笔记本 BIOS 开发过程中读取 GPIO 的方法及实现
——以 SKL 平台为例

张月蓉
（安徽开放大学安徽继续教育网络园区管理中心，安徽 合肥 230022）

摘要：笔记本 BIOS 开发属于计算机底层开发，对开发者来说开发周期越来越紧凑，需要调试功能非常多，而且技术难度较大，需要经常查阅各类行业标准文档。GPIO 在笔记本 BIOS 开发过程中经常需要调试功能，目前使用的读取 GPIO 工具功能不够完善，无法满足实际开发需要。本文基于 Visual Studio 开发环境，以英特尔 SKL 平台为例，详细介绍利用 MFC 开发读取 GPIO 的工具。

关键词：GPIO；BIOS 开发；MFC

BIOS 是 Basic Inport/Output System 的缩写，即基本输入 / 输出系统。实际上 BIOS 是被固化在计算机 ROM（只读存储器）芯片上的一组程序。现实中很多人把它当成了一块芯片或是错认成 CMOS，它保存着计算机最重要的基本输入输出的程序、系统设置信息、开机上电自检程序和系统启动自举程序，为计算机提供最底层的、最直接的硬件控制与支持。更形象地说，BIOS 就是硬件与软件程序之间的一个“桥梁”或者说是接口（虽然它本身也只是一个程序），负责解决硬件的即时需求，并按软件对硬件的操作要求具体执行，负责在电脑开启时检测，初始化系统设备，装入操作系统并调度操作系统与 EC/ 硬件之间的沟通。

当今市场上有多种笔记本电脑品牌，如联想、DELL、惠普、小米等。每家品牌的电脑都是经过完整的笔记本开发周期来完成的，周期有长有短。在笔记本开发周期中涉及硬件、软件 BIOS、Driver、OS 以及机构等多方面。市场日新月异，竞争激烈，各个电脑品牌不断压缩开发周期，导致开发人员时间紧迫，需要协助调试工具。对笔记本开发过程中 BIOS 开发者而言在开发过程中经常需要读取寄存器的状态，如电池状态、是否有显卡等，这些信息都需要读取 GPIO 端口。我们知道 CPU 是由多种寄存器组成的，GPIO 端口实际是一组寄存器，CPU 配置文档明确规定了每个寄存器的地址，在短时间的开发周期内开发者需要频繁读取寄存器着实影响工作效率，而且对其他非开发人员来说读寄存器也是个盲盒。因此，为笔记本 BIOS 开发者提供一个能够读取 GPIO 端口的工具显得尤为重要，不仅方便开发者获得信息提高工作效率，也为其他非开发人员提供一种读取 GPIO 的手段。

一、设计

（一）GPIO

GPIO（General Purpose I/O Ports）可翻译为通用输入 / 输出端口，通俗地说，就是元器件上信号检测引脚，可以通过这些引脚输出高低电平或者通过它们读入引脚的状态是高电平或是低电平。GPIO 是个比较重要的概念，用户可以通过 GPIO 口和硬件进行数据交互（如 UART）、控制硬件工作（如 LED、蜂鸣器等）、读取硬件的工作状态信号（如中断信号）等。GPIO 口的使用非常广泛，掌握了 GPIO，差不多相当于掌握了操作硬件的能力。

GPIO 功能类似 8051 的 P0–P3。是利用工业标准 I2C、SMBus 或 SPI 接口简化了 I/O 口的扩展。可以通过程序控制，用户自由地使用插针。Pin 可根据实际情况用作一般输入（GPI）或一般输出（GPO）或一般输入输出（GPIO），如 CLK 发生器、芯片选择等。对于输入，Pin 电位可通过读取寄存器来确定；对于输出，Pin Pin 可输出高电位或写入寄存器来降低电位；对于其他特殊功能，还有其他寄存器来控制它们。下图 1 是一般输入接口（GPI）电路图，图 2 是一般输出接口（GPO）电路图。

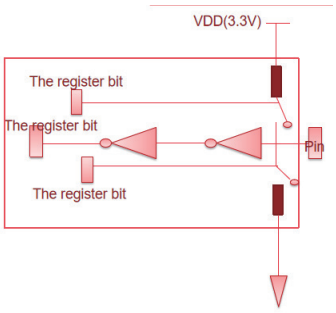

图 1　GPI 电路图

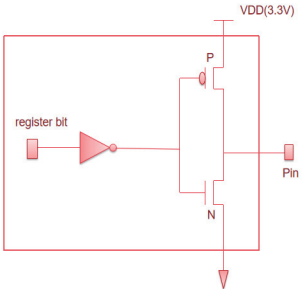

图 2　GPO 电路图

（二）SKL 平台

SKL 即为 Skylake 简称，Skylake 是英特尔第六代微处理器架构，采用的是 14 纳米制程，是 Intel Haswell 微架构及 Intel Broadwell 微架构的继任者。

根据标准文档 PCHGPIO 信号被分成多个组（如 GPP_A、GPP_B 等），每个引脚组都有一个专用的电源引脚，可以设置为 1.8V 或 3.3V。所有的 GPIO 引脚都能被配置为输入或者输出引脚。很多 GPIO 引脚是与其他功能复用的。在 SKL 平台上 GPIO 被分为如下几组，如下表 1 所示。A 至 G8 组可为低电平或者高电平，最后 2 组只能为高电平。

表 1　SKL 平台 GPIO 分组表

GPIO Group	Power Pins	Voltage
Primary Well Group A (GPP_A)	VCCPGPPA	1.8V or 3.3V
Primary Well Group B (GPP_B) Primary Well Group C (GPP_C) Primary Well Group H (GPP_H)	VCCPGPPBCH	1.8V or 3.3V
Primary Well Group D (GPP_D)	VCCPGPPD	1.8V or 3.3V
Primary Well Group E (GPP_E) Primary Well Group F (GPP_F)	VCCPGPPEF	1.8V or 3.3V
Primary Well Group G (GPP_G)	VCCPGPPG	1.8V or 3.3V
Primary Well Group I (GPP_I)	VCCPRIM_3P3	3.3V
Deep Sleep Well Group (GPD)	VCCDSW_3P3	3.3V

二、实现

（一）开发环境

笔记本开发过程中，系统启动到操作系统后，需要在操作系统里面动态实时获取某些 GPIO 接口的信号，是高电平还是低电平。因为笔记本 GPIO 读取工具主要面向软硬件开发人员，界面不复杂，利用 MFC 框架即可快速开发图形界面的 windows 程序。MFC 即 MicrosoftFoundation Classes，是微软公司提供的一个类库（class libraries），以 C++ 类的形式封装了 Windows API，并且包含一个应用程序框架，以减少应用程序开发人员的工作量。其中包含大量 Windows 句柄封装类和很多 Windows 的内建控件和组件的封装类。使用 MFC 开发灵活性较大，极大地方便软件开发。

Visual Studio 是微软公司推出的开发工具套件系列产品，简称 VS，是目前比较流行的 Windows 平台应用程序开发环境，其包括

了整个软件生命周期中所需要的大部分工具，如 UML 工具、代码管控工具、集成开发环境等。VS 可以用来创建 Windows 平台下的 Windows 应用程序和网络应用程序，也可以用来创建网络服务、智能设备应用程序和 Office 插件，还可开发安卓平台应用及 IOS 平台应用。总之 Visual Studio 开发环境使用广泛，读取 GPIO 工具即使用 VS 开发环境进行开发。

（二）开发环境搭建

首先，下载并安装 Visual studio，打开 VS 新建 -> 项目：选择 MFC 应用程序，名称这里用 RwSKLGpioTool，然后点击确定。

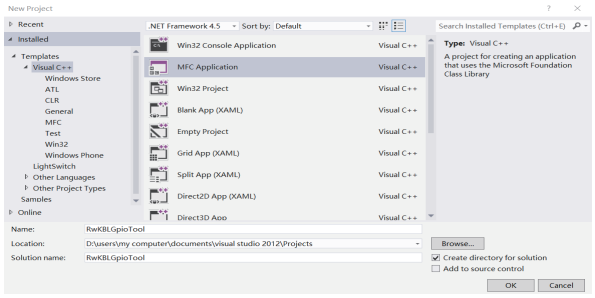

图 3　MFC 应用程序创建（一）

出现 MFC 生成向导：这里选择基于对话框，其他默认。然后点击完成。这样一个 MFC 应用程序环境搭建完成。

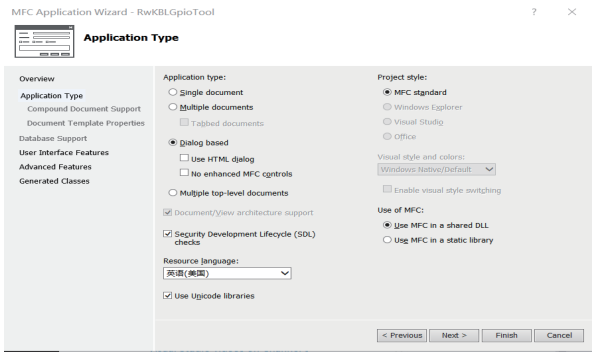

图 4　MFC 应用程序创建（二）

（三）工具开发

RwSKLGpioTool 项目生成后，通过拖拉控件及修改空间属性来设计工具主界面。根据 GPIO 分组，GPIO 分为 7 组，每组有多根引脚。工具图形界面需要 2 个选择框 Group 和 SubGroup 分别来供用户选择 GPIO 组数和引脚序号。每个 GPIO 引脚寄存器有 32 个比特位，因此工具界面上有 32 个比特位显示框，用来显示引脚寄存器每个比特位的数值。为了方便使用者更快速读取寄存器，界面上还显示寄存器八进制数据。界面最下方是信息显示框，显示选择引脚的主要二进制位信息。

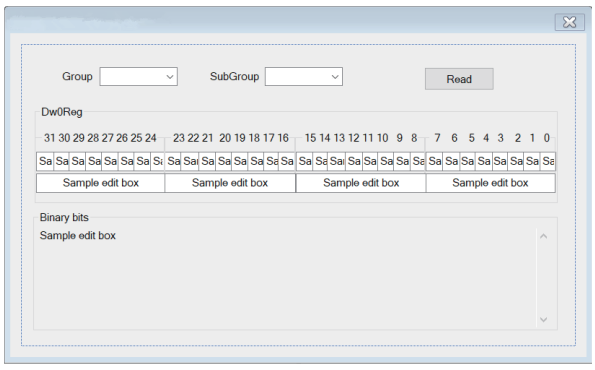

图 5　读取 GPIO 工具界面设计

图形界面设计完成后，双击控件，可自动跳转到点击控件的事件回调处理函数。在回调函数里面编写代码，当工具运行时用户点击控件，那么回调函数即可运行。读 GPIO 工具界面最重要的即 Read 按钮，因此 Read 按钮回调函数是重中之重。工具 SKL GPIO 介绍，Read 按钮回调函数设计如下：

```
void CRwGpioDlg::OnBnClickedButton1()
{
    UINT32 DataValue=0;
    UINT8 Group, Sub, Gpiogroup;
    UINT16 Subgroup, GpioAdrr = 0x1600;
    UINT32 PCRADRR = NULL;
    CString bin;
    CString strTemp;
    CString str0, str1, str2, str3, str4, str5, str6, str7, str8, str9, Groupp, Subb;
    UpdateData(TRUE);
    Group = ((CComboBox*)GetDlgItem(IDC_COMBO1))->GetCurSel();
    Gpiogroup = mPchLpGpioGroupInfo[Group].Community;
    Sub = ((CComboBox*)GetDlgItem(IDC_COMBO2))->GetCurSel();
    Subgroup = mPchLpGpioGroupInfo[Group].PadCfgOffset + Sub * 0x8;
    PCRADRR = (PCH_PCR_BASE_ADDRESS|((UINT8)(Gpiogroup) << 16)|(UINT16)(Subgroup));
    DataValue = MmioRead32(PCRADRR);
    bin = DecimalToBinary(DataValue);
    UpdateData(TRUE);
    str_GPNCONFIG_DataBin0 = bin[31]; //_T("1");
....
    str_GPNCONFIG_DataBin31 = bin[0]; //_T("1");
    bin.Format(_T("%08X"), DataValue);
    str_GPNCONFIG_Hex3.Format(_T("%02X"), (UINT8)(DataValue >> 24));
    str_GPNCONFIG_Hex2.Format(_T("%02X"), (UINT8)(DataValue >> 16));
    str_GPNCONFIG_Hex1.Format(_T("%02X"), (UINT8)(DataValue >> 8));
    str_GPNCONFIG_Hex0.Format(_T("%02X"), (UINT8)(DataValue));
    UpdateData(FALSE);
     ((CButton*)GetDlgItem(IDC_COMBO1))->GetWindowText(str5);
     ((CButton*)GetDlgItem(IDC_COMBO2))->GetWindowText(str6);
      strTemp = _T("Reading group[") + str5 + _T("]pin[") + str6 + _T("]Gpio register.....\r\n\r\n");
    // print b10
      if ((str_GPNCONFIG_DataBin11.Compare(_T("0")) == 0)){
                if((str_GPNCONFIG_DataBin10.Compare(_T("0")) == 0)){//bit11=0.bit10=0
          strTemp += _T("Dw0[Bit11~Bit10]----Pad Mode: \t00 = GPIO mode .\r\n");
```

```
                                    //print b9 or b8
            if （（str_GPNCONFIG_DataBin9.Compare（_T（"0"））
== 0））{//bit9=0
              if （（str_GPNCONFIG_DataBin8.Compare（_T（"0"））
== 0））{
                                strTemp += _T( "Dw0 [Bit9]----GPIO
RX Disable：\t0 = Input .\r\n"）；
                                strTemp += _T（"Dw0 [Bit8]----GPIO
RX Disable：\t0 = Output .\r\n"）；
          strTemp += _T（"This Pin is an double Pin，can both use
GPI or GPO .\r\n"）；
                              }
                              else{
                                strTemp += _T( "Dw0 [Bit9]----GPIO
RX Disable：\t0 = Input .\r\n"）；
                                if （（str_GPNCONFIG_DataBin1.
Compare（_T（"1"））== 0））{
                                 strTemp += _T( "DW0 [Bit1]---GPIO
RX State：\t 1 = High.\r\n"）；
                                }
                                else {
                                        strTemp += _T（"Dw0
[Bit1]---GPIO RX State：\t 0 = Low.\r\n"）；
                                }
                              }
                        }
                        else {//bit9=1
                          if （（str_GPNCONFIG_DataBin8.Compare（_
T（"0"））== 0））{
          strTemp += _T（"Dw0 [Bit8]----GPIO TX Disable：\t 0 =
Output .\r\n"）；
                                if （（str_GPNCONFIG_DataBin0.
Compare（_T（"1"））== 0））{
                                 strTemp += _T( "Dw0 [Bit0]---GPIO
TX State：\t 1 = High.\r\n"）；
                                }
                                else {
                                 strTemp += _T( "Dw0 [Bit0]---GPIO
TX State：\t 0 = Low.\r\n"）；
                                }
                              }
                              else {
                                strTemp += _T( "Dw0 [Bit9]----GPIO
RX Disable：\t1 = Disable Input .\r\n"）；
                                strTemp += _T( "Dw0 [Bit8]----GPIO
RX Disable：\t1 = Disable Output .\r\n"）；
                                 strTemp += _T（"This Pin is not set
GPI or GPO.\r\n"）；
                              }
                          }
                        }
                        else {//bit11 = 0.bit10 = 1
                          strTemp += _T（"Dw0 [Bit11~Bit10]----Pad
Mode：\t 01 = Native mode 1 .\r\n"）；
                        }
                    }
                    else {//biy11=1
                      if （（str_GPNCONFIG_DataBin10.Compare（_T（"0"））
== 0））{//bit11=1.bit10=0
                        strTemp += _T（"Dw0 [Bit11~Bit10]----Pad
Mode：\t 10 = Native mode 2.\r\n"）；
                        }
                      else{
                        strTemp += _T（"Dw0 [Bit11~Bit10]----Pad
Mode：\t 11 = Native mode 3.\r\n"）；
                      }
                    }
                    EditValue = strTemp;
                    UpdateData（FALSE）；
                }
```

（四）实现

图形界面设计好，回调函数实现之后，即可编译调试代码。编译成功后可执行软件就生成了，把生成的可执行 EXE 文件拷贝到 S K L 平台笔记本上即可运行程序进行测试。下图是在 SKL 平台机器上运行读取 GPIO 工具的结果图。示例中使用者读取的是第三组 GPIO 里面的引脚 8（引脚编号从 0 开始）信息。根据读取得信息我们可以直观地知道 C8 引脚是一个双引脚，即可为输入引脚，也可以为输出引脚。

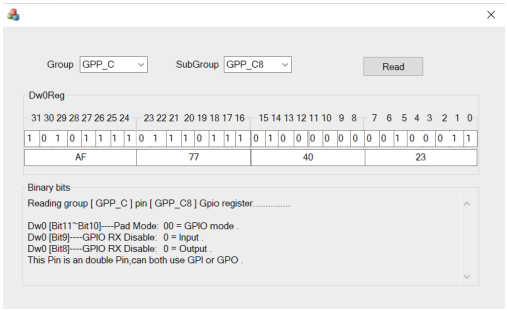

图 6　读取 GPIO 工具运行效果

三、总结

使用读取 GPIO 工具能够直观看到主板上 GPIO 引脚的配置状态和引脚数值，极大地方便了开发者，不用每次都去查找标准文档，查询每个二进制数值含义，直观有效地获得想要的信息；对应业余爱好者来说使用这个工具也能够深入学习 GPIO 相关信息。但是这个工具也有不足之处，工具是基于 SKL 平台开发的，随着英特尔平台芯片不断发展，架构不断升级，这个工具对后面的平台可能就不适应了。新平台需要开发新的配套读取 GPIO 工具。但是对目前来说，这个工具还是可以很好地被开发者使用。

参考文献：

[1] 小年 . 全面接触 BIOS[J]. 电脑爱好者 .2020（18）：62-67.

[2] 疏凤 .YHFT — DSP GPIO 部件的设计实现及其应用研究 [D]. 北京：国防科学技术大学，2010.

基金项目：本文系安徽省重大教学改革研究项目“省级继续教育网络园区背景下的高等学历继续教育教学过程质量保障与评价体系构建与实践”（课题编号：2017jyxm0965）；安徽省级教学研究项目“基于区块链的安徽继续教育公共机制研究”（课题编号：2019zdjg08）

作者简介：张月蓉（1989-），女，安徽滁州人，助教，硕士，研究方向：远程教育、计算机应用。