第4章 GPIO及程序框架

4.3 GPIO构件封装方法与驱动构件封装规范

4.4 利用构件方法控制小灯闪烁

4.5 工程文件组织框架与第一个C语言工程分析

4.3 GPIO构件封装方法与驱动构件封装规范

4.3 GPIO构件封装方法与规范——建议与必要性

建议：

- 按底层硬件操作功能封装构件，给出函数名与接口函数，以供实际编程使用。

必要性：

- 利用软件构件技术，可以有效提高编程人员的开发效率并且使得程序更加规范。
- 将底层驱动封装成构件，可以减少重复劳动，使应用程序开发者能够更多关注软件优化与稳定性。

4.3 GPIO构件封装方法与规范

移植

考虑使用与移植方便。要对构件的共性与个性进行分析，抽取出构件的属性和对外接口函数。

复用

使用同一芯片的应用系统，构件不更改，直接使用；同系列芯片的同功能底层驱动复用时，仅改动头文件；不同芯片的同功能底层驱动复用时，头文件与源程序文件改动尽可能少。

交流

方便不同驱动构件使用者相互之间的交流，减少因函数封装不同带来的麻烦。

4.3 GPIO构件封装方法与规范——封装规范概要

1.每个构件由头文件（.h）和源文件（.c）两个独立文件组成，放在以构件命名的文件夹中，驱动头文件（.h）中仅仅包含了对外接口函数的声明，相当于使用指南。

例如封装好的GPIO构件存放于放置底层驱动文件夹“05_Driver”下的“gpio”文件夹中，方便拷贝使用。

4.3 GPIO构件封装方法与规范——封装规范概要

2.需满足设计构件的最基本要求。

（1）考虑使用与移植方便。
（2）要有统一、规范的编码风格与注释。
（3）宏的使用限制。
（4）不使用全局变量。

4.3 GPIO构件封装方法与规范——以MSP432的GPIO为例

GPIO

01 有84个引脚可以作为GPIO，分布在11个端口

02 不可能使用直接地址去操作相关寄存器，那样无法实现软件移植与复用

03 应该把对GPIO引脚的操作封装成构件，通过函数调用与传参的方式实现对引脚的干预与状态获取

4.3 GPIO构件封装方法与规范——GPIO驱动构件封装要点

由于芯片引脚具有复用特性，应把引脚设置成GPIO功能；同时定义成输入或输出。

初始化

设置状态

对于输出，希望通过函数设置引脚是高电平（逻辑1）还是低电平（逻辑0）。

对于输入，希望通过函数获得引脚的状态是高电平（逻辑1）还是低电平（逻辑0）。

获得状态

状态反转

希望通过函数实现引脚的状态转换。

若引脚被设置成输入，还可以设定内部上下拉，MSP432内部上下拉电阻大小为20~50KΩ。

上下拉使能

4.3 GPIO构件封装方法与规范——gpio_init

```
//=========================================================================
//函数名称：gpio_init
//函数返回：无
//参数说明：port_pin：（端口号）|（引脚号）（如：（PTB_NUM）|（9） 表示为B口9号脚）
//          dir：引脚方向（0=输入，1=输出,可用引脚方向宏定义）
//          state：端口引脚初始状态（0=低电平，1=高电平）
//功能概要：初始化指定端口引脚作为GPIO引脚功能，并定义为输入或输出，若是输出，
//          还指定初始状态是低电平或高电平
//=========================================================================
void gpio_init(uint_16 port_pin, uint_8 dir, uint_8 state);
```

函数名称 所注释函数的名称

函数返回 用来说明函数返回变量的数据类型与含义

参数说明 用来说明入口参数的数据类型与含义

功能概要 简单说明该函数实现功能

4.3 GPIO构件封装方法与规范——GPIO驱动构件部分源码

```
void gpio_init(uint_16 port_pin, uint_8 dir, uint_8 state)
{
    //局部变量声明
    PORT_Type *port_ptr;                          //声明port_ptr为PORT结构体类型指针
    GPIO_Type *gpio_ptr;                          //声明gpio_ptr为GPIO结构体类型指针
    uint_8 port,pin;                              //声明端口port、引脚pin变量
    //根据带入参数port_pin，解析出端口与引脚分别赋给port,pin
    gpio_get_port_pin(port_pin,&port,&pin);
    //根据port，给局部变量port_ptr、gpio_ptr赋值（获得两个基地址）
    port_ptr = PORT_ARR[port];                    //获得PORT模块相应口基地址
    gpio_ptr = GPIO_ARR[port];                    //获得GPIO模块相应口基地址

    //设定相应端口的相应引脚功port能为GPIO（即令引脚控制寄存器的MUX=0b001）
    PORT_PCR_REG(port_ptr,pin) &= ~PORT_PCR_MUX_MASK;//置D10-D8=000
    PORT_PCR_REG(port_ptr,pin) |= PORT_PCR_MUX(1);//置D10-D8=001

    //根据带入参数dir，决定引脚为输出还是输入
    if(dir == 1)  //输出引脚
    {
        BSET(pin,GPIO_PDDR_REG(gpio_ptr));//数据方向寄存器的pin位=1，定义为输出
        gpio_set(port_pin,state);                 //调用gpio_set函数，设定引脚初始状态
    }
    else          //输入引脚
    {
        BCLR(pin,GPIO_PDDR_REG(gpio_ptr));//数据方向寄存器的pin位=0，定义为输入
    }
}
```

4.4 利用构件方法控制小灯闪烁

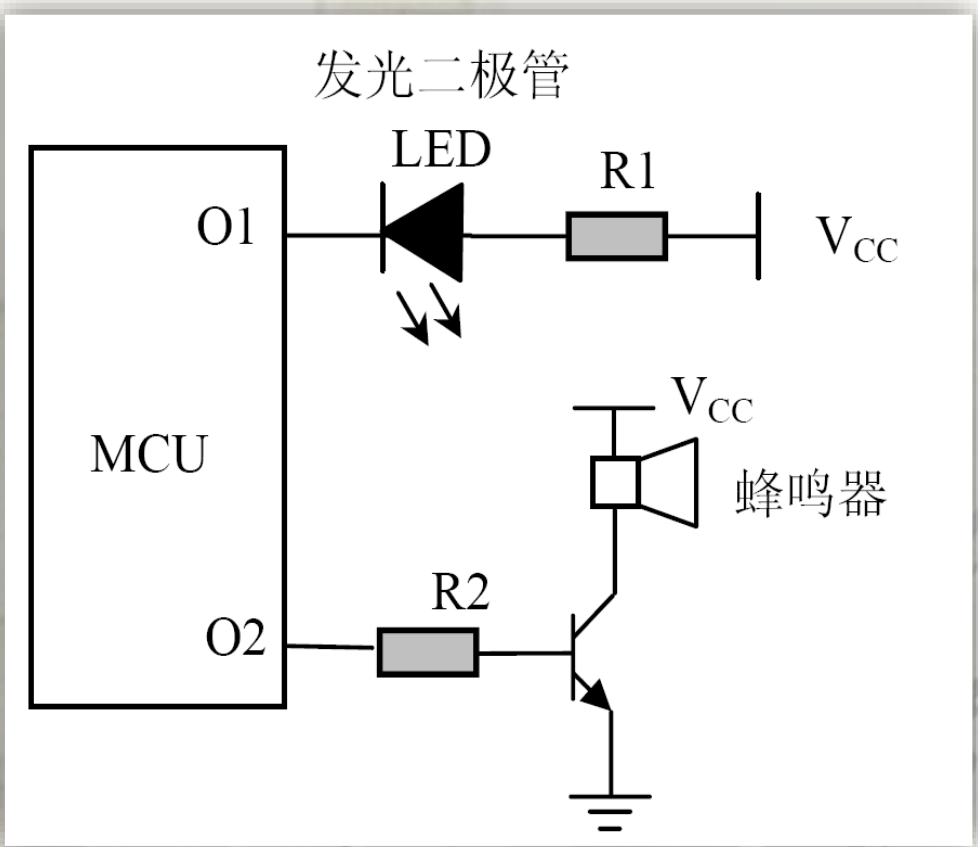
4.4 利用构件方法控制小灯闪烁

底层驱动设计
——GPIO

应用构件设计
——light

我们把调用芯片底层驱动构件设计的面向具体应用的构件，称为应用构件。

4.4 利用构件方法控制小灯闪烁



4.4 利用构件方法控制小灯闪烁

小灯驱动构件的函数

01 小灯初始化
light_init()

02 设置小灯状态
light_control()

03 翻转小灯状态
light_change()

4.4 利用构件方法控制小灯闪烁

小灯初始化 light_init()

```
//=====================================================================
//函数名称：light_init
//函数参数：port_pin：（端口号）|（引脚号）（如：（PTB_NUM）|（9） 表示为B口9号脚）
//        state：设定小灯状态。由light.h中宏定义。
//函数返回：无
//功能概要：指示灯驱动初始化。
//=====================================================================
void light_init(uint_16 port_pin, uint_8 state)
{
    gpio_init(port_pin, GPIO_OUTPUT, state);
}
```

4.4 利用构件方法控制小灯闪烁

设置小灯状态 light_control()

```
//=====================================================================
//函数名称：light_control
//函数参数：port_pin：（端口号）|（引脚号）（如：（PTB_NUM）|（9） 表示为B口9号脚）
//        state：设定小灯状态。由light.h中宏定义。
//函数返回：无
//功能概要：控制指示灯亮暗。
//=====================================================================
void light_control(uint_16 port_pin, uint_8 state)
{
    gpio_set(port_pin, state);
}
```

4.4 利用构件方法控制小灯闪烁

翻转小灯状态 light_change()

```
//=========================================================================
//函数名称：light_change
//函数参数：port_pin：（端口号)|(引脚号）（如：(PTB_NUM)|(9) 表示为B口9号脚）
//函数返回：无
//功能概要：切换指示灯亮暗。
//=========================================================================
void light_change(uint_16 port_pin)
{
    gpio_reverse(port_pin);
}
```

4.4 利用构件方法控制小灯闪烁——编程步骤

1.在light.h文件中给小灯起名字，并明确与MCU连接引脚，进行宏定义

```
//指示灯端口及引脚定义
#define  LIGHT_RED      (PTA_NUM|5)   //红色RUN灯使用的端口号/引脚
#define  LIGHT_GREEN    (PTA_NUM|12)   //绿色RUN灯使用的端口号/引脚
#define  LIGHT_BLUE     (PTA_NUM|13)   //蓝色RUN灯使用的端口号/引脚
```

2.在light.h头文件中对小灯亮、暗进行宏定义，方便编程

```
//灯状态宏定义（灯亮、灯暗对应的物理电平由硬件接法决定）
#define  LIGHT_ON        0     //灯亮
#define  LIGHT_OFF       1     //灯暗
```

4.4 利用构件方法控制小灯闪烁——编程步骤

3.在main函数中初始化LED灯的初始状态

```
//绿灯初始化
light_init(LIGHT_GREEN,LIGHT_ON);
```

4.在功能函数中控制小灯状态发生变化

```
//绿灯暗
light_control(LIGHT_GREEN,LIGHT_OFF);
```

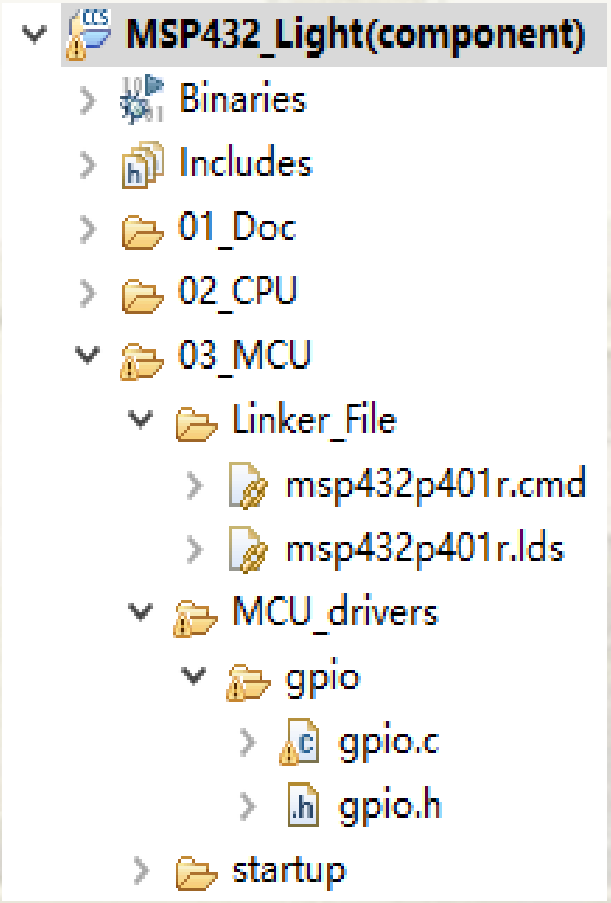
4.5 工程文件组织框架与第一个C语言工程分析

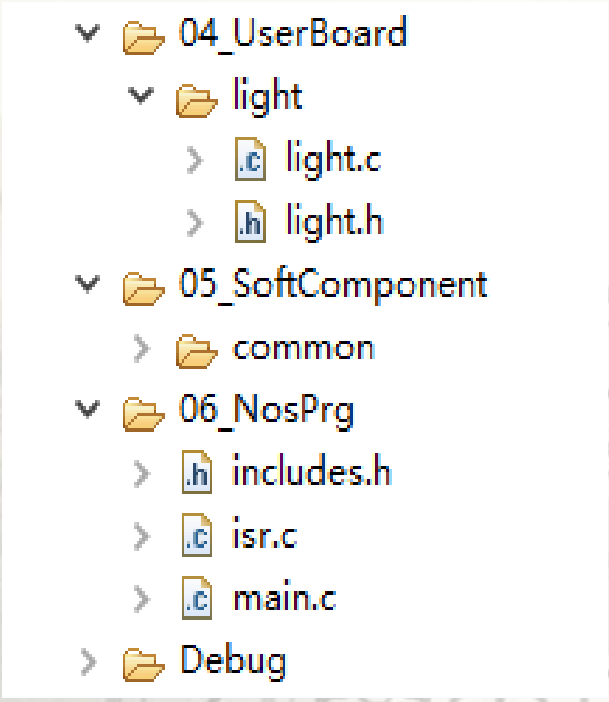
4.5 工程文件组织框架

- 嵌入式系统工程包含若干文件，包括程序文件、头文件、与编译调试相关的文件、工程说明文件、开发环境生成文件等
- 合理组织这些文件，规范工程组织，可以提高项目的开发效率、提高阅读清晰度、提高可维护性、降低维护难度。
- 这个工程框架也可被称为软件最小系统框架，因为它包含的工程的最基本要素。
- 软件最小系统框架是一个能够点亮一个发光二管的，甚至带有串口调试构件的，包含工程规范完整要素的可移植与可复用的工程模板。

4.5 工程文件组织框架——树形工程结构模板

树形工程结构模板





4.5 工程文件组织框架

名称	文件夹		简明功能及特点
文档文件夹	01_Doc		工程改动时，及时记录。
CPU文件夹	02_CPU		与内核相关的文件。
MCU文件夹	03_MCU	Linker_File	链接文件夹，存放链接文件。
		MCU_drivers	MCU底层构件文件夹，存放芯片级硬件驱动。
		startup	启动文件夹。存放芯片头文件及芯片初始化文件
用户板文件夹	04_UserBoard		用户板文件夹，存放应用级硬件驱动，即应用构件。
软件构件文件夹	05_SoftComponent		抽象软件构件文件夹，存放硬件不直接相关的软件构件。
无操作系统源程序文件夹	06_NosPrg		为了便于过渡到实时操作系统RTOS工程结构，特命名该文件夹。含主程序文件、中断服务例程文件等。这些文件是实际应用级开发人员进行编程的主要对象。

4.5 工程文件组织框架——main函数之前的执行过程

1 CPU(内核)

CPU（内核）相关文件（core_cm4.h、core_cmFunc.h、core_cmInstr.h 、 core_cmSimd.h ）位于工程框架的“..\02_CPU”文件夹内，的内核相关头文件

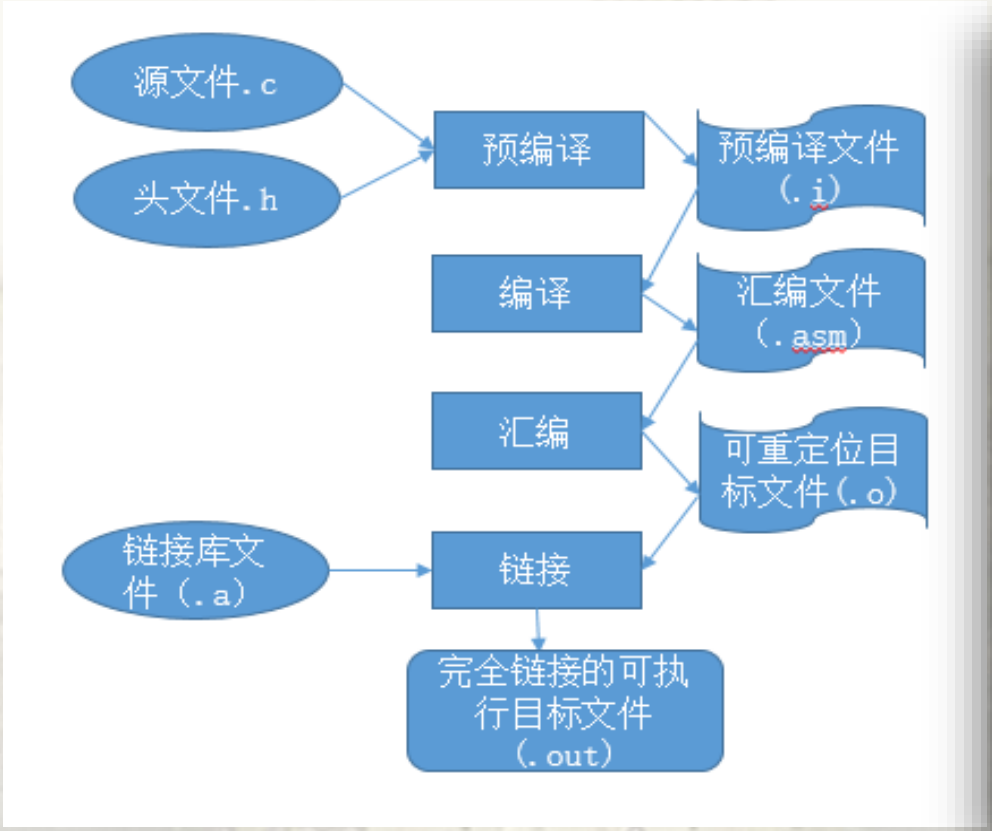
2 MCU（芯片）

MCU文件夹（芯片）相关文件（msp432p401r.h、startup_msp432p401r_ccs.h、system_msp432p401r.h 、system_ msp432p401r.c、msp_compatibility.h、等）位于工程框架的“..\03_MCU\ startup”文件夹内。由芯片厂商提供。

3 应用程序源代码文件

在工程框架的“..\ 06_NosPrg”文件夹内放置着总头文件includes.h、main.c及中断服务例程文件isr.c。

4.5 工程文件组织框架——main函数之前的执行过程



从源代码到最后的可执行文件可以认为需要经过编译、汇编和链接三个过程。每个源代码文件在编译和汇编后都会生成一个可重定位的目标文件，链接器可以将这些中间文件组合成最终的可执行目标文件。

4.5 工程文件组织框架——main函数之前的执行过程

1. 查询保存在Flash存储区首端的中断向量表，取出第一个表项的内容设定为堆栈初始化指针
2. 取出第二个表项的内容作为启动函数（__STACK_END，位于startup_msp432p401r_ccs.c）的入口地址
3. 在__STACK_END函数中，将通用寄存器清零，之后调用Reset_Handler函数（位于startup_msp432p401r_ccs.c）
4. 在sysinit函数中主要完成芯片时钟系统的配置。main函数（位于main.c）为开发人员自定义的执行程序函数

谢 谢!