# 实验一 认识ARM开发板

## 实验目的

1. 对单片机硬件形成直接观察；
2. 掌握连接方法；
3. 通过对单片机型号的观察，通过多种途径查找相应驱动程序，并熟练掌握单片机驱动程序的安装。

## 背景知识

单片机又称单片微控制器,它不是完成某一个逻辑功能的芯片,而是把一个计算机系统集成到一个芯片上。相当于一个微型的计算机，和计算机相比，单片机只缺少了I/O设备。概括的讲：一块芯片就成了一台计算机。它的体积小、质量轻、价格便宜、为学习、应用和开发提供了便利条件。同时，学习使用单片机是了解计算机原理与结构的最佳选择。

一般而言，专门开发用于学习用途的单片机又叫学习板，或实验板。从外形上看，不同的实验板各有形状，但一般都集成有内核、数码管、数字按键、电源、串口、LED灯、启动按钮与调试接口等，也可以通过USB或并口外接液晶屏。图1中可以见到芯片的型号是STC89C51RC，主频晶振为12.000MHz。

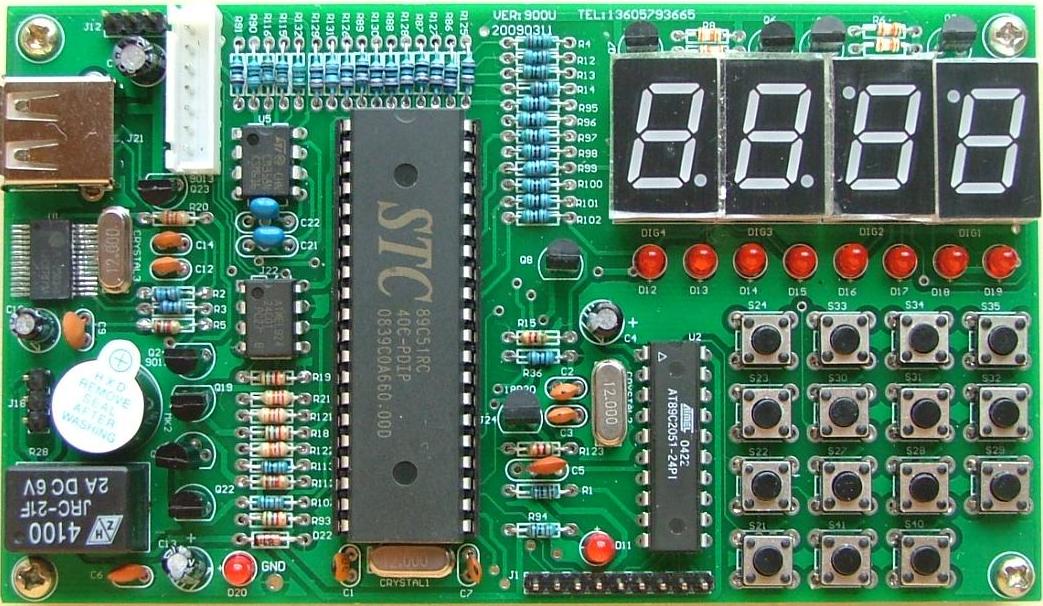


图1 开发板外形

本次嵌入式系统课程设计的开发主要面向单片机或开发板进行编程。在Keil μVersion4集成开发环境下，语言可以选择C或者汇编，硬件上将采用意法半导体（中国）投资有限公司的STM32f10xx实验板。

要在实验板上进行各种实验，需要把实验板和电脑连接起来。我们实验室中的宿主机系统采用的是Window XP，要连接实验板与电脑，首先需要通过串口或者USB口用连线进行硬件连接，然后再安装驱动程序，在操作系统中进行“注册”以后才能够被使用。下面的实验分别就这些工作一一进行。

## 实验内容和步骤

1. 观察实验板外形

根据提供的实验板，仔细观察，依次找出芯片及其型号、串口、并口、启动按钮等各部件的位置与特征，为硬件连接做好准备。

1. 与电脑硬件连接

步骤1：准备好连接线。由于现在个人电脑上多数配置了一个或多个USB接口，而串口连接不如USB方便，因此需要准备好USB转串口的连线。当然也可以直接使用串口连接。



图3 串行接口

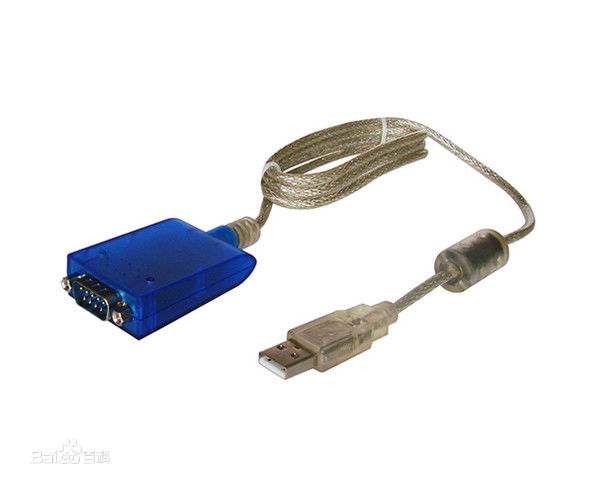


图2 USB转串口连线

步骤2：用合适数据线将实验板和计算机连接起来。

步骤3：连接好电源线，打开电源开关。STM32f10xx系列芯片的工作电压是3.3v，打开电源开关前请确认电源适配器的工作电压。

注意：在进行硬件操作的时候，切忌打开电源开关！

1. 安装驱动程序

步骤1：根据芯片的型号，在网上查找合适的驱动程序，并进行下载，也可以找指导老师索取驱动。

步骤2：选择驱动、系统版本，32位或64位。

步骤3：双击驱动xxxx.exe文件。

步骤4：安装完成后进入计算机的设备管理器，就会发现多了一个COM端口。

# 实验二 流水灯

## 实验目的

进一步熟悉keil开发环境和实验板的使用。了解并熟悉实验板上单片机I/O口和LED灯的电路结构，掌握STM32单片机I/O口编程方法，掌握顺序控制程序的简单编程。

## 背景知识

1. 原理图

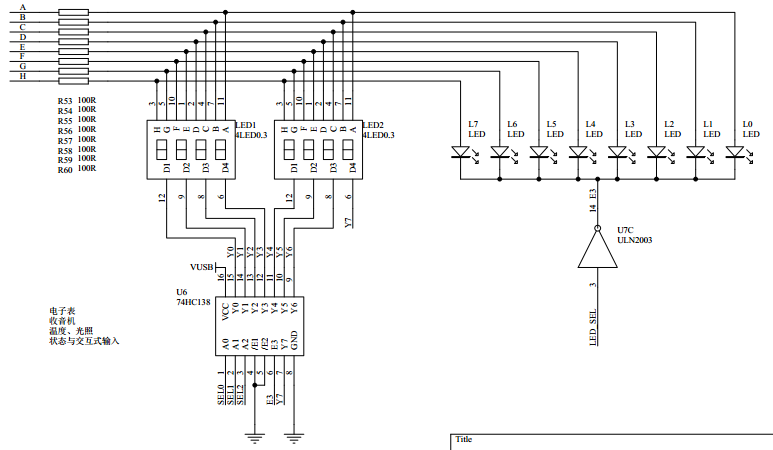


图4 流水灯电路图

1. 基本原理

I/O口是单片机与外界联系的通道，它可对各类外部信号（模拟量、开关量、频率信号）进行检测、判断和处理，并可控制各类外部设备。单片机通过I/O与外部世界互相感知。

在图4中的电路中，有L0,L1,L2,L3,L4,L5,L6,L7共八个发光二极管，当引脚LED\_SEL输入为1，对于A、B、C、D、E、F、G、H引脚，只要输入为1，则点亮相连接的发光二极管。

A~H引脚连接STM32F108VB芯片的PE8~PE15，程序初始化时，对其进行初始设置。引脚LED\_SEL为1时，发光二极管才工作，否则右边的数码管工作。注意，LED\_SEL连接于PB3，该引脚具有复用功能，在默认状态下，该引脚的IO不可用，需对AFIO\_MAPR寄存器进行设置，设置其为IO可用。

1. STM32微处理器启动文件Stm32f10x.s

当前的嵌入式应用程序开发过程里，C语言成为了绝大部分场合的最佳选择。如此一来main函数似乎成为了理所当然的起点，因为C程序往往从main函数开始执行。但这个在通用计算机上不成问题的事情在单片机上却变得很麻烦： 微控制器（单片机）上电后，由于没有我们熟悉的完备的操作系统，MCU如何才能找到并执行main函数？很显然微控制器无法从硬件上定位main函数的入口地址，因为使用C语言作为开发语言后，变量/函数的地址就由编译器在编译时自行分配，这样一来main函数的入口地址在微控制器的内部存储空间中不再是绝对不变的，而变得“浮动”起来。要解决这个问题，就必须对程序入口进行定位。因此，必须引入类似的启动文件（Bootloader），在单片机加电启动时定位入口。

不论性能高校，结构简繁，抑或是价格贵贱，每一种微控制器（处理器）都必须有启动文件。启动文件的作用就是负责执行微控制器从“复位”到“开始执行main函数”中间这段时间（称为启动过程）所必须进行的工作。最为常见的51、AVR或STM32等微控制器当然也有对应的启动文件，但开发环境往往自动完整地提供了这个启动文件，不需要开发人员再行干预启动过程，只需要从main函数开始进行应用程序的设计即可。

对于STM32微控制器，ST公司在keil uvision4中提供了现成的直接可用的启动文件，程序开发人员可以直接引用启动文件后直接进行C应用程序的开发，这样能大大减少开发人员从其他微控制器平台跳转到STM32平台，也降低了适应STM32微控制器的难度。这相对于上一代ARM的代表作ARM9，启动文件简直就是一道程序人员必须翻过去的难关。

相对于ARM7或ARM9内核架构，Cortex M3的启动方式有了比较大的变化。ARM7/ARM9内核的控制器在复位后，CPU会从存储空间的绝对地址0x000000取出第一条指令执行复位中断服务程序的方式启动，即固定了复位后的起始地址为0x000000（PC=0x000000），同时中断向量表的位置并不是固定的。而Cortex-M3的情况正好相反，有3种情况：

1. 通过boot引脚设置可以将中断向量表定位与SRAM区，即起始地址为0x200000000，同时复位后PC指针指向0x20000000处；
2. 通过boot引脚设置可以将中断向量表定位与FLASH区，即起始地址为0x80000000，同时复位后PC指针指向0x8000000处；
3. 通过boot引脚设置可以将中断向量表定位于内置Bootloader区，这里不对此进行论述。

Cortex-M3内核规定，起始地址必须存放堆顶指针，而第二个地址则必须存放复位中断入口向量地址，这样在Cortex-M3内核复位后，会自动从起始地址的下一个32位空间取出复位中断入口向量，跳转执行复位中断服务程序。这样，M3内核通过固定中断向量表的位置，而使得起始地址可以变化。

在STM32平台的固件库中，提供了一个名为stm32f10x.s的启动文件。在该文件中定义了STM32的堆栈大小以及各种中断的名字及入口函数名称，还有启动相关的汇编代码。STM32f10x.s是MDK提供的启动代码，从其内容来看，它只定义了3个串口、4个定时器。实际上STM32的系列产品有5个串口的型号，但即使是采用的STM32F103ZET6，而启动文件用的是STM32F10x.s的话，也只能正常使用串口1～3的中断，而串口4和5的中断则无法正常使用，定时器也只能使用TIM1~4，而TIM5～8同样无法使用。

表1： 启动文件的选择

|  |  |
| --- | --- |
| 文 件 名 | 适 用 产 品 |
| Stm32f10x\_ld.s | 小容量：FLASH <= 32K |
| Stm32f10x\_ld.s | 中容量：64K <= FLASH <= 128K |
| Stm32f10x\_ld.s | 大容量：256K <= FLASH |

因此STM32F10x.s并不能适用所有的STM32型号，通常只能针对不同型号的STM32处理器选择不同的启动文件。ST公司还另外提供了3个启动文件，分别适用于不同容量的STM32芯片。

在后面的各个实验中均将使用到启动文件，在Keil安装文件夹下进行搜索即可得到很多个，直接拷贝出来即可。在Keil uvision4下还可以搜索出stm32f10x\_vector.s和stm32f10xOPT.s两个文件，感兴趣的同学可以自行查找其中的区别。

## 实验内容和步骤

1. 实验准备工作

步骤1：建立文件夹。在D盘下建立自己个人文件夹xxx，建议所有文件夹名称全部用英文字母打头，并在xxx下建立文件夹led（绝对路径为d:\xxx\led）

步骤2：在keil安装目录中查找下列文件，并拷贝到刚建的led文件夹下来。

stm32f10x.s stm32f10x\_it.h stm32f10x\_conf.h cortexm3\_macro.h

stm32f10x\_map.h stm32f10x\_type.h stm32f10x\_nvic.h

实际上可以在搜索时使用通配符“stm32f10x\*.\*，一次全部找到。

1. 建立keil项目

步骤1: 启动keil，选择Project | new uvision Project，输入项目名称“led”，后缀不用写，系统自动追加为“.uvproj”。选择存储的文件夹为d:\xxx\led，然后单击OK进入下个界面；

步骤2：芯片选择。选择芯片为STMicroelectronics公司的STM32F103VB，点击OK完成项目建立。

1. 项目设置

步骤1：Target设置。点击Target Option按钮，进入Target页，选择晶振频率为8MHz，Read/Only Memory区的IROM1（即板上FLASH区）的起始地址为0x08000000，大小为0x20000（这个内存尺寸对本程序都太大了），Read/Write Memory区的起始地址为0x20000000，大小为0x5000；

步骤2：输出设置。点击进入Output页，点选“Create HEX File”要求生成.HEX文件；

步骤3：连接设置。点击进入Linker页，勾选“Use Memory Layout From Target Dialog”。至此设置完成。

1. 编写代码

步骤1：点击“File | new…”，生成一个空白文件，选择“File | save”保存为d:\xxx\led\led.c。下面所有的.c文件都如此操作，不再赘述。这是一个存储与硬件相关函数的文件，其中包含了二极管与数码管的初始化和实现。

步骤2：用同样方法建立与led.c相关的led.h，只是在保存时加上后缀名.h，同样保存到d:\xxx\led\文件夹下；

步骤3：依次建立delay.c，led.c，sys.c以及相应的.h文件，统一保存到d:\xxx\led\文件夹下；

步骤4：建立main.c并保存。

1. 加入文件到工程中。

步骤1：按“+”展开project窗口中的Target 1，出现Source Group 1，右击之，在弹出的菜单中选择“Add Files to Group ‘Source Group 1’”，控制文件对话框进入d:\xxx\led，选择stm32f10x.s，双击该文件或选择后单击“OK”按钮，即将stm32f10x.s这个启动文件加入到项目中去。

步骤2：用同样的方法依次吧led.c，delay.c，sys.c，usart.c，main.c加入到项目中，然后点击“Close”按钮关闭对话框。

步骤3：点击“Save all”保存所有文件。建议所有文件更改后立即保存，以免丢失修改。

注：1. Project窗口中的Target 1、Source Group 1等名称均可以更改，建议使用更有意义的英文名字；

2.考虑到良好的项目组织问题，可以右击“Target 1”选择“Add Group”增加组，将.c文件按类别分门别类的放好，组名也需要取有意义的名字。这里建议设置三个组：Hardware，system和user，分别对应硬件led.c，而与系统相关的延迟函数delay.c，NVIC的sys.c以及与串口相关的usart.c可以放到system组，最后main.c则归到user组中去，这样使工程组织显得更有条理，对于大的工程这种良好的习惯往往很有价值。

3.硬盘中的文件也可类似存放，以方便查找，提供效率。

1. 编译调试

步骤1：单击按钮栏“Build”或“Rebuild”，对整个工程进行编译连接，观察文件夹下增加了哪些文件；如果出现了错误，则在下方的Command窗口中按照错误信息进行分析；

步骤2：如果编译连接顺利通过，则可以在按钮栏“Start/Stop debug session”进行调试，调试方法与VC大致相同。可以单步、设置断点、观察寄存器以及存储器状态等等。

1. 烧录运行

步骤1：顺利编译连接之后，在按钮栏上选择“Download”按钮将可运行文件上载到开发板上；

步骤2：在开发板上运行程序并观察LED等的变化情况。

1. 记录实验结果并分析

## 程序清单

1. Led.h

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*二极管及数码管头文件\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef \_\_LED\_H

#define \_\_LED\_H

#include "sys.h"

//LED端口定义

#define LED\_SEL PBout(3) //PB3 二极管灯？数码管

//位选

#define SEL0 PBout(0)

#define SEL1 PBout(1)

#define SEL2 PBout(2)

//段选

#define LED0 PEout(8)

#define LED1 PEout(9)

#define LED2 PEout(10)

#define LED3 PEout(11)

#define LED4 PEout(12)

#define LED5 PEout(13)

#define LED6 PEout(14)

#define LED7 PEout(15)

void LED\_Init(void);//初始化

void LedValue( u8 value ); //设置数码管E-A的值；流水灯L7-L0的值

void SetLed(u8 w, u8 value); //设置数码管哪位显示什么数字

void PortationDisplay(u8 w, u8 value); //数码管显示带小数点的数值

#endif

1. Led.c

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*二极管及数码管初始化或实现\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "led.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*数码管段选\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

u8 segTable[] = {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f};

u8 segTablePortation[] = {0xbf,0x86,0xdb,0xcf,0xe6,0xed,0xfd,0x87,0xff,0xef};

/\* 初始化二极管LED灯的引脚端口 \*\*\*\*\*/

void LED\_Init(void)

{

RCC->APB2ENR|=1<<0; //使能AFIO

RCC->APB2ENR|=1<<3; //使能PORTB时钟

RCC->APB2ENR|=1<<6; //使能PORTE时钟

AFIO->MAPR |= 0x02000000; //设置PB.3为I/O口可用，且可以SW仿真

GPIOB->CRL &= 0xFFFF0FFF;

GPIOB->CRL |= 0x00003000; //PB.3推挽输出

GPIOB->ODR |= 0x00000008; //PB.3输出高

GPIOE->CRH&=0X00000000;

GPIOE->CRH|=0X33333333; //PE.8-15推挽输出

GPIOE->ODR|=0x0000ff00; //PE.8-15输出低

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 流水灯选择，或数码管段选

\* value:显示的数值对应的段选二进制值

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void LedValue(u8 value)

{

LED0 = (value&0x01)?1:0;

LED1 = (value&0x02)?1:0;

LED2 = (value&0x04)?1:0;

LED3 = (value&0x08)?1:0;

LED4 = (value&0x10)?1:0;

LED5 = (value&0x20)?1:0;

LED6 = (value&0x40)?1:0;

LED7 = (value&0x80)?1:0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 数码管显示不带小数点的数值

\* 参数 w:显示的位置，即位选，左-右：0-7

\* value:要显示的数值

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void SetLed(u8 w, u8 value)

{

SEL0 = w%2;

SEL1 = w/2%2;

SEL2 = w/4;

LedValue(segTable[value]);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 数码管显示带小数点的数值

\* 参数 w:显示的位置，即位选，左-右：0-7

\* value:要显示的数值

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void PortationDisplay(u8 w, u8 value)

{

SEL0 = w%2;

SEL1 = w/2%2;

SEL2 = w/4;

LedValue( segTablePortation[value] );

}

1. Delay.h

#ifndef \_\_DELAY\_H

#define \_\_DELAY\_H

#include <stm32f10x\_map.h>

#include <stm32f10x\_nvic.h>

//修正了中断中调用出现死循环的错误

//防止延时不准确,采用do while结构!

void delay\_init(u8 SYSCLK);

void delay\_ms(u16 nms);

void delay\_us(u32 nus);

#endif

1. Delay.c

#include <stm32f10x\_map.h>

#include <stm32f10x\_nvic.h>

#include "delay.h"

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

static u8 fac\_us=0;//us延时倍乘数

static u16 fac\_ms=0;//ms延时倍乘数

//初始化延迟函数

//SYSTICK的时钟固定为HCLK时钟的1/8

//SYSCLK:系统时钟

void delay\_init(u8 SYSCLK)

{

SysTick->CTRL&=0xfffffffb;//bit2清空,选择外部时钟 HCLK/8

fac\_us=SYSCLK/8;

fac\_ms=(u16)fac\_us\*1000;

}

//延时nms

//注意nms的范围

//SysTick->LOAD为24位寄存器,所以,最大延时为:

//nms<=0xffffff\*8\*1000/SYSCLK

//SYSCLK单位为Hz,nms单位为ms

//对72M条件下,nms<=1864

void delay\_ms(u16 nms)

{

u32 temp;

SysTick->LOAD=(u32)nms\*fac\_ms;//时间加载(SysTick->LOAD为24bit)

SysTick->VAL =0x00; //清空计数器

SysTick->CTRL=0x01 ; //开始倒数

do

{

temp=SysTick->CTRL;

}

while(temp&0x01&&!(temp&(1<<16)));//等待时间到达

SysTick->CTRL=0x00; //关闭计数器

SysTick->VAL =0X00; //清空计数器

}

// 延时nus

// nus为要延时的us数.

void delay\_us(u32 nus)

{

u32 temp;

SysTick->LOAD=nus\*fac\_us; //时间加载

SysTick->VAL=0x00; //清空计数器

SysTick->CTRL=0x01 ; //开始倒数

do

{

temp=SysTick->CTRL;

}

while(temp&0x01&&!(temp&(1<<16)));//等待时间到达

SysTick->CTRL=0x00; //关闭计数器

SysTick->VAL =0X00; //清空计数器

}

1. Sys.h

#ifndef \_\_SYS\_H

#define \_\_SYS\_H

#include <stm32f10x\_map.h>

#include <stm32f10x\_nvic.h>

#define BITBAND(addr, bitnum) ((addr & 0xF0000000)+0x2000000+((addr &0xFFFFF)<<5)+(bitnum<<2))

#define MEM\_ADDR(addr) \*((volatile unsigned long \*)(addr))

#define BIT\_ADDR(addr, bitnum) MEM\_ADDR(BITBAND(addr, bitnum))

//IO口地址映射

#define GPIOA\_ODR\_Addr (GPIOA\_BASE+12) //0x4001080C

#define GPIOB\_ODR\_Addr (GPIOB\_BASE+12) //0x40010C0C

#define GPIOC\_ODR\_Addr (GPIOC\_BASE+12) //0x4001100C

#define GPIOD\_ODR\_Addr (GPIOD\_BASE+12) //0x4001140C

#define GPIOE\_ODR\_Addr (GPIOE\_BASE+12) //0x4001180C

#define GPIOF\_ODR\_Addr (GPIOF\_BASE+12) //0x40011A0C

#define GPIOG\_ODR\_Addr (GPIOG\_BASE+12) //0x40011E0C

#define GPIOA\_IDR\_Addr (GPIOA\_BASE+8) //0x40010808

#define GPIOB\_IDR\_Addr (GPIOB\_BASE+8) //0x40010C08

#define GPIOC\_IDR\_Addr (GPIOC\_BASE+8) //0x40011008

#define GPIOD\_IDR\_Addr (GPIOD\_BASE+8) //0x40011408

#define GPIOE\_IDR\_Addr (GPIOE\_BASE+8) //0x40011808

#define GPIOF\_IDR\_Addr (GPIOF\_BASE+8) //0x40011A08

#define GPIOG\_IDR\_Addr (GPIOG\_BASE+8) //0x40011E08

//IO口操作,只对单一的IO口!

//确保n的值小于16!

#define PAout(n) BIT\_ADDR(GPIOA\_ODR\_Addr,n) //输出

#define PAin(n) BIT\_ADDR(GPIOA\_IDR\_Addr,n) //输入

#define PBout(n) BIT\_ADDR(GPIOB\_ODR\_Addr,n) //输出

#define PBin(n) BIT\_ADDR(GPIOB\_IDR\_Addr,n) //输入

#define PCout(n) BIT\_ADDR(GPIOC\_ODR\_Addr,n) //输出

#define PCin(n) BIT\_ADDR(GPIOC\_IDR\_Addr,n) //输入

#define PDout(n) BIT\_ADDR(GPIOD\_ODR\_Addr,n) //输出

#define PDin(n) BIT\_ADDR(GPIOD\_IDR\_Addr,n) //输入

#define PEout(n) BIT\_ADDR(GPIOE\_ODR\_Addr,n) //输出

#define PEin(n) BIT\_ADDR(GPIOE\_IDR\_Addr,n) //输入

#define PFout(n) BIT\_ADDR(GPIOF\_ODR\_Addr,n) //输出

#define PFin(n) BIT\_ADDR(GPIOF\_IDR\_Addr,n) //输入

#define PGout(n) BIT\_ADDR(GPIOG\_ODR\_Addr,n) //输出

#define PGin(n) BIT\_ADDR(GPIOG\_IDR\_Addr,n) //输入

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//Ex\_NVIC\_Config专用定义

#define GPIO\_A 0

#define GPIO\_B 1

#define GPIO\_C 2

#define GPIO\_D 3

#define GPIO\_E 4

#define GPIO\_F 5

#define GPIO\_G 6

#define FTIR 1 //下降沿触发

#define RTIR 2 //上升沿触发

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//JTAG模式设置定义

#define JTAG\_SWD\_DISABLE 0X02

#define SWD\_ENABLE 0X01

#define JTAG\_SWD\_ENABLE 0X00

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//void BKP\_Write(u8 reg,u16 dat); //写入后备寄存器

void Stm32\_Clock\_Init(u8 PLL); //时钟初始化

void Sys\_Soft\_Reset(void); //系统软复位

void Sys\_Standby(void); //待机模式

void MY\_NVIC\_SetVectorTable(u32 NVIC\_VectTab, u32 Offset);//设置偏移地址

void MY\_NVIC\_PriorityGroupConfig(u8 NVIC\_Group);//设置NVIC分组

void MY\_NVIC\_Init(u8 NVIC\_PreemptionPriority,u8 NVIC\_SubPriority,u8 NVIC\_Channel, u8 NVIC\_Group);//设置中断

void Ex\_NVIC\_Config(u8 GPIOx,u8 BITx,u8 TRIM);//外部中断配置函数(只对GPIOA~G)

void JTAG\_Set(u8 mode);

#endif

1. Sys.c

#include "sys.h"

//设置向量表偏移地址 //NVIC\_VectTab:基址 //Offset:偏移量

//CHECK OK //091207

void MY\_NVIC\_SetVectorTable(u32 NVIC\_VectTab, u32 Offset)

{

//检查参数合法性

assert\_param(IS\_NVIC\_VECTTAB(NVIC\_VectTab));

assert\_param(IS\_NVIC\_OFFSET(Offset));

SCB->VTOR = NVIC\_VectTab|(Offset & (u32)0x1FFFFF80);//设置NVIC的向量表偏移寄存器

//用于标识向量表是在CODE区还是在RAM区

}

//设置NVIC分组

//NVIC\_Group:NVIC分组 0~4 总共5组

//CHECK OK

//091209

void MY\_NVIC\_PriorityGroupConfig(u8 NVIC\_Group)

{

u32 temp,temp1;

temp1=(~NVIC\_Group)&0x07;//取后三位

temp1<<=8;

temp=SCB->AIRCR; //读取先前的设置

temp&=0X0000F8FF; //清空先前分组

temp|=0X05FA0000; //写入钥匙

temp|=temp1;

SCB->AIRCR=temp; //设置分组

}

//设置NVIC

//NVIC\_PreemptionPriority:抢占优先级

//NVIC\_SubPriority :响应优先级

//NVIC\_Channel :中断编号

//NVIC\_Group :中断分组 0~4

//注意优先级不能超过设定的组的范围!否则会有意想不到的错误

//组划分:

//组0:0位抢占优先级,4位响应优先级

//组1:1位抢占优先级,3位响应优先级

//组2:2位抢占优先级,2位响应优先级

//组3:3位抢占优先级,1位响应优先级

//组4:4位抢占优先级,0位响应优先级

//NVIC\_SubPriority和NVIC\_PreemptionPriority的原则是,数值越小,越优先

//CHECK OK

//100329

void MY\_NVIC\_Init(u8 NVIC\_PreemptionPriority,u8 NVIC\_SubPriority,u8 NVIC\_Channel,u8 NVIC\_Group)

{

u32 temp;

u8 IPRADDR=NVIC\_Channel/4; //每组只能存4个,得到组地址

u8 IPROFFSET=NVIC\_Channel%4;//在组内的偏移

IPROFFSET=IPROFFSET\*8+4; //得到偏移的确切位置

MY\_NVIC\_PriorityGroupConfig(NVIC\_Group);//设置分组

temp=NVIC\_PreemptionPriority<<(4-NVIC\_Group);

temp|=NVIC\_SubPriority&(0x0f>>NVIC\_Group);

temp&=0xf;//取低四位

if(NVIC\_Channel<32)NVIC->ISER[0]|=1<<NVIC\_Channel;//使能中断位(要清除的话,相反操作就OK)

else NVIC->ISER[1]|=1<<(NVIC\_Channel-32);

NVIC->IPR[IPRADDR]|=temp<<IPROFFSET;//设置响应优先级和抢断优先级

}

//外部中断配置函数

//只针对GPIOA~G;不包括PVD,RTC和USB唤醒这三个

//参数:GPIOx:0~6,代表GPIOA~G;BITx:需要使能的位;TRIM:触发模式,1,下升沿;2,上降沿;3，任意电平触发

//该函数一次只能配置1个IO口,多个IO口,需多次调用

//该函数会自动开启对应中断,以及屏蔽线

//待测试...

void Ex\_NVIC\_Config(u8 GPIOx,u8 BITx,u8 TRIM)

{

u8 EXTADDR;

u8 EXTOFFSET;

EXTADDR=BITx/4;//得到中断寄存器组的编号

EXTOFFSET=(BITx%4)\*4;

RCC->APB2ENR|=0x01;//使能io复用时钟

AFIO->EXTICR[EXTADDR]&=~(0x000F<<EXTOFFSET);//清除原来设置！！！

AFIO->EXTICR[EXTADDR]|=GPIOx<<EXTOFFSET;//EXTI.BITx映射到GPIOx.BITx

//自动设置

EXTI->IMR|=1<<BITx;// 开启line BITx上的中断

//EXTI->EMR|=1<<BITx;//不屏蔽line BITx上的事件 (如果不屏蔽这句,在硬件上是可以的,但是在软件仿真的时候无法进入中断!)

if(TRIM&0x01)EXTI->FTSR|=1<<BITx;//line BITx上事件下降沿触发

if(TRIM&0x02)EXTI->RTSR|=1<<BITx;//line BITx上事件上升降沿触发

}

//不能在这里执行所有外设复位!否则至少引起串口不工作.

//把所有时钟寄存器复位

//CHECK OK

//091209

void MYRCC\_DeInit(void)

{

RCC->APB1RSTR = 0x00000000;//复位结束

RCC->APB2RSTR = 0x00000000;

RCC->AHBENR = 0x00000014; //睡眠模式闪存和SRAM时钟使能.其他关闭.

RCC->APB2ENR = 0x00000000; //外设时钟关闭.

RCC->APB1ENR = 0x00000000;

RCC->CR |= 0x00000001; //使能内部高速时钟HSION

RCC->CFGR &= 0xF8FF0000; //复位SW[1:0],HPRE[3:0],PPRE1[2:0],PPRE2[2:0],ADCPRE[1:0],MCO[2:0]

RCC->CR &= 0xFEF6FFFF; //复位HSEON,CSSON,PLLON

RCC->CR &= 0xFFFBFFFF; //复位HSEBYP

RCC->CFGR &= 0xFF80FFFF; //复位PLLSRC, PLLXTPRE, PLLMUL[3:0] and USBPRE

RCC->CIR = 0x00000000; //关闭所有中断

//配置向量表

#ifdef VECT\_TAB\_RAM

MY\_NVIC\_SetVectorTable(NVIC\_VectTab\_RAM, 0x0);

#else

MY\_NVIC\_SetVectorTable(NVIC\_VectTab\_FLASH, 0x0);

#endif

}

//THUMB指令不支持汇编内联

//采用如下方法实现执行汇编指令WFI

//CHECK OK

//091209

\_\_asm void WFI\_SET(void)

{

WFI;

}

//进入待机模式

//check ok

//091202

void Sys\_Standby(void)

{

SCB->SCR|=1<<2;//使能SLEEPDEEP位 (SYS->CTRL)

RCC->APB1ENR|=1<<28; //使能电源时钟

PWR->CSR|=1<<8; //设置WKUP用于唤醒

PWR->CR|=1<<2; //清除Wake-up 标志

PWR->CR|=1<<1; //PDDS置位

WFI\_SET(); //执行WFI指令

}

//后备寄存器写入操作

//reg:寄存器编号

//reg:要写入的数值

////check ok

////091202

//void BKP\_Write(u8 reg,u16 dat)

//{

// RCC->APB1ENR|=1<<28; //使能电源时钟

// RCC->APB1ENR|=1<<27; //使能备份时钟

// PWR->CR|=1<<8; //取消备份区写保护

// switch(reg)

// {

// case 1:

// BKP->DR1=dat;

// break;

// case 2:

// BKP->DR2=dat;

// break;

// case 3:

// BKP->DR3=dat;

// break;

// case 4:

// BKP->DR4=dat;

// break;

// case 5:

// BKP->DR5=dat;

// break;

// case 6:

// BKP->DR6=dat;

// break;

// case 7:

// BKP->DR7=dat;

// break;

// case 8:

// BKP->DR8=dat;

// break;

// case 9:

// BKP->DR9=dat;

// break;

// case 10:

// BKP->DR10=dat;

// break;

// }

//}

//系统软复位

//CHECK OK

//091209

void Sys\_Soft\_Reset(void)

{

SCB->AIRCR =0X05FA0000|(u32)0x04;

}

//JTAG模式设置,用于设置JTAG的模式

//mode:jtag,swd模式设置;00,全使能;01,使能SWD;10,全关闭;

//CHECK OK

//100818

void JTAG\_Set(u8 mode)

{

u32 temp;

temp=mode;

temp<<=25;

RCC->APB2ENR|=1<<0; //开启辅助时钟

AFIO->MAPR&=0XF8FFFFFF; //清除MAPR的[26:24]

AFIO->MAPR|=temp; //设置jtag模式

}

//系统时钟初始化函数

//pll:选择的倍频数，从2开始，最大值为16

//CHECK OK

//091209

void Stm32\_Clock\_Init(u8 PLL)

{

unsigned char temp=0;

MYRCC\_DeInit(); //复位并配置向量表

RCC->CR|=0x00010000; //外部高速时钟使能HSEON

while(!(RCC->CR>>17));//等待外部时钟就绪

RCC->CFGR=0X00000400; //APB1=DIV2;APB2=DIV1;AHB=DIV1;

PLL-=2;//抵消2个单位

RCC->CFGR|=PLL<<18; //设置PLL值 2~16

RCC->CFGR|=1<<16; //PLLSRC ON

FLASH->ACR|=0x32; //FLASH 2个延时周期

RCC->CR|=0x01000000; //PLLON

while(!(RCC->CR>>25));//等待PLL锁定

RCC->CFGR|=0x00000002;//PLL作为系统时钟

while(temp!=0x02) //等待PLL作为系统时钟设置成功

{

temp=RCC->CFGR>>2;

temp&=0x03;

}

}

1. Usart.h

#ifndef \_\_USART\_H

#define \_\_USART\_H

#include <stm32f10x\_map.h>

#include <stm32f10x\_nvic.h>

#include "stdio.h"

extern u8 USART\_RX\_BUF[64]; //接收缓冲,最大63个字节.末字节为换行符

extern u8 USART\_RX\_STA; //接收状态标记

//如果想串口中断接收，请不要注释以下宏定义

//#define EN\_USART1\_RX //使能串口1接收

void uart\_init(u32 pclk2,u32 bound);

#endif

1. Usart.c

#include "sys.h"

#include "usart.h"

//加入以下代码,支持printf函数

#if 1

#pragma import(\_\_use\_no\_semihosting)

//标准库需要的支持函数

struct \_\_FILE

{

int handle;

/\* Whatever you require here. If the only file you are using is \*/

/\* standard output using printf() for debugging, no file handling \*/

/\* is required. \*/

};

/\* FILE is typedef’ d in stdio.h. \*/

FILE \_\_stdout;

//定义\_sys\_exit()以避免使用半主机模式

\_sys\_exit(int x)

{

x = x;

}

//重定义fputc函数

int fputc(int ch, FILE \*f)

{

while((USART1->SR&0X40)==0);//循环发送,直到发送完毕

USART1->DR = (u8) ch;

return ch;

}

#endif

//end

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#ifdef EN\_USART1\_RX //如果使能了接收

//串口1中断服务程序

//注意,读取USARTx->SR能避免莫名其妙的错误

u8 USART\_RX\_BUF[64]; //接收缓冲,最大64个字节.

//接收状态

//bit7，接收完成标志

//bit6，接收到0x0d

//bit5~0，接收到的有效字节数目

u8 USART\_RX\_STA=0; //接收状态标记

void USART1\_IRQHandler(void)

{

u8 res;

if(USART1->SR&(1<<5))//接收到数据

{

res=USART1->DR;

if((USART\_RX\_STA&0x80)==0)//接收未完成

{

if(USART\_RX\_STA&0x40)//接收到了0x0d

{

if(res!=0x0a)USART\_RX\_STA=0;//接收错误,重新开始

else USART\_RX\_STA|=0x80; //接收完成了

}else //还没收到0X0D

{

if(res==0x0d)USART\_RX\_STA|=0x40;

else

{

USART\_RX\_BUF[USART\_RX\_STA&0X3F]=res;

USART\_RX\_STA++;

if(USART\_RX\_STA>63)USART\_RX\_STA=0;//接收数据错误,重新开始接收

}

}

}

}

}

#endif

//初始化IO 串口1 pclk2:PCLK2时钟频率(Mhz)

//bound:波特率 CHECK OK 091209

void uart\_init(u32 pclk2,u32 bound)

{

float temp;

u16 mantissa;

u16 fraction;

temp=(float)(pclk2\*1000000)/(bound\*16);//得到USARTDIV

mantissa=temp; //得到整数部分

fraction=(temp-mantissa)\*16; //得到小数部分

mantissa<<=4;

mantissa+=fraction;

RCC->APB2ENR|=1<<2; //使能PORTA口时钟

RCC->APB2ENR|=1<<14; //使能串口时钟

GPIOA->CRH&=0XFFFFF00F;

GPIOA->CRH|=0X000008B0;//IO状态设置

RCC->APB2RSTR|=1<<14; //复位串口1

RCC->APB2RSTR&=~(1<<14);//停止复位

//波特率设置

USART1->BRR=mantissa; // 波特率设置

USART1->CR1|=0X200C; //1位停止,无校验位.

#ifdef EN\_USART1\_RX //如果使能了接收

//使能接收中断

USART1->CR1|=1<<8; //PE中断使能

USART1->CR1|=1<<5; //接收缓冲区非空中断使能

MY\_NVIC\_Init(3,3,USART1\_IRQChannel,2);//组2，最低优先级

#endif

}

1. Main.c

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*流水灯\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

现象：二极管从左至右依次全部点亮 \*/

#include "sys.h"

#include "delay.h"

#include "led.h"

u8 light;

int main( void )

{

Stm32\_Clock\_Init( 6 ); //6倍频

delay\_init( 72 ); //12M外部晶振

LED\_Init();

GPIOE->ODR &= ~(0xff<<8);

LED\_SEL = 1; //选择二极管

light = 0x01;

while( 1 )

{

// LED0 = (light&0x01)?1:0;

// LED1 = light&0x02?1:0;

// LED2 = light&0x04?1:0;

// LED3 = light&0x08?1:0;

// LED4 = light&0x10?1:0;

// LED5 = light&0x20?1:0;

// LED6 = light&0x40?1:0;

// LED7 = light&0x80?1:0;

GPIOE->ODR |= (light<<8);

delay\_ms( 300 );

light = light<<1;

if( light==0x00 )

{

GPIOE->ODR &= ~(0xff<<8);

delay\_ms( 300 );

light = 0x01;

}

}

}

# 实验三 8位数码管动态扫描

## 实验目的

1. 了解8位数码管动态显示原理；
2. 掌握stm32f10x扩展端口的方法。

## 背景知识

1. 原理图

本实验电路接线与实验二完全相同，具体参见图4，下面仅给出LED数码管的引脚定义。

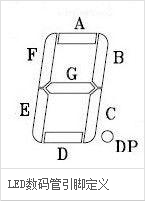


图5 数码管引脚

1. 基本原理

数码管中的A~G、DP段分别连接到电路图中的A~G、H线上，当某段上有一定的电压差值时，便会点亮该段。

当E3输入为1，也就是LED\_SEL输入为0时，根据SEL0~SEL2的值确定选中的数码管，即位选，再根据A~H引脚的高低电平，点亮对应段，即段选。

寄存器的具体说明可参见附录2。

1. 实验现象与操作

对于给出的8位数码管动态扫描案例，下载后，在开发板上可观察到8个数码管从左至右依次显示对应的数字，且每一个数码显示的数字在1-9之间循环。

可以通过加快扫描频率，使得八位数码管在人眼看上去是同时显示。在后续的案例中可以看到该现象。

## 实验内容和步骤

用与实验二相同的方法建立keil工程文件，每个工程文件应该对应一个独立的文件夹。然后依次建立各个头文件和C代码文件，加入到工程中，编译连接后形成.HEX文件，上载到实验板后运行观察。

修改晶振频率，重新编译连接后再次进行观察，对两次结果加以比较，并分析由此产生的视觉效果差异。

## 程序清单

1. **Led.h（同实验二）**
2. **Led.c（同实验二）**
3. **Delay.h（同实验二）**
4. **Delay.c（同实验二）**
5. **Sys.h（同实验二）**
6. **Sys.c（同实验二）**
7. **Usart.h**

#ifndef \_\_USART\_H

#define \_\_USART\_H

#include <stm32f10x\_map.h>

#include <stm32f10x\_nvic.h>

#include "stdio.h"

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//支持适应不同频率下的串口波特率设置.

//加入了对printf的支持

//增加了串口接收命令功能.

//修正了printf第一个字符丢失的bug

extern u8 USART\_RX\_BUF[64]; //接收缓冲,最大63个字节.末字节为换行符

extern u8 USART\_RX\_STA; //接收状态标记

//如果想串口中断接收，请不要注释以下宏定义

//#define EN\_USART1\_RX //使能串口1接收

void uart\_init(u32 pclk2,u32 bound);

#endif

1. **Usart.c**

#include "sys.h"

#include "usart.h"

//支持适应不同频率下的串口波特率设置.

//加入了对printf的支持

//增加了串口接收命令功能.

//修正了printf第一个字符丢失的bug

//加入以下代码,支持printf函数,而不需要选择use MicroLIB

#if 1

#pragma import(\_\_use\_no\_semihosting)

//标准库需要的支持函数

struct \_\_FILE

{

int handle;

/\* Whatever you require here. If the only file you are using is \*/

/\* standard output using printf() for debugging, no file handling \*/

/\* is required. \*/

};

/\* FILE is typedef’ d in stdio.h. \*/

FILE \_\_stdout;

//定义\_sys\_exit()以避免使用半主机模式

\_sys\_exit(int x)

{

x = x;

}

//重定义fputc函数

int fputc(int ch, FILE \*f)

{

while((USART1->SR&0X40)==0);//循环发送,直到发送完毕

USART1->DR = (u8) ch;

return ch;

}

#endif

//end

#ifdef EN\_USART1\_RX //如果使能了接收

//串口1中断服务程序

//注意,读取USARTx->SR能避免莫名其妙的错误

u8 USART\_RX\_BUF[64]; //接收缓冲,最大64个字节.

//接收状态

//bit7，接收完成标志

//bit6，接收到0x0d

//bit5~0，接收到的有效字节数目

u8 USART\_RX\_STA=0; //接收状态标记

void USART1\_IRQHandler(void)

{

u8 res;

if(USART1->SR&(1<<5))//接收到数据

{

res=USART1->DR;

if((USART\_RX\_STA&0x80)==0)//接收未完成

{

if(USART\_RX\_STA&0x40)//接收到了0x0d

{

if(res!=0x0a)USART\_RX\_STA=0;//接收错误,重新开始

else USART\_RX\_STA|=0x80; //接收完成了

}else //还没收到0X0D

{

if(res==0x0d)USART\_RX\_STA|=0x40;

else

{

USART\_RX\_BUF[USART\_RX\_STA&0X3F]=res;

USART\_RX\_STA++;

if(USART\_RX\_STA>63)USART\_RX\_STA=0;//接收数据错误,重新开始接收

}

}

}

}

}

#endif

//初始化IO 串口1

//pclk2:PCLK2时钟频率(Mhz)

//bound:波特率

//CHECK OK

//091209

void uart\_init(u32 pclk2,u32 bound)

{

float temp;

u16 mantissa;

u16 fraction;

temp=(float)(pclk2\*1000000)/(bound\*16);//得到USARTDIV

mantissa=temp; //得到整数部分

fraction=(temp-mantissa)\*16; //得到小数部分

mantissa<<=4;

mantissa+=fraction;

RCC->APB2ENR|=1<<2; //使能PORTA口时钟

RCC->APB2ENR|=1<<14; //使能串口时钟

GPIOA->CRH&=0XFFFFF00F;

GPIOA->CRH|=0X000008B0;//IO状态设置

RCC->APB2RSTR|=1<<14; //复位串口1

RCC->APB2RSTR&=~(1<<14);//停止复位

//波特率设置

USART1->BRR=mantissa; // 波特率设置

USART1->CR1|=0X200C; //1位停止,无校验位.

#ifdef EN\_USART1\_RX //如果使能了接收

//使能接收中断

USART1->CR1|=1<<8; //PE中断使能

USART1->CR1|=1<<5; //接收缓冲区非空中断使能

MY\_NVIC\_Init(3,3,USART1\_IRQChannel,2);//组2，最低优先级

#endif

}

1. **Main.c**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*8位数码管动态扫描\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 8位数码管动态扫描

\* 每位数码管显示的数字从0加到9

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "sys.h"

#include "delay.h"

#include "led.h"

#define uchar unsigned char

uchar show\_w1,show\_w2,show\_w3,show\_w4,show\_w5,show\_w6,show\_w7,show\_w8,flag,count;

uchar seg[] = {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*主函数\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main()

{

//uchar i;

Stm32\_Clock\_Init( 6 );

delay\_init( 72 );

LED\_Init();

LED\_SEL = 0;

show\_w1=0;

show\_w2=1;

show\_w3=2;

show\_w4=3;

show\_w5=4;

show\_w6=5;

show\_w7=6;

show\_w8=7;

while(1)

{

// for( i=0; i<8; i++ )

// {

// SEL2 = i/4;

// SEL1 = i%4/2;

// SEL0 = i%2;

//

// switch(i)

// {

// case 0: LedValue( seg[show\_w1%10] ); delay\_ms(100); break;

// case 1: LedValue( seg[show\_w2%10] ); delay\_ms(100); break;

// case 2: LedValue( seg[show\_w3%10] ); delay\_ms(100); break;

// case 3: LedValue( seg[show\_w4%10] ); delay\_ms(100); break;

// case 4: LedValue( seg[show\_w5%10] ); delay\_ms(100); break;

// case 5: LedValue( seg[show\_w6%10] ); delay\_ms(100); break;

// case 6: LedValue( seg[show\_w7%10] ); delay\_ms(100); break;

// default: LedValue( seg[show\_w8%10] ); delay\_ms(100); break;

// }

// }

SetLed(0, show\_w1%10);

delay\_ms(100);

SetLed(1, show\_w2%10);

delay\_ms(100);

SetLed(2, show\_w3%10);

delay\_ms(100);

SetLed(3, show\_w4%10);

delay\_ms(100);

SetLed(4, show\_w5%10);

delay\_ms(100);

SetLed(5, show\_w6%10);

delay\_ms(100);

SetLed(6, show\_w7%10);

delay\_ms(100);

SetLed(7, show\_w8%10);

delay\_ms(100);

show\_w1++;

show\_w2++;

show\_w3++;

show\_w4++;

show\_w5++;

show\_w6++;

show\_w7++;

show\_w8++;

}

}

# 实验四 按键输入

## 实验目的

1. 通过本实验了解STM32F103RB硬件结构，掌握各引脚功能和接线；
2. 了解GPIO各寄存器的功能与应用，加深对按键输入检测过程的理解。

## 实验内容和步骤

步骤1：建立Key工程，芯片选用STM32103RB，其他设置与实验二相同。拷贝参考程序清单中的代码，编译后形成可执行文件；

步骤2：编译链接成功后，上载到实验板，用不同的速率敲击或长按键盘按键，或者同时按下多个按键，观察LED的反转速度有什么变化，结合按键接收原理，分析这个过程；

步骤3：仔细阅读程序，不断修改延迟参数，找出用户满意度最高的参数设置。

## 给出实验结论

## 参考程序清单

1. **Led.h（同实验二）**
2. **Led.c（同实验二）**
3. **Key.h**

#ifndef \_\_KEY\_H

#define \_\_KEY\_H

#include "sys.h"

//按键输入 驱动代码

//修改按键扫描函数，使整个代码可以支持SWD下载。

#define KEY1 PCin(2) //PC2

#define KEY2 PCin(1) //PC1

#define KEY3 PCin(0) //PC0

void KEY\_Init(void);//IO初始化

u8 KEY\_Scan(void); //按键扫描函数

#endif

1. **Key.c**

#include "key.h"

#include "delay.h"

//按键输入 驱动代码

//PC0.1.2 设置成输入

void KEY\_Init(void)

{

RCC->APB2ENR|=1<<4; //使能PORTC时钟

GPIOC->CRL&=0XFFFFF000;//PC0-2设置成输入

GPIOC->CRL|=0X00000888;

}

//按键处理函数

//返回按键值

//0，没有任何按键按下

//1，KEY1按下

//2，KEY2按下

//3，KEY3按下

//注意此函数有响应优先级,KEY1>KEY2>KEY3!!

u8 KEY\_Scan(void)

{

static u8 key\_up=1;//按键按松开标志

if(key\_up && (KEY1==0 || KEY2==0 || KEY3==0))

{

delay\_ms(10);//去抖动

key\_up=0;

if(KEY1==0)

{

return 1;

}

else if(KEY2==0)

{

return 2;

}

else if(KEY3==0)

{

return 3;

}

}

else if(KEY1==1 && KEY2==1 && KEY3==1)

key\_up=1;

return 0;// 无按键按下

}

1. **Delay.h（同实验二）**
2. **Delay.c（同实验二）**
3. **Sys.h（同实验二）**
4. **Sys.c（同实验二）**
5. **Main.c**

#include "sys.h"

#include "delay.h"

#include "led.h"

#include "key.h"

//按键输入实验

int main(void)

{

u8 t;

Stm32\_Clock\_Init(9); //系统时钟设置

delay\_init(72); //延时初始化

LED\_Init(); //初始化与LED连接的硬件接口

KEY\_Init(); //初始化与按键连接的硬件接口

while(1)

{

t=KEY\_Scan();//得到键值

if(t)

{

switch(t)

{

case 1:

LED0=!LED0;

break;

case 2:

LED3=!LED3;

break;

case 3:

LED7=!LED7;

break;

}

}else delay\_ms(10);

}

}

# 实验五 窗口看门狗

## 实验目的

1. 加深对软件看门狗原理和作用的理解；
2. 掌握STM32定时器资源的使用。

## 背景知识

1. 窗口看门狗原理

窗口看门狗（WWDG）通常被用来检测由外部干扰或不可预见的逻辑条件造成的应用程序背离原来正常的运行序列而产生的软件故障。除非递减计数器的值在T6位（WWDG->CR的第六位）变成0之前被刷新，看门狗电路在达到预置的时间周期前，会产生一个MCU复位。在递减计数器达到窗口配置寄存器（WWDG->CRF）数值之前，如果7位的递减计数器数值（在控制寄存器中）被刷新，那么也将产生一个MCU复位。这表明递减计数器需要在一个有限的时间窗口窗口中被刷新，其关系可以用图6来说明。

本实验将实现如下功能：程序一运行则开启WWDG,并使得L0亮500ms后关闭，进入死循环。等待WWDG中断的到来，在中断里面，喂狗，并执行L1的翻转操作。所以可以看到L1不停的闪烁，而L0只在刚启动的时候闪一下。

T[6:0]CNT递减计数器

W[6:0]

3Fh

时间

T6位

刷新窗口

不允许刷新

复位

图6 WWDG原理图

在图6中，T[6：0]就是WWDG\_CR的第7位，W[6:0]即是WWDG->CFR的低7位T[6:0]就是窗口看门狗的计数器，而W[6:0]则是窗口看门狗的上窗口，下窗口值是固定的0x40。当窗口看门狗的计数器在上窗口值之外被刷新，或者低于下窗口值都会产生复位。

上窗口值W[6:0]是由用户自己设定的，根据实际要求来设计窗口值，但是一定要确保窗口值大于0x40，否则窗口就不存在了。

窗口看门狗的超时公式如下：

Twwdg = (4096 \* 2^WDGTB \*(T[5:0] + 1) / Fpclk1

其中：

Twwdg：WWDG超时间隔（单位ms）

Fpclk1：APB1的时钟频率（单位KHz）

WDGTB：WWDG的预分频系数

T[5:0]:窗口看门狗的计数器低6位

根据上面公式，假设Fpclk1 = 36MHz，那么可以得到最小-最大超时时间表：

表2 超时时间表

|  |  |  |
| --- | --- | --- |
| WDTB | 最小超时值 | 最大超时值 |
| 0 | 113us | 7.28ms |
| 1 | 227us | 14.56ms |
| 2 | 455us | 29.12ms |
| 3 | 910us | 58.25ms |

因此，STM32F的窗口看门狗中有一个7位的递减计数器，它会在出现下列情况之一的时候产生看门狗复位：

1. 当计数器的数值从0x40减到0x3F时；
2. 当喂狗的时候如果计数器的值大于某一设定数值时，此设定数值在WWDG\_CFR寄存器中定义。

对于一般的看门狗，程序可以在它产生复位前的任意时刻刷新看门狗，但这也有一个隐患，有可能程序跑乱了又跑回正常的地方，或者跑乱的程序正好执行了刷新看门狗操作，这种情况下一般的看门狗就检测不出来了。

1. 窗口看门狗相关寄存器

**控制寄存器WWDG\_CR**

该寄存器只使用低8位，其中位7（WDGA）由软件置1，但仅能由硬件在复位后变0，当WDGA=1时，看门狗可以产生复位。该位为0：禁止看门狗；为1：启用看门狗。位6到位0用来存储看门狗的计数器值，每（4096×2^WDGTB）个周期减1，当计数器从40h变成3Fh时（T6=0），看门狗产生复位。

**配置寄存器WWDG\_CFR**

位9（EWI）：提前唤醒中断，若此位置1，则当计数器值达到40h时，产生中断。此中断只能由硬件在复位后清除。

位8～7：时基。00表示CK计时器时钟（PCLK1 / 4096）/1；01：表示CK计时器时钟（PCLK1 / 4096）/2；10：表示CK计时器时钟（PCLK1 / 4096）/4；11表示CK计时器时钟（PCLK1 / 4096）/8。

位6～0：7位窗口值

状态寄存器WWDG\_SR

用来记录当前是否有提前唤醒的标志。该寄存器仅有位0有效，其他均保留。当计数器达到40h时，此位由硬件置1.它必须通过软件写0来清除。对此位写1无效。即使中断未被使能，在计数器的值达到0x40的时候，此位也被置1.

## 实验内容和步骤

步骤1：使能WWDG时钟。WWDG不同于IWDG，IWDG有自己的40KHz时钟，不存在使能问题。而WWDG使用的是PCLK1的时钟，需要先使能时钟。

步骤2：设置WWDG\_CFR和WWDG\_CR两个寄存器。在时钟使能完后，通过设置CFR和CR两个寄存器，来完成对WWDG的配置，具体包括使能窗口看门狗、开启中断、设置计数器的初始值、设置窗口值并设置分频数WDGTB等。

步骤3：开启WWDG中断并分组在设置完WWDG后，需要配置该中断的分组及使能，这里主要通过MY\_NVIC\_Init函数实现。

步骤4：编写中断服务函数。最后，还是要编写窗口看门狗的中断服务函数，通过该函数来喂狗。喂狗要快，否则当窗口看门狗计数器值减到0x3F的时候，就会引起软复位了。在中断服务函数里面也要将状态寄存器的EWIF位清空。

项目中使用的处理器为STM32F103RB。

## 源程序清单

1. delay.c led.c sys.c usart.c及相关.h文件与流水灯实验相同
2. **exti.h**

#ifndef \_\_EXTI\_H

#define \_\_EXIT\_H

#include "sys.h"

void EXTIX\_Init(void);//IO初始化

#endif

1. **exti.c**

**#include "exti.h"**

**#include "led.h"**

**#include "key.h"**

**#include "delay.h"**

**//外部中断 驱动代码**

**//外部中断0服务程序**

**void EXTI0\_IRQHandler(void)**

**{**

**delay\_ms(10);//消抖**

**if(KEY3==0) //按键3**

**{**

**LED7=!LED7;**

**}**

**EXTI->PR=1<<0; //清除LINE0上的中断标志位**

**}**

**//外部中断1服务程序**

**void EXTI1\_IRQHandler(void)**

**{**

**delay\_ms(10);//消抖**

**if(KEY2==0) //按键2**

**{**

**LED3=!LED3;**

**}**

**EXTI->PR=1<<1; //清除LINE1上的中断标志位**

**}**

**//外部中断2服务程序**

**void EXTI2\_IRQHandler(void)**

**{**

**delay\_ms(10);//消抖**

**if(KEY1==0) //按键1**

**{**

**LED0=!LED0;**

**}**

**EXTI->PR=1<<2; //清除LINE2上的中断标志位**

**}**

**//外部中断初始化程序**

**//初始化PC0-2为中断输入.**

**void EXTIX\_Init(void)**

**{**

**RCC->APB2ENR|=1<<4; //使能PORTC时钟**

**GPIOC->CRL&=0XFFFFF000;//PC0-2设置成输入**

**GPIOC->CRL|=0X00000888;**

**Ex\_NVIC\_Config(GPIO\_C,0,FTIR);//下降沿触发**

**Ex\_NVIC\_Config(GPIO\_C,1,FTIR);//下降沿触发**

**Ex\_NVIC\_Config(GPIO\_C,2,FTIR);//下降沿触发**

**MY\_NVIC\_Init(2,2,EXTI0\_IRQChannel,2);//抢占2，子优先级2，组2**

**MY\_NVIC\_Init(2,1,EXTI1\_IRQChannel,2);//抢占2，子优先级1，组2**

**MY\_NVIC\_Init(2,0,EXTI2\_IRQChannel,2);//抢占2，子优先级1，组2**

**}**

1. **wdg.h**

**#ifndef \_\_WDG\_H**

**#define \_\_WDG\_H**

**#include "sys.h"**

**//看门狗 驱动代码**

**void IWDG\_Init(u8 prer,u16 rlr); //初始化IWDG**

**void IWDG\_Feed(void); //IWDG喂狗**

**void WWDG\_Init(u8 tr,u8 wr,u8 fprer);//初始化WWDG**

**void WWDG\_Set\_Counter(u8 cnt); //设置WWDG的计数器**

**#endif**

1. **wdg.c**

**#include "wdg.h"**

**#include "led.h"**

**//看门狗 驱动代码**

**//初始化独立看门狗**

**//prer:分频数:0~7(只有低3位有效!)**

**//分频因子=4\*2^prer.但最大值只能是256!**

**//rlr:重装载寄存器值:低11位有效.**

**//时间计算(大概):Tout=((4\*2^prer)\*rlr)/40 (ms).**

**void IWDG\_Init(u8 prer,u16 rlr)**

**{**

**IWDG->KR=0X5555;//使能对IWDG->PR和IWDG->RLR的写**

**IWDG->PR=prer; //设置分频系数**

**IWDG->RLR=rlr; //从加载寄存器 IWDG->RLR**

**IWDG->KR=0XAAAA;//reload**

**IWDG->KR=0XCCCC;//使能看门狗**

**}**

**//喂独立看门狗**

**void IWDG\_Feed(void)**

**{**

**IWDG->KR=0XAAAA;//reload**

**}**

**//保存WWDG计数器的设置值,默认为最大.**

**u8 WWDG\_CNT=0x7f;**

**//初始化窗口看门狗**

**//tr :T[6:0],用于存储计数器的值**

**//wr :W[6:0],用于存储窗口值**

**//fprer:窗口看门狗的实际设置**

**//低2位有效.Fwwdg=PCLK1/4096/2^fprer.**

**void WWDG\_Init(u8 tr,u8 wr,u8 fprer)**

**{**

**RCC->APB1ENR|=1<<11; //使能wwdg时钟**

**WWDG\_CNT=tr&WWDG\_CNT; //初始化WWDG\_CNT.**

**WWDG->CFR|=fprer<<7; //PCLK1/4096再除2^fprer**

**WWDG->CFR|=1<<9; //使能提前唤醒中断**

**WWDG->CFR&=0XFF80;**

**WWDG->CFR|=wr; //设定窗口值**

**WWDG->CR|=WWDG\_CNT|(1<<7); //开启看门狗,设置7位计数器**

**MY\_NVIC\_Init(2,3,WWDG\_IRQChannel,2);//抢占2，子优先级3，组2**

**}**

**//重设置WWDG计数器的值**

**void WWDG\_Set\_Counter(u8 cnt)**

**{**

**WWDG->CR|=(cnt&0x7F);//重设置7位计数器**

**}**

**//窗口看门狗中断服务程序**

**void WWDG\_IRQHandler(void)**

**{**

**u8 wr,tr;**

**wr=WWDG->CFR&0X7F;**

**tr=WWDG->CR&0X7F;**

**if(tr<wr)WWDG\_Set\_Counter(WWDG\_CNT);//只有当计数器的值,小于窗口寄存器的值才能写CR!!**

**WWDG->SR=0X00;//清除提前唤醒中断标志位**

**LED1=!LED1;**

**}**

1. **main.c**

**#include "sys.h"**

**#include "usart.h"**

**#include "delay.h"**

**#include "led.h"**

**#include "key.h"**

**#include "exti.h"**

**#include "wdg.h"**

**//窗口看门狗实验**

**int main(void)**

**{**

**Stm32\_Clock\_Init(9); //系统时钟设置**

**delay\_init(72); //延时初始化**

**uart\_init(72,9600); //串口初始化**

**LED\_Init(); //初始化与LED连接的硬件接口**

**LED0=1;**

**delay\_ms(500);**

**WWDG\_Init(0X7F,0X5F,3);//计数器值为7f,窗口寄存器为5f,分频数为8**

**while(1)**

**{**

**LED0=0;**

**}**

**}**

# 实验六 定时器中断

## 实验目的

1. 初步掌握中断编程的一般原理与基本方法；
2. 掌握定时器寄存器状态配置；
3. 进一步理解STM32系列处理器的定时器资源与设置。

## 背景知识

STM32的通用定时器是一个通过可编程预分频器（PSC）驱动的16位自动装载计数器（CNT）构成。STM32的通用定时器可以被用于：测量输入信号的脉冲长度(输入捕获)或者产生输出波形(输出比较和PWM)等。 使用定时器预分频器和RCC时钟控制器预分频器，脉冲长度和波形周期可以在几个微秒到几个毫秒间调整。STM32的每个通用定时器都是完全独立的，没有互相共享的任何资源。

STM3的通用TIMx (TIM2、TIM3、TIM4和TIM5)定时器功能包括：

1)16位向上、向下、向上/向下自动装载计数器（TIMx\_CNT）

2)16位可编程(可以实时修改)预分频器(TIMx\_PSC)，计数器时钟频率的分频系数为1～65535之间的任意数值。

3）4个独立通道（TIMx\_CH1~4），这些通道可以用来作为：

A．输入捕获

B．输出比较

C．PWM生成(边缘或中间对齐模式)

D．单脉冲模式输出

 4）可使用外部信号（TIMx\_ETR）控制定时器和定时器互连（可以用1个定时器控制另外一个定时器）的同步电路。

5）如下事件发生时产生中断/DMA：

A．更新：计数器向上溢出/向下溢出，计数器初始化(通过软件或者内部/外部触发)

B．触发事件(计数器启动、停止、初始化或者由内部/外部触发计数)

C．输入捕获

D．输出比较

E．支持针对定位的增量(正交)编码器和霍尔传感器电路

F．触发输入作为外部时钟或者按周期的电流管理

由于STM32通用定时器比较复杂，这里我们不再多介绍，请大家直接参考《STM32参考手册》第211页，通用定时器一章。下面我们介绍一下与我们这节实验密切相关的几个通用定时器的寄存器。

本实验将实现如下功能：利用TIM3的定时器中断来控制L1的翻转，在主函数用L0的翻转来提示程序正在运行。下面提供的代码可以JTAG/SWD下载仿真。

## 参考程序清单

1. Exti.c key.c led.c wdg.c delay.c sys.c usart.c均与前面的实验相同；
2. Timer.h

#ifndef \_\_TIMER\_H

#define \_\_TIMER\_H

#include "sys.h"

void Timerx\_Init(u16 arr,u16 psc);

#endif

1. Timer.c

#include "timer.h"

#include "led.h"

//通用定时器 驱动代码

//定时器3中断服务程序

void TIM3\_IRQHandler(void)

{

if(TIM3->SR&0X0001)//溢出中断

{

LED1=!LED1;

}

TIM3->SR&=~(1<<0);//清除中断标志位

}

//通用定时器中断初始化

//这里时钟选择为APB1的2倍，而APB1为36M

//arr：自动重装值。

//psc：时钟预分频数

//这里使用的是定时器3!

void Timerx\_Init(u16 arr,u16 psc)

{

RCC->APB1ENR|=1<<1;//TIM3时钟使能

TIM3->ARR=arr; //设定计数器自动重装值//刚好1ms

TIM3->PSC=psc; //预分频器7200,得到10Khz的计数时钟

//这两个东东要同时设置才可以使用中断

TIM3->DIER|=1<<0; //允许更新中断

TIM3->DIER|=1<<6; //允许触发中断

TIM3->CR1|=0x01; //使能定时器3

MY\_NVIC\_Init(1,3,TIM3\_IRQChannel,2);//抢占1，子优先级3，组2

}

1. Main.c

#include "sys.h"

#include "usart.h"

#include "delay.h"

#include "led.h"

#include "key.h"

#include "exti.h"

#include "wdg.h"

#include "timer.h"

//定时器中断 实验

int main(void)

{

Stm32\_Clock\_Init(9); //系统时钟设置

delay\_init(72); //延时初始化

uart\_init(72,9600); //串口初始化

LED\_Init(); //初始化与LED连接的硬件接口

Timerx\_Init(5000,7199);//10Khz的计数频率，计数到5000为500ms

while(1)

{

LED0=!LED0;

delay\_ms(200);

}

}

# 附录1 Keil C编译器常见错误与警告

1. Warning 280:’i’:unreferenced local variable 说明局部变量i 在函数中未作任何的存取操作解决方法消除函数中i 变量的宣告。

2． Warning 206:’Music3’:missing function-prototype  说明Music3( )函数未作宣告或未作外部宣告所以无法给其他函数调用解决方法将叙述void Music3(void)写在程序的最前端作宣告如果是其他文件的函数则要写成extern void Music3(void),即作外部宣告。

3． Compling :C:\8051\MANN.C  Error:318:can’t open file ‘beep.h’ 说明在编译C:\8051\MANN.C 。

程序过程中由于main.c 用了指令＃i nclude “beep.h”,但却找不到所致解决方法编写一个beep.h 的包含档并存入到c:\8051 的工作目录中。

4． Compling:C:\8051\LED.C  Error 237:’LedOn’:function already has a body 说明LedOn( )函数名称重复定义即有两个以上一样的函数名称解决方法修正其中的一个函数名称使得函数名称都是独立的。

5． \*\*\*WARNING 16:UNCALLED SEGMENT,IGNORED FOR OVERLAY PROCESS  SEGMENT: ?PR?\_DELAYX1MS?DELAY  说明DelayX1ms( )函数未被其它函数调用也会占用程序记忆体空间解决方法去掉DelayX1ms( )函数或利用条件编译#if „..#endif,可保留该函数并不编译 \*\*\* WARNING。

6．XDATA SPACE MEMORY OVERLAP  FROM : 0025H  TO: 0025H  说明外部资料ROM 的0025H 重复定义地址解决方法外部资料ROM 的定义如下Pdata unsigned char XFR\_ADC \_at\_0x25 其中XFR\_ADC 变量的名称为0x25,请检查是否有其它的变量名称也是定义在0x25 处并修正它。

7． WARNING 206:’DelayX1ms’: missing function-prototype  C:\8051\INPUT.C  Error 267 :’DelayX1ms ‘:requires ANSI-style prototype C:\8051\INPUT.C  说明程序中有调用DelayX1ms 函数但该函数没定义即未编写程序内容或函数已定义但未作宣告解决方法编写DelayX1ms 的内容编写完后也要作宣告或作外部宣告可在delay.h 的包含档宣告成外部以便其它函数调用。

8． \*\*\*WARNING 1:UNRESOLVED EXTERNAL SYMBOL  SYMBOL:MUSIC3  MODULE:C:\8051\MUSIC.OBJ(MUSIC)  \*\*\*WARNING 2:REFERENCE MADE TO UNRESOLVED EXTERNAL  SYMBOL:MUSIC3  MODULE:C:\8051\MUSIC.OBJ(MUSIC)  ADDRESS:0018H  说明程序中有调用MUSIC 函数但未将该函数的含扩档C 加入到工程档Prj 作编译和连接解决方法设MUSIC3 函数在MUSIC C 里将MUSIC C 添加到工程文件中去。

9． \*\*\*ERROR 107:ADDESS SPACE OVERFLOW  SPACE: DATA  SEGMENT: \_DATA\_GOUP\_  LENGTH: 0018H  \*\*\*ERROR 118: REFERENCE MADE TO ERRONEOUS EXTERNAL  SYMBOL: VOLUME  MODULE: C:\8051\OSDM.OBJ (OSDM)  ADDRESS: 4036H  说明data 存储空间的地址范围为0~0x7f,当公用变量数目和函数里的局部变量如果存储模式设为SMALL 则局部变量先使用工作寄存器R2~R7 作暂存当存储器不够用时则会以data 型别的空间作暂存的个数超过0x7f 时就会出现地址不够的现象  解决方法将以data 型别定义的公共变量修改为idata 型别的定义说明。

# 附录2 STM32F103xx模块框图



# 附录3 STM32F103xx存储映射

****

# **附录4 课程设计报告参考格式**

湖南科技大学计算机科学与工程学院

嵌入式系统课程设计报告

学 号：

姓 名：

班 级：

指导老师：

完成时间：

1. 实验标题
2. 实验目的
3. 总体设计（含背景知识或基本原理与算法，或模块介绍、设计步骤等）
4. 详细设计（含主要模块结构、程序流程图、关键代码等）
5. 实验结果与分析
6. 小结与心得体会