NFC 代码初窥

1 运作流程初步

代码分为两个版本，c 和 c++，两个版本的功能其实一样，只是实现的方法略有不同
在工程\Bfl\ex\ExampleProject.sln 可以看到整个的运作流程
main 入口存在于 ExampleProject.c
因为示例程序是运行于 pc 命令行环境中，主函数中有一些是进行参数控制的，这部分我们可以先略过
关注点是参数 o ，这个参数标名了 nfc 的运行模式
其中
1 ... Mifare Reader
2 ... FeliCa Reader (just Polling)
3 ... NFC Initiator
4 ... NFC Target ----------卡模式
5 ... ISO14443-4 Reader
6 ... Power Down Example

Line:251

```
case 4 :
        /* Activate SDD of PN51x and automatic send receive */
      printf( "[I] Acting as NFC Target.\n" );
      res = ActivateNfcTarget(comHandle, settings);
      break;
```

2 对于 Bus abstract module 和 Register control module 的理解

虽然这个工程是用 c 写的，但是其中大量用到了面向对象的思想，在下面的分析中可以体现。
以下部分简称 Bus abstract module 为 bal，Register control module 为 RegCtl
bal 的任务就是分 Windows 和 Linux 两种环境进行串口操作（最底层的串口操作）。这种区分是通过 ifdef win32 来实现的

RegCtl 实际上就是实现了一个接口，在高层的调用中都是直接调用这些接口函数，避免了通讯方法的限制。我们从其最精华的 phcsBflRegCtl_t 结构体开始看起：

```
typedef struct
{
    /* Methods: */
    pphcsBflRegCtl_SetReg_t     SetRegister;     /* SetRegister member function. */
    pphcsBflRegCtl_GetReg_t    GetRegister;     /* GetRegister member function. */
    pphcsBflRegCtl_ModReg_t    ModifyRegister;         /* ModifyRegister member function. */
    pphcsBflRegCtl_SetMultiReg_t    SetRegisterMultiple;        /*SetRegisterMultiple member function. */
    pphcsBflRegCtl_GetMultiReg_t   GetRegisterMultiple;        /*GetRegisterMultiple member function. */

    void *mp_Members;     /* Internal variables of the C-interface. Usually a structure is behind
                             this pointer. The type of the structure depends on the
                             implementation
                            requirements. */
    #ifdef PHFL_BFL_CPP
    void    *mp_CallingObject;     /* Used by the "Glue-Class" to reference the wrapping
                                      C++ object, calling into the C-interface. */
    #endif

    /* Lower edge: */
    phcsBflBal_t     *mp_Lower;
} phcsBflRegCtl_t;
```

上面连续五个变量其实都是函数指针，而这些函数指针会在各种通讯方式(iic,spi,serial,parallel)的 initialise 方法中被初始化
举例来说在 phcsBflRegCtl_SerHw1Init 这个串口模块的初始化函数我们可以看到一些端倪

```
void phcsBflRegCtl_SerHw1Init(phcsBflRegCtl_t         *cif,
                              void    *p_params,
```

```
                                        phcsBflBal_t        *p_lower)
{
    /* Glue together and init the operation parameters: */
    cif->mp_Members         = p_params;
    cif->mp_Lower           = p_lower;
    ((phcsBflRegCtl_SerHw1Params_t*)cif->mp_Members)->dummy = 0;
    /* Initialize the function pointers: */
    cif->GetRegister                =  phcsBflRegCtl_SerHw1GetReg;     //这些都是串口模块的
函数名
    cif->SetRegister                = phcsBflRegCtl_SerHw1SetReg;
    cif->ModifyRegister             = phcsBflRegCtl_SerHw1ModReg;
    cif->GetRegisterMultiple        = phcsBflRegCtl_SerHw1GetMultiReg;
    cif->SetRegisterMultiple        = phcsBflRegCtl_SerHw1SetMultiReg;
}
```

那么串口模块和 bal 有什么区别呢？实际上，串口模块里的函数到最后都会调用 bar 里的函数以达到与操作系统无关化，这貌似是串口的特例，其他的通讯方式没有这么纠结。这是由本程序的特殊性决定的(示例程序是在 PC 上运行的)。

这些初始化实际上会被执行两次，一次在 ExampleProject.c，一次在 NFCTarget.c 中
这是由 ActivateNfcTarget(comHandle, settings)函数的传参策略决定的。在 main 函数的一开始，实际上已经定义了：

```
phcsBflBal_t        bal;
phcsBflRegCtl_t rc_reg_ctl;
```

main 在对这些东西初始化了之后，会根据 bal 得到其所创立的串口的句柄 comHandle 传进去，而不是把 bal 和 rc_reg_ctl 传入 ActivateNfcTarget
所以在 ActivateNfcTarget 中又会反过来根据 comHandle 其创立 bal。

3 目前的现状

由于该库组织良好，我们只要对在初始化 rc_reg_ctl 时用上 spi 的相应初始化函数就行了，这减轻了我们很大的工作量。那么目前的主要的就是去理解这个 ActivateNfcTarget 函数了，研究一下怎么把这个函数改成适用于 spi 接口的.

2011/1/12　手机支付组