P1310 软件开发手册

文档版本　　01
发布日期　　2009-9-14

深圳市华禹高科技有限公司
地址：　深圳市福田区益田路益田花园 27 栋裙楼西 2 楼 201 室。　邮编：518038
网址：www.huayucomm.com
电话：0755-82842606
传真：0755-82842601
深圳市创易电子
网址：

这篇文档介绍了开发环境如何搭建，并且介绍了 P1310 上一些典型应用和实例，所有实例都有对应的源代码，旨在帮助用户快速开发。演示代码主要是由 Win32Api，开发中还可以考虑使用.net 开发。

目录

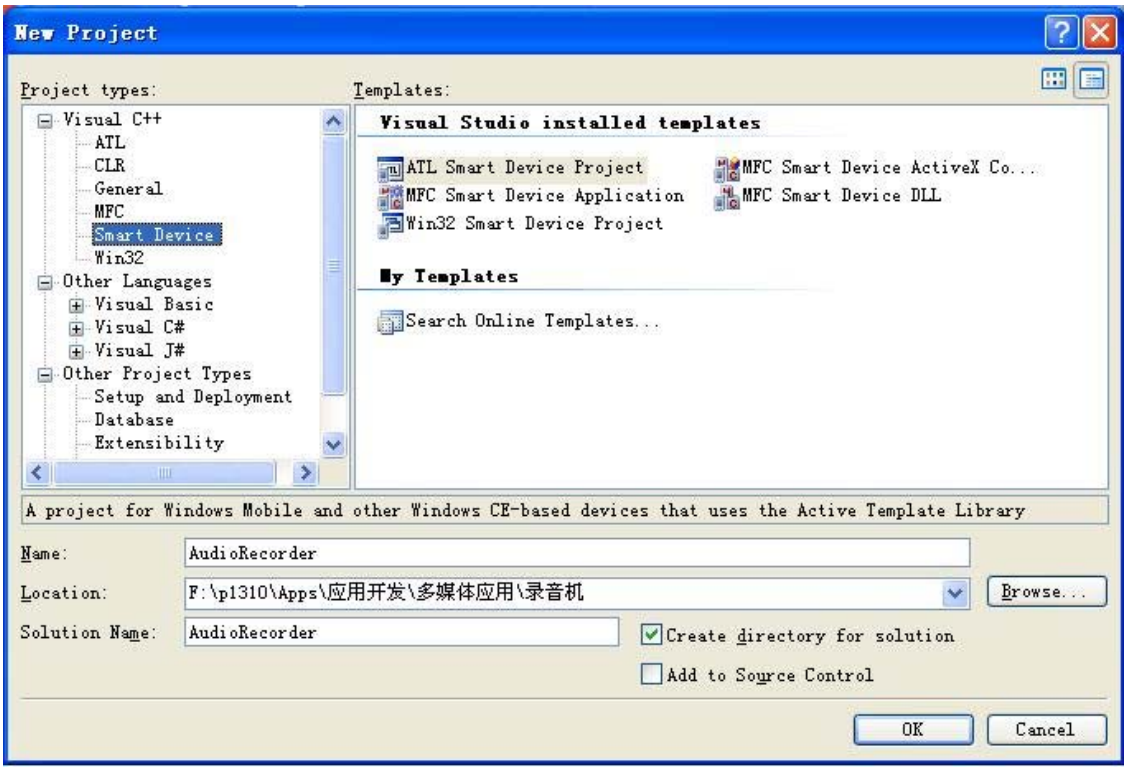
搭建开发环境

EVC 是 VC 的嵌入式版本. 但是微软已经不再维护和支持 EVC，如果已经安装了 EVC 且不需要.net 开发，可以使用 EVC，它比较轻巧。使用 EVC 建议安装 EVC sp4 补丁。

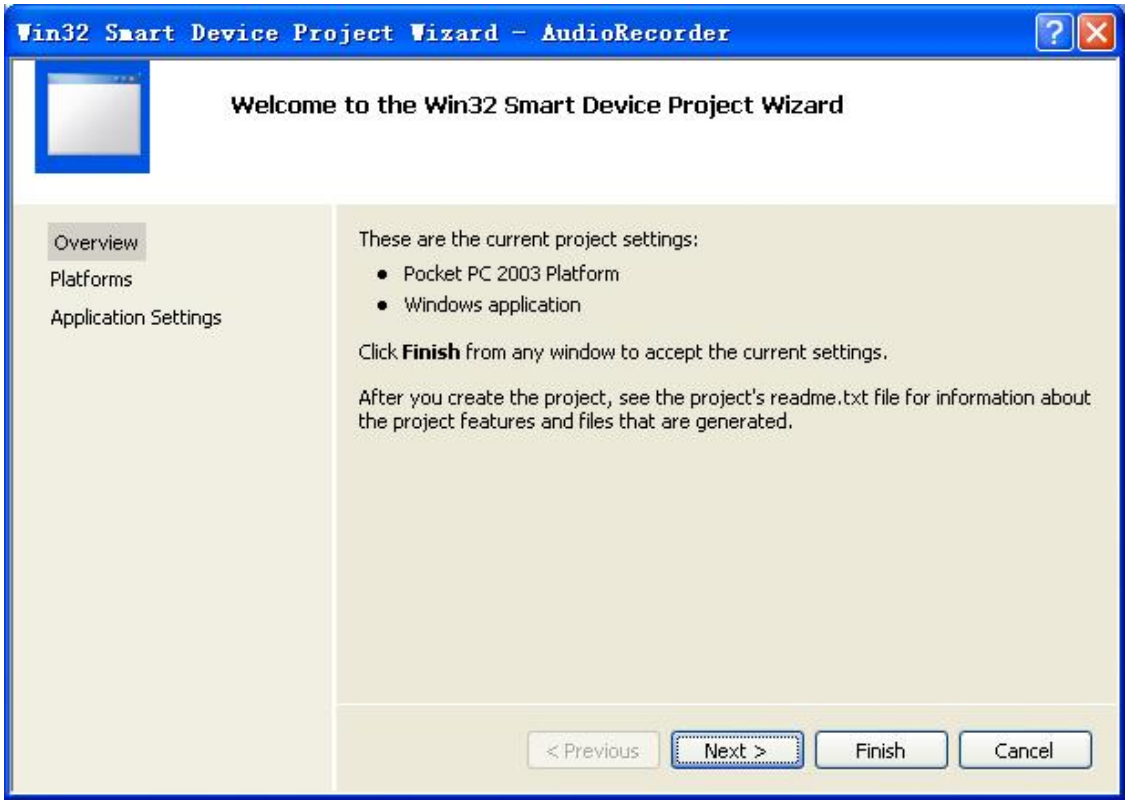
1. 安装 vs2005.安装 vs2005 的 sp1 补丁。安装 standard sdk。Sdk 可以从微软下载（地址）图片演示如何创建一个 helloworld 的项目。

下面介绍如何使用 VS2005 来开发应用程序

1. 从 VS2005 菜单【文件】中选择【新建】 【项目】或者 Ctl+Shift+N 的快捷键会弹出下面的向导。

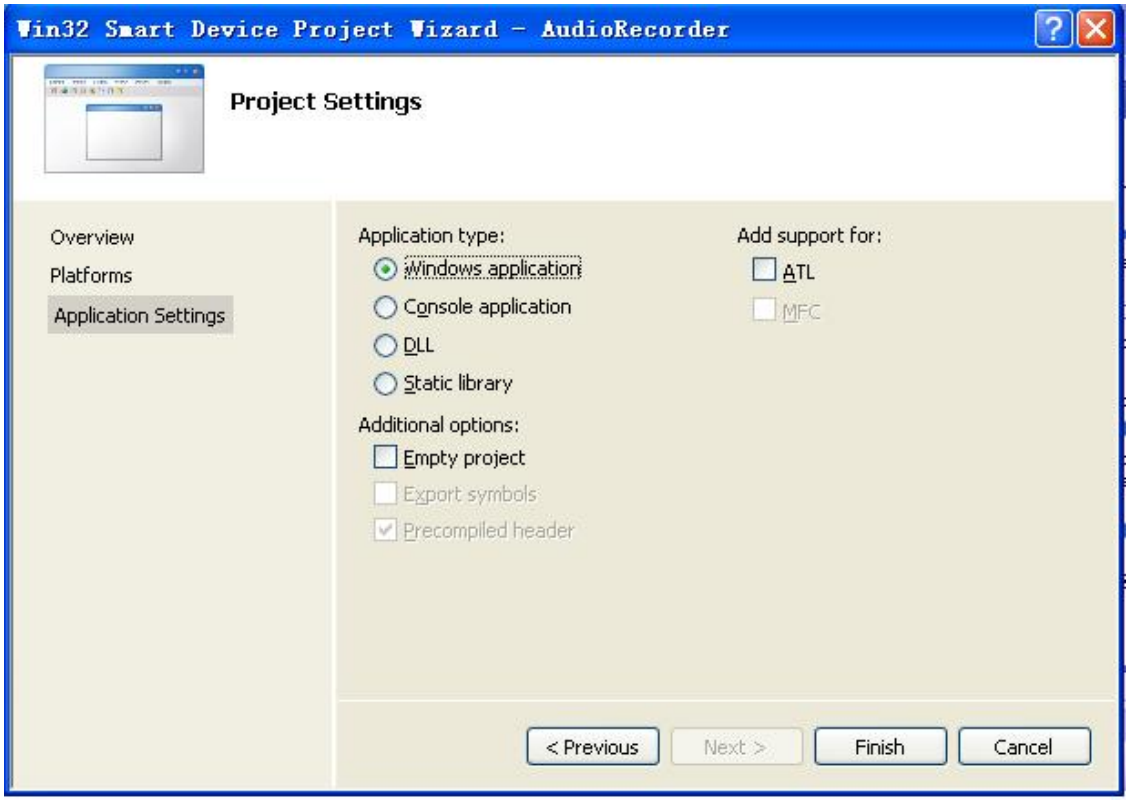


在左边的 Project type 中选择 Visual Basic 或者 Visuanl C#可以开发.net 应用程序。如果用 Win32 编程，从 Visual C++选择 **Smart Device**，在 Templates 框中选 **Win32 SmartDeviceProject**， 然后在下方输入框设置工程项目的名字，项目保存位置，最后点 OK 按钮进入下一步。



点 next 进入下一步

这时候会列举出电脑上安装的可用 SDK，选择 STANDARDSDK_500。安装 SDK 的方法见（待补充）继续下一步

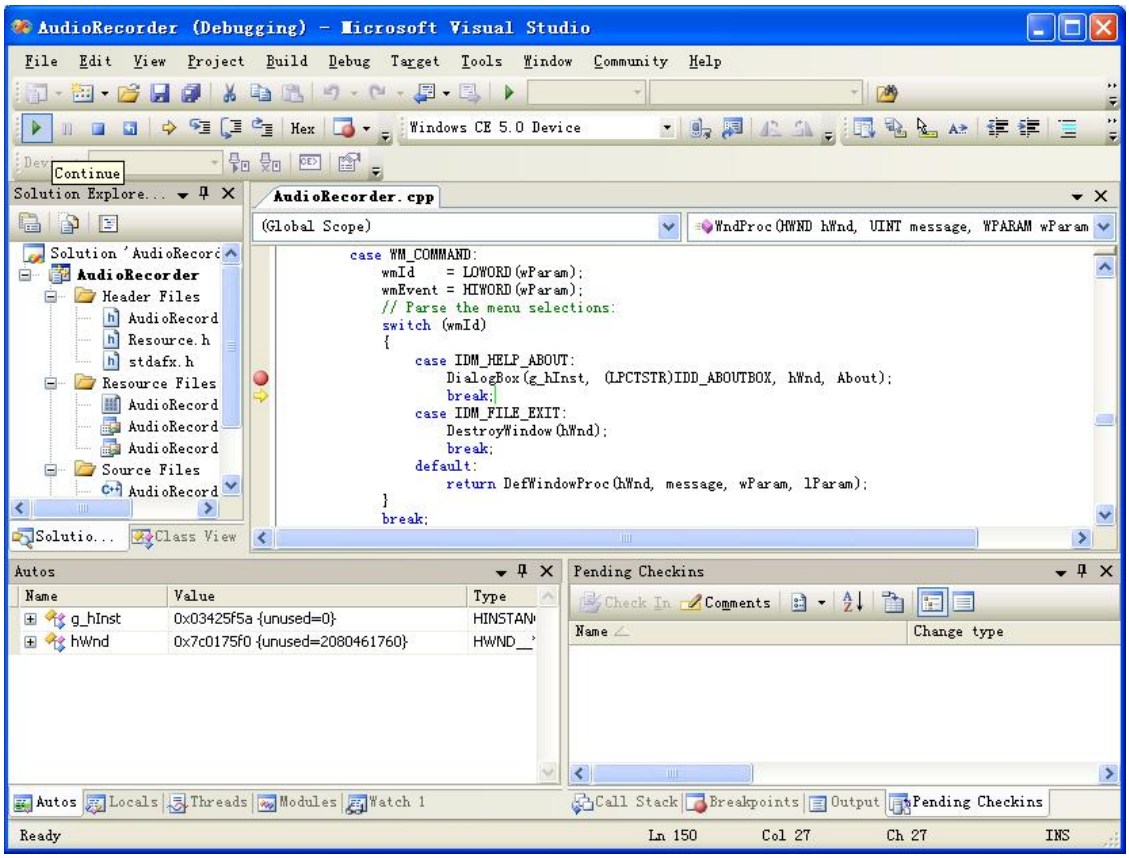


最后点 Finish 完成。

【联机调试】

1. 确定将 P1310 是通过 USB 或者其他接口连接到 PC，并且 ActiveSync 已经成功连接。(连接方式参见：P1310 用户手册)
2. 选择部署对象。设置成 P1310 设备
3. 直接点击 VS2005 的运行 F5，就可以让我们的项目编译完成并且部署拷贝到 P1310 设备上运行了，运行效果如下图。

比如我们在下图的代码处按 F9 设置断点。然后按 F5 使程序运行在 P1310.



当用户在 P1310 设备上点击这个工程的 Help 菜单时候，程序会暂停在我们设置的断点处，这时候就可以使用 查看变量，让系统单步前进等等调试手段，和电脑上的开发程序的经验一致。

多媒体应用设计

【WaveOut】

如何使用 WaveOut API。请参考录音机演示代码中的播放功能。

【录音机】

程序路径：Apps\应用开发\多媒体应用\录音机\AudioRecorder

描述：演示了录音功能和 wav 文件读写功能。源码完全开放。可以自己灵活设置参数。按下录音按钮，默认开始录音 5 秒钟，录音完成后保存到根目录的 test.wav 文件。按下播放按钮开始播放刚才录制的 test.wav 音频文件。

【mixer】

【媒体播放器设计】

编译著名的 Tcpmp 的源码。该播放器是 ffmpeg，可以播放除了 rmvb 和 flash 外几乎所有的音频，视频和图像格式。

【照相机】

网络应用设计

【远程桌面项目的实例】
源码路径：
描述：该项目包含电脑端程序 host 和 ce 端程序。双方通过 socket 连接。Ce 把当前桌面通过 socket 网络发送给电脑，电脑就可以实时看到 ce 桌面的一举一动。如下图。并且在电脑窗口中用电脑鼠标和键盘可以远程控制 ce 设备。

【蓝牙控制】 演示如何在程序中实现蓝牙发现，配对，数据通讯。
【wifi 控制】 演示如何在程序中实现 wifi 网卡的启用，禁用，设置 ssid 信息和连接。

系统设置

【关机设置】
注册表 HKCU\ControlPanel\PowerButton
ShutDownHoldTime 代表按键多少秒时间将关机
ShutDownExe 关机按下后启动一个程序，这个值该程序的路径
ShutDownEvent 关机按下后设置一个全局事件，这个值指定事件名
ShutDownAudio 指定关机声音
ShutDownMode 决定关机键按下后的动作。

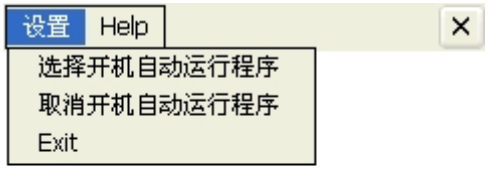
【背光】
注册表 HKCU\ControlPanel\BackLight
BatteryTimeout 和 ACTimeout 分别设置在电池供电和 AC 供电情况下，过多少秒无用户操作系统将自动关闭背光。
可在控制面板【显示】-【背景光】设置。

打开背光 SetDevicePower(L"BAK1:", 0, (CEDEVICE_POWER_STATE)D0);
关闭背光 SetDevicePower(L"BAK1:", 0, (CEDEVICE_POWER_STATE)D4);

P1310V4 版本的背光改变成 PWM 控制亮度。演示代码请参考\P1310\1 - SoftWare\1 - 开发\1 - Apps\系统控制\背光控制\BackLightCtl 工程。

【触摸校准】
(略)

【设置开机启动程序】
演示代码路径：Apps\系统控制\开机启动应用程序\SetStartApp
原理说明：系统开机会依据注册表来决定开机启动的顺序和步骤。注册表设置的路径位于 HKLM\Init。 操作系统的 shell 对应的应用程序是 explorer.exe。

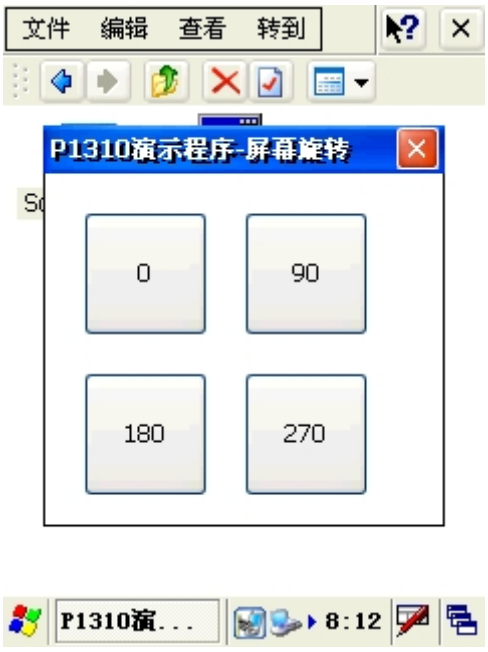


【挂起】
注册表路径 HKLM\Explorer\
Suspend = 0，开始菜单将隐藏挂起，否则显示出挂起。

程序中可以使用下面函式来挂起系统
```
SetSystemPowerState( NULL, POWER_STATE_SUSPEND, POWER_FORCE );
```
或者
```
PowerPolicyNotify(PPN_SUSPENDKEYPRESSED, 0);
```

【任务栏隐藏】
```
HWND hStatusBar=FindWindow(TEXT("HHTaskBar"),NULL);
ShowWindow(hStatusBar,SW_HIDE);
```
隐藏任务栏 可以全屏显示程序这样程序显示的时候就可以全屏显示了

【屏幕旋转】
演示代码路径 P1310\Apps\应用开发\屏幕\ScreenRotate



旋转 90 度后：

屏幕旋转同时旋转了触摸屏坐标，所以旋转后不需要重设触摸。但是这种方式旋转是短效的，在重启系统后恢复原始方向。如果要永久性的设置旋转，需要设置注册表 HKLM\system\GDI\Rotation 项下的 angle 键值，可以设置 0，90,180,270 等数值。(十进制)。

【抓屏】

请参考远程桌面项目实例。

另外介绍 P1310 设备上可以使用的非典型方式。直接显存访问。P1310 的显存开始地址是 0xA0100000. 可以直接访问显存获得当前屏幕。

```
SetKMode(TRUE);
memcpy(buffer, 0xA0100000, 240*320*2);
SetKMode(FALSE);
```

原理：利用 WinCE5 的 Kernel Mode 的特性，应用程序进入 Kernel Mode 后可以直接访问全地址空间。执行 SetKMode 是允许本应用访问内核地址空间。然后把 240*320*2 的一段空间拷贝到缓冲。因为屏幕的分辨率是 240*320，RGB565 格式。所以一帧的 size=240*320*2.如

果要做抓屏程序，最后压缩 jpg 文件。

【透明效果-alpha 混合】

请参考照相机的实例。

2 张图片通过 alpha 运算混合在一起，从而实现透明效果，火焰效果等等。对于一般程序的 alpha 混合，可以使用 GDI 提供的 api 函数 **AlphaBlend**()进行混合运算。P1310 已经默认支持 AlphaBlend()可以直接调用。

针对 RGB565 格式图片的 alpha 函数请参考照相机实例。

【OSD】

OSD：On Screen Display。在屏幕显示。比如显示器的设置都是在屏幕上显示一个 OSD 界面进行操作。不论背景如何，OSD 永远显示在最前面。2440 的硬件没有 2D 支持，并不支持硬件 OSD，演示用软件模拟 OSD 应用。

【编写控制面板程序】

（略）

【修改系统字体 — 使用雅黑字体】

（略）

【多国语言】

（略）

【翻转和甩屏】

P1310 有动作感应功能。并且提供了接口对动作进行设置。

注册表路径和设置

[HKEY_CURRENT_USER\ControlPanel\Gsensor]

"AutoRotate"=dword:0 // 设置成 1 时候，系统检测到旋转会自动旋转。

"AutoMute"=dword:1 // 设置成 1 时候，翻转后，会自动静音

"BackLightCtl"=dword:1 // 设置成 1 时候，翻转后，会自动关闭背光

"ExePathOnShack"="" // 设定一个路径，检测到甩动和剧烈震动后自动调用程序

"ExePathOnTap"=""// 设定一个路径，检测到 tap 后自动调用程序

【滚动的小球】

代码路径：\P1310\1 - SoftWare\1 - 开发\1 - Apps\系统控制\重力加速度传感器\RunBall\

周期性访问加速度传感器，将查询到的加速度值反映到窗口位置，使得窗口会随着 P1310 的位置在屏幕上往各个方向移动。效果像是一个放在乒乓球板上的球。该程序仅仅演示加速度传感器的作用，应用中可以使用其他显示技术可以获得更好的效果。

类似的原理，使用加速度传感器还可以作出 iphone 上的各种效果，如晃动的果冻，逐渐倒空的水杯等等……

驱动

【编写驱动程序】

一、流式驱动程序的设计：（在PB中添加驱动）

在使用的BSP包的Driver目录下，新建一驱动目录，假设选择为Gpio；在Gpio目录下新建Gpio.def文件，用于表示dll的导出函数；新建一Gpio.cpp文件，在Gpio.cpp文件中添加如下函数：

LED _Init　　LED _Deinit　　LED _Open　　LED _Close　　LED _IOControl
LED _PowerUp　LED _PowerDown　　LED _Read　LED _Write　LED _Seek

更改Gpio.def文件中的内容，标识DLL导出的函数

例如

```
LIBRARY   Gpio
 EXPORTS
 LED_Init
 LED_Deinit
 LED_Open
 LED_Close
 LED_PowerUp
 LED_PowerDown
```

LED_Write
LED_Read
LED_Seek
LED_IOControl
注意其中的文件对应关系，在这里LED为设备文件名，实际文件名为Gpio.def

在BSP包里别的驱动目录下拷贝一makefile文件，该文件指出了驱动程序的链接和编译方法，改文件仅仅一行：

```
!INCLUDE $(_MAKEENVROOT)\makefile.def
```

在BSP包里别的驱动目录下拷贝一sources文件，该文件用于设置链接器和编译器，指出驱动程序的编译和链接方法。

```
RELEASETYPE=PLATFORM                    //发布类型：BSP包中的一个驱动
TARGETNAME=Gpio                         //驱动动态库名称
TARGETTYPE=DYNLINK                      //目标类型：动态库
DLLENTRY=DllEntry                       //动态库入口函数名称：DllEntry

TARGETLIBS= \                           //链接需要的库
    $(_COMMONSDKROOT)\lib\$(_CPUINDPATH)\coredll.lib \

INCLUDES=$(INCLUDES);../../inc

SOURCES= \                              //要编译的原代码文件
    Gpio.c
```

编辑目录\smdk2440a\src\drivers下的dirs文件。用文本编辑器打开该文件，找到“DIRS=”等式，在该等式中插入一行，如下

```
DIRS=ceddk      \
...(省略)...    \
 SDHC       \
 touch      \
 wavedev    \
 Gpio       \                   //注意文件名      此为要插入的行
```

插入“Gpio ”以后，Platform Builder在编译操作系统时，就会自动编译Gpio驱动程序。编辑完该文件以后，保存该文件。

用Platform Builder 5.0打开构建的S3C2440平台，在Workspace窗口中选择FileView选项卡，然后选择Gpio。对比修改前的图可以看到Gpio后少了(excluded from build),说明编译操作系统时将同时自动编译该驱动。

在Workspace窗口中选择PlatformView选项卡，选择Platform.bib文件，单击该文件。在Platform Builder中打开该文件，在该文件中加入

Gpio.dll　　$(_FLATRELEASEDIR)\Gpio.dll　　NK　SH

指明在生成Windows CE内核映像时自动将Gpio.dll加入到内核映像中。上述代码的含义是：$(_FLATRELEASEDIR)表示操作系统映像的生成目录，内核映像（NK.nb0或NK.bin文件）中的Gpio.dll模块来自该目录下的Gpio.dll文件。SH指明该文件的属性：S指明为系统文件；H指明为隐藏文件。

单击Platform.reg文件，在Platform Builder中打开该文件，在该文件中加入

```
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\LED]
"Prefix"="LED"            //设备名
"Dll"="Gpio.dll"          //设备驱动程序编译后的动态链接库的名称
"FriendlyName"="Gpio "
"Index"=dword:0           //该设备的索引号
"Order"=dword:0           //设备驱动程序的加载顺序
```

如此，一个简单的 LED 驱动就添加并最终会打包到 WinCE 的 image 中。

二、Windows CE.Net中的中断处理

在Windows CE中,各种外设是通过中断和Windows CE的核心进行通信的。其过程为：

1）硬件设备产生硬件中断。

2）OAL（OEM Adaption Layer，即原始设备制造商适配层）完成硬件中断到逻辑中断（SYSINTR）的转换。

3）系统识别逻辑中断，同时进行相应的处理。

当系统发现中断以后，又将分为两部分进行处理。它们分别为：

- 中断服务例程（Interrupt Service Routine，ISR）。
- 中断服务线程（Interrupt Service Thread，IST）。

1. 中断服务例程ISR

ISR的主要作用是；

3　如果数据可能丢失或者被下一个中断改写，则ISR将数据从设备读取到缓冲区中。

4　ISR清楚该设备上的中断条件。

5　ISR向内核返回一个SYSINTR。

6　内核设置供IST等待的中断事件。

7　调度程序调度等待的IST以执行后续的中断服务操作。

2. 中断服务线程IST

中断服务线程IST是运行在用户空间的中断服务线程,它负责接到系统逻辑中断号之后的中断服务处理。中断服务线程IST在运行的时候首先要注册自己,然后再与一个系统事件关联。通常情况下，中断服务线程IST使用InterruptInitialize函数来注册自己，然后使用WaitForSingleObject函数来等待终端请求事件。一段时间以后，系统的调度器就会调度这个线程工作，来进行中断事务的处理。

通常情况下一个IST的具体实现过程为：

1　创建一个事件。

2　得到系统的中断号。

3　创建一个挂起的中断服务线程IST。

4　设置中断服务线程IST的优先级别。

5　调用函数InterruptInitialize 通知系统注册中断。

6　恢复中断服务线程IST，IST开始服务。

Windows CE.Net中断发生和处理的过程为：

2　硬件设备产生中断信号。

3　中断信号被异常处理程序（ExcEption handler）捕获。

4　异常处理程序激活挂在的中断服务例程ISR。

5　中断服务例程ISR响应硬件，并操作硬件设备。

6　中断服务例程ISR将中断影射。

7　中断服务例程ISR设置相应的核心中断标识。

8　中断服务调度程序与OAL例程合作，设置中断事件，操纵在特定中断事件上的等待队列。

9　OAL例程操作硬件，改变硬件的中断状态，将其设置为开中断。

10　中断事件激活系统调度程序，在相隔一段时间以后调度中断服务线程。

11　中断服务线程处理中断相关的事件，并调用系统的支持库。

12　系统的支持库访问硬件设备，完成对设备的操作。

特别说明：

• 系统的中断响应是分阶段限制的。当核心的处理程序已经捕获了中断以后，将会禁止所有同级别的或者较低级别的中断。如果在这个过程当中，有另外的级别较高的中断发生的话，将会出现中断嵌套的情况。

• 当中断事件被设置以后，中断服务例程ISR的任务就已经完成了。

• 当处理过程完成了第一阶段的任务以后，系统就没有必要再限制除同种中断之外的其他设备事件。因此系统会开放这些中断。当最后中断处理完毕的时候，系统将会彻底的恢复到常规状态。

•设置事件之后到中断服务线程IST的调度是有一个短暂的时间间隔，也就是说并不是直接将控制转移到中断服务线程IST上。

•中断服务线程IST的优先级一般高于普通的线程，正是这种特性使得它们可以抢占普通线程的时间片来优先对中断进行处理。

三、带中断处理的流驱动实验的编写

ISR一般在将Windows CE 移植到某一处理器时已经完成，驱动开发人员不必再编写。那么如何让ISR通知IST中断发生呢？可以利用InterruptInitialize()向内核注册一个事件。该函数的原型及各参数的含义如程序清单：

```
BOOL InterruptInitialize(
                    DWORD idInt,      //中断的逻辑中断号
                    HANDLE hEvent,    //事件，中断发生时，ISR将触发该事件
                    LPVOID pvData,    //传递给OEMInterruptEnable()函数的数据块指针
                    DWORD cbData,     //pvData的大小
                   );
```

调用该函数后，就将该中断（对应的逻辑中断号为idInt）与事件hEvent关联起来，并使能该中断。当该中断触发时，ISR触发事件hEvent生效。

由此可见，中断服务线程IST要做的工作就是：在IST线程中等待事件hEvent生效，即等待中断发生；然后处理中断要做的所有工作；最后调用内核函数voidInterruptDone（DWORD idInt）（其中idInt为中断的逻辑中断号）通知内核中断处理已经结束。

为了取得硬件中断源的某一个中断对应的逻辑中断号，Windows CE提供了I/O请求命令

IOCTL_HAL_REQUEST_SYSINTR，用于申请该中断对应的逻辑中断号。

程序编写的一般步骤：

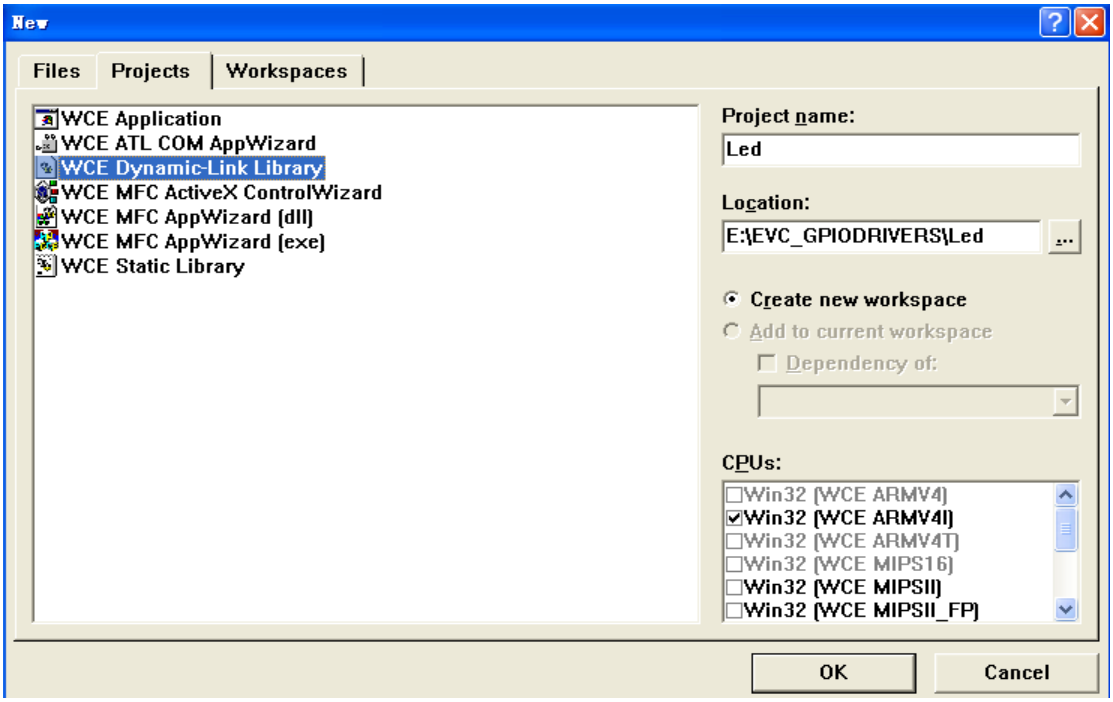
1.需要定义一个全局变量g_bKillIST，它用于当驱动卸载时通知中断服务线程退出，这样才能完全卸载驱动。

2.为GPIO寄存器地址申请对应的虚拟地址，在Windows CE中，程序访问的地址都是虚拟地址，因此要访问硬件物理地址，必须将物理地址空间映射到虚拟空间。

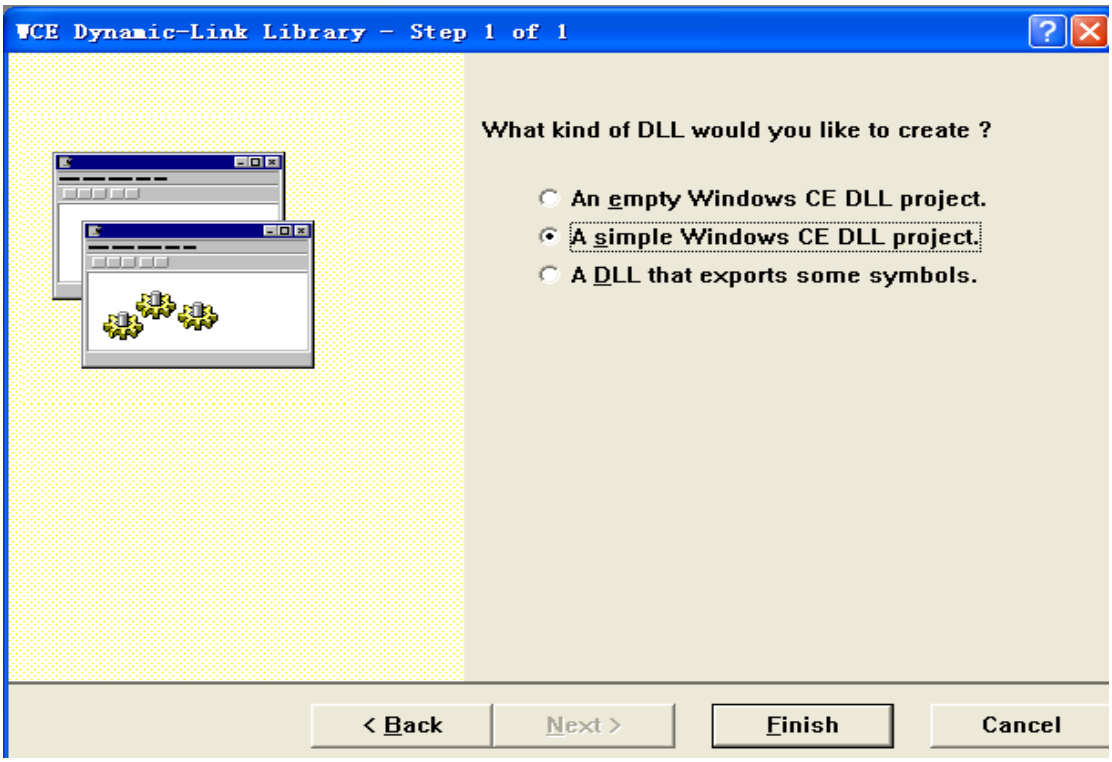
VirtualAlloc()函数的功能是申请一块虚拟内存空间。VirtualCopy()函数的功能是将VirtualAlloc()函数申请到的虚拟空间（起始地址为v_pIOPregs)映射到GPIO寄存器的物理地址（PVOID）(S3C2440A_BASE_REG_PA_IOPORT >> 8)。经过映射后，通过全局结构体变量指针v_pIOPregs就可以访问GPIO寄存器了。

四、在EVC下面led驱动例程的编写示例

1.打开EVC，选择File/new 对话框，选择 Projects WCE Dynamic-Link Library，输入工程名字Led,工程目录E:\EVC_GPIODRIVERS\Led，处理器指令集 ARMV4I. 点击 OK。



2.选择如下，点击 Finish。

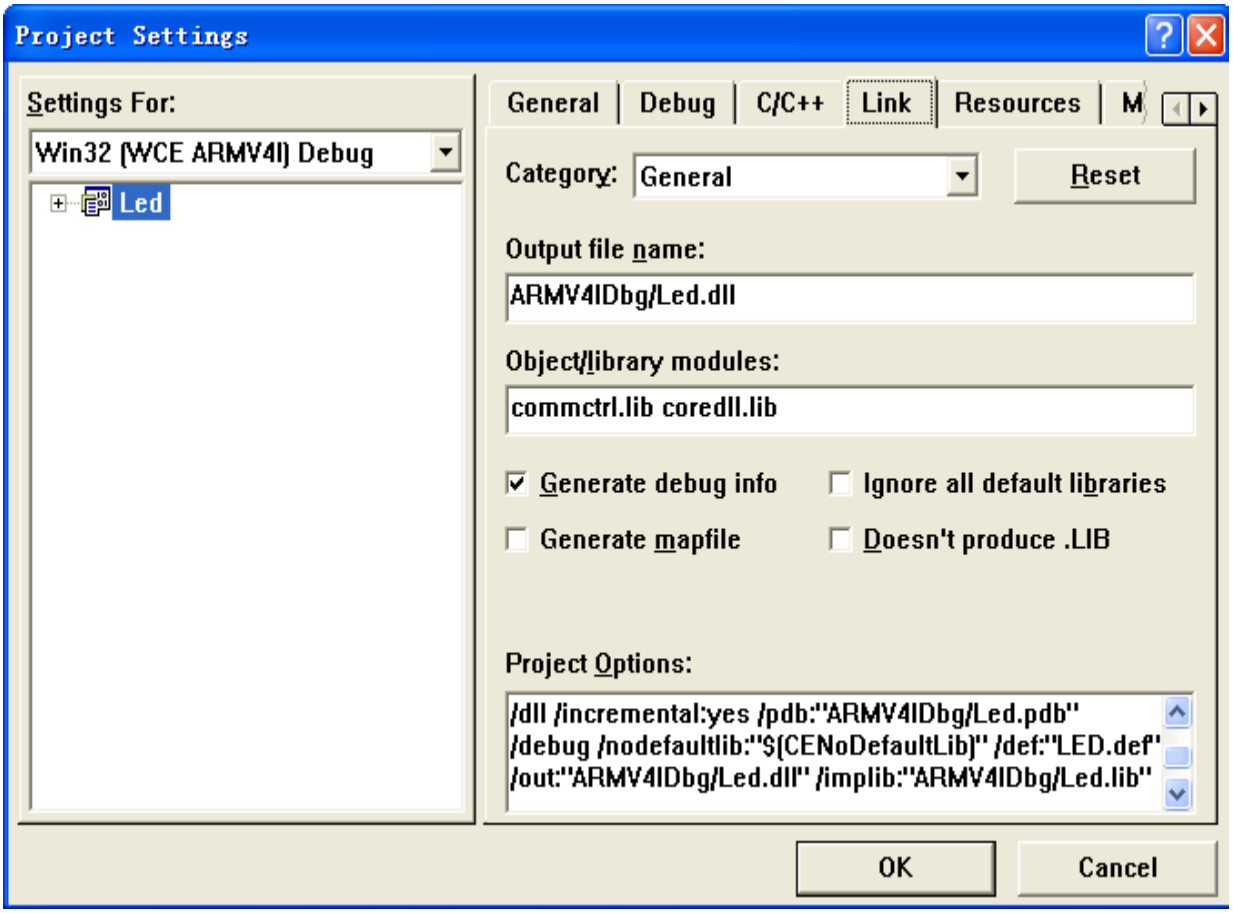


3. E:\EVC_GPIODRIVERS\Led 目录下新建 Led.def 文件，文件内容：

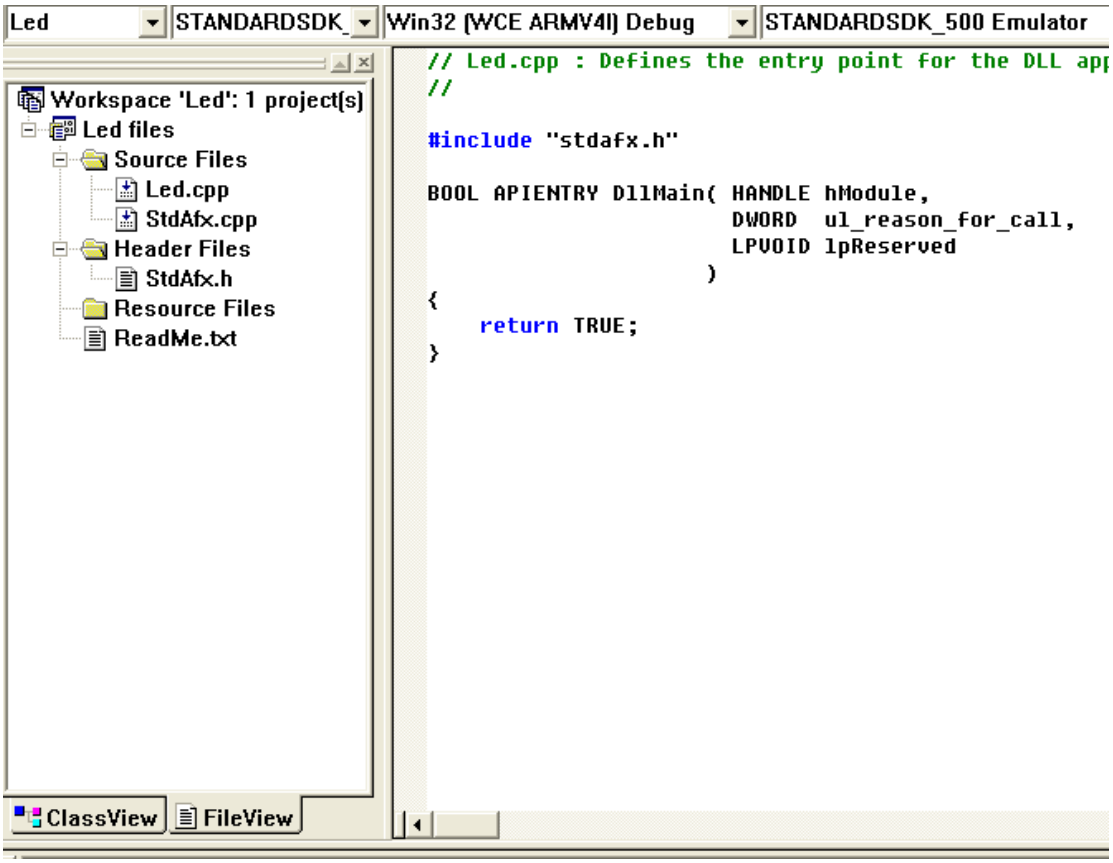
```
LIBRARY GPIO

EXPORTS LED_Init
        LED_Deinit
        LED_Open
        LED_Close
        LED_Read
        LED_Write
        LED_Seek
        LED_IOControl
        LED_PowerDown
        LED_PowerUp
```

4. 把/def:”Led.def” 添加到如下图所示位置，点击 OK。



5.打开如下图所示选项卡，在 Led.cpp 中添加代码。



附件（Led.cpp）：

```
// LED.cpp : Defines the entry point for the DLL application.
//
#include "stdafx.h"

#include "s2440.h"
#include "pkfuncs.h"

void Virtual_Alloc();
BOOL LEDGpioInit();


#define IO_CTL_LED_ON  0x05

#define IO_CTL_LED_OFF 0x0a


#define LED_OUPUT()   s2440IOP->rGPCCON  = (s2440IOP->rGPCCON  &~(3
<< 12)) | (1<< 12); // 按键灯 GPC6 == OUTPUT.
```

```
#define LED_ON()   s2440IOP->rGPCDAT |= (0x01<<6)
#define LED_OFF()   s2440IOP->rGPCDAT &=~(0x01<<6);

volatile IOPreg *s2440IOP = (IOPreg *)IOP_BASE;

BOOL APIENTRY DllMain( HANDLE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                     )
{
    switch (ul_reason_for_call)
    {
        case DLL_PROCESS_ATTACH:
            RETAILMSG(1, (TEXT("LED: DLL_PROCESS_ATTACH\r\n")));
            DisableThreadLibraryCalls((HMODULE) hModule);
            break;
        case DLL_THREAD_ATTACH:
            break;
        case DLL_THREAD_DETACH:
            break;
        case DLL_PROCESS_DETACH:
            RETAILMSG(1, (TEXT("LED: DLL_PROCESS_DETACH\r\n")));
            break;
    }
    return TRUE;
}

void Virtual_Alloc(void)
{
    // GPIO Virtual alloc
    s2440IOP          =          (volatile          IOPreg          *)
VirtualAlloc(0,sizeof(IOPreg),MEM_RESERVE, PAGE_NOACCESS);
    if(s2440IOP == NULL) {
        RETAILMSG(1,(TEXT("For         s2440IOP:         VirtualAlloc
faiLED!\r\n")));
    }
    else {

    if(!VirtualCopy((PVOID)s2440IOP,(PVOID)(IOP_BASE),sizeof(IOPreg
),PAGE_READWRITE | PAGE_NOCACHE )) {
            RETAILMSG(1,(TEXT("For        s2440IOP:        VirtualCopy
faiLED!\r\n")));
        }
    }
```

```
}

BOOL LEDGpioInit()
{
    RETAILMSG(1,(TEXT("LED_Gpio_Setting----\r\n")));
    LED_OUPUT();
    return TRUE;
}

//-------------------------------------------------------------------
-------------
BOOL LED_IOControl(DWORD hOpenContext,
                DWORD dwCode,
                PBYTE pBufIn,
                DWORD dwLenIn,
                PBYTE pBufOut,
                DWORD dwLenOut,
                PDWORD pdwActualOut)
{
    switch(dwCode)
    {
    case IO_CTL_LED_ON:
        LED_ON();
        break;
    case IO_CTL_LED_OFF:
        LED_OFF();
        break;
    default:
        break;
    }

    RETAILMSG(1,(TEXT("LED:Ioctl code = 0x%x\r\n"), dwCode));
    return TRUE;
}

//-------------------------------------------------------------------
-------------
//-------------------------------------------------------------------
-------------
BOOL LED_Init(VOID)
{
    RETAILMSG(1,(TEXT("USERLED: LED_init\r\n")));
    Virtual_Alloc();
    LEDGpioInit();
```

```
    return TRUE;
}

//-------------------------------------------------------------------
-------------
//-------------------------------------------------------------------
-------------
BOOL LED_Deinit(DWORD hDeviceContext)
{
    BOOL bRet = TRUE;

    RETAILMSG(1,(TEXT("USERLED: LED_Deinit\r\n")));

    return TRUE;
}

//-------------------------------------------------------------------
-------------
//-------------------------------------------------------------------
-------------
DWORD  LED_Open(DWORD  hDeviceContext,  DWORD  AccessCode,  DWORD
ShareMode)
{
    RETAILMSG(1,(TEXT("USERLED: LED_Open\r\n")));
    return TRUE;
}

//-------------------------------------------------------------------
-------------
//-------------------------------------------------------------------
-------------
BOOL LED_Close(DWORD hOpenContext)
{
    RETAILMSG(1,(TEXT("USERLED: LED_Close\r\n")));
    return TRUE;
}

//-------------------------------------------------------------------
-------------
//-------------------------------------------------------------------
-------------
void LED_PowerDown(DWORD hDeviceContext)
{
    RETAILMSG(1,(TEXT("USERLED: LED_PowerDown\r\n")));
```

```
}

//-------------------------------------------------------------------
-------------
//-------------------------------------------------------------------
-------------
void LED_PowerUp(DWORD hDeviceContext)
{
   RETAILMSG(1,(TEXT("USERLED: LED_PowerUp\r\n")));
}

//-------------------------------------------------------------------
-------------
//-------------------------------------------------------------------
-------------
DWORD LED_Read(DWORD hOpenContext, LPVOID pBuffer, DWORD Count)
{
   RETAILMSG(1,(TEXT("USERLED: LED_Read\r\n")));
   return TRUE;
}

//-------------------------------------------------------------------
-------------
//-------------------------------------------------------------------
-------------
DWORD LED_Seek(DWORD hOpenContext, long Amount, DWORD Type)
{
   RETAILMSG(1,(TEXT("USERLED: LED_Seek\r\n")));
   return 0;
}

//-------------------------------------------------------------------
-------------
//-------------------------------------------------------------------
-------------
DWORD  LED_Write(DWORD  hOpenContext,  LPCVOID  pSourceBytes,  DWORD
NumberOfBytes)
{
   RETAILMSG(1,(TEXT("USERLED: LED_Write\r\n")));
   return 0;
}
```

本程序所需的 2 个头文件 s2440.h，pkfuncs.h 在例程 Led 目录下。

五、在EVC下面led应用程序的编写示例

1. 动态加载驱动

```
HANDLE m_hIR;
m_hIR = RegisterDevice(TEXT("LED"),1,TEXT("LED.dll"),2);  //这里是你的流驱动的文件名
if(m_hIR == NULL){
       ::MessageBox(NULL,NULL,_T("load driver fail!"),MB_OK);
        }
else
       ::MessageBox(NULL,NULL,_T("load driver sucess"),MB_OK);
```

2. 卸载驱动

```
::MessageBox(NULL,NULL,_T("unload driver"),MB_OK);
DeregisterDevice(m_hIR);
```

3.打开驱动

```
HANDLE hLed = INVALID_HANDLE_VALUE;
// 打开 GPIO 驱动
hLed = CreateFile(TEXT("LED1:"), GENERIC_READ | GENERIC_WRITE, 0,
                   NULL, OPEN_EXISTING, 0, 0);

if (hLed == INVALID_HANDLE_VALUE)
   {
       MessageBox(_T("打开 LED 驱动失败!"));
       return;
   }
else
       MessageBox(_T("打开 LED 驱动成功!"));
```

4. 向驱动发控制命令

```
#define IO_CTL_LED_ON   0x05
#define IO_CTL_LED_OFF  0x0a

::DeviceIoControl(hLed, IO_CTL_LED_ON, NULL, 1, NULL, 0, NULL, NULL);
::DeviceIoControl(hLed, IO_CTL_LED_OFF, NULL, 1, NULL, 0, NULL, NULL);
```

5.关闭驱动

```
if(hLed!=INVALID_HANDLE_VALUE)
   {
       CloseHandle(hLed);
       hLed =INVALID_HANDLE_VALUE;
```

```
}
```

【加载驱动程序】

1. 自动加载方式

在注册表 HKLM\drivers\builtin\项下增加新的驱动的方式,比如增加一个 adc.dll 驱动:

```
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\ADC]
    "Dll"="adc.dll"
    "Prefix"="ADC"
    "Index"=dword:1
    "Order"=dword:1
```

那么在开机时候，wince 会自动加载 builtin 项下的所有驱动，包括我们添加的 ADC 驱动。

2. 手动加载方式

有 2 个 api 可以用于加载驱动，RegisterDevice 和 ActivateDeviceEx。 前者更简洁，后者需要设置注册表并提供注册表路径。 推荐后者， 后者提供更灵活的配置，加入电源管理并且方便驱动全局管理。

```
WCHAR  szRegPath[MAX_PATH] = {L"Drivers\\camera"};
g_hCameraDriver = ActivateDeviceEx(szRegPath, NULL, 0, NULL);
```

系统平台定制

安装好 platfrom builder 5：简称 PB5.

将 P1310 的 BSP 拷贝到 platfrom 目录。

点 PB5 的 File – Manager Catalog Items – Import，在弹出的窗口中选择 P1310BSP 目录下的 cec 文件。

打开 PB5，点 File – New Platform， 根据向导输入工程名和选择 BSP 包 SAMSUNG SMDK2440:ARMV4I 和模版。 模版推荐 Mobile Handheld。后面 next。

在 PB5 菜单 Platform – Settings… 窗口中选择合适的 locale 和 Build Options。Locale 代表系统的语言环境。 Build Options 选择 SYSGEN_SHELL=1, IMGEBOOT=1,IMGNOTALLKMODE=1 这三项。

在 PB5 的 catalog 窗口选择自己需要的系统组件，右键点击需要的组件 Add to OS Design 即可加入组件。加入后，在左边的工程项目窗口可以看到新加入的组件。

组件选择完毕后，菜单 Build OS – Sysgen。 成功编译和打包后出现 nk.bin 就是最终需要的操作系统的镜像。