

## 前言

正确的USART通信要求发送和接收波特率的匹配度足够高，否则可能发生通信错误。

当在两个设备之间建立通信链路时，自动波特率检测十分有用，因为从设备能够检测到主控制器的波特率并进行相应的自我调整。这需要一种自动机制来确定波特率。

某些STM32器件中内置的USART外设提供许多功能，包括硬件自动波特率检测。

本应用笔记旨在介绍STM32微控制器的自动波特率检测功能，并为没有在硬件中实现此功能的STM32器件提供替代软件方法。

本应用笔记适用于 [表 1](#) 中所列产品。

表1. 适用产品

类型	产品系列
微控制器	STM32F0系列、STM32F1系列、STM32F3系列、STM32F2系列、STM32F4系列、STM32F7系列、STM32L0系列、STM32L1系列、STM32L4系列。

# 目录

<b>1</b>	<b>硬件自动波特率检测</b> .....	<b>5</b>
1.1	特性概述 .....	5
1.2	自动波特率检测模式 .....	7
1.3	ABR误差计算 .....	8
<b>2</b>	<b>软件自动波特率检测</b> .....	<b>9</b>
<b>3</b>	<b>软件和硬件方法的设置</b> .....	<b>10</b>
3.1	USART1配置示例 .....	10
3.2	硬件自动波特率检测 .....	11
3.3	软件自动波特率检测 .....	13
3.4	结果分析 .....	16
	3.4.1 误差计算 .....	16
	3.4.2 软件和硬件方法的比较 .....	17
<b>4</b>	<b>结论</b> .....	<b>20</b>
<b>5</b>	<b>版本历史</b> .....	<b>21</b>

## 图片索引

图1.	软件自动波特率检测概述 .....	9
图2.	fCK = 72 MHz时ABR的误差计算, 115200 bits/s预期波特率 .....	16
图3.	ABR误差比较 (fCK = HSI时钟, 使用模式2进行硬件检测) .....	17
图4.	ABR误差比较 (fCK = 72MHz, 使用模式2进行硬件检测) .....	18
图5.	波特率比较 (fCK = 72MHz, 预期波特率 = 9 Mbits/s, 使用模式2进行硬件检测) .....	19

# 表格索引

表1.	适用产品 .....	1
表2.	STM32系列的USART硬件自动波特率检测 .....	5
表3.	STM32 USART接口上的硬件自动波特率检测 .....	6
表4.	自动波特率检测模式 .....	7
表5.	软件自动波特率检测详情 .....	13
表6.	文档版本历史 .....	21
表7.	中文文档版本历史 .....	21

# 1 硬件自动波特率检测

## 1.1 特性概述

自动波特率检测（ABR）使接收设备能够接受来自各种以不同速率工作的发送设备的数据，无需事先建立数据速率。

在一些STM32产品中，USART能够使用专用硬件自动确定波特率。

[表 2](#)提供了支持自动波特率检测的STM32系列设备的概述。

**表2. STM32系列的USART硬件自动波特率检测**

产品	支持ABR
<b>主流</b>	
STM32F0	有
STM32F1	无
STM32F3	有
<b>高性能</b>	
STM32F2	无
STM32F4	无
STM32F7	有
<b>超低功耗</b>	
STM32L0	有
STM32L1	无
STM32L4	有

对于内置ABR的STM32系列设备而言，并非所有实例化USART接口均支持自动波特率检测。[表 3](#)详细说明了这一限制。

表3. STM32 USART接口上的硬件自动波特率检测<sup>(1)(2)(3)</sup>

端口	STM32 F0										STM32 F3						STM32 F7			STM32L0			STM32L4		
	STM32F030x4、 STM32F030x6	STM32F030x8	STM32F070x6	STM32F070xB	STM32F030xC	STM32F03x	STM32F05x	STM32F04x	STM32F04x STM32F07x	STM32F09x	STM32F37xx	STM32F302xB/C STM32F302xD/E	STM32F302x6/8	STM32F303xB/C STM32F358xC STM32F303xD/E STM32F398xE	STM32F303x6/8 STM32F328x8	STM32F334xx	STM32F301x6/8 STM32F318x8	STM32F745xx STM32F746xx	STM32F756xx	STM32L0x1	STM32L0x2	STM32L0x3	STM32L4x1 / STM32L4x2 / STM32L4x3 / STM32L4x5 / STM32L4x6		
USART 1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		
USART 2	0	-	-	X	X	0	-	-	X	X	X	-	X	-	-	X	X	X	X	X	X	X	X		
USART 3	0	0	0	-	-	0	0	0	-	X	X	X	-	X	-	-	X	X	X	0	0	0	X		
USART 4	0	0	0	-	-	0	0	0	-	-	-	0	-	0	0	0	X	X	-	-	-	-	X		
USART 5	0	0	0	0	0	0	0	0	0	-	-	0	-	0	0	0	X	X	-	-	-	-	X		
USART 6	0	0	0	0	0	0	0	0	0	-	-	0	0	0	0	0	X	X	0	0	0	0	0		
USART 7	0	0	0	0	0	0	0	0	0	-	-	0	0	0	0	0	X	X	0	0	0	0	0		
USART 8	0	0	0	0	0	0	0	0	0	-	-	0	0	0	0	0	X	X	0	0	0	0	0		

1. X: 支持
2. -: 不支持
3. 0: USART实例不可用

## 1.2 自动波特率检测模式

ABR是指接收设备通过检查第一个字符（通常是预先选择的标志字符）确定传入数据速率的过程。

STM32产品上的自动波特率检测功能内置的各种模式基于不同字符模式：

- 以“1”位为开头的任意字符：**模式0**
- 以10xx模式开头的任何字符：**模式1**
- 0x7F：**模式2**
- 0x55：**模式3**

表4. 自动波特率检测模式

ABR模式	说明	波形
0	接收的字符为以“1”位为开头的字符。这种情况下，USART会测量起始位的持续时间（下降沿到上升沿）。	<p>MSv43521</p>
1	以10xx模式开头的任何字符。这种情况下，USART会测量起始位和第一个数据位的持续时间。从下降沿到下降沿测得的持续时间，可在信号斜率较小时确保较高的精度。	<p>MSv43522</p>
2	0x7F字符帧。在此情况下，将首先在起始位结束处更新波特率，然后是在位6结束时更新波特率。	<p>MSv43523</p>
3	0x55字符帧。在此情况下，将首先在开始位结束处更新波特率，然后是位0的结束处，最后是在位6的结束处。同时，对RX线路的每个中间转换执行其它检查。	<p>MSv43524</p>

在激活自动波特率检测之前，必须通过USARTx\_CR2寄存器中的ABRMOD[1:0]字段选择一种ABR模式。在所有ABR模式下，都会在同步数据接收期间多次检测波特率，并将每一次的检测值与上一次的检测值进行比较。

注：在7位数据长度模式下，不支持0x7F和0x55帧检测ABR模式。

### 1.3 ABR误差计算

由USART时钟源（fCK）决定通信速率范围（尤其是最大通信速率）。接收器采用不同的用户可配置过采样技术，可区分有效输入数据和噪声，从而用于恢复数据。这可以在最大通信速率与抗噪声/时钟不准确性之间实现平衡。

可通过编程USARTx\_CR1寄存器中的OVER8位来选择过采样方法，可以是波特率时钟的16倍或8倍。

USART时钟源频率必须与预期通信速率兼容：

- **16倍过采样**时，波特率介于fCK/65535与fCK/16之间。
- **8倍过采样**时，波特率介于fCK/65535与fCK/8之间。

波特率误差取决于USART时钟源、过采样方法和ABR模式。

$$\text{误差 (\%)} = \left| \frac{\text{预期波特率} - \text{实际波特率}}{\text{预期波特率}} \right| \cdot 100$$

其中：

- *预期波特率*取决于发送设备
- *实际波特率*是USART接收器使用自动波特率检测操作确定的波特率。



## 2 软件自动波特率检测

如果不支持硬件自动波特率检测，可采用本节描述的软件方法。

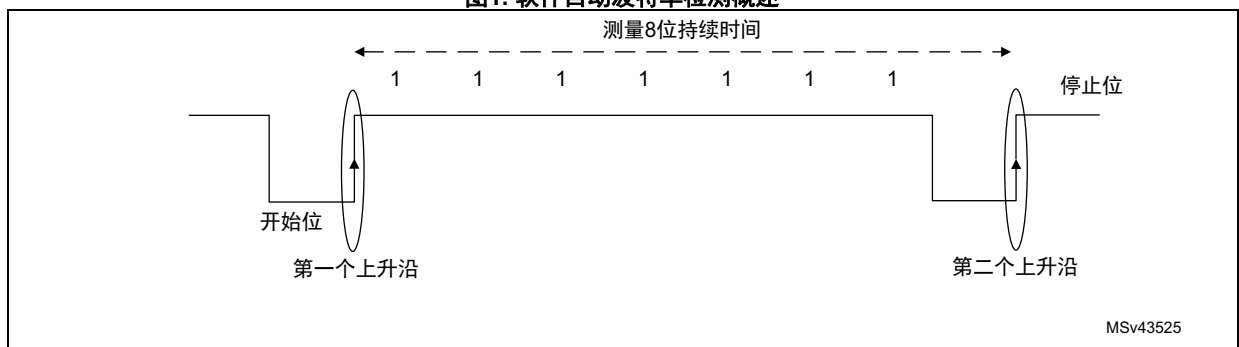
软件方法的理念是发送0x7F数据帧到USARTx\_RX引脚。这将连接到EXTI线路，该线路被配置为在每个上升沿生成中断。

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，因此

- 位时间 = 计算的持续时间 / 8
- 波特率 = 1/位时间

然后，根据计算的波特率值进行USARTx\_BRR寄存器编程。

图1. 软件自动波特率检测概述



### 3 软件和硬件方法的设置

此设置示例使用的是内置硬件自动波特率检测功能的STM32F303xD/E。

PC应用“超级终端”用于向/从STM32F303发送/接收数据帧。因此，测试的是介于600 bits/s至115200 bits/s之间的标准波特率。使用另一个STM32F3器件作为发送器测试可以达到的最高波特率值（9 Mbits/s）。

#### 3.1 USART1配置示例

在两个示例中，STM32 USART1的配置如下：

```

/###-1- Configure the UART peripheral #####*/
/* Put the USART peripheral in the Asynchronous mode (UART Mode) */
/* UART configured as follows:
  - Word Length = 8 Bits
  - Stop Bit   = One Stop bit
  - Parity    = NONE parity
  - BaudRate  = 115200 baud It can be any other value as the USARTx_BRR register will be
reprogrammed
  - Hardware flow control disabled (RTS and CTS signals)
  - The oversampling mode is 8 or 16 (Both are tested)
*/
UartHandle.Instance      = USARTx;
UartHandle.Init.BaudRate = 115200;
UartHandle.Init.WordLength = UART_WORDLENGTH_8B;
UartHandle.Init.StopBits = UART_STOPBITS_1;
UartHandle.Init.Parity = UART_PARITY_NONE;
UartHandle.Init.HwFlowCtl = UART_HWCONTROL_NONE;
UartHandle.Init.Mode = UART_MODE_TX_RX;
UartHandle.Init.OverSampling = UART_OVERSAMPLING_16;

```

**注：** USART1时钟源是使用HSE PLL时钟源的72 MHz系统时钟。（某些测试使用HSI时钟作为USART1时钟源来执行。这是为了检查HSI不准确性对结果的影响。）

## 3.2 硬件自动波特率检测

USART1被配置为自动检测波特率。用户必须在USART1初始化函数中选择ABR模式，如下所示：

```
/*##-2- Configure the AutoBaudRate method */
UartHandle.AdvancedInit.AdvFeatureInit =UART_ADVFEATURE_AUTOBAUDRATE_INIT;
UartHandle.AdvancedInit.AutoBaudRateEnable =
UART_ADVFEATURE_AUTOBAUDRATE_ENABLE;

/*Uncomment your appropriate mode */

//UartHandle.AdvancedInit.AutoBaudRateMode =
UART_ADVFEATURE_AUTOBAUDRATE_ONSTARTBIT;
//UartHandle.AdvancedInit.AutoBaudRateMode =
UART_ADVFEATURE_AUTOBAUDRATE_ONFALLINGEDGE;
//UartHandle.AdvancedInit.AutoBaudRateMode =
UART_ADVFEATURE_AUTOBAUDRATE_ON0X7FFRAME;
//UartHandle.AdvancedInit.AutoBaudRateMode =
UART_ADVFEATURE_AUTOBAUDRATE_ON0X55FRAME;

if (HAL_UART_Init(&UartHandle) != HAL_OK)
{
/* Initialization Error */
Error_Handler();
}

/* Wait until Receive enable acknowledge flag is set */
while(__HAL_UART_GET_FLAG(&UartHandle,UART_FLAG_REACK) == RESET)
{}

/* Wait until Transmit enable acknowledge flag is set */
while(__HAL_UART_GET_FLAG(&UartHandle,UART_FLAG_TEACK) == RESET)
{}

/* Loop until the end of Autobaudrate phase */
while(__HAL_UART_GET_FLAG(&UartHandle,UART_FLAG_ABRF) == RESET)
{}

```

在整个初始化过程完成后，USART等待从超级终端接收数据，然后开始自动波特率检测阶段。通过ABRF标志监测此阶段的结束。

- 如果自动波特率检测操作不成功，则ABRE标志置位
- 如果自动波特率检测操作成功完成，则向超级终端发送确认数据。

```
/* If AutoBaudBate error occurred */

if (__HAL_UART_GET_FLAG(&UartHandle, UART_FLAG_ABRE) != RESET)
{
    Error_Handler();
}
else
{
    /* Wait until RXNE flag is set */
    while(__HAL_UART_GET_FLAG(&UartHandle, UART_FLAG_RXNE) == RESET)
    {}
    /* Send acknowledgement message*/
    if (HAL_UART_Transmit_DMA(&UartHandle, (uint8_t *)aTxBuffer, TXBUFFERSIZE) != HAL_OK)
    {
        /* Transfer error in transmission process */
        Error_Handler();
    }
    while (HAL_UART_GetState(&UartHandle) != HAL_UART_STATE_READY)
    {
    }
}
```

### 3.3 软件自动波特率检测

表 5详细说明了软件方法。

表5. 软件自动波特率检测详情

动作	代码
HAL库初始化。 暂停Tick递增，以防止被 Systick中断唤醒。	HAL_Init(); HAL_SuspendTick();
将系统时钟配置为72 MHz。 可最后在主程序中执行 SystemCoreClockUpdate函数 来验证CPU工作频率。	System Clock source = PLL (HSE) PLLMUL = RCC_PLL_MUL9 (9) Flash Latency(WS) = 2
配置UART外设。	请参见 <a href="#">第 3.1节: USART1配置示例</a> 。
配置USARTx RX引脚以在每个 上升沿生成中断。	<pre>static void EXTILine1_Config(void) {   GPIO_InitTypeDef GPIO_InitStructure;   /* Enable GPIOE clock */   __GPIOE_CLK_ENABLE();   /* Configure PE1 pin as input floating */   GPIO_InitStructure.Mode = GPIO_MODE_IT_RISING;   GPIO_InitStructure.Pull = GPIO_NOPULL;   GPIO_InitStructure.Pin = GPIO_PIN_1;   HAL_GPIO_Init(GPIOE, &amp;GPIO_InitStructure);   /* Enable and set EXTI Line0 Interrupt to the lowest priority */   HAL_NVIC_SetPriority(EXTI1_IRQn, 2, 2);   HAL_NVIC_EnableIRQ(EXTI1_IRQn); }</pre>

表5. 软件自动波特率检测详情 (续)

动作	代码
<p>在Rx引脚上接收到0x7F, 等待中断结束。 启动 <a href="#">第 2 节: 软件自动波特率检测</a> 中描述的自动波特率检测序列。</p>	<pre> /*Wait until the end of interrupt */ while (end_interrupt_flag != 1) {     BSP_LED_On(LED2); } /* Autobaudrate sequence : Update BRR register */ Autobaudrate(); /* Send acknowledgement */ if (HAL_UART_Transmit_DMA(&amp;UartHandle, (uint8_t *)aTxBuffer, TXBUFFERSIZE) != HAL_OK) {     /* Transfer error in transmission process */     Error_Handler(); } while (HAL_UART_GetState(&amp;UartHandle) != HAL_UART_STATE_READY) {} /* 无限循环 */ while (1) {}                 </pre>
<p>自动波特率检测函数</p>	<pre> static void Autobaudrate(void) {     float tmp=0, elapsed;     uint32_t USART1_clk=0;     uint32_t start_time_val=0;     uint32_t BRR=0;     tmp += 0xFFFFFFFF - stop_time_val;     tmp -= start_time_val;     elapsed =(tmp/(SystemCoreClock/1000000))/8;     USART1_clk=SystemCoreClock;     if( (USART1-&gt;CR1 &amp; 0x8000)== 0x8000)     {         /*In case of oversampling by 8*/         BRR =(elapsed*((2*USART1_clk)/1000000))+1;         USART1-&gt;BRR= BRR;     }     else     {         /*In case of oversampling by 16*/         BRR =(elapsed* ((USART1_clk)/1000000))+1;         USART1-&gt;BRR=BRR;     } }                 </pre>

表5. 软件自动波特率检测详情 (续)

动作	代码
外部线路1中断请求: 第一个上升沿: temp=0启动 systick定时器 第二个上升沿: - 禁用SysTick计数器 - 获取编码时间 - SysTick计数器清零	<pre> void EXTI1_IRQHandler() {   HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_1);   if(temp==0)   {     HAL_SYSTICK_Config(0xFFFFF);     temp++;   }   else   {     SysTick-&gt;CTRL &amp;= SysTick_Counter_Disable;     /* Stop the Timer and get the encoding time */     GETMYTIME(&amp;stop_time_val);     /* Clear the SysTick Counter */     SysTick-&gt;VAL = SysTick_Counter_Clear;     /* Clear the temp flag*/     temp=0;     /*end of interrupt*/     interrupt_flag=1;   } } </pre>
所需项目定义	<pre> #define SysTick_Counter_Disable ((uint32_t)0xFFFFF) #define SysTick_Counter_Enable ((uint32_t)0x00000001) #define SysTick_Counter_Clear ((uint32_t)0x00000000) #define GETMYTIME(_t) (*_t=SysTick-&gt;VAL) </pre>

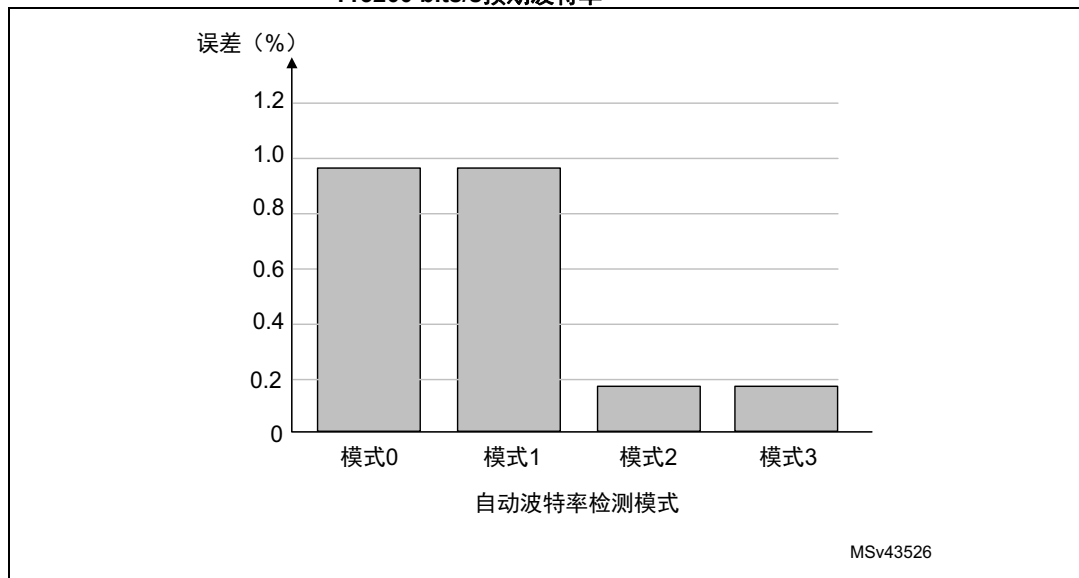
## 3.4 结果分析

### 3.4.1 误差计算

图 2显示ABR模式2和3的精确度高于模式0和1；它们的波特率误差值更低。

不过，由于预期波特率与实际波特率之间的误差小于1%，因此所有模式的结果均正常。

图2. fCK = 72 MHz时ABR的误差计算，  
115200 bits/s预期波特率





### 3.4.2 软件和硬件方法的比较

图 3显示在通常情况下，当由72 MHz系统时钟为USART提供时钟（HSE作为PLL时钟源）时，结果优于USART时钟源使用HSI时钟。这要归因于HSI的相对不准确性。

图3. ABR误差比较（fCK = HSI时钟，使用模式2进行硬件检测）

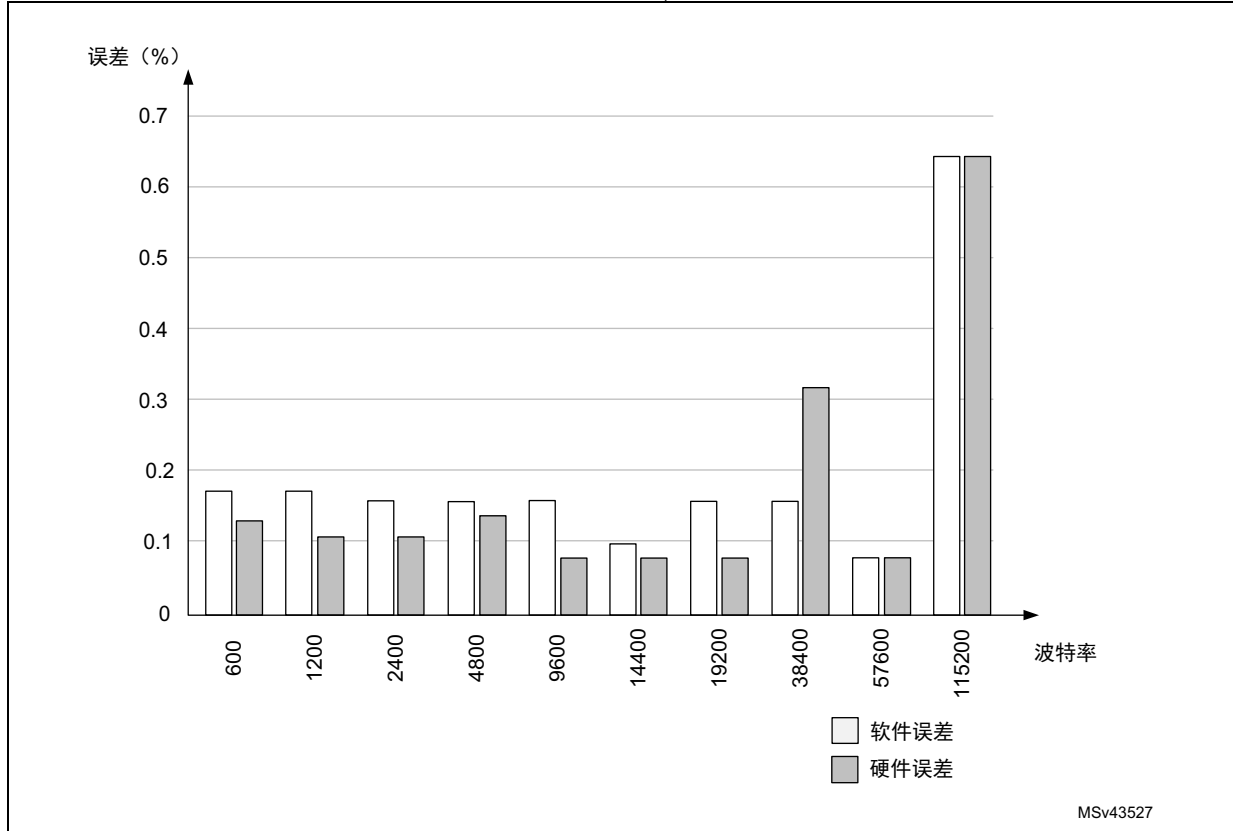


图 4显示在大多数情况下，硬件方法提供的结果优于软件方法。不过，在某些情况下，软件方法能够提供相比于使用硬件方法时更好的结果。

图4. ABR误差比较 (fCK = 72MHz, 使用模式2进行硬件检测)

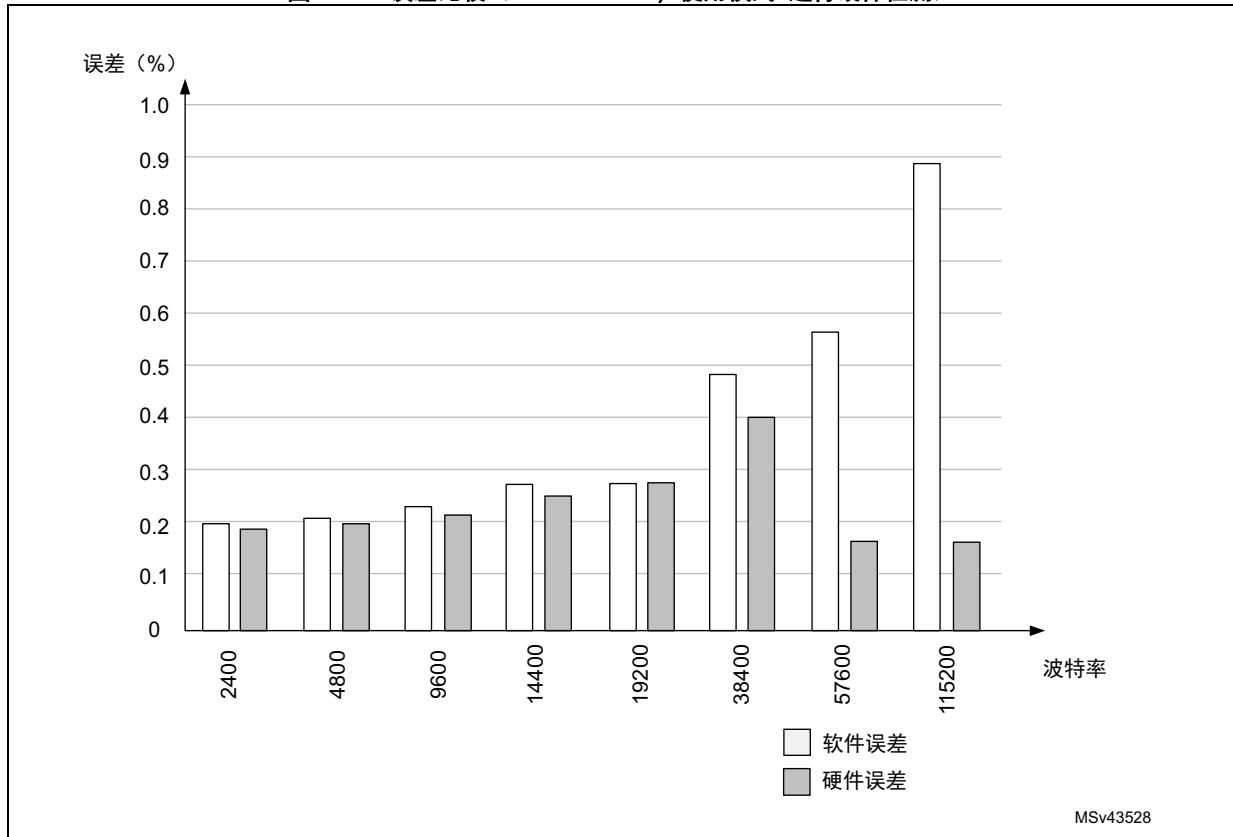
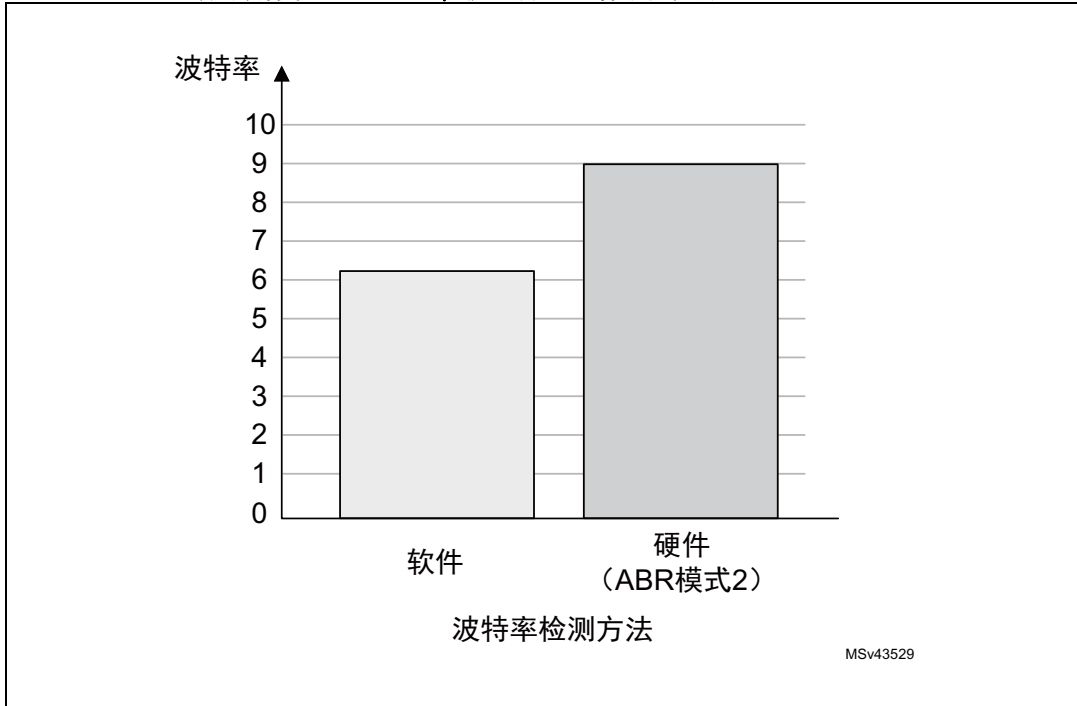


图 5显示:

- 使用硬件方法时, 达到最大波特率9 Mbits/s时误差为0%。
- 使用软件方法时, 达到最大波特率时误差为约30%, 这要归因于执行中断处理程序所花费的CPU周期。

图5. 波特率比较 (fCK = 72MHz, 预期波特率 = 9 Mbits/s, 使用模式2进行硬件检测)



## 4 结论

此应用笔记描述了某些STM32器件内置的硬件自动波特率检测功能。它还提供了在软件中实现此功能的技术，作为STM32器件没有在硬件中实现此功能的解决方案。

尽管示例中的自动波特率检测均应用在示例的开头部分，但是可以进行扩展并在每次发送和接收设备检测到通信错误时使用。当主机使用不同波特率进行通信时，这一特性可实现应用的稳健性。

## 5 版本历史

表6. 文档版本历史

日期	版本	变更
2016年11月 15日	1	初始版本

表7. 中文文档版本历史

日期	版本	变更
2017年11月 19日	1	中文初始版本

**重要通知 - 请仔细阅读**

意法半导体公司及其子公司 (“ST”) 保留随时对 ST 产品和 / 或本文档进行变更、更正、增强、修改和改进的权利，恕不另行通知。买方在订货之前应获取关于 ST 产品的最新信息。ST 产品的销售依照订单确认时的相关 ST 销售条款。

买方自行负责对 ST 产品的选择和使用，ST 概不承担与应用协助或买方产品设计相关的任何责任。

ST 不对任何知识产权进行任何明示或默示的授权或许可。

转售的 ST 产品如有不同于此处提供的信息的规定，将导致 ST 针对该产品授予的任何保证失效。

ST 和 ST 徽标是 ST 的商标。所有其他产品或服务名称均为其各自所有者的财产。

本文档中的信息取代本文档所有早期版本中提供的信息。本文档的中文版本为英文版本的翻译件，仅供参考之用；若中文版本与英文版本有任何冲突或不一致，则以英文版本为准。

© 2017 STMicroelectronics - 保留所有权利