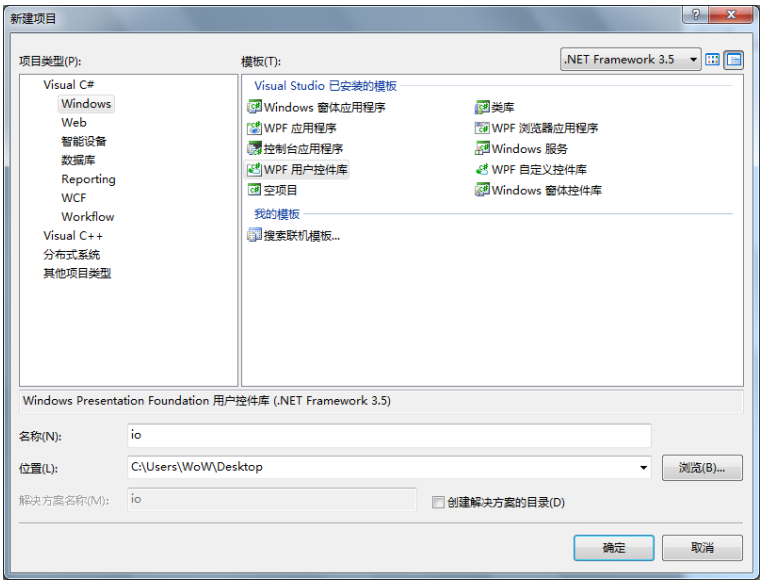
iConfig 插件编写

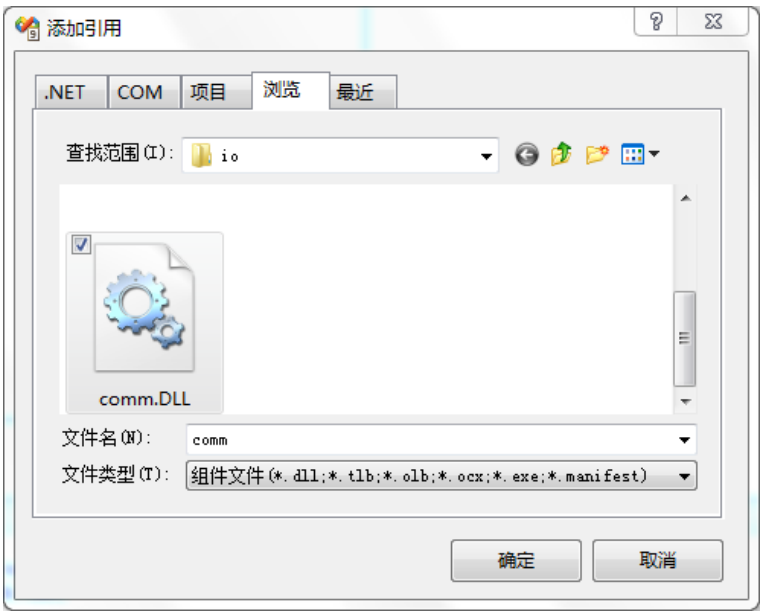
以一个简单的端口初始化插件为例

- 在 VisualStudio2008 中新建一个 WPF 用户控件（所有的插件都是 WPF 用户控件）

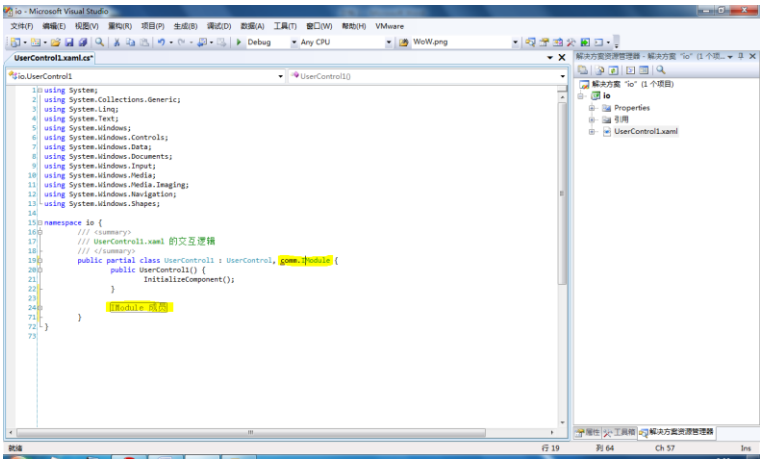


- 然后添加对“comm.dll”的引用

comm.dll 包含了插件所必须实现的接口定义。



- 在插件的代码中继承该接口。



为了简单起见，这个简单的插件只对端口 A 进行初始化，也就是只负责 PORTA 和 DDRA，为此，增加 PORTA 和 DDRA 两个相关属性。

```
ice io {
 /// <summary>
 /// UserControl1.xaml 的交互逻辑
 /// </summary>
 public partial class UserControl1 : UserControl, comm.IModule {

        public byte PORTA { get; set; }
        public byte DDRA { get; set; }

        public UserControl1() {
                InitializeComponent();
        }

        IModule 成员
 }
```

最后的代码：

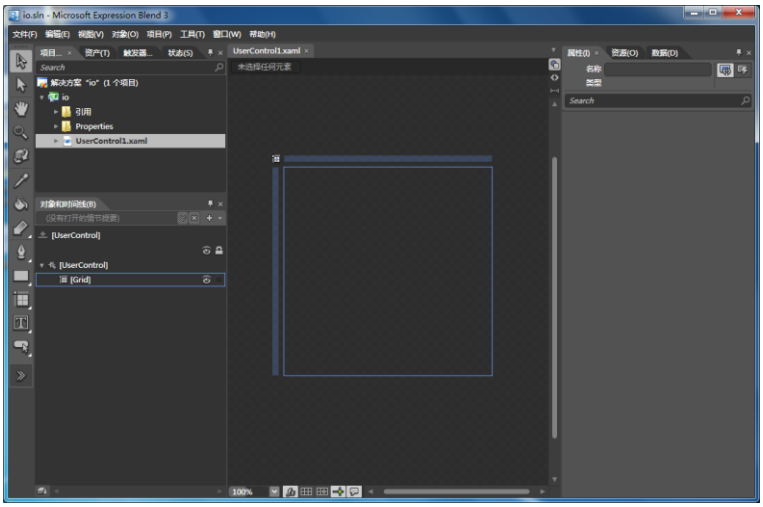
```
public partial class UserControl1 : UserControl, comm.IModule {

        public byte PORTA { get; set; }
        public byte DDRA { get; set; }

        public UserControl1() {
            InitializeComponent();
        }

        #region IModule 成员

        /// <summary>
        /// 返回给T4模板引擎的参数，是一个字典。
        /// </summary>
        public Dictionary<string, object> Data {
            get {
                // 创建一个新的字典。
                Dictionary<string, object> data = new Dictionary<string, object>();
                // 将需要传递给T4模板引擎的数据加载到字典中。
                data.Add("DDRA", DDRA);
                data.Add("PORTA", PORTA);
                return data;
            }
        }
```

```
/// <summary>
/// 是否使用该插件，现在没有使用到这个属性，可以暂时不用关心。
/// </summary>
public bool Enable {
    get {
        throw new NotImplementedException();
    }
    set {
        throw new NotImplementedException();
    }
}

/// <summary>
/// 在创建C语言文件时，由主程序提供的一个环境变量，指系统的工作频率。
/// </summary>
public int F_CPU {
    get;
    set;
}

/// <summary>
/// 在main.c中调用的初始化函数名称。
/// </summary>
public string Function_Initial {
    get { return "io_initial()"; }
}

/// <summary>
/// 在main.c中应该包含的头文件。
/// </summary>
public string Include {
    get { return "io.h"; }
}

/// <summary>
/// 如果插件需要在创建C代码文件之前获取相关环境变量，这引发该事件。
/// </summary>
public event comm.DGetEnvironment OnGetEnvironment;

/// <summary>
/// 插件所用到的T4模板文件列表
/// </summary>
public Dictionary<string, string> T4File {
    get {
        Dictionary<string, string> file_list = new Dictionary<string, string>();
        // io.tt 模板文件生成 io.c
        file_list.Add("io.tt", "io.c");
        // io_include.tt 模板文件生成 io.h
        file_list.Add("io_include.tt", "io.h");
        return file_list;
    }
}
```

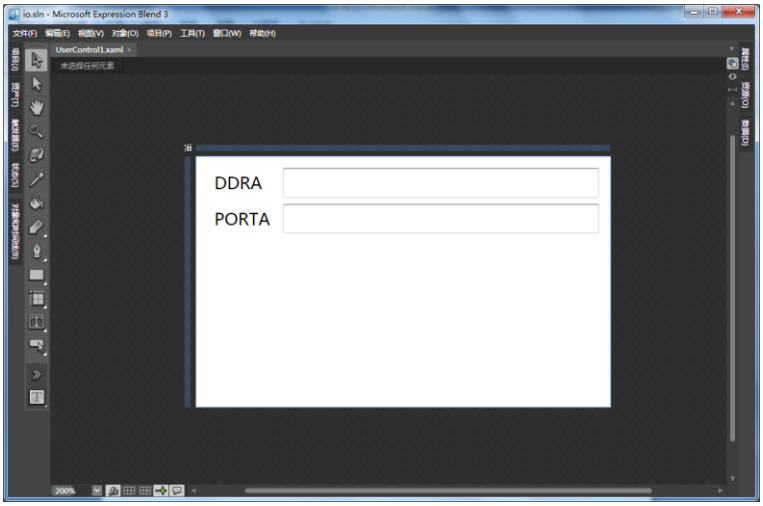
```
        /// <summary>
        /// 插件的标题，显示在UI界面上。
        /// </summary>
        public string Title {
            get { return "GPIO - PORTA"; }
        }

        /// <summary>
        /// 在创建C代码是由主程序调用，辅助计算、更新相关参数，
        /// 例如根据F_CPU和设定的波特率计算相关寄存器的值。
        /// </summary>
        public void Update() {
            return;
        }

        #endregion
    }
```

- 现在，所有的逻辑代码编辑完成，开始绘制界面。

使用 Blend 打开项目文件。



- 绘制一个简单的用户界面并执行相关的数据绑定。



- 这里是绘制好的用户界面 XAML 代码

```
<UserControl
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    x:Name="userControl" x:Class="io.UserControl1" Background="White"
    mc:Ignorable="d" d:DesignWidth="298.944" d:DesignHeight="180.673">
    <Grid>

        <Label HorizontalAlignment="Left" Margin="8,31.836,0,0"
               VerticalAlignment="Top" Content="PORTA"/>
        <TextBox Margin="62.28,33.836,8,0" VerticalAlignment="Top"
             Text="{Binding PORTA, ElementName=userControl, Mode=Default}"
             TextWrapping="Wrap"/>
        <Label HorizontalAlignment="Left" Margin="8,6,0,0" VerticalAlignment="Top"
               Content="DDRA"/>
        <TextBox Margin="62.28,8,8,0" VerticalAlignment="Top"
             Text="{Binding DDRA, ElementName=userControl, Mode=Default}"
             TextWrapping="Wrap"/>

    </Grid>
</UserControl>
```

- 然后编写模板文件

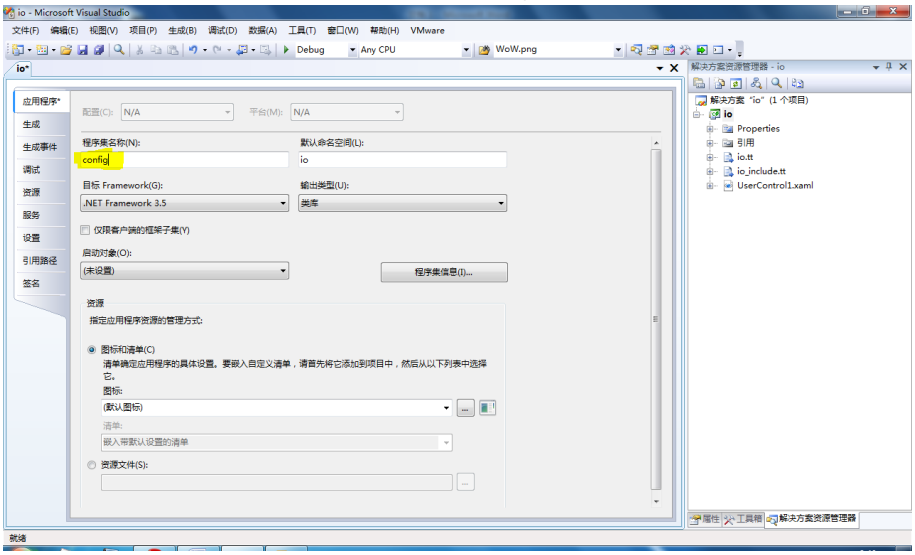
io.tt

```
<#@ template language="C#" HostSpecific="True" #>
<# Dictionary<string, object> Data = (Host as Host).Data; #>
#include <avr\io.h>
#include "io.h"

void io_initial(void) {
<# foreach(KeyValuePair<string, object> i in Data) { #>
    <#= i.Key #> = <#= i.Value #>;
<# } #>
}
```

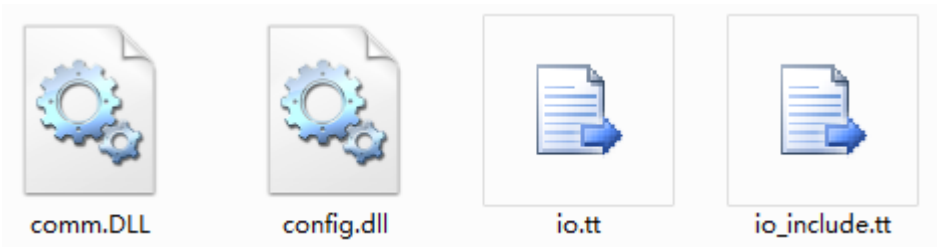
io_include.tt

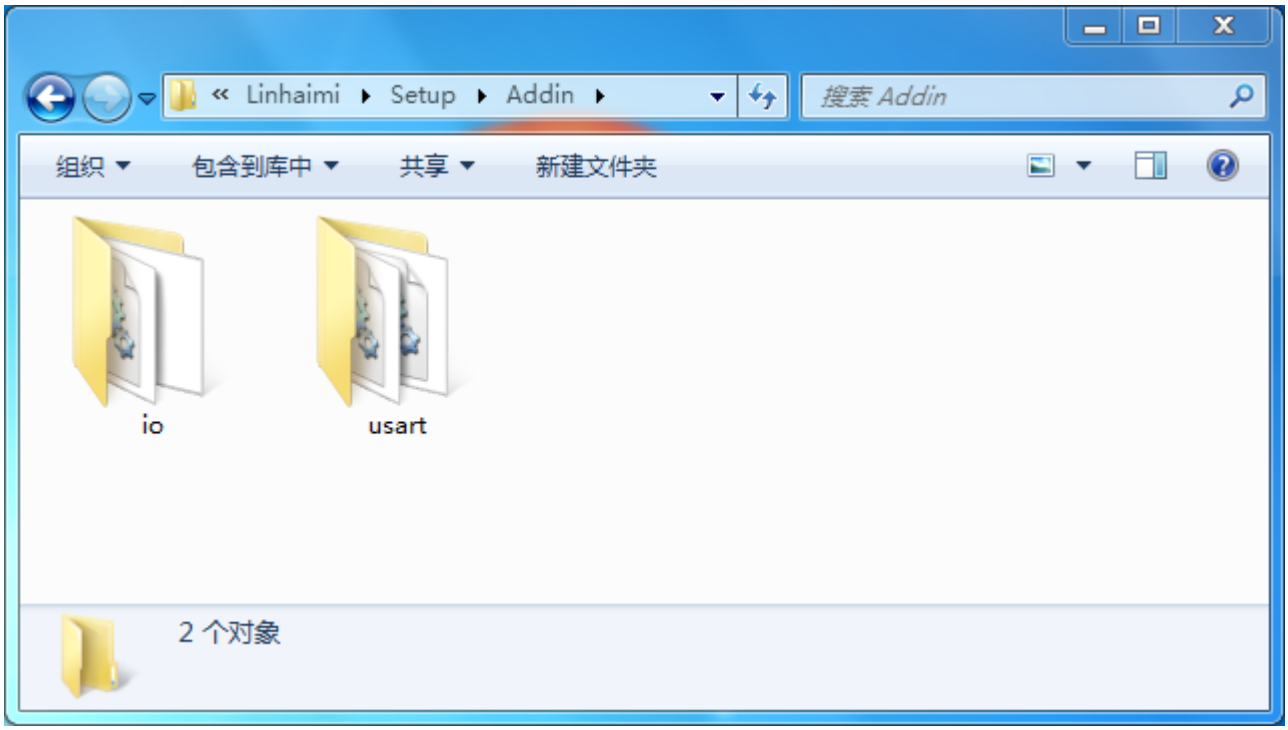
```
#ifndef _IO_h_
#define _IO_h_

#include <avr\io.h>

void io_initial(void);

#endif
```

- 在项目属性设定当中，将生成哦 Dll 文件名称改为 Config.dll

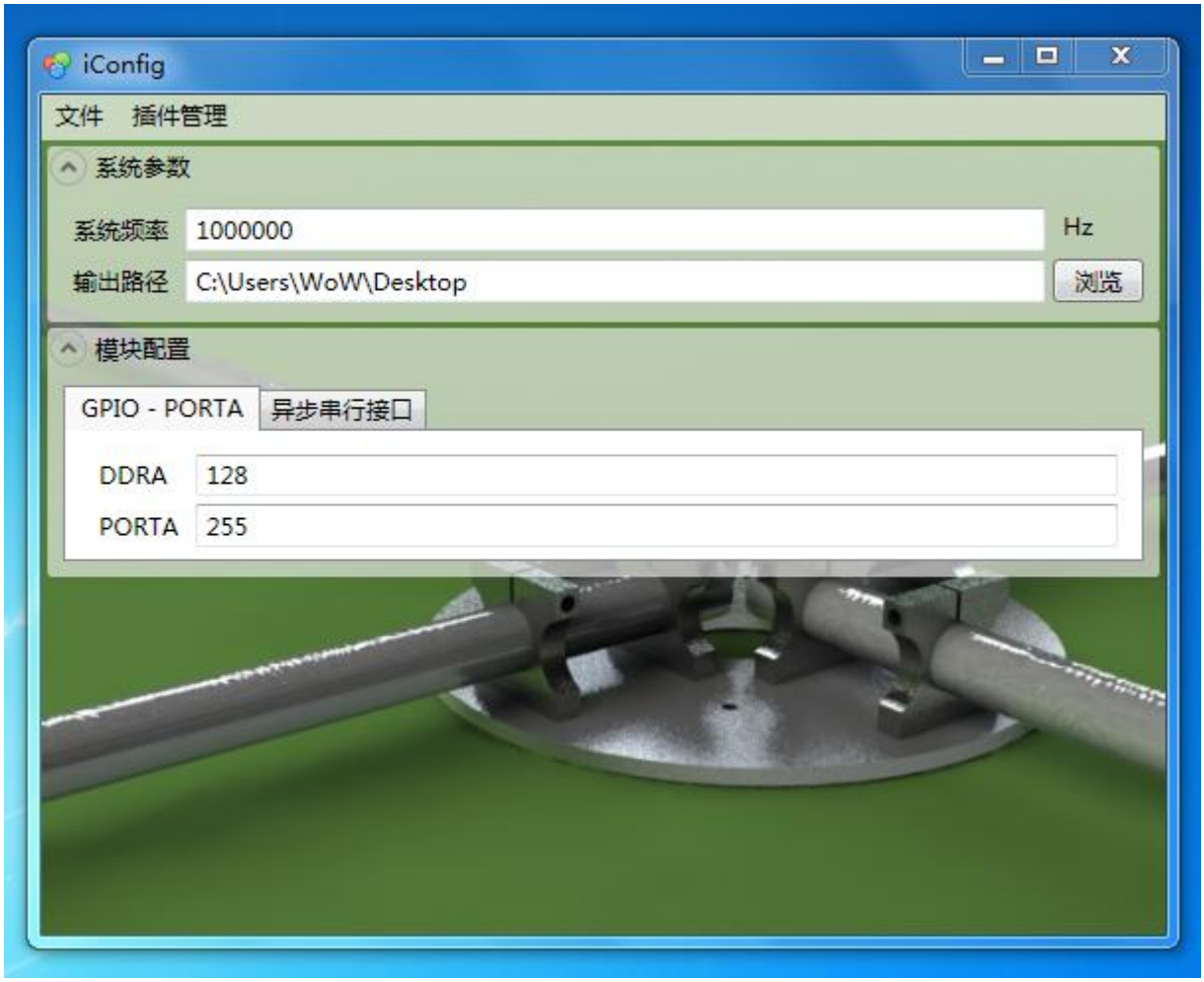


- 安装插件，到 AddIn 中新建一个 io 子目录当，将最后将得到的 4 个文件拷贝进去。





- 插件可以正常工作了。


iConfig
文件
插件管理
系统参数
系统频率
1000000
Hz
输出路径
C:\Users\WoW\Desktop
浏览
模块配置
GPIO - PORTA
异步串行接口
DDRA
128
PORTA
255