众所周知，RS485 的工作流程是，在发送时必须把 GPIO 输出一个电平，在接收时，必须把 GPIO 设置为另一个电平，这样才可以让 485 芯片处于接收数据的状态，并且要及时切换 485 的收发状态，否则有可能造成 485 最后一个字节没有发送，或第 1 个字节无法接收等现象

CPU：AM335X
操作系统：WINCE
在 WINCE 中，将 RS485 的操作，整合到串口驱动中，实现在 WINCE 中的串口通讯
AM335X 操作串口的源码位置：
\WINCE700\platform\common\src\soc\COMMON_TI_V1\COMMON_TI_AMXX\SERIAL
MDD 层源码的位置：
WINCE700\platform\common\src\soc\COMMON_TI_V1\COMMON_TI\SERIAL\COM_MDD2
开始分析程序：
先分析操作串口的部分：此部分和具体的 CPU 寄存器操作有关，具体的寄存器设置都在此文件中完成，例如：串口收发，中断，FIFO 的设置等
GetSerialObject 函数，由 MDD 层调用，这里会返回一个 HW_VTBL 结构体给 MDD 层，MDD 层以后就通过函数指针的调用方式，来操作串口设备
我们在看几个重要的函数
HWOpen：打开串口函数，在 WINDOWS API 调用 CreateFile 函数后就会调用到这里，打开串口后，我们先将 RS485 设置为接收数据的状态

```
//设置485模式
rs485_init(pPdd, pPdd->hGpio);
rs485_send_recv_switch(pPdd, RS485RECV, pPdd->hGpio);
rs485_422_switch(pPdd->UARTIndex, SERIAL_MODE_RS485);
```


HWTxIntr：发送数据函数，此函数在 FIFO 触发等级到达后，会被调用，这里要实现 485 的状态的切换

```
if (*pLength == 0)
{
    //数据发送完成 关闭中断
    DEBUGMSG(ZONE_THREAD, (L"HWTxIntr Disable TX interrupt\r\n"));
    // Disable TX interrupt
    pPdd->intrMask &= ~UART_IER_THR;
    OUTREG8(&pUartRegs->IER, pPdd->intrMask);
    LeaveCriticalSection(&pPdd->hwCS);
    SetAutoIdle(pPdd, FALSE);

    //while ((INREG8(&pUartRegs->LSR) & UART_LSR_TX_SR_E) == 0);/
    rs485_send_recv_switch(pPdd, RS485RECV, pPdd->hGpio);
    SetSCRTXShiftEmptyInt(pPdd, FALSE);
    goto cleanUp;
}
```

在这里原先的做法是循环等待 LSR 寄存器的移位寄存器为空

当*pLendgth=0 时，就代表发送的内容完成了，因此我们切换为收的状态

```
if (AM335x_UARTFifo_Size > *pLength)
{
    SetSCRTXShiftEmptyInt(pPdd, TRUE);
}
```


否则，就是有发送的内容，我们判断当*pLength 长度小于 AM335X 的 FIFO 大小时，就开启 FIFO 为空，并且发送移位寄存器同时为空的中断

SetSCRTXShiftEmptyInt 函数的实现如下：

```
static VOID SetSCRTXShiftEmptyInt(UARTPDD *pPdd, UINT8 imode)
{
    //串口2 4 5都是485的
    if (2 == pPdd->UARTIndex || 4 == pPdd->UARTIndex || 5 == pPdd->UARTIndex)
    {
        if (imode)
        {
            pPdd->CurrentSCR = UART_SCR_TX_TRIG_GRANU1 | UART_SCR_RX_TRIG_GRANU1 | UART_SCR_TX_EMPTY_CTL;
            OUTREG8(&pPdd->pUartRegs->SCR, pPdd->CurrentSCR);
        }
        else
        {
            pPdd->CurrentSCR = UART_SCR_TX_TRIG_GRANU1 | UART_SCR_RX_TRIG_GRANU1;
            OUTREG8(&pPdd->pUartRegs->SCR, pPdd->CurrentSCR);
        }
    }
}
```

主要是设置 SCR 寄存器

HWIOCtl 函数中，我们增加 2 个操作

```
case IOCTL_SET485SENDMODE://485发送模式
    rs485_send_recv_switch(pPdd, RS485SEND, pPdd->hGpio);
    break;

case IOCTL_SET485RECVMODE://485接收模式
    rs485_send_recv_switch(pPdd, RS485RECV, pPdd->hGpio);
    break;
```

可以单独的设置 485 的收发 GPIO 电平

现在看一下 MDD 层：

COM_Write 函数：此函数调用 WriteFile 函数后，会调用到这里，

```
pFuncTbl->HWIoctl(pHWHead, IOCTL_SET485SENDMODE, NULL, 0, NULL, 0, NULL);

// We call the same write routine that a TX_INTR does.  It queus as
// much data as possible, then returns.  From then on, the normal
// interrupt mechanism kicks in.
DoTxData( pSerialHead ); //开始发送数据,同时发送中断会到来
```

我们在 DoTxData 函数上面增加一个 HWIoctl 函数，用此方法实现 485 状态的切换
然后调用 DoTxData 函数进行数据的发送，同时发送中断会到来

到这里，大致的流程已经结束，到目前发送此驱动即使使用 232 通讯，在触摸屏被按住的情况下，效率也不是很高，还要想办法优化一下此驱动。