LED控制实验

1953729 吴浩泽

## 实验目的

掌握利用S3C2410X芯片地址总线扩展的I/O来驱动LED显示。

了解ARM芯片中利用总线扩展I/O口的使用方法。

## 实验设备

硬件：Embest EduKit-IV平台，ULINK2仿真器套件，PC机。

软件：μVision IDE for ARM集成开发环境，Windows 98/2000/NT/XP。

## 实验内容

编写程序，控制实验平台的发光二极管LED1,LED2,LED3,LED4，使它们有规律的点亮和熄灭，具体顺序如下：LED1亮->LED2亮->LED3亮->LED4亮->LED1灭->LED2灭->LED3灭->LED4灭->全亮->全灭，如此反复。

## 实验原理

在开发LED驱动之前，首先了解本实验的原理图：EduKit-IV设计了5个LED（D1～D5）用于指示和控制系统的状态，其中D2指示电源的状态，其他4个的状态是用户可编程的（SYSLED1～SYSLED4），在EduKit-IV中，这4个 LED的状态通过扩展I/O接口进行控制。

EduKit-IV LED所用到的扩展I/O如图所示：



片选信号的产生

利用3/8译码器将A18-A20扩展了7个外设片选信号CS1-CS7。CS1和CS2引出到外部扩展接口EXCON\_B3，CS3和CS4为总线扩展输入的芯片74HC541的片选。CS5，CS6，和CS7为总线扩展输出的芯片74HC573的片选。

片选信号在接入74HC573前经过了如下处理：



OLE信号的产生

其中CS5，CS6，CS7 3个片选信号和写使能信号通过74HC32或门输出一个选通信号LE为低电平。



LE信号的产生

前面或门输出的LE选通信号经过74HC04反相得到高电平后再连接到扩展输出芯片74HC573。

EduKit-IV LED接口电路如下图所示。在本实验平台上，芯片74VHC573DT的选通物理地址为0x21180000，当访问这个物理地址的时候，就可以访问其上的硬件资源了。这里可以把其理解为一个寄存器，寄存器地址是0x21180000,它的低4位控制了4个LED灯，通过访问地址为0x21180000的寄存器，往其低4位置高/低电平，从而控制相应的4个LED灯的亮/灭。

（注意：寄存器0x21180000是只写的，在软件编程时只能往里写数据，不能从里读数据）



向LED写入数据



LED1-4连接图

LED1-4这4个LED采用了共阳极的接法，分别与SYSLED1-4相连，通过SYSLED1-4引脚的高低电平来控制发光二极管的亮与灭。当这几个管脚输出高电平的时候发光二极管熄灭，反之，发光二极管点亮。

## 实验步骤

**1. 准备实验环境**

使用ULINK2仿真器连接Embest EduKit-IV实验平台的主板JTAG接口；使用Embest EduKit-IV实验平台附带的交叉串口线，连接实验平台主板上的COM2和PC机的串口（一般PC只有一个串口，如果有多个请自行选择，笔记本没有串口设备的可购买USB转串口适配器扩充）；使用Embest EduKit-IV实验平台附带的电源适配器，连接实验平台主板上的电源接口。

**2. 串口接收设置**

在PC机上运行windows自带的超级终端串口通信程序，或者使用实验平台附带光盘内设置好了的超级终端，设置超级终端：波特率115200、1位停止位、无校验位、无硬件流控制，或者使用其它串口通信程序。（注：超级终端串口的选择根据用户的PC串口硬件不同，请自行选择，如果PC机只有一个串口，一般是COM1）

**3. 打开实验例程**

1）拷贝实验平台附带光盘DISK3\_S3C2410\03-Codes\01-MDK\Mini2410-IV文件夹到MDK的安装路径：Keil\ARM\Boards\Embest\（如果本实验之前已经拷贝，可以跳过这一步）。（注：用户也可拷贝工程到任意目录，本实验为了便于教学，故统一实验路径）

2）运行μVision IDE for ARM软件，点击菜单栏“Project”，选择“Open Project…”，在弹出的对话框选择实验例程目录LED\_Test子目录下的LED\_Test.Uv2工程。

3）默认打开的工程在源码编辑窗口会显示实验例程的说明文件readme.txt，详细阅读并理解实验内容。

4）工程提供了两种运行方式：一是下载到SDRAM中调试运行，二是固化到Nor Flash中运行。用户可以在工具栏Select Target下拉框中选择在RAM中调试运行还是固化Flash中运行。如下图所示：



选择运行方式

下面实验将介绍下载到SDRAM中调试运行，所以我们在Select Target下拉框中选择LED\_Test IN RAM。

5）接下来开始编译链接工程，在菜单栏“Projiet”选择“Build target”或者“Rebuild all target files”编译整个工程，用户也可以在工具栏单击“”或者“”进行编译。

6） 编译完成后，在输出窗口可以看到编译提示信息，比如“".\SDRAM\LED\_Test.axf" - 0 Error(s), 1 Warning(s).”，如果显示“0 Error(s)”即表示编译成功。

7）拨动实验平台电源开关，给实验平台上电，单击菜单栏Debug->Start/Stop Debug Session项将编译出来的映像文件下载到SDRAM中，或者单击工具栏“”按钮来下载。

8）下载完成后，单击菜单栏Debug->Run项运行程序，或者单击工具栏“”按钮来全速运行程序。用户也可以使用进行单步调试程序。

9）全速运行后，用户可以在超级终端看到程序运行的信息。

10）用户可以Stop程序运行，使用μVision IDE for ARM的一些调试窗口跟踪查看程序运行的信息。

注：如果在第4）步用户选择在Flash中运行，则编译链接成功后，单击菜单栏Flash->Download项将程序固化到NorFlash中，或者单击工具栏按钮“”固化程序，从实验平台的主板拔出JTAG线，给实验平台重新上电，程序将自动运行。

**4. 观察实验结果**

在执行到第8）步时，可以看到超级终端上输出如下字符。观察发光二极管的亮灭情况，可以观察到的现象与前面实验内容中的相符，说明实验成功的实现了利用总线扩展I/0对LED的驱动。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* 英蓓特EduKit系列嵌入式教学系统平台 \*\*

\*\* Embest EduKit Series Embedded Teaching Platform \*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Expand I/O (Diode Led) Test Example

Please Look At The LEDS

end.

## 实验参考程序

|  |
| --- |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* File： led\_test.c  \* Author: embest  \* Desc： Led\_Test  \* History:  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*------------------------------------------------------------------------------------------\*/  /\* include files \*/  /\*------------------------------------------------------------------------------------------\*/  #include "2410lib.h"  /\*------------------------------------------------------------------------------------------\*/  /\* constant define \*/  /\*------------------------------------------------------------------------------------------\*/  #define LEDADDR (\*(volatile unsigned char\*)0x21180000) // LED Address  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* name: led\_on  \* func: turn on the leds one by one  \* para: none  \* ret: none  \* modify:  \* comment:  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void led\_on(void)  {  int i,nOut;  nOut = 0xFF;  LEDADDR = nOut & 0xFE;  for(i = 0; i < 100000; i++);  LEDADDR = nOut & 0xFC;  for(i = 0; i < 100000; i++);  LEDADDR = nOut & 0xF8;  for(i = 0; i < 100000; i++);  LEDADDR = nOut & 0xF0;  for(i = 0; i < 100000; i++);  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* name: led\_off  \* func: turn off the leds one by one  \* para: none  \* ret: none  \* modify:  \* comment:  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void led\_off(void)  {  int i,nOut;  nOut = 0xF0;  LEDADDR = nOut | 0x01;  for(i = 0; i < 100000; i++);  LEDADDR = nOut | 0x03;  for(i = 0; i < 100000; i++);  LEDADDR = nOut | 0x07;  for(i = 0; i < 100000; i++);  LEDADDR = nOut | 0x0F;  for(i = 0; i < 100000; i++);  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* name: led\_on\_off  \* func: turn on the 4 leds and then turn off the 4 leds  \* para: none  \* ret: none  \* modify:  \* comment:  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void led\_on\_off(void)  {  int i;  LEDADDR = 0xF0;  for(i = 0; i < 100000; i++);  LEDADDR = 0xFF;  for(i = 0; i < 100000; i++);  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* name: led\_test  \* func: i/o control test(led)  \* para: none  \* ret: none  \* modify:  \* comment:  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void led\_test(void)  {  uart\_printf(" Expand I/O (Diode Led) Test Example\n");  uart\_printf(" Please Look At The LEDS \n");  led\_on();  led\_off();  led\_on\_off();  delay(2000);  uart\_printf(" end.\n");  } |

## 读书笔记

一、ARM

ARM是处理器，但不是实实在的硬件。它是一种叫作“核”的东西。

1、主要设计ARM架构的RISC处理器

2、ARM公司不生产芯片，只是授权内核给生产和销售半导体的合作伙伴，另外也提供基于ARM架构的开发设计技术。

二、SOC概念

System on Chip，简称Soc,也即片上系统。从狭义角度讲，它是信息系统核心的芯片集成，是将系统关键部件集成在一块芯片上；从广义角度讲，Soc是一个微小型系统，如果说中央处理器（CPU）是大脑，那么Soc就是包括大脑、心脏、眼睛和手的系统。

三、基于Arm的处理器发展及分类

1、内核的种类

2、 按应用方向分类

2.1、MCU&FPGA方向：Cortex-M1、Cortex-M3、ARM7TDMI

2.2、Real-Time Embedded(实时嵌入式系统)： ARM7TDMI、ARM968E-S、ARM946E-S、ARM1156T2[F]-S、Cortex-R4(F/X)

2.3、Application(生活应用)：ARM926EJ-S、ARM117J[F]-S、ARM11MPCore、CortexA8、CortexA9

3、按性能、用途分类

3.1、高性能方向

TI(德州仪器)

Omap3430 Cortex-A8核600MHz （诺基亚N96）

Omap3530 Cortex-A8核600MHz ARM+DSP双核

Sansung(三星)

S5PC100 Cortex-A8核 600MHz（iphone 3GS和M9）

S5PC110/S5PV210 Cortex-A8核 1GHz

Freescale(飞思卡尔)

i.MX512/i.MX513/i.MX515 CortexA8核 600M~1GHz(上网本)

3.2、低功耗、低成本的微控制器方向

Cortex-M3

Stm32系列 36MHz~72MHz (LQFP48基本型 $1.8)

Cortex-M1

Cortex-M0

3.3、实时方向

Cortex-R4

3.4、安全方向

SecurCore

SC300 (Cortex-M3)

SC200 (ARM9)

SC100 (ARM7)

四、体系结构

（一）编程模型

1、数据和指令类型

1.1、ARM采用的是32位架构

1.2、ARM约定：Byte--->8bit

Halfword---->16bit

Word---->32bit

Doubleword---->64bit (Cortex-A处理器)

1.3、大部分ARM Core提供 ARM(32-bit)和Thumb (16-bit)两种 指令集

2、处理器的工作模式

2.1、ARM有8个基本工作模式

User非特权模式，大部分任务执行在这种模式

FIQ当一个高优先级(fast)中断产生时将会进入这种模式

IRQ当一个低 优先级(normal)中断产生时将会进入这种模式

Supervisor当复位和软中断指令执行时将 会进入这种模式

Abort当存取异常时 将 会进入这种模式

Undef当执行未定义指令时 将 会进入这种模式

System使用和User模式相同寄存器集的特权模式

Cortex-A特有模式

Monitor是为了安全而扩展出的用于执行安全监控代码的模式；也是一种特权模式

3、数据格式：字节编址，字节对齐，小端格式

字节顺序：ARM可以用little/big endian格式存取数据。

4、寄存器组：r0~r12, r13(sp), r14(lr), r15(pc), psr(cpsr, spsr)

4.1ARM共有37个32-Bits长的寄存器

1个用作PC

1个用作CPSR

5个用作SPSR

30个通用寄存器

4.2Cortex-A体系结构下有40个 32-Bits长的寄存器

比ARM多出3个寄存器：Monitor模式r13\_mon, r14\_mon, spsr\_mon

4.3当前处理器的模式决定着哪组寄存器可操作，任何模式都可以存取：

相应的r0-r12子集

相应的r13（sp）和r14（lr）

相应的r15（pc）

相应的CPSR

4.4特权模式（除system模式）还可以存取

相应的spsr

5、程序状态寄存器：NZCV(标识计算结果),I,F,T, M[4..0]

N：负数标志

Z：零标志

C：借位或进位标志

V：溢出标志

I：IRQ中断禁止位

F：FIQ 中断禁止位

T：处理器处于ARM状态或Thumb（ThumbEE）状态

M[4...0]：处理器工作模式

6、程序计数器PC：ARM时4字节对齐（值都是4的倍数，指令32位长）， THUMB时2字节对齐

6.1处理器执行在ARM状态

所有指令为32bits宽

所有指令必须word对齐

pc值由bits[31:2]决定， bits[1:0]未定义（所以指令不能halfword/byte对齐）

6.2处理器执行在Thumb状态

所有指令为16bits宽

所有指令必须 halfword对齐

pc值由bits[31:2]决定， bits[1:0]未定义（所以指令不能byte对齐）

6.3处理器执行在Jazelle状态

所有指令8bits宽

处理器执行word存取一次取4条指令

7、异常向量：事件发生时，PC值的固定跳转位置

0x00 -reset

0x04 -und

0x08 -swi

0x0c -prefetch about

0x10 -data about

0x14 -rsvd

0x18 -irq

0x1c -fiq

当异常产生时，ARM core：

拷贝CPSR到SPSR\_<mode>；

设置适当的CPSR位：

改变处理器状态进入ARM态

改变处理器模式进入相应的异常模式

设置中断禁止相应中断（如果需要）

保存返回地址到LR\_<mode>

设置PC为相应的异常向量

返回时，异常处理需要：

从SPSR\_<mode>恢复CPSR

从LR\_<mode>恢复PC

Note:这些操作只能在ARM态执行

（二）、指令集

ARM指令32位长

大部分为单周期指令

都可以条件执行

Load/Store结构

（三）、系统设计

CPU - RAM - ROM

五、处理器内核

1)ARM7TDMI：

三级流水线结构

PC值偏移：指向取指的内存位置，而不是正在执行的指令（正在执行的指令已经在ALU中）

流水线分析：最佳流水线结构，流水线中断，分支流水线，中断流水线

字节对齐：单字节对齐，双字节对齐，指令都是32位的，所以每次读一条指令即读取4字节

2)ARM9

5级流水线互锁：产生条件，解决方案

3)ARM10

4)ARM11

5)ARMV6

6)ARMV7

六、汇编语言

1)编程学习环境

Keil RealView MDK V4.22a

文件格式：c文件，s文件，h文件

工程建立，文件添加，编译，进入/退出仿真

2)汇编文件格式

段定义，入口函数定义，文件结束

area reset, code, readonly

entry

reset

mov r0,#1

b .

end

伪指令

adr,adrl,ldr, nop

3)指令集

3.1、数据处理指令 ARM内核的数据计算，只能处理寄存器里的数据，指令机器码能理解最好，不理解对指令的功能学习不会有 影响

数据传送：mov,mvn,msr,mrs

算术运算：add,adc,sub,sbc,rsb,rsc

逻辑运算：and,orr,eor,bic

比较指令：cmp,cmn

测试指令：tst,teq

乘法：mul,

立即数：必须可使用imma[7..0] ror (rot \* 2)进行表达

3.2、装载，存储指令 在寄存器和存储器间进行数据交换

ldr, ldrb, ldrh, ldrsb, ldrsh， 带符号装载只在半字和字节有效，需要作符号位扩展，保证装载的数符号位不变。

str, strb, strh

获得地址：[基地址+偏移]

基地址：ldr r0, =buf

偏移值：可以立即数（12-bit），可以是寄存值，也可以寄存值+预偏移（移位）

预偏移：地址值先加，buf[0x12]，如果需要更新基址，指令后面加‘！’

后偏移：先取值，后更新地址

3.3、跳转指令 控制程序流程,32MB的直接跳转范围（原因是什么？）

b 直接跳转

bl 带链接跳转，保存PC到LR中，用于子程序调用

bx 带模式切换的跳转，会修改CPSR里的M[4..0]

3.4、程序状态寄存器存取 修改PSR值，改变模式，禁止或打开IRQ/FIQ等

mrs r0, cpsr

msr r0, cpsr

3.5、协处理器 控制协处理器动作

3.6、异常产生指令 软件中断

七、汇编指令

1、LDM/STM

4种基址调整方式，IA，IB，DA，DB

内存拷贝

子程序调用时，用作堆栈处理

2、SWP交换数据

ldr

str

3、SWI软件中断

主要应用于操作环境下的系统调用，例如在x86下的int 86指令

处理过程类似于IRQ或FIQ的处理过程

lr = pc + 4 保存swi下一条指令地址，以便处理完成可以正确返回

pc = 0x08 跳转到SWI中断向量中

实现：计算得到系统调用号，然后调用C语言中的do\_swi函数，实现系统的调用

4、PSR传送

清除I位操作（IRQ中断控制位）

mrs r0, cpsr

and r0, r0, #0xffffff7f

msr cpsr\_c, r0

5、寻址方式：找到变量的方法

常量值：#define PI 3.14

变量值：

寄存器

内存

全局变量：全局数据区，根据指针访问

局部变量：存放于堆栈

模块变量：函数里定义的static变量，文件里定义的static变量

6、GNU伪指令

ldr

.text

.data

.bss

.align

.global/.globl ---> extern int g\_val;

.equ sym value ---> #define sym value

7、FIQ处理为什么比IRQ更快

独立寄存器

优先级

中断向量位置，可以省去一条跳转指令

8、系统复位，start.s

设置中断向量

[看门狗关闭]

[系统倍频，PLL设置]

[设置内存，DRAM]

[拷贝代码到DRAM]

[控制中断状态]

设置各个工作模式下的堆栈（SP），建立C语言运行环境

跳转到main函数中执行

9、混合编程，ATPCS

R0 ~ R3 用于保存第0~第3个参数，其它的参数放到堆栈中

R4 ~ R12用于作为局部变量，除了R12，如果有使用的话，必须先压栈保存，函数退出后恢复

R0用于保存返回值

10、内联汇编