昆明学院学报　2020，42（6）：88～92，97　ISSN 1674－5639　CN 53－1211/G4
9Journal of Kunming University

物联网工程中 GPIO 模拟串口通用构件研究

石　栋
（昆明学院 信息工程学院，云南 昆明 650214）

摘要：GPIO 模拟串口是物联网工程中常用的串口扩展方法．针对通用 MCU 及其应用场景，分析了串行通信波特率与 MCU 时钟频率的数学关系，给出计算位长的数学模型．同时根据物联网工程中的基本原理，提出一种含有串口基本要素的 GPIO 模拟串口构件设计方案．测试结果显示，该构件可在不同时钟频率和不同波特率下正常工作，适用于不同类型的 MCU，具备可移植性与可复用性，能直接应用于物联网工程的串口扩展设计．

关键词：GPIO 模拟串口；构件；可移植；可复用；设计方法
中图分类号：TP391　**文献标识码**：A　**文章编号**：1674－5639（2020）06－0088－05
DOI：10.14091/j.cnki.kmxyxb.2020.06.019

Research on GPIO Analog Serial Port General Components in Internet of Things Engineering

SHI Dong
（College of Information Engineering，Kunming University，Kunming，Yunnan，China　650214）

Abstract：GPIO analog serial port is a commonly used serial port expansion method in the Internet of Things engineering practice. According to the global MCU and its specific application scenarios，we analyzed the the mathematical relationship between the serial communication baud rate and the MCU clock frequency to give a mathematical model for calculating the bit length. Meanwhile，according to the basic principle of the Internet of Things engineering，this paper presents a design scheme of GPIO analog serial port component with the essential factors of serial port. Experiments show that the component can work well at different clock frequencies and different baud rates，and it is suitable for different types of MCUs with good portability and reusability，and it can be directly applied to the extended design of serial port in Internet of Things engineering.

Key words：GPIO analog serial port；component；portable；reusable；design method

串口是嵌入式系统开发中微控制器（MCU）间或微控制器与上位机之间最基本的通信手段．随着物联网技术的不断发展，MCU 的功能越来越强大．但考虑到成本问题，MCU 生产商在 MCU 上仅会配置 1～3 个 UART，以至于在嵌入式系统的开发过程中如果需要较多的串口通信时，就必须采用扩展串口的方式来实现．

目前，相关文献报道的扩展串口方法主要有 3 种基本方案：1）利用硬件扩展多串口通信[1]．该方式是利用数字控制的模拟数据选择/分配器，或利用串口的扩展芯片来扩展串口．2）利用 MCU 定时器的定时功能实现软件模拟串口．该类型的设计又分为两种，一种是利用 MCU 上的定时器进行延时，从而用模拟的方法实现串口功能[2-5]，另一种是使用定时器的输出比较、输入捕获的功能实现 UART 功能[6]．3）利用软件空循环的方式实现模拟串口[7]．这种方法使用 for 空循环语句进行延时，以实现控制串口通信每一位的持续时间．

在现有方案中，方案 1 的扩展方法需增加硬件成本；方案 2 需占用 MCU 定时器资源；方案 3 无法实现多时钟频率、多波特率下正常工作，可移植性差．因此，本文通过分析串口波特率、MCU 内核时钟频率和延时函数之间的数学关系，借此给出准确延时的计算方案，从而在不增加硬件资源、不占用 MCU 定时器资源的情况下，利用纯软件方法实现可以在多内核时钟频率、多波特率下正常工

收稿日期：2020－01－07
基金项目：昆明学院科学研究基金资助项目“基于 NB-IoT 的物联网应用工程框架研究”（XJZ01707）．
作者简介：石栋（1972—），男，云南昭通人，讲师，主要从事物联网系统开发研究．

作，且具备可移植、可复用的 GPIO 模拟串口构件设计. 同时鉴于目前文献中尚未解决用软件方法实现串口功能时代码时间开销的测算问题，或只能粗略估算代码时间开销、仅能实现少量个字符接收的问题[7]，提出一种较为准确的调试方法，从而较好地解决模拟串口的测试问题.

1　通用异步串口的技术要点分析

1.1　异步串口通信的技术要点

常见串口的基本要素包含串行通信数据格式、波特率和奇偶校验. 图 1[8] 给出了 8 位数据、无奇偶校验情况的传送格式.

图1　串行通信数据格式

波特率是串口每秒内传送的位数，其倒数就是发送一位的位长，也称为位的持续时间. 而奇偶校验分为奇校验和偶校验，在使用时其仅能验证字符中含“1”的位数是奇数个还是偶数个，校验功能有限，实际应用中多不使用. 因此本设计不考虑该设计方法.

1.2　GPIO 模拟串口的技术要点

要实现通过 GPIO 模拟串口与上位机等其他设备的通信，模拟串口要具备与普通串口相同的异步串口通信格式、波特率等. 从图 1 可以看出，每发送（接收）一位都要持续一定的时间长度，即位长. 一方面位长是波特率 f_{baud} 的倒数，另一方面位长还等于位长的时钟周期数与 MCU 内核时钟周期（内核时钟频率 f_{CPU} 的倒数）的乘积. 如果使用软件延时的方法实现位长，则软件延时所需时钟周期数就应该等于位长的时钟周期数. 那么软件延时数 N 就应该与波特率 f_{baud} 和 MCU 内核时钟频率 f_{CPU} 间存在数学关系 $N=f(f_{baud}, f_{CPU})$，找到这一关系后，就可以用软件的方式实现串口位长的精确延时，同时利用该数学关系还可实现 GPIO 模拟串口的可复用与可移植.

2　delay( ) 函数延时公式推导

本文采用 delay( ) 函数来实现位延时：

```
void delay(uint32 N)
{
  uint_32 i;
  for(i = 0;i < N;i + + )
    _asm(“NOP”);
}
```

其中，“NOP”是汇编指令，称为空指令. 该指令的功能是无操作，通常用作指令对齐或延时[9]. 如果能测算出当 $N=1$，2，3，…时，delay 函数执行所需要的时钟周期数，并找到其入口参数 N 与时钟周期数的关系，就可以利用 delay 函数实现模拟串口的位延时.

2.1　delay( ) 执行周期数的测量

delay( ) 函数执行的时钟周期数可借助定时器测量：将定时器的工作时钟频率设定为 MCU 的内核时钟频率，记录下 delay( ) 函数开始执行时和执行结束时定时器的计数值，就可以计算出 delay( ) 函数执行的时钟周期数. 由于 Cortex-M 系列处理器包含一个简单时钟定时器 SysTick[10]. 因此，本文利用 ARM 公司的 Cortex-M0 + 系列 MCU 中的 SysTick定时器进行说明，测试方法如下：

```
……//初始化并使能 SysTick 定时器.
start_ ticks = SysTick - >VAL; //保存 delay( ) 函数开始时定时器计数值.
  delay(N); //delay( ) 函数执行.
end_ ticks = SysTick - >VAL; //保存 delay( ) 函数结束时定时器计数值.
……//计算输出语句或函数执行的时钟周期数.
```

在 NXP 公司基于 ARM Cortex-M0 + 内核的 KL25 上，测量出 delay( ) 函数入口参数 N 与函数执行所需时钟周期数（N_{tick}）的关系，如表 1 所示.

表 1　delay( ) 函数入口参数与时钟周期数的关系

入口参数（N）	1	2	3	4	5	6	7	8	9	…
时钟周期数（N_{tick}）	55	68	81	94	107	120	133	146	159	…

从表 1 可以看出，当 delay( ) 函数入口参数为 1 时，执行 delay( ) 函数需要的时钟周期数是 55 次，随后 delay( ) 函数的入口参数每增加 1，delay( ) 函数执行需要的时钟周期数就增加 13 次. 由此不难得

出，delay( )函数的入口参数(N)与执行时钟周期数(N_{tick})的关系为：

$$N_{tick}=13(N-1)+55,(N>0), \quad (1)$$

(1) 式中的常数 13 和 55 除决定于 MCU 的 CPU 系列外，还与使用的编译器环境有关. 本文是在 NXP 的 KDS 环境下使用 GNU 编译器实现的. 在不同系列的 MCU 和不同的编译环境下可将公式抽象为：

$$N_{tick}=A(N-1)+B,(N>0), \quad (2)$$

其中 A 和 B 为两个待定常数，可在具体的 MCU 和编译环境中测量确定. 本文以 KL25 在 GNU 编译环境下进行，则 $A=13$，$B=55$.

2.2 串口波特率与 MCU 内核时钟频率的关系

串口的波特率f_{baud}代表了串口 1 s 能够传输的位(Bit)数，其倒数就是串口传输一个二进制位(1 Bit)的位长. 那么，串口传输一个二进制位(1 Bit)所需的时钟周期数就应该等于串口传输一个二进制位(1 Bit)所持续的时间与内核时钟周期的比值，也就是内核时钟频率f_{CPU}与波特率f_{baud}的比值，即：

$$N_{bit}=\frac{f_{CPU}}{f_{baud}}. \quad (3)$$

2.3 delay( )函数延时次数的公式推导

用 delay( )函数来实现模拟串口的位延时，则 delay( )函数执行延时所需的时钟周期数应等于串口传输 1 Bit 的时钟周期数. 即：

$$N_{ticks}=N_{bit}. \quad (4)$$

将公式(1)和公式(3)带入公式(4)，可得：

$$N=\frac{1}{13}(\frac{f_{CPU}}{f_{baud}}-55)+1. \quad (5)$$

从公式(5)可以得出，只要确定 MCU 的内核时钟频率以及串口的波特率，就可以确定 delay( )函数的入口参数，因此就可用 delay( )函数在 GPIO 模拟串口实现位延时. 即使内核时钟频率改变或波特率改变，也可用公式(5)计算出新入口参数 N.

3 可移植与可复用 GPIO 模拟串口构件设计

为 delay( )函数找到准确的入口参数后，就可以编程实现模拟串口的功能. 本文采用苏州大学嵌入式实验中心为 KL25 开发的底层驱动，并选用 KL25 开发板，在 NXP 公司提供的 KDS 软件环境中实现模拟串口功能. 为实现可移植与可复用的目的，工程采用构件化的思想将模拟串口的功能进行封装，包含 iouart. h 和 iouart. c 文件. iouart. h 是模拟串口构件的头文件，其内容包含声明保存延时函数入口参数的数组变量，以及模拟串口号的宏定义和函数声明. iouart. c 提供模拟串口的具体实现. 这里仅就模拟串口的初始化、发送和接收功能进行描述，将这 3 个功能封装成 3 个构件：初始化函数 iouart_ init( )；发送一字节函数 iouart_ send1( )；接收一字节函数 iouart_ re1( ).

3.1 GPIO 模拟串口构件初始化设计

将 GPIO 模拟串口的初始化功能封装成构件：iouart_ init (uint_ 8 uartNo, uint_ 32 baud)，参数 uartNo 和 baud 分别代表模拟串口编号和待用的波特率. 该构件需完成 3 个任务：1) 在 MCU 上选择的 2 个空闲 I/O 引脚，将其复用为 GPIO 功能. 2) 将发送功能引脚的 GPIO 功能定义为输出，并令其输出“1”，即输出空闲电平；将用作接收功能引脚的 GPIO 功能定义为输入. 3) 将 MCU 的内核时钟频率(已知值)和选定的波特率代入公式(5)中计算 delay( )函数的入口参数，并将计算出来的入口参数结果放入到一个全局变量中供 delay( )函数使用. 基本程序如下：

```
void iouart_init(uint_8 uartNo,uint_32 baud)
{
  ……//串口号解析及局部变量的声明.
    gpio_init(Uport[ioN],1,1);//初始化 TX 引脚:GPIO 功能、输出、高电平.
    gpio_init(Uport[ioN+1],0,1);//初始化 RX 引脚:GPIO 功能、输入.
    //计算特定波特率下“NOP”指令执行的延时次数,保存到全局变量 nop_ticks[ ]中供延时函数使用.
    nop_ticks[nopN] = ((SystemCoreClock * 1.0/baud) -55.0)/13.0 +1;
}
```

代码中，nop_ ticks [ ] 是保存 delay( )函数入口参数的数组，SystemCoreClock 用于保存系统时钟频率，由 MCU 在启动时配置好时钟频率后赋值. 模拟串口引脚号放置于 Uport [ ] 数组中，在“iouart. c”文件中编写.

3.2 GPIO 模拟串口构件发送功能设计

GPIO 模拟串口构件发送功能封装为构件：iouart_send1(uint_8uartNo, uint_32 ch). 设计只需严格按照图 1 所示的串行通信数据格式设计即可. 构

件函数基本程序如下：

```
void iouart_send1(uint_8 uartNo, uint_8 ch)
{
  ……//函数初始化.
  gpio_set(Uport[ioN],0);//发送起始位.
  delay(nop_ticks[nopN]);//延时一位位长.
  //发送 8 位数据.
  for(i=0;i<8;i++)
  {
    j= ((ch>>i)&0x01);//获取第 i 位状态.
    gpio_set(Uport[ioN],j);//发送 1 位.
    delay(nop_ticks[nopN]);//数据位延时.
  }
  gpio_set(Uport[ioN],1);//发送停止位.
  delay(nop_ticks[nopN]);//停止位延时.
}
```

3.3　GPIO 模拟串口构件接收功能设计

GPIO 模拟串口构件接收功能封装成构件：iouart_re1(uint_8uartNo). 设计按照图 1 所示的串行通信数据格式. 但与发送功能比较，有 3 点不同：1）将开始位持续时间延长为原来的 1.5 倍，保证可以在每一个数据位的中间位置读取接收数据，避免在电平变换时刻错误读取接收引脚的数值. 2）在接收数据位的每一位时采用连续接收 3 次的策略，实现滤波功能，避免因信号波动导致读数错误. 3）接收到停止位后不做位延时而直接返回数据，保证模拟串口接收连续字符时有充裕的处理时间. uint_8 iouart_re1(uint_8 uartNo)函数的核心代码为：

```
uint_8 iouart_re1(uint_8 uartNo)
{
  ……//利用模拟串口号解析出 RX 引脚号及 delay 入口参数.
  k=gpio_get(Uport[ioN]);//获取起始位.
  delay(nop_ticks[nopN]);
  for(i=0;i<8;i++)//读 8 位数据.
  {
    //连读 3 次滤波,同时可减少由于延时不准确的影响.
    k1=gpio_get(Uport[ioN]);
    _asm("NOP");
    k2=gpio_get(Uport[ioN]);
    _asm("NOP");
    k3=gpio_get(Uport[ioN]);
    delay(nop_ticks[nopN]);
    j=0;
    //挑选 3 次读中 2 次相同的值为最终值.
    k1+=(k2+k3);
    if(k1>=2) j=1;
    dat|=((j)<<i);
  }
  k=gpio_get(Uport[ioN]);//读停止位.
  gpio_clear_int(Uport[ioN]);//清除 ISF 中断标志.
  return dat
}
```

3.4　GPIO 模拟串口构件的调试方法

实现了发送、接收功能的基本编程后，由于程序语句执行、子函数调用都需要时间，就会导致模拟串口在发送接收数据时的时序出现问题. 因此还需要对构件进行调试方能正常使用. 为使 GPIO 模拟串口能够正常工作在多种内核时钟频率下，且能够使用多种波特率，调试最好选在低内核时钟频率、高波特率下进行.

3.4.1　发送构件的调试

由于本构件的位延时可以做到非常准确，因此发送时序的开始位、停止位可以少调试甚至不调试，仅需调试数据位的发送时序. 下面以数据位的调试简要说明其调试方法：1）根据使用的内核时钟频率和波特率，计算出位长的理论时钟周期数 N_{tick}（公式（3））. 2）利用 SysTick 定时器测量发送构件 iouart_ send1()发送 1 个字节所用的时钟周期数 $10N_{real}$（N_{real}是发送 1 位所需的周期数，发送 1 个字节包含 1 个开始位、1 个结束位、8 个数据位），这样可以得到实际位长的时钟周期数 N_{real}. 然后比较 N_{data}与 N_{tick}，其差值 $\Delta N = N_{real} - N_{tick}$就是由于数据位语句执行消耗的时钟周期数，这样数据位的位延时就应该减少 $\triangle N/13$ 次，则数据位 delay()函数的入口参数为 $N - \triangle N/13$，至此完成数据位的调试.

3.4.2　接收构件的调试

本文模拟串口采用 GPIO 中断实现接收功能，则接收功能的调试有两个位置：开始位和数据位. 调试开始位时，定时器 SysTick 应从进入 GPIO 中断服务例程时开始计时，到 iouart_ re1()函数开始

位延时后计时结束，此时定时器测得的计数值就是开始位的实际位长 N_{real}，之后处理方法与发送功能的数据位调试方法基本一致．而数据位的调试与发送功能的数据位调试基本一致，此处不再详述．

4 测试与分析

为实现 GPIO 模拟串口的通信功能，本文使用苏州大学 NXP 嵌入式实验中心的 KL25 开发板，在 KDS3.0 环境下利用 KL25 工程框架进行测试．测试中本文设计的场景是，从上位机发送“0x01 ~ 0xFF”共 255 个字符，GPIO 模拟串口接收到字符后再回发到上位机．接收回发功能在中断中实现．

测试结果如图 2 所示．该图是在 KL25 内核时钟频率为 48 MHz、波特率为 9 600 bps 情况下的测试结果．经过改变系统内核时钟频率和波特率进行反复测试的结果表明，本构件在内核时钟频率分别为 24，48 MHz，波特率可以在 300，600，1 200，2 400，4 800，9 600，1 440，19 200，38 400 bps 下正常工作．若将其移植到 NXP 基于 Cortex-M4F 的 K64MCU 上，只需重新确定公式（2）的待定参数 A 和 B，模拟串口构件就可在 K64 上正常工作，从而实现了基于 GPIO 模拟串口的通用性和可复用性．

图2 内核时钟频率48 MHz、波特率9 600 bps下的模拟串口测试

5 结语

本文从计算位长的时钟周期数出发，测算了 delay 函数的入口参数 N 及 MCU 内核时钟频率、串口波特率三者间的关系，提出位延时的数学模型 $N=f(f_{baud},f_{CPU})$，并根据该模型设计并实现了 GPIO 模拟串口构件．利用此方法设计的 GPIO 模拟串口构件可在不同的内核时钟频率、不同的波特率下，实现 GPIO 模拟串口的可移植和可复用．同时针对目前文献尚未报道如何很好地解决模拟串口调试方法的问题，提出利用定时器实现 GPIO 模拟串口发送、接收功能的调试．此外，本设计方法可以使 GPIO 模拟串口具备较好的工作性能．

（下转第 97 页）

[4] 卿平英. 生物制剂在类风湿关节炎应用中的感染风险评估与选择 [J]. 西部医学，2019，31（8）：1305－1307.
[5] 张颖. 刘健教授治疗类风湿关节炎临床经验 [J]. 风湿病与关节炎，2018，11（7）：42－44.
[6] 朱丽芳. 类风湿关节炎患者生存质量的研究进展 [J]. 风湿病与关节炎，2018，7（4）：76－79.
[7] 杨玉慧，许秀丽，张波. 治疗类风湿性关节炎新药托法替布 [J]. 临床药物治疗杂志，2018，16（12）：53－55.
[8] SINGH J A，SAAG K G，BRIDGES S L J，et al. 2015 American College of Rheumatology guideline for the treatment of rheumatoid arthritis [J]. Arthritis Care & Research，2016，68（1）：1－26.
[9] 姚华勤. 延续性健康教育对使用生物制剂的类风湿关节炎治疗中的应用效果 [C] //中国中西医结合学会. 第 16 届中国中西医结合风湿病学术会议论文集. 2018：137－139.
[10] MOREL J，CONSTANTIN A，BARON G，et al. Risk factors of serious infections in patients with rheumatoid arthritis treated with tocilizumab in the French Registry REGATE [J]. Rheumatology，2017，56（10）：1746－1754.
[11] 波特. 默克诊疗手册：上册 [M]. 王卫平，译. 北京：人民卫生出版社，2014：462－463.
[12] 治疗指南有限公司. 治疗指南：风湿病学分册 [M]. 董怡，译. 3 版. 北京：化学工业出版社，2018：117－118.

（上接第 92 页）

[参考文献]

[1] 田炳丽，丁风雷，王冠琳，等. 硬件扩展单片机多串口通讯方式的研究 [J]. 通信技术，2010（10）：153－154.
[2] 杨勇涛，赵雅兴. DSP 的软件 UART 实现 [J]. 半导体技术，2003（10）：61－64.
[3] 周龙甫，呼永河，范泉水，等. 模拟串口技术在多传感器数据采集中的应用 [J]. 医疗卫生装备，2013（10）：4－5，13.
[4] 王富东，邵光庆. 单片机多串口通讯技术及其应用 [J]. 仪器仪表学报，2002（S1）：262－264.
[5] 郑正学，李炜. 单片机模拟串口数据接收程序的实现及优化 [J]. 单片机与嵌入式系统应用，2016（8）：68－70.
[6] 郑志雄，胡爱兰. LPC1768 的全双工 UART 的软件模拟实现 [J]. 单片机与嵌入式系统应用，2013（6）：25－28.
[7] 吴名陵，汪小澄. 单片机多路模拟串口的实现及其性能分析 [J]. 自动化仪表，2008（10）：67－69.
[8] 王宜怀，吴瑾，文瑾. 嵌入式技术基础与实践：ARM Cortex-M0 + Kinetis L 系列微控制器 [M]. 4 版. 北京：清华大学出版社，2017：124－125.
[9] ARM Ltd. ARMv6-M reference manual [EB/OL]. [2019－12－15]. https://silver.arm.com/download/ARM_and_AMBA_Architecture/AR585－DA－70000－r0p0－00rel0/DDI0419C_arm_architecture_v6m_reference_manual.
[10] ARM Ltd. Cortex-M0 + technical reference manual [EB/OL]. [2019－12－15]. http://infocenter.arm.com/help/index.jsp topic =/com.arm.doc.set.cortexm/index.html.