3.8 LED 数码管驱动

3.8.1 实验目的

1．熟悉 EVC 集成开发环境以及相关配置。
2．利用 EVC 编写一个针对实际硬件的驱动程序。

3.8.2 实验内容

编写一个针对硬件的驱动程序，硬件是 LED。

3.8.3 实验设备

PC 机操作系统，Platform Builder 集成开发环境，OURS-PXA270-EP 实验箱。

3.8.4 基础知识

WinCE 操作系统的驱动程序共有 19 种类型分别对应相关类型的硬件设备。

其中的流接口驱动是最简单最灵活的一种驱动类型。LED 的驱动程序使用流接口驱动类型很容易完成，是学习 WinCE 驱动的入门例子。

流接口驱动程序与操作系统和硬件的关系如下图：

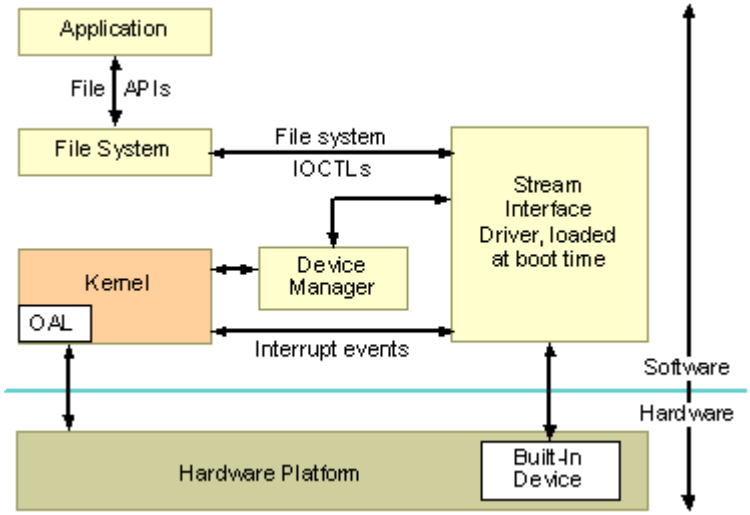


图 3.8.1 流接口驱动程序与操作系统和硬件的关系

WinCE 的设备驱动类型：

Audio Drivers　　Battery Drivers
Block Drivers　　Bluetooth HCI Transport Driver
Direct3D Device Driver Interface　　DirectDraw Display Drivers
Display Drivers　　DVD-Video Renderer
IEEE 1394 Drivers　　Keyboard Drivers

Notification LED Drivers　　Parallel Port Drivers
PC Card Drivers　　Printer Drivers
Serial Port Drivers　　Smart Card Drivers
Stream Interface Drivers　　Touch Screen Drivers
USB Drivers

应用程序使用 WinCE 操作系统的文件 API 来和流接口驱动通信。

使用 CreateFile API 打开设备、使用 CloseHandle API 关闭设备、使用 ReadFile 和 WriteFile 读写设备、使用 DeviceIoControl 控制设备。

1．设备名：

应用程序在操作设备之前必须先打开设备，如下代码：

```
CreateFile( TEXT("LED1:"),
        ENERIC_READ|GENERIC_WRITE,
        0,NULL, OPEN_EXISTING, 0, 0);
```

第一个参数是设备名，设备名由三个字符的设备名前缀、一个数字的设备索引号和一个冒号组成。设备名前缀代表了具体的设备，在本例中使用“LED”作设备名前缀。索引号是在系统中有多个此类设备时指定使用哪个设备，最后的冒号是 WinCE 操作系统的要求。

2．编写流接口驱动程序：

流接口驱动程序实质上是一个导出了特定函数的动态连接库（DLL）文件。当应用程序打开或读写设备的时候操作系统内核会调用相关的导出函数来实现相关的功能。流接口驱动程序要实现的 DLL 函数见下表：

表 流接口驱动程序要实现的 DLL 函数

函数名称	
XXX_Close	当应用程序调用 CloseHandle（）函数关闭设备时操作系统内核会调用它
XXX_Deinit	当设备管理器卸载一个驱动程序的时候会调用这个函数
XXX_Init	设备管理器初始化一个具体设备的时候会调用这个函数
XXX_IOContorl	应用程序可以使用 DeviceIoControl()函数调用这个函数
XXX_Open	在使用 CreateFile()函数打开一个设备的时候操作系统内核会调用这个函数
XXX_PowerDown	在系统挂起前会调用这个函数
XXX_PowerUp	在系统启动时会调用这个函数
XXX_Read	应用程序调用 ReadFile()函数时操作系统内核会调用它
XXX_Seek	应用程序使用 SetFilePointer()函数操作文件指针时操作系统会调用这个函数
XXX_Write	应用程序调用 WriteFile()函数时操作系统内核会调用它

其中电源管理的部分是可选的，在实际开发中接口名称中的 XXX 三个字母由设备驱动的设备文件名前缀代替。例如，如果一个流接口驱动程序的设备文件名前缀为“STR”，那么它相应要实现的 DLL 接口就为 STR_Close，STR_Deinit，STR_Init，STR_IOControl 等。

以下分别介绍几个主要的流接口驱动接口函数。

（1） DWORD XXX_Open (DWORD hDeviceContext，
DWORD AccessCode，
DWORD ShareMode)

参数：DWORD hDeviceContext，设备驱动的句柄，由 XXX_Init 函数创建的时候返回。

DWORD AccessCode，传给驱动程序使用的地址，这个地址跟读和写有关。

DWORD ShareMode，共享模式，这个参数用于一些特殊的设备。例如一些 PC 卡的设备读或写的时候是否可以共享。

返回值：返回驱动程序引用事例句柄。

描述：这个函数用于打开一个设备驱动程序，当应用程序准备对某一个设备进行读或写操作时，系统必须先执行 CreateFile（）这个函数用于打开这个设备。这个函数执行了以后系统才能够执行读和写操作。

（2） BOOL XXX_Close （DWORD hOpenContext）

参数：DWORD hOpenContext，设备驱动的引用事例句柄，由 XXX_Open 创建。

返回值：调用成功返回 TRUE，失败返回 FALSE。

描述：这个函数用于关闭一个驱动程序的引用实例。应用程序通过 CloseHandle（）来调用这个函数，当执行完这个函数的时候驱动程序引用的事例，hOpenContext 将不再有效。

（3） DWORD XXX_Init (DWORD dwContext)

参数：DWORD dwContext，指向字符串的指针。通常这个参数都为一个流接口驱动在注册表内的设置。

返回值：如果调用成功返回一个驱动程序的句柄。

描述：当用户开始使用一个设备的时候，例如，当 PC 卡初始化的时候，设备管理器调用这个函数来初始化 PC 卡设备。这个函数并不是由应用程序直接调用的，而是通过设备管理器提供的 ActivateDeviceEx（）函数来调用的。函数执行后如果成功则返回一个设备的句柄。

（4） BOOL XXX_Deinit （DWORD hDeviceContext）

参数：DWORD hDeviceContext 由 XXX_Init 创建时生成的设备句柄。

返回值：调用成功返回 TRUE，失败返回 FALSE。

描述：当一个用户需要卸载一个驱动程序的时候，设备管理器调用这个函数来卸载这个驱动程序，应用程序不能够直接调用这个函数设备管理器，通过 DeactivateDevice（）函数调用这个函数。

（5） DWORD XXX_Read（DWORD hOpenContext,
LPVOID pBuffer,
DWORD Count）

参数：DWORD hOpenContext，由 CreatFile（）函数返回的句柄。

LPVOID pBuffer，一个缓冲区地址用于从驱动读数据。

DWORD Count，需要读缓冲区的长度。

返回值：实际读取字节的长度。

描述：这个函数与 ReadFile 很相似，当一个流接口驱动程序已经被打开后，可以使用 ReadFile（）函数对这个设备进行读操作，ReadFile（）里面的 hFile 参数就是这个设备的引用实例句柄 hOpenContext，而参数 lpBuffer 将传给 pBuffer，用于表示要读/写缓冲区的地址，参数 nNumberOfBytes To Read 将传给 Count，用于表示要读/写的缓冲区的长度。同样，返回的参数：如果操作成功则返回实际读/写的地址，如果操作失败则返回值为-1。

（6） DWORD XXX_Write（DWORD hOpenContext,
LPVOID pBuffer,
DWORD Count）

参数：DWORD hOpenContext，由 CreatFile（）函数返回的句柄。
LPVOID pBuffer，一个缓冲区地址用于从驱动写数据。
DWORD Count，需要写缓冲区的长度。

返回值：实际写入字节的长度。

描述：当一个流接口驱动程序打开后，可以使用 WriteFile（）函数进行写操作。

（7） BOOL XXX_IOControl （DWORD hOpenContext,
WORD dwCode，
PBYTE pBufIn，
DWORD dwLenIn，
PBYTE pBufOut，
DWORD dwLenOut，
PDWORD pdwActualOut）

参数：DWORD hOpenContext，由 CreatFile（）函数返回的句柄。
WORD dwCode，特殊的 WORD 型标识用于描述这次 IOControl 操作的语义，一般这个都由用户自己定义。
PBYTE pBufIn，缓冲区指针指向需要传送给驱动程序使用的数据。
DWORD dwLenIn，要传送给驱动程序使用数据的长度。
PBYTE pBufOut，缓冲区指针指向驱动程序传给应用程序使用的数据。
DWORD dwLenOut，要传送给应用成熟使用数据的长度。
PDWORD pdwActualOut，DWORD 型指针用于返回实际处理数据长度。

返回值：调用成功返回 TRUE，失败返回 FALSE。

描述：这个函数通常用于向设备发送一个命令。应用程序使用 DeviceIOControl 函数来通知操作系统调用这个函数。通过参数 dwCode 来通知驱动程序要执行的操作。这个函数扩展了流接口驱动程序的功能。

另外用户可以通过更改 HKEY_LOCAL_MACHINE\Drivers\BuiltIn\YourDevice\Ioctl 健来使自己的流接口驱动程序在加载的时候执行固定的操作。

（8） Void XXX_PowerDown(DWORD hDeviceContext)

参数：DWORD hDeviceContext，由 XXX_Init 创建时生成的设备句柄。

返回值：无返回值。

（9） Void XXX_PowerUp(DWORD hDeviceContext)

参数：DWORD hDeviceContext，由 XXX_Init 创建时生成的设备句柄。

返回值：无返回值。

描述：PowerDownhePowerUp 这两个函数通常都必须要硬件的支持才能够有效，也就是说相关的硬件必须支持 PowerDown 和 PowerUp 这两个模式。

3.8.5 实验原理及说明

1．LED 的发光原理：

LED（Light Emitting Diode），即发光二极管。是一种半导体固体发光器件。它是利用固体半导体置于一个有引线的架子上，然后四周用环氧树脂密封，起到保护内部芯线的作用，所以 LED 的抗震性能好。

发光二极管的核心部分是由 p 型半导体和 n 型半导体组成的晶片，如图 3.8.2，在 p 型半导体和 n 型半导体之间有一个过渡层，称为 p-n 结。在某些半导体材料的 PN 结中，注入的少数载流子与多数载流子复合时会把多余的能量以光的形式释放出来，从而把电能直接转换为光能。PN 结加反向电压，少数载流子难以注入，故不发光。这种利用注入式电致发光原理制作的二极管叫发光二极管，通称 LED。 当它处于正向工作状态时（即两端加上正向电压），电流从 LED 阳极流向阴极时，半导体晶体就发出从紫外到红外不同颜色的光线，光的强弱与电流有关。

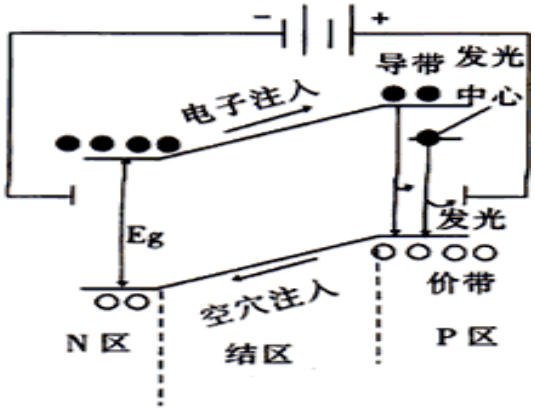


图 3.8.2

2．八段 LED 显示器

八段 LED 显示器由 8 个发光二极管组成，如图 3.8.3，图 3.8.4。基中 7 个长条形的发光管排列成“日”字形，另一个贺点形的发光管在显示器的右下角作为显示小数点用，它能显示各种数字及部份英文字母。LED 显示器有两种不同的形式：一种是 8 个发光二极管的阳极都连在一起的，称之为共阳极 LED 显示器；另一种是 8 个发光二极管的阴极都连在一起的，称之为共阴极 LED 显示器。

共阴和共阳结构的 LED 显示器各笔划段名和安排位置是相同的。当二极管导通时，相应的笔划段发亮，由发亮的笔划段组合而显示的各种字符。8 个笔划段 hgfedcba 对应于一个字节（8 位）的 D7 D6 D5 D4 D3 D2 D1 D0，于是用 8 位二进制码就可以表示欲显示字符的字形代码。例如，对于共阴 LED 显示器，当共阴极接地（为零电平），而阳极 hgfedcba 各段为 0111011 时，显示器显示"P"字符，即对于共阴极 LED 显示器，“P”字符的字形码是 73H。如果是共阳 LED 显示器，共阳极接高电平，显示“P”字符的字形代码应为 10001100（8CH）。这里必须注意的是：很多产品为方

便接线，常不按规则的方法去对应字段与位的关系，这时字形码就必须根据接线来自行设计了。

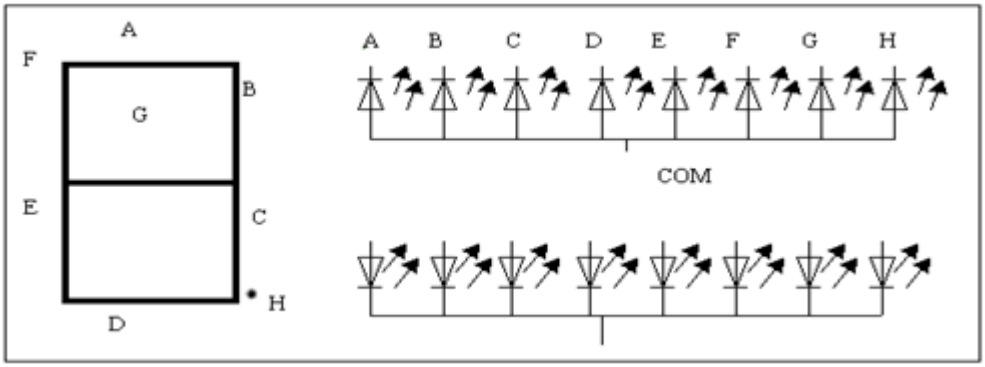


图 3.8.3

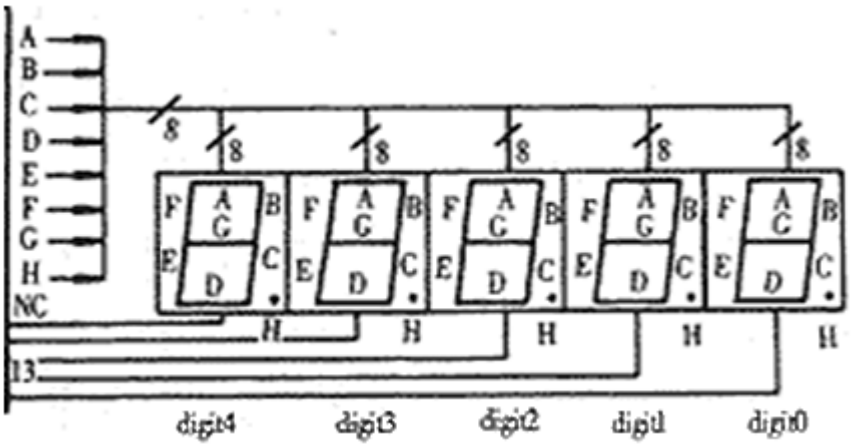
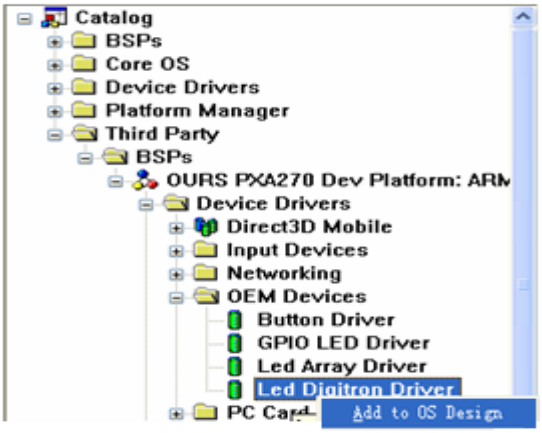


图 3.8.4

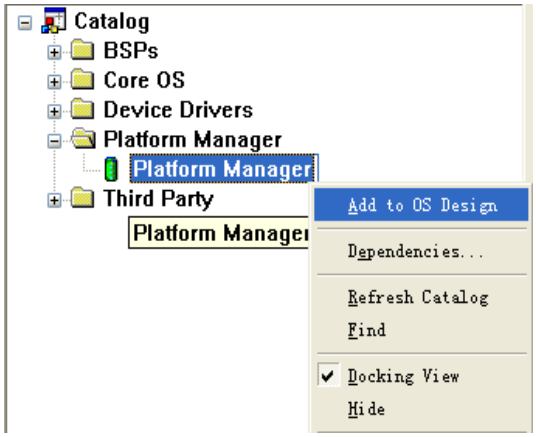
3.8.6 实验步骤

1．在 3.2 节定制好的平台基础上添加 LED 驱动，在 Third Party 中找到我们安装的 BSP 中的驱动，将其加入内核，并编译。如果要在 WINCE 中加入对小键盘和鼠标的支持，还需要加入相应的驱动程序。这样除了自己定制的功能外，还可以增加对相应硬件的支持。设置步骤如下所示。

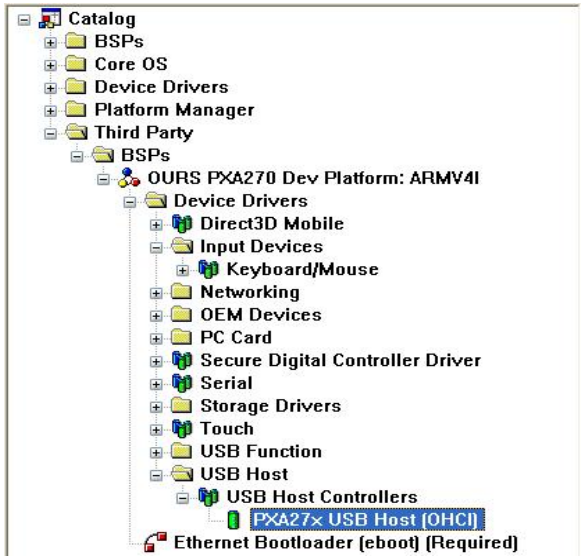
添加 LED 驱动：

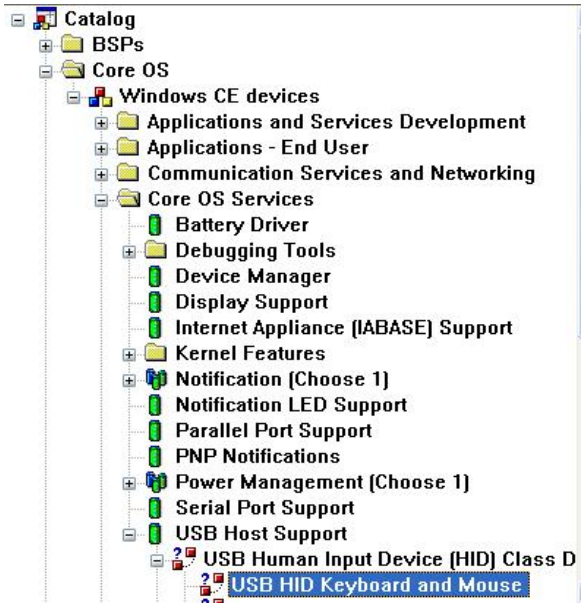


添加平台管理器基础引擎：

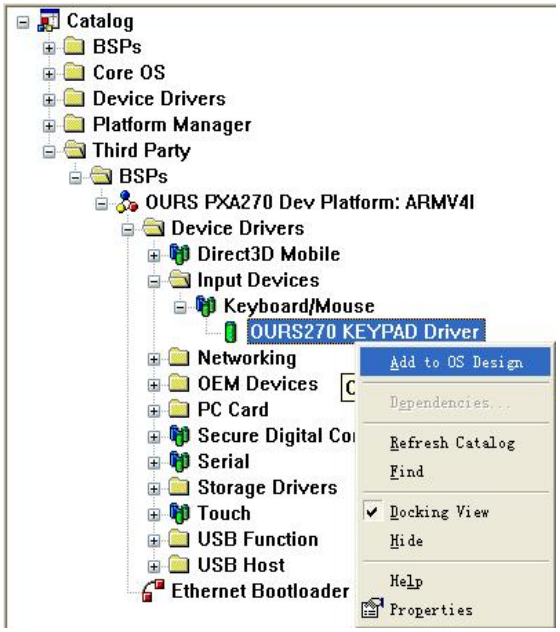


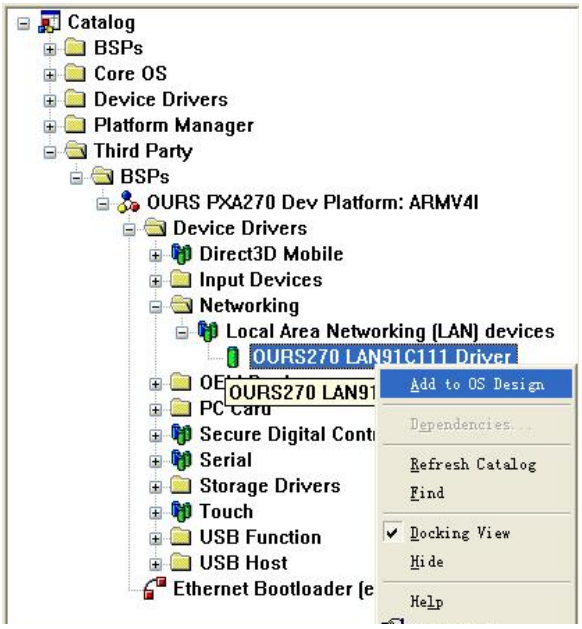
添加 USB 的驱动：





添加键盘驱动和网卡驱动：

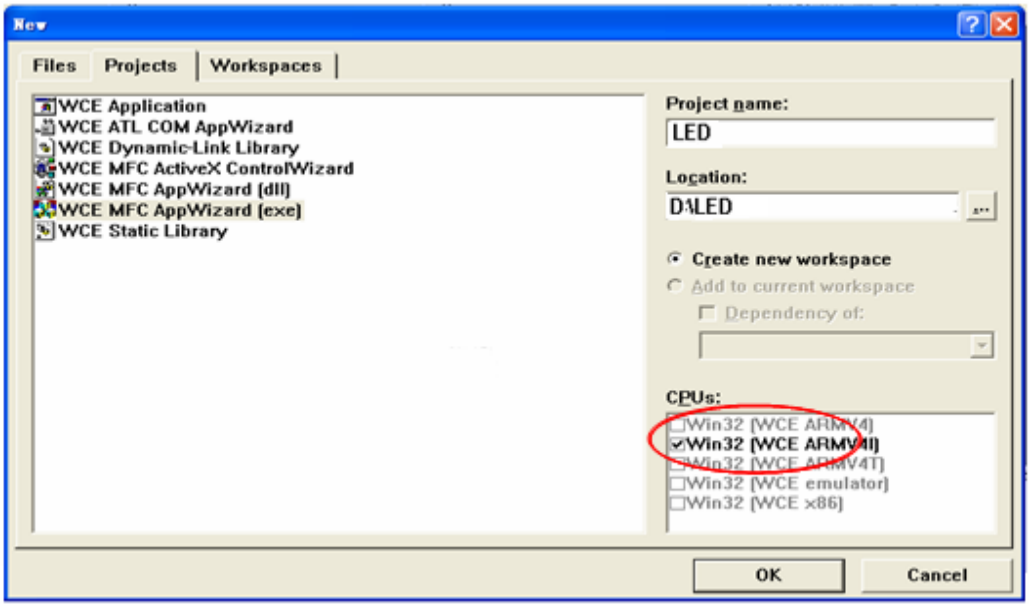




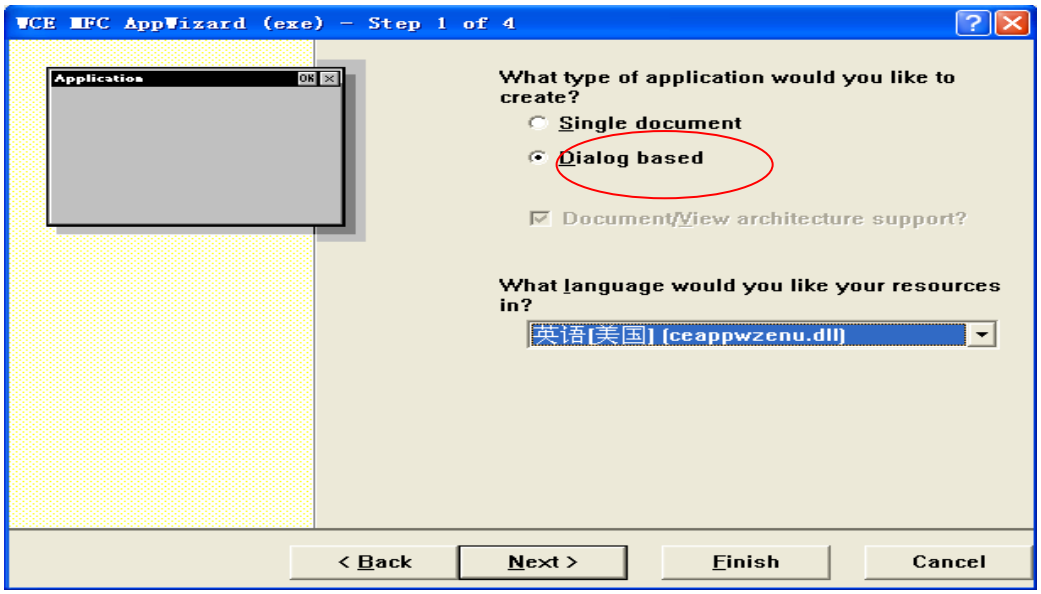
我们添加进去的驱动代码就可以在 WINCE500 平台下的相应区域看到了。我们在实验开始的时候安装了 Ours270 的 BSP，所以可以看到相应硬件驱动的支持。

2．根据实验 3.3 介绍的方法，编译内核。

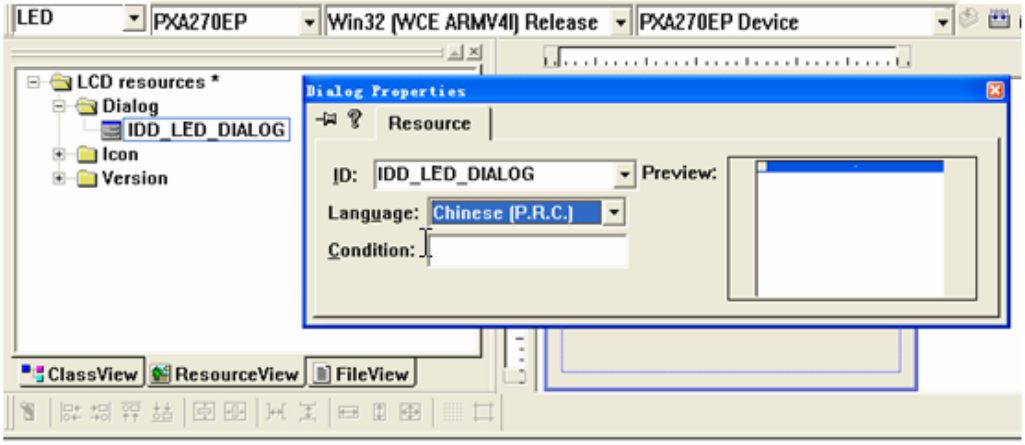
3．在机器已安装实验箱所附带的 SDK 的基础上，启动 MS eMbedded Visual C++ 4.0，利用 EVC 的工程建立向导，建立一个新的工程如下图所示，Project 里面我们选择 WCE MFC AppWizard[exe]，工程名字我们叫 LED，其中右下的 CPUs 选择 Win32（WCE ARMV4I），选择完成以后，点击 OK，进入下一个画面。之所以选择 Win32（WCE ARMV4I）是因为我们将在试验箱上运行编写的程序。如果要在 PC 机的模拟器上运行编写的程序，我们可以选择 WIN32 （WCE emulator）。



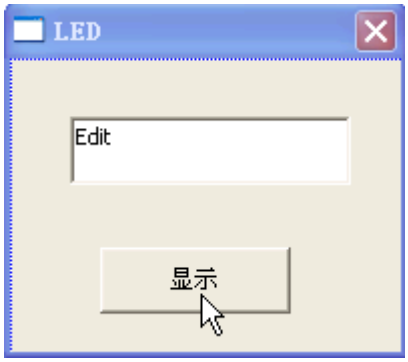
4. 我们将要用对话框制作我们程序，为此我们在下图的 What type of application would you like to create?选项里面选择 Dialog based，其他的选项为默认就可以。由于我们无需设置其他的选项，所以直接点击 Finish。



5．在 ResourceView 里面的 IDD_LED_DIALOG 右键选择 properties，将弹出如下图所示对话框，Language 中选择 Chinese（P.R.C）。只有这样，在以后的对话框中写入中文才不会出现乱码的情况。



6．在 ID 为 IDD_LED_DIALOG 对话框上，编辑如下控件，包括两个：编辑控件，其 ID 为 IDC_EDIT_NUM，Caption 为“显示”的 button 控件，其 ID 为 IDC_BUTTON_SET。



7．在 LEDDlg.h 中，类声明前添加一个全局变量：

const　BYTE ledfont[ ] = {0xfc, 0x60,0xda,0xf2,0x66,0xb6,0xbe,0xe0,0xfe,0xf6};

类定义中，添加成员变量：

Public:

HANDLE m_hDev；//声明了一个句柄，用来接收 CreateFile 的返回值，

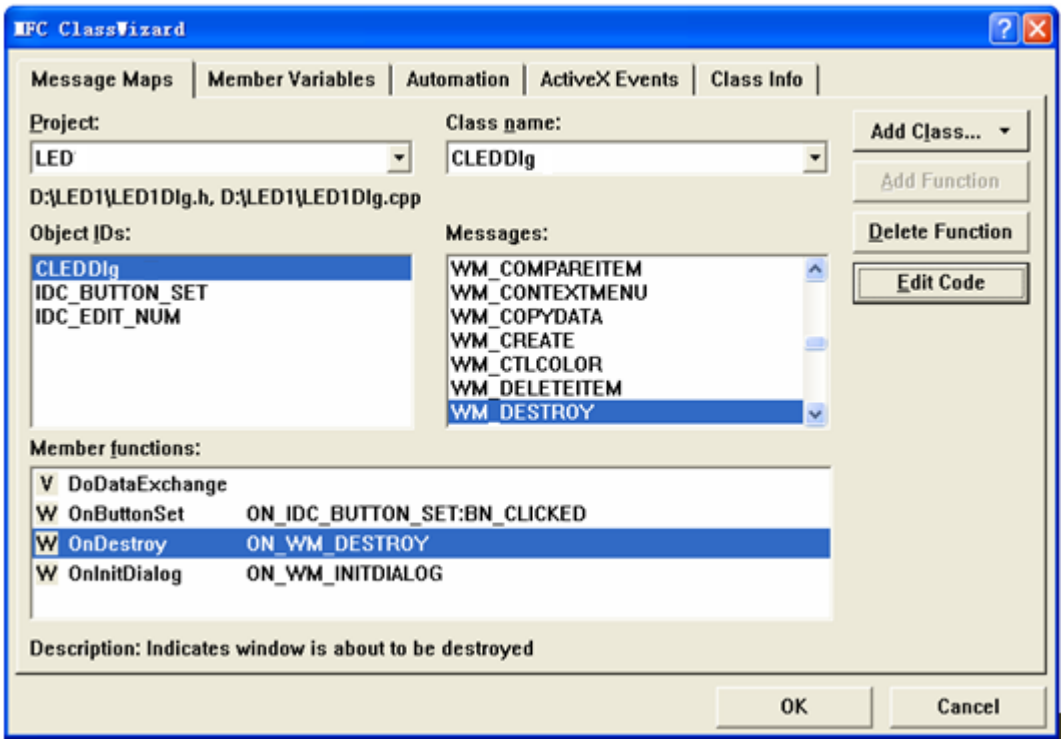
DWORD m_dwWrittenBytes；//声明了一个双字类型的变量，用作 CreateFile 的参数，

8．双击“显示”button 控件，为按钮添加消息响应函数 afx_msg void OnButtonSet();并在实现中添加如下代码：

```
void CLEDDlg::OnButtonSet()
{
    // TODO: Add your control notification handler code here
    BOOL b;
    BYTE data[4];
    DWORD bytes;

    UINT val = GetDlgItemInt(IDC_EDIT_NUM,&b,FALSE);
```

```
    //获取 Edit 框的内容
    memset(data,0,4);
    if(b)
    {
        int i,n;
        char str[16];
        n = sprintf(str,"%d",val);
        n = n > 4 ? 4:n;
        for(i = 0; i < n;i++)
        {
            data[i] = ledfont[str[i] - '0'];
        }
    }
    //调用驱动程序写到硬件上去
    WriteFile(m_hDev,data,4,&bytes,NULL);
}
```

9．选择 View->ClassWizar，在弹出的对话框中，我们添加系统用来处理窗口失败时候发送的消息，如图所示，点击“Edit Code”按钮。




```
void CLEDDlg::OnDestroy()
{
    CDialog::OnDestroy();
```

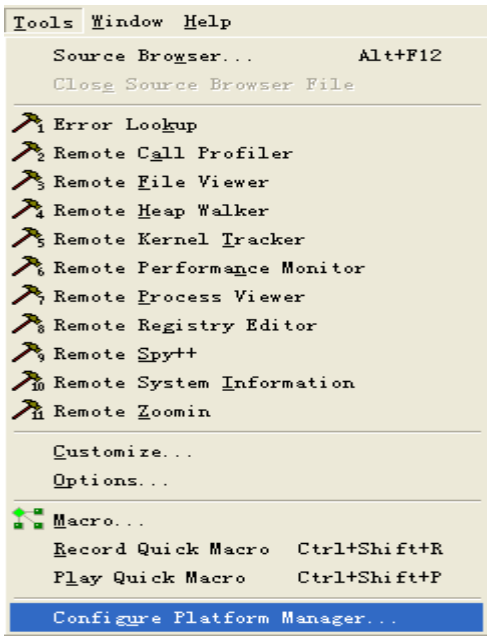
```
    // TODO: Add your message handler code here
    CloseHandle(m_hDev);
}
```

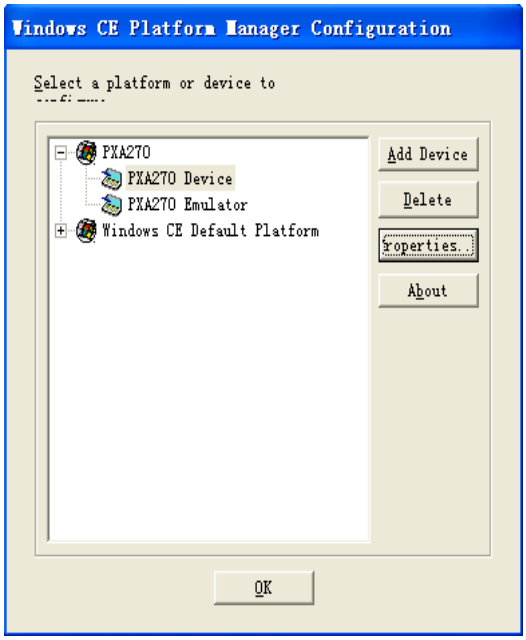
10．这个地方还有个细节需要注意，在对话框初始化的时候，我们需要打开设备，并且做一个确认，如果失败，则弹出提示对话框。这部分程序代码添加在 OnInitDialog 中，具体代码如下：

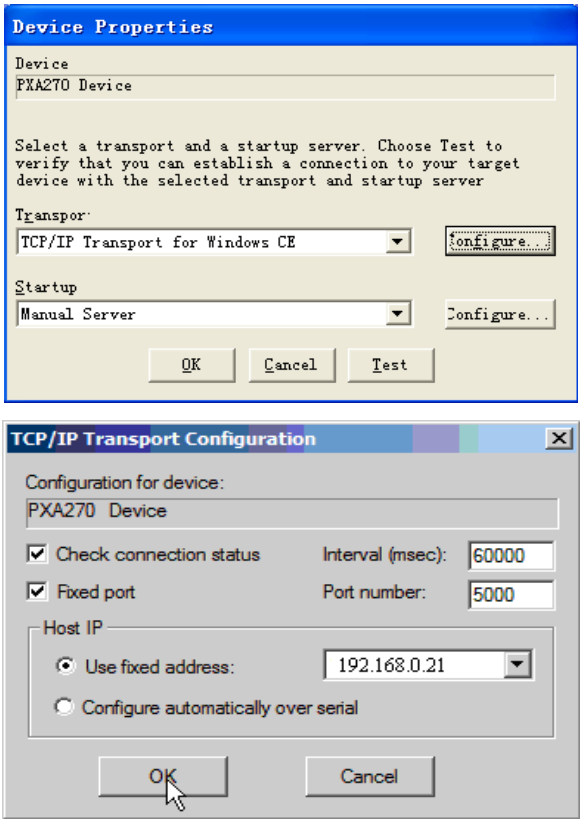
```
  // TODO: Add extra initialization here
    m_hDev = CreateFile(_T("LED2:"),GENERIC_WRITE,0,NULL,
        OPEN_EXISTING,0,0);

    if(m_hDev == INVALID_HANDLE_VALUE)
    {
        AfxMessageBox(_T("打开设备失败"));
    }
```

11．在我们 EVC 环境下，选择 Build->Rebuild All，生成可执行文件。

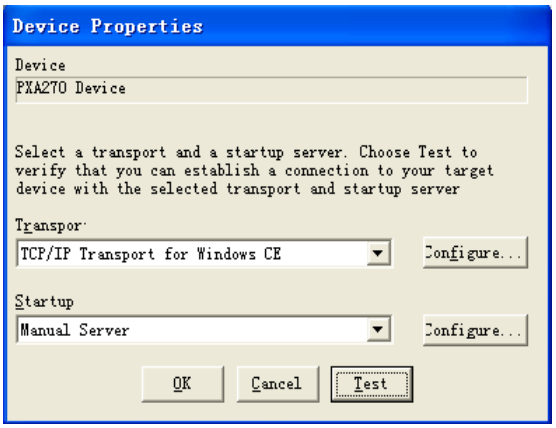
12．打开实验箱电源，启动 WinCE 平台，将宿主机和实验箱网口用交叉线连接好。在 EVC 主菜单 Tools 再单击“Cofigue Platform Manager…”菜单项，在弹出的对话框“Windows CE Platform Manager Configuration”中选择树形控件中“PXA270”|“PXA270 Device”，再单击右边“Properties”按钮。在弹出的对话框“Device Properties”中单击“Transport”下拉框右边的“Configue…”按钮。在弹出的对话框“TCP/IP Transport Configuration”中可以设置超时值、端口号、主机 IP 等，如下图所示。

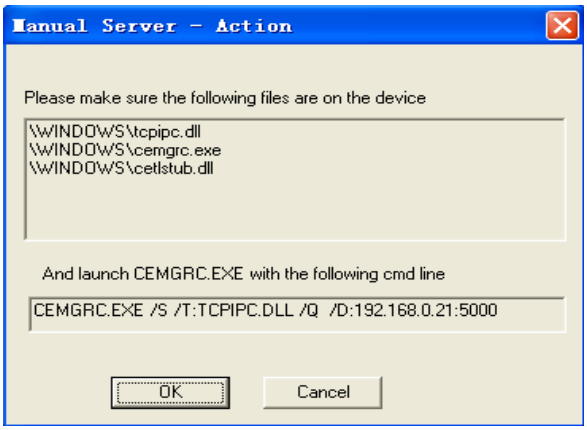






13. 设置完毕后，点击 Device Properties 下的 Test 按钮后，出现提示，在实验箱 WinCE 的命令行中键入“CEMGRC.EXE /S /T:TCPIPC.DLL /Q /D:192.168.0.21:5000”，完成此项操作后，点击 OK。





连接过程需要 2 分钟左右，如果连接不成功，请尝试再运行“CEMGRC.EXE /S /T:TCPIPC.DLL /Q /D:192.168.80.21:5000”。直至成功建立连接。

14．测试连接成功以后，如下图所示，依次选择 SDK、CPU 和设备。

15．点击 RUN 按钮后，连接成功后，即可在液晶显示屏上看到程序运行的结果，而在“我的设备”里也可以看到下载到系统里的.exe 文件，双击该文件，亦可运行程序。

16．运行程序以后，通过实验设备上的键盘在 Edit 对话框里面输入 4 位数字，然后按“显示”按钮，实验箱上的数码管就显示相应的数字，说明驱动加载成功。