嵌入式实践与基础手札（2）

## 4.3 GPIO驱动构件封装方法与驱动构件封装规范

### 4.3.1 设计GPIO 驱动构件的必要性及GPIO 驱动构件封装要点分析

#### 1.设计GPIO驱动构件的必要性

软件构件(Software Component) 技术的出现，为实现软件构件的工业化生产提供了理论与技术基石。将软件构件技术应用到嵌入式软件开发中，可以大大提高嵌入式开发的开发效率与稳定性。软件构件的封装性、可移植性与可复用性是软件构件的基本特性，采用构件技术设计软件，可以使软件具有更好地开放性、通用性和适应性。

以KL25 的GPIO 为例， 它有61 个引脚可以作为GPIO, 分布在5 个端口，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。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，这样的软件才便于维护与移植，因此设计GPIO 驱动构件十分必要。同时，底层驱动构件的封装，也为在操作系统下对底层硬件的操作提供了基础。

#### 2. GPIO 驱动构件封装要点分析

同样以GPIO 驱动构件为例，进行封装要点分析。即分析应该设计哪几个函数及入口参数。GPIO 引脚可以被定义成输入、输出两种情况：若是输入，程序需要获得引脚的状态（逻辑1 或O); 若是输出，程序可以设置引脚状态（逻辑1 或0) 。MCU 的PORT 模块分为许多端口，每个端口有若于引脚。GPIO 驱动构件可以实现对所有GPIO 引脚统一编程。GPIO 驱动构件由gpio. h 、gpio. C 两个文件组成，如要使用GPIO 驱动构件，只需要将两个文件加入到所建工程中，由此方便了对GPIO 的编程操作。

##### 1）模块初始化（gpio\_init）

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；若是输出，还要给出初始状态。所以GPIO 模块初始化函数gpio\_init 的参数为哪个引脚、是输人还是输出、若是输出其状态是什么，函数不必有返回值。其中，引脚可用一个16 位数据描述，高8 位表示端口号，低8 位表示端口内的引脚号

void gpio\_init(uint\_16 port\_pin,uint\_8 dir,uint\_8 state)

##### 2) 设置引脚状态(gpio\_set)

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) 。人口参数应该是哪个引脚，输出其状态是什么，函数不必有返回值。这样设置引脚状态的函数原型可以设计为：

void gpio\_set(uint\_16 port\_pin, uint\_8 state)

##### 3) 获得引脚状态(gpio\_get)

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), 入口参数应该是哪个引脚，函数需要返回值引脚状态。这样设置引脚状态的函数原型可以设计为：

uint\_8 gpio\_get(uint\_16 port\_pin)

##### 4 ) 引脚状态反转(void gpio\_ reve rse)

类似的分析，可以设计引脚状态反转函数的原型为：void gpio\_reverse(uint\_16 port\_pin)

##### 5 ) 引脚上下拉使能函数(void gpio\_pull)

若引脚被设置成输入， 还可以设定内部上下拉， KL2 5 内部上下拉电阻大小为20 ~50 kΩ 。引脚上下拉使能函数的原型为：void g,pio\_pull(uint\_16 port\_pin, uint\_8 pullselect)

### 4.3.2 底层驱动构件封装规范概要与构件封装的前期准备

底层驱动构件封装规范见5. 3 节，本节给出概要与前期准备，以便读者在认识第一个构件前以及在开始设计构件时，少走弯路，做出来的构件符合基本规范，便千移植、复用、交流。

#### 1. 底层驱动构件封装规范概要

##### 1) 底层驱动构件的组成、存放位置与内容

每个构件由头文件(. h) 与源文件(. c) 两个独立文件组成，放在以构件名命名的文件夹中。驱动构件头文件(. h ) 中仅包含对外接口函数的声明，是构件的使用指南. 以构件名命名。

例如， GIPO 构件命名为gpio( 使用小写，目的是与内部函数名前缀统一）。设计好的GPIO 构件存放于".. KL25 共用驱动\KL25 底层驱动构件\gpio" 文件夹中，供复制使用。基本要求是调用者只看头文件即可使用构件。对外接口函数及内部函数的实现在构件源程序文件(. c) 中。同时应注意，头文件声明对外接口函数的顺序与源程序文件实现对外接口函数的顺序应保持一致。

##### 2) 设计构件的最基本要求

这里摘要给出设计构件的最基本要求。

(1) 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。

(2) 要有统一、规范的编码风格与注释。主要涉及文件、函数、变械、宏及结构体类型的命名规范；涉及空格与空行、缩进、断行等的排版规范；涉及文件头、函数头、行及边等的注释规范。

(3) 宏的使用限制。宏的使用具有两面性， 有提高可维护性一面，也有降低阅读性一面，因此不要随意使用宏。

(4) 不使用全局变量。构件封装时，禁止使用全局变量。

## 4.4 实验：利用构件方法控制小灯闪烁

实例解释：

定义文件 key.h key.c 分别代表 驱动构件的头文件和源文件，下面解释

#### 1)key.h

#ifndef \_LIGHT\_H //防止重复定义（\_LIGHT\_H 开头)

#define \_LIGHT\_H

//头文件包含

#include "common.h" //包含公共要素头文件

#include "gpio.h" //用到gpio构件

**//指示灯端口及引脚定义**

#define LIGHT\_RED (PTB\_NUM|19) //红色RUN灯使用的端口号/引脚

#define LIGHT\_BLUE (PTB\_NUM|9) //蓝色RUN灯使用的端口号/引脚

#define LIGHT\_GREEN (PTB\_NUM|18) //绿色RUN灯使用的端口号/引脚

**//定义接受高低电平的输入引脚**

#define key1 (PTB\_NUM|0)

#define key2 (PTB\_NUM|2)

#define key3 (PTA\_NUM|2)

#define key4 (PTA\_NUM|5)

//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

//=================接口函数声明=======================================

//==========================================================

//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);

//==========================================================

//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);

//==========================================================

//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);

**//定义输入引脚key初始化函数，需要有返回值,定义为输入引脚**

void key\_init(uint\_16 port\_pin);

uint\_8 key\_get(uint\_16 port\_pin);//获取输入引脚的输入信号

#endif //防止重复定义（\_LIGHT\_H 结尾)

#### 2)key.c

//=====================================================================

//文件名称：key.c

//功能概要：小灯构件源文件

//=====================================================================

#include "key.h"

//=====================================================================

//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);

}

//=====================================================================

//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);

}

//==========================================================

//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);

}

void key\_init(uint\_16 port\_pin)

{

gpio\_init(port\_pin,0,1);//定义为输入引脚

}

uint\_8 key\_get(uint\_16 port\_pin)

{

return gpio\_get(port\_pin);//获得输入引脚的输入

}

#### 3)源文件main.c

注：通过B0,B2两个输入引脚分别获取输入的状态，制作两个呼吸灯，分别对应两种不同的高低组合来实现

源文件：

//说明见工程文件夹下的Doc文件夹内Readme.txt文件

//======================================================================

#include "includes.h" //包含总头文件

int main(void) {

//（1） 声明主函数使用的变量

uint\_32 mRuncount; //主循环计数器

uint\_8 mflag;

uint\_8 k1;

uint\_8 k2;//B0,B2

uint\_8 k3;

uint\_8 k4;

//定义四个状态变量负责返回输入引脚接受的输入

//（2） 关总中断

DISABLE\_INTERRUPTS;

//（3） 给有关变量赋初值

mRuncount = 0; //主循环计数器

mflag = 0; //灯控制标志

//（4） 初始化外设模块

light\_init(LIGHT\_RED, LIGHT\_OFF); //红灯初始化

light\_init(LIGHT\_BLUE, LIGHT\_OFF); //蓝灯初始化

light\_init(LIGHT\_GREEN, LIGHT\_OFF); //绿灯初始化

//初始化开关为输入引脚

key\_init(key1);

key\_init(key2);

//（5） 使能模块中断

//（6） 开总中断

ENABLE\_INTERRUPTS;

//进入主循环

//主循环开始==================================================================

for (;;) {

k1 = key\_get(key1);

k2 = key\_get(key2);

if ((k1 == 1) && (k2 == 0)) {

for (;;) {

mRuncount++; //主循环次数计数器+1

if (mRuncount >= RUN\_COUNTER\_MAX) //主循环次数计数器大于设定的宏常数

{

mRuncount = 0;

k1 = key\_get(key1);

k2 = key\_get(key2);

if ((k1 == 0) && (k2 == 1))

break;

switch (mflag) {

case 0: {

light\_change(LIGHT\_RED);

light\_control(LIGHT\_BLUE, LIGHT\_OFF);

light\_control(LIGHT\_GREEN, LIGHT\_OFF);

mflag = 1;

break;

}

case 1: {

light\_change(LIGHT\_BLUE);

light\_control(LIGHT\_RED, LIGHT\_OFF);

light\_control(LIGHT\_GREEN, LIGHT\_OFF);

mflag = 0;

break;

}

default: {

}

}

}

}

} else if ((k1 == 0) && (k2 == 1)) {

for (;;) {

mRuncount++; //主循环次数计数器+1

if (mRuncount >= RUN\_COUNTER\_MAX) //主循环次数计数器大于设定的宏常数

{

mRuncount = 0;

k1 = key\_get(key1);

k2 = key\_get(key2);

if ((k1 == 1) && (k2 == 0))

break;

switch (mflag) {

case 0: {

light\_change(LIGHT\_GREEN);

light\_control(LIGHT\_BLUE, LIGHT\_OFF);

light\_control(LIGHT\_RED, LIGHT\_OFF);

mflag = 1;

break;

}

case 1: {

light\_change(LIGHT\_BLUE);

light\_control(LIGHT\_RED, LIGHT\_OFF);

light\_control(LIGHT\_GREEN, LIGHT\_OFF);

mflag = 0;

break;

}

default: {

}

}

}

}

} else {

}

}

//以下加入用户程序--------------------------------------------------------

} //主循环end\_for

//主循环结束==================================================================

#### 4) 实验分析

这次实验通过利用GPIO构件写了light.h 组件 控制小灯的亮暗

在主函数main.c中通过初始化两个引脚作为开关输入，设定了两对不同的输入状态作为开关，通过选择语句控制不同的输入状态对应不同的流水灯。

# ch 6串口通信模块

## 6.1 异步串行通信的通用基础知识

### 6.1.1 串行通信的基本概念

在计算机中，通常一个信息单位用8 位二进制表示，称为一个” 字节"(Byte) 。串行通信的特点是：数据以字节为单位，按位的顺序（例如最高位优先）从一条传输线上发送出去。

1.异步串行通信的格式

格式的空闲状态为"l"' 发送器通过发送一个" O"表示一个字节传输的开始，随后是数据位（在MCU 中一般是8 位或9 位，可以包含校验位） 。最后，发送器发送1 位或2 位的停止位，表示一个字节传送结束。若继续发送下一字节，则重新发送开始位（这就是异步的含义了），开始一个新的字节传送。若不发送新的字节， 则维持"l" 的状态，使发送数据线处千空闲。从开始位到停止位结束的时间间隔称为一字节帧(Byte Frame) 。

2.串行通信的波特率

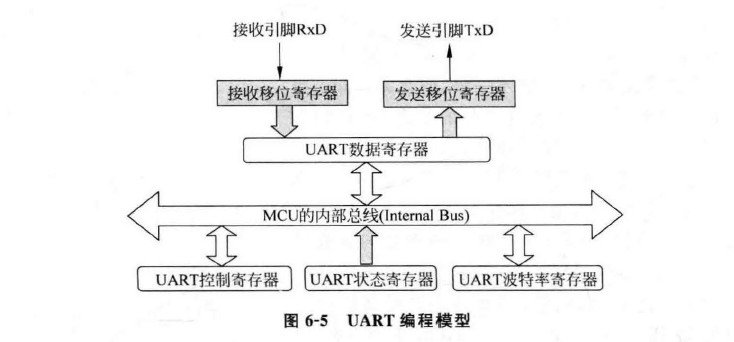
位长(Bit Length), 也称为位的持续时间(Bit Duration), 其倒数就是单位时间内传送的位数。人们把每秒内传送的位数叫作波特率(Baud Rate) 。波特率的单位是：位／ 秒，记为bps

3.奇偶校验

字符奇偶校验检查(Character~a rity Chec king) 称为垂直冗余检查( Vertical Redundancy Checking, VRC), 它是为每个字符增加一个额外位使字符中"1" 的个数为奇数或偶数

### 6.1.4 串行通信编程模型

从基本原理角度看，串行通信接口UART 的主要功能是： 接收时，把外部的单线输入的数据变成一个字节的并行数据送入MCU 内部；发送时，把需要发送的一个字节的并行数据转换为单线输出编程时，程序员并不直接与“发送移位寄存器”和“接收移位寄存器“打交道，只与数据寄存器打交道，所以MCU 中并没有设置”发送移位寄存器”和“接收移位寄存器”的映像地址。发送时，程序员通过判定状态寄存器的相应位，了解是否可以发送一个新的数据。若可以发送， 则将待发送的数据放入"UA RT 数据寄存器”中就可以了，剩下的工作由MCU 自动完成：将数据从"UART 数据寄存器”送到“发送移位寄存器＂，硬件驱动将“发送移位寄存器” 的数据一位一位地按照规定的波特率移到发送引脚TxD, 供对方接收。接收时，数据一位一位地从接收引脚RxD 进入“接收移位寄存器”，当收到一个完整字节时， MCU 会自动将数据送入"UART 数据寄存器”，并将状态寄存器的相应位改变，供程序员判定并取出数据。



## 6.2 KL25 /26 芯片UART 驱动构件及使用方法

### 6.2.2 UART 驱动构件基本要素分析与头文件

UART 驱动构件由头文件uart. h 及源代码文件uart. c 组成，放入uart 文件夹中，供应用程序开发调用。UART 具有初始化、发送和接收三种基本操作

UART 驱动构件的基本函数，与寄存器直接打交道的有：初始化、发送单个字节与接收单个字节的函数，以及使能及禁止接收中断、获取接收中断状态的函数

串口驱动构件可封装下列9 个基本功能函数

1）串口初始化

2）发送单个字节

3）发送字符串

4）接受单个字节

5）接受N个字节

6）使能串口接收中断

7）禁止串口接收中断

8）获取接收中断状态

9）发送N个字节

## 实验：通过初始化UART通信串口，完成通信

#### 1.实验准备：KL25实验板 KDS开发环境

#### 2.实验目的：

通过改写UART构件的源码来实现串口输入字符改变灯亮的颜色

#### 3.实验源码：

需要的构件一个是控制灯亮的LIGHT.H的GPIO构件，用来控制小灯的亮灭

另一个则是在isr.c中改写串口通信源码，完成串口通信的要求（其中控制小灯亮灭仍然采用上次实验的控件light）

下面给出isr.h中的源码

##### isr.h

**void** **UART2\_IRQHandler**(**void**) {

light\_init(LIGHT\_RED, LIGHT\_OFF); //红灯初始化

light\_init(LIGHT\_BLUE, LIGHT\_OFF); //蓝灯初始化

light\_init(LIGHT\_GREEN, LIGHT\_OFF);

uint\_8 ch;

uint\_8 flag;

DISABLE\_INTERRUPTS; //关总中断

**if** (uart\_get\_re\_int(UART\_2)) {

ch = uart\_re1(UART\_2, &flag); //调用接收一个字节的函数，清接收中断位

**if** (flag) {

uart\_send1(UART\_2, ch);

**switch** (ch) {

**case** 'b': {

light\_change(LIGHT\_BLUE);

light\_control(LIGHT\_RED, LIGHT\_OFF);

light\_control(LIGHT\_GREEN, LIGHT\_OFF);

uart\_send\_string(UART\_2, "blue\_UART\_2!\r\n");

**break**; //向原串口发回一个字节

}

**case** 'r': {

light\_change(LIGHT\_RED);

light\_control(LIGHT\_BLUE, LIGHT\_OFF);

light\_control(LIGHT\_GREEN, LIGHT\_OFF);

uart\_send\_string(UART\_2, "red\_UART\_2!\r\n");

**break**;

}

**case** 'g': {

light\_change(LIGHT\_GREEN);

light\_control(LIGHT\_BLUE, LIGHT\_OFF);

light\_control(LIGHT\_RED, LIGHT\_OFF);

uart\_send\_string(UART\_2, "green\_UART\_2!\r\n");

**break**;

}

**default**: {

}

}

}

}

ENABLE\_INTERRUPTS;

}

PS:这里是改写串口2的源码，完成输入字符 ‘b g r’ 控制灯的亮暗；

#### 4.实验分析:

这次试验只是简单的改写串口通信构件，完成输入字符控制灯的亮暗。通过这次实验，初步的认识了UART构件的作用以及怎么样使用该构件完成串口通信。但是对进一步的深层次使用还是存在很多的生疏，需要花很多时间来熟悉使用