率是 USART 接收端使用自动波特率检测操作确定的波特率。

1.3. 硬件配置示例

以下示例基于 MCU 型号为 CS32F030C8T6，外设采用 USART1。配置流程如下：

初始化 USART1，并选择 ABR 模式。

```
    // 1- Configure the UART peripheral
    usart_config_t usart_config_struct = {0};

    // Clock Config
    __RCU_AHB_CLK_ENABLE(RCU_AHB_PERI_GPIOA);
    __RCU_APB2_CLK_ENABLE(RCU_APB2_PERI_USART1)

    // GPIO MF Config
    gpio_mf_config(GPIOA, GPIO_PIN_9, GPIO_MF_SEL1);
    gpio_mf_config(GPIOA, GPIO_PIN_10, GPIO_MF_SEL1);

    gpio_mode_set(GPIOA, GPIO_PIN_9|GPIO_PIN_10, GPIO_MODE_MF_PP(GPIO_SPEED_MEDIUM));

    // USART Config
    __USART_DEF_INIT(USART1);
    usart_config_struct.baud_rate = 115200;
    usart_config_struct.data_width = USART_DATA_WIDTH_8;
    usart_config_struct.stop_bits = USART_STOP_BIT_1;
    usart_config_struct.parity = USART_PARITY_NO;
    usart_config_struct.flow_control = USART_FLOW_CONTROL_NONE;
    usart_config_struct.usart_mode = USART_MODE_RX | USART_MODE_TX;
    usart_init(USART1, &usart_config_struct);

    __USART_ENABLE(USART1);

    //2- Configure the AutoBaudRate method
    usart_auto_baud_rate_config(USART1, USART_AUTO_RATE_MEASURE_START_BIT); // Configure the
AutoBaudRate

    __USART_FUNC_ENABLE(USART1, AUTO_BAUDRATE); // Enable AutoBaudRate
    while (__USART_FLAG_STATUS_GET(USART1, RENACT) == RESET)
    ; // Wait receice enable acknowledge flag is set

    while (__USART_FLAG_STATUS_GET(USART1, TENACT) == RESET)
    ; // Wait Transmit enable acknowledge flag is set
    while (__USART_FLAG_STATUS_GET(USART1, ABRT) == RESET)
    ; // end of Autobaudrate phase
```

在初始化过程完成后，USART 等待接收数据，然后进入自动波特率检测阶段。通过 ABRTF 标志位监测此阶段是否结束。检测 ABRTERRF 标志位，判断自动波特率检测操作是否成功。

```
// If AutoBaudBate error occurred
if (__USART_FLAG_STATUS_GET(USART1, ABRTERR) != RESET)
{
    //error handler
}
else
{
    while (__USART_FLAG_STATUS_GET(USART1, RXNE) == RESET)
        ; // Wait  RXNE flag  set
    while (__USART_FLAG_STATUS_GET(USART1, TXE) == RESET)
        ; // Wait TXE flag  set

    printf("The received character is: %c\n\r", __USART_DATA_RECV(USART1));
}
```

2. 软件自动波特率检测

2.1. 软件自动波特率检测概述

本章节介绍软件方法进行自动波特率检测，适用于 MCU 不支持硬件自动波特率检测或采用非硬件预设模式。

软件检测的思路是发送 0x7F 数据帧到 USART_RX 引脚，并将该引脚配置为每个上升沿产生中断。使用 Systick 定时器测量两个上升沿之间间隔的持续时间。此持续时间对应于 8 位的持续时间，从而计算通讯速率。

- 位时间 = 计算的持续时间/8
- 波特率 = 1/位时间

将计算的波特率配置到 USARTx_BRT 寄存器中。

图 1　软件自动波特率检测原理

2.2. 软件配置示例

软件配置自动波特率检测流程如下所示。

- 初始化时钟配置，本示例配置 PLL 为 48MHz。
- 配置 GPIO 为输入模式，中断方式是上升沿触发中断。
- 等待接收 0x7F，测量上升沿间隔时间，计算波特率。
- 配置 USART。

参考源码如下所示。

```
static volatile uint8_t times = 0;
static volatile uint16_t divider = 0;
static volatile uint32_t timer_cnt = 0;
static uint16_t old_pin;

static void timer_start(void)
{
```

```
    SysTick->CTRL  = 0;
    SysTick->LOAD  = 0xFFFFFF;
    SysTick->VAL   = 0UL;
    SysTick->CTRL  = SysTick_CTRL_CLKSOURCE_Msk |
                     SysTick_CTRL_TICKINT_Msk   |
                      SysTick_CTRL_ENABLE_Msk;
}

static uint32_t timer_stop(void)
{
    SysTick->CTRL = 0;
    return SysTick->VAL;
}

static void exti_isr(uint16_t pin)
{
    if (old_pin == 0 || old_pin != pin) {
        timer_start();
        old_pin = pin;
        return;
    }

    if (old_pin == pin) {
        timer_cnt = timer_stop();
    }
}

/**
  * @brief This function handles EXTI line 4 to 15 interrupts.
  */
static void EXTI4_15_IRQHandler(void)
{
    if((EXTI->PDF & GPIO_PIN_10))
    {
        EXTI->PDF = GPIO_PIN_10;
        exti_isr(GPIO_PIN_10);
    }
}


static void auto_baud(void)
{
    uint16_t reg = 0;

    __RCU_APB2_CLK_ENABLE(RCU_APB2_PERI_SYSCFG);

    gpio_mode_set(GPIOA, GPIO_PIN_10, GPIO_MODE_IN_PD);

    syscfg_exti_line_config(SYSCFG_EXTI_PORT_PA, SYSCFG_EXTI_PIN_10); // Connect EXTI10 Line
to PA10 pin

    __EXTI_INTR_ENABLE(EXTI_LINE_10);
```

```
    __EXTI_EDGE_ENABLE(EXTI_EDGE_RISING, EXTI_LINE_10);

    NVIC_EnableIRQ(IRQn_EXTI4_15);

    while(timer_cnt == 0);

    timer_cnt = 0xFFFFFF - timer_cnt;
    timer_cnt = (48000000 << 3)/timer_cnt;

    /* (divider * 10) computing in case Oversampling mode is 16 Samples */
    divider = (uint16_t)(48000000 / (timer_cnt));
    reg  = (uint16_t)(48000000 % (timer_cnt));

    if (reg >= (timer_cnt/2))
    {
        divider++;
    }
}

void uart_init()
{
    usart_config_t usart_config_struct = {0};
    /* RCU Connfig */
    __RCU_AHB_CLK_ENABLE(RCU_AHB_PERI_GPIOA);

    rcu_usartclk_config(RCU_USART1CLK_CFG_PCLK);

    auto_baud();

    __RCU_APB2_CLK_ENABLE(RCU_APB2_PERI_USART1);

    // GPIO MF Config
    gpio_mf_config(GPIOA, GPIO_PIN_9, GPIO_MF_SEL1);
    gpio_mf_config(GPIOA, GPIO_PIN_10, GPIO_MF_SEL1);

    gpio_mode_set(GPIOA, GPIO_PIN_9|GPIO_PIN_10, GPIO_MODE_MF_PP(GPIO_SPEED_MEDIUM));

    NVIC_EnableIRQ(IRQn_USART1);

    // USART Config
    USART1->CTR1 = 0x0000000C;
    USART1->BRT  = divider;
    USART1->CTR1 |= USART_CTR1_RXNEIE;
    USART1->CTR1 |= USART_CTR1_UEN;

    __USART_ENABLE(USART1);
}
```

免责声明和版权公告

本文档中的信息，包括供参考的 URL 地址，如有变更，恕不另行通知。

本文档可能引用了第三方的信息，所有引用的信息均为“按现状”提供，芯海科技不对信息的准确性、真实性做任何保证。

芯海科技不对本文档的内容做任何保证，包括内容的适销性、是否适用于特定用途，也不提供任何其他芯海科技提案、规格书或样品在他处提到的任何保证。

芯海科技不对本文档是否侵犯第三方权利做任何保证，也不对使用本文档内信息导致的任何侵犯知识产权的行为负责。本文档在此未以禁止反言或其他方式授予任何知识产权许可，不管是明示许可还是暗示许可。

Wi-Fi 联盟成员标志归 Wi-Fi 联盟所有。蓝牙标志是 Bluetooth SIG 的注册商标。

文档中提到的所有商标名称、商标和注册商标均属其各自所有者的财产，特此声明。