IO 流驱动开发及相应的应用程序实例讲解

我们的流驱动编写了如下四个文件 GPI.cpp、source、makefile、GPI.def
下面我们依次介绍：
1.GPI.cpp
该文件实现标准流接口驱动函数
DWORD XXX_Init(LPCTSTR pContext,LPCVOID lpvBusContext);
DWORD XXX_Open (DWORD dwData, DWORD dwAccess, DWORD dwShareMode);
BOOL GPI_Close(DWORD dwData) ;
DWORD XXX_Write(DWORD dwData, LPCVOID pBuf, DWORD Len);
DWORD XXX_Read(DWORD dwData, LPVOID pBuf, DWORD Len);
BOOL XXX_Deinit(DWORD dwData);
BOOL WINAPI
XXX_DllEntry(HANDLE hInstDll,DWORD dwReason,LPVOID lpvReserved)；

设备管理器使用 XXX 前缀，在实现流接口时，必须用适合于特定实现的前缀来代替 XXX，除非使用了 DEVFLAGS_NAKEDENTRIES 标志。XXX 通常是在设备注册表中由 Prefix 注册来定义的。

- DWORD XXX_Init(LPCTSTR pContext,LPCVOID lpvBusContext);

```
DWORD GPI_Init(LPCTSTR pContext,LPCVOID lpvBusContext)
{
    DWORD isSuccess=1;//assume success
    DWORD error=0;   //接受出错码
     /* IO Register Allocation */
    RETAILMSG(1,(TEXT("GPI_init\r\n")));
   //为 v_pIOPregs 变量分配虚拟空间
    v_pIOPregs = (volatile IOPreg *)VirtualAlloc(0, sizeof(IOPreg),
MEM_RESERVE,PAGE_NOACCESS);
    if (v_pIOPregs == NULL) //分配虚拟空间不成功
    {
        ERRORMSG(1,(TEXT("For IOPregs : VirtualAlloc failed!\r\n")));
        error=GetLastError();    //获得出错码
        printf("For IOPregs : VirtualAlloc failed!%d\r\n",error);
        isSuccess = 0;
    }
    else
    {
        if(!VirtualCopy((PVOID)v_pIOPregs, (PVOID)(IOP_BASE) , sizeof(IOPreg),
PAGE_READWRITE | PAGE_NOCACHE))  /*把 v_pIOPregs 虚拟空间映射到物理内存*/
        {
            ERRORMSG(1,(TEXT("For IOPregs: VirtualCopy failed!\r\n")));
            isSuccess = 0;
        }
```

```
    }
    if (!isSuccess)   /*分配虚拟空间或映射到物理内存失败*/
    {
        if (v_pIOPregs) /*  如果 v_pIOPregs 变量非无效*/
        {
            VirtualFree((PVOID) v_pIOPregs, 0, MEM_RELEASE); /*释放虚拟空间*/
        }
        v_pIOPregs = NULL;
    }
    return (isSuccess);
}
```

- DWORD XXX_Open (DWORD dwData, DWORD dwAccess, DWORD dwShareMode)

```
DWORD GPI_Open (DWORD dwData, DWORD dwAccess, DWORD dwShareMode)
{
    RETAILMSG(1,(TEXT("*****GPILED: GPI_OPEN\r\n")));
    return (1);
}
```

当应用程序调用 CreatFile 函数时，即会调用驱动 GPI_Open 函数。该函数由于没做什么直接返回 1.

- BOOL XXX_Close(DWORD dwData)

```
BOOL  GPI_Close(DWORD dwData)
{
   RETAILMSG(1,(TEXT("*****GPILED: GPI_Close\r\n")));
    return (TRUE);
}
```

当应用程序调用 closeHandle 时，即会调用驱动中 GPI_Close 函数。该函数由于没做什么直接返回 1.

- DWORD XXX_Write(DWORD dwData, LPCVOID pBuf, DWORD Len);

```
DWORD GPI_Write(DWORD dwData, LPCVOID pBuf, DWORD Len)
{
        BYTE* pdatabuf;
        BYTE  gpioNum;
        BYTE  gpioState;
        pdatabuf = (BYTE*)pBuf;
        gpioNum = *pdatabuf++;   //指明是哪个 led
        gpioState = *pdatabuf;    //指明是亮还是灭
        RETAILMSG(1,(TEXT("*****GPILED: GPI_Write\r\n")));
        if(gpioNum == 0 || gpioNum == 1 || gpioNum == 2|| gpioNum == 3)      //这里可以选
择你要用的 gpio 口  GPFO3,4,5,6
        {
```

```
                if(gpioState == 1)

                {
                    v_pIOPregs->rGPFCON |= (1<<(gpioNum*2+6));
                    v_pIOPregs->rGPFCON &= ~(1<<(gpioNum*2+7));
                    v_pIOPregs->rGPFUP &=   ~(1<<gpioNum+3);
                    v_pIOPregs->rGPFDAT &= ~(1<<gpioNum+3);          //设置相应寄
存器

                }
                else
                {
                    v_pIOPregs->rGPFCON |= (1<<(gpioNum*2+6));
                    v_pIOPregs->rGPFCON &= ~(1<<(gpioNum*2+7));
                    v_pIOPregs->rGPFUP &=   ~(1<<gpioNum+3);
                    v_pIOPregs->rGPFDAT |= (1<<gpioNum+3);
                }
            }
            else
            {
                if(gpioNum == 5)      //BELL
                {
                    if(gpioState == 1)
                    {
                        v_pIOPregs->rGPBCON |= (1<<(gpioNum-5));
                        v_pIOPregs->rGPBCON &= ~(1<<(gpioNum-5+1));
                        v_pIOPregs->rGPBUP &=   ~(1<<gpioNum-5);
                        v_pIOPregs->rGPBDAT |= (1<<gpioNum-5);
                    }
                    else
                    {
                        v_pIOPregs->rGPBCON |= (1<<(gpioNum-5));
                        v_pIOPregs->rGPBCON &= ~(1<<(gpioNum-5+1));
                        v_pIOPregs->rGPBUP &=   ~(1<<gpioNum-5);
                        v_pIOPregs->rGPBDAT &= ~(1<<gpioNum-5);
                    }
                }
            }
        return 1;
}
```

当应用程序调用 WriteFile 函数写数据到设备时，操作系统会调用 XXX_Write 函数。

- DWORD XXX_Read(DWORD dwData, LPVOID pBuf, DWORD Len);

```
DWORD GPI_Read(DWORD dwData,    LPVOID pBuf, DWORD Len)
{
        RETAILMSG(1,(TEXT("state:       %d\r\n"),*pdatabuf));
         return 1;
}
```
我们没有用到该函数，让其直接返回 1

- BOOL XXX_Deinit(DWORD dwData);

```
BOOL    GPI_Deinit(DWORD dwData)
{
        if(v_pIOPregs)
        {
            VirtualFree((PVOID) v_pIOPregs, 0, MEM_RELEASE); /*释放虚拟空间*/
            v_pIOPregs = NULL;
        RETAILMSG(1,(TEXT("*****GPILED: GPI_Deinit\r\n")));
        }
        return 1;
}
```
卸载设备的时候调用该函数。

- XXX_DllEntry(HANDLE hInstDll,DWORD dwReason,LPVOID lpvReserved)

```
BOOL WINAPI
GPI_DllEntry(HANDLE    hInstDll,DWORD     dwReason,LPVOID    lpvReserved )
{
        return 1;
}
```
指定动态库的入口函数。

其他的函数我们暂时没有用到，这里不做介绍了。

2.source 文件

```
TARGETNAME=gpi      // 指定目标名

PREPROCESSDEFFILE=1
RELEASETYPE=PLATFORM

TARGETTYPE=DYNLINK     //指定编译成 dll 即动态链接库


TARGETLIBS= \
    $(_COMMONSDKROOT)\lib\$(_CPUINDPATH)\coredll.lib       //指定要用到的库

SOURCELIBS=    \
```

$(_COMMONOAKROOT)\lib\$(_CPUINDPATH)\ceddk.lib

DEFFILE=gpi.def　　//指定 gpi.def 流接口函数暴露文件

PREPROCESSDEFFILE=1

DLLENTRY=GPI_DllEntry　//指定 GPI_DllEntry 为驱动入口函数

SOURCES=\
　　GPI.cpp　　//指定要编译的源文件

INCLUDES=$(_COMMONDDKROOT)\inc;$(_COMMONOAKROOT)\inc;$(_COMMONSDKROOT)\inc;..\..\inc　//指定头文件路径

注意：该文件中尽量不要有中文注释，否则在编译的过程中可能会有错误。

3.makefile 文件
!INCLUDE $(_MAKEENVROOT)\makefile.def
只有这一句

4.GPI.def 文件
LIBRARY　　GPI

EXPORTS
　　GPI_Init
　　GPI_Deinit
　　GPI_Open
　　GPI_Close
　　GPI_Read
　　GPI_Write
　　GPI_DllEntry
暴露流接口函数

编译步骤：
1.在 bsp 包 SDK2440 的 DRIVERS 文件夹下新建文件夹 GPI，将上述 4 个文件放在该文件夹下。
2.打开 PB，在 FileView 窗口中找到你的 GPI 文件夹，右击，点 Build Cureent Project。编译成功后，在 smdk2440—>target—>ARMV4I—>retail 下会发现 gpi.dll 文件
这样驱动就编译好了，但还没有到 nk 中。
如果要手动加载驱动还需要一个单独的注册表文件：GPI.reg 内容如下
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\GPI]

"Index"=dword:1
"Prefix"="GPI"　　//驱动函数中的 XXX 必须和 Prefix 一致
"Dll"="GPI.dll"
"Order"=dword:0

驱动就到此结束。应以用应用程序 LedText 来测试该驱动程序。

下面说一下如何将该驱动编译到内核中：
1.在 WINCE500\PLATFORM\smdk2440\DRIVERS 目录下修改 dirs 文件，在最后添加 GPI \。
2.在 smdk2440—>FILE 文件下修改 platform.bib 和 platform.reg 文件。
在 platform.bib 添加如下内容：
GPI.dll　　$(_FLATRELEASEDIR)\GPI.dll　　NK SH
在 platform.reg 添加如下内容：
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\GPI]
"Index"=dword:1
"Prefix"="GPI"
"Dll"="GPI.dll"
"Order"=dword:0
3.如果你的内核曾 Build 过就可以直接 Sysgen 了，当然也可以 Build and Sysgen
注意把 Copy files to Release Directory　Aflter Build 和 Make Run-time Image After Build 选上
4.最后就可以把生成 nk.bin 文件按手册中的步骤烧到 flash 中。
Nk.bin 的路径 ：工程路径\RelDir\smdk2440_ARMV4I_Release