MSP430 单片机 MSP430I2XX 系列驱动库 使用指南

版本号：1.97.00.19

序言

1.1 综述

德州仪器®MSP430®外设驱动程序库是一组基于访问 MSP430i2xx 家族微控制器的外设驱动程序。虽然它们不是纯粹的操作系统意义上的驱动程序,但他们提供一种容易使用外设的机制。

驱动的功能和 组织都是遵循以下设计目标：

- 他们都是完全由 C 语言编写（除了那些 C 绝对不可能实现的功能外）。
- 他们演示如何在常见的操作模式下使用外设。
- 他们很容易理解。
- 他们在内存和处理器的使用上是合理高效的。
- 他们尽可能独立（模块化）。
- 在可能的情况下，计算指令在编译时候执行，而不是在运行时完成（代码编译的高效性）。
- 他们可以适用于不只一个的工具链构建（目前提供三种：CSS、GCC、IAR）。

基于以上的设计目标可能有以下不好的影响：

- 从一个代码大小和执行速度来考虑，驱动程序不一定是高效的代码。而最有效的操作外设的代码是在应用程序有特定需求下进行量身定制而编写的。更进一步的对驱动程序代码尺寸优化将会使得驱动代码更难以理解。
- 驱动程序不能够实现硬件的全部功能。一些外设提供的复杂功能不能通过本库函数提供的驱动程序实现。但是你可以使用现有代码作为参考来添加额外的功能支持。
- API 清除了所有错误校验代码。因为错误校验通常只是用于最初的程序开发期间，它可以被删除用于改善代码的大小和速度。

对于许多应用程序，可以使用驱动程序来开发。单是在某些情况下，驱动程序为了满足应用程序的功能，存储或处理的需求将会增强或重写（代码的冗余度提高了）。如果是这样的情况，可以把现有的驱动程序作为操作外设的参考程序。

一些驱动库 API 取相应的外设的基地址作为第一个参数。这个基地址是是从 MSP430 单片机特点的头文件（或单片机数据表）获取的。各种外设的例程演示如何使用这些基地址。

驱动库支持以下工具链:

- IAR Embedded Workbench®
- Texas Instruments Code Composer Studio™

使用断言语句调试

默认情况下禁用断言语句。

要启用断言语句需要编辑 hw_regaccess.h ，该文件在 inc 文件夹。对#define NDEBUG 进行添加注释符号。变成//#define NDEBUG

也就是把#define NDEBUG 这句注释掉。断言只有在项目优化尺寸时候才在 CCS 中起作用。

版权声明：以上内容均翻译自 TI 英文版 MSP430i2xx 系列库函数使用手册引言部分。有不合理和错误之处，请回帖说明，在此谢谢各位网友，此文档内容版权还归 TI 所有。

第一章　怎样使用库函数（CCS 版）

1.1　开发环境介绍

CCS6.1 是目前 CCStudio 的最新版本，也是支持最全面的版本，本文档所有内容均在该版本下测试完成。经过作者测试 CCS5 系列还没有引入库函数概念，建议升级到最新版 CCStudio6.1.

下载地址：

http://processors.wiki.ti.com/index.php/Download_CCS　　或通过百度网盘下载

http://pan.baidu.com/s/1dDhaY0T

——

启动 CCS6，单击菜单栏的 View-> Resource Explorer（Examples）

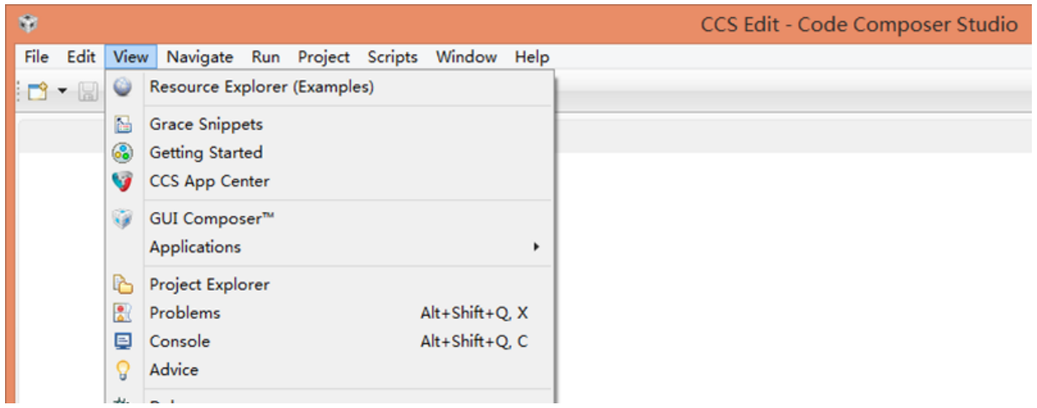

图 1

我们看到 CCS 菜单栏 View 下第一个 Resource explorer(Examples)，单击。弹出窗口就是 TI Resource Explorer。如图 2，我们可以看到我们安装的 MSP430ware/Lirbrary

如果你安装时候没有勾选安装 MSP430ware，请通过 Help/check for updates 或 App Center 升级安装。

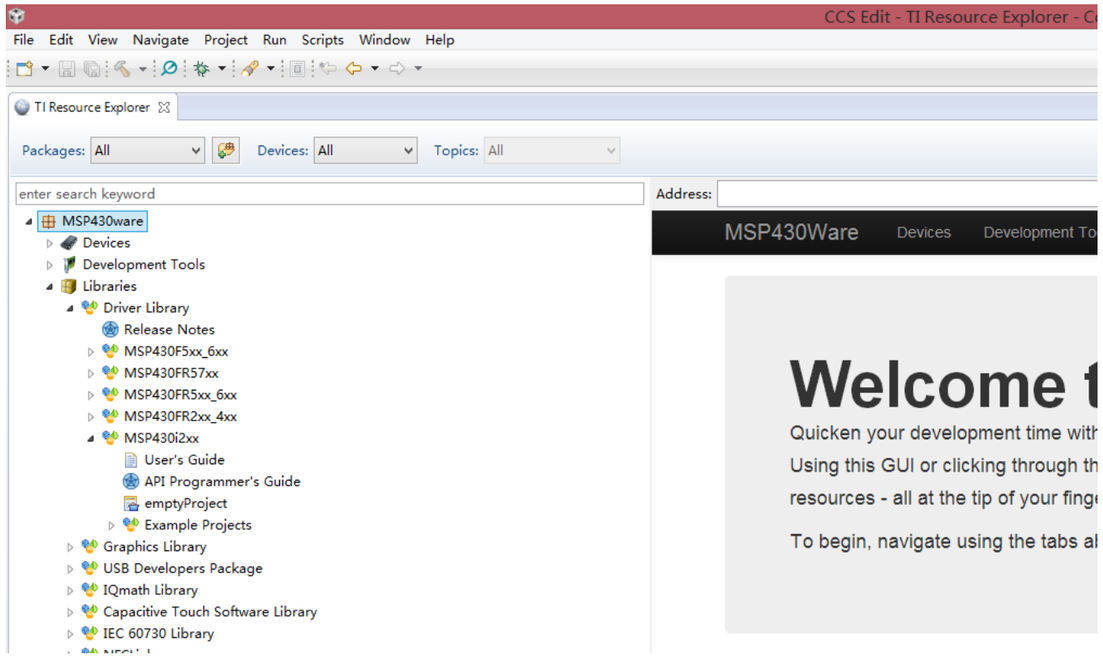

图 2

点击 MSP430ware ，你将看到介绍页面. 通过左边的列表可以找到我们这里要介绍的 MSP430i2xx 系列的用户手册（User’s Guide）、API 编程手册（API Programmer’s Guide）以及创建空项目（emptyProject），还有例程（Example Projects）。看右侧的窗口有几个方块，我们单击第一个 Driver Library。弹出如下窗口。单击蓝色字体的 here，

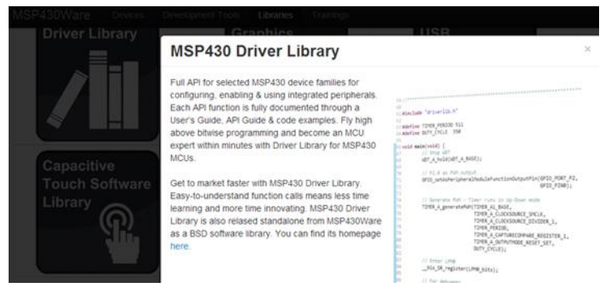

图 3

将会在 CCS 右侧的窗口弹出网页，如图 4。在该页面可以下载到最新版的 MSP430 驱动库函数包。或登录 http://www.ti.com/tool/mspdriverlib

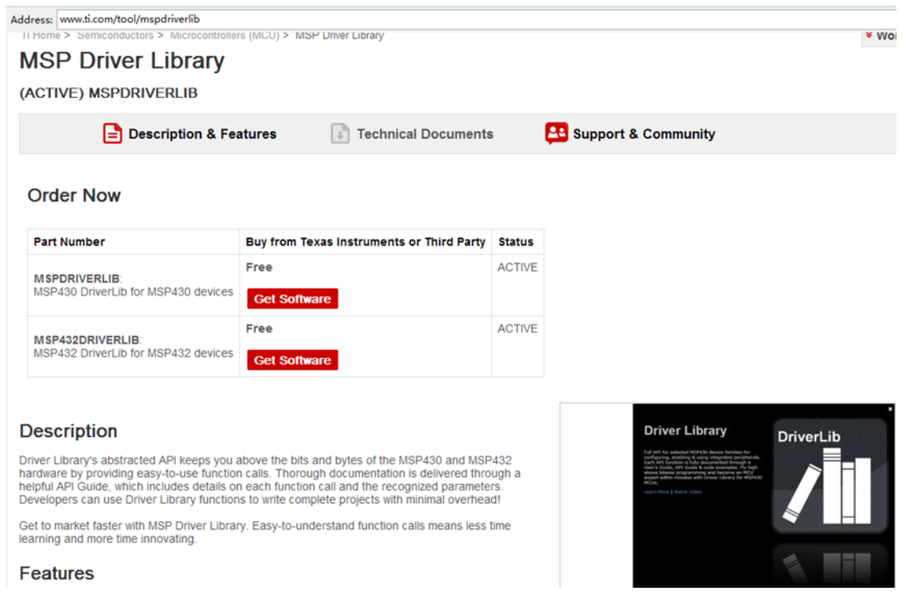

图 4

你可以在你的安装目录下找到本教程将要讲解的库函数源文件。

C:\ti\msp430\MSP430ware_1_97_00_47\driverlib\driverlib\MSP430i2xx

或者在 CCS 的 TI Resource Explorer 里找到 MSP430ware->Libraries->MSP430i2xx->API Programmer’s Guide。

这时右边窗口会出现如下图所示。左边分别有三个选项：Modules（模块）、Data Structures（数据结构）、Files（文件）。如图 5 所示。

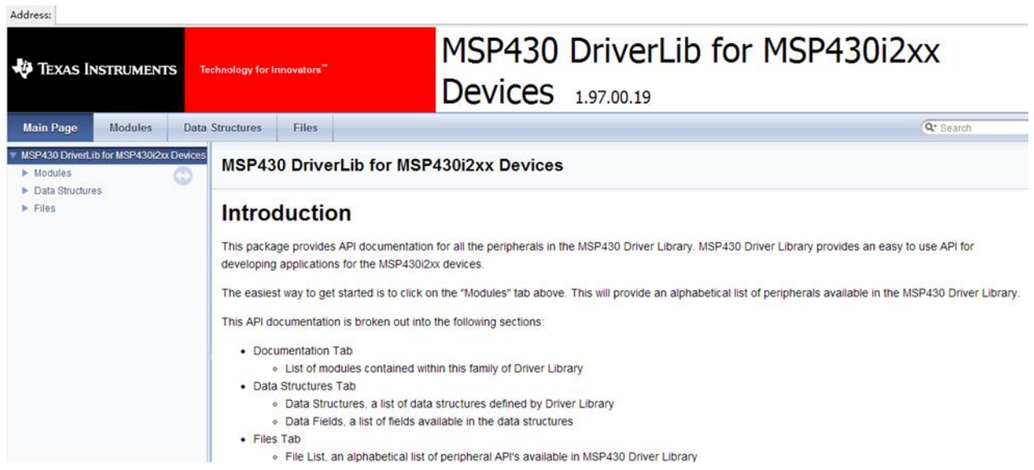

图 5

通过 Modules(模块)可以查看该系列单片机所有外设模块的库函数介绍，如图 6 所示为 GPIO_clearInterruptFlag 函数的相关说明。

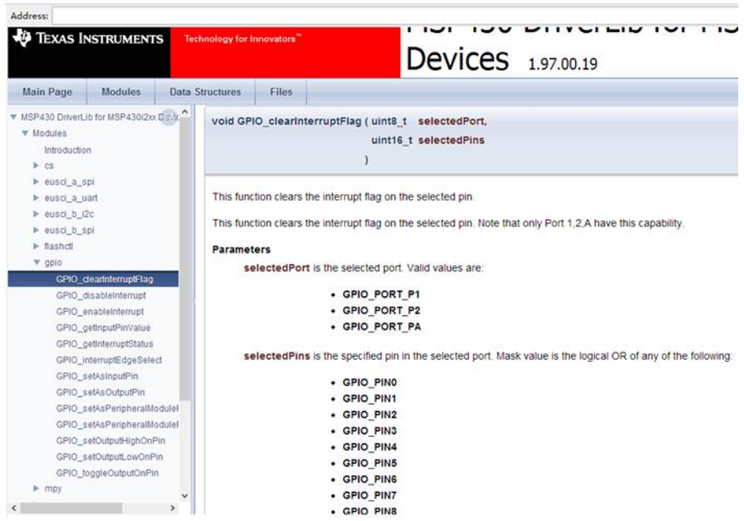

图 6

通过 Files 选项我们可以看到头文件列表，我们找到 gpio.h 单击，右边就弹出该文件的内容，我们翻页找到上面对应 那个函数，通过函数的注释，我们也可以看懂该函数包含了两个参数，以及参数的取值范围，该函数的作用。我们发现默认的注释字体是红色的，如果你觉得 CCS 给的配色十分的不愉快，可以使用 Notepad++。

图 7

笔者经常使用 Notepad++阅读代码的，在 CCS 里你也可以通过修改配置来改变代码显示的配色，具体方法这里不作介绍。

```
//!        - \b GPIO_PIN14
//!        - \b GPIO_PIN15
//!
//! Modified bits of \b PxIFG register.
//!
//! \return None
//
//*****************************************************************************
extern void GPIO_clearInterruptFlag(uint8_t selectedPort,
                                    uint16_t selectedPins);
```

图 8

1.2 怎么使用 Driverlib(驱动库)创建一个新的用户工程

单击 File->New->CCS Project，弹出新建工程窗口，如图 9 所示。

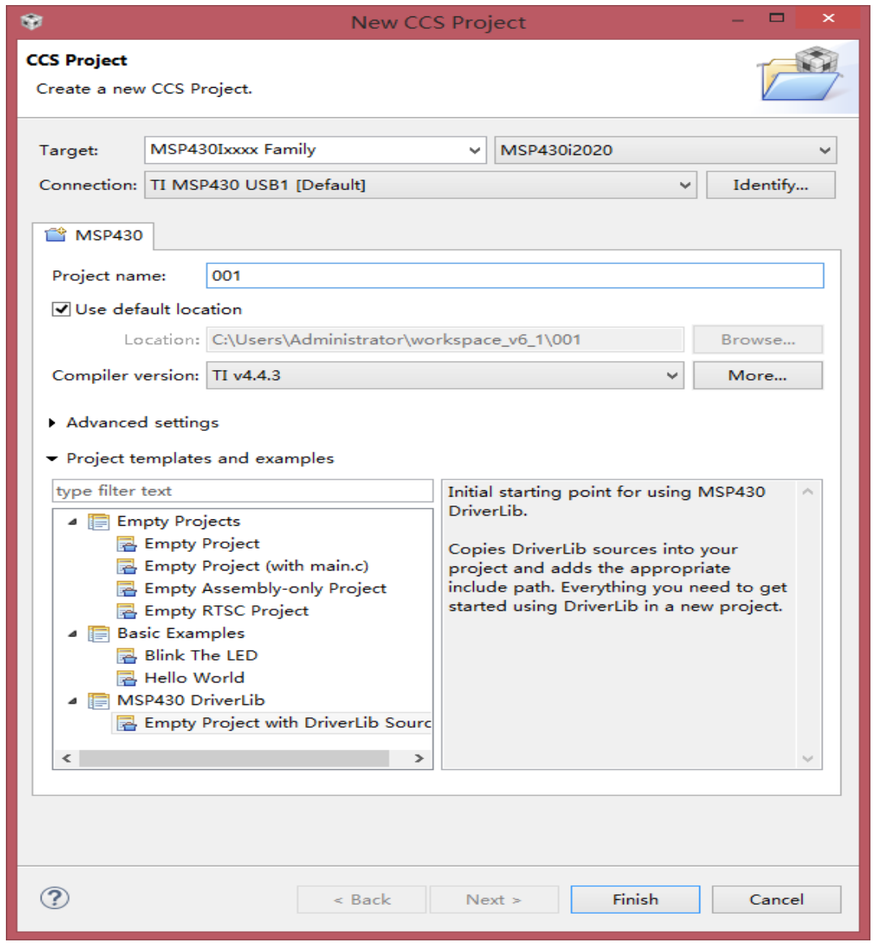

图 9

通过选项进行配置，如果图 9 所示，Target 选择 MSP430Ixxx Family，器件型号选择 MSP430i2020。填入 Project name(项目名字)。在 Project templates and examples（项目模板和示例）选择 MSP430 DriverLib 下的 Empty Project with DriverLib Source，之后单击 Finish 完成创建基于库函数的项目。并且弹出如下窗口。

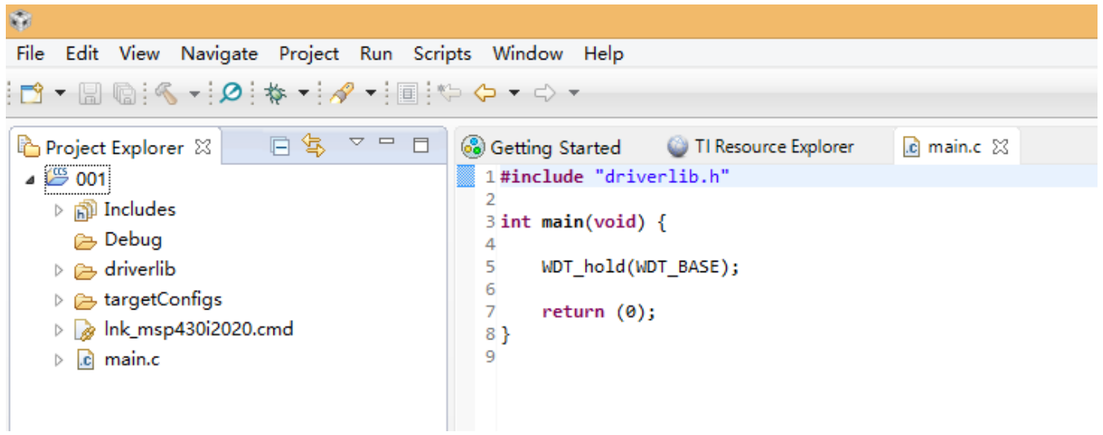

图 10

1.3 怎样给已经存在的项目添加库函数

在已存在的应用项目单击右键，单击 Source -> Apply Project Template，如图 11。打开向导窗口，如图 12。

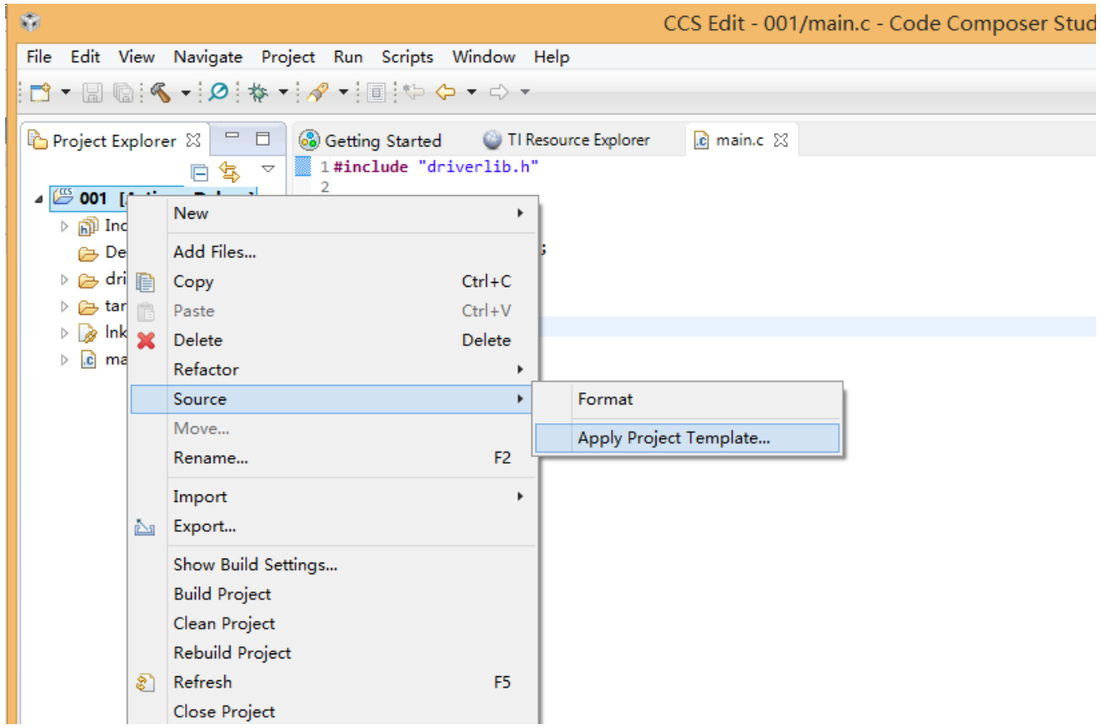

图 11

在图 12 选择 Add Local Copy of DriverLib. 单击 Finish 完成向现有的项目添加一个 DriverLib MSP430i2xx 副本。

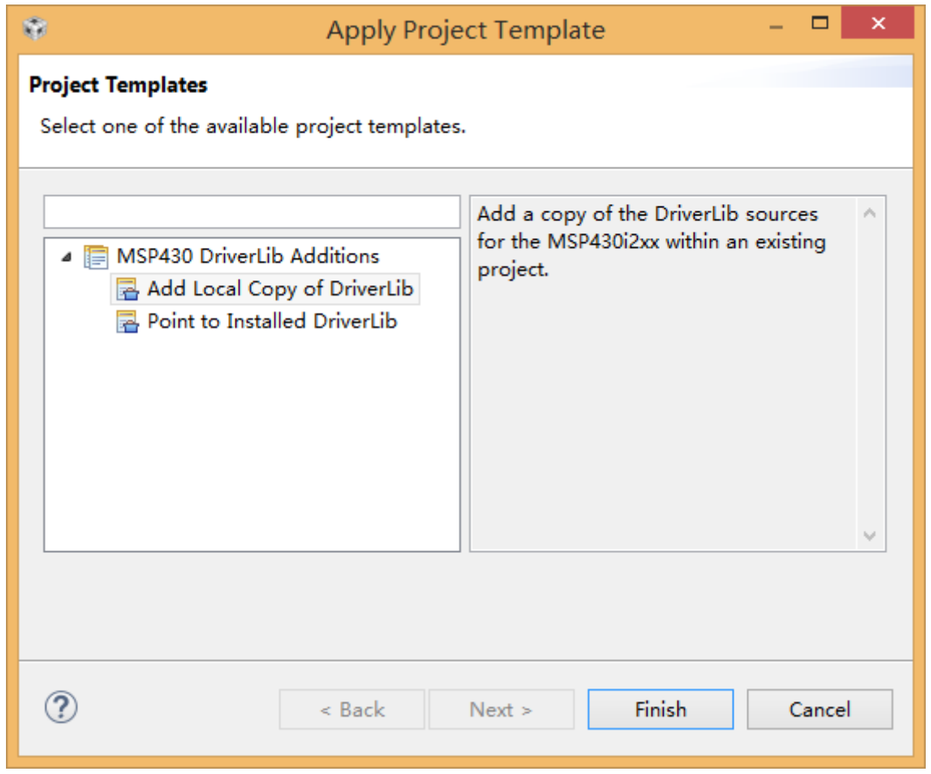

图 12

第二章 GPIO

GPIO 是 General Purpose Input/Output（通用输入输出）的简称，是单片机最基本，最必须，最重要的外设，每个 GPIO 端口可通过软件分别配置成输入或输出。

数字 I/O(GPIO)API（Application Programming Interface,应用程序编程接口）为使用 MP430i2xx GPIO 模块提供了一组函数。这些函数可以设置、使能使用 I/O 管脚（针脚、插脚），设置他们是否有中断，以及对管脚值的读和写。数字 I / O 功能包括:

- 可对单独的 I/O 独立编程
- 可对输入输出任意组合
- 可单独配置 P1 和 P2 端口中断，另外一些型号可能包含其他端口的中断。
- 具有独立的输入和输出数据寄存器
- 可单独配置上拉或下拉电阻

2.1 本章引言

该系列单片机具备的 I/O 端口数量最高可达 12 组（P1 至 P11，外加 PJ 端口）。大部分呢端口每组包含 8 个管脚；然后，一些端口可用的管脚较少，不足 8 个（端口包含可用管脚的具体情况需要查看对应芯片的数据手册）。每个 I/O 管脚可用分别配置为输入或输出，并且每个可用单独进行读/写操作。

端口 P1 和 P2 都具备中断功能。P1 和 P2 两组端口的每一位管脚都可以单独配置成在输入一个上升沿或下降沿时候触发中断的功能。

P1 所有的管脚位中断源自一个单独的中断向量 P1IV，P2 所有的管脚中断院子另外一个单独的中断向量 P2IV。在一些单片机中，可能具备额外中断功能的端口，且包含他们各自的中断向量（具体情况，需要从该型号单片机的数据手册中获取）。

单个端口可以作为可以以字节宽度进行访问，也可以合二为一组合成字宽，以字的格式进行访问。P1/P2, P3/P4, P5/P6, P7/P8,四对端口分别对应 PA、PB、PC、PD。除了中断向量 P1IV 和 P2IV 外，所有的端口寄存器是按照这种命名约定方式处理的，也就是说不存在中断向量 PAIV。当以字为单位操作 PA 时候，所有的 16 位被写入端口。当用字节操作 PA 端口的低字节时候，高字节保持不变。同样，用字节指令写 PA 端口的高字节时候，低字节不变。当写端口时候，使用的位宽少于该端口的最大位宽时候，不用的位是不受影响的。端口 PB、P、PD、PE 和 PF 也是类似的。

使用字操作读 PA 端口，可以读到 16 位管脚的全部状态。使用字节操作，读 PA 端口（P1 或 P2）的低字节或高字节，分别只能读出低字节或高字节。使用字节操作，读 PA 端口，并存储到一个通用寄存器，本字节只被写入到寄存器的最低有效字节。目的寄存器的高位有效字节将自动被清除。端口 PB、PC、PD、PE 和 PF 类似。当读那些可能少于最大位宽的端口时候，未使用的位读作 0（类似于端口 PJ）。

GPIO 管脚可使用函数 GPIO_setAsOutputPin() 或 GPIO_setAsInputPin()配置为一个 I/O 管脚。使用外设分配函数 GPIO_setAsPeripheralModuleFunctionOutputPin()或 GPIO_setAsPeripheralModuleFunctionInputPin().可以把 GPIO 管脚替换成外围模块功能（即使用管脚 I/O 功能外的其他功能）。

这个驱动程序包含在 gpio.c 文件里，gpio.h 头文件包含该应用程序使用的 API 定义。

2.2 函数总览

1	**void GPIO_setAsOutputPin (uint8_t selectedPort, uint16_t selectedPins)**
	该函数可以配置所选管脚作为输出管脚
2	**void GPIO_setAsInputPin (uint8_t selectedPort, uint16_t selectedPins)**
	该函数可以配置所选管脚作为输入管脚
3	**void GPIO_setAsPeripheralModuleFunctionOutputPin (uint8_t selectedPort, uint16_t selectedPins, uint8_t mode)**
	该函数配置所选管脚上输出方向的外设
4	**void GPIO_setAsPeripheralModuleFunctionInputPin (uint8_t selectedPort, uint16_t selectedPins, uint8_t mode)**
	该函数配置所选管脚上输入方向的外设
5	**void GPIO_setOutputHighOnPin (uint8_t selectedPort, uint16_t selectedPins)**
	该函数在所选管脚上输出高电平
6	**void GPIO_setOutputLowOnPin (uint8_t selectedPort, uint16_t selectedPins)**
	该函数在所选管脚上输出低电平
7	**void GPIO_toggleOutputOnPin (uint8_t selectedPort, uint16_t selectedPins)**
	该函数翻转所选管脚的输出电平
8	**uint8 t GPIO_getInputPinValue (uint8_t selectedPort, uint16_t selectedPins)**

	该函数可以获取所选管脚的输入值
9	**void GPIO_enableInterrupt (uint8_t selectedPort, uint16_t selectedPins)**
	该函数使能所选管脚的端口中断功能
10	**void GPIO_disableInterrupt (uint8_t selectedPort, uint16_t selectedPins)**
	该函数禁用所选管脚端口的中断
11	**uint16 t GPIO_getInterruptStatus (uint8_t selectedPort, uint16_t selectedPins)**
	该函数可以获取被选中的端口的中断状态
12	**void GPIO_clearInterruptFlag (uint8 t selectedPort, uint16 t selectedPins)**
	该函数可以清除所选管脚的中断标志
13	**void GPIO_interruptEdgeSelect (uint8_t selectedPort, uint16_t selectedPins, uint8_t edgeSelect)**
	该函数可以对所选管脚的中断触发边沿类型进行选择（是上升沿触发还是下降沿触发）

详细描述

GPIO API 函数分为三组：配置 GPIO 管脚的、处理中断的和访问管脚电平值的。

配置 GPIO 管脚的有:

- **GPIO_setAsOutputPin()**
- **GPIO_setAsInputPin()**
- **GPIO_setAsInputPinWithPullDownresistor()**
- **GPIO_setAsInputPinWithPullUpresistor()**
- **GPIO_setAsPeripheralModuleFunctionOutputPin()**
- **GPIO_setAsPeripheralModuleFunctionInputPin()**

处理 GPIO 中断的有：

- **GPIO_enableInterrupt()**
- **GPIO_disableInterrupt()**
- **GPIO_clearInterruptFlag()**
- **GPIO_getInterruptStatus()**
- **GPIO_interruptEdgeSelect()**

管脚状态访问的：

- **GPIO_setOutputHighOnPin()**
- **GPIO_setOutputLowOnPin()**
- **GPIO_toggleOutputOnPin()**
- **GPIO_getInputPinValue()**

函数文档

void GPIO_clearInterruptFlag (uint8 t selectedPort, uint16 t selectedPins)

该函数可以清除所选管脚的中断标志。需要注意的是只针对端口 1、2、A 有效。因为只有 P1,P2,PA(前面说过 P1 和 P2 组合后作为字标记为 PA)有效。

该函数具有两个参数：selectedPort（所选端口）和 selectedPins（所选管脚）。

selectedPort 可选的有效值有三个：GPIO_PORT_P1、GPIO_PORT_P2、GPIO_PORT_PA。

selectedPins 是所选端口上的管脚。其掩码值可以是 GPIO_PIN0、GPIO_PIN1、GPIO_PIN2……GPIO_PIN15 等十六个值的逻辑或。

该函数是通过修改寄存器 PxIFG 实现，返回值 None(空)。

void GPIO_disableInterrupt (uint8_t selectedPort, uint16_t selectedPins)

该函数禁用所选管脚的端口中断。需要注意的是，只有端口 P1,P2,PA 具有该功能。

该函数具有两个参数：selectedPort（所选端口）和 selectedPins（所选管脚）。

selectedPort 可选的有效值有三个：GPIO_PORT_P1、GPIO_PORT_P2、GPIO_PORT_PA。

selectedPins 是所选端口上的管脚。其掩码值可以是 GPIO_PIN0、GPIO_PIN1、GPIO_PIN2……GPIO_PIN15 等十六个值的逻辑或。

该函数是通过修改寄存器 PxIE 实现，返回值 None(空)。

void GPIO_enableInterrupt (uint8_t selectedPort, uint16_t selectedPins)

该函数使能所选管脚的端口中断。需要注意的是，只有端口 P1,P2,PA 具有该功能。

该函数具有两个参数：selectedPort（所选端口）和 selectedPins（所选管脚）。

selectedPort 可选的有效值有三个：GPIO_PORT_P1、GPIO_PORT_P2、GPIO_PORT_PA。

selectedPins 是所选端口上的管脚。其掩码值可以是 GPIO_PIN0、GPIO_PIN1、GPIO_PIN2……GPIO_PIN15 等十六个值的逻辑或。

该函数是通过修改寄存器 PxIE 实现，返回值 None(空)。

uint8 t GPIO_getInputPinValue (uint8_t selectedPort, uint16_t selectedPins)

该函数获取所选管脚的输入值。

该函数具有两个参数：selectedPort（所选端口）和 selectedPins（所选管脚）。

selectedPort 可选的有效值有 18（11+7）个：GPIO_PORT_P1、GPIO_PORT_P2……GPIO_PORT_P11，GPIO_PORT_PA、GPIO_PORT_PB、GPIO_PORT_PC、GPIO_PORT_PD、GPIO_PORT_PE、GPIO_PORT_PF、GPIO_PORT_PJ。

注意：数字标记的为以字节为操作位宽（8 位）的，字母标记的为以字为位宽（16 位）操作的。

selectedPins 是所选端口上的管脚。其掩码值可以是 GPIO_PIN0、GPIO_PIN1、GPIO_PIN2……GPIO_PIN15 等十六个值的逻辑或。

返回值：GPIO_INPUT_PIN_HIGH 或 GPIO_INPUT_PIN_LOW 二者中之一。即管脚的状态值。

uint16 t GPIO_getInterruptStatus (uint8_t selectedPort, uint16_t selectedPins)

该函数用于获取所选管脚的中断状态。需要注意的是只有 P1、P2、PA 具有该功能。

该函数具有两个参数：selectedPort（所选端口）和 selectedPins（所选管脚）。

selectedPort 可选的有效值有三个：GPIO_PORT_P1、GPIO_PORT_P2、GPIO_PORT_PA。

selectedPins 是所选端口上的管脚。其掩码值可以是 GPIO_PIN0、GPIO_PIN1、GPIO_PIN2……GPIO_PIN15 等十六个值的逻辑或。

返回值： GPIO_PIN0、GPIO_PIN1、GPIO_PIN2……GPIO_PIN15 等十六个值的逻辑或。用于显示所选管脚的中断状态（默认值：0）。

void GPIO_interruptEdgeSelect (uint8_t selectedPort, uint16_t selectedPins, uint8_t edgeSelect)

该函数选择什么边沿触发中断，即选择是上升沿还是下降沿触发所选管脚中断。

该函数具有三个参数：selectedPort（所选端口）、selectedPins（所选管脚）和 edgeSelect（边沿选择）。

selectedPort 可选的有效值有 18（11+7）个：GPIO_PORT_P1、GPIO_PORT_P2……GPIO_PORT_P11，GPIO_PORT_PA、GPIO_PORT_PB、GPIO_PORT_PC、GPIO_PORT_PD、GPIO_PORT_PE、GPIO_PORT_PF、GPIO_PORT_PJ。

注意：数字标记的为以字节为操作位宽（8 位）的，字母标记的为以字为位宽（16 位）操作的。

selectedPins 是所选端口上的管脚。其掩码值可以是 GPIO_PIN0、GPIO_PIN1、GPIO_PIN2……GPIO_PIN15 等十六个值的逻辑或。

edgeSelect 是对中断触发边沿的选择。其值有两个，GPIO_HIGH_TO_LOW_TRANSITION（下降沿触发）和 GPIO_LOW_TO_HIGH_TRANSITION（上升沿触发）。

该函数是通过修改寄存器 PxIES 实现，返回值 None(空)。

void GPIO_setAsInputPin (uint8_t selectedPort, uint16_t selectedPins)

该函数把所选管脚配置为输入管脚。

该函数具有两个参数：selectedPort（所选端口）和 selectedPins（所选管脚）。

selectedPort 可选的有效值有 18（11+7）个：GPIO_PORT_P1、GPIO_PORT_P2……GPIO_PORT_P11，GPIO_PORT_PA、GPIO_PORT_PB、GPIO_PORT_PC、GPIO_PORT_PD、GPIO_PORT_PE、GPIO_PORT_PF、GPIO_PORT_PJ。

注意：数字标记的为以字节为操作位宽（8 位）的，字母标记的为以字为位宽（16 位）操作的。

selectedPins 是所选端口上的管脚。其掩码值可以是 GPIO_PIN0、GPIO_PIN1、GPIO_PIN2……GPIO_PIN15 等十六个值的逻辑或。

该函数是通过修改寄存器 PxDIR、PxREN、和 PxSEL 的位实现，返回值 None(空)。

void GPIO_setAsOutputPin (uint8_t selectedPort, uint16_t selectedPins)

该函数把所选管脚配置为输出管脚。

该函数具有两个参数：selectedPort（所选端口）和 selectedPins（所选管脚）。

selectedPort 可选的有效值有 18（11+7）个：GPIO_PORT_P1、GPIO_PORT_P2……GPIO_PORT_P11，GPIO_PORT_PA、GPIO_PORT_PB、GPIO_PORT_PC、GPIO_PORT_PD、GPIO_PORT_PE、GPIO_PORT_PF、GPIO_PORT_PJ。

注意：数字标记的为以字节为操作位宽（8 位）的，字母标记的为以字为位宽（16 位）操作的。

selectedPins 是所选端口上的管脚。其掩码值可以是 GPIO_PIN0、GPIO_PIN1、GPIO_PIN2……GPIO_PIN15 等十六个值的逻辑或。

该函数是通过修改寄存器 PxDIR 和 PxSEL 的位实现，返回值 None(空)。

void GPIO_setAsPeripheralModuleFunctionInputPin (uint8_t selectedPort, uint16_t selectedPins, uint8_t mode)

该函数配置所选管脚上输入方向的外设功能，要么第一功能模块，要么第二功能模块，要么第三功能模块。需要注意的是 MSP430F5xx/MSP4306xx 家族不支持这个函数模式。

该函数具有三个参数：selectedPort（所选端口）、selectedPins（所选管脚）和 mode（模式）。

selectedPort 可选的有效值有 18（11+7）个：GPIO_PORT_P1、GPIO_PORT_P2……GPIO_PORT_P11，GPIO_PORT_PA、GPIO_PORT_PB、GPIO_PORT_PC、GPIO_PORT_PD、GPIO_PORT_PE、GPIO_PORT_PF、GPIO_PORT_PJ。

注意：数字标记的为以字节为操作位宽（8 位）的，字母标记的为以字为位宽（16 位）操作的。

selectedPins 是所选端口上的管脚。其掩码值可以是 GPIO_PIN0、GPIO_PIN1、GPIO_PIN2……GPIO_PIN15 等十六个值的逻辑或。

mode 所选端口处的外设选择，一般端口提供了三种模式的外设模块，通过该参数可以进行选择使用。可选的值是 GPIO_PRIMARY_MODULE_FUNCTION、GPIO_SECONDARY_MODULE_FUNCTION 和 GPIO_TERNARY_MODULE_FUNCTION。如果不知道这三种外设模式分别是什么，可以查看多选单片机的技术手册的管脚分布图。

该函数是通过修改寄存器 PxDIR 和 PxSEL 的位实现，返回值 None(空)。

void GPIO_setAsPeripheralModuleFunctionOutputPin (uint8_t selectedPort, uint16_t selectedPins, uint8_t mode)

该函数配置所选管脚上输出方向的外设功能，要么第一功能模块，要么第二功能模块，要么第三功能模块。需要注意的是 MSP430F5xx/MSP4306xx 家族不支持这个函数模式。

该函数具有三个参数：selectedPort（所选端口）、selectedPins（所选管脚）和 mode（模式）。

selectedPort 可选的有效值有 18（11+7）个：GPIO_PORT_P1、GPIO_PORT_P2……GPIO_PORT_P11，GPIO_PORT_PA、GPIO_PORT_PB、GPIO_PORT_PC、GPIO_PORT_PD、GPIO_PORT_PE、GPIO_PORT_PF、GPIO_PORT_PJ。

注意：数字标记的为以字节为操作位宽（8 位）的，字母标记的为以字为位宽（16 位）操作的。

selectedPins 是所选端口上的管脚。其掩码值可以是 GPIO_PIN0、GPIO_PIN1、GPIO_PIN2……GPIO_PIN15 等十六个值的逻辑或。

mode 所选端口处的外设选择，一般端口提供了三种模式的外设模块，通过该参数可以进行选择使用。可选的值是 GPIO_PRIMARY_MODULE_FUNCTION、GPIO_SECONDARY_MODULE_FUNCTION 和 GPIO_TERNARY_MODULE_FUNCTION。如果不知道这三种外设模式分别是什么，可以查看多选单片机的技术手册的管脚分布图。

该函数是通过修改寄存器 PxDIR 和 PxSEL 的位实现，返回值 None(空)。

void GPIO_setOutputHighOnPin (uint8_t selectedPort, uint16_t selectedPins)

该函数在所选端口的管脚配置为输出高电平。

该函数具有两个参数：selectedPort（所选端口）和 selectedPins（所选管脚）。

selectedPort 可选的有效值有 18（11+7）个：GPIO_PORT_P1、GPIO_PORT_P2……GPIO_PORT_P11，GPIO_PORT_PA、GPIO_PORT_PB、GPIO_PORT_PC、GPIO_PORT_PD、GPIO_PORT_PE、GPIO_PORT_PF、GPIO_PORT_PJ。

注意：数字标记的为以字节为操作位宽（8 位）的，字母标记的为以字为位宽（16 位）操作的。

selectedPins 是所选端口上的管脚。其掩码值可以是 GPIO_PIN0、GPIO_PIN1、GPIO_PIN2……GPIO_PIN15 等十六个值的逻辑或。

该函数是通过修改寄存器 PxOUT 的位实现，返回值 None(空)。

void GPIO_setOutputLowOnPin (uint8_t selectedPort, uint16_t selectedPins)

该函数在所选端口的管脚配置为输出低电平。

该函数具有两个参数：selectedPort（所选端口）和 selectedPins（所选管脚）。

selectedPort 可选的有效值有 18（11+7）个：GPIO_PORT_P1、GPIO_PORT_P2……GPIO_PORT_P11，GPIO_PORT_PA、GPIO_PORT_PB、GPIO_PORT_PC、GPIO_PORT_PD、GPIO_PORT_PE、GPIO_PORT_PF、GPIO_PORT_PJ。

注意：数字标记的为以字节为操作位宽（8 位）的，字母标记的为以字为位宽（16 位）操作的。

selectedPins 是所选端口上的管脚。其掩码值可以是 GPIO_PIN0、GPIO_PIN1、GPIO_PIN2……GPIO_PIN15 等十六个值的逻辑或。

该函数是通过修改寄存器 PxOUT 的位实现，返回值 None(空)。

void GPIO_toggleOutputOnPin (uint8_t selectedPort, uint16_t selectedPins)

该函数在所选端口的管脚进行电平的翻转，进行高低电平切换。

该函数具有两个参数：selectedPort（所选端口）和 selectedPins（所选管脚）。

selectedPort 可选的有效值有 18（11+7）个：GPIO_PORT_P1、GPIO_PORT_P2……GPIO_PORT_P11，GPIO_PORT_PA、GPIO_PORT_PB、GPIO_PORT_PC、GPIO_PORT_PD、GPIO_PORT_PE、GPIO_PORT_PF、GPIO_PORT_PJ。

注意：数字标记的为以字节为操作位宽（8 位）的，字母标记的为以字为位宽（16 位）操作的。

selectedPins 是所选端口上的管脚。其掩码值可以是 GPIO_PIN0、GPIO_PIN1、GPIO_PIN2……GPIO_PIN15 等十六个值的逻辑或。

该函数是通过修改寄存器 PxOUT 的位实现，返回值 None(空)。

2.3 例程

下面简单的例程用于展示如何使用 GPIO API。

```
// 把 P1.0 设置为输出端口
GPIO_setAsOutputPin (GPIO_PORT_P1, GPIO_PIN0);

// 把 P1.4 设置为输入端口
GPIO_setAsInputPin (GPIO_PORT_P1, GPIO_PIN4);
while (1)
{
// 测试管脚 P1.4

if (GPIO_INPUT_PIN_HIGH == GPIO_getInputPinValue(GPIO_PORT_P1, GPIO_PIN4))

GPIO_setOutputHighOnPin (GPIO_PORT_P1, GPIO_PIN0);        // 如果 P1.4 输入高电平，置位 P1.0。
else
GPIO_setOutputLowOnPin (GPIO_PORT_P1, GPIO_PIN0);        //否则，管脚 P1.0 输清零
}
```

2.4 本章小结

本章节介绍了 GPIO 相关的 13 个操作函数，这些函数的参数变量有：selectedPort、selectedPin 和 mode。其中只有在使用 IO 管脚内的非 IO 功能的外设时候才使用第三个 参数 mode 进行选择功能。

表 1 和表 2 给出了只使用两个参变量 selectedPort、selectedPins 的函数，参变量所选范围的列表。

表 3 给出了使用三个变量 selectedPort、selectedPins、mode 的函数，参变量的所选范围。

我们注意到函数的命名规则是：

以函数

uint8_t GPIO_getInputPinValue(uint8_t selectedPort,uint16_t selectedPins) 为例

我们看到函数名以大些的 GPIO 开头，然后下划线，接着小写的 get 开头，后面的单子分别是大写字母打头。在引用库函数时候要注意到这些细节。另外看函数参变量类似，均告诉了你该值的类型与位宽。

另外注意到跟中断有关的函数只有一个 GPIO_interruptEdgeSelect() 中断边沿选择的 selectedPort 是可以有 18 个值可以选择，其余只有 3 个值可以选择。

另外要记住，对于管脚变量 selectedPins 输入时候是可以使用可选值的逻辑或运算的。

我们观察函数名，发现按照字面意思就可以知道该函数的作用，也就是属于自然语言表达的函数名。

例如：GPIO_setOutputLowOnPin (GPIO_PORT_P1, GPIO_PIN0);

GPIO_ 表示这个函数属于 GPIO 外设族的，后面 set(设置)Output（输出）Low(低电平)On(在)Pin(管脚)……，连起来读就是：设置输出低电平在管脚（xx,xx）只需要我们在参数里天上我们指定的管脚就行了，根本不用管关心需要操作什么寄存器，毕竟寄存器名字读起来没有自然语言表达更清楚明了。

希望大家踊跃发言参与本次学习活动，跟版主一起学习，并提出你的观点，本版主虚心接受大家的批评和建议，并愿和大家一起学习进步。活动结束将会根据网友提出的宝贵建议更新本文档，并将完整版上传给大家下载学习。

表 1

	selectedPort	selectedPins
GPIO_setAsOutputPin() GPIO_setAsInputPin() GPIO_setAsPeripheralModuleFunctionOutputPin() GPIO_setAsPeripheralModuleFunctionInputPin() GPIO_setOutputHighOnPin() GPIO_setOutputLowOnPin() GPIO_toggleOutputOnPin() GPIO_getInputPinValue() GPIO_interruptEdgeSelect()	GPIO_PORT_P1 GPIO_PORT_P2 GPIO_PORT_P3 GPIO_PORT_P4 GPIO_PORT_P5 GPIO_PORT_P6 GPIO_PORT_P7 GPIO_PORT_P8 GPIO_PORT_P9 GPIO_PORT_P10 GPIO_PORT_P11 GPIO_PORT_PA GPIO_PORT_PB GPIO_PORT_PC GPIO_PORT_PD GPIO_PORT_PE GPIO_PORT_PF GPIO_PORT_PJ	GPIO_PIN0 GPIO_PIN1 GPIO_PIN2 GPIO_PIN3 GPIO_PIN4 GPIO_PIN5 GPIO_PIN6 GPIO_PIN7 GPIO_PIN8 GPIO_PIN9 GPIO_PIN10 GPIO_PIN11 GPIO_PIN12 GPIO_PIN13 GPIO_PIN14 GPIO_PIN15

表 2

	selectedPort	selectedPins
GPIO_enableInterrupt() GPIO_disableInterrupt() GPIO_getInterruptStatus() GPIO_clearInterruptFlag()	GPIO_PORT_P1 GPIO_PORT_P2 GPIO_PORT_PA	GPIO_PIN0 GPIO_PIN1 GPIO_PIN2 GPIO_PIN3 GPIO_PIN4 GPIO_PIN5 GPIO_PIN6 GPIO_PIN7 GPIO_PIN8 GPIO_PIN9 GPIO_PIN10 GPIO_PIN11 GPIO_PIN12 GPIO_PIN13 GPIO_PIN14 GPIO_PIN15

表 3

	selectedPort	selectedPins	mode
GPIO_setAsPeripheralModuleFunctionOutputPin() GPIO_setAsPeripheralModuleFunctionInputPin()	GPIO_PORT_P1		
	GPIO_PORT_P2	GPIO_PIN0	
	GPIO_PORT_P3	GPIO_PIN1	
	GPIO_PORT_P4	GPIO_PIN2	
	GPIO_PORT_P5	GPIO_PIN3	
	GPIO_PORT_P6	GPIO_PIN4	
	GPIO_PORT_P7	GPIO_PIN5	
	GPIO_PORT_P8	GPIO_PIN6	GPIO_PRIMARY_MODULE_FUNCTION
	GPIO_PORT_P9	GPIO_PIN7	GPIO_SECONDARY_MODULE_FUNCTION
	GPIO_PORT_P10	GPIO_PIN8	GPIO_TERNARY_MODULE_FUNCTION
	GPIO_PORT_P11	GPIO_PIN9	
	GPIO_PORT_PA	GPIO_PIN10	
	GPIO_PORT_PB	GPIO_PIN11	
	GPIO_PORT_PC	GPIO_PIN12	
	GPIO_PORT_PD	GPIO_PIN13	
	GPIO_PORT_PE	GPIO_PIN14	
	GPIO_PORT_PF	GPIO_PIN15	
	GPIO_PORT_PJ		

2.4 问题提出

1．为什么关于中断边沿选择的函数 GPIO_interruptEdgeSelect()的参数 selectedPort 可以有 18 个量可以选择，而不是像其他 4 个关于中断设置的函数，只有 GPIO_PORT_P1、GPIO_PORT_P2、GPIO_PORT_PA 三个量可以选择？

2．在表 3 中给出的 selectedPort 所选的 18 个量，我用不同颜色做了标记，根据本章 2.1 引言部分介绍，PA 就是 P1 和 P2 的合，那么在位的对应关系上，是怎样对应的，是高位在前还是低位在前？请结合手册找出该对应关系。

3．结合例程，使用库函数编写程序：P1.1 配置为中断上升沿触发按钮，在 P2 上有 8 个 LED 灯，初始时候灯是间隔点亮的，记为 P2.0 为亮。当按键触发时候全部灭掉，第二次中断触发的时候，轮流点亮。第三次触发的时候全部点亮。如此按键，将会循环上面的操作（即:全灭->轮流跑马灯->全亮）。

4．谈谈你对本库函数的认识，什么情况下使用方便，什么情况下使用不方便。

本科内容主要学习了解基于 CCStudio6 的库函数开发过程，学会阅读库函数，以及了解 GPIO_相关的 13 个函数的功能和使用方法。另外请关注本帖，学习的同时跟帖与网友做好学习交流。