USB 总线转接芯片 CH341

手册（二）：并口及同步串口说明

版本：3C

https://wch.cn

1、并口功能说明

1.1. 一般说明

CH341 的并口是主动式并口，在计算机端的程序控制下，可以直接从外部电路输入输出数据，一般不需要外接单片机/DSP/MCU。

CH341 的并口主要有 2 种接口方式：EPP 方式和 MEM 方式。EPP 方式类似于 EPP V1.7 或者 EPP V1.9 规范，MEM 方式类似于 INTEL 时序 SRAM 存储器的读写方式。芯片复位后的默认方式是 EPP，在 USB 配置完成后，计算机端的程序可以随时控制 CH341 在上述 2 种方式之间进行切换。另外还有一种 BUS 方式，以地址和数据复用总线的方式提供 7 位地址和 8 位数据。

CH341B 和 CH341F 芯片支持批量化功能程序定制。被动并口请参考高速接口芯片 CH346 手册。

1.2. EPP 并口

EPP 并口的主要引脚包括 WR#引脚、DS#引脚、AS#引脚、WAIT#引脚，相关信号的时序说明可以参照 EPP 规范 V1.7 和 V1.9。

EPP 方式通过 WR#、DS#和 AS#的逻辑组合执行具体操作。WR#用于指示当前的数据或地址传输方向，对计算机端而言，高电平是对外部电路执行读操作，低电平是对外部电路执行写操作。选通信号是低电平有效的脉冲信号，选通信号包括数据选通 DS#和地址选通 AS#，DS#有效执行数据操作，AS#有效执行地址操作。EPP 的实际操作发生于选通信号有效期间，例如：在 WR#为高电平期间 DS#输出脉冲，则执行一个数据读操作；在 WR#为低电平期间 AS#输出脉冲，则执行一个地址写操作。

CH341B/F/A 的 EPP 数据读写操作 DS#支持 WAIT#等待信号，在 CH341 开始输出低电平选通信号后，如果 WAIT#为低电平，那么选通信号将继续保持低电平直到 WAIT#恢复为高电平或者 85uS 等待超时后才结束输出。

CH341B/F/A 的 EPP 地址读写操作 AS#不支持 WAIT#等待信号，所以 EPP 地址读写操作比数据读写操作略快一些。

写操作的选通信号的低电平有效宽度最小是 0.16uS 或者 0.25uS，读操作的选通信号的低电平有效宽度最小是 0.25uS 或者 0.33uS，理想状态下的最大传输速度是 800KB/S，在 WINDOWS XP SP2 环境下使用连续的大数据块进行速度测试，实测传输速度约为：下传 510KB/S，上传 560KB/S。

1.3. MEM 并口

MEM 并口的主要引脚包括 WR#引脚、RD#引脚（DS#引脚的别名）、A0 引脚（AS#引脚的别名）、WAIT#引脚。

MEM 方式类似于存储器的读写方式，WR#和 RD#都是低电平有效的脉冲信号。MEM 的实际操作发生于 WR#或者 RD#有效期间，对计算机端而言，当 WR#有效时对外部电路执行写操作，当 RD#有效时对外部电路执行读操作。A0 用于指示当前读写操作的地址，例如：将 A0 和 A0 的反相分别用于两个外部设备的片选；或者将 A0=1 时的操作指向外部设备的命令端口，而将 A0=0 时的操作指向数据端口。

CH341B/F/A 的 MEM 读写操作支持 WAIT#等待信号。

WR#的低电平有效宽度最小是 0.25uS，RD#的低电平有效宽度最小是 0.33uS，理想状态下的最大传输速度是 800KB/S。实测传输速度与 EPP 数据读写差不多，但略低于 EPP 地址读写操作的速度。

1.4. BUS 并口

BUS 并口的主要引脚包括 WR#引脚、RD#引脚（DS#引脚的别名）、ALE 引脚（AS#引脚的别名）、WAIT#

引脚。

BUS 方式相当于更多地址线的 MEM 方式，WR#和 RD#都是低电平有效的脉冲信号，在每次读写脉冲发生之前，数据总线 D0-D7 先输出 7 位将要操作的目标地址，接着 ALE 信号输出高电平，使外部的地址锁存器电路（例如 74LS373）将 7 位地址锁存后输出，用于外部设备的片选，然后，WR#或 RD#其中之一输出有效的读写脉冲。BUS 的实际读写操作发生于 WR#或者 RD#有效期间，对计算机端而言，当 WR#有效时对外部电路执行写操作，当 RD#有效时对外部电路执行读操作。

CH341B/F/A 的 BUS 读写操作支持 WAIT#等待信号。

WR#的低电平有效宽度最小是 0.33uS，RD#的低电平有效宽度最小是 0.51uS，理想状态下的最大传输速度是 300KB/S，但在进行多个 I/O 地址的独立读写操作时速度可能只有 1K/S。

1.5. 辅助引脚

辅助引脚包括 RST#引脚和 INT#引脚，以及 ERR#、SLCT、PEMP 等引脚。

RST#引脚是复位输出引脚，当其为低电平时，说明 CH341 芯片正在复位或者计算机端的程序要求复位外部电路。INT#引脚是中断请求输入引脚，当其检测到上升沿时，计算机端的程序将会收到中断通知。其它引脚都是自定义的通用输入引脚，计算机端的应用程序可以查询其引脚状态。

2、同步串口功能说明

2.1. 一般说明

CH341 的同步串口是主动式串口，只能作为 Host/Master 主机端，在计算机端的程序控制下，可以直接从外部电路输入输出数据，一般不需要外接单片机/DSP/MCU。

CH341B/F/A 采用 FlexWire 技术，通过计算机相关程序控制进行组合可以实现：2 线串口、4 线串口、5 线串口，其中以 2 线串口及 4 线串口较为常用。高速或被动 SPI 请参考 CH347 和 CH346 手册。

2.2. 2 线串口

2 线串口的主要引脚包括 SCL 引脚、SDA 引脚。SCL 用于单向输出同步时钟，开漏输出且内置上拉电阻，SDA 用于准双向数据输入输出，开漏输出及输入且内置上拉电阻。

2 线串口的基本操作元素包括：起始位、停止位、位输出、位输入。

起始位定义为当 SCL 为高电平时，SDA 输出下降沿（从高电平切换为低电平）。

停止位定义为当 SCL 为高电平时，SDA 输出上升沿（从低电平切换为高电平）。

位输出定义为当 SCL 为低电平时，SDA 输出位数据，然后 SCL 输出高电平脉冲。

位输入定义为 SCL 输出高电平脉冲，在下降沿之前从 SDA 输入位数据。

字节输出定义为 8 个位输出及 1 个位输入用于应答。

字节输入定义为 8 个位输入及 1 个位输出用于应答。

2 线串口的数据输入和输出以字节为单位，每个字节含 8 个位，高位在前。

CH341 的 2 线串口支持两线串口的 A/D、D/A、存储器及 I/O 扩展芯片。例如，常见的 24C 系列串行 EEPROM：24C01A 到 24C16、24C32 到 24C1024 等。

2.3. 4 线串口

4 线串口的主要引脚包括 DCK 引脚、DIN 引脚、DOUT 引脚、片选引脚 CS0、CS1、CS2。DCK 用于单向输出同步时钟，DIN 用于单向输入数据，DOUT 用于单向输出数据，片选引脚 CSn 用于选择设备。

4 线串口的基本操作元素包括：片选选中、片选结束、位输出、位输入。

片选选中定义为片选引脚 CSn 输出有效电平（可以定义为高电平或低电平）。

片选结束定义为片选引脚 CSn 输出非有效电平。

位输出定义为当 DCK 为低电平时，DOUT 输出位数据，然后 DCK 输出高电平脉冲。

位输入定义为当 DCK 为高电平时，从 DIN 输入位数据。

字节输出定义为 8 个位输出，字节输入定义为 8 个位输入。

4 线串口的数据输入和输出以字节为单位，每个字节含 8 个位，支持低位在前和高位在前。

2.4. 5 线串口

5 线串口的主要引脚包括 DCK 引脚、DIN 引脚、DIN2 引脚、DOUT 引脚、DOUT2 引脚、片选引脚 CSO、CS1、CS2。DCK 用于单向输出同步时钟，DIN 和 DIN2 用于单向输入数据，DOUT 和 DOUT2 用于单向输出数据，片选引脚 CSn 用于选择外部设备。

5 线串口是 4 线串口中增加一组数据输入 DIN2 和一组数据输出 DOUT2 的版本。

5 线串口的基本操作元素包括：片选选中、片选结束、双位输出、双位输入。

双位输出定义为当 DCK 为低电平时，DOUT 和 DOUT2 输出位数据，然后 DCK 输出高电平脉冲。一个字节 8 位数据，高 4 位从 DOUT 输出，低 4 位从 DOUT2 输出。

双位输入定义为当 DCK 为高电平时，从 DIN 和 DIN2 输入位数据。一个字节 8 个数据，高 4 位来自 DIN 输入，低 4 位来自 DIN2 输入。

字节输出定义为一个字节 8 位分为两组各 4 位分别输出。

字节输入定义为分别输入两组各 4 位组合成一个字节。

5 线串口的数据输入和输出以半字节为单位，每个半字节含 4 个位，支持低位在前和高位在前。要分别在 DOUT 和 DOUT2 产生字节输出，那么需要两个字节输出组合。

CH341 的 5 线串口用于仿真具有较多 I/O 的同步串行接口。

2.5. 位操作

正常情况下，CH341 的同步串口操作是以字节为基本单位的，一次操作可能是 1 个字节或者几个甚至几十个字节。而实际应用中可能需要输入或者输出非 8 倍数的数据位，例如某 A/D 采集芯片需要输入 10 位数据，为了方便这种应用，CH341 的 FlexWire（TM）技术提供了一个基本操作，能够一次输入或者输出 1 位数据，重复使用该操作能够输入 2 位数据，再加上一个字节操作，从而实现 10 位数据的输入输出。该方法仅适用于 4 线串口或者 5 线串口，能够控制 DCK、DOUT、DOUT2、CSO、CS1、CS2 引脚产生一个位输出，能够从 DCK、DIN、DIN2、DOUT、DOUT2、CSO、CS1、CS2 引脚实现一个位输入。更复杂功能建议基于 CH32X035、CH543 等 Type-C/USB 接口 MCU 通过编程实现。

2.6. 辅助引脚

同步串口应用下的辅助引脚与并口应用下的相同，请参考并口功能说明。

3、计算机端的软件

在计算机端的 Windows 操作系统下，CH341 的并口驱动程序和动态链接库 DLL 向应用程序提供了应用层接口，包括：设备管理 API、并口数据传输 API、同步串口数据传输 API、中断处理 API。

有关 API 参数的说明请参考 CH341DLL.H，主要 API 如下。

3.1. 设备管理 API

```
CH341OpenDevice(  // 打开 CH341 设备,返回句柄,出错则无效
    ULONG    iIndex );  // 指定 CH341 设备序号,0 对应第一个设备
```

将 CH341 作为设备，使用前必须先打开，然后才能使用

```
CH341CloseDevice(  // 关闭 CH341 设备
    ULONG    iIndex );  // 指定 CH341 设备序号
```

用完 CH341 后，或者应用程序退出前，应该关闭 CH341 设备

CH341SetDeviceNotify(// 设定设备事件通知程序
ULONG iIndex, // 指定 CH341 设备序号,0 对应第一个设备
PCHAR iDeviceID, // 可选参数,指向字符串,指定被监控的设备的 ID,字符串以\0 终止
mPCH341_NOTIFY_ROUTINE iNotifyRoutine); // 指定设备事件回调程序
用于应用程序监控 CH341 设备的插拔事件，确保应用程序随时知道 USB 设备是否存在，防止在 USB 设备拔出后收发数据，并及时响应 USB 设备的插入

CH341GetStatus(// 通过 CH341 直接输入数据和状态，类似的 API 还有 CH341GetInput
ULONG iIndex, // 指定 CH341 设备序号
PULONG iStatus); // 指向一个双字单元,用于保存状态数据
获取的状态数据中:位 7-位 0 对应 CH341 的 D7-D0 引脚,位 8 对应 CH341 的 ERR#引脚,位 9 对应 CH341 的 PEMP 引脚，位 10 对应 CH341 的 INT#引脚，位 11 对应 CH341 的 SLCT 引脚，位 13 对应 CH341 的 BUSY/WAIT#引脚，位 14 对应 CH341 的 AUTOFD#/DATAS#引脚，位 15 对应 CH341 的 SLCTIN#/ADDRS#引脚，位 23 对应 CH341 的 SDA 引脚

CH341SetOutput(// 设置 CH341 的 I/O 方向,并通过 CH341 直接输出数据
// 谨慎使用该 API，防止修改 I/O 方向使输入引脚变为输出导致与其它输出引脚之间短路而损坏
ULONG iIndex, // 指定 CH341 设备序号
ULONG iEnable, // 数据有效标志
ULONG iSetDirOut, // 设置 I/O 方向,位清 0 则对应引脚为输入,位置 1 则对应引脚为输出
ULONG iSetDataOut); // 输出数据,如果 I/O 方向为输出,那么位数据将通过引脚输出
上述的 I/O 方向和输出数据以 32 位数据表示，其中：位 7-位 0 对应 CH341 的 D7-D0 引脚，位 8 对应 CH341 的 ERR#引脚，位 9 对应 CH341 的 PEMP 引脚，位 10 对应 CH341 的 INT#引脚，位 11 对应 CH341 的 SLCT 引脚，位 13 对应 CH341 的 WAIT#引脚，位 14 对应 CH341 的 DATAS#/READ#引脚，位 15 对应 CH341 的 ADDRS#/ADDR/ALE 引脚
另外，以下引脚只能输出，不考虑 I/O 方向：位 16 对应 CH341 的 RESET#引脚，位 17 对应 CH341 的 WRITE#引脚，位 18 对应 CH341 的 SCL 引脚，位 29 对应 CH341 的 SDA 引脚

CH341Set_D5_D0(// 设置 CH341 的 D5-D0 引脚的 I/O 方向,并通过 D5-D0 引脚直接输出数据
// 谨慎使用该 API，防止修改 I/O 方向使输入引脚变为输出导致与其它输出引脚之间短路而损坏
ULONG iIndex, // 指定 CH341 设备序号
ULONG iSetDirOut, // 设置 D5-D0 各引脚的 I/O 方向,清 0 则引脚为输入,置 1 则引脚为输出
ULONG iSetDataOut); // 设置 D5-D0 各引脚的输出数据,仅当 I/O 方向为输出时生效

3.2. 中断处理 API

CH341SetIntRoutine(// 设定中断服务程序
ULONG iIndex, // 指定 CH341 设备序号
mPCH341_INT_ROUTINE iIntRoutine); // 指定中断服务程序,为 NULL 则取消中断服务
设置 CH341 的中断服务程序，iIntRoutine 是一个符合 mPCH341_INT_ROUTINE 格式的子程序，
当 CH341 的 INT#引脚出现上升沿时，DLL 自动调用 iIntRoutine，并向其提供一个引脚状态参数，
引脚状态参数中，位为 1 则说明对应的引脚为高电平，位为 0 则说明对应的引脚为低电平，
位 7-位 0 对应 CH341 的 D7-D0 引脚，位 8 对应 CH341 的 ERR#引脚，位 9 对应 CH341 的 PEMP 引脚，
位 10 对应 CH341 的 INT#引脚，位 11 对应 CH341 的 SLCT 引脚
例如：主程序

```
main {
    ......
    CH341OpenDevice( 0 );  // 打开设备，针对 0#设备，如果有多个，可以计数
```

```
        CH341SetIntRoutine( 0, myInterruptEvent );  // 设置中断服务程序
        ......读写数据，或者在接收到中断服务程序的通知后处理中断
        CH341CloseDevice( 0 );  // 用完后关闭设备
    }
```

中断服务程序，当 CH341 的 INT#引脚出现上升沿时，DLL 会自动调用该子程序

```
    void    CALLBACK    myInterruptEvent ( unsigned long    PinStatus ) {
        if ( PinStatus & mStateBitERR ) printf("发生中断时 ERR#引脚为高电平");
        else printf("发生中断时 ERR#引脚为低电平");
        ......自己处理或者通知主程序处理
    }
```

3.3. 并口数据传输 API

```
CH341InitParallel(  // 复位并初始化并口,RST#输出低电平脉冲
    ULONG    iIndex,  // 指定 CH341 设备序号
    ULONG    iMode );  // 指定并口模式: 0 为 EPP 模式, 2 为 MEM 模式, >=256 保持当前模式
```

在 CH341 上电时自动初始化并口，如果需要，也可以重新初始化并口，以清除缓冲区。
在初始化过程中，RST#引脚会输出 100uS 左右宽度的低电平脉冲，用于通知外部设备复位，

```
CH341EppReadData(  // EPP 方式读数据: WR#=1, DS#=0, AS#=1, D0-D7=input
    ULONG    iIndex,  // 指定 CH341 设备序号
    PVOID    oBuffer,  // 指向一个足够大的缓冲区,用于保存读取的数据
    PULONG   ioLength );  // 指向长度单元,输入时为准备读取的长度,返回后为实际读取的长度
```

以 EPP 时序连续读取数据，长度为 0 到 4096 字节，例如：

```
    UCHAR    buf[1024];
    ULONG    len=1024;
    CH341EppReadData( 0, buf, &len );  // 针对 0#设备以 EPP 方式读取 1KB 数据
```

```
CH341EppReadAddr(  // EPP 方式读地址: WR#=1, DS#=1, AS#=0, D0-D7=input
    ULONG    iIndex,  // 指定 CH341 设备序号
    PVOID    oBuffer,  // 指向一个足够大的缓冲区,用于保存读取的地址数据
    PULONG   ioLength );  // 指向长度单元,输入时为准备读取的长度,返回后为实际读取的长度
```

以 EPP 时序连续读取地址数据，长度为 0 到 4096 字节，在标准 EPP 时序中通常用不到

```
CH341EppWriteData(  // EPP 方式写数据: WR#=0, DS#=0, AS#=1, D0-D7=output
    ULONG    iIndex,  // 指定 CH341 设备序号
    PVOID    iBuffer,  // 指向一个缓冲区,放置准备写出的数据
    PULONG   ioLength );  // 指向长度单元,输入时为准备写出的长度,返回后为实际写出的长度
```

以 EPP 时序连续写出数据，长度为 0 到 4096 字节，例如：

```
    UCHAR    buf[1024];
    ULONG    len=1024;
    在 buf 中放置数据，准备以 EPP 方式写出
    CH341EppWriteData( 0, buf, &len );  // 针对 0#设备以 EPP 方式写出 1KB 数据
```

```
CH341EppWriteAddr(  // EPP 方式写地址: WR#=0, DS#=1, AS#=0, D0-D7=output
    ULONG    iIndex,  // 指定 CH341 设备序号
    PVOID    iBuffer,  // 指向一个缓冲区,放置准备写出的地址数据
    PULONG   ioLength );  // 指向长度单元,输入时为准备写出的长度,返回后为实际写出的长度
```

以 EPP 时序连续写出地址数据，长度为 0 到 4096 字节，在标准 EPP 时序中通常只写一个字节的地址

```
CH341EppSetAddr(  // EPP 方式设置地址: WR#=0, DS#=1, AS#=0, D0-D7=output
    ULONG    iIndex,  // 指定 CH341 设备序号
    UCHAR    iAddr );  // 指定 EPP 地址
```

以 EPP 时序输出一个地址，是 CH341EppWriteAddr 的简化

```
CH341MemReadAddr0(  // MEM 方式读地址 0: WR#=1, DS#/RD#=0, AS#/ADDR=0, D0-D7=input
    ULONG    iIndex,  // 指定 CH341 设备序号
    PVOID    oBuffer,  // 指向一个足够大的缓冲区,用于保存从地址 0 读取的数据
    PULONG   ioLength );  // 指向长度单元,输入时为准备读取的长度,返回后为实际读取的长度
```

以 MEM 时序连续读取数据，长度为 0 到 4096 字节，读操作期间 ADDR=0，例如：

```
    UCHAR    buf[1024];
    ULONG    len=1024;
    CH341MemReadAddr0( 0, buf, &len );  // 针对 0#设备以 MEM 方式从地址 0 读取 1K 数据
```

```
CH341MemReadAddr1(  // MEM 方式读地址 1: WR#=1, DS#/RD#=0, AS#/ADDR=1, D0-D7=input
    ULONG    iIndex,  // 指定 CH341 设备序号
    PVOID    oBuffer,  // 指向一个足够大的缓冲区,用于保存从地址 1 读取的数据
    PULONG   ioLength );  // 指向长度单元,输入时为准备读取的长度,返回后为实际读取的长度
```

以 MEM 时序连续读取数据，长度为 0 到 4096 字节，读操作期间 ADDR=1

```
CH341MemWriteAddr0(  // MEM 方式写地址 0: WR#=0, DS#/RD#=1, AS#/ADDR=0, D0-D7=output
    ULONG    iIndex,  // 指定 CH341 设备序号
    PVOID    iBuffer,  // 指向一个缓冲区,放置准备向地址 0 写出的数据
    PULONG   ioLength );  // 指向长度单元,输入时为准备写出的长度,返回后为实际写出的长度
```

以 MEM 时序连续写出数据，长度为 0 到 4096 字节，写操作期间 ADDR=0，例如：

```
    UCHAR    buf[1024];
    ULONG    len=1024;
    在 buf 中放置数据，准备以 MEM 方式写出
    CH341MemWriteAddr0( 0, buf, &len );  // 针对 0#设备以 MEM 方式向地址 0 写出 1K 数据
```

```
CH341MemWriteAddr1(  // MEM 方式写地址 1: WR#=0, DS#/RD#=1, AS#/ADDR=1, D0-D7=output
    ULONG    iIndex,  // 指定 CH341 设备序号
    PVOID    iBuffer,  // 指向一个缓冲区,放置准备向地址 1 写出的数据
    PULONG   ioLength );  // 指向长度单元,输入时为准备写出的长度,返回后为实际写出的长度
```

以 MEM 时序连续写出数据，长度为 0 到 4096 字节，写操作期间 ADDR=1

3.4. 同步串口数据传输 API

```
CH341ReadI2C(  // 从两线串口读取一个字节数据，仅适用于 7 位地址的设备
    ULONG    iIndex,  // 指定 CH341 设备序号
    UCHAR    iDevice,  // 低 7 位指定设备地址
    UCHAR    iAddr,  // 指定数据单元的地址
    PUCHAR   oByte );  // 指向一个字节单元,用于保存读取的字节数据
```

```
CH341WriteI2C(  // 向两线串口写入一个字节数据，仅适用于 7 位地址的设备
    ULONG    iIndex,  // 指定 CH341 设备序号
    UCHAR    iDevice,  // 低 7 位指定设备地址
    UCHAR    iAddr,  // 指定数据单元的地址
```

```
    UCHAR    iByte );  // 待写入的字节数据

CH341WriteRead(  // 执行数据流命令,先输出再输入
    ULONG    iIndex,  // 指定 CH341 设备序号
    ULONG    iWriteLength,  // 写长度,准备写出的长度
    PVOID    iWriteBuffer,  // 指向一个缓冲区,放置准备写出的数据
    ULONG    iReadStep,  // 准备读取的单个块的长度, 总长度为(iReadStep*iReadTimes)
    ULONG    iReadTimes,  // 准备读取的次数
    PULONG   oReadLength,  // 指向长度单元,返回后为实际读取的长度
    PVOID    oReadBuffer );  // 指向一个足够大的缓冲区,用于保存读取的数据
```

先输出数据再输入数据，执行数据流命令，适用于同步串口等。

```
CH341SetStream(  // 设置同步串口流模式
    ULONG    iIndex,  // 指定 CH341 设备序号
    ULONG    iMode );  // 指定模式,见下行
// 位 1 位 0: I2C 速度/SCL 频率, 00=低速 20KHz,01=标准 100KHz,10=快速 400KHz,11=高速 750KHz
// 位 2:      SPI 的 I/O 数/IO 引脚, 0=单入单出(4 线接口),1=双入双出(5 线接口)
// 位 7:      SPI 字节中的位顺序, 0=低位在前, 1=高位在前
// 其它保留,必须为 0

CH341StreamI2C(  // 处理两线串口的数据流，适用于所有两线串口的设备
    ULONG    iIndex,  // 指定 CH341 设备序号
    ULONG    iWriteLength,  // 准备写出的数据字节数
    PVOID    iWriteBuffer,  // 指向缓冲区,放置准备写出的数据,首字节是设备地址及读写位
    ULONG    iReadLength,  // 准备读取的数据字节数
    PVOID    oReadBuffer );  // 指向缓冲区,返回后是读入的数据
```

对两线串口设备进行操作。例如，从 24C256 中 3200H 开始的地址读出 256 字节的数据：

```
    UCHAR   OutBuf[5], InBuf[300];  // 待写数据缓冲区，读出数据缓冲区
    OutBuf[0]=0xA1; OutBuf[1]=0x32; OutBuf[2]=0x00;  // 待写数据：设备地址及单元地址
    CH341StreamI2C( 0, 3, OutBuf, 256, InBuf );  // 针对 0#设备处理两线串口的数据流
```

```
CH341ReadEEPROM(  // 从 EEPROM 中读取数据块,速度约 56K 字节
    ULONG        iIndex,  // 指定 CH341 设备序号
    EEPROM_TYPE  iEepromID,  // 指定 EEPROM 型号
    ULONG        iAddr,  // 指定数据单元的地址
    ULONG        iLength,  // 准备读取的数据字节数
    PUCHAR       oBuffer );  // 指向一个缓冲区,返回后是读入的数据
```

读写 EEPROM 的 API 支持从 24C01 到 24C16 和从 24C32 到 24C4096 的各种型号的 EEPROM 存储器。

```
CH341WriteEEPROM(  // 向 EEPROM 中写入数据块
    ULONG        iIndex,  // 指定 CH341 设备序号
    EEPROM_TYPE  iEepromID,  // 指定 EEPROM 型号
    ULONG        iAddr,  // 指定数据单元的地址
    ULONG        iLength,  // 准备写出的数据字节数
    PUCHAR       iBuffer );  // 指向一个缓冲区,放置准备写出的数据

CH341StreamSPI4(  // 处理 SPI 数据流,4 线接口,速度约 68K 字节
// SPI 时序: DCK 时钟输出, 默认为低, DOUT 在时钟上升沿之前输出, DIN 在时钟下降沿之后输入
    ULONG    iIndex,  // 指定 CH341 设备序号
    ULONG    iChipSelect,  // 片选控制, 位 7 为 0 则忽略片选控制, 位 7 为 1 则参数有效
```

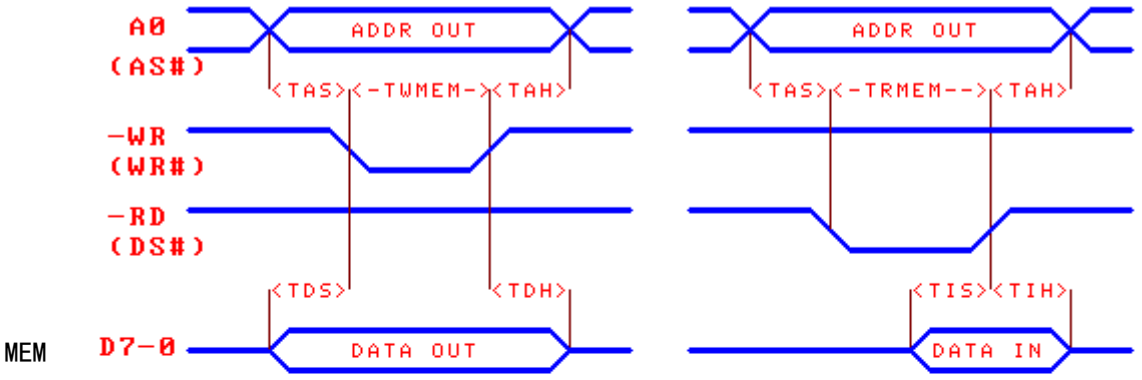
```
    ULONG    iLength,   // 准备传输的数据字节数
    PVOID    ioBuffer );  // 缓冲区,放置准备从 DOUT 写出的数据,返回后是从 DIN 读入的数据
```

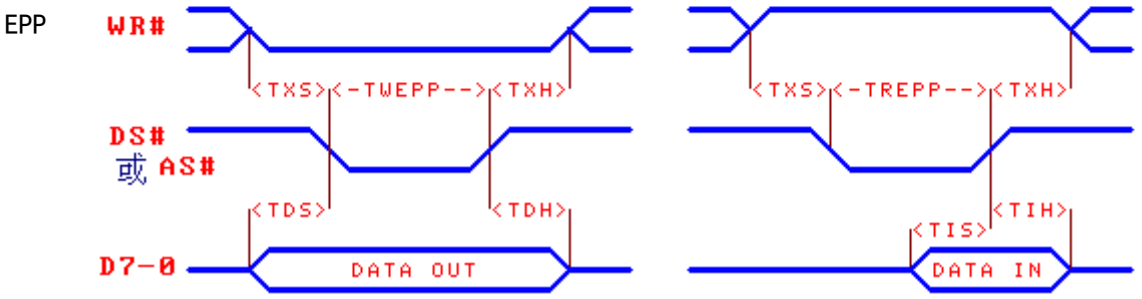
有关 DLL 中各个 API 的使用实例请参考 CH341 评估板资料中的各个源程序及例子。

4、参数

4.1. MEM 方式时序参数（测试条件：TA=25℃，参考附图前半部分）

名称	参数说明	最小值	典型值	最大值	单位
TWMEM	写选通 WR#的低电平有效宽度	230	250	100000	nS
TRMEM	读选通 RD#的低电平有效宽度	300	330	100000	nS
TAS	WR#或 RD#有效前的地址建立时间	80			nS
TAH	WR#或 RD#有效后的地址保持时间	230			nS
TDS	WR#有效前的数据输出建立时间	80			nS
TDH	WR#有效后的数据输出保持时间	300			nS
TIS	RD#无效前的数据输入建立时间	170			nS
TIH	RD#无效后的数据输入保持时间	0			nS



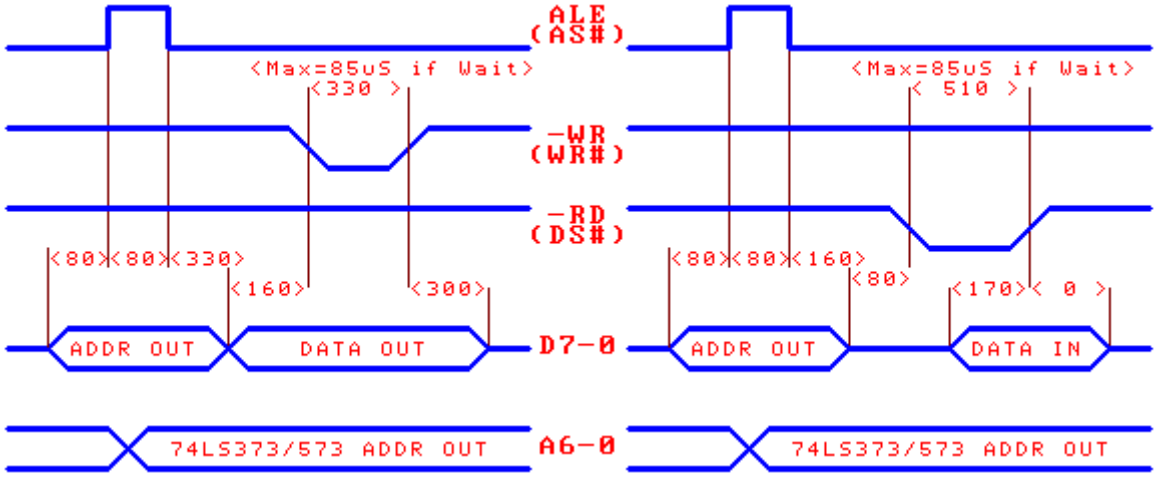


4.2. EPP 方式时序参数（测试条件：TA=25℃，参考附图后半部分）

名称	参数说明	最小值	典型值	最大值	单位
TWEPPD	数据写操作的选通的低电平有效宽度	220	250	100000	nS
TWEPPA	地址写操作的选通的低电平有效宽度	150	160	10000	nS
TREPPD	数据读操作的选通的低电平有效宽度	300	330	100000	nS
TREPPA	地址读操作的选通的低电平有效宽度	220	250	10000	nS
TXS	选通有效前的方向 WR#建立时间	500			nS
TXH	选通有效后的方向 WR#保持时间	150			nS
TDS	选通有效前的数据输出建立时间	80			nS

TDH	选通有效后的数据输出保持时间	220			nS
TIS	选通无效前的数据输入建立时间	170			nS
TIH	选通无效后的数据输入保持时间	0			nS

4.3. BUS 方式时序参数（测试条件：TA=25℃，参考下图，数值单位为 nS）

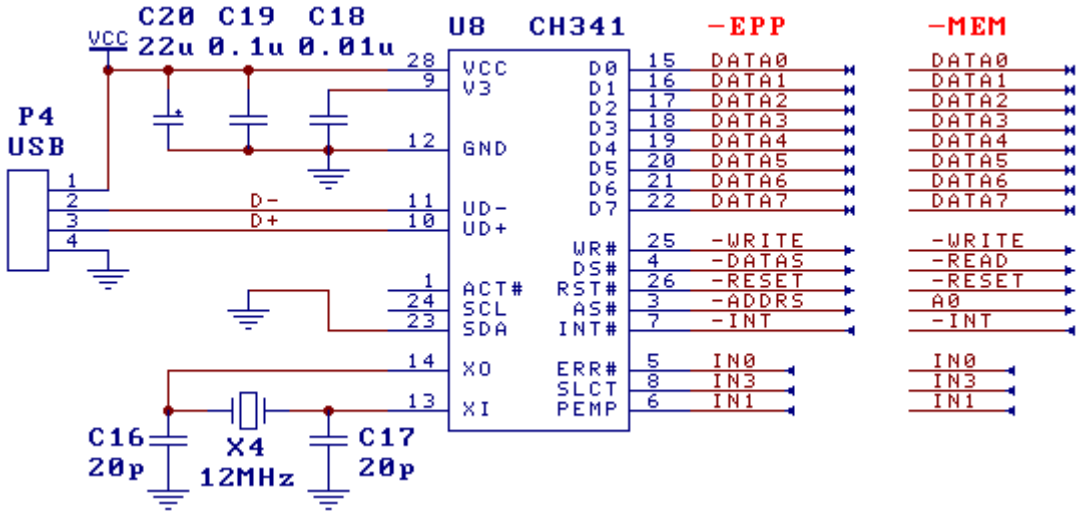


5、应用

5.1. 基本连接（下图）

P4 是 USB 端口，USB 总线包括一对 5V 电源线和一对数据信号线，通常，+5V 电源线是红色，接地线是黑色，D+信号线是绿色，D-信号线是白色。USB 总线提供的电源电流通常可以达到 500mA，一般情况下，CH341 芯片和低功耗的 USB 产品可以直接使用 USB 总线提供的 5V 电源。如果 USB 产品通过其它供电方式提供常备电源，那么 CH341 也应该使用该常备电源，如果需要同时使用 USB 总线的电源，那么可以通过阻值约为 1Ω 的电阻连接 USB 总线的 5V 电源线与 USB 产品的 5V 常备电源，并且两者的接地线直接相连接。

电容 C18 用于 CH341 内部电源节点退耦，C18 是容量为 0.01μF～0.1μF 的高频电容。电容 C19 和 C20 用于外部电源退耦，C19 是容量为 0.1μF 的高频电容。晶体 X4、电容 C16 和 C17 用于时钟振荡电路。X4 的频率是 12MHz，C16 和 C17 是容量为 15pF～30pF 的高频电容。如果使用内置时钟的 CH341B/F/C 芯片，那么可以去掉 X4 和 C16，并将 C17 换成 0Ω 电阻。



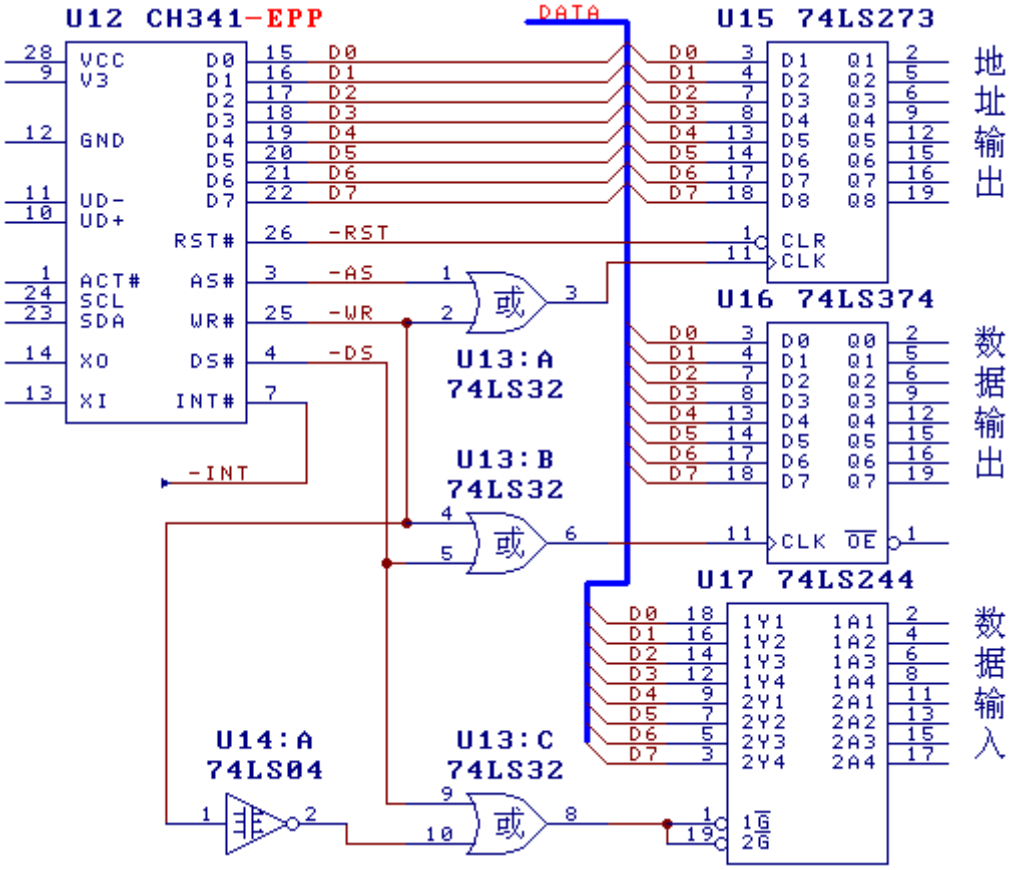
在设计印刷线路板 PCB 时，需要注意：退耦电容 C18 和 C19 尽量靠近 CH341 的相连引脚；使 D+和 D-信号线贴近平行布线，尽量在两侧提供地线或者覆铜，减少来自外界的信号干扰；尽量缩短 XI

和 XO 引脚相关信号线的长度，为了减少高频干扰，可以在相关元器件周边环绕地线或者覆铜。

图中 SDA 引脚直接接地，所以 CH341 工作于并口方式。

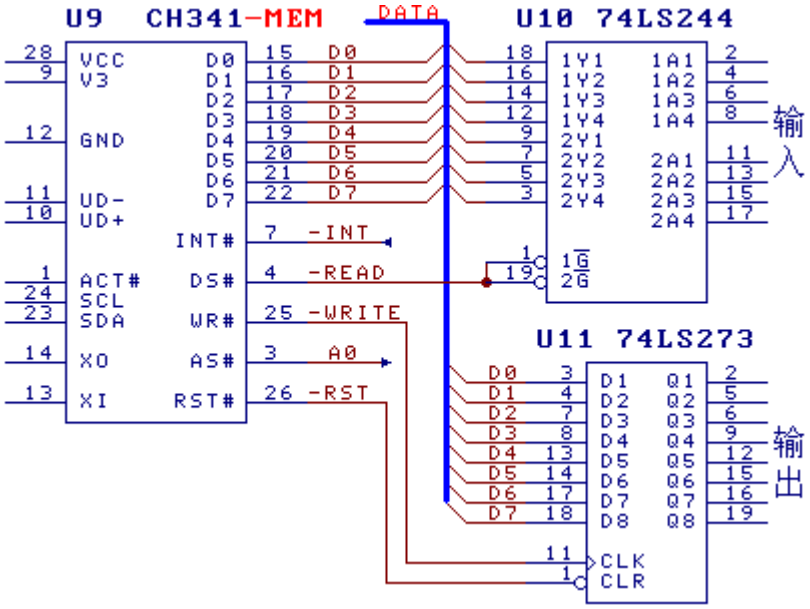
5.2. EPP 并口方式应用（下图）

图中使用非门以及或门进行译码，产生较多的控制信号，如果实际应用只需要较少的输入和输出，那么可以省去这些译码，而直接使用 AS#、DS#等控制信号进行简单的实现。



5.3. MEM 并口方式应用（下图）

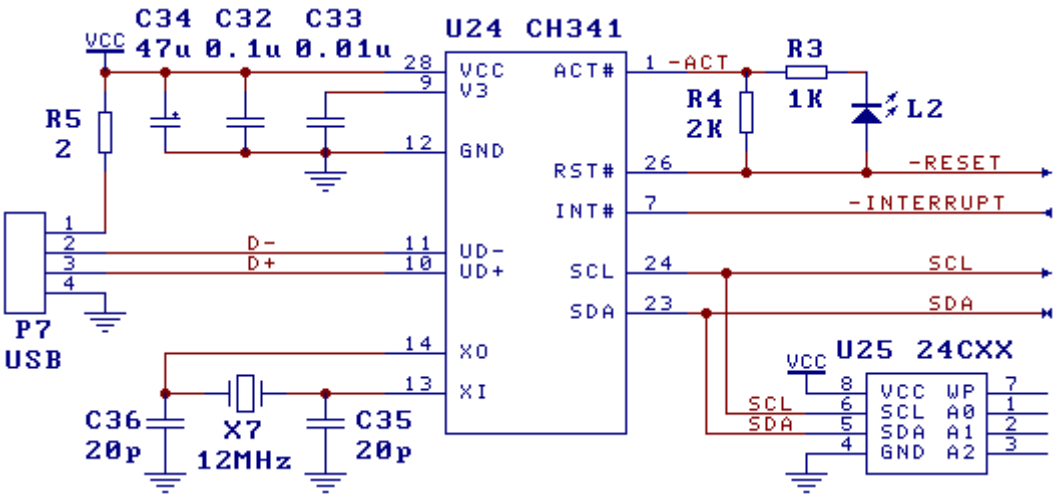
图中尚未用到 A0 引脚（AS#引脚），可以实现 8 位数字信号输入和 8 位数字信号输出。如果使用 A0 控制 74LS139 分别对-READ 和-WRITE 进行地址片选，那么 CH341 可以连接两组 74LS244+74LS273，从而实现 16 位数字信号输入和 16 位数字信号输出。



5.4. 两线同步串口应用（下图）

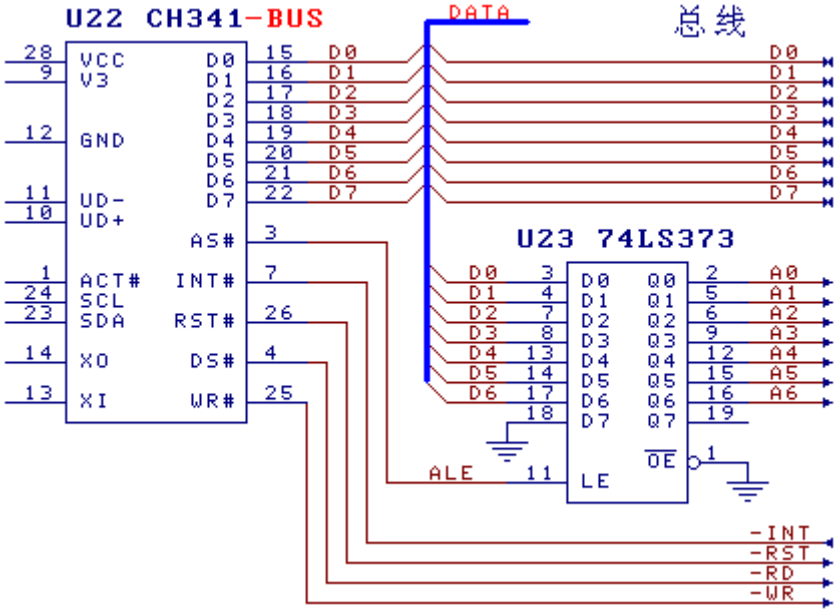
2 线同步串口支持多个设备的地址识别，采用数据流方式读写数据，支持一次读写较大的数据块。CH341 的两线串口支持 20KHz/100KHz/400KHz/750KHz 的速度，与具有硬件两线串口的设备连接时可以选择较高的速度，与软件模拟两线串口的单片机连接时只能选择较低的速度（例如 20KHz）。

图中使用电阻 R4 强制 ACT#引脚在 CH341 功能配置期间为低电平，从而禁止 CH341 配置时访问 2 线同步串口而影响总线上的其它设备，如果不需要 LED 显示，那么可以省去电阻 R3 和发光管 L2，并且可以将 R4 接 RST#端改为接 GND。



5.5. BUS 并口方式应用（下图）

图中用 ALE 控制 U23 锁存得到 7 位地址 A0-A6，可以用于驱动存储器或再次译码产生多个片选。



5.6. 数字输入输出

如果只需要少于 8 个的输入输出引脚，那么还可以省掉外部的各种 74LS 器件，直接使用 CH341 的 D7～D0 以及其它引脚，通过调用 DLL 中的 CH341Set_D5_D0 等 API 实现简单的输入输出。