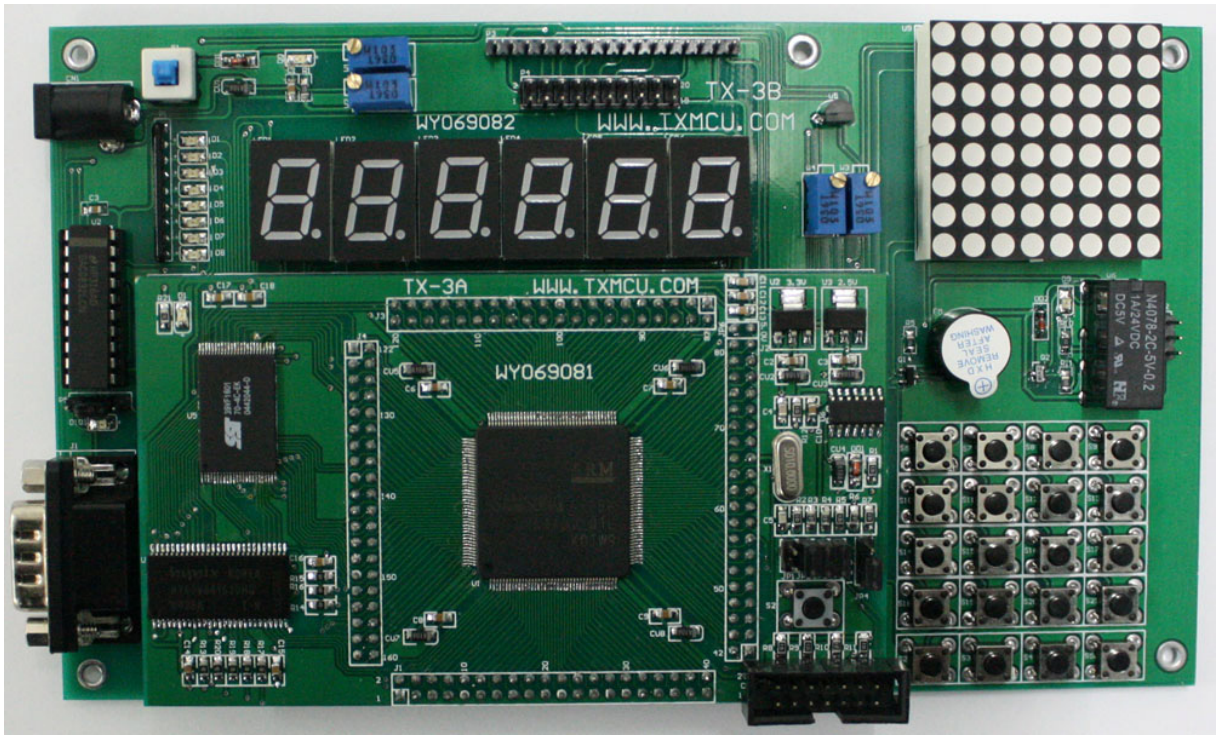
TX-3A 44B0 开发板使用手册



V1.0(2008.12.15)

哈尔滨天祥电子有限公司

网站：www.txmcu.com

论坛： bbs.txmcu.com

目录

第一章 开发板介绍

1.1 处理器介绍

Samsung(三星)公司推出的16/32位RISC处理器S3C44B0X为手持设备和工业类控制提供了一种高性能低成本的解决方案。为了降低整个系统的成本，S3C44B0X 内部集成了丰富的片内外设，包括：8K的cache、可选的片内SRAM、LCD控制器、带有握手信号的双通道UART、4通道DMA控制器、系统管理器(片选逻辑、FP/EDO/SDRAM控制器)、带有PWM功能的5通道定时器、I/O 端口、RTC 实时时钟、8通道10位ADC、IIC、IIS总线接口、同步SIO接口以及用于时钟管理的PLL锁相环。

S3C44B0X采用了ARM7TDMI内核，0.25um标准宏单元和存储编译器。TDMI的每一个字母代表一种功能：

- T(Thumb)，支持16位压缩指令集Thumb。
- D(Debug)，支持片上Debug。
- M(Multiplier)，内嵌硬件乘法器。
- I(ICE)，嵌入式ICE，支持片上断点和调试。

S3C44B0X极低的功耗以及低廉的成本使其非常适合对成本和功耗敏感的项目。同时S3C44B0X还采用了一种新的总线结构，即SAMBAII(三星ARM CPU 嵌入式微处理器总线结构)，S3C44B0X通过集成全面、通用的片内外设，大大减少了系统电路中除处理器外的器件需求，从而最小化系统成本。下列是其集成的片内外设：

- 外部存储器控制器(FP/EDO/SDRAM 控制器，片选逻辑)。
- LCD控制器(最大支持256色STN，LCD 具有一个专用的DMA通道)。
- 2个通用DMA通道，2个外设DMA通道并有片外请求管脚。
- 2 个 UART，带有握手协议(支持 irDA1.0，具有 16 字节 FIFO)。

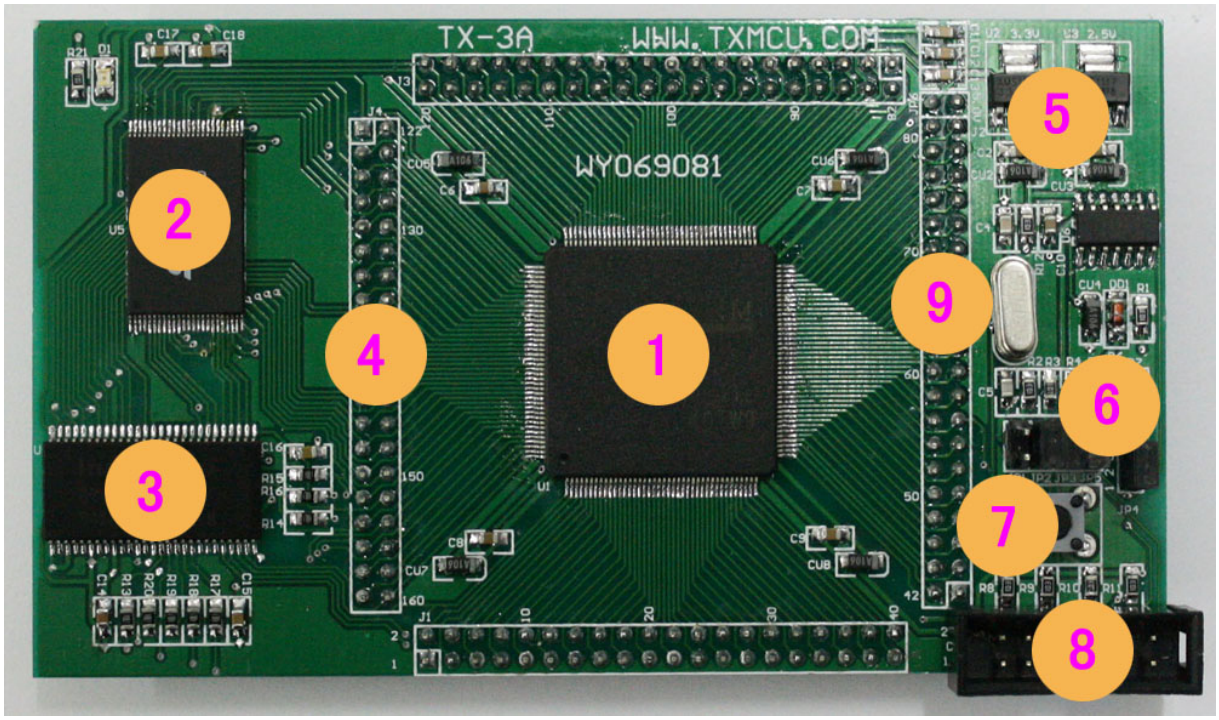
- 1 通道 SIO。
- IIC 多主总线接口。
- IIS 总线控制器。
- 5 个 PWM 定时器和 1 通道内部定时器。
- 看门狗定时器。
- 71 个通用 I/O 口，最多支持 8 个片外中断源。
- 功耗管理：普通、慢速、空闲和停止模式。
- 8 通道 10 位 ADC。
- 具有日历功能的 RTC。
- 带 PLL 的片内时钟发生器

使用 S3C44B0X 来构建系统，能够降低整个系统的成本，我们列举以下一些可以采用 S3C44B0X 构建的系统：

- GPS 电话。
- PDA(个人数字助理)。
- 掌上游戏机。
- 指纹识别系统。
- 终端汽车导航系统。
- 工业控制应用。
- MP3、MP4 等手持设备应用。

1.2 TX-3A 主板介绍

主板依据三星公司提供的评估板的核心部分设计而来，其各部分介绍如下：



1、 核心控制芯片S3C44B0。

2、 2M字节FLASH SST39VF1601。

3、 8M字节SDRAM HY57V641620。

4、 160个引脚全部引出，方便外扩实验。

5、 AMS1117-3.3和AMS1117-2.5 电压转换芯片。

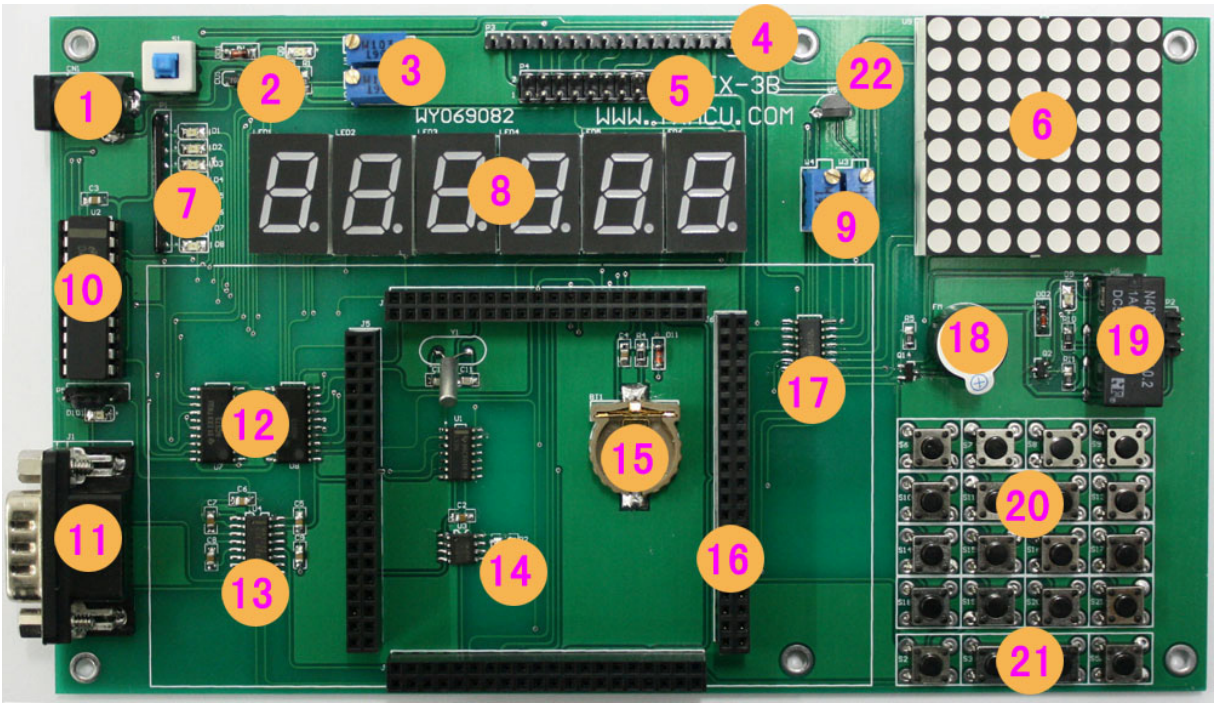
6、 大小端及总线位数选择跳线。

7、 系统复位按键。

8、 14PIN JTAG调试下载接口。

9、 主板10M晶振。

1.3 TX-3B 入门级扩展板介绍



1、 5V直流电源接口。

2、 防反接保护二极管。

3、 1602液晶和12864液晶对比度调节电位器。

4、 1602液晶接口。

5、 12864液晶接口。

6、 8X8共阴极点阵。

7、 8个发光二极管。

8、 6个单体共阴极数码管。

9、 两路A/D模数转换模拟输入端。

10、 数模转换器DAC0832。

11、 与计算机通信用的串口。

12、 两个锁存器74HC573。

13、 RS232/TTL电平转换芯片MAX232。

14、 IIC总线EEPROM芯片AT24C02。

15、 RTC时钟外接电池座。

16、 与核心板对接插座。

17、 串入并出移位寄存器74HC595，用来驱动点阵。

18、 蜂鸣器。

19、 双路继电器。

20、 4X4矩阵键盘。

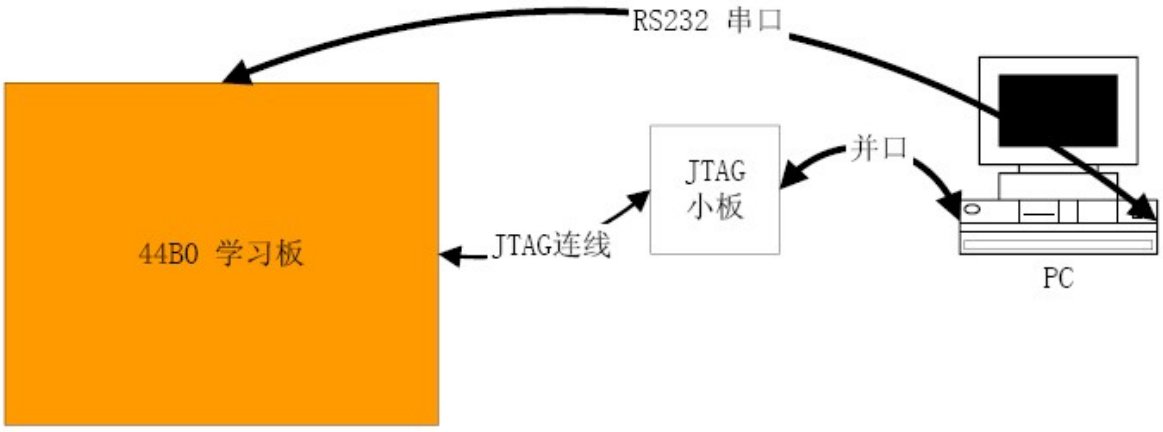
21、 4个独立按键。

22、 单总线数字温度传感器DS18B20。

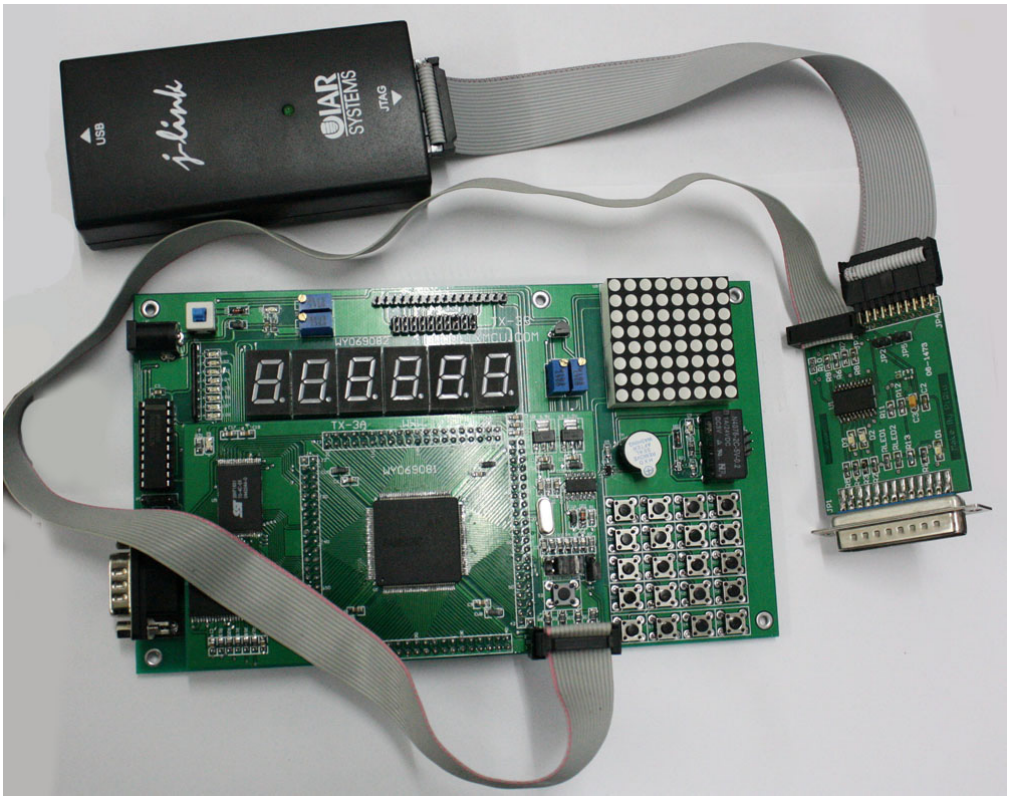
第二章 硬件测试

2.1 硬件连接示意

硬件连接示意图：



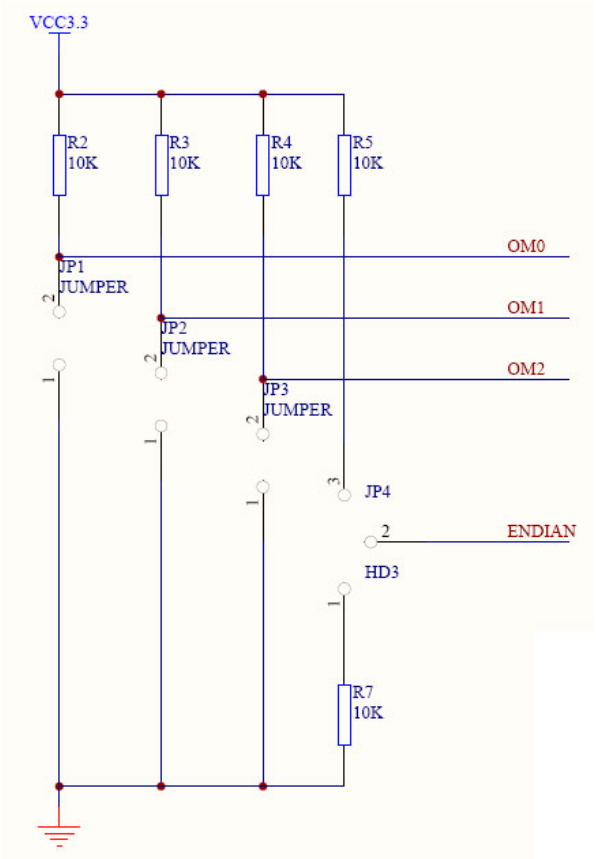
如果用户计算机没有并口的话，请用我公司的 ARM JLINK 全功能 USB 仿真器连接 JTAG 小板来做测试，连接方法见下图。关于这部分详细内容请大家参考随机光盘中的视频部分。



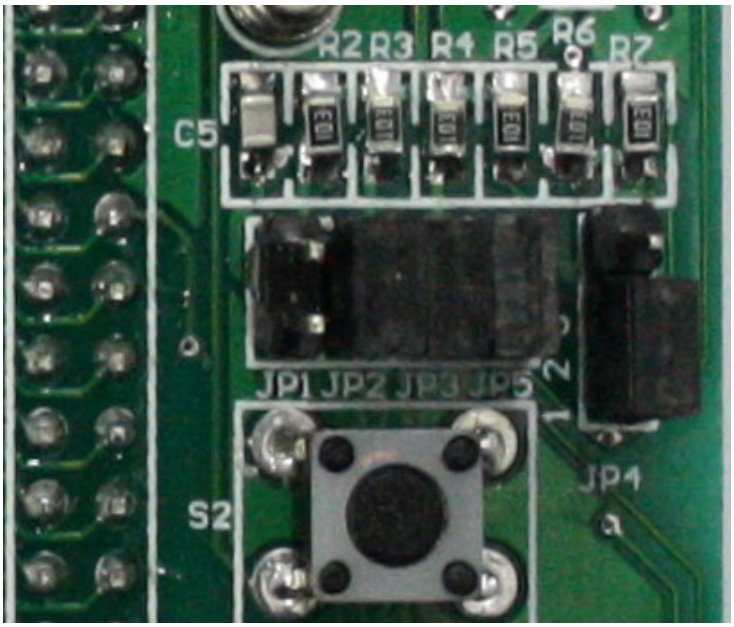
2.2 开发板上跳线设置

（1）主板上跳线设置

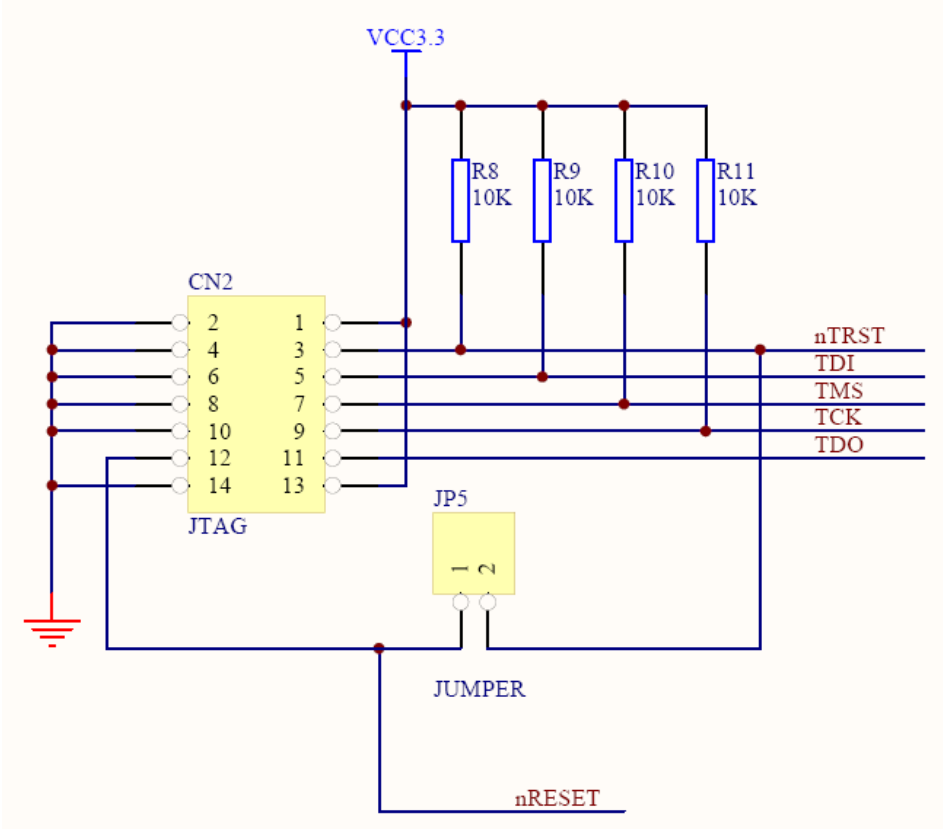
主板上主要是设置 ARM7 的大小端模式和数据位宽度，这里我们设置小端模式，16 位数据宽度，原理图如下图所示：



开发板上实际连接方式如下图：

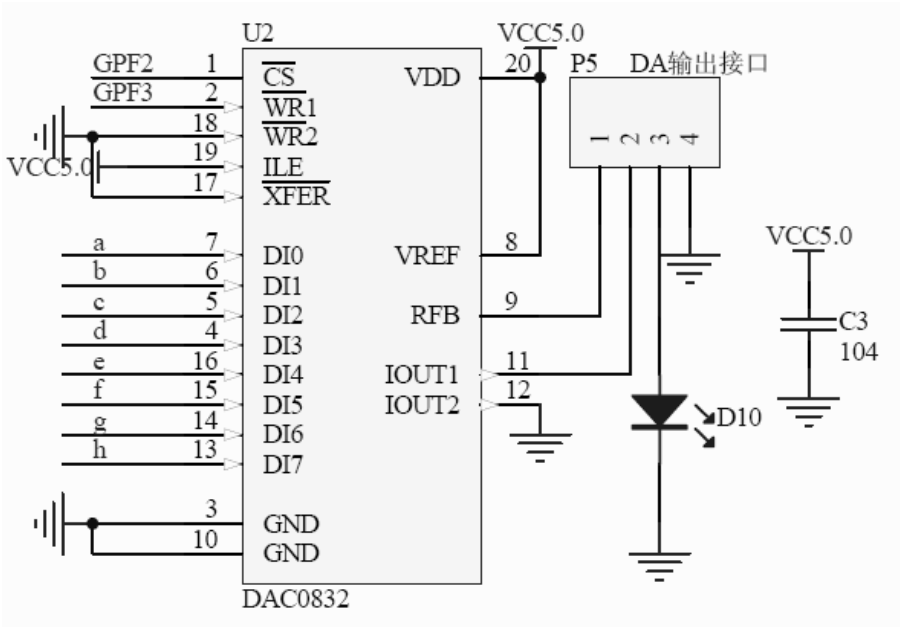


其中 JP5 为 JTAG 部分跳线，原理图如下图所示：

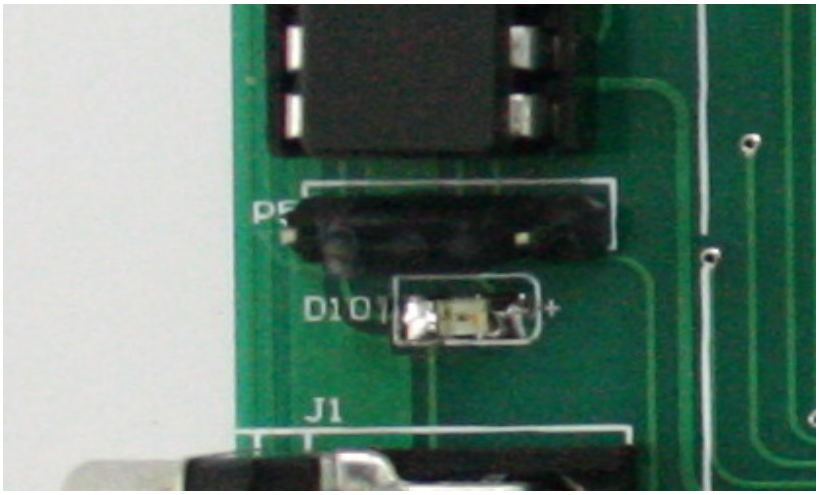


（2）底板上跳线设置

底板上跳线只有一个，就是 DAC0832 的模拟信号输出端与发光二极管小灯的短接，原理图上如下图所示：



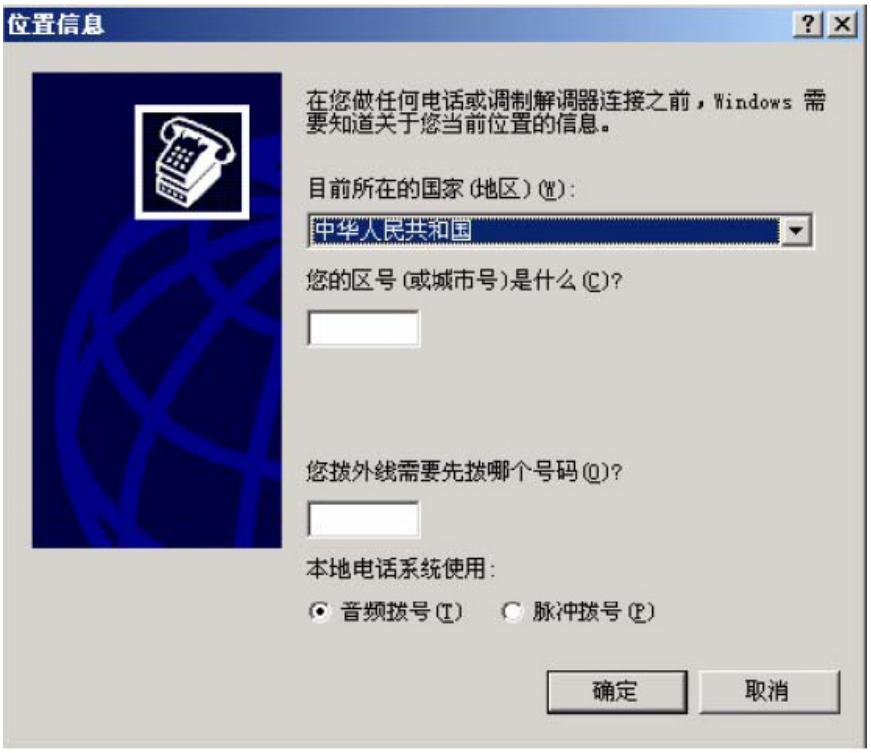
开发板上实际连接方法如下图所示,跳接四个针中间的两个就可以：



2.3 超级终端建立

打开 PC 机的超级终端软件（注：Windows 自带了超级终端软件，当然用户也可以使用其他超级终端软件，比如光盘中的SecureCRT 就是非常好用的一种），我们这里以 Windows 自带的超级终端为例，点击[开始]->[程序]->[附件]->[通讯]->[超级终端]

首次运行这个超级终端的时候将弹出以下设置界面:



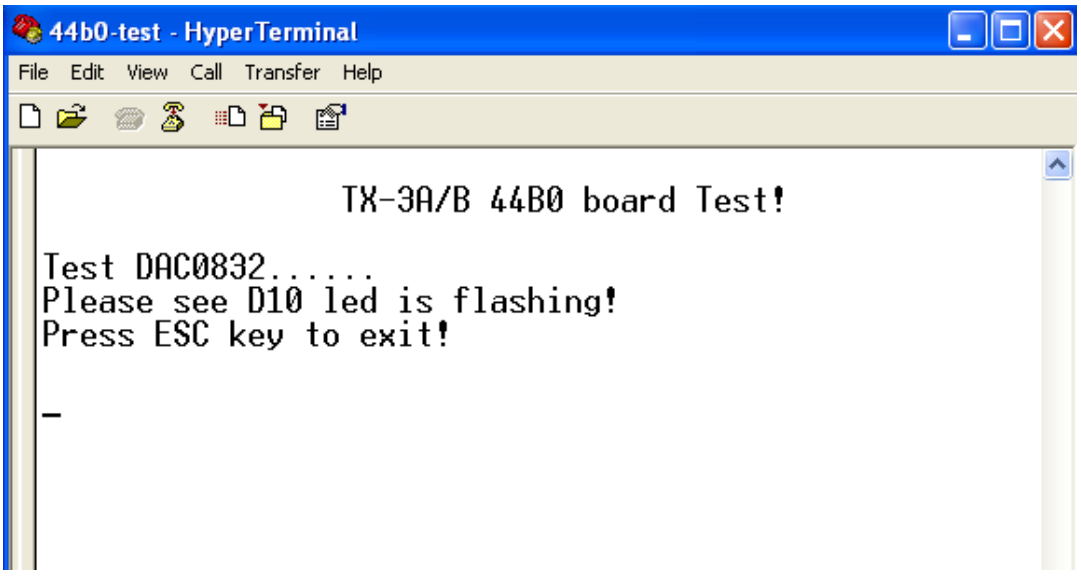
在区号对话框中随意输入一个区号：比如 0451, 点击确定，再随后的对话框中再次点击确定，继续将弹出下图的设置窗口，在这里我们将给我们

新建的链接取一个名字，比如 S3C44B0，然后点击确定继续。

接下来的画面显示的是需要设置对应的串口号，如果是台式机通常就是COM1。如果是笔记本用户，请大家查看设备管理器，然后选择好对应的串口，这里一定要注意，务必要选择好正确的串口号，否则开发板与计算机将无法连接。

点击确定后，接下来我们将对超级终端的各种参数进行详细配置，下图中，我们设置的参数是：波特率115200，8位数据，1位停止位，无奇偶校验，无数据流控制。

本系统配备的电源是 5V，1A 的开关型直流电源，系统上电后，核心板和底板上的两个电源指示灯都应该点亮，核心板的Flash中已经烧录了整板测试程序，如果串口连接正常，超级终端界面将显示下图所示：



接下来大家可按照随机光盘中所带的视频教程中讲解的方法依次测试所有的测试代码。

第三章 软件安装及介绍

3.1 软件安装方法

软件安装的详细方法请大家参考随机光盘中的视频讲解。

3.2 ADS 1.2 集成开发环境

ADS 集成开发环境是ARM 公司推出的ARM 核微控制器集成开发工具，英文全称为ARM Developer Suite，成熟版本为ADS1.2。ADS1.2 支持ARM10之前的所有ARM 系列微控制器，支持软件调试及JTAG 硬件仿真调试，支持汇编、C、C++源程序，具有编译效率高、系统库功能强等特点，可以在Windows98、Windows XP、Windows2000 以及RedHat Linux上运行。

这里将简单介绍使用ADS1.2 建立工程，编译连接设置，调试操作等等。

3.3 ADS 1.2 集成开发环境的组成

ADS 1.2 由6 个部分组成，如表3.1 所示

表3.1 ADS 1.2 的组成部分

名称	描述	使用方式
代码生成工具	ARM 汇编器， ARM 的C、C++编译器， Thumb 的C、C++编译器， ARM 连接器	由CodeWarrior IDE 调用
集成开发环境	CodeWarrior IDE	工程管理，编译连接
调试器	AXD， ADW/ADU， armsd	仿真调试
指令模拟器	ARMulator	由AXD 调用
ARM 开发包	一些底层的例程， 实用程序(如fromELF)	一些实用程序由CodeWarrior IDE 调用
ARM 应用库	C、C++函数库等	用户程序使用

由于用户一般直接操作的是CodeWarrior IDE 集成开发环境和AXD 调试器，所以这里我们只介绍这两部分软件的使用，其它部分的详细说明参考ADS 1.2 的在线帮助文档或相关资料。

3.4 CodeWarrior IDE 简介

ADS 1.2 使用了CodeWarrior IDE 集成开发环境，并集成了ARM 汇编器、ARM 的C/C++编译器、Thumb 的C/C++编译器、ARM 连接器，包含工程管理器、代码生成接口、语法敏感(对关键字以不同颜色显示)编辑器、源文件和类浏览器等等。CodeWarrior IDE 主窗口如图3.1 所示。

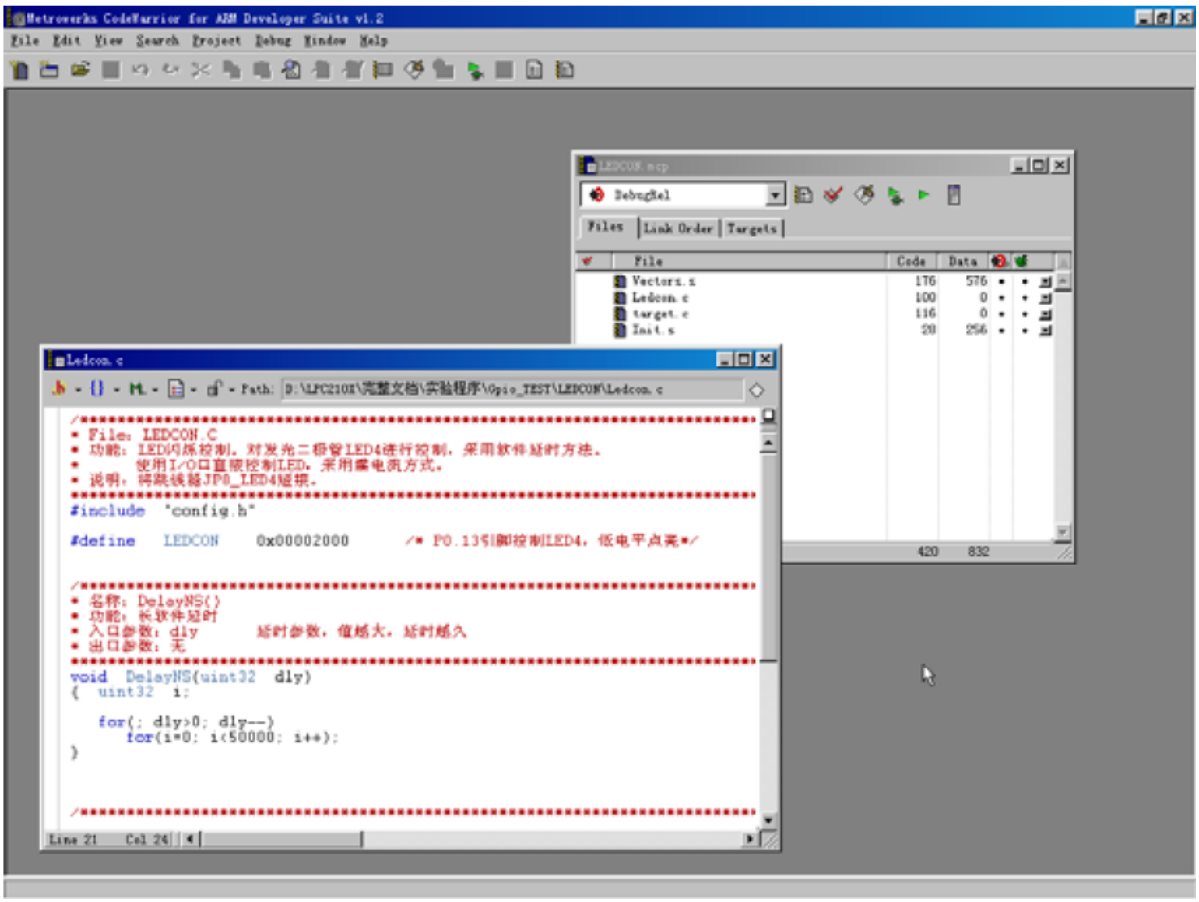


图3.1 CodeWarrior 开发环境

3.5 AXD 调试器简介

AXD 调试器为ARM 扩展调试器(即ARM eXtended Debugger)，包括ADW/ADU 的所有特性，支持硬件仿真和软件仿真(ARMulator)。AXD 能够装载映像文件到目标内存，具有单步、全速和断点等调试功能，可以观察变量、寄存器和内存的数据等等。AXD 调试器主窗口如图3.2 所示。

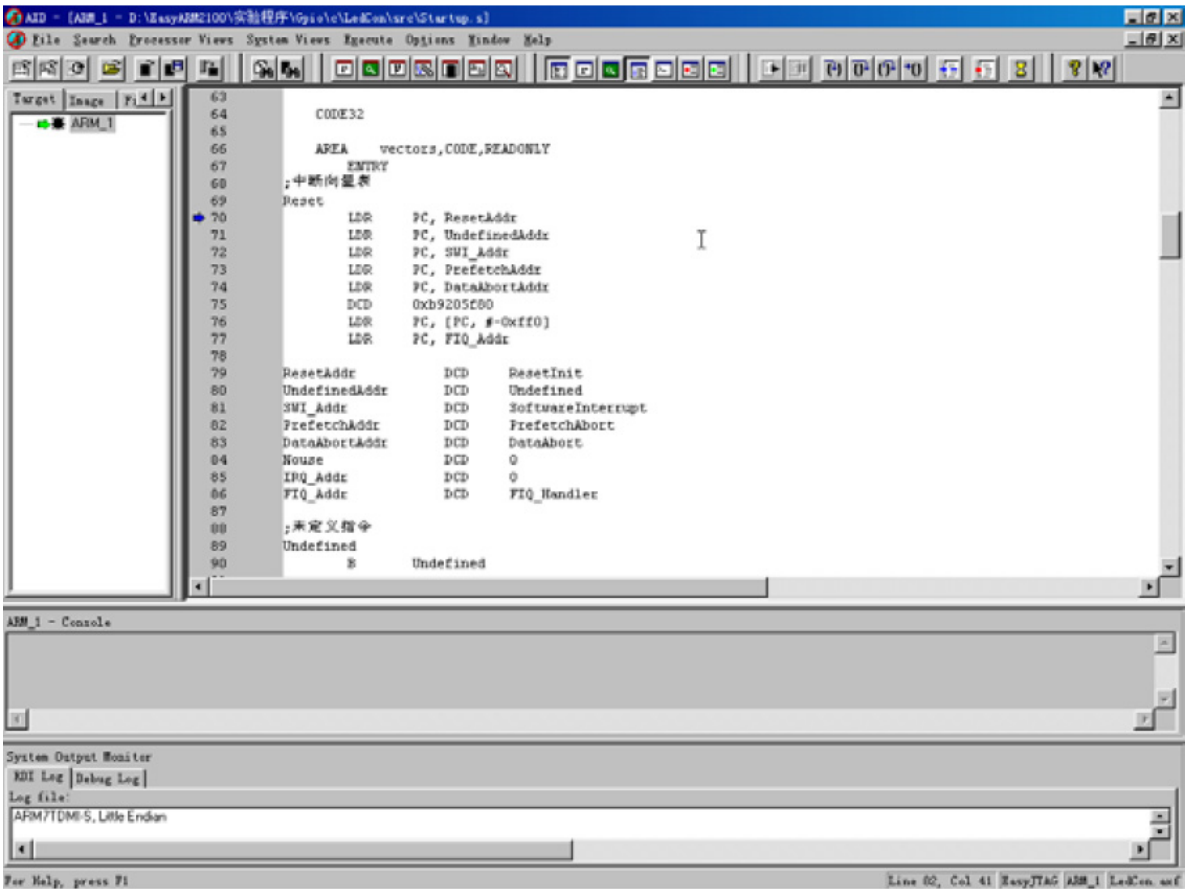

AXD - [ARM_1 - D:\EasyARM2100\实验程序\Gpio\e\LedCon\src\Startup.s]
File Search Processor Views System Views Execute Options Window Help
Target Image
ARM_1
63
64 CODE32
65
66 AREA vectors,CODE,READONLY
67 ENTRY
68 ;中断向量表
69 Reset
70 LDR PC, ResetAddr
71 LDR PC, UndefinedAddr
72 LDR PC, SWI_Addr
73 LDR PC, PrefetchAddr
74 LDR PC, DataAbortAddr
75 DCD 0xb9205f80
76 LDR PC, [PC, #-0xff0]
77 LDR PC, FIQ_Addr
78
79 ResetAddr DCD ResetInit
80 UndefinedAddr DCD Undefined
81 SWI_Addr DCD SoftwareInterrupt
82 PrefetchAddr DCD PrefetchAbort
83 DataAbortAddr DCD DataAbort
84 Nouse DCD 0
85 IRQ_Addr DCD 0
86 FIQ_Addr DCD FIQ_Handler
87
88 ;未定义指令
89 Undefined
90 B Undefined
ARM_1 - Console
System Output Monitor
RDI Log
Debug Log
Log file:
ARM7TDMI-S, Little Endian
For Help, press F1
Line 82, Col 41
EasyJTAG
ARM_1
LedCon.mcf

图3.2 AXD 调试器

第四章 工程建立

4.1 配置 ADS 集成开发环境

（1）运行 ADS 1.2 集成开发环境（CodeWarrior for ARM Developer Suite）。选择 File｜New 命令，在对话框中选择 Project 选项卡，如图 4-1 所示，新建一个工程文件。其中示例的工程名为 Exp6.mcp。单击 Set 按钮可为该工程选择路径，如图 4-2 所示，选中 Create Folder 复选框后将以图 4-1 中的 Project name 或图 4-2 中的文件名作为创建目录的名称，这样可以将所有与该工程相关的文件放到该工程目录下，便于管理工程。

注意：在图 4-2 工程模板列表中选择 ARM Executable Image 通用模板。随后将一步一步地把它配置成针对 TX-3A 开发板的模板 44B0 ARM Executable Image，并把它复制到 ADS1.2 安装目录下的 Stationery 目录中（所有的工程模板都在此目录下）。以后新建工程时，在工程模板列表中直接选中 44B0 ARM Executable Image 模板选项即可，不必每次重新配置模板。

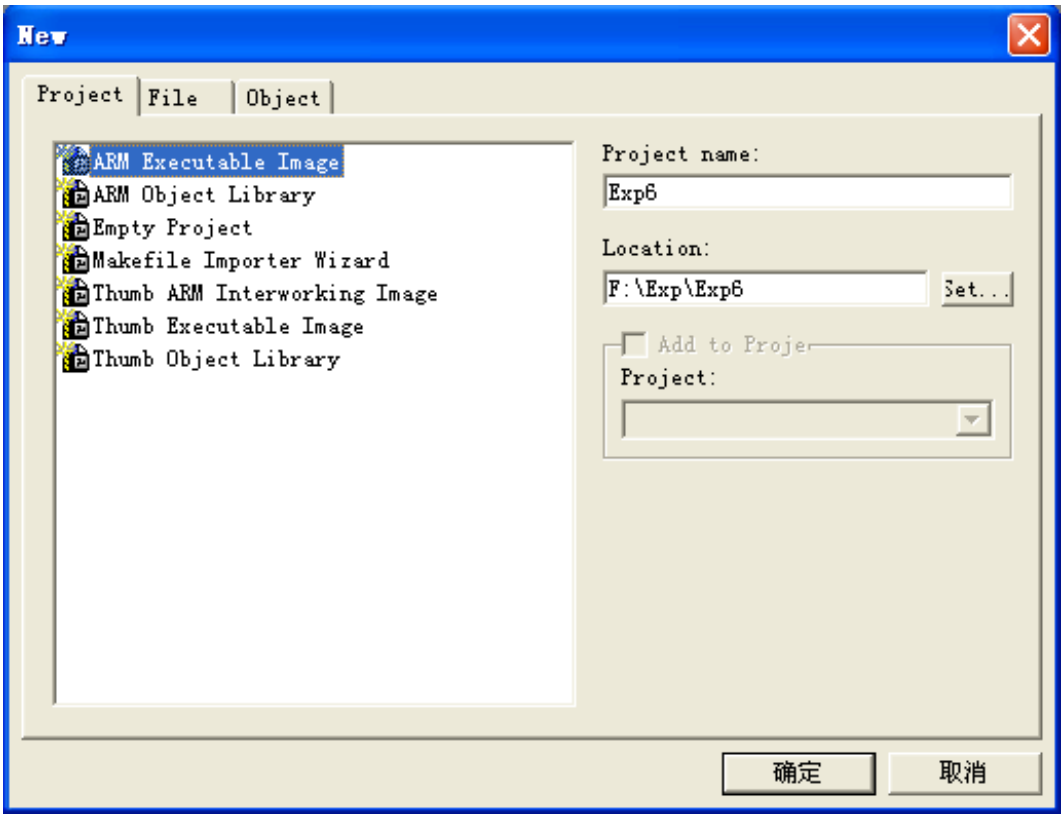


图 4-1 新建工程

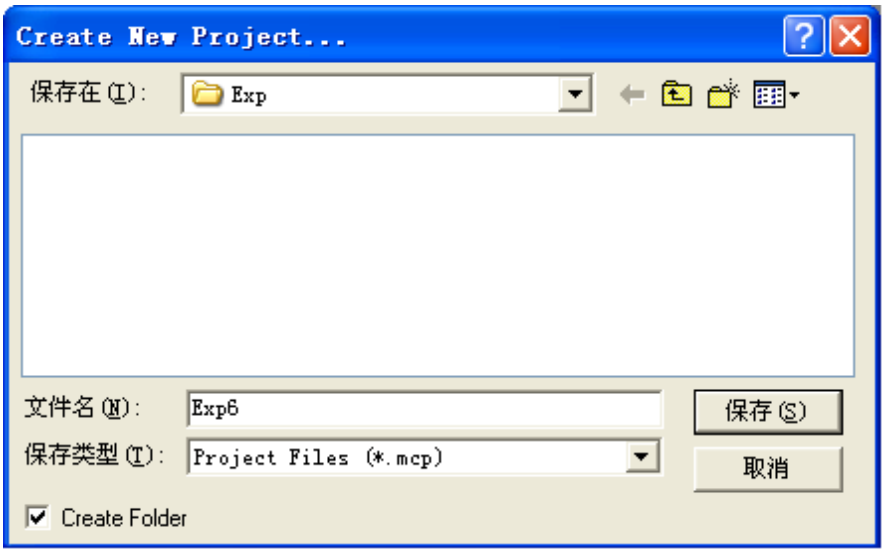


图 4-2 保存工程

（2）在新建的工程中，如图 4-3 所示，选择 DebugRel 版本，使用 Edit|Debug RelSettings 命令对 DebugRel 版本进行参数设置。

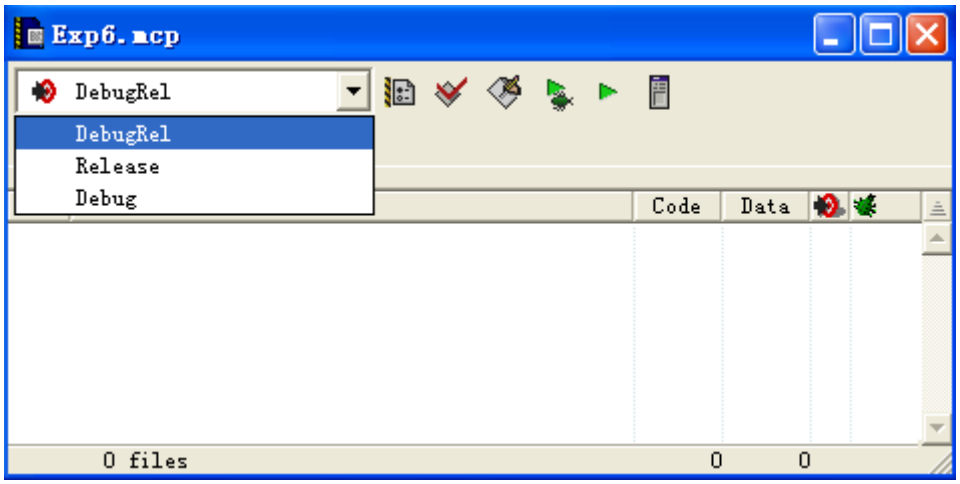


图 4-3　选择版本

（3）在 DebugRel Settings 对话框中选择 Target Settings 选项，如图 4-4 所示。在 Post-linker 列表框中选择 ARM fromELF，单击右下角的 Apply 按钮使其有效。

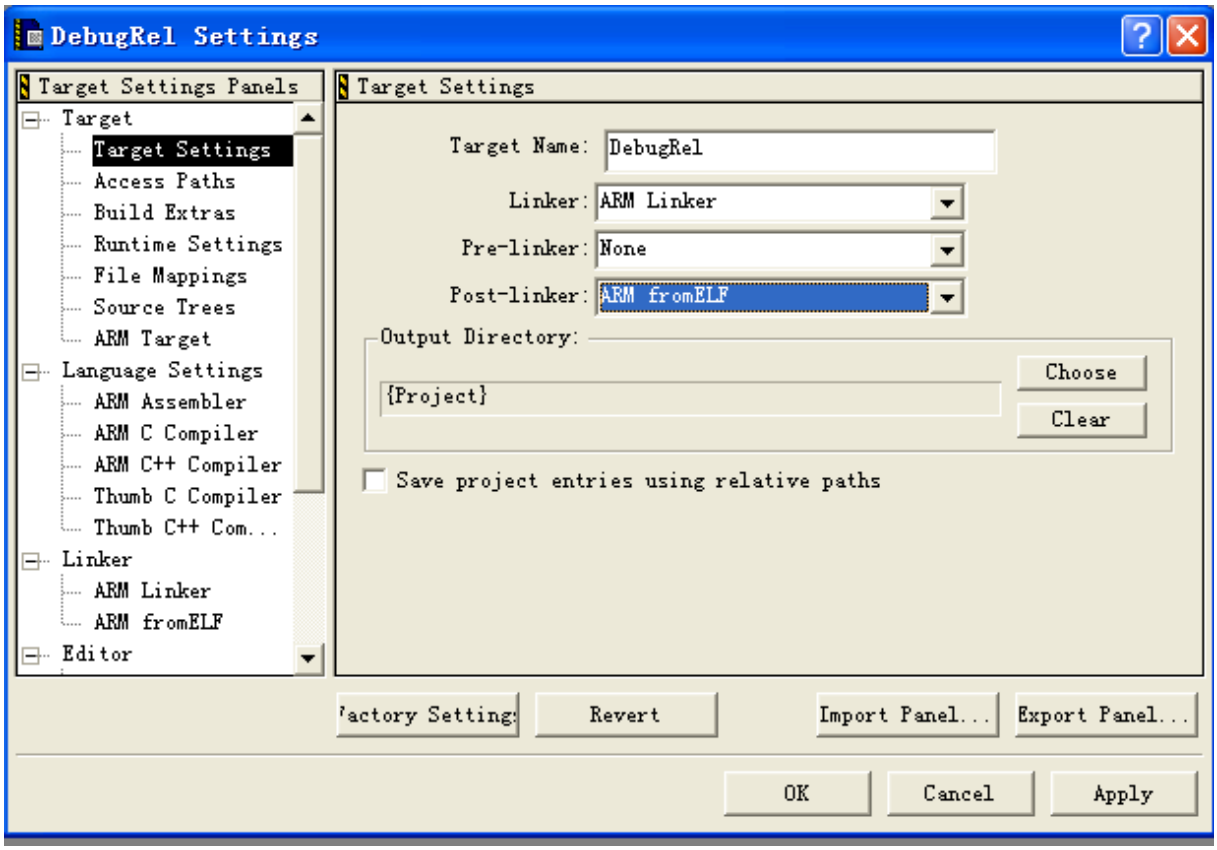


图 4-4　选择 Target Settings

（4）在 DebugRelSettings 对话框中选择 ARM Linker 选项，如图 4-5 所示。在 Output 选项卡的 Linktype 选项组中有 3 种类型的连接方式，常用的是 Simple 和 Scattered 两种。Simple 是一种简单设置，我们在这里选中 Simple 单选按钮，并按照下面的步骤进行设置。

（5）选中 Simple 单选按钮，如图 4-5 所示。在 Simple image 选项组中设置连接的 ReadOnly（只读）和 Read-Write（读写）地址。地址 0x0c008000 是在开发板上 SDRAM 的真实地址（从 0x0c000000 起始）范围内，是由系统的硬件决定的；后面的 RW Base 不用写，或写成 0x00000000 也行。

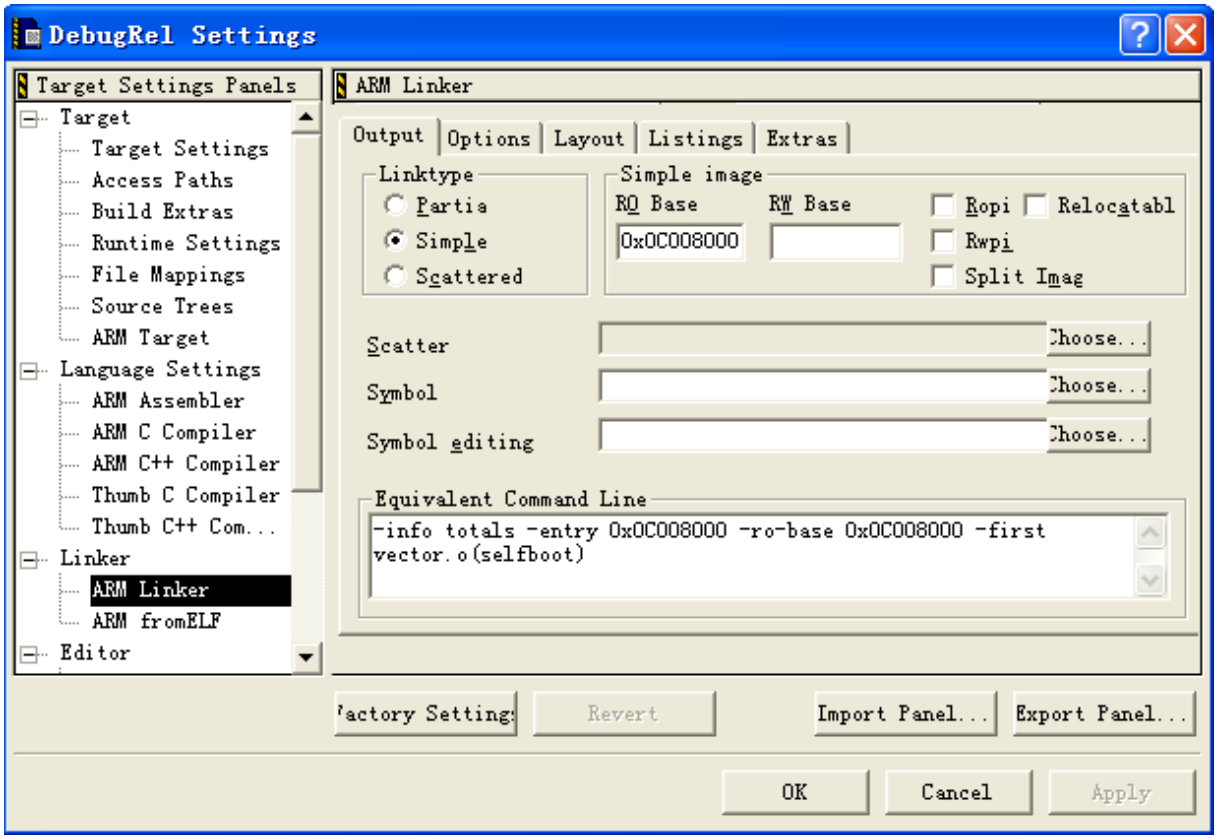


图 4-5 设置连接地址范围

提示：

① 程序移植到 ADS 后，首先执行用汇编语言写的初始化代码，包

括中断向量和内存空间的初始化。在该段代码中使用

```
IMPORT  _ _main;（注意 main 前面是两个半字下划线）
B     _ _main
```

进行系统内部的标准 C 函数初始化，然后调用用户在 C 语言中定义的 main()函数（注意：两个 main 都是小写），并且在嵌入式应用中用户在 C 语言中定义的 main 函数中不能有参数（int main(void)）。

② 不能有系统定义的软中断，在汇编语言中可以使用

```
IMPORT_ _use_no_semihosting_swi
```

来检测，在 C 语言中使用

```
#pragma import(_ _use_no_semihosting_swi)  //
ensure no functions that use semihosting
```

（6）在 DebugRel Settings 对话框中选择 ARM Linker 选项，如图 4-6 所示。单击 Layout 选项卡。在 Layout 选项卡的 Place at beginning of image 选项组中设置程序的入口模块。TX-3A 开发板在生成的代码中，程序是从 vector.s 开始运行的，vector.s 经过编译生成 vector.o 文件，所以这里在 Object/Symbol/ 项里设为 vector.o，Section 项设为 selfboot。关于这段代码大家可打开 TX-3A 配套代码中的 vector.s 文件查看如下代码段：

```
;*************************************************
    AREA   SelfBoot,   CODE,  READONLY
    IMPORT    UDF_INS_VECTOR
    IMPORT    SWI_SVC_VECTOR
    IMPORT    INS_ABT_VECTOR
    IMPORT    DAT_ABT_VECTOR
    IMPORT    IRQ_SVC_VECTOR
    IMPORT    FIQ_SVC_VECTOR
    ENTRY
```

```
    IF :DEF: |ads$version|
    ELSE
    EXPORT    __main
__main
    ENDIF
ResetEntry
    b   SYS_RST_HANDLER
    b   UDF_INS_HANDLER
    b   SWI_SVC_HANDLER
    b   INS_ABT_HANDLER
    b   DAT_ABT_HANDLER
    b   .
    b   IRQ_SVC_HANDLER
    b   FIQ_SVC_HANDLER
;*******************************************************
```

上面代码中“AREA SelfBoot, CODE, READONLY”此行中有个 selfboot,这个字母组合必须与图 4-6 中的 Section 中一致，因为这是总程序的入口处。

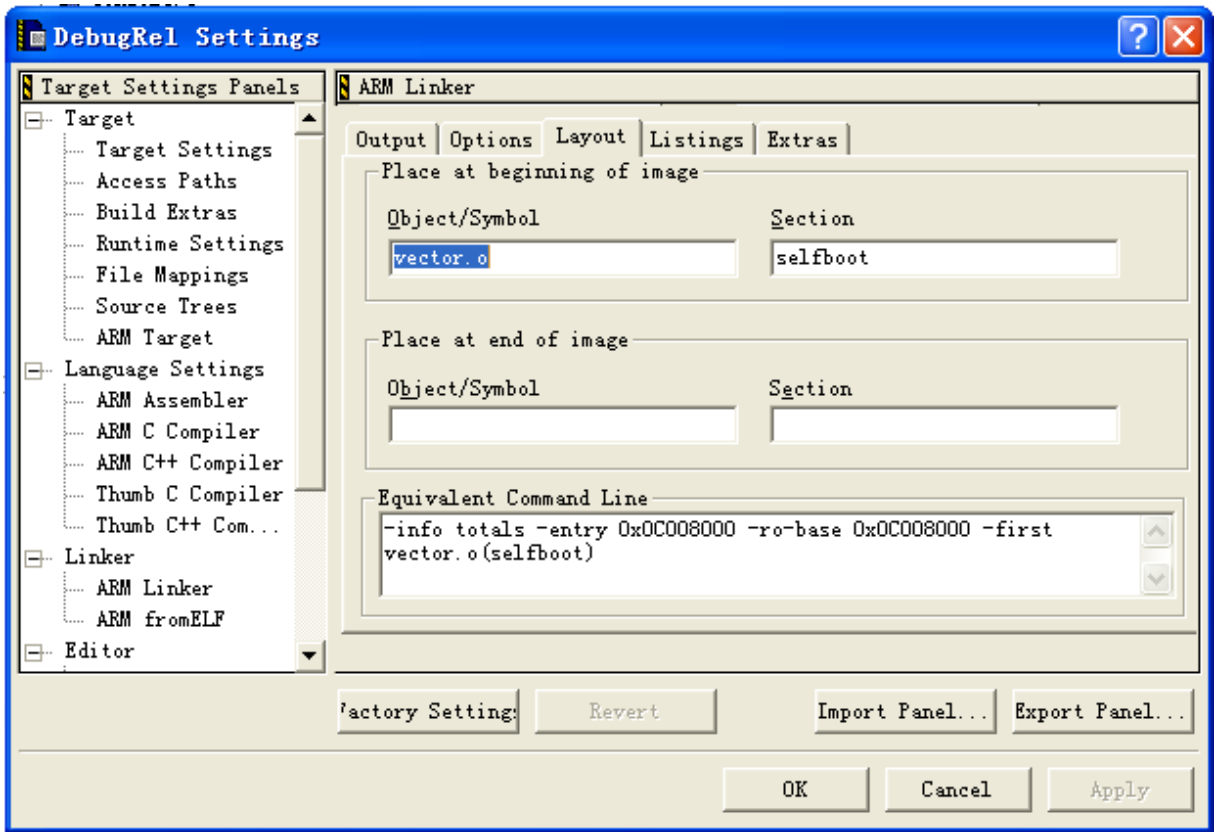


图 4-6 设置程序入口

（7）在 DebugRel Settings 对话框中选择 ARM fromELF 选项，如图 4-7 所示。在 Output format 框中设置输出文件格式为 Intel 32 bit Hes,在 Output file name 框中可随意设置输出文件名，在这里我们设置为 led.hex，这就是要下载到开发板上 flash 中的十六进制文件。

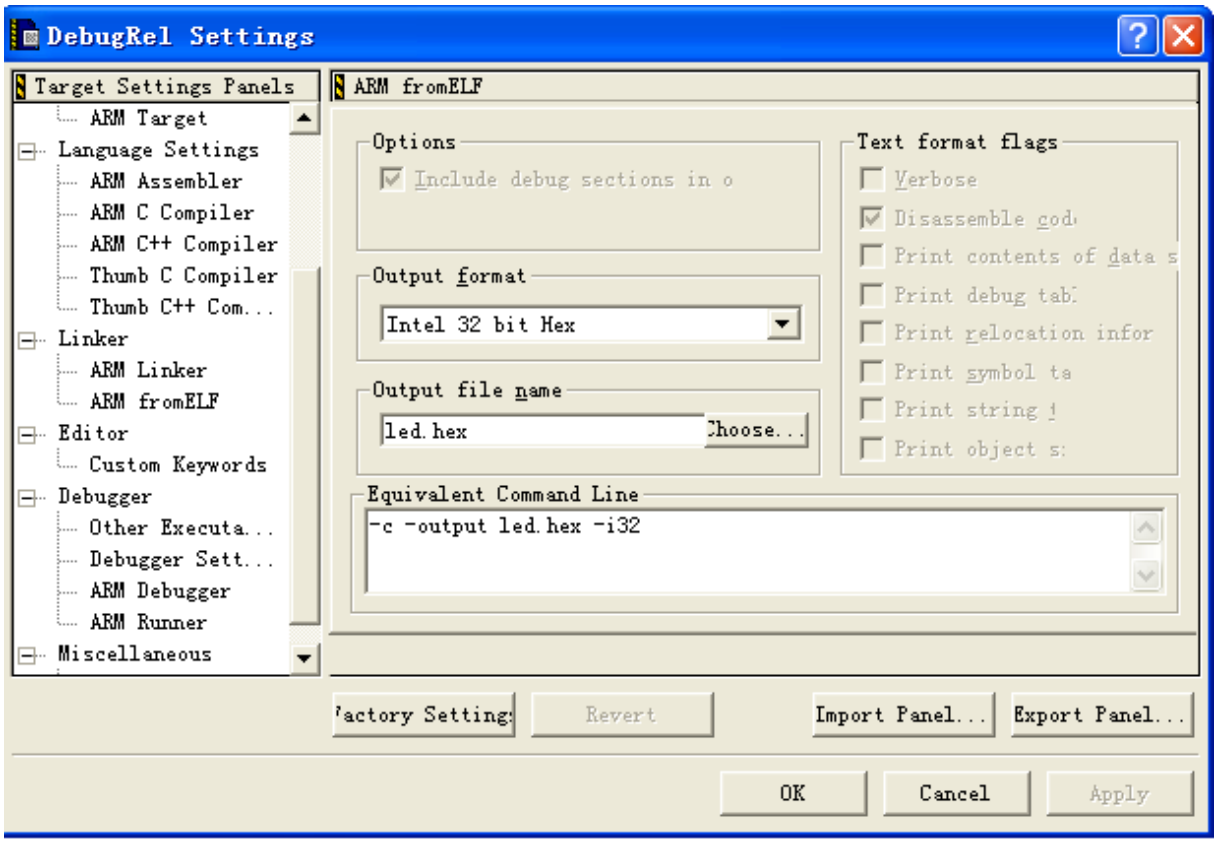


图 4-7 下载文件格式设置

（8）（此步可选，保存模版以后不用再做设置），设置完成后，可以将该新建的空工程文件作为模板保存以便以后使用，将工程文件名改为 44B0 ARM Executable.mcp。然后在 ADS 1.2 软件安装目录下的 Stationery 目录下新建名为 44B0 ARM Executable Image 的模板目录，再将刚设置完的 44B0 ARM Executable.mcp 工程模板文件存放到该目录下即可。这样以后新建工程的时候，就能看到以 44B0 ARM Executable Image 为名字的模板了。

提示：

建议用户直接将 TX-3A 随机光盘中的工程文件拷备到自己的电脑里 ,在此工程上直接修改源代码文件或新建文件即可，这样可省去新建工程的烦琐过程。

4.2 建立工程文件

配置好 ARM ADS 针对 TX-3A 的开发环境后，可以执行 Project|Add Files 命令把和工程相关的所有文件加入到工程中。ADS 1.2 不能自动按文件类别对这些文件进行分类，若需要，可以执行 Project|Create Group 命令创建文件组，然后分别将不同类的文件加入到不同的组，以方便管理。如图 4-9 所示。更为简单的办法是，在新建工程时 ADS 创建了和工程同名的目录，在该目录下按类别创建子目录并存放工程文件。选中所有目录拖动到任务栏上的 ADS 任务条上，中途不要松开鼠标。当 ADS 窗口恢复后再拖动到工程文件窗口，并松开鼠标。这样 ADS 将以子目录名建立同名文件组并以此对文件分类。在这里我们将 TX-3A 44B0 system test 工程中的 INC 和 SRC 文件夹全部复制到刚才新建的工程中，用上述办法添加到新工程中，在新添加文件时会弹出如图 4-8 所示对话框，这里我们将下面两个对勾去掉，仅留下 DebugRel 项，添加后的工程界面如图 4-9 所示：

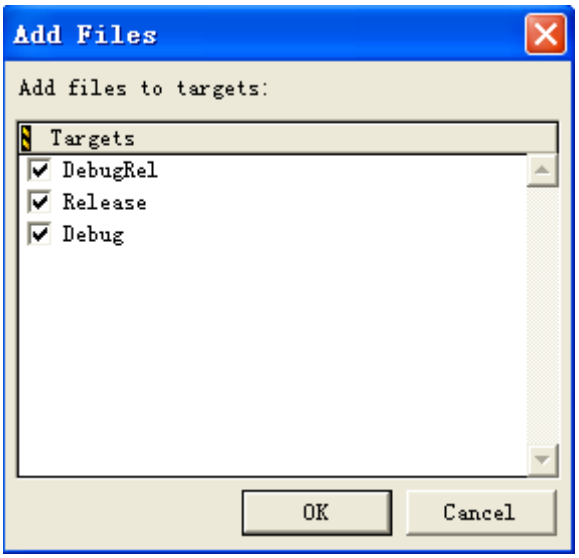


图 4-8　工程类别选择

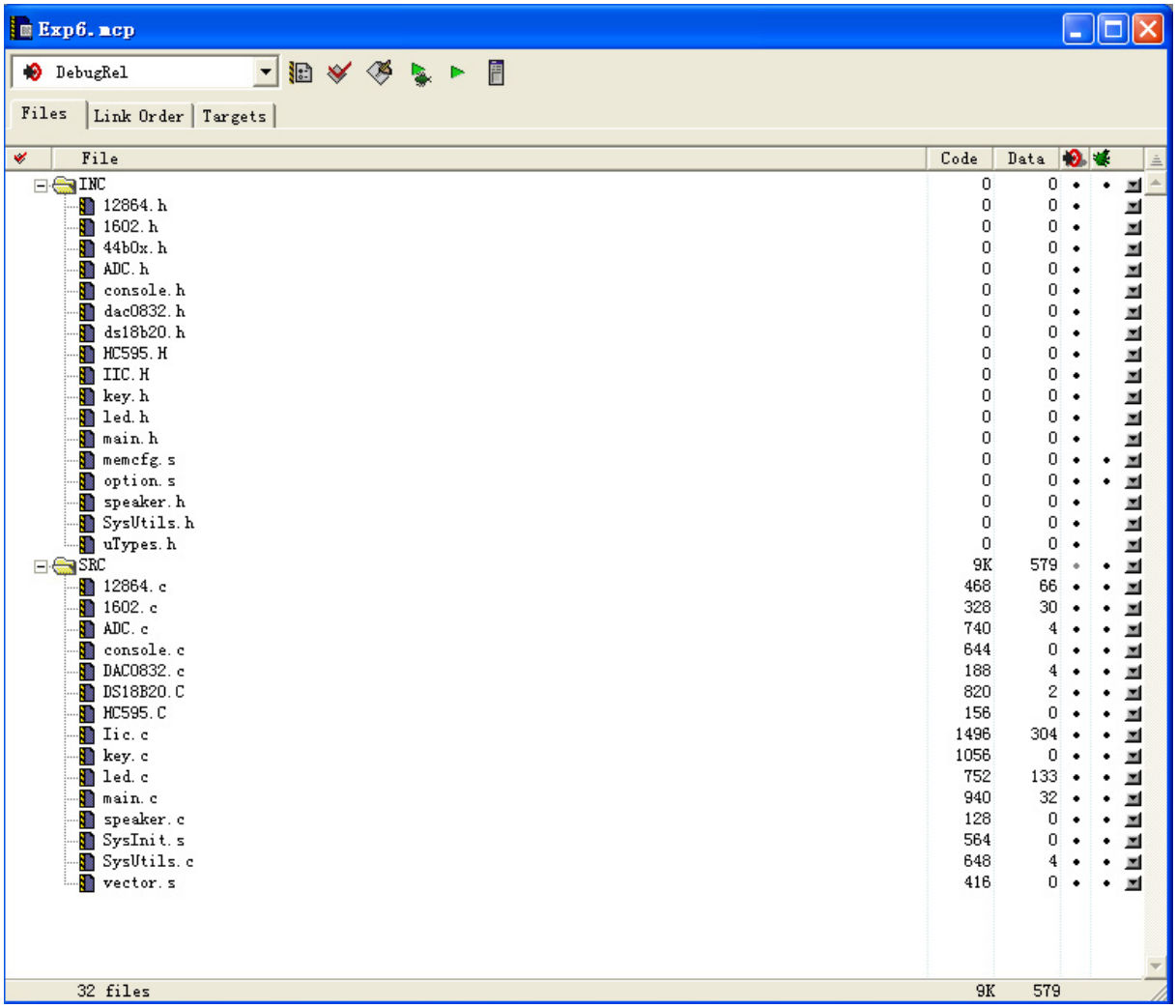


图 4-9 添加文件后的工程界面

双击图 4-9 中的 Main.c 打开该文件，可以看到 Main()函数的内容如下：

```
/*********************************************/
/*
    文件名  : main.c
    说  明  : 主文件
    作  者  : 郭天祥
    日  期 : 2008.9
*/
/*********************************************/
#include "..\inc\main.h"
#include "..\inc\dac0832.h"

//申明函数
void IoConfig (void);
void ledTest (void);
U16 Scan_4Key(void);
U16 Scan_4X4Key(void);

//我们用这个片编存储我们的 IRQ ISR 入口地址
U32 pIrqStart = 0;
U32 pIrqFinish = 0;
U32 pIrqHandler = 0;

U32 g_pTopOfROM = 0;
U32 g_pBaseOfROM = 0;
U32 g_pBaseOfBSS = 0;
U32 g_pBaseOfZero = 0;
U32 g_pEndOfBSS = 0;

extern U32 GetBaseOfROM (void);
extern U32 GetEndOfROM (void);
extern U32 GetBaseOfBSS (void);
extern U32 GetBaseOfZero (void);
extern U32 GetEndOfBSS (void);

/*主函数*/
int main(void)
{
  //U16 i;

  sysUtilsSetPllValue (24, 6, 1);  //设置片内的锁相环
```

```
  IoConfig (); //io 端口的初始化

  Clr_Pled();     //关闭跑马灯
  Clr_DZ();    //关闭点阵显示
  Clr_led();      //关闭数码管显示

  console_init (115200); //初始化控制台

  printf ("\n\n\t\tTX-3A/B 44B0 board Test!\n\n");

  printf("Test DAC0832......\n");
  printf( "Please see D10 led is flashing!\n" ) ;
  printf( "Press ESC key to exit!\n\n" ) ;
  Test_0832(); //测试 DAC0832

  printf("Test 键盘......\n");
  Test_key();     //键盘测试
//  printf("Test over!\n\n");

  printf("Test ADC......\n");
  ADC_Test();     //ADC 测试
//  printf("Test over!\n\n");

  printf("Test 蜂鸣器......\n");
  Test_Speaker();//喇叭测试
  printf("Buzzer est over!\n\n");

  printf("Test 数码管......\n");
  printf( "You will see 0123245!\n" );
    printf( "Press ESC key to exit!\n\n" ) ;
  Test_led();      //测试数码管
//  printf("Test over!\n\n");

  printf("Test 流水灯......\n");
  printf( "Press ESC key to exit!\n\n" ) ;
  Test_Pled(); //测试跑马灯
//  printf("Test over!\n\n");

  printf("Test 点阵......\n");
  Show_WWW();
```

```
    //  Test_DZ();       //点阵测试
      printf("Test over!\n\n");

      printf("Test 继电器......\n");
      Test_Relay();  //测试继电器
      printf("Test over!\n\n");

      printf("Test AT24c02......\n");
      Test_Iic();    //测试 IIc 器件
      printf("Test over!\n\n");

      printf("Test 温度传感器 DS18b20......\n");
      printf( "Press ESC key to exit!\n\n" ) ;
      Test_18b20();  //测试 18b20
      printf("Test over!\n\n");

      printf("Are you ready to test 1602?(y/n)");
      if('y'==getch())
      {
        printf("y\n");
        printf("Test 1602......\n");
        printf( "You will see TX-3A 44B0 and
www.txmcu.com!\n\n" ) ;
        Lcd_1602Test(); //测试 LCD1602
        printf("Test over!\n\n");
      }
      else
        printf("n\n");

      printf("Are you ready to test 12864?(y/n)");
      if('y'==getch())
      {
        printf("y\n");
        printf("\nTest 12864......\n");
        LCD12864_Test();//测试 LCD12864;
      }
      else
        printf("n\n");

      printf("测试结束!\n\n");
      //主循环/end
      while(1);
```

```
    return 0;
}
```

提示：读者通过查看其他源文件的内容以对系统运行有所了解，可以发现 ADS 的文本编辑器已经有了很大的改善，文本按语法分颜色显示。读者可以根据喜好在 Edit 菜单下的 Preferences 窗口中进行设置。

4.3 进行程序的在线仿真、调试

在线仿真、调试及程序烧写方法请查看随机光盘中的视频教程,这里简单介绍下 AXD 的功能按钮。

4.4 选择调试目标

当工程编译连接通过后，在工程窗口中点击“**Debug**”图标按钮，即可启动**AXD** 进行调试(也可以通过【开始】菜单起动**AXD)**。点击菜单【Options】选择【Configure Target…】，即弹出Choose Target 窗口，如图3-10所示。在没有添加其它仿真驱动程序前，Target 项中只有两项，分别为ADP(JTAG硬件仿真)和ARMUL(软件仿真)。

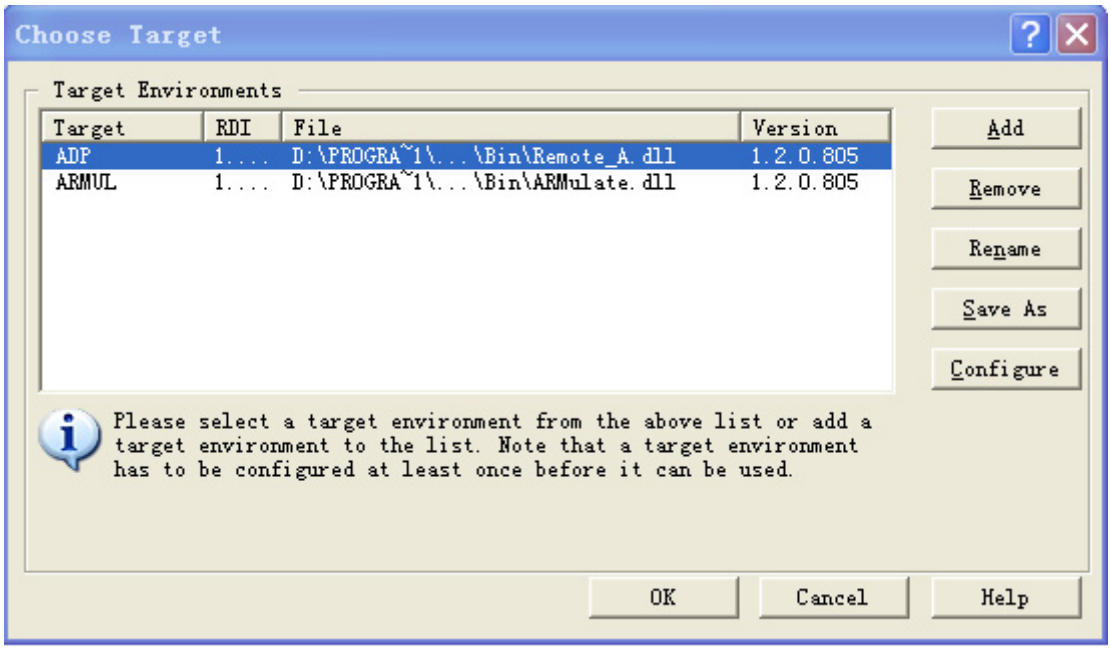


图4-10 AXD配置窗口

安装光盘软件目录中的H-JTAG V0.2.1.EXE，在图4-10中点击Add，定位到H-JTAG的安装目录，加载JTAG并口仿真驱动程序如图4-11所示。

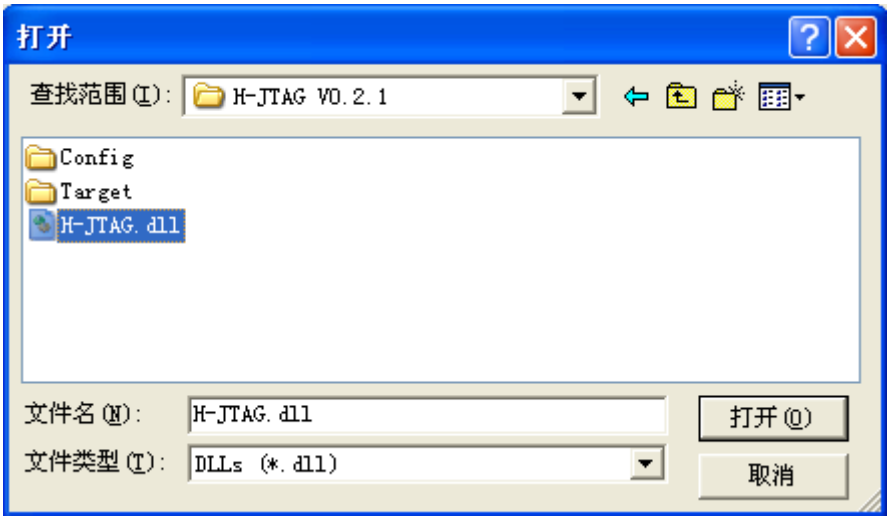


图4-11 加载JTAG并口仿真驱动程序

加载后的界面如图4-12所示：

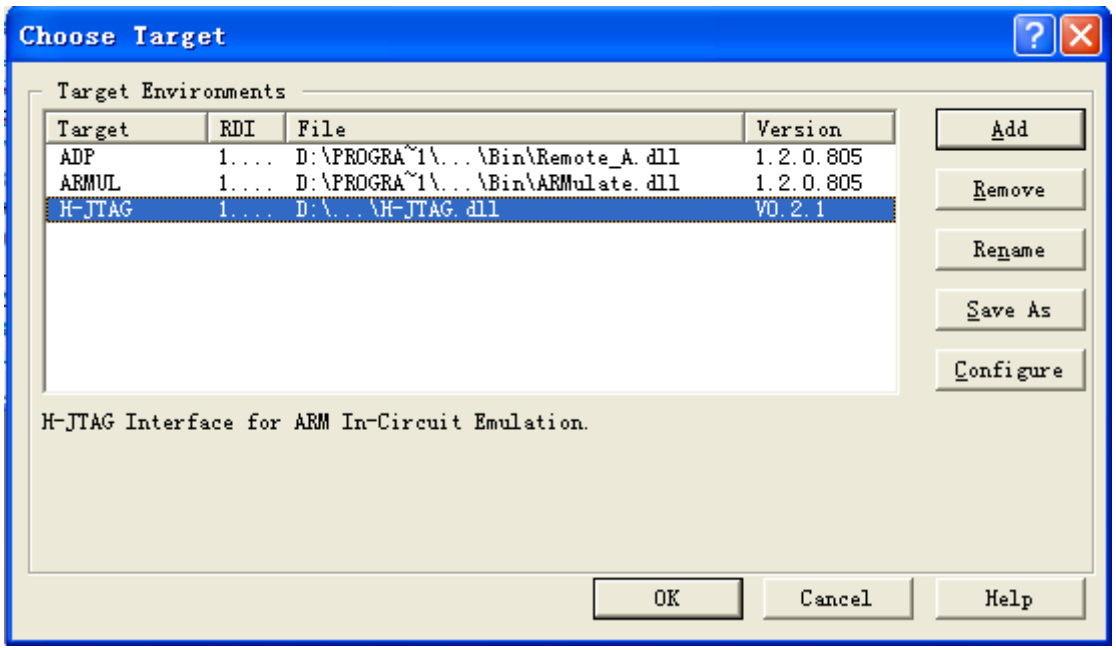


图4-12 加载JTAG并口仿真驱动程序后

点击OK，点击【**File**】选择【**Load Image...**】加载**ELF** 格式的可执行文件，即***.axf** 文件。说明：当工程编译连接通过后，在“工程名\工程名_Data\当前的生成目标”目录下就会生成一个*.axf 调试文件。比如工程TX-3A 44B0 system test，当前的生成目标DebugRel，编译连接通过后，则在...TX-3A 44B0 system test\led_Data\DebugRel目录下生成led.axf文件。

4.5 调试工具条

AXD运行调试工具条如图4-13所示，调试观察窗口工具条如图4-14 所示，文件操作工具条如图4-15 所示。

图4-13 AXD运行调试工具条


全速运行(Go)


停止运行(Stop)

单步运行(Step In)，与Step 命令不同之处在于对函数调用语句，Step In 命令将进入该函数

单步运行(Step)，每次执行一条语句，这时函数调用将被作为一条语句执行。

单步运行(Step Out)，执行完当前被调用的函数，停止在函数调用的下一条语句。

运行到光标(Run To Cursor)，运行程序直到当前光标所在行时停止。

设置断点(Toggle BreakPoint)

图 4-14 调试观察窗口工具条

打开寄存器窗口(Processor Registers)

打开观察窗口(Processor Watch)

打开变量观察窗口(Context Variable)

打开存储器观察窗口(Memory)

打开反汇编窗口(Disassembly)

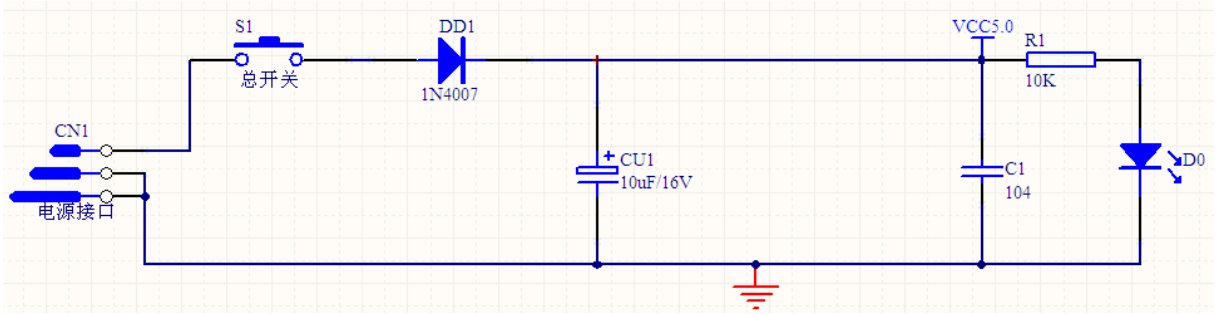
图4-15 文件操作工具条

加载调试文件(Load Image)

重新加载文件(Reload Current Image)。由于 AXD 没有复位命令，所以通常使用 Reload 实现复位(直接更改 PC 寄存器为零也能实现复位)。

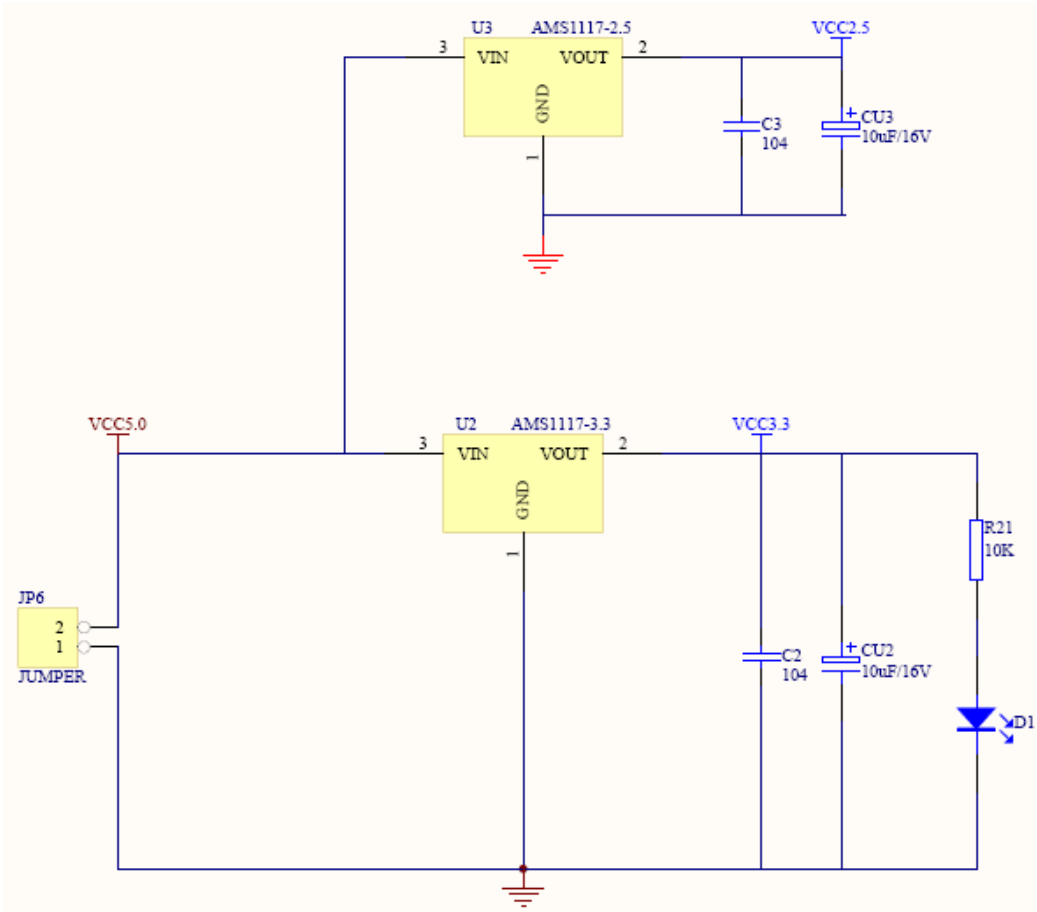
第五章 例程代码及原理图讲解

5.1 底板电源接口部分

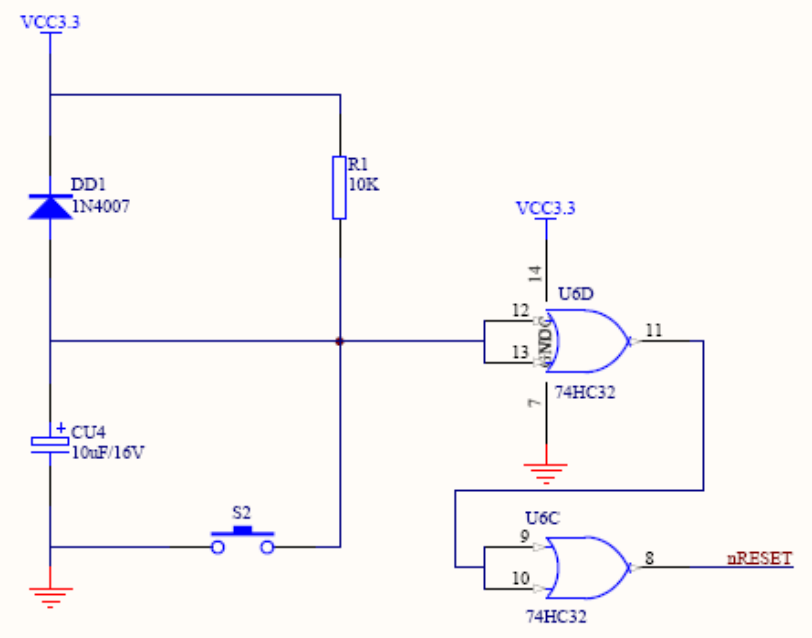


电源由开关 S1 控制通断，二极管 DD1 为防反接二极管，当电源极性接反时可有效保护板上元件不受损坏，D0 为电源指示灯。

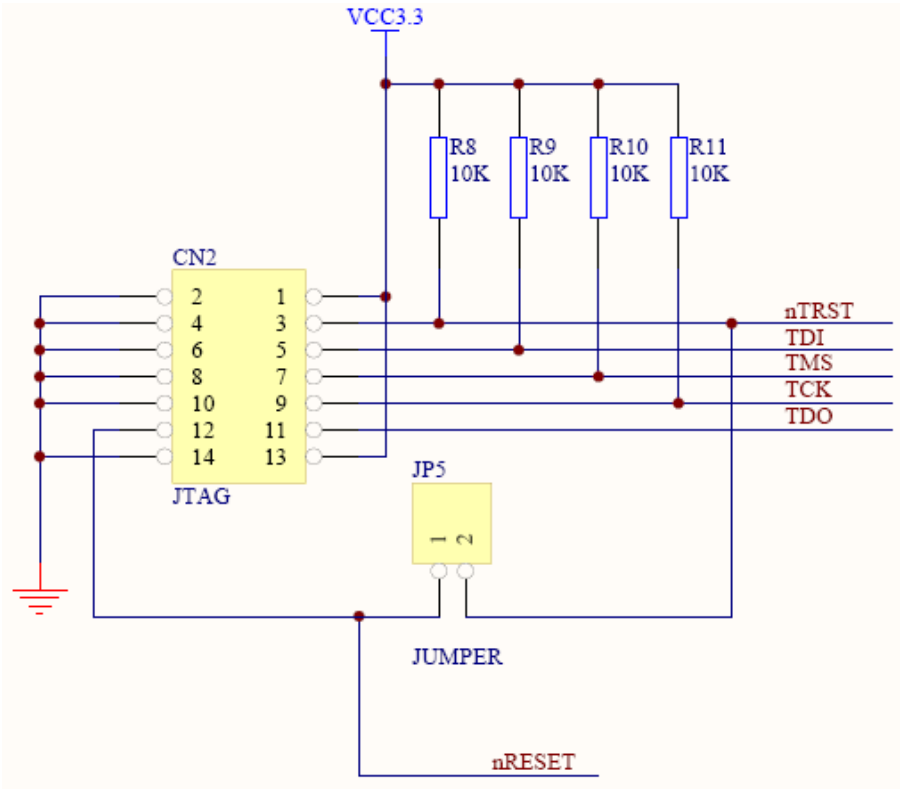
5.2 核心板电源部分



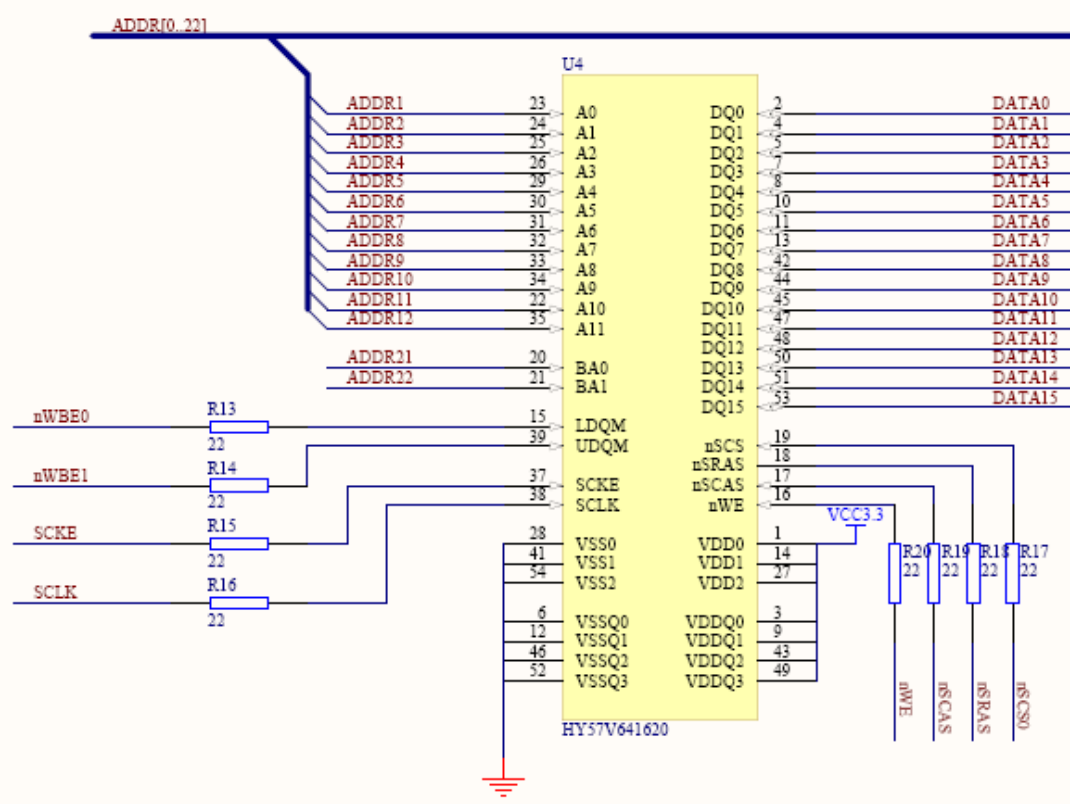
5.3 主复位电路



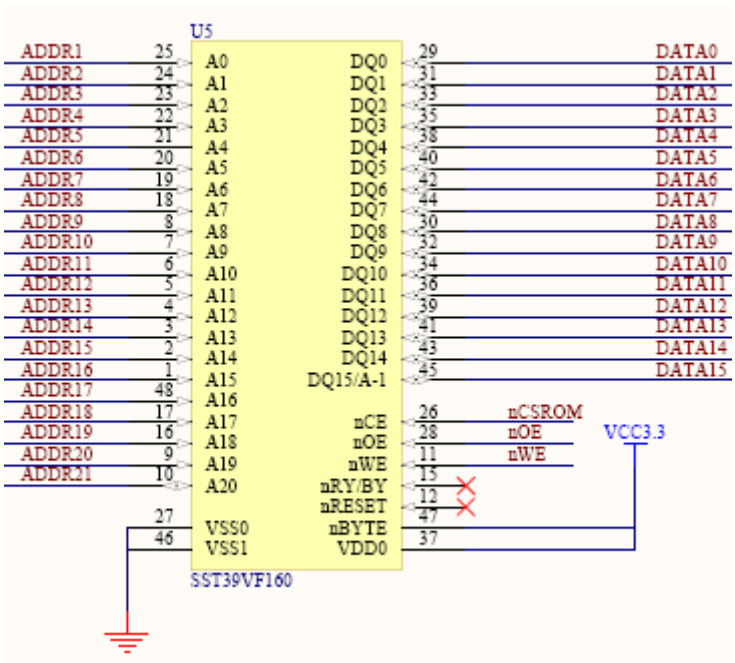
5.4 JTAG 下载部分电路



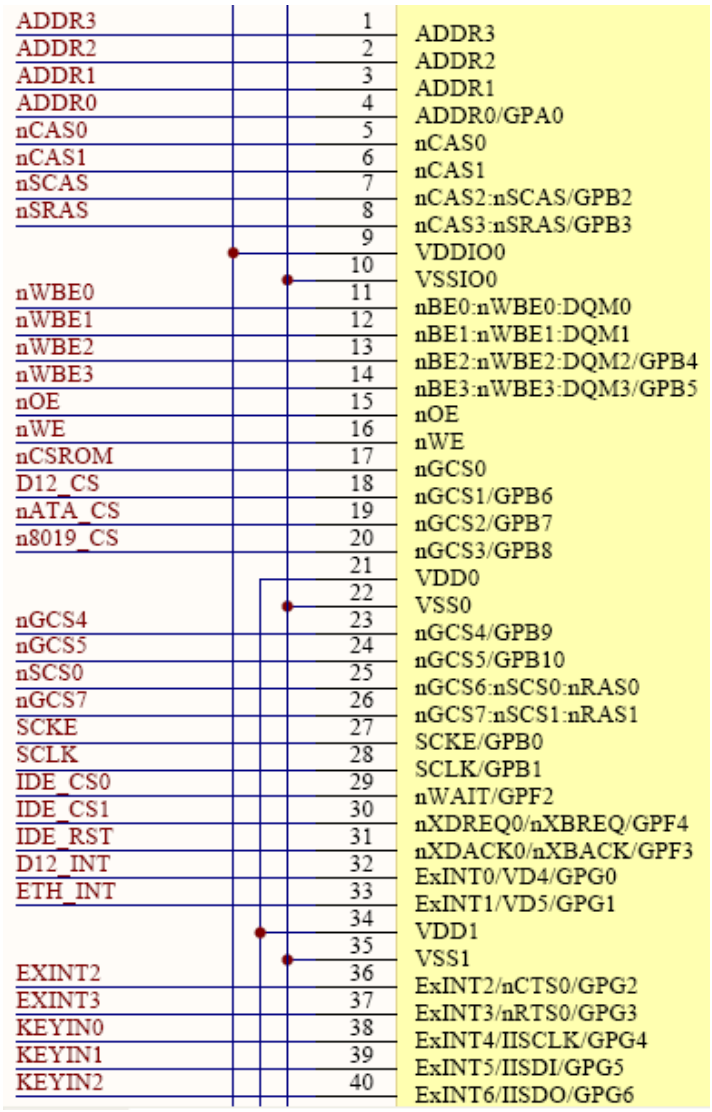
5.5 SDRAM 部分电路



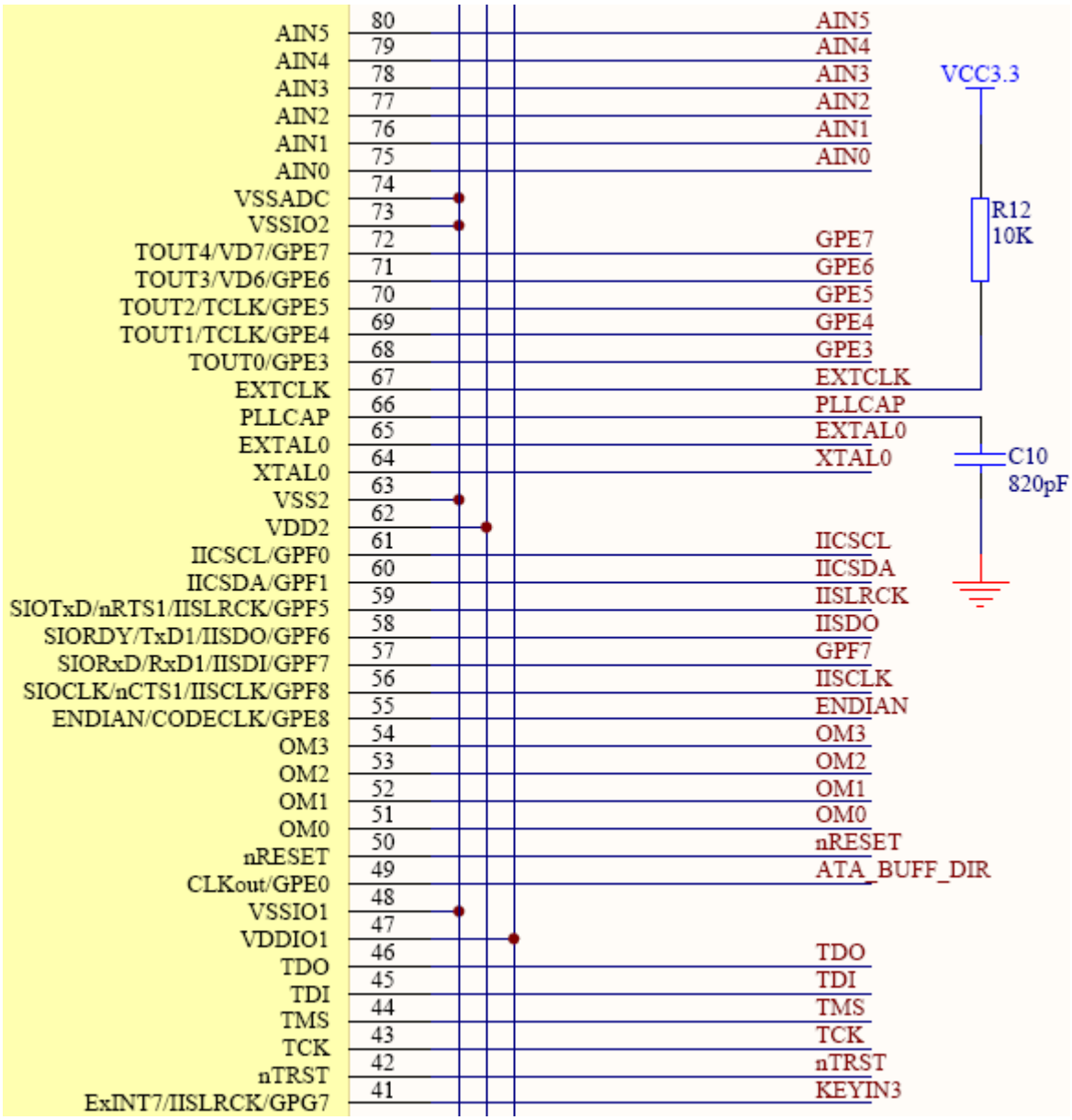
5.6 FLASH 部分电路



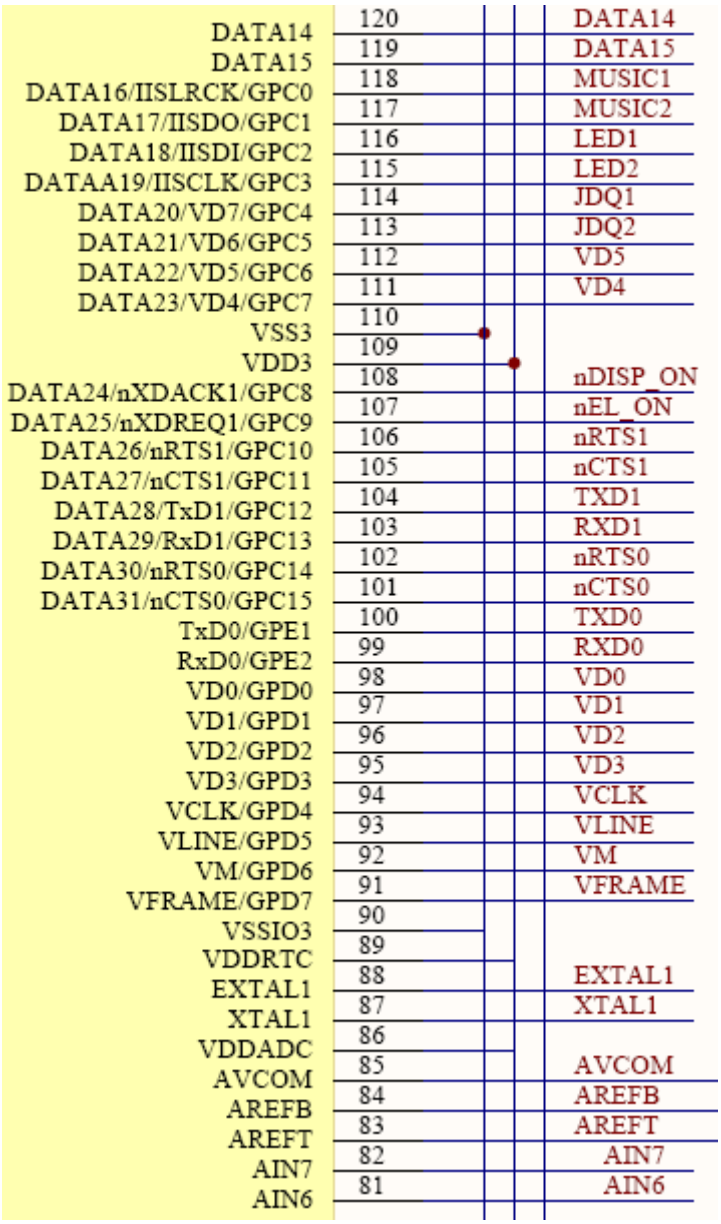
5.7 44B0 全部引脚引出电路



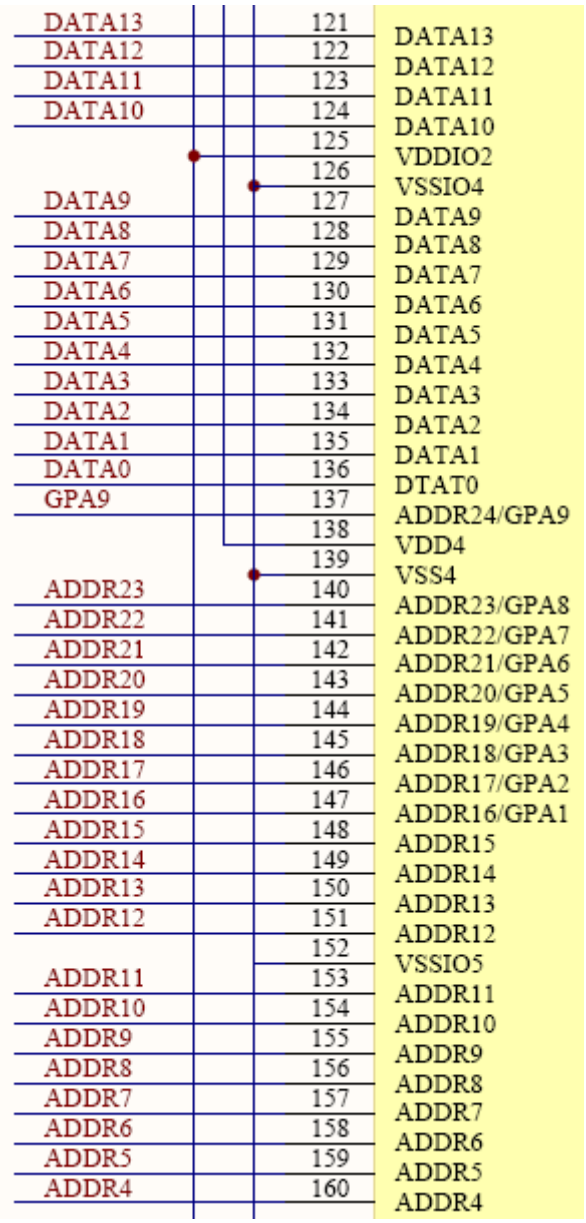
1-40 引脚


AIN5 80 AIN5
AIN4 79 AIN4
AIN3 78 AIN3
AIN2 77 AIN2
AIN1 76 AIN1
AIN0 75 AIN0
VSSADC 74
VSSIO2 73
TOUT4/VD7/GPE7 72 GPE7
TOUT3/VD6/GPE6 71 GPE6
TOUT2/TCLK/GPE5 70 GPE5
TOUT1/TCLK/GPE4 69 GPE4
TOUT0/GPE3 68 GPE3
EXTCLK 67 EXTCLK
PLLCAP 66 PLLCAP
EXTAL0 65 EXTAL0
XTAL0 64 XTAL0
VSS2 63
VDD2 62
IICSCL/GPF0 61 IICSCL
IICSDA/GPF1 60 IICSDA
SIOTxD/nRTS1/IISLRCK/GPF5 59 IISLRCK
SIORDY/TxD1/IISDO/GPF6 58 IISDO
SIORxD/RxD1/IISDI/GPF7 57 GPF7
SIOCLK/nCTS1/IISCLK/GPF8 56 IISCLK
ENDIAN/CODECLK/GPE8 55 ENDIAN
OM3 54 OM3
OM2 53 OM2
OM1 52 OM1
OM0 51 OM0
nRESET 50 nRESET
CLKout/GPE0 49 ATA_BUFF_DIR
VSSIO1 48
VDDIO1 47
TDO 46 TDO
TDI 45 TDI
TMS 44 TMS
TCK 43 TCK
nTRST 42 nTRST
ExINT7/IISLRCK/GPG7 41 KEYIN3
VCC3.3
R12
10K
C10
820pF

41-80 引脚

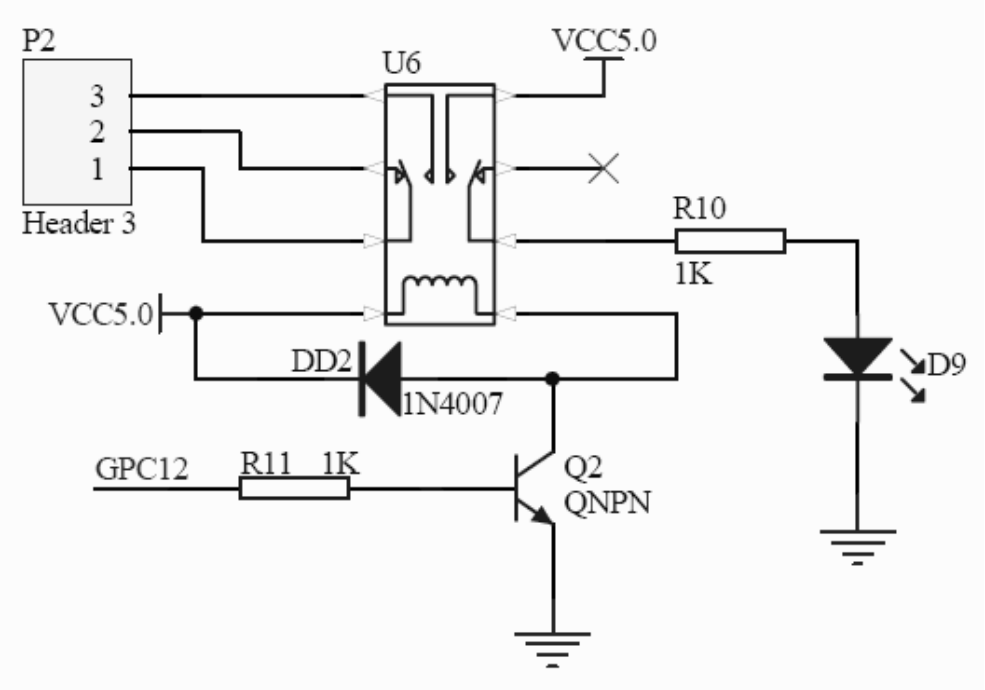

DATA14 120 DATA14
DATA15 119 DATA15
DATA16/IISLRCK/GPC0 118 MUSIC1
DATA17/IISDO/GPC1 117 MUSIC2
DATA18/IISDI/GPC2 116 LED1
DATAA19/IISCLK/GPC3 115 LED2
DATA20/VD7/GPC4 114 JDQ1
DATA21/VD6/GPC5 113 JDQ2
DATA22/VD5/GPC6 112 VD5
DATA23/VD4/GPC7 111 VD4
VSS3 110
VDD3 109
DATA24/nXDACK1/GPC8 108 nDISP_ON
DATA25/nXDREQ1/GPC9 107 nEL_ON
DATA26/nRTS1/GPC10 106 nRTS1
DATA27/nCTS1/GPC11 105 nCTS1
DATA28/TxD1/GPC12 104 TXD1
DATA29/RxD1/GPC13 103 RXD1
DATA30/nRTS0/GPC14 102 nRTS0
DATA31/nCTS0/GPC15 101 nCTS0
TxD0/GPE1 100 TXD0
RxD0/GPE2 99 RXD0
VD0/GPD0 98 VD0
VD1/GPD1 97 VD1
VD2/GPD2 96 VD2
VD3/GPD3 95 VD3
VCLK/GPD4 94 VCLK
VLINE/GPD5 93 VLINE
VM/GPD6 92 VM
VFRAME/GPD7 91 VFRAME
VSSIO3 90
VDDRTC 89
EXTAL1 88 EXTAL1
XTAL1 87 XTAL1
VDDADC 86
AVCOM 85 AVCOM
AREFB 84 AREFB
AREFT 83 AREFT
AIN7 82 AIN7
AIN6 81 AIN6

81-120 引脚

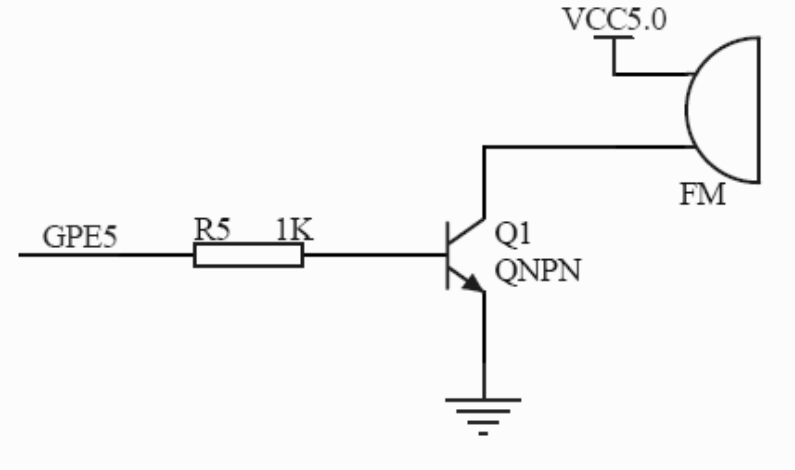

DATA13 121 DATA13
DATA12 122 DATA12
DATA11 123 DATA11
DATA10 124 DATA10
125 VDDIO2
126 VSSIO4
DATA9 127 DATA9
DATA8 128 DATA8
DATA7 129 DATA7
DATA6 130 DATA6
DATA5 131 DATA5
DATA4 132 DATA4
DATA3 133 DATA3
DATA2 134 DATA2
DATA1 135 DATA1
DATA0 136 DTAT0
GPA9 137 ADDR24/GPA9
138 VDD4
139 VSS4
ADDR23 140 ADDR23/GPA8
ADDR22 141 ADDR22/GPA7
ADDR21 142 ADDR21/GPA6
ADDR20 143 ADDR20/GPA5
ADDR19 144 ADDR19/GPA4
ADDR18 145 ADDR18/GPA3
ADDR17 146 ADDR17/GPA2
ADDR16 147 ADDR16/GPA1
ADDR15 148 ADDR15
ADDR14 149 ADDR14
ADDR13 150 ADDR13
ADDR12 151 ADDR12
152 VSSIO5
ADDR11 153 ADDR11
ADDR10 154 ADDR10
ADDR9 155 ADDR9
ADDR8 156 ADDR8
ADDR7 157 ADDR7
ADDR6 158 ADDR6
ADDR5 159 ADDR5
ADDR4 160 ADDR4

121-160 引脚

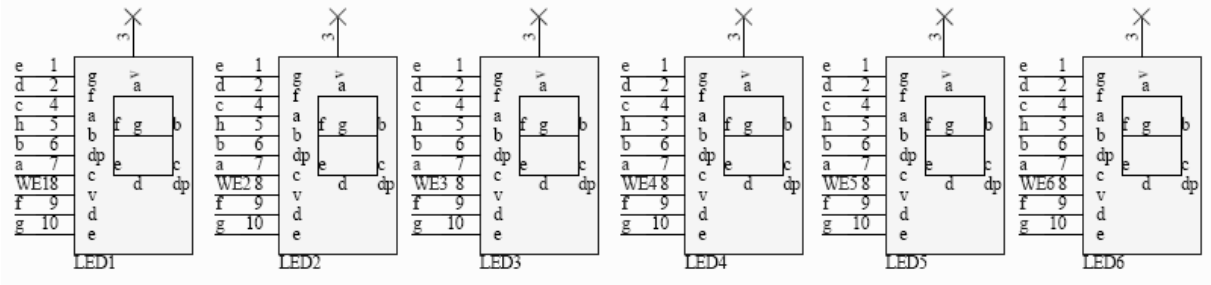
5.8 继电器部分电路



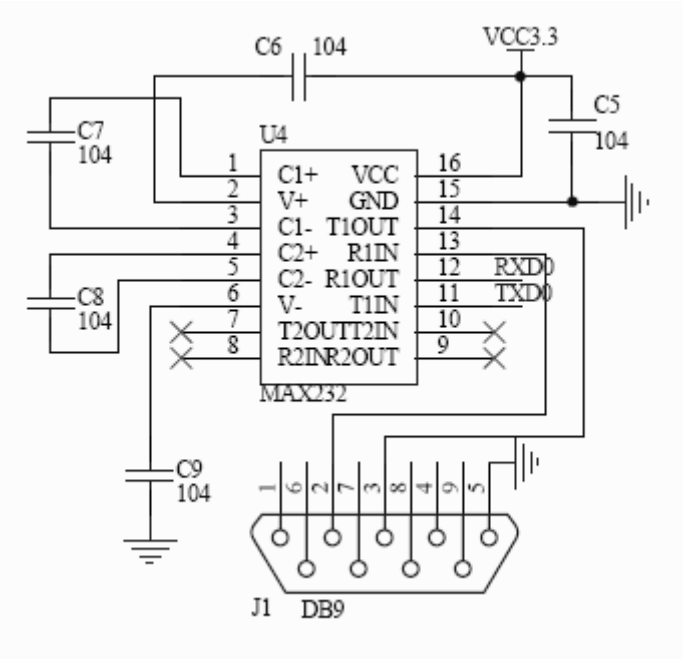
5.9 蜂鸣器电路



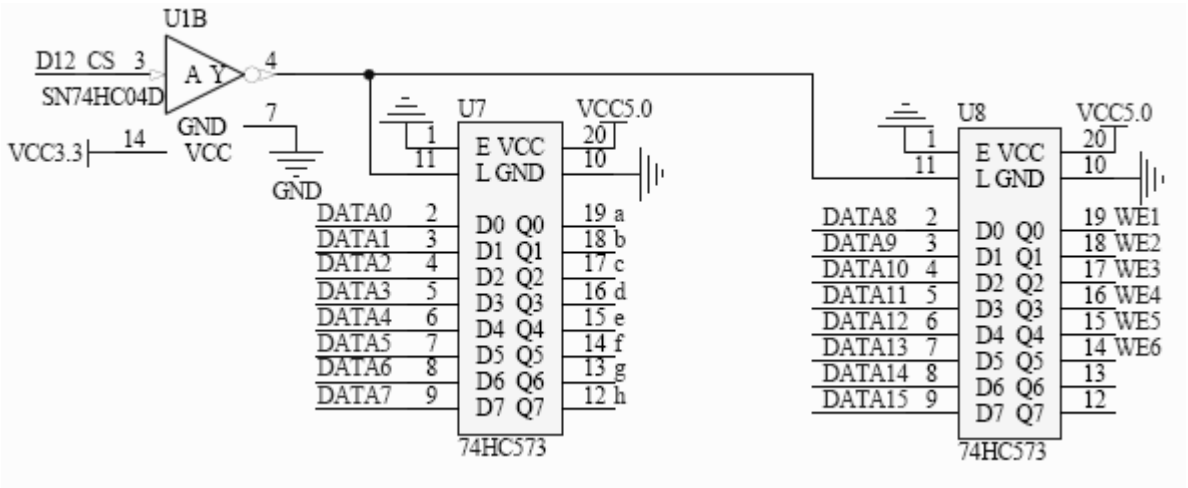
5.10 数码管电路



5.11 串口电路

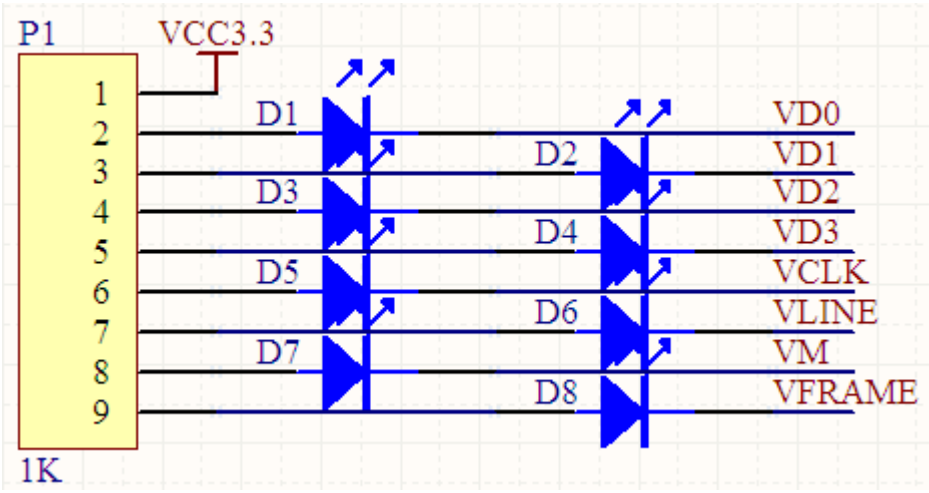


5.12 锁存器电路



5.13 一个 LED 灯闪烁

原理图



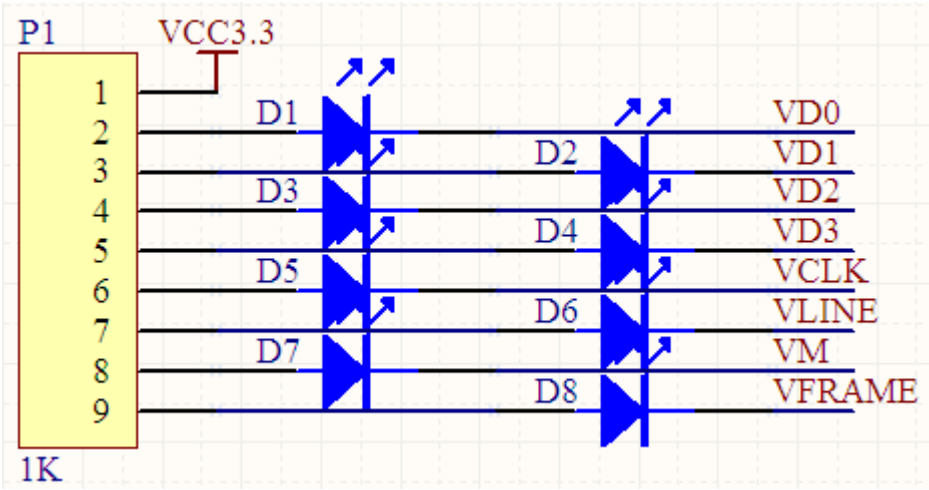
C 代码

```
#include "..\inc\uTypes.h"
#include "..\inc\44b0x.h"
#include "..\inc\console.h"
#include "..\inc\sysUtils.h"

//申明函数
void IoConfig (void);
void ledTest (void);
```

```
    //我们用这个片编存储我们的 IRQ ISR 入口地址
    U32 pIrqStart = 0;
    U32 pIrqFinish = 0;
    U32 pIrqHandler = 0;

    int main(void)
    {
      U8 i;
      sysUtilsSetPllValue (24, 6, 1);//设置片内倍频器
      IoConfig ();     //设置端口
      console_init (115200);//设置端口波特率
      //主循环
      for (;;)
      {
          rPDATD^=(1<<0);
          sysUtilsUSecDelay(500000);//延时 200ms
      }
      return 0;
}
```

5.14 流水灯实验

原理图



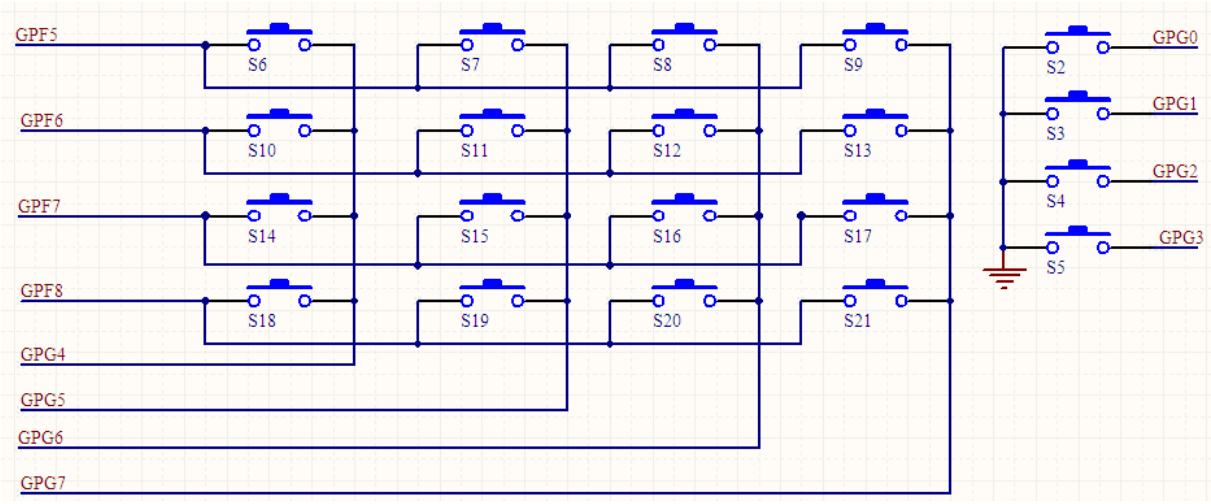
C 代码

```
#include "..\inc\uTypes.h"
```

```
#include "..\inc\44b0x.h"
#include "..\inc\console.h"
#include "..\inc\sysUtils.h"

//申明函数
void IoConfig (void);
void ledTest (void);

//我们用这个片编存储我们的 IRQ ISR 入口地址
U32 pIrqStart = 0;
U32 pIrqFinish = 0;
U32 pIrqHandler = 0;

int main(void)
{
   U8 i;
   sysUtilsSetPllValue (24, 6, 1);//设置片内倍频器，2 倍频
   IoConfig ();        //设置端口

   console_init (115200);//设置端口波特率

   //主循环
   for (;;)
   {
       //在主循环中一直循环执行跑马灯程序
       for(i=0;i<8;i++)
       {
           rPDATD=~(1<<i);
           sysUtilsUSecDelay(2000);//延时 200ms
       }
   }
   return 0;
}
```

5.15 键盘检测实验

原理图



C 代码

```
#include "..\inc\main.h"
//我们用这个片编存储我们的 IRQ ISR 入口地址
U32 pIrqStart = 0;
U32 pIrqFinish = 0;
U32 pIrqHandler = 0;
/*主函数*/
int main(void)
{
   //U16 i;
   sysUtilsSetPllValue (24, 6, 1); //设置片内的锁相环
   IoConfig ();//io 端口的初始化
   Clr_Pled();      //关闭跑马灯
   Clr_DZ();        //关闭点阵显示
   Clr_led();       //关闭数码管显示
   console_init (115200);     //初始化控制台
   printf("Test adc.....\n");
   while(1)
       Test_key();
   printf("Test over!\n\n");
   while(1);
   return 0;
}
```

Test_key()代码

```
#include "..\inc\key.h"
#include "..\inc\led.h"
#include "..\inc\speaker.h"
```

```
/*4X4 键盘的扫描*/
U16 Scan_4X4Key(void)
{
   U16 keytmp=0;
   rPDATG=0xef;  //第一列
   if((rPDATF&0x1e0)!=0x1e0)
   {
       keytmp=(rPDATF&0x1e0);
       Set_Speaker();
       ysUtilsUSecDelay(20000);
       Clr_Speaker();
       if((rPDATF&0x1e0)!=0x1e0)
       {
           while((rPDATF&0x1e0)==keytmp)
               sysUtilsUSecDelay(200);
           return ((rPDATG&0xf0)|(keytmp)>>5);
       }
   }

   rPDATG=0xdf;  //第二列
   if((rPDATF&0x1e0)!=0x1e0)
   {
       keytmp=(rPDATF&0x1e0);
       Set_Speaker();
               sysUtilsUSecDelay(20000);
           Clr_Speaker();
       if((rPDATF&0x1e0)!=0x1e0)
       {
           while((rPDATF&0x1e0)==keytmp)
               sysUtilsUSecDelay(200);
           return ((rPDATG&0xf0)|(keytmp)>>5);
       }
   }

   rPDATG=0xbf;  //第三列
   if((rPDATF&0x1e0)!=0x1e0)
   {
```

```
        keytmp=(rPDATF&0x1e0);
        Set_Speaker();
                sysUtilsUSecDelay(20000);
            Clr_Speaker();
        if((rPDATF&0x1e0)!=0x1e0)
        {
            while((rPDATF&0x1e0)==keytmp)
                sysUtilsUSecDelay(200);
            return ((rPDATG&0xf0)|(keytmp)>>5);
        }
    }

    rPDATG=0x7f; //第四列
    if((rPDATF&0x1e0)!=0x1e0)
    {
        keytmp=(rPDATF&0x1e0);
        Set_Speaker();
                sysUtilsUSecDelay(20000);
            Clr_Speaker();
        if((rPDATF&0x1e0)!=0x1e0)
        {
            while((rPDATF&0x1e0)==keytmp)
                sysUtilsUSecDelay(200);
            return ((rPDATG&0xf0)|(keytmp)>>5);
        }
    }
    return 0x8000;
}
/*4 个独立键盘扫描*/
U16 Scan_4Key(void)
{
    U8 keytmp;
    //扫描独立的 4 个键
    if((rPDATG&0xf)!=0x0f)
    {
        keytmp=(rPDATG&0xf);
        Set_Speaker();
                sysUtilsUSecDelay(20000);
```

```
            Clr_Speaker();
        if((rPDATG&0xf)!=0x0f)
        {
            while((rPDATG&0xf)==keytmp)
                sysUtilsUSecDelay(200);
            return (keytmp);
        }
    }
    return 0x8000;
}

void Test_key(void)
{
    U16 i;U8 show=0;
    printf( "\nKey Test\n" ) ;
        printf( "Please press buttons and see what happen\n" ) ;
        printf( "Press ESC key to exit!\n\n" ) ;
    while(!( kbhit() && (getkey()==ESC_KEY)))
    {
        i=Scan_4X4Key();
        if(i!=0x8000)
        {
            switch(i)
                {
                case 0xEE:
                    show=1;
                    //printf("Key1 \n");
                break;
                case 0xDE:
                    show=2;
                    //printf("Key2 \n");
                break;
                case 0xBE:
                    show=3;
                    //printf("Key3 \n");
                break;
                case 0x7E:
                    show=4;
```

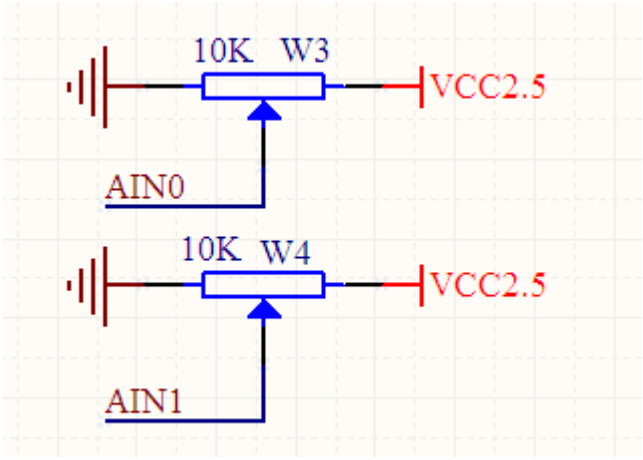
```
        //printf("Key4 \n");
    break;
    case 0xED:
        show=5;
        //printf("Key5 \n");
    break;
    case 0xDD:
        show=6;
        //printf("Key6 \n");
    break;
    case 0xBD:
        show=7;
        //printf("Key7 \n");
    break;
    case 0x7D:
        show=8;
        //printf("Key8 \n");
    break;
    case 0xEB:
        show=9;
        //printf("Key9 \n");
    break;
    case 0xDB:
        show=10;
        //printf("Key10 \n");
    break;
    case 0xBB:
        show=11;
        //printf("Key11 \n");
    break;
    case 0x7B:
        show=12;
        //printf("Key12 \n");
    break;
    case 0xE7:
        show=13;
        //printf("Key13 \n");
    break;
```

```
            case 0xD7:
                show=14;
                //printf("Key14 \n");
            break;
            case 0xB7:
                show=15;
                //printf("Key15 \n");
            break;
            case 0x77:
                show=16;
                //printf("Key16 \n");
            break;
            }
            Show_Led(0,show,0);
}

i=Scan_4Key();
    if(i!=0x8000)
    {
        switch(i)
        {
        case 0x0E:
            show=17;
            //printf("Key17 \n");
        break;
        case 0x0D:
            show=18;
            //printf("Key18 \n");
        break;
        case 0x0B:
            show=19;
            //printf("Key19 \n");
        break;
        case 0x07:
            show=20;
            //printf("Key20 \n");
        break;
        }
```

```
                Show_Led(0,show,0);
            }
        }
    }
```

5.16 AD 模数转换实验

原理图



C 代码

```
#include "..\inc\main.h"

//我们用这个片编存储我们的 IRQ ISR 入口地址
U32 pIrqStart = 0;
U32 pIrqFinish = 0;
U32 pIrqHandler = 0;
/*主函数*/
int main(void)
{
    sysUtilsSetPllValue (24, 6, 1); //设置片内的锁相环
    IoConfig ();//io 端口的初始化
    Clr_Pled();       //关闭跑马灯
    Clr_DZ();         //关闭点阵显示
    Clr_led();        //关闭数码管显示
    console_init (115200);     //初始化控制台
    printf("Test adc.....\n");
    while(1)
```

```
        ADC_Test();
    printf("Test over!\n\n");
    while(1);
    return 0;
}
```

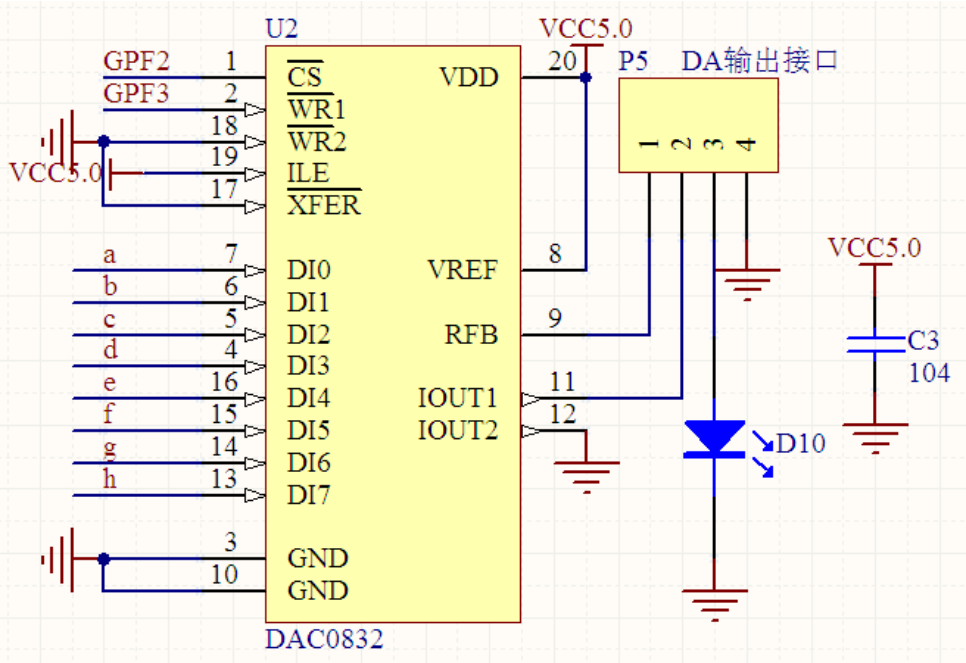
ADC_Test()代码

```
#include "..\inc\ADC.h"
#include "..\inc\led.h"
void ADC_Init(void)
{
    rCLKCON = rCLKCON | (1<<12);        //控制系统主时钟进入 ADC 单
元模块
    rADCCON = 0x1|(0<<2);
    sysUtilsUSecDelay(100);        //延时若干个 100us
    rADCPSR = 10;
}
unsigned short Read_Adc(unsigned char ch)
{
    int i;
      static int prevCh=-1;

      if(prevCh!=ch)
      {
       rADCCON = 0x1|(ch<<2); //设置 AD 转换通道
          for(i=0;i<150;i++);      //最小 15uS
      }
      rADCCON=0x1|(ch<<2);     //开始 AD 转换
      while(rADCCON & 0x1);     //避免标志 FLAG 错误
      while(!(rADCCON & 0x40));          //等待 AD 转换结束
      for(i = 0; i < rADCPSR; i++);      //避免第二次标志 FLAG 错误
      prevCh=ch;
      return rADCDAT;          //返回 AD 转换值
}
void ADC_Test(void)
{
       int a0=0,a1=0;
       float b0=0,b1=0;
       printf( "\nADC Test\n" ) ;
```

```
            printf( "Please change R1 and see what happen\n" ) ;
            printf( "Press ESC key to exit!\n\n" ) ;
            ADC_Init() ;          //模数转换初始化
            while( !( kbhit && (getkey()==ESC_KEY) ) )
            {
               a0 = Read_Adc(0);       //ADC 某一通道进行转换，返回转换
的数据
               a1 = Read_Adc(1);       //ADC 某一通道进行转换，返回转换
的数据
               b1=a0*2.500/1024;
               b0=a1*2.500/1024;
               Show_Led(0,b0,1);
               sysUtilsUSecDelay(2000);
               Show_Led(1,((U16)(b0*10)%10),0);
               sysUtilsUSecDelay(2000);
               Show_Led(2,((U16)(b0*100)%10),0);
               sysUtilsUSecDelay(2000);
               Show_Led(3,b1,1);
               sysUtilsUSecDelay(2000);
               Show_Led(4,((U16)(b1*10)%10),0);
               sysUtilsUSecDelay(2000);
               Show_Led(5,((U16)(b1*100)%10),0);
               sysUtilsUSecDelay(2000);
            }
        }
```

5.17 DA 数模转换实验

原理图



C 代码

```
#include "..\inc\main.h"
//我们用这个片编存储我们的 IRQ ISR 入口地址
U32 pIrqStart = 0;
U32 pIrqFinish = 0;
U32 pIrqHandler = 0;
/*主函数*/
int main(void)
{
   //U16 i;
   sysUtilsSetPllValue (24, 6, 1); //设置片内的锁相环
   IoConfig ();//io 端口的初始化
   console_init (115200);     //初始化控制台
   printf("Test 0832.....\n");
   while(1)
       Test_0832();
   printf("Test over!\n\n");
   while(1);
   return 0;
}
```

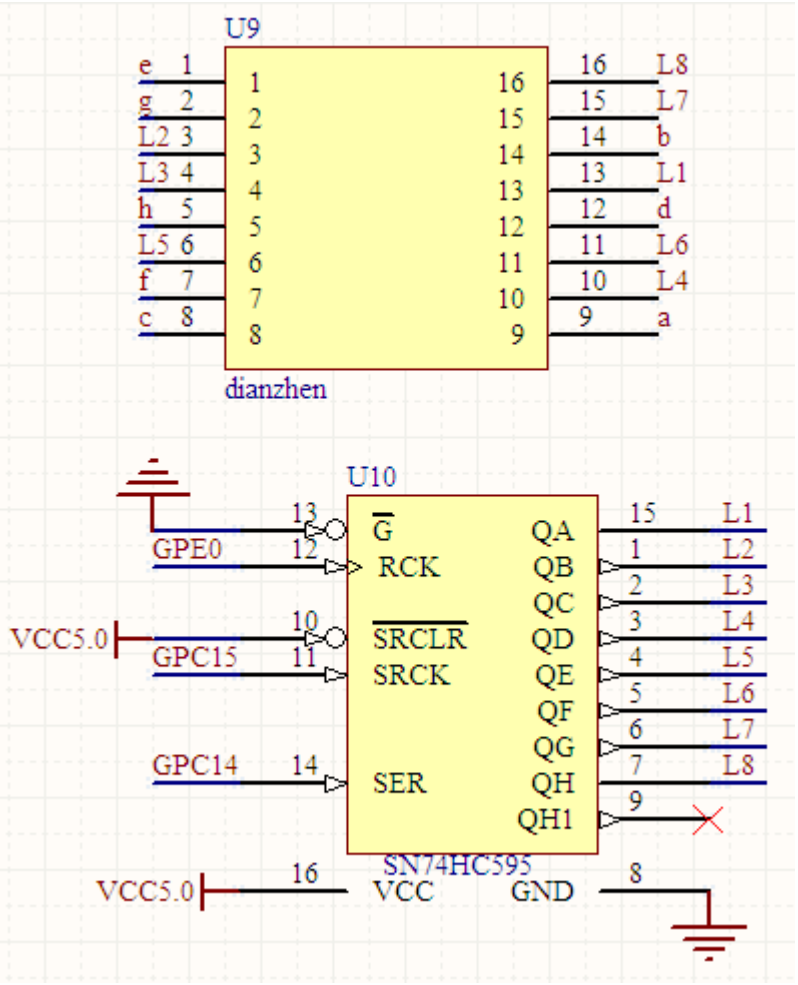
Test_0832()代码

```
#include "..\inc\dac0832.h"
#include "..\inc\sysUtils.h"
```

```
#include "..\inc\console.h"
#define    ESC_KEY         0x1b
#define DAC0832_WR_H() rPDATF|=0x08;
#define DAC0832_WR_L() rPDATF&=~0x08;
#define DAC0832_CS_H() rPDATF|=0x04;
#define DAC0832_CS_L() rPDATF&=~0x04;
//总线地址声明
volatile U16 * DAC0832_MAdd = (volatile U16 *)(0x2000000);
void Write_Dac0832(U8 data)
{
   DAC0832_CS_L();
   sysUtilsUSecDelay(5);
   * DAC0832_MAdd=data|0xff00;
   sysUtilsUSecDelay(5);
   DAC0832_WR_L();
   sysUtilsUSecDelay(5);
   DAC0832_WR_H();
   sysUtilsUSecDelay(5);
   DAC0832_CS_H();
}

void Test_0832(void)
{
   U8 i=0;
   while(!( kbhit() && (getkey()==ESC_KEY)))
   {
       Write_Dac0832(i++);
       sysUtilsUSecDelay(50000);
   }
}
```

5.18 点阵显示实验

原理图



C 代码

```
#include "..\inc\main.h"
//我们用这个片编存储我们的 IRQ ISR 入口地址
U32 pIrqStart = 0;
U32 pIrqFinish = 0;
U32 pIrqHandler = 0;
/*主函数*/
int main(void)
{
   //U16 i;
   sysUtilsSetPllValue (24, 6, 1); //设置片内的锁相环

   IoConfig ();//io 端口的初始化
   Clr_Pled();      //关闭跑马灯
   Clr_DZ();        //关闭点阵显示
   Clr_led();       //关闭数码管显示
   console_init (115200);     //初始化控制台
```

```
    printf("Test  点阵.....\n");
    while(1)
        Show_WWW();//打出滚动的字
    printf("Test over!\n\n");
    while(1);
    return 0;
}
```

Show_WWW()代码

```
#include "..\inc\led.h"
#include "..\inc\console.h"
#define   ESC_KEY         0x1b
//声明总线地址
volatile U16 * LedDBuffer = (volatile U16 *)(0x2000000);
volatile U16 * LedSBuffer = (volatile U16 *)(0x2000000);
static U8
SHOWDATA[21]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F,0x77
,0x7C,0x39,0x5E,0x79,0x71,0x76,0x01,0x40,0x08,0x73};     //共阴

void Clr_led(void);
/* W(0) W(1) W(2) .(3) T(4) X(5) M(6)
  C(7) U(8) .(9) C(10) O(11) M(12)*/
U8 ZI_MO[]=
{
0xF8,0x07,0x3C,0xF0,0x3C,0x07,0xF8,0x00,/*"W",0*/
0xF8,0x07,0x3C,0xF0,0x3C,0x07,0xF8,0x00,/*"W",1*/
0xF8,0x07,0x3C,0xF0,0x3C,0x07,0xF8,0x00,/*"W",2*/
0x00,0x01,0x00,0x00,0x00,0x00,0x00,0x00,/*".",3*/
0x80,0x80,0x80,0xFF,0x80,0x80,0x80,0x00,/*"T",4*/
0x81,0xC3,0x24,0x18,0x24,0xC3,0x81,0x00,/*"X",5*/
0xFF,0x70,0x0E,0x03,0x1E,0x60,0xFF,0x00,/*"M",6*/
0x3C,0x42,0x81,0x81,0x81,0xC3,0x66,0x00,/*"C",7*/
0xFE,0x01,0x01,0x01,0x01,0x01,0xFE,0x00,/*"U",8*/
0x00,0x01,0x00,0x00,0x00,0x00,0x00,0x00,/*".",9*/
0x3C,0x42,0x81,0x81,0x81,0xC3,0x66,0x00,/*"C",10*/
0x00,0x7E,0xC3,0x81,0x81,0xC3,0x7E,0x00,/*"O",11*/
0xFF,0x70,0x0E,0x03,0x1E,0x60,0xFF,0x00,/*"M",12*/
};
/************************************************
```

```
函数名：void Test_led(void)
**************************************************/
void Test_Pled(void)
{
   U8 i;
   for(i=1;i<9;i++)
   {
       sysUtilsLightLed(i,TRUE);
       sysUtilsUSecDelay(500000);
   }
   for(i=1;i<9;i++)
   {
       sysUtilsLightLed(i,FALSE);
       sysUtilsUSecDelay(500000);
   }
}
//测试数码管
void Test_led(void)
{
   U8 j;
   while(!( kbhit() && (getkey()==ESC_KEY)))
   {
       //点阵屏测试
       for(j=0;j<6;j++)
       {
           *LedDBuffer=((~(1<<j))<<8)+SHOWDATA[j];  //点亮第 j 段
           sysUtilsUSecDelay(2000); //延时 1ms
       }
   }
   Clr_led();        //关闭数码管显示
}

//x  第几个数码管
//data  要显示的数据
//P  是否加小数点
void Show_Led(U8 x,U8 data,U8 P)
{
   *LedDBuffer=((~(1<<x))<<8)+SHOWDATA[data]+P*0x80;     //点亮第
```

j 段

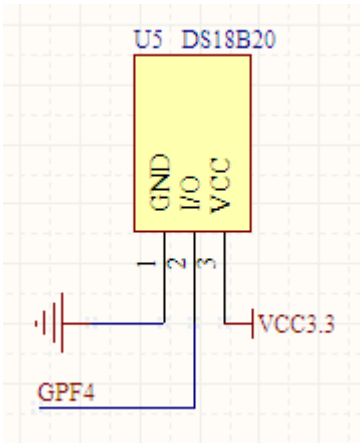
```
}
//关闭跑马灯显示
void Clr_Pled(void)
{
  sysUtilsLightLed(LED_ALL,FALSE);
}
//数码管关闭
void Clr_led(void)
{
  *LedSBuffer=0xffff;        //关闭数码管显示
}
//点阵清屏
void Clr_DZ(void)
{
  Send_595(0xffff);      //点阵清屏
}
/*****************************************
函数：void Test_DZ(void)
功能：点阵屏测试
*****************************************/
void Test_DZ(void)
{
  U8 i,j;
  for(j=0;j<8;j++)
  {
      *LedDBuffer=((1<<j)|0xff00);     //点亮第 j 段
      for(i=0;i<8;i++)
      {
          Send_595(~(1<<i));    //
          sysUtilsUSecDelay(100000);//延时 100ms
      }
      Clr_DZ();             //清屏
  }
}
//循环显示"WWW.TXMCU.COM"
void Show_WWW(void)
{
```

```
    U8 i=0,j=0,t;
    for(i=0;i<104;i++)
    {
        for(t=0;t<5;t++)
        {
            for(j=0;j<8;j++)
            {
                *LedDBuffer=(ZI_MO[j+i]|0xff00);
                //Send_595(~(1<<i));//低电平点亮
                                    //取字模时取反码
                Send_595(~(1<<(7-j)));
                sysUtilsUSecDelay(200);
                Clr_DZ();           //清屏
            }
        }
    }
}
//继电器测试
void Test_Relay(void)
{
    U8 i;
    for (i=0;i<5;i++)
    {
        rPDATC |=MASK_BIT(12);
        sysUtilsUSecDelay(500000);
        rPDATC &=~MASK_BIT(12);
        sysUtilsUSecDelay(500000);
    }
}
/*****************************************************
  参数:dwLedNo     指定对哪个 LED 操作
       :bLight      指定点亮或者熄灭.TRUE  点亮,FALSE 熄灭
*****************************************************/
void sysUtilsLightLed (U32 dwLedNo, BOOL bLight)
{
    if(dwLedNo==LED_ALL)
    {
```

```
        if(bLight==TRUE)
            rPDATD&=~(0xff);
        else
            rPDATD|=0xff;
    }
    else
    {
        if(bLight==TRUE)
            rPDATD&=~(1<<(dwLedNo-1));
        else
            rPDATD|=(1<<(dwLedNo-1));
    }
}
```

5.19 DS18B20 数字温度传感器实验

原理图



C 代码

```
#include "..\inc\main.h"
//我们用这个片编存储我们的 IRQ ISR 入口地址
U32 pIrqStart = 0;
U32 pIrqFinish = 0;
U32 pIrqHandler = 0;
/*主函数*/
int main(void)
{
```

```
    //U16 i;
    sysUtilsSetPllValue (24, 6, 1); //设置片内的锁相环
    IoConfig ();//io 端口的初始化
    Clr_Pled();      //关闭跑马灯
    Clr_DZ();        //关闭点阵显示
    Clr_led();       //关闭数码管显示
    console_init (115200);     //初始化控制台
    printf("Test 18b20.....\n");
    while(1)
        Test_18b20();
    printf("Test over!\n\n");
    while(1);
    return 0;
}
```

Test_18b20()代码

```
#include "..\inc\uTypes.h"
#include "..\inc\sysUtils.h"
#include "..\inc\ds18b20.h"
#include "..\inc\console.h"
#define   ESC_KEY        0x1b
//DS18B20 的端口为 GPF4
#define DS_H() rPDATF|=(1<<4)      //端口高
#define DS_L() rPDATF&=~(1<<4)         //端口低
#define DS_OUT() rPCONF|=(01<<(4*2));rPCONF&=(01<<(4*2))   //端
口为输出
#define DS_IN() rPCONF&=~(11<<(4*2))                        //端口为
输入
#define DS_R() (rPDATF&(1<<4))
U16 temp;                       //variable of temperature
/*************************
函数：void DS_Reset(void)
功能：18b20 复位
*************************/
void DS_Reset(void)         //send reset and initialization command
{
    DS_OUT();
    DS_L();
```

```
    sysUtilsUSecDelay(700);
    DS_H();
    sysUtilsUSecDelay(4);
    DS_IN();
    sysUtilsUSecDelay(100);
    sysUtilsUSecDelay(250);//等待低电平过去
}
/*********************************
函数：U8 tmpreadbit(void)
功能：读取 18b20 的一位数据
*********************************/
U8 tmpreadbit(void)
{
    U8 dat;
    DS_OUT();
    DS_L();
    sysUtilsUSecDelay(2);
    DS_IN();
    sysUtilsUSecDelay(10);
    if(DS_R()!=0)
        dat=1;
    else
        dat=0;
    sysUtilsUSecDelay(50);
    return (dat);
}
/*********************************
函数：U8 tmpread(void)
功能：读取一个字节的数据
*********************************/
U8 tmpread(void)
{
    U8 i,j,dat;
    dat=0;
    for(i=1;i<=8;i++)
    {
        j=tmpreadbit();
        dat=(j<<7)|(dat>>1);     //读出的数据最低位在最前面，这样刚
```

好一个字节在 DAT 里

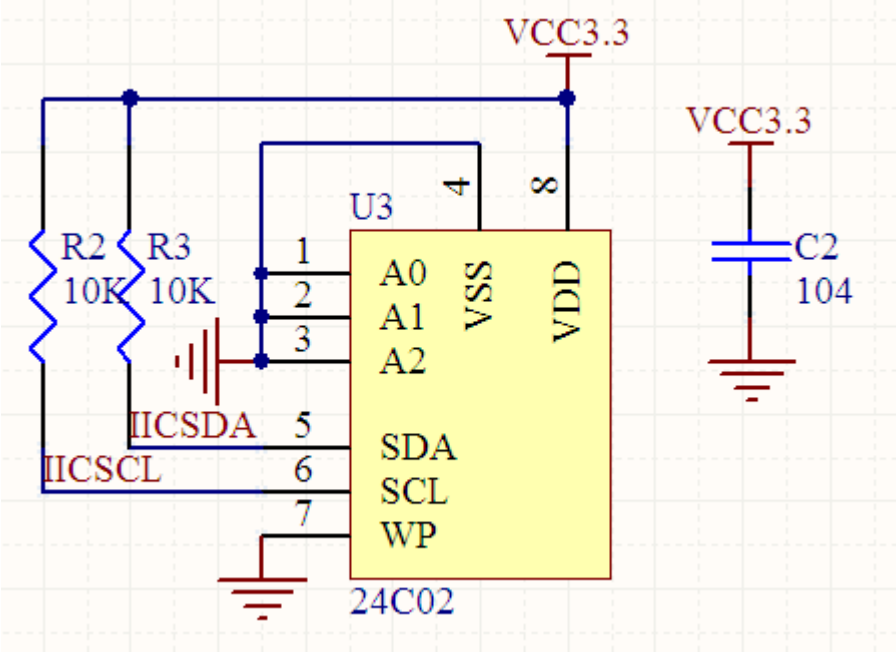
```
    }
    return(dat);
}
/*********************************
函数：void tmpwritebyte(U8 dat)
功能：向 18b20 写入一个字节数据
*********************************/
void tmpwritebyte(U8 dat)
{
    U8 j;
    U8 testb;
    DS_OUT();
    for(j=1;j<=8;j++)
    {
        testb=dat&0x01;
        dat=dat>>1;
        if(testb)
        {
            DS_L();
            sysUtilsUSecDelay(8);
            DS_H();
            sysUtilsUSecDelay(50);
        }
        else
        {
            DS_L();
            sysUtilsUSecDelay(90);
            DS_H();
            sysUtilsUSecDelay(8);
        }
    }
}

void Tmp_Change(void)    //DS18B20 begin change
{
    DS_Reset();
    sysUtilsUSecDelay(2);/*delay(1);*/
```

```
    tmpwritebyte(0xcc);    // address all drivers on bus
    tmpwritebyte(0x44);    //    initiates a single temperature conversion
}

U16 tmp(void)                          //get the temperature
{
    float tt;
    U8 a,b;
    DS_Reset();
    sysUtilsUSecDelay(4);/*delay(1);*/
    tmpwritebyte(0xcc);
    tmpwritebyte(0xbe);
    a=tmpread();
    b=tmpread();
    temp=b;
    temp<<=8;                      //two byte    compose a int variable
    temp=temp|a;
    tt=temp*0.0625;
    temp=tt*10+0.5;
    return temp;
}

void readrom(void)                 //read the serial
{
    U8 sn1,sn2;
    DS_Reset();
    sysUtilsUSecDelay(4);
    tmpwritebyte(0x33);
    sn1=tmpread();
    sn2=tmpread();
}

//18b20 测试
void Test_18b20(void)
{
    U16 i;
    while( !( kbhit() && (getkey()==ESC_KEY)))
    {
```

```
        Tmp_Change();
        sysUtilsUSecDelay (5000);    //延迟 5ms
        i=tmp();
        printf("Now temperature is %d.%d 'C.\n",i/10,i%10);
        sysUtilsUSecDelay (500000);    //延迟 500ms
        }
}
```

5.20 IIC 总线 AT24C02 存储器实验

原理图



C 代码

```
#include "..\inc\main.h"
//我们用这个片编存储我们的 IRQ ISR 入口地址
U32 pIrqStart = 0;
U32 pIrqFinish = 0;
U32 pIrqHandler = 0;
/*主函数*/
int main(void)
{
   //U16 i;
   sysUtilsSetPllValue (24, 6, 1); //设置片内的锁相环
```

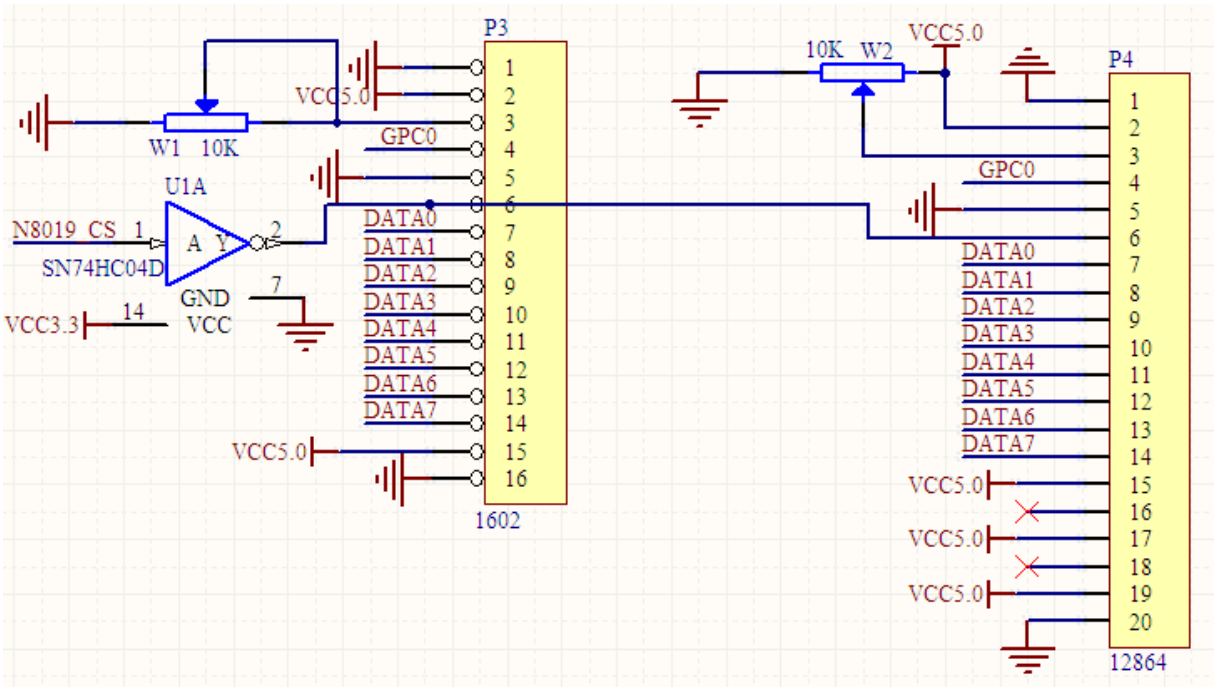
```
    IoConfig ();//io 端口的初始化
    console_init (115200);     //初始化控制台
    printf("Test 24c02.....\n");
    while(1)
        Test_Iic();
    printf("Test over!\n\n");
    while(1);
    return 0;
}
```

Test_Iic()代码

代码过长，请查看光盘。

5.21 1602 液晶实验

原理图



C 代码

```
#include "..\inc\main.h"
//我们用这个片编存储我们的 IRQ ISR 入口地址
U32 pIrqStart = 0;
U32 pIrqFinish = 0;
U32 pIrqHandler = 0;
```

```
/*主函数*/
int main(void)
{
   //U16 i;
   sysUtilsSetPllValue (24, 6, 1); //设置片内的锁相环
   IoConfig ();//io 端口的初始化
   console_init (115200);     //初始化控制台
   printf("Test 24c02.....\n");
   while(1)
        Lcd_1602Test();
   printf("Test over!\n\n");
   while(1);
   return 0;
}
```

Lcd_1602Test()代码

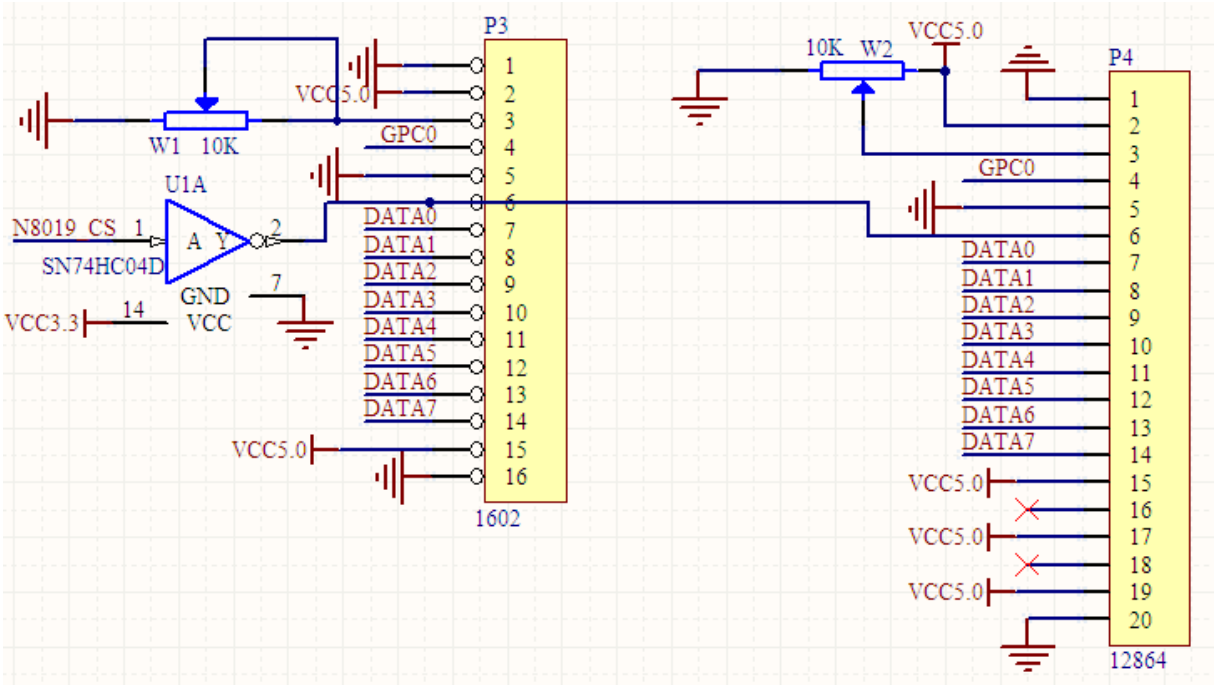
```
#include "..\inc\1602.h"
#include "..\inc\uTypes.h"
#include "..\inc\sysUtils.h"

#define LCD1602_WR_H() rPDATC|=0x01;
#define LCD1602_WR_L() rPDATC&=~0x01;

//总线地址声明
volatile U8 * LCD1602_MAdd = (volatile U8 *)(0x6000000);

U8 table1[]="TX-1B MCU";
U8 table2[]="WWW.TXMCU.COM";

//写命令
void write_com(U8 com)
{
   LCD1602_WR_L();
   sysUtilsUSecDelay(1);
   * LCD1602_MAdd=com;
   sysUtilsUSecDelay(15000);
}
//写数据
void write_date(U8 date)
```

```
{
   LCD1602_WR_H();
   sysUtilsUSecDelay(1);
   * LCD1602_MAdd=date;
   sysUtilsUSecDelay(15000);
}

void Init_1602(void)
{
   write_com(0x38);
   write_com(0x0f);
   write_com(0x06);
   write_com(0x01);
}

void Lcd_1602Test(void)
{
   U8 a;

   Init_1602();

   write_com(0x80+17);
   sysUtilsUSecDelay(2000);
   for(a=0;a<9;a++)
   {
       write_date(table1[a]);
       sysUtilsUSecDelay(2000);
   }

   write_com(0xc0+17);
   sysUtilsUSecDelay(5000);
   for(a=0;a<13;a++)
   {
       write_date(table2[a]);
       sysUtilsUSecDelay(4000);
   }

   for(a=0;a<16;a++)
```

```
    {
        write_com(0x18);
        sysUtilsUSecDelay(250000);
    }
}。
```

5.22 12864 液晶实验

原理图



C 代码

```
#include "..\inc\main.h"
//我们用这个片编存储我们的 IRQ ISR 入口地址
U32 pIrqStart = 0;
U32 pIrqFinish = 0;
U32 pIrqHandler = 0;
/*主函数*/
int main(void)
{
   //U16 i;
   sysUtilsSetPllValue (24, 6, 1); //设置片内的锁相环
   IoConfig ();//io 端口的初始化
   console_init (115200);     //初始化控制台
```

```
    printf("Test 24c02.....\n");
    while(1)
        LCD12864_Test();
    printf("Test over!\n\n");
    while(1);
    return 0;
}
```

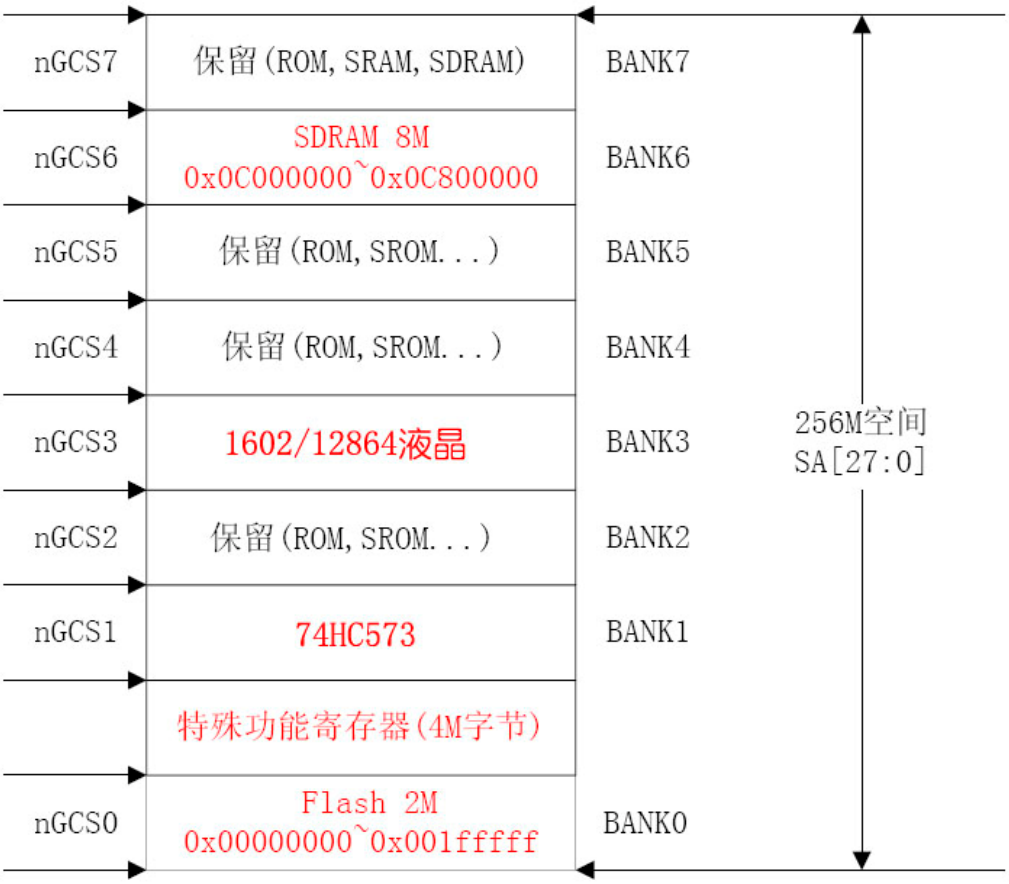
LCD12864_Test()代码

```
#include "..\inc\12864.h"
#include "..\inc\sysUtils.h"

//总线地址声明
volatile U8 * LCD12864_MAdd = (volatile U8 *)(0x6000000);
#define LCD12864_WR_H() rPDATC|=0x01;
#define LCD12864_WR_L() rPDATC&=~0x01;

void lcd_wcmd(U8 cmd);
void lcd_wdat(U8 dat);
void lcd_init(void);
void lcd_pos(U8 X,U8 Y);

static U8 dis1[] = {"天祥电子"};
static U8 dis2[] = {"www.txmcu.com"};
static U8 dis3[] = {"TX-1C 学习板  "};
static U8 dis4[] = {"是您最好的选择！   "};

U8 IRDIS[2];
U8 IRCOM[4];

void lcd_wcmd(U8 cmd)
{
    sysUtilsUSecDelay(150000);
    LCD12864_WR_L();
    * LCD12864_MAdd=cmd;
    sysUtilsUSecDelay(15000);
}
void lcd_wdat(U8 dat)
{
```

```
    sysUtilsUSecDelay(150000);
    LCD12864_WR_H();
    * LCD12864_MAdd=dat;
    sysUtilsUSecDelay(15000);
}
/**LCD 初始化设定****/
void lcd_init(void)
{
    //LCD_PSB = 1;                //并口方式
    lcd_wcmd(0x34);               //扩充指令操作
    sysUtilsUSecDelay(5000);/*delay(5);*/
    lcd_wcmd(0x30);               //基本指令操作
    sysUtilsUSecDelay(5000);/*delay(5);*/
    lcd_wcmd(0x0C);               //显示开，关光标
    sysUtilsUSecDelay(5000);/*delay(5);*/
    lcd_wcmd(0x01);               //清除 LCD 的显示内容
    sysUtilsUSecDelay(5000);/*delay(5);*/
}
/************************************************************
函数：void LCD12864_Test(void)
功能：LCD12864 测试程序
************************************************************/
void LCD12864_Test(void)
{
      U8 i;
      sysUtilsUSecDelay(5000);/*delay(10);*/        //延时
      lcd_init();                          //初始化 LCD
      lcd_pos(0,0);       //设置显示位置为第一行的第 1 个字符
      i = 0;

      while(dis1[i] != '\0')
      {                                    //显示字符
        lcd_wdat(dis1[i]);
        i++;
      }

      lcd_pos(1,0);         //设置显示位置为第二行的第 1 个字符
      i = 0;
```

```
        while(dis2[i] != '\0')
        {
         lcd_wdat(dis2[i]);          //显示字符
         i++;
        }

    lcd_pos(2,0);          //设置显示位置为第三行的第 1 个字符
        i = 0;
        while(dis3[i] != '\0')
        {
              lcd_wdat(dis3[i]);          //显示字符
              i++;
        }

    lcd_pos(3,0);          //设置显示位置为第四行的第 1 个字符
        i = 0;
        while(dis4[i] != '\0')
        {
              lcd_wdat(dis4[i]);          //显示字符
              i++;
        }
}

/*  设定显示位置  */
void lcd_pos(U8 X,U8 Y)
{
    U8    pos;
    if (X==0)
       {X=0x80;}
    else if (X==1)
       {X=0x90;}
    else if (X==2)
       {X=0x88;}
    else if (X==3)
       {X=0x98;}
    pos = X+Y ;
    lcd_wcmd(pos);        //显示地址
}
```

第六章 系统片选及地址空间分页



文件名:	Rescue.doc
目录:	C:\Documents and Settings\lianzi\My Documents
模板:	C:\Documents and Settings\lianzi\Application Data\Microsoft\Templates\Normal.dotm
标题:	
主题:	
作者:	
关键词:	
备注:	
创建日期:	2008-12-14 16:25:00
修订号:	23
上次保存日期:	2008-12-14 23:29:00
上次保存者:	
编辑时间总计:	161 分钟
上次打印时间:	2008-12-14 23:33:00
打印最终结果	
页数:	76
字数:	5,033 (约)
字符数:	28,693 (约)