AN1005
应用笔记

UM800Y 提高 Uart 波特率精度方法

版本：V1.0

广芯微电子（广州）股份有限公司

http://www.unicmicro.com/

版本修订

版本	日期	描述
V1.0	2022.03.24	初始版

虽然 uart 通信对波特率误差容错范围很大，当用户使用芯片内部时钟进行串口通讯时，由于叠加时钟在全温全压条件下的误差以及波特率计算误差，在个别波特率配置下，可能会出现波特率偏差较大的场景。请根据用户手册中 uart 章节中的波特率误差表选择合适的波特率。

本文是针对提高 uart 波特率精度的方法建议，

一、波特率计算引起的误差

建议根据不同波特率的计算误差，选择合适的波特率（参照 uart 文档中波特率误差表，如果表中没有对应的波特率，可参照如下内容进行自行配置）

例如串口 0：

1. 波特率计算公式

$$\text{Baud Rate} = \frac{\text{SYSCK}}{16\times(1024-\text{S0REL})} \quad (1)$$

2. 假如系统时钟(SYSCK)为 16MHz，波特率为 115200，根据公式（1）

115200 = 16000000/16*(1024 – S0REL)

可得到 S0REL = 1015

3. 把 2 计算得到的 S0REL 值代入公式（1），即

Baud Rate = 16000000/16 * (1024 - 1015)

可得到 Baud Rate = 111111

所以计算得到的误差 = (115200 – 111111) / 115200 = 3.5%

4. 步骤 3 是把 SYSCK 当作完全准确的 16M 来计算的，假设 UM800Y 内部高速 RC 全温全压范围内误差是+/-3%，按最小的+/-3%来计算，

SYSCK = 16000000 – 1600000 * 0.03 = 15520000

把此系统时钟代入公式（1），即

Baud Rate = 15520000/ 16 * (1024 - 1015)

可得到 Baud Rate = 107777

所以误差 = (115200 –107777) / 115200 = 6.4%

5. 按最大的 3%来计算，SYSCK = 16000000 + 1600000 * 0.03 = 16480000

把此系统时钟代入公式（1），即

Baud Rate = 16480000/ 16 * (1024 - 1015)

可得到 Baud Rate = 114444

所以误差 = (115200 –114444) / 115200 = 0.65%

二、时钟精度引起的误差

芯片在出厂时，已进行常温下的时钟精度校准，在实际使用中，由于温度环境的影响，时钟会有一定范围内的偏移，从而影响 uart 波特率精度，为了提高 uart 波特率精度，可以优化修改 uart 初始化函数和时钟初始化函数(如下图)，在利用系统时钟进行波特率计算时，用计算出来的系统时钟值（system_core_clock 是系统时钟值，该全局变量在 clock_init()函数中已配置 ）。

```
void uart0_init(uint32_t baud_rate)
{

    PCLK0 |= (1<<0);                                    //开UART0时钟使能
    PRESET0 |= (1<<0);                                  //UART0正常工作
    EUARTEN |= (1<<0);                                  //打开EUART0功能，P2.6作

    tx_flag = 0;

    SM0 = 0;
    SM1 = 1;                                           //选择模式一

    TI0 = 0;                                           //发送完成中断状态清除
    RI0 = 0;                                           //接收完成中断状态清除

    uart0_set_baud_rate(system_core_clock, baud_rate);  //设置波特率
    REN0 = 1;                                          //使能接收

}
```

```
void clock_init(uint32_t system_clk_hz)
{
    uint32_t crystal_clk = 0;
    CLKCON |= (0x1<<2);             //RCH时钟源打开
    while((CLKCON&0x20) != 0x20);  //等待RC16M时钟稳定
    rch24m = ((uint32_t)(*(volatile uint8_t xdata*)(0x91AB))<<24) |  \
             ((uint32_t)(*(volatile uint8_t xdata*)(0x91AA))<<16) |  \
             ((uint32_t)(*(volatile uint8_t xdata*)(0x91A9))<<8)  |  \
              (uint32_t)(*(volatile uint8_t xdata*)(0x91A8));
    if ((rch24m <= 24120000) && (rch24m >= 23880000))    //0.5%以内
    {
        crystal_clk = 24000000*2;
    }
    else if(rch24m == 0xFFFFFFFF)
    {
        crystal_clk = 24000000*2;
    }
    else
    {
        crystal_clk = rch24m*2;
    }
    switch(system_clk_hz)
    {
        case 24000000:
            RCHDIV = (0x1<<0);                      //RCH 2分频
            system_core_clock = crystal_clk/2;
            break;
        case 16000000:
            RCHDIV = (0x2<<0);                      //RCH 3分频
            system_core_clock = crystal_clk/3;
            break;
        case 12000000:
            RCHDIV = (0x3<<0);                      //RCH 4分频
            system_core_clock = crystal_clk/4;
            break;
        default:
            RCHDIV = (0x2<<0);                      //RCH 3分频
            system_core_clock = crystal_clk/3;
            break;
    }
    CLKCON &= ~(0x1<<0);                    //RCH为系统时钟
    SYSDIV &= ~(7<<0);                      //系统时钟不分频
}
```