安之谋 HMI977 人机界面 CE 开发手册

版 权 声 明

北京安之谋科技有限公司，多年来一直致力于高质量嵌入式软硬件的开发。由安之谋科技提供 HMI977 人机界面平台可运行独家提供的 CE6，除了具有常见的功能之外，还提供了各种方便客户二次开发和生产的功能。

1. 产品特点和定位

本产品采用了易于生产和维护的国产台系处理器。同时专门移植了 Windows CE 6.0 BSP。具有运行稳定，功耗低等特点。

本产品可用于一些运行 CE 或 Linux 的老产品的平滑替换和平滑升级。安之谋科技的软件团队可以在底层为客户定制，从而达到和 Atmel/ SamSung/TI/Freescale/NXP/CirrusLogic 等多家平台实现兼容。让客户的原有 APP 能够直接在新的平台上运行。

2. 客户如何二次开发

安之谋科技还特别提供定制的了 Windows CE 6.0 R3 的 SDK，用于客户编写和调试应用程序。

建议客户使用 MFC 或者.Net 开发界面程序。

开发环境可以是 Visual Studio 2005，也可以是 Visual Studio 2008。SDK 的安装和调试，请参照下面特定章节。

3. 如何选择固件

安之谋科技提供了各种类型的固件供客户选择。选择一个合适的固件，能让产品更加稳定和易用。

选择固件首先要看开发应用程序使用什么语言，如果使用 C#或者 VB.NET，那么最好选择集成.NET 的固件。.NET 分两个版本 2.0 和 3.5，在编写应用程序之前要确定好版本。如果选择 MFC 开发，则可以使用精简版的固件。

另外，固件还分为几种不同类型的文件系统：

RAMROM 文件系统：默认固件采用这种文件系统。Flash 默认映射为 NandFlash 目录，，用户需要注意的是只有这个目录能够保存文件，在其他目录产生的文件都会在重新启动后丢失。这种类型的文件系统的优点是不容易积累系统的垃圾。

ROMONLY 文件系统：这种类型的文件系统，把 Flash 存储映射为根目录，系统所有位置的文件都将得到保存。这个类型的固件适用于需要自行安装一些软件的客户。

RAMBASE 文件系统：这种类型的固件，Flash 默认映射为 NandFlash 目录，但是系统不保存任何注册表信息到 NandFlash 目录，每次重新启动都会是一个全新的系统。

4. 烧写固件前的准备

注意：**这里介绍的是裸板烧写固件的方法。如果开发板能正常启动的，请参照“使用 U 盘升级固件”，直接用 U 盘或者 SD 卡升级烧写固件更加方便一些。**

安之谋科技提供的 CE 固件分为以下几个文件：

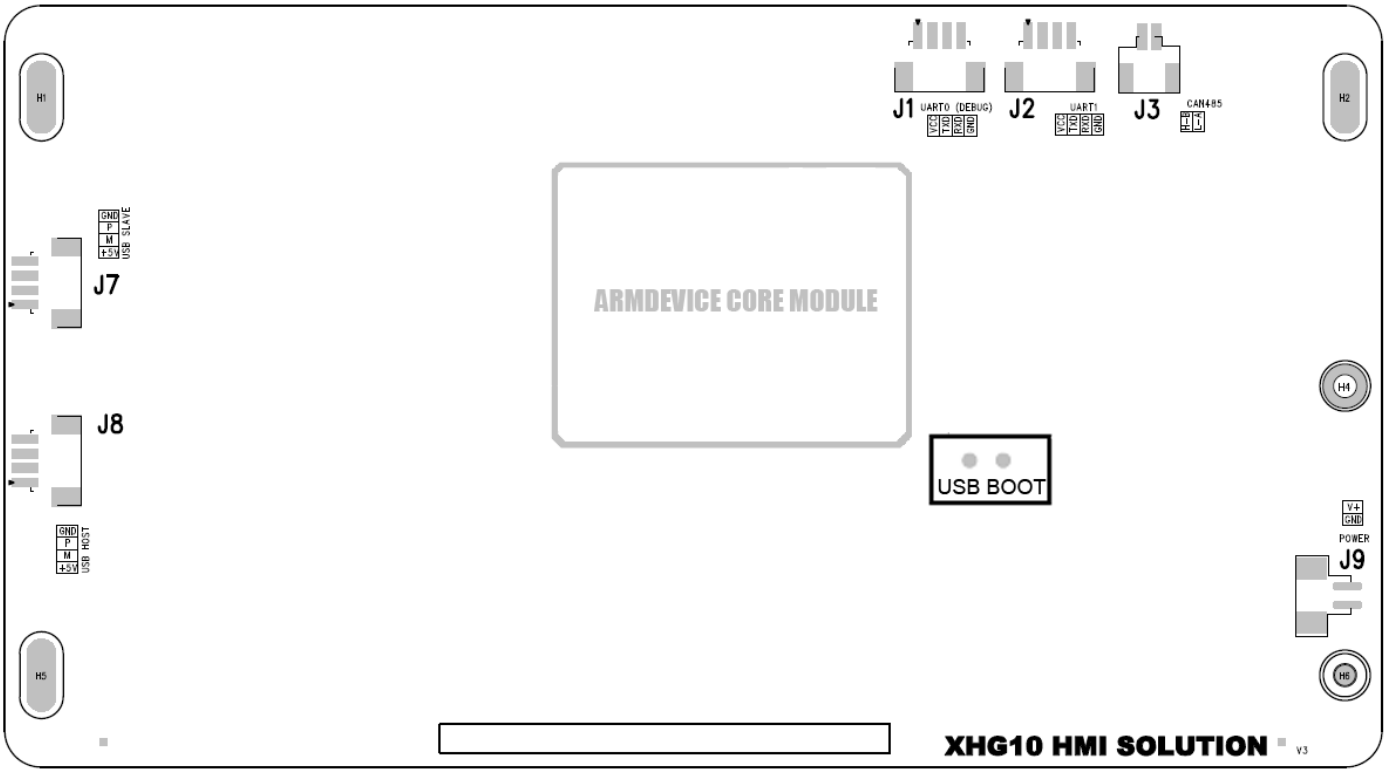
NANDBOOT.bin	基于 NandFlash 的启动文件
EBOOT.bin	基于 NandFlash 的 EBOOT

Logo.bmp	启动后显示的第一屏，一般用于显示一个 Logo
ENK.bin	用于烧写的 NK 文件

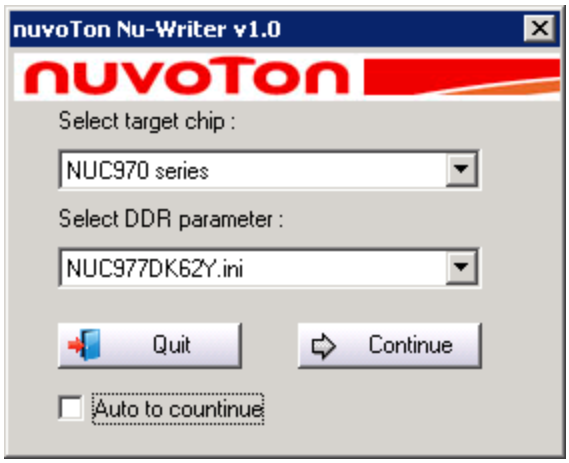
另外，烧写还需要 NuWriter 工具。该工具为安之谋修改版，随固件一起发布。
PC 机建议运行 Windows7 操作系统。并安装 USB 驱动程序 WinUSB4NuVCOM_NUC970.exe

5. 烧写固件

开始烧写固件时，首先需要将下图中的 USB Boot 两个测试点短路后再上电。然后用 USB 线连接 PC 机和 J7 接口。这时 PC 机会提示加载之前安装好的 USB 驱动程序。

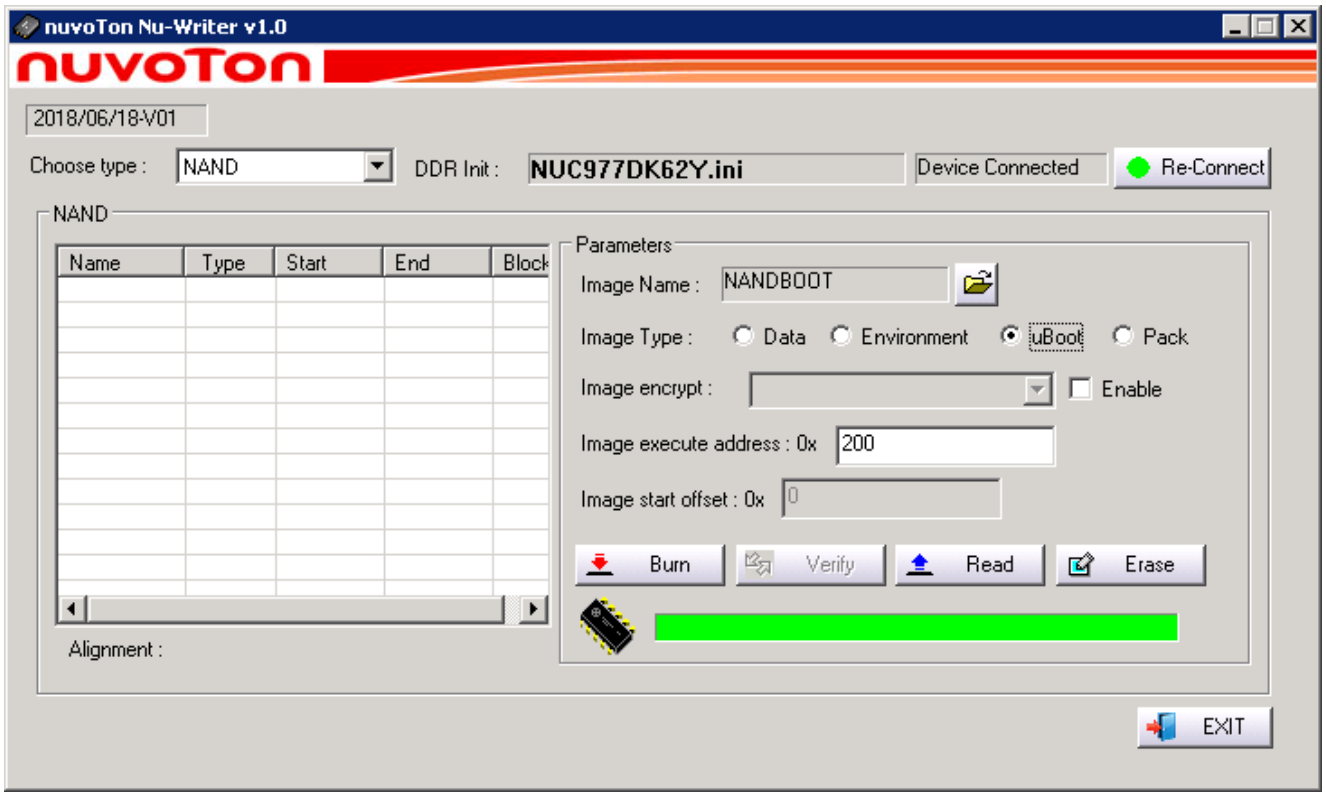


设备驱动程序加载完成之后，就可是运行 NuWriter 工具进行烧写了。第一个界面按下图选择， 然后点“Continue”按钮。



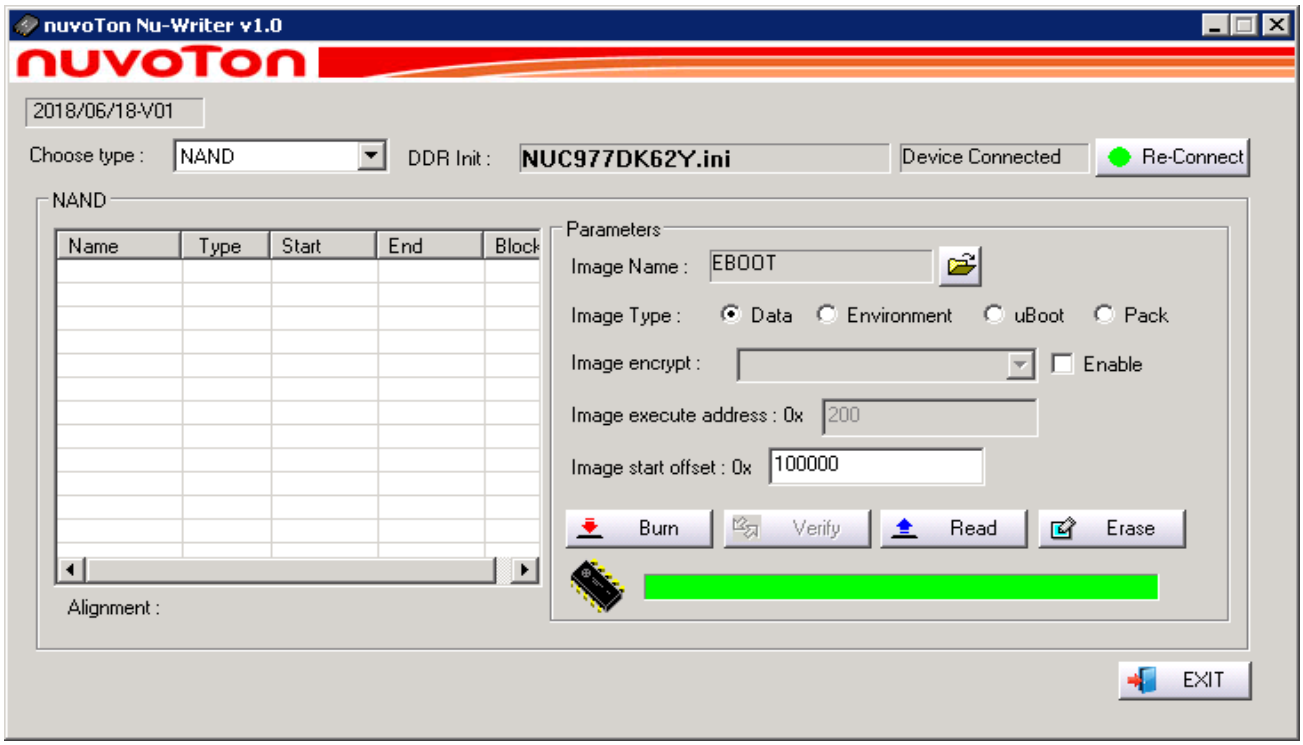
4.1 烧写 NANDBOOT

进入烧写界面后，“Choose Type”选择 NAND，
点 Image Name 右侧按钮，选择文件 NANDBOOT.bin
Image Type 设置为 uBoot，如下图，设置好之后，按下 Burn 按钮。然后等待烧写完成。



4.2 烧写 Eboot，Logo，ENK

点 Image Name 右侧按钮，选择文件 Eboot.bin
Image Type 设置为 Data，
Image Start Offset 设置为 0x100000，
如下图，设置好之后，按下 Burn 按钮。然后等待烧写完成。



按照相同的方法烧写，Logo.bmp 烧写 Offset 为 0x200000， ENK.bin 的烧写 Offset 为 0x400000

4.3 烧写完成

烧写完成后重新上电，然后等待初始化完成即可。等待时间大约是 2 分钟。

6. NandFlash 的分配

```
NAND memory layout   (Armdevice HMI977)

0x00000000 - 0x00100000    NANDBOOT
0x00100000 - 0x00200000    EBOOT
0x00200000 - 0x00400000    LOGO BITMAP
0x00400000 - 0x02000000    NK
0x02000000 -               BINFS and DOS PARTITION
```

7. BSP 特性介绍

安之谋科技提供的 WinCE BSP，以丰富的贴近用户的功能，在业内受到广大客户的好评。
目前安之谋科技为国内外多家客户提供独家定制的下列平台的 Windows CE 6.0 BSP：

SamSung S3C2416 Windows CE 6.0 BSP.
SamSung S3C6410 Windows CE 6.0 BSP.
Atmel At91SAM9G10 Windows CE 6.0 BSP.
Atmel At91SAM9G45 Windows CE 6.0 BSP.
Cirrus Logic EP9608 Windows CE 6.0 BSP.
TI AM335X Windows CE 6.0 BSP.

HMI977 人机界面 Windows CE 6.0 BSP 中实现提供的驱动有：

LCD 显示
Touch Pannel
串口
USB
SDHC
Ethernet
CAN
等等

以下为目前在 HMI977 人机界面 Windows CE 6.0 BSP 中实现的额外功能。

6.1 开机 Logo

支持美观的开机 Logo 以及加载进度条。

6.2 支持 MultiBin 技术

实现了 NAND Flash 的 XIP，减少内存占用，加快启动速度。

6.3 U 盘更新工具

只需要将 ENK 文件放在 U 盘根目录，然后插入 U 盘，即可自动更新。方便终端用户更新 ROM 或程序。

6.4 Windows CE6.0 R3 的 SDK

提供 Windows CE6.0 R3 的 SDK，支持模拟调试和真机在线调试。用户可以非常方便的开发和调试应用程序。

8. 安装 SDK

安装 SDK 要求本机已经安装了 Visual Studio 2005 SP1。该 SDK 支持虚拟机调试和真机调试。
安装完 SDK 之后，就可以打开 HMITest 工程进行编译和调试了。HMITest 工程源代码请自行下载。

9. 访问 GPIO

设备名 PIO1
控制代码：

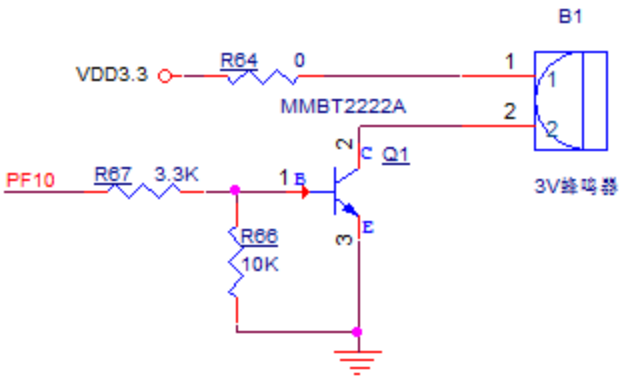
```
// IOCTL Code For GPIO
#define IOCTL_SET_DIR                   0x01
#define IOCTL_READ_DATA                 0x02
#define IOCTL_WRITE_DATA                0x03
#define IOCTL_SET_INTERRUPT             0x04
```

IOCTL_SET_DIR：设置 GPIO 的输入输出方向
IOCTL_READ_DATA：读取 GPIO 的输入值
IOCTL_WRITE_DATA：设置 GPIO 的输出值
IOCTL_SET_INTERRUPT：设置 GPIO 中断的类型
示例代码:

```
DWORD dwIOParam[3];
HANDLE hPioHandler;
hPioHandler = CreateFile(TEXT("PIO1:"), GENERIC_READ | GENERIC_WRITE, 0, NULL, OPEN_EXISTING, 0, NULL);
if(hPioHandler != INVALID_HANDLE_VALUE)
{
	dwIOParam[0]= GPIO_B;			//GPIO 为 PB.6
	dwIOParam[1]= 6;			//GPIO 为 PB.6
	dwIOParam[2]= GPIO_OUTPUT;		//设置为 GPIO_OUTPUT，如果是读取就设置为 GPIO_INPUT
	DeviceIoControl(hPioHandler, IOCTL_SET_DIR, dwIOParam, 3*sizeof(DWORD), NULL, 0, NULL, NULL);


	dwIOParam[0]= GPIO_B;		// GPIO 为 PB.6
	dwIOParam[1]= 6;		// GPIO 为 PB.6
	dwIOParam[2]= 0;			//这里 0 是输出低电平点亮，1 是输出高电平关掉。
	DeviceIoControl(hPioHandler, IOCTL_WRITE_DATA, dwIOParam, 3*sizeof(DWORD), NULL, 0, NULL, NULL);

	 CloseHandle(hPioHandler);
}
```

10. 控制蜂鸣器

蜂鸣器控制电路：



示例代码，切换蜂鸣器的状态

```
DWORD dwIOParam[3];
BYTE bIOVal=0;
HANDLE hPioHandler;
hPioHandler = CreateFile(TEXT("PIO1:"), GENERIC_READ | GENERIC_WRITE, 0, NULL, OPEN_EXISTING, 0, NULL);
if(hPioHandler != INVALID_HANDLE_VALUE)
{
	dwIOParam[0]= GPIO_F;		//蜂鸣器对应的 GPIO 为 PF.10
	dwIOParam[1]= 10;			//蜂鸣器对应的 GPIO 为 PF.10
	dwIOParam[2]= GPIO_OUTPUT;		//设置为 GPIO_OUTPUT
	DeviceIoControl(hPioHandler, IOCTL_SET_DIR, dwIOParam, 3*sizeof(DWORD), NULL, 0, NULL, NULL);

	dwIOParam[0]= GPIO_F;		//蜂鸣器对应的 GPIO 为 PF.10
	dwIOParam[1]= 10;			//蜂鸣器对应的 GPIO 为 PF.10
	DeviceIoControl(hPioHandler, IOCTL_READ_DATA, dwIOParam, 3*sizeof(DWORD), &bIOVal, sizeof(BYTE), NULL, NULL);

	dwIOParam[0]= GPIO_F;		//蜂鸣器对应的 GPIO 为 PF.10
	dwIOParam[1]= 10;			//蜂鸣器对应的 GPIO 为 PF.10
```

```
    dwIOParam[2]= bIOVal?0:1;     //蜂鸣器状态切换
    DeviceIoControl(hPioHandler, IOCTL_WRITE_DATA, dwIOParam, 3*sizeof(DWORD), NULL, 0, NULL, NULL);

    CloseHandle(hPioHandler);
}
```

11. 触摸屏

HMI977 支持两种类型的触摸屏，一种是 4 线电阻屏，一种是 6P 接口的 GT911 电容屏。支持不同的触摸屏需要烧写对应的固件。当自行改装接上电容屏的时候，需要确认 R170 是否移除。自行改装接上电阻屏的时候，需要确认 R170 是否焊上，阻值为 33 欧姆。

HMI977 支持在触摸的时候，联动蜂鸣器。该功能通过如下注册表控制：

```
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\Touch]
    "BeepOnTouch"=dword:1
```

设置 BeepOnTouch=1 的时候，触摸时会有蜂鸣器联动。有这个需求的客户可以在应用程序中自行设置。

12. 控制 LCD 背光

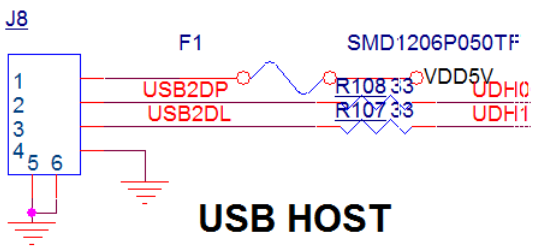
背光控制用 PG.10。
PG.10 的控制代码，参考普通的 IO 控制代码。

13. 网口

网口默认设置为 DHCP。可使用同一网段的 PC 机和板子连接，进行远程调试和开发。具体步骤参看《Windows CE 6.0 基于以太网的远程调试》。

14. USB 主口

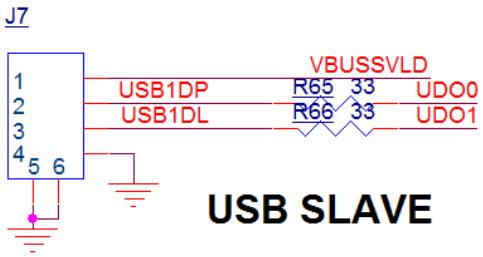
USB 主口采用 PH2.0-4P 接口，接口位号为 J2。线序如下：



请按照正确线序制作转接线。

15. USB 从口

USB 主口采用 PH2.0-4P 接口，接口位号为 J1。线序如下：



请按照正确线序制作转接线。
USB 从口可用于裸板烧写。
在 WindowsCE 中默认配置为 FileSync 类型的 ActiveSync 端口。可以通过该端口进行文件拷贝，或者进行程序调试。
WindowsXP 主机需要安装“Microsoft ActiveSync 4.5”软件。
Windows7 以上的主机，需要安装“Windows Mobile Device Center 6.1” 软件。
具体参看微软公司的相关文档。

16. 软件复位

软件复位，可使用以下代码：

```
#define IOCTL_KLIB_USER                     256
#define IOCTL_USER_REBOOT     CTL_CODE(FILE_DEVICE_HAL, IOCTL_KLIB_USER + 301, METHOD_BUFFERED, FILE_ANY_ACCESS)
HANDLE    hFile;
DWORD nBootType=1;
hFile = CreateFile(TEXT("KIP1:"), GENERIC_READ | GENERIC_WRITE, 0, NULL, OPEN_EXISTING, 0, NULL);
if(hFile != INVALID_HANDLE_VALUE)
{
      DeviceIoControl(hFile, IOCTL_USER_REBOOT, &nBootType, sizeof(DWORD), NULL, 0, NULL, NULL);
      CloseHandle(hFile);
}
```

17. 看门狗

看门狗驱动设备节点为 WTD1:，该驱动提供了以下控制接口。

打开看门狗
　　接口代码： WATCHDOG_ENABLE
　　说明：打开看门狗之后，看门狗开始以 2.5 秒左右的最大喂狗间隔开始运行，如果 2.5 秒内没有喂狗动作，系统会复位重启。

关闭看门狗
　　接口代码： WATCHDOG_DISABLE
　　说明：关掉看门狗

开始内核自动喂狗
　　接口代码： WATCHDOG_START_AUTO_FEED
　　说明：如果使用内核自动喂狗，内核会每隔 1 秒喂一次狗，这一功能用于监控内核是否死机。

手动单次喂狗

接口代码： WATCHDOG_FEED

说明：应用程序手动喂狗，可用于监视应用程序是否死掉。如果使用这种喂狗方式，就一定不要再打开自动喂狗模式。

参考代码如下所示:

```
HANDLE hWDTHandler = INVALID_HANDLE_VALUE;

void init_watchdog()
{
        if(hWDTHandler==INVALID_HANDLE_VALUE)
      {
            hWDTHandler = CreateFile(TEXT("WTD1:"), GENERIC_READ | GENERIC_WRITE, 0, NULL, OPEN_EXISTING, 0, NULL);
              RETAILMSG(1, (TEXT("WTD = %x\r\n"), (UINT)hWDTHandler));
      }
}


void enable_watchdog(int mode)
{
      if(hWDTHandler != INVALID_HANDLE_VALUE)
      {
              DeviceIoControl(hWDTHandler, WATCHDOG_ENABLE, NULL, 0, NULL, 0, NULL, NULL);
              if(mode ==0) //manually feed
              {
              }
              if(mode ==1) //auto feed in kernel
              {
                      DeviceIoControl(hWDTHandler, WATCHDOG_START_AUTO_FEED, NULL, 0, NULL, 0, NULL, NULL);
              }
       }
}

void disable_watchdog()
{
      if(hWDTHandler != INVALID_HANDLE_VALUE)
      {
              DeviceIoControl(hWDTHandler, WATCHDOG_DISABLE, NULL, 0, NULL, 0, NULL, NULL);
       }
}

void feed_watchdog()
{
      if(hWDTHandler != INVALID_HANDLE_VALUE)
      {
              DeviceIoControl(hWDTHandler, WATCHDOG_FEED, NULL, 0, NULL, 0, NULL, NULL);
       }
}
```

18. GPIO 按键

假设按键接到 GPIO 口 PJ.4。可使用普通 GPIO 的中断功能，监测按键状态。
示例代码如下：

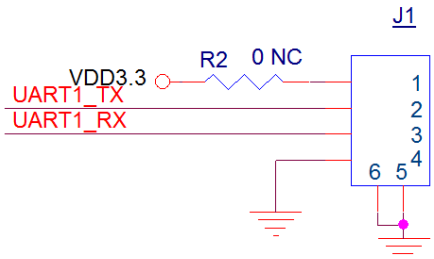
```
DWORD CHMITestDlg::DetectKeyThread( IN PVOID context )
{
      BOOL isKeyDown=FALSE;
      BOOL bRet;
      BYTE bIOVal=0;
      TCHAR szEventName[32];
      DWORD dwIOParam[3];
      CHMITestDlg * pDlg = (CHMITestDlg*)context;

      dwIOParam[0]= GPIO_J;     //按键使用的 GPIO 为 PJ.4
      dwIOParam[1]= 4;          //按键使用的 GPIO 为 PJ.4
      dwIOParam[2]= GPIO_TRIGGER_EDGE | GPIO_EDGE_RISING | GPIO_EDGE_FALLING;   //设置为上升沿+下降沿触发
```

```
    DeviceIoControl(pDlg->hPioHandler, IOCTL_SET_INTERRUPT,
        (LPVOID)dwIOParam, sizeof(DWORD)*3, NULL, 0, NULL, NULL);

    _stprintf(szEventName, TEXT("GPIOEV%d"), NUC_GPIO_INDEX(GPIO_J, 4));
    pDlg->hUserKeyEvent = CreateEvent(NULL, FALSE, FALSE, szEventName);
    if(pDlg->hUserKeyEvent != INVALID_HANDLE_VALUE)
    {
        pDlg->UserKeyThreadRun = TRUE;
        while(pDlg->UserKeyThreadRun)
        {
            WaitForSingleObject( pDlg->hUserKeyEvent, INFINITE );
            if(pDlg->UserKeyThreadRun)
            {
                dwIOParam[0]= GPIO_J;
                dwIOParam[1]= 4;
                bRet = DeviceIoControl(pDlg->hPioHandler, IOCTL_READ_DATA,
                    dwIOParam, 3*sizeof(DWORD), &bIOVal, sizeof(BYTE), NULL, NULL);
                if(bRet == FALSE)
                {
                    pDlg->AppendDisplayText(TEXT("IOCTL_READ_DATA Failed\r\n"));
                    continue;
                }
                if(bIOVal)
                {
                    // 这里收到的是原始的 GPIO 中断，对于按键，需要做一个简单的过滤。过滤掉多余的中断。
                    if(isKeyDown)
                    {
                        pDlg->AppendDisplayText(TEXT("User Key Pressed Up\r\n"));
                        isKeyDown = FALSE;
                    }
                }
                else
                {
                    //这里也一样加上过滤
                    if(!isKeyDown)
                    {
                        isKeyDown = TRUE;
                        pDlg->AppendDisplayText(TEXT("User Key Pressed Down\r\n"));
                    }
                }

            }
        }
    }
    return 0;
}
```

实际运行效果，请参看 HMITest 测试程序。

19. RS232 串口 COM1

COM1 串口为 RS232 电平的三线串口，接口位号为 J1。线序如下：



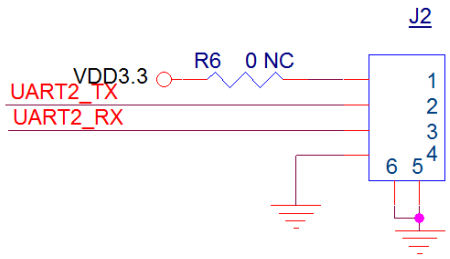
如果外界设备需要 3.3V 供电，可以自行焊上 R2 电阻。
该串口为通用串口，编程接口参看 Windows 串口编程的相关文档即可。

20. COM1 串口调试信息

COM1 在系统启动过程中，可以选择作为调试打印串口。也可以选择不输出调试信息。
这个是通过板上的电阻 R97 来实现控制的。
焊上 R97，则启动过程有打印信息。
去掉 R97，则启动过程没有打印信息。

21. RS232 串口 COM2

COM2 串口为 RS232 电平的三线串口，接口位号为 J2。线序如下：

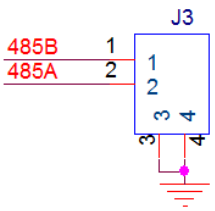


如果外界设备需要 3.3V 供电，可以自行焊上 R6 电阻。

该串口为通用串口，编程接口参看 Windows 串口编程的相关文档即可。

22. RS485 串口 COM3

COM3 串口为 RS485 电平的半双工差分接口，接口位号为 J3。

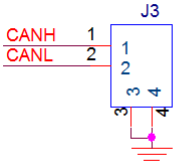


该串口为通用串口，编程接口参看 Windows 串口编程的相关文档即可。

开发板采用了 CAN 接口和 RS485 接口的二选一设计，CAN 和 RS485 同是使用 J3 作为外部接口。通过焊接不同的芯片来实现这一功能。使用 RS485 功能时，请确认硬件是否匹配。

23. CAN 接口

CAN 接口位号为 J3。



开发板采用了 CAN 接口和 RS485 接口的二选一设计，CAN 和 RS485 同是使用 J3 作为外部接口。通过焊接不同的芯片来实现这一功能。使用 CAN 功能时，请确认硬件是否匹配。

设备名 CBS1
控制代码：

```
#define IOCTL_CAN_TX                (1)    //发送一帧数据
#define IOCTL_CAN_RX                (2)    //接收一帧数据
#define IOCTL_CAN_SETUP_RX          (3)    //设置接收的参数
#define IOCTL_CAN_SET_BAUD          (4)    //设置波特率
```

打开设备

```
hCanHandler = CreateFile(TEXT("CBS1:"), GENERIC_READ | GENERIC_WRITE, 0, NULL, OPEN_EXISTING, 0, NULL);
```

设置波特率为 500K，并获得实际波特率。

```
DWORD dwRealBaud, dwBaud = 500000;
DeviceIoControl(hCanHandler, IOCTL_CAN_SET_BAUD, &dwBaud, sizeof(DWORD), &dwRealBaud, sizeof(DWORD), NULL, NULL);
```

设置接收参数，并开始接收循环数据。

```
CAN_IOCTL_MSG tMsg;
tMsg.nMsgObj = RX_MSG_INDEX(0);         //使用接收窗口 0
tMsg.msg.IdType      =    CAN_STD_ID;       //接收标准帧
tMsg.msg.Id             = 0X7ff;                  //接收 ID
DeviceIoControl(hCanHandler, IOCTL_CAN_SETUP_RX, &tMsg, sizeof(CAN_IOCTL_MSG), NULL, 0, NULL, NULL);
BYTE* pRecvData = (BYTE*)malloc(65536);;
while(TRUE)
{
      tMsg.nMsgObj = RX_MSG_INDEX(0);         //使用接收窗口 0
      tMsg.nWaitMs = 1000;                          //超时设置为 1 秒, 1 秒后没有接收到数据，DeviceIoControl 函数返回 FALSE。
      DWORD dwRecvCount=0;
      bRet=DeviceIoControl(hCanHandler, IOCTL_CAN_RX, &tMsg, sizeof(CAN_IOCTL_MSG), pRecvData, 65536, &dwRecvCount, NULL);
      if(bRet)
      {
            ...
      }
}
free((void*)pRecvData);
```

发送一帧数据

```
CAN_IOCTL_MSG tMsg;
tMsg.nMsgObj = TX_MSG_INDEX(0);
tMsg.nWaitMs = 1000;                          //发送超时设置为 1 秒，1 秒后没有发送成功，DeviceIoControl 函数返回 FALSE。
tMsg.msg.FrameType= DATA_FRAME;         //发送数据帧
tMsg.msg.IdType      = CAN_STD_ID;        //发送标准帧
tMsg.msg.Id             = 0X7FF;               //ID
tMsg.msg.DLC           = 8;                      //数据长度
tMsg.msg.Data[0]    = 0x5A;
tMsg.msg.Data[1]    = 0x5A;
tMsg.msg.Data[2]    = 0x5A;
tMsg.msg.Data[3]    = 0x5A;
tMsg.msg.Data[4]    = 0x5A;
tMsg.msg.Data[5]    = 0x5A;
tMsg.msg.Data[6]    = 0x5A;
tMsg.msg.Data[7]    = 0x5A;
bRet=DeviceIoControl(hCanHandler, IOCTL_CAN_TX, &tMsg, sizeof(CAN_IOCTL_MSG), NULL, 0, NULL, NULL);

if(bRet)
{
      TRACE(TEXT("CAN 发送成功\r\n"));
}
else
{
```

```
    TRACE(TEXT("CAN 发送失败\r\n"));
}
```

更加详细的 CAN 接口使用方法，请参考我司提供的 CANTest 程序源代码。
HMI977 的 CAN 接口和 RS485 采用的是二选一设计，具体参看文档《NUC97X 的串口 CAN 兼容设计》

24. 用 U 盘升级固件,开机画面和 App

固件和开机画面升级：
把 ENKUP.bin, EBOOT.bin，Logo.bmp 等文件拷贝到 U 盘根目录，然后把 U 盘插到 USB 主口上。系统会自动弹出升级界面，点升级按钮，升级完成后重新启动板子即可。

开机画面格式：
开机画面 Logo.bmp 文件为 RGB565 格式，行序倒置。画面大小和 LCD 大小一致。用 PS 等软件可以保存成这种格式。

App 升级：
在电脑自行制作应用程序升级包 UpdateApp.zip (升级包内的所有文件会由升级程序自动拷贝到\NandFlash\App 目录下)。在 zip 文件内的根目录中，创建 autorun.ini，文件内容为需要开机自动执行的 EXE，每个 EXE 一行，EXE 路径写以\NandFlash\App 开头的路径。
把 UpdateApp.zip 拷贝到 U 盘根目录。在板子上插入 U 盘，然后按照屏幕提示升级即可。(升级时无需退出原来的程序) 。升级完成拔掉 U 盘断电重启。系统启动后自动运行 autorun.ini 指定的程序，不再进入桌面。

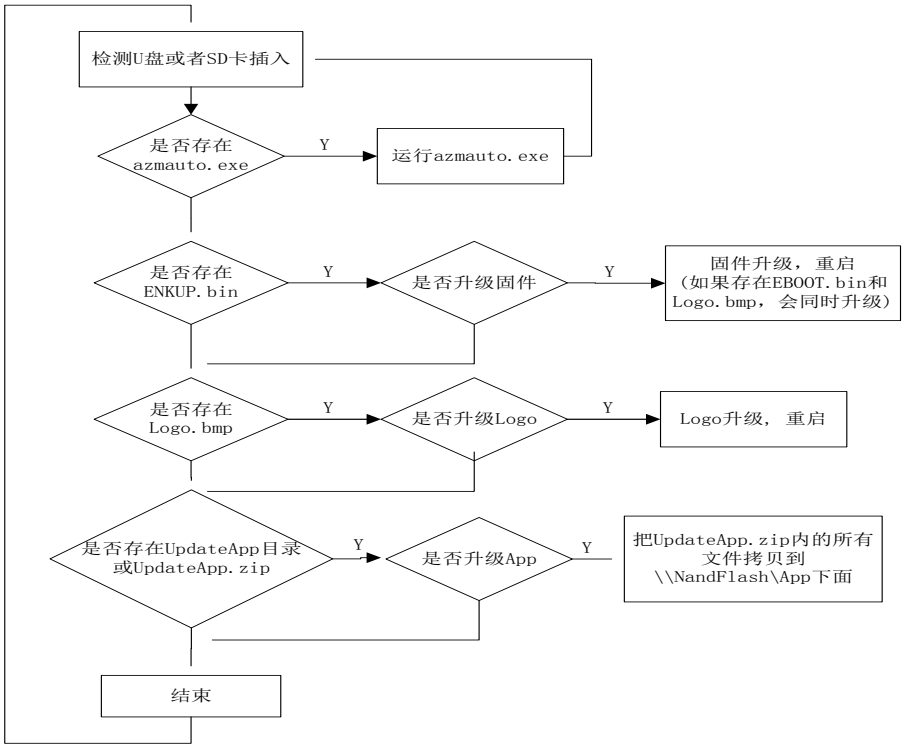
如对制作升级包有疑问，可参考我司提供的示例 UpdateApp.zip
以上升级方式同样适用于 SD 卡。可以关机状态先插入有升级文件的 SD 卡，然后再开机。

默认固件带有此项功能，如果实际产品不需要这项功能，使用如下代码屏蔽 Launch130 即可。

```
HKEY hReg;
if(RegOpenKeyEx(HKEY_LOCAL_MACHINE, TEXT("init"), 0, 0, &hReg)==ERROR_SUCCESS)
{
    RegDeleteValue(hReg, TEXT("Launch130"));
    RegCloseKey(hReg);
}
```

25. U 盘自动升级和运行程序的逻辑

系统对 U 盘插入后处理逻辑如下：



26. 唯一序列号

HMI977 提供了一个唯一序列号供应用程序使用。该序列号可以通过获取网卡 MAC 地址获得。

```
DWORD dwLen;
  IP_ADAPTER_INFO mInfo={0};
  TCHAR szShowID[256]={0};

  dwLen = sizeof(IP_ADAPTER_INFO);
  GetAdaptersInfo(&mInfo, &dwLen);
  swprintf_s(szShowID, sizeof(szShowID)/sizeof(TCHAR),
        TEXT("Mac Address & Board Unique ID : %02X-%02X-%02X-%02X-%02X-%02X"),
        mInfo.Address[0], mInfo.Address[1], mInfo.Address[2],
        mInfo.Address[3], mInfo.Address[4], mInfo.Address[5]);
  MessageBox(NULL, szShowID, TEXT("ID"), MB_OK|MB_ICONINFORMATION);
```

如果是不带网卡的定制板，也可以用下面这个接口获取该序列号：

```
  DWORD dwLen;
  HANDLE    hFile;
  BYTE mMacAddr[8]={0};
  TCHAR szShowID[256]={0};

  hFile = CreateFile(TEXT("KIP1:"), GENERIC_READ | GENERIC_WRITE, 0, NULL, OPEN_EXISTING, 0, NULL);
  if(hFile != INVALID_HANDLE_VALUE)
  {
        DeviceIoControl(hFile, 102, NULL, 0, mMacAddr, sizeof(mMacAddr), &dwLen, NULL);
        CloseHandle(hFile);
  }

  swprintf_s(szShowID, sizeof(szShowID)/sizeof(TCHAR),
        TEXT("Mac Address & Board Unique ID : %02X-%02X-%02X-%02X-%02X-%02X"),
        mMacAddr[0], mMacAddr[1], mMacAddr[2],
        mMacAddr[3], mMacAddr[4], mMacAddr[5]);
  MessageBox(NULL, szShowID, TEXT("ID"), MB_OK|MB_ICONINFORMATION);
```

27. 开机进度条

开机过程的进度条，默认在启动应用程序的时候就跳到 100%并不再刷新。如果有的应用程序启动比较慢，可以通过注册表，让进度条多走一会儿。

注册表选项如下，默认值为 0。如果需要多走 5 秒，则写入 5 即可。

```
[HKEY_LOCAL_MACHINE\ARMDEVICE]
"DelayOffProgress"=dword:0
```

开机进度条的位置和颜色均可以通过修改注册表来控制。

```
[HKEY_LOCAL_MACHINE\ARMDEVICE\LoadingProgress]
    "topx"=dword:0              ;左上角 X 坐标
    "topy"=dword:0              ;左上角 Y 坐标
    "bottomx"=dword:0           ;右下角 X 坐标
    "bottomy"=dword:0           ;右下角 Y 坐标
    "frameline"=dword:1         ;边框的像素宽度
    "framecolor"=dword:ff0000   ;边框的颜色 RGB 值，红色为 0xFF0000，以此类推。
    "color"=dword:A8DCE0        ;进度条的颜色 RGB 值，红色为 0xFF0000，以此类推。
```

28. 使用 Visual Studio 2005 开发 C++应用

使用 Visual Studio 2005 开发 WinCE 的 C++程序，需要安装以下内容：

先安装 Visual Studio 2005，选择 C++相关组件

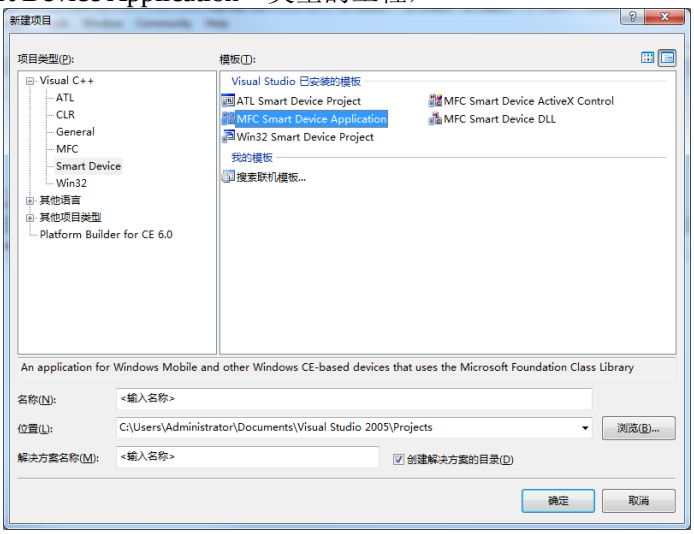
再安装 Visual Studio 2005 Service Pack 1

如果是 Win7 或 Win10 系统，还需要安装 Visual Studio 2005 Service Pack 1 Update for Windows Vista

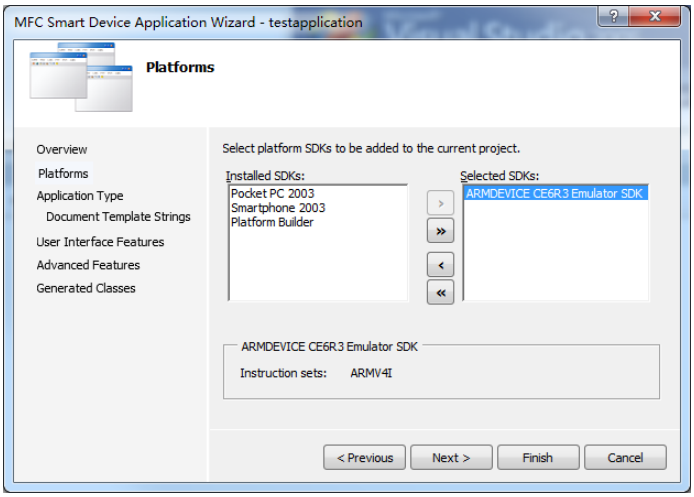
最后安装我司提供的 SDK 包 ARMDEVICE_WCE6R3SDK.msi

按照如下步骤，我们新建一个 C++工程，然后进行联机调试。（以下示例使用英文版 Visual Studio 2005）

首先新建一个“MFC Smart Device Application”类型的工程，



在选择平台的界面中，取消默认的“PocketPC 2003”，使用“ARMDEVICE CER3 Emulator SDK”，如下图：



成功创建工程之后，请参照《安之谋科技 CE6 开发板基于以太网的远程调试》来进行远程调试。

29. 使用 Visual Studio 2005 开发 C#应用

使用 Visual Studio 2005 开发 WinCE 的 C#程序，需要安装以下内容：

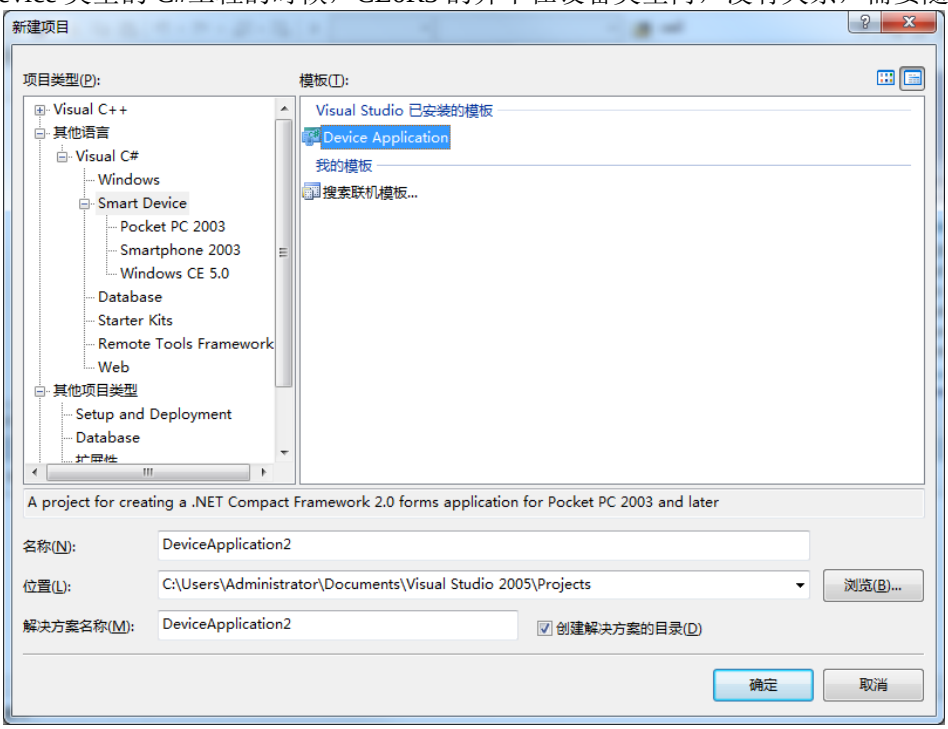
先安装 Visual Studio 2005，选择 C#相关组件

再安装 Visual Studio 2005 Service Pack 1

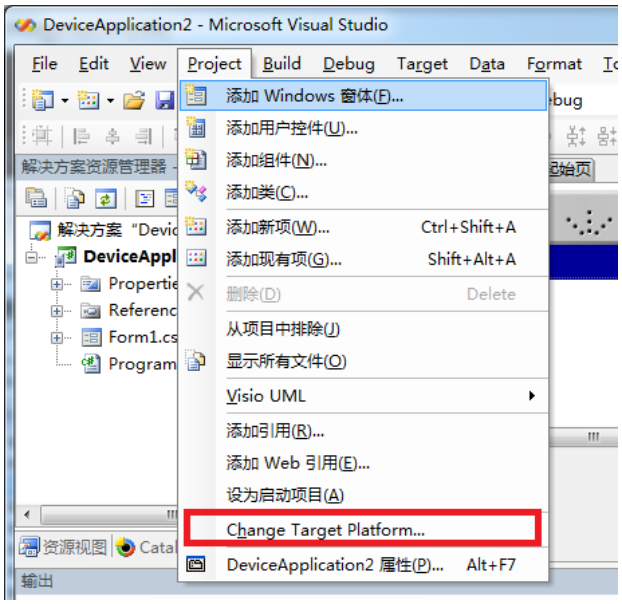
如果是 Win7 或 Win10 系统，还需要安装 Visual Studio 2005 Service Pack 1 Update for Windows Vista

最后安装我司提供的 SDK 包 ARMDEVICE_WCE6R3SDK.msi

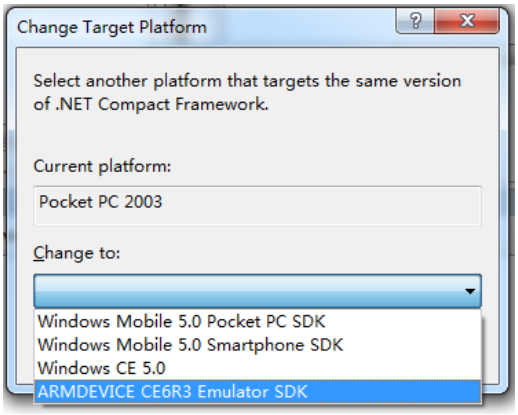
按照如下步骤，我们新建一个 C#工程，然后进行联机调试。（以下示例使用英文版 Visual Studio 2005）
在创建 Smart Device 类型的 C#工程的时候，CE6R3 的并不在设备类型内，没有关系，需要随便选一个就行。



成功创建工程之后，打开 Project 菜单，然后选择“Change Target Platform”，如下图：



在随后的对话框中，将目标平台设置为“ARMDEVICE CE6R3 Emulator SDK”即可。



最后请参照《安之谋科技 CE6 开发板基于以太网的远程调试》来进行远程调试。

30. 使用 Visual Studio 2008 开发 C++应用

使用 Visual Studio 2008 开发 WinCE 的 C++程序，需要安装以下内容：

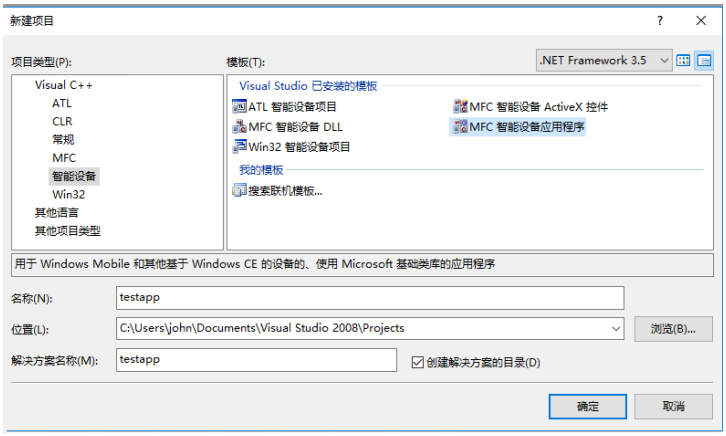
先安装 Visual Studio 2008，选择 C++相关组件

再安装 Visual Studio 2008 Service Pack 1

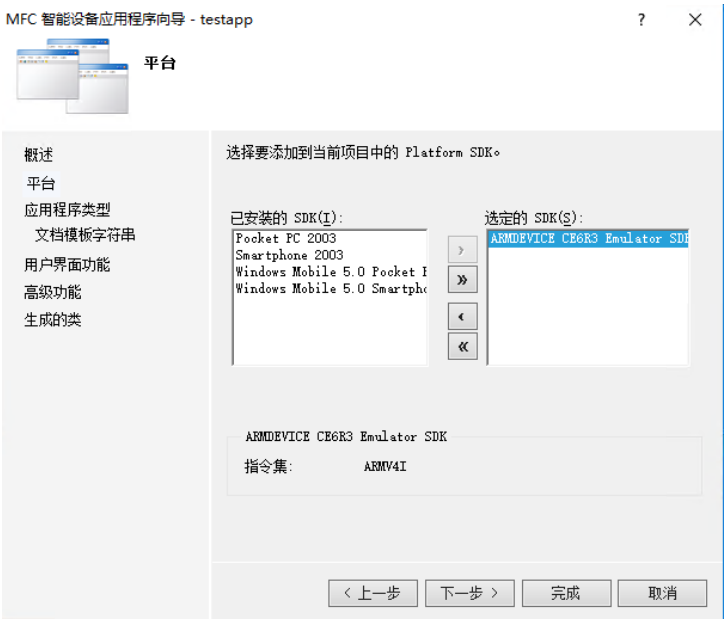
最后安装我司提供的 SDK 包 ARMDEVICE_WCE6R3SDK.msi

按照如下步骤，我们新建一个 C++工程，然后进行联机调试。（以下示例使用中文版 Visual Studio 2008）

首先新建一个“MFC 智能设备应用程序”类型的工程，



在选择平台的界面中，取消默认的“PocketPC 2003”，使用“ARMDEVICE CER3 Emulator SDK”，如下图：



成功创建工程之后，请参照《安之谋科技 CE6 开发板基于以太网的远程调试》来进行远程调试。

31. 使用 Visual Studio 2008 开发 C#应用

使用 Visual Studio 2008 开发 WinCE 的 C#程序，需要安装以下内容：

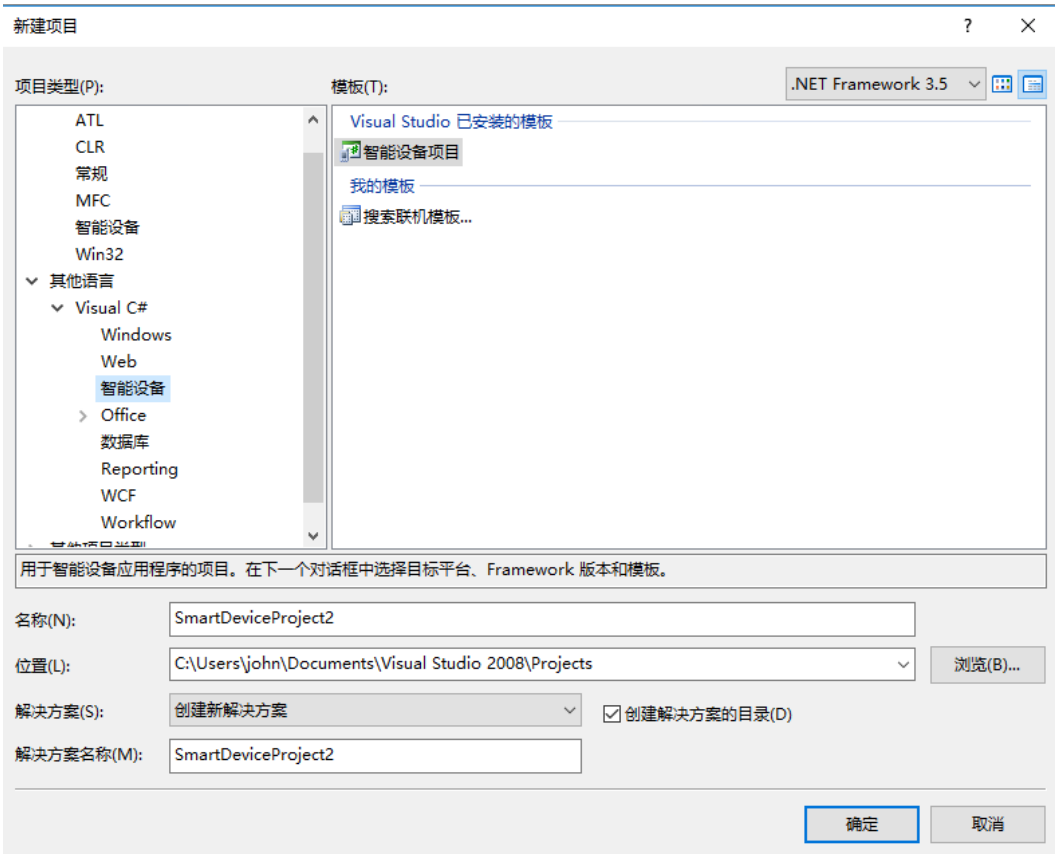
先安装 Visual Studio 2008，选择 C#相关组件

再安装 Visual Studio 2008 Service Pack 1

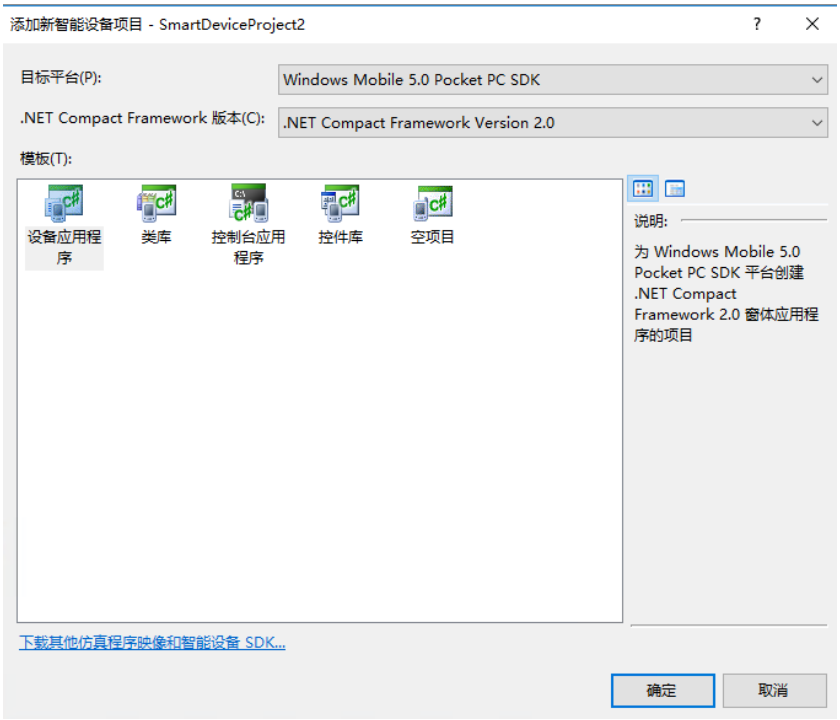
最后安装我司提供的 SDK 包 ARMDEVICE_WCE6R3SDK.msi

按照如下步骤，我们新建一个 C#工程，然后进行联机调试。（以下示例使用中文版 Visual Studio 2008）

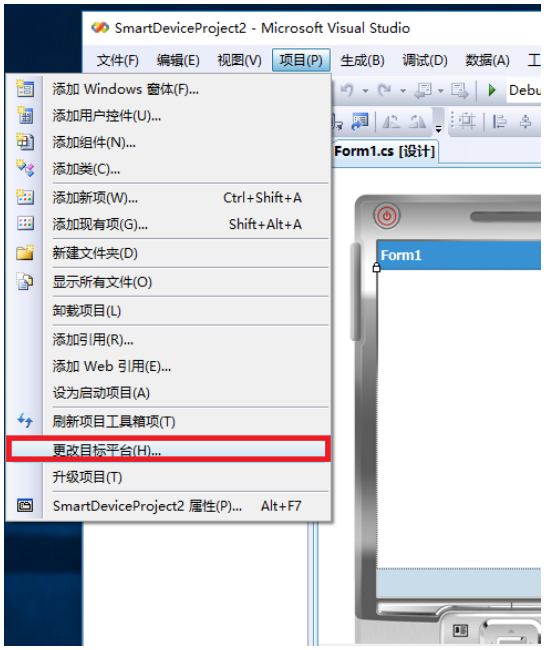
首先创建“智能设备项目”。



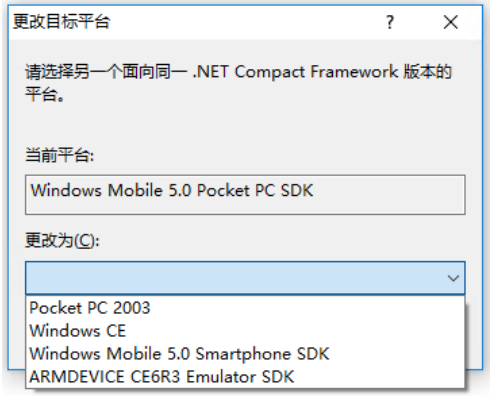
在下一步设置目标平台的向导界面中，CE6R3 的并不在内，没有关系，需要随便选一个就行。.Net 版本建议选 2.0，因为出厂固件默认带 2.0 版本的.Net 库。



成功创建工程之后，打开项目菜单，然后选择“更改目标平台”，如下图：



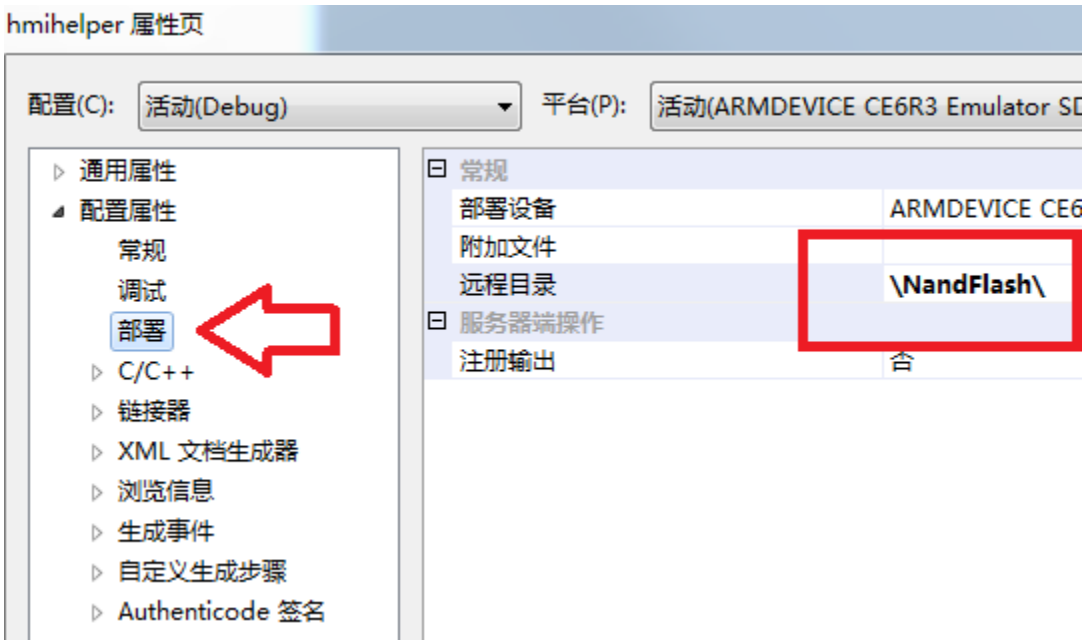
在随后的对话框中，将目标平台设置为“ARMDEVICE CE6R3 Emulator SDK”即可。



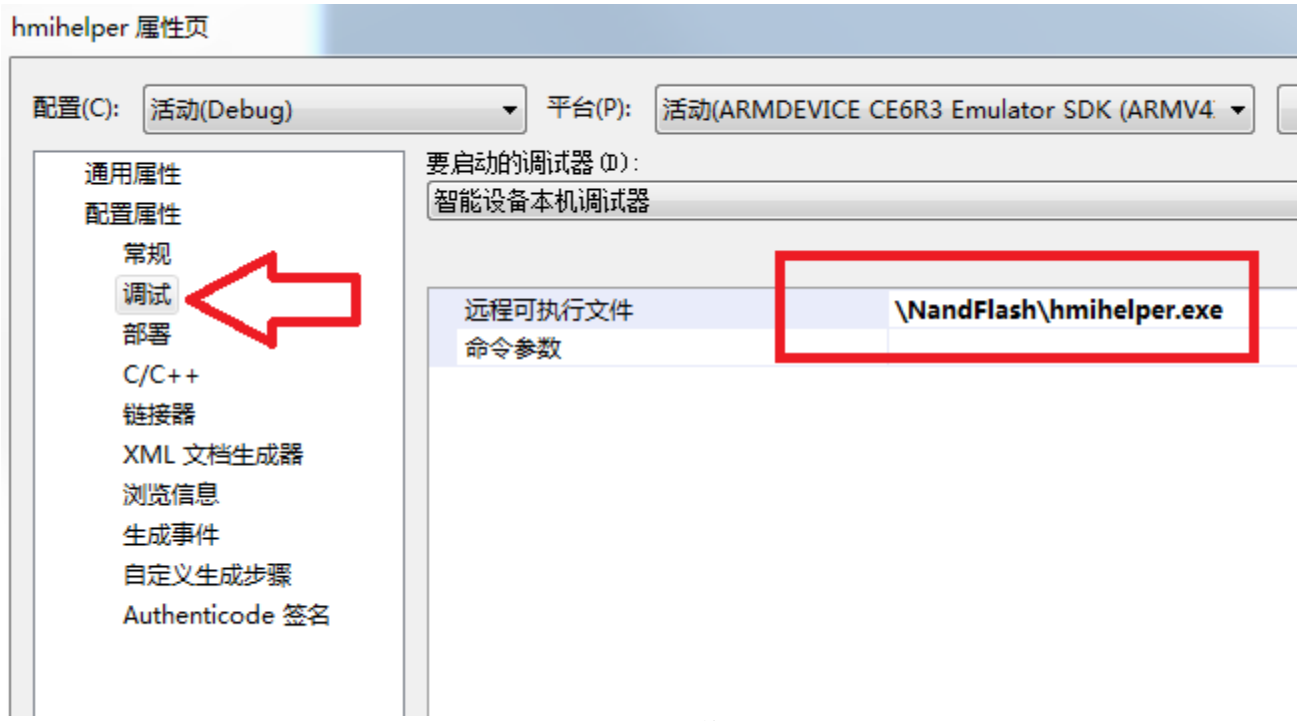
最后请参照《安之谋科技 CE6 开发板基于以太网的远程调试》来进行远程调试。

32. 设置 VisualStudio 工程的部署目录和调试目录

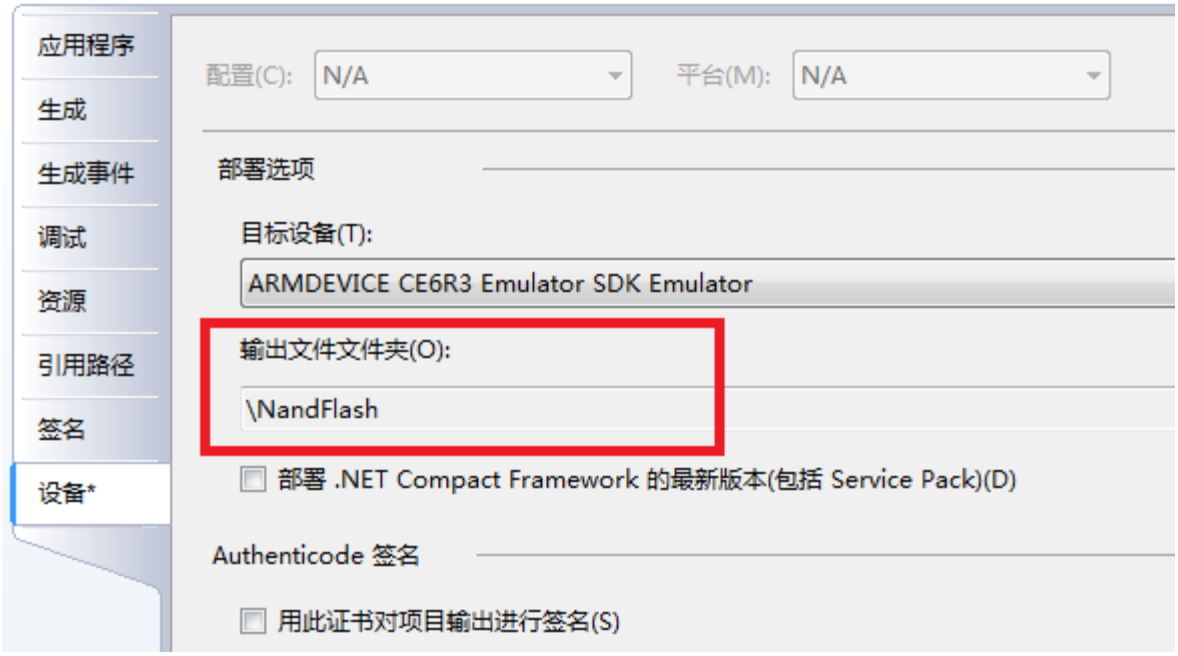
默认固件采用的 ROMRAM 类型文件系统，只有\NandFlash 目录支持存储，因此调试的时候，需要把工程的部署目录和调试目录都修改成\NandFlash 目录下。



VS2005/VS2008 C++ 修改部署目录



VS2005/VS2008 C++ 修改调试目录



VS2005/VS2008 C# 修改输出目录

C#调试的时候，不要选择“部署.NET Compact Framwork 的最新版本”，而应该直接在板子上下载对应.NET 版本的固件。

33. C#库和参考例程

我司提供的一个含源码的配套库供 C#程序调用，可以实现 GPIO 的读写，板子的复位等操作。
详情参看文档《HMI97X 的 DotNet 开发例程和配套库》和相关代码。