**1.1  UART串口**

通用异步接收器和发送器(Universal Asynchronous Receiver and Transmitter)简称UART。通常是嵌入式设备中默认都会配置的通信接口。这是因为，很多嵌入式设备没有显示屏，无法获得嵌入式设备实时数据信 息，通过UART串口和超级终端相连，打印嵌入式设备输出信息。并且在对嵌入式系统进行跟踪和调试时，UART串口了是必要的通信手段。比如：网络路由 器，交换机等都要通过串口来进行配置。UART串口还是许多硬件数据输出的主要接口，如GPS接收器就是通过UART串口输出GPS接收数据的。

**1.1.1  异步通信和同步通信**

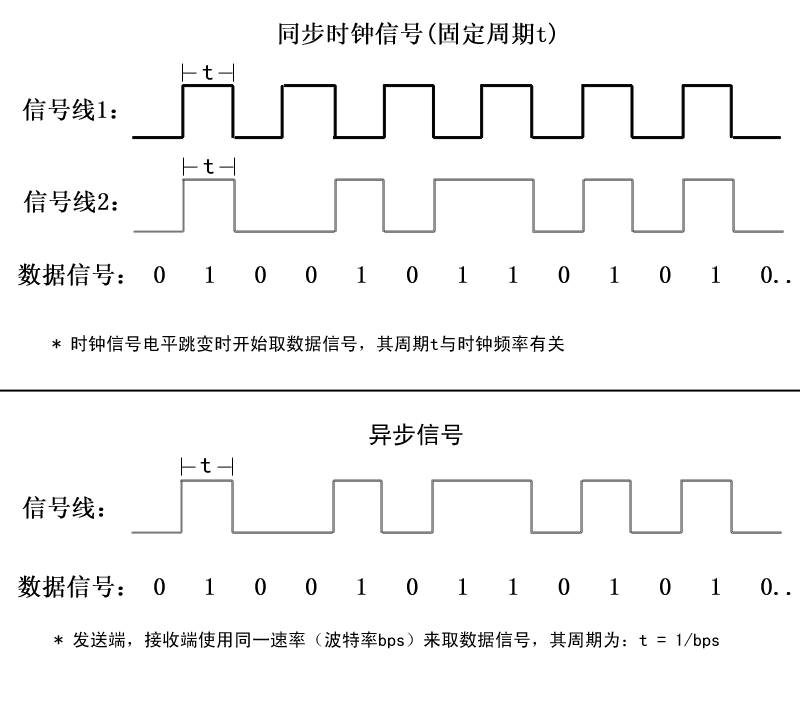
****

图2-56同步信号与异步信号

同步通信技术

在发送数据信号的时候，会同时送出一根同步时钟信号，用来同步发送方和接收方的数据采样频率。如图2-56所示，同步通信时，信号线1是一根同步时 钟信号线，以固定的频率进行电平的切换，其频率周期为t，在每个电平的上升沿之后进行对同步送出的数据信号线2进行采样（高电平代表1，低电平代表0）， 根据采样数据电平高低取得输出数据信息。如果双方没有同步时钟的话，那么接收方就不知道采样周期，也就不能正常的取得数据信息。

异步通信技术

在异步通信技术中，数据发送方和数据接收方没有同步时钟，只有数据信号线，只不过发送端和接收端会按照协商好的协议（固定频率）来进行数据采样。数 据发送方以每秒钟57600bits的速度发送数据，接收方也以57600bits的速度去接收数据，这样就可以保证数据的有效和正确。通常异步通信中使 用波特率（Baud-Rate）来规定双方传输速度，其单位为bps（bits per second每秒传输位数）。

**1.1.2  数据的串行和并行通信方式**

串行通信好比是一列纵队，每个数据元素依次纵向排列。如图2-57所示，传输时一个比特一个比特的串行传输，每个时钟周期传输一个比特，这种传输方 式相对比较简单，速度较慢，但是使用总线数较少，通常一根接收线，一根发送线即可实现串行通信。它的缺点是要增加额外的数据来控制一个数据帧的开始和结 束。

并行通信好比一排横队，齐头并进同时传输。这种通信方式每个时钟周期传输的数据量和其总线宽度成正比，但是实现较为复杂。UART通信采用的是串行方式进行通信的。

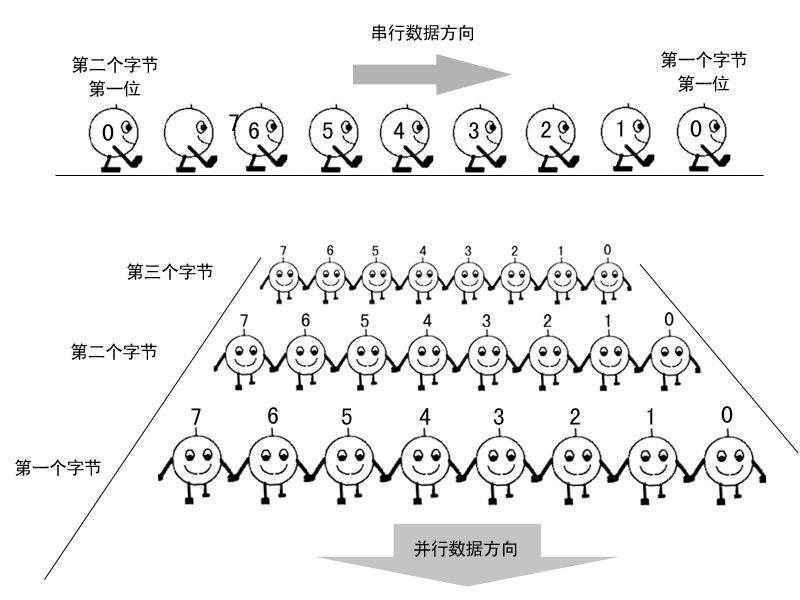


图2-57串行数据通信与并行数据通信

**1.1.3  数据通信传输模式**

在数据通信过程中，发送方和接收方为了实现数据的正确发送和接收，通常会有一个状态寄存器来描述当前数据接收和发送状态，当发送方有数据发送时，会 查看发送状态寄存器，看是否允许发送数据（如果上一次数据还没有发送完毕，不允许继续数据发送），在发送允许情况下再送出新数据。同样，接收端通过查看接 收状态寄存器，确定是否有新数据到达，如果有数据到达，将去接收数据缓冲区读取数据。

（1）轮询模式

通过程序执行流，不停的检测状态寄存器的结果，如果当前可发送或接收，则发送或接收数据。其过程可以用下面伪代码来表示。

;轮询方式实现数据发送伪代码

Send(){

While(1){

                   if(发送状态 ==可发送)

                            执行数据发送操作;

          }

}

;轮询方式实现数据接收伪代码

Receive(){

While(1){

                   if(接收状态 ==有数据到达)

                            执行数据接收操作;

          }

}

由程序可知，这种方式实现简单，但在进行数据接收和发送时都要进入循环检查状态寄存器的值，当没有数据到达或数据不可发送时，CPU会一直空转，其它程序又得不到CPU的执行权，很影响系统的效率。

（2）中断模式

中断方式是指，当数据到达或数据可发送时，产生中断，通知CPU去发送或接收数据，这种方式将通信硬件和CPU独立出来，通信硬件只有在发送或接收 条件准备好之后中，才通知CPU去处理数据，在通信条件没有准备好的时候，CPU去处理其它程序，显然这种方式更合理，这种方式要求通信硬件要求比较高， 需要支持产生中断信号。

（3）DMA模式

通常实现数据的转移或拷贝时，CPU将从源地址处复制数据到寄存器，然后将寄存器数据再写入目的地址处，该复制过程需要CPU来执行。 S3C2440支持DMA传输模式，DMA传输是指在CPU不干涉的情况下，DMA硬件自动实现数据的转移和复制，在DMA传输过程中，CPU几乎不用干 涉，这样可以让CPU安心的去做自己的事情。虽然如此，但是DMA在传输数据过程中要占用总线，在大批数据传输时，系统总线会被DMA通道占用，也会影响 系统的效率。S3C2440 UART控制器支持DMA方式传输串口通信数据。

**1.1.4  S3C2440 UART控制器**

S3C2440 UART控制器，提供了三个独立的异步串行I/O端口，每个端口都可以在中断模式或DMA模式下工作，换而言之，UART可以生成中断或DMA请求用于 CPU和UART之间的数据传输。UART串口挂接在APB总线上，APB总线最高可以达到50MHz工作频率，在使用APB时钟频率时可以达到最高 115.2Kbps波特率的通信速度。如果UART串口接收外部设备提供UEXTCLK（外部时钟），UART可以在更高的速度下工作。每个UART串口 在接收装置和发送装置里分别包含一个64Byte的FIFO缓冲区，用于缓存发送数据和接收数据。

由于UART是串行异步通信方式，因此在UART通信过程中每次只能传输1位（bit）,若干位组成一个数据帧（frame），帧是UART通信中最基本单元，它主要包含：开始位，数据位，校验位（如果开启了数据校验，要包含校验位），和停止位，帧结构如图2-58所示。

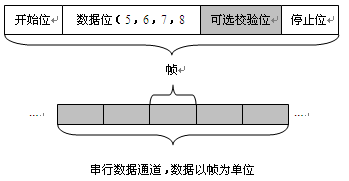


图2-58 UART数据帧结构

UART在通信之前要在发送端和接收端约定好帧结构，也就是约定好传输数据帧格式。

l 开始位：必须包含在数据帧中，表示一个帧的开始。

l 数据位：可选5，6，7，8位，该位长度可由编程人员指定。

l 校验位：如果在开启了数据校验时，该位必须指定。

l 停止位：可选1，2位，该位长度可由编程人员指定。

通信双方约定好帧格式后，指定同一波特率，以保证双方数据传输的同步。

**1.1.5  S3C2440 UART串口工作原理**

每个UART包含一个波特率产生器，发送器，接收器和一个控制单元，如下图所示：

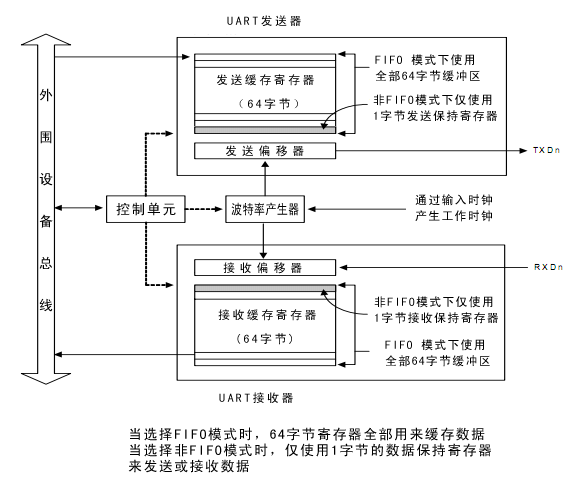


图2-59 UART硬件结构

UART是以异步方式实现通信的，其采样速度由波特率决定，波特率产生器的工作频率可以由PCLK（外围设备频率），FCLK/n（CPU工作频率 的分频），UEXTCLK（外部输入时钟）三个时钟作为输入频率，波特率设置寄存器是可编程的，用户可以设置其波特率决定发送和接收的频率。发送器和接收 器包含了64Byte的FIFO和数据移位器。UART通信是面向字节流的，待发送数据写到FIFO之后，被拷贝到数据移位器（1字节大小）里，数据通过 发送数据管脚TXDn发出。同样道理，接收数据通过RXDn管脚来接收数据（1字节大小）到接收移位器，然后将其拷贝到FIFO接收缓冲区里。

（1）数据发送

发送的数据帧可编程的，它的一个帧长度是用户指定的，它包括一个开始位，5~8个数据位，一个可选的奇偶校验位和1~2个停止位，数据帧格式可以通 过设置ULCONn寄存器来设置。发送器也可以产生一个终止信号，它是由一个全部为0的数据帧组成。在当前发送数据被完全传输完以后，该模块发送一个终止 信号。在终止信号发送后，它可以继续通过FIFO（FIFO）或发送保持寄存器（NON-FIFO）发送数据。

（2）数据接收

同样接收端的数据也是可编程的，接收器可以侦测到溢出错误奇偶校验错误，帧错误和终止条件，每个错误都可以设置一个错误标志。

l 溢出错误是指在旧数据被读取到之前，新数据覆盖了旧数据

l 奇偶校验错误是指接收器侦测到了接收数据校验结果失败，接收数据无效

l 帧错误是指接收到的数据没有一个有效的停止位，无法判定数据帧结束

l 终止条件是指RxDn接收到保持逻辑0状态持续长于一个数据帧的传输时间

（3）自动流控AFC（Auto Float Control）

UART0和UART1支持有nRTS和nCTS的自动流控，UART2不支持流控。在AFC情况下，通信双方nRTS和nCTS管脚分别连接对方的nCTS和nRTS管脚。通过软件控制数据帧的发送和接收。

在开启AFC时，发送端接收发送前要判断nCTS信号状态，当接收到nCTS激活信号时，发送数据帧。该nCTS管脚连接对方nRTS管脚。接收端 在准备接收数据帧前，其接收器FIFO有大于32个字节的空闲空间，nRTS管脚会发送激活信号，当其接收FIFO小于32个字节的空闲空间，nRTS必 须置非激活状态。如图2-60所示。

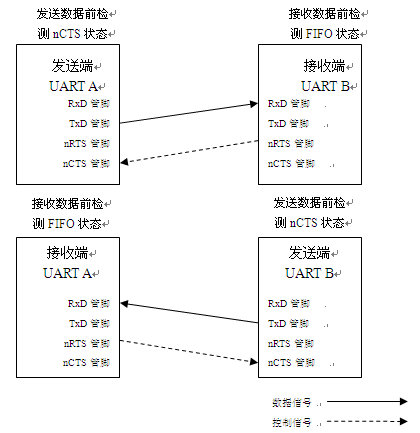


图2-60自动流控数据传输

（4）波特率

在UART中波特率发生器为发送器和接收器提供工作时钟。波特率发生器的时钟源可以选择S3C2440A的内部系统时钟(PCLK，FCLK/n) 或UEXTCLK（外部时钟源），可以通过设置UCONn寄存器来设置波特率发生器的输入时钟源。通常我们选择使用PCLK作为UART工作时钟。

UART控制器中没有对波特率进行设置的寄存器，而是通过设置一个除数因子，来决定其波特率。其计算公式如下：

UART除数（UBRDIVn） = (int)(CLK/(buad rate \* 16 )) - 1

其中：UBRDIVn的取值范围应该为1~2^16-1。例如：波特率为115200bps，PCLK时钟为其工作频率，采用50MHz，UBRDIVn为：

UBRDIVn = (int)(50M / (115200 x 16)) - 1 = 26

在系统时钟未初始化时，PCLK = 12MHz，如果波特率采用57600bps，那么UBRDIVn为：

UBRDIVn = (int)(12M / (57600 x 16)) - 1 = 12

当使用外部时钟源时，如果外部时钟小于PCLK时钟，则UEXTCLK应该设置为0。

（5）波特率的错误容忍率（Baud-Rate Error Torlerance）

数据信号在传输过程中由于外界电磁干扰，信号减弱等原因，当时钟频率较低，传输速率较高时会产生误差，当误差达到一定值时，会出现数据信号不能正常 识别，造成通信异常。好比如，在普通列车轨道上试图行驶高速列车一样，由于高速列车对轨道要求很高，当速度达到一定程度，很可能造成事故。业界的波特率的 错误容忍率为1.86%(3 / 160)，如果大于该值则应该选择较低的波特率或提高输入时钟频率。

错误容忍率计算公式为：

UART Error = (tUPCLK - tUEXACT)/ tUEXACT \* 100%

注： tUPCLK为UART的真实工作时钟频率：tUPCLK = (UBRDIVn + 1) \* 16 \* 1Frame / PCLK

tUEXACT为UART理想工作时钟频率：tUEXACT = 1Frame / baud-rate

其中：1Frame为数据帧的长度 =开始位 +数据位 +可选校验位 +停止位

假如，波特率采用115200bps，PCLK时钟为50MHz，波特率除数因子UBRDIVn为26（通过前面UBRDIVn计算公式算出），采用1个停止位，8个数据位，无校验的8N1方式通信时，其错误容忍率为：

          tUPCLK = 27 \* 16 \* 10 / 50M = 0.0000864

          tUEXACT = 10 / 115200 = 0.0000868

          UART Error  = | 0.0000864 - 0.0000868 | / 0.0000868 = 0.46%

在开发板没有初始化系统时钟前，开发板工作在12MHz下，假如我们将波特率设置为115200bps，采用PCLK为系统默认时钟12MHz，8N1数据帧格式通信，那么：

UBRDIVn = (int)(12M / (115200 \* 16)) - 1 = 6

其错误容忍率：

                  tUPCLK = 7 \* 16 \* 10 / 12M = 0.0000933

          tUEXACT = 10 / 115200 = 0.0000868

          UART Error = | 0.0000933 - 0.0000868 | / 0.0000868 = 7.5%

其错误容忍率大于1.86%，因此在12MHz频率下，波特率不能设置为115200，现在将波特率设置为56700bps，采用8N1数据帧格式通信，那么：

          UBRDIVn = (int)(12M / (57600 \* 16)) - 1 = 12

          tUPCLK = 13 \* 16 \* 10 / 12M = 0.000173

          tUEXACT = 10 / 57600 = 0.0001736

          UART Error = | 0. 000173 - 0. 0001736 | / 0. 0001736 = 0.345%

采用波特率为56700bps，8N1数据帧格式通信时，其错误容忍率小于标准的1.86%，因此可以正常工作。

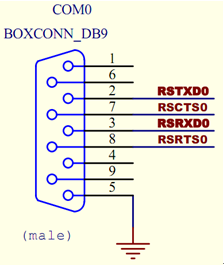


图2-61 MINI2440开发板串口硬件图

UART的接口

图2-61为MINI2440开发板引出UART串口接线图，它采用DB9接口公头（有接线柱的端口，只有接线孔的为母头），其有9根信号 线，UART通信过程中用到了信号线2 RSTXD0（数据发送管脚）它和串口线母头TXDx信号线相接（x代表0号，1号，2号串口），信号3 RSRXD0（数据接收管脚）和串口线母头RXDx相接（x代表0号，1号，2号串口），信号线5（接地管脚），信号线7 RSCTS0（数据发送流控制管脚）和串口线母头nCTSx相接，信号线8 RSRTS0（数据接收流控制管脚）和串口线母头nRTSx相接。如果UART中没有开启AFC流控的话，只要用到信号线2，信号线3和信号线5。



图2-62 MINI2440串口管脚接线

通过MINI2440硬件CPU管脚图可以看出，RSTXD0和RSRXD0连接到CPU的GPH2和GPH3管脚上的，而GPH2和GPH3是CPU复用管脚，因此我们要对GPH2和 GPH3对应寄存器进行设置，其对应寄存器为GPHCON。

表2-23 GPIO端口H设置寄存器（GPHCON）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **寄存器名** | **地址** | **是否读写** | **描述** | **复位默认值** |
| GPHCON | 0x56000070 | R/W | GPIO端口H配置寄存器 | 0x0 |
| GPHDAT | 0x56000074 | R/W | GPIO端口H数据寄存器 | 未定义 |
| GPHUP | 0x56000078 | R/W | GPIO端口H上拉无效寄存器 | 0x000 |

|  |  |  |  |
| --- | --- | --- | --- |
| **GPHCON** | **位** | **描述** | **初始值** |
| … | … | … | … |
| GPH3 | [7:6] | 设置当前管脚功能：  00 = 输入端口 01 = 输出端口  10 = RXD[0]配置为串口0的接收数据管脚  11 = 保留 | 0 |
| GPH2 | [5:4] | 设置当前管脚功能：  00 = 输入端口 01 = 输出端口  10 = RXD[0]配置为串口0的接收数据管脚  11 = 保留 | 0 |
| … | … | … | … |

GPHCON[7:6]和GPHCON[5:4]为RSTXD0和RSRXD0管脚设置位，将其功能设置为了UART专用通信管脚，因此应该设置其为0b10，分别用于UART数据的接收和发送。

GPHCON | = 0xa0;

表2-24 GPIO端口H上拉电阻设置寄存器（GPHUP）

|  |  |  |  |
| --- | --- | --- | --- |
| **GPHUP** | **位** | **描述** | **初始值** |
| GPH[10:0] | [10:0] | 设置对应管脚GPHn的是否启用上拉功能  0 = 启用上拉功能  1 = 禁用上拉功能 | 0 |

GPHUP上拉电阻设置寄存器：上拉电阻用来稳定电平信号，保障传输数据的正确，GPHUP里设置其内部上拉。

GPHUP = 0x0;

表2-25 UART0串行控制寄存器（ULCON0）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **寄存器名** | **地址** | **是否读写** | **描述** | **复位默认值** |
| ULCON0 | 0x50000000 | R/W | 串口0串行控制寄存器 | 0x00 |

|  |  |  |  |
| --- | --- | --- | --- |
| **ULCON0** | **位** | **描述** | **初始值** |
| 保留 | [7] |  | 0 |
| 红外模式 | [6] | 选择串口0是否使用红外模式：  0 =正常通信模式  1 =红外通信模式 | 0 |
| 校验模式 | [5:3] | 设置串口0在数据接收和发送时采用的校验方式：  0xx =无校验  100 =奇校验  101 =偶校验  110 =强制校验/检测是否为1  111 =强制校验/检测是否为0 | 000 |
| 停止位 | [2] | 设置串口0停止位数：  0 =每个数据帧一个停止位  1 =每个数据帧二个停止位 | 0 |
| 数据位 | [1:0] | 设置串口0数据位数：  00 = 5个数据位 01 = 6个数据位  10 = 7个数据位 11 = 8个数据位 | 00 |

通过设置ULCON0来设置UART0通信方式，ULCON0[6]选择通信方式为一般通信模式或红外通信模式，ULCON0[5:3]设置串口0校验方式，ULCON0[2]设置串口0停止位数，ULCON0[1：0]设置串口0的数据位数。

我们选择一般通信模式，无校验位，1个停止位，8个数据位的数据通信方式。因此：

ULCON0 = 0x03;

表2-26 UART0串口控制寄存器（UCON0）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **寄存器名** | **地址** | **是否读写** | **描述** | **复位默认值** |
| UCON0 | 0x50000004 | R/W | 串口0控制寄存器 | 0x00 |

|  |  |  |  |
| --- | --- | --- | --- |
| **UCON0** | **位** | **描述** | **初始值** |
| FCLK分频因子 | [15:12] | 当UART0选择FCLK作为时钟源时，设置其FCLK的分频因子  UART0工作时钟频率 = FCLK/ FCLK分频因子 + 6 | 0000 |
| UART时钟源选择 | [11:10] | 选择UART0的工作时钟PCLK，UEXTCLK，FCLK/n：  00,10 = PCLK  01 = UEXTCLK  11 = FCLK/n  当选择FCLK/n作为UART0工作时钟时还要做其它设置，具体请读者自行查看硬件手册 | 00 |
| 发送数据中断  产生类型 | [9] | 设置UART0中断请求类型，在非FIFO传输模式下，一旦发送数据缓冲区为空，立即产生中断信号，在FIFO传输模式下达到发送数据触发条件时立即产生中断信号：  0 =脉冲触发  1 =电平触发 | 0 |
| 接收数据中断  产生类型 | [8] | 设置UART0中断请求类型，在非FIFO传输模式下，一旦接收到数据，立即产生中断信号，在FIFO传输模式下达到接收数据触发条件时立即产生中断信号：  0 =脉冲触发  1 =电平触发 | 0 |
| 接收数据超时 | [7] | 设置当接收数据时，如果数据超时，是否产生接收中断：  0 =不开启超时中断 1 = 开启超时中断  10 = 7个数据位 11 = 8个数据位 | 0 |
| 接收数据错误中断 | [6] | 设置当接收数据时，如果产生异常，如传输中止，帧错误，校验错误时，是否产生接收状态中断信号：  0 =不产生错误状态中断 1 = 产生错误状态中断 | 0 |
| 回送模式 | [5] | 设置该位时UART会进入回送模式，该模式仅用于测试  0 =正常模式 1 = 回送模式 | 0 |
| 发送终止信号 | [4] | 设置该位时，UART会发送一个帧长度的终止信号，发送完毕后，该位自动恢复为0  0 =正常传输 1 = 发送终止信号 | 0 |
| 发送模式 | [3:2] | 设置采用哪个方式执行数据写入发送缓冲区  00 =无效  01 =中断请求或查询模式  10 = DMA0请求 | 00 |
| 接收模式 | [1:0] | 设置采用哪个方式执行数据写入接收缓冲区  00 =无效  01 =中断请求或查询模式  10 = DMA0请求 | 00 |

通常UART串口采用PCLK作为输入工作时钟，采用简单的轮询方式进行数据接收和发送，不开启数据接收超时，数据产生错误时不产生错误状态中断，因此：

UCON0 = 0x05;

表2-27 UART FIFO控制寄存器（UFCON0）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **寄存器名** | **地址** | **是否读写** | **描述** | **复位默认值** |
| UFCON0 | 0x50000008 | R/W | 串口0 FIFO控制寄存器 | 0x00 |

|  |  |  |  |
| --- | --- | --- | --- |
| **UFCON0** | **位** | **描述** | **初始值** |
| 发送数据  触发级别 | [7:6] | 设置FIFO发送模式的触发级别：  00 = FIFO为空触发  01 = 16字节触发  10 = 32字节触发    11 = 48字节触发 | 00 |
| 接收数据  触发级别 | [5:4] | 设置FIFO接收模式的触发级别：  00 = FIFO为空触发  01 = 16字节触发  10 = 32字节触发    11 = 48字节触发 | 00 |
| 保留 | [3] |  | 0 |
| 发送FIFO重置 | [2] | 在重置FIFO后自动清除发送缓冲区  0 =正常模式      1 = 自动清除 | 0 |
| 接收FIFO重置 | [1] | 在重置FIFO后自动清除接收缓冲区  0 =正常模式      1 = 自动清除 | 0 |
| 启用FIFO | [0] | 0 =不启用FIFO    1 = 启用FIFO | 0 |

表2-28 UART MODEM控制寄存器（UMCON0）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **寄存器名** | **地址** | **是否读写** | **描述** | **复位默认值** |
| UMCON0 | 0x5000000C | R/W | 串口0MODEM控制寄存器 | 0x00 |

|  |  |  |  |
| --- | --- | --- | --- |
| **UMCON0** | **位** | **描述** | **初始值** |
| 保留 | [7:5] | 必须全部置0 | 000 |
| AFC自动流控 | [4] | 0 =不开启流控    1 = 开启流控 | 0 |
| 保留 | [3:1] | 必须全部置0 | 000 |
| 请求发送 | [0] | 如果启用AFC，该位无效，S3C2440会自动控制nRTS，如果不启用AFC，nRTS必须由软件控制  0 =高电平激活nRTS     1 = 低电平激活nRTS | 0 |

表2-29 UART发送/接收状态寄存器（UTRSTAT0）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **寄存器名** | **地址** | **是否读写** | **描述** | **复位默认值** |
| UTRSTAT0 | 0x50000010 | R/W | 串口0发送/接收状态寄存器 | 0x06 |

|  |  |  |  |
| --- | --- | --- | --- |
| **UTRSTAT0** | **位** | **描述** | **初始值** |
| 发送器为空 | [2] | 当发送缓存寄存器中没有数据要发送且发送移位寄存器为空时，自动置1  0 =非空  1 =发送器为空（发送缓存和移位寄存器） | 1 |
| 发送缓存寄存器为空 | [1] | 当发送缓存寄存器为空时，自动置1  0 =发送缓存寄存器非空  1 =发送缓存寄存器为空 | 1 |
| 接收缓存寄存器为空 | [0] | 当接收缓存寄存器有数据到达时，自动置1  0 =接收缓存寄存器为空  1 =缓存寄存器接收数据 | 0 |

表2-30 UART发送缓存寄存器（UTXH0）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **寄存器名** | **地址** | **是否读写** | **描述** | **复位默认值** |
| UTXH0 | 0x50000020(L)  0x50000023(B) | W | 串口0发送缓存寄存器 | - |

表2-31 UART接收缓存寄存器（URXH0）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **寄存器名** | **地址** | **是否读写** | **描述** | **复位默认值** |
| URXH0 | 0x50000024(L)  0x50000027(B) | R | 串口0接收缓存寄存器 | - |

表2-32 UART波特率除数寄存器（UBRDIV0）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **寄存器名** | **地址** | **是否读写** | **描述** | **复位默认值** |
| UBRDIV0 | 0x50000028 | R/W | 串口0波特率除数寄存器 | - |

|  |  |  |  |
| --- | --- | --- | --- |
| **UBRDIV0** | **位** | **描述** | **初始值** |
| 波特率除数 | [15:0] | 设置波特率除数（大于0）使用外部输入时钟时可以置0 | - |

上述寄存器是是和UART通信相关寄存器，使用简单的无FIFO，无自动流控AFC时，设置如下：

UFCON0 = 0x00;       // 不使用FIFO

UMCON0 = 0x00;      // 不使用流控

UBRDIV0 = 26;           // 波特率为115200,PCLK=50Mhz

UBRDIV0 = 53;           // 波特率为57600,PCLK=50Mhz

UBRDIV0 = 12;          // 波特率为57600,PCLK=12Mhz

UTXH0和URXH0分别是数据发送和接收寄存器，发送数据时通过轮询方式判断发送状态寄存器的状态，当可以发送数据时，执行UTXH0寄存器写入操作，接收数据时，以轮询方式检测接收状态寄存器状态，当有数据到达时，读取URXH0寄存器里的数据即可取得串口数据。

#define TXD0READY   (1<<2)  //发送数据状态OK

#define RXD0READY   (1)     //接收数据状态OK

/\* UART串口单个字符打印函数 \*/

extern void putc(unsigned char c)

{

    while( ! (UTRSTAT0 & TXD0READY) );

    UTXH0 = c;

}

/\* UART串口接受单个字符函数 \*/

extern unsigned char getc(void)

{

    while( ! (UTRSTAT0 & RXD0READY) );

    return URXH0;

}

**1.1.6  UART串口驱动实验**

init.s：本程序文件对看门狗，内存等基本硬件做初始化，然后跳入到xmain.c中的xmain函数执行。

;

; UART串口实验

;

GPBCON            EQU      0x56000010

GPBDAT            EQU      0x56000014

        AREA Init, CODE, READONLY

ENTRY

start

; close watchdog

        ldr r0, = 0x53000000   ; 将看门狗控制寄存器地址放入r0

          mov r1, #0

        str r1, [r0]                     ; 设置看门狗控制寄存器的值为0

        bl initmem                              ; 跳转到initmem代码段，初始化内存

        IMPORT xmain                     ; 引入main.c中的xmain函数

        ldr sp, =0x34000000             ; 调用C程序之前先初始化栈指针

        ldr lr, =loop                            ; 设置xmain函数的返回地址

        ldr pc, =xmain                        ; 跳转到C程序中的xmain函数的入口处执行

loop

    b loop                                               ; 死循环

initmem                                                     ; 内存初始化

        ldr r0, =0x48000000              ; 加载内存相关寄存器首地址r0

        ldr r1, =0x48000034              ; 加载内存相关寄存器尾地址到r1

        adr r2, memdata                     ; 将寄存器配置数据地址段首地址加载到r2

initmemloop

        ldr r3, [r2], #4               ; 循环设置存寄存器

          str r3, [r0], #4

          teq r0, r1

        bne initmemloop                    ; 循环到最后一个寄存器时退出函数

          mov pc,lr

memdata

          DCD 0x22000000                ;BWSCON

          DCD 0x00000700                 ;BANKCON0

          DCD 0x00000700                ;BANKCON1

          DCD 0x00000700                 ;BANKCON2

          DCD 0x00000700     ;BANKCON3

          DCD 0x00000700                 ;BANKCON4

          DCD 0x00000700                 ;BANKCON5

          DCD 0x00018005                 ;BANKCON6

          DCD 0x00018005                ;BANKCON7

          DCD 0x008e07a3                  ;REFRESH

          DCD 0x000000b1                 ;BANKSIZE

          DCD 0x00000030                 ;MRSRB6

          DCD 0x00000030                 ;MRSRB7

          END

xmain.c：uart\_init函数对UART0进行初始化，然后进入死循环内，不停打印字符串“Uart串口打印试验”。

/\* xmain.c \*/

/\* GPIO registers \*/

#define       GPHCON                              (\*(volatile unsigned long \*)0x56000070)

#define       GPHDAT                              (\*(volatile unsigned long \*)0x56000074)

#define       GPHUP                                 (\*(volatile unsigned long \*)0x56000078)

/\*UART registers\*/

#define       ULCON0                               (\*(volatile unsigned long \*)0x50000000)

#define       UCON0                                 (\*(volatile unsigned long \*)0x50000004)

#define       UFCON0                            (\*(volatile unsigned long \*)0x50000008)

#define       UMCON0                             (\*(volatile unsigned long \*)0x5000000c)

#define       UTRSTAT0                           (\*(volatile unsigned long \*)0x50000010)

#define       UTXH0                                (\*(volatile unsigned char \*)0x50000020)

#define       URXH0                               (\*(volatile unsigned char \*)0x50000024)

#define       UBRDIV0                             (\*(volatile unsigned long \*)0x50000028)

#define       TXD0READY    (1<<2)  //发送数据状态OK

#define       RXD0READY   (1)              //接收数据状态OK

/\* UART串口初始化 \*/

void uart\_init( )

{

        GPHCON |= 0xa0;                //GPH2,GPH3 used as TXD0,RXD0

          GPHUP     = 0x0;                //GPH2,GPH3内部上拉

        ULCON0  = 0x03;             //8N1

        UCON0     = 0x05;            //查询方式为轮询或中断;时钟选择为PCLK

        UFCON0 = 0x00;                 //不使用FIFO

        UMCON0 = 0x00;                //不使用流控

        UBRDIV0 = 12;                   //波特率为57600,PCLK=12Mhz

}

/\* UART串口单个字符打印函数 \*/

extern void putc(unsigned char c)

{

        while( ! (UTRSTAT0 & TXD0READY) );

        UTXH0 = c;

}

/\* UART串口接受单个字符函数 \*/

extern unsigned char getc(void)

{

        while( ! (UTRSTAT0 & RXD0READY) );

        return URXH0;

}

/\* UART串口字符串打印函数 \*/

extern int printk(const char\* str)

{

        int i = 0;

        while( str[i] ){

                   putc( (unsigned char) str[i++] );

        }

        return i;

}

\_\_inline void delay(int msec)

{

        int i, j;

        for(i = 1000; i > 0; i--)

                   for(j = msec\*10; j > 0; j--)

                            /\* do nothing \*/;

}

/\* xmain通过UART串口打印字符串 \*/

int xmain()

{

        uart\_init();

        while(1) {

               delay(10);

               printk("Uart串口打印试验/r/n");

      }

      return 0;

}

当编译并将生成Norflash，在->所有程序->通讯mini2440，在之后弹出的COM波特率1个停止位，PC串口（笔记本通常没有串口，可以买一个Uart串口打印试验”字符串。