西 安 邮 电 大 学

(计算机学院)

嵌入式系统板级电路装配

课程设计实验报告

专业名称： 计算机科学与技术

班　　级： 计科 1801

学　　号： 04181016

姓　　名： 周铭海

指导教师： 马博

实验日期： 2021 年 11 月 22 日—12 月 03 日

第一周：开发板硬件装配

一、 开发板硬件结构

开发板由 PACK 板和底板构成。

PACK 板板载一枚基于 ARM7 TDMI-S 核、LQFP48 封装的 LPC2132 芯片，该芯片是 NXP（飞利浦创建）设计的一款基于 ARM7TDMI-S 的高性能 32 位 RISC 微控制器，具有 32KB 片内 Flash，8KB 片内 SRAM，具有 JTAG 仿真调试和 ISP 编程功能。

底板上包括有按键、发光二极管等常用的功能器件，具有 RS-232 接口电路和 I2C 存储器电路。LPC2132 的引脚可以与这些功能器件相连，方便用户连接外部电路进行开发。可以通过 RS-232 转换电路，与上位机进行 UART 通信。

板子总体分为电源电路，晶振电路，复位电路，LED 电路，按键电路，串口电路，JTAG 调试电路等几部分，如下图所示：

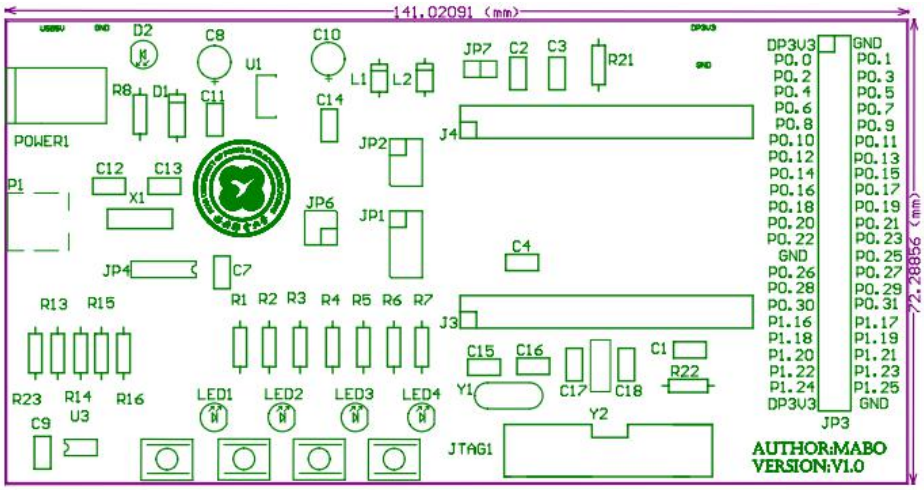

图 1　开发板底板元件布局图

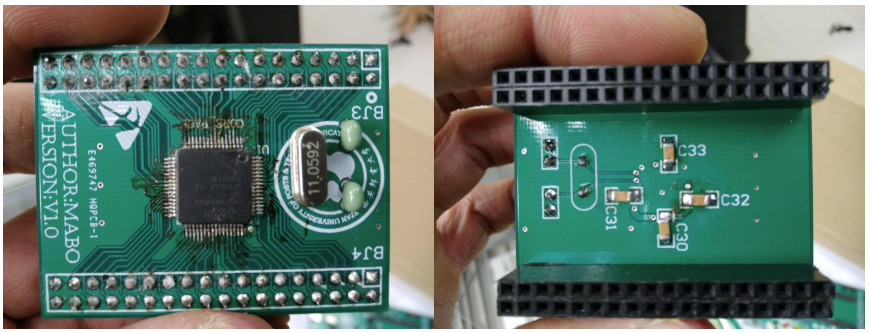

图 2　PACK 板正反面布局图

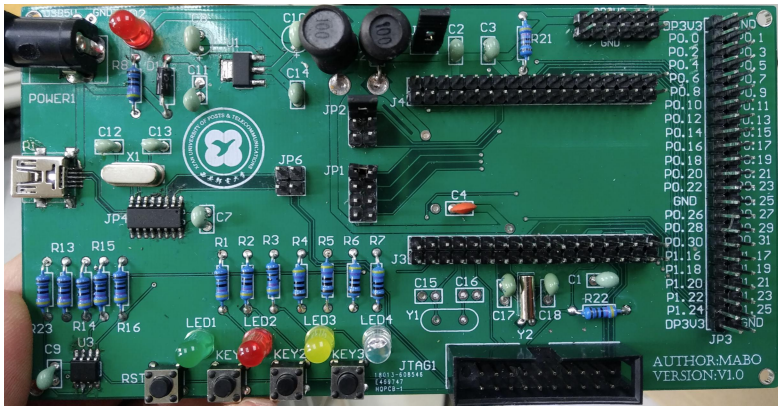

图 3　开发板底板实物图

二、 硬件组成及分析

1、电源电路

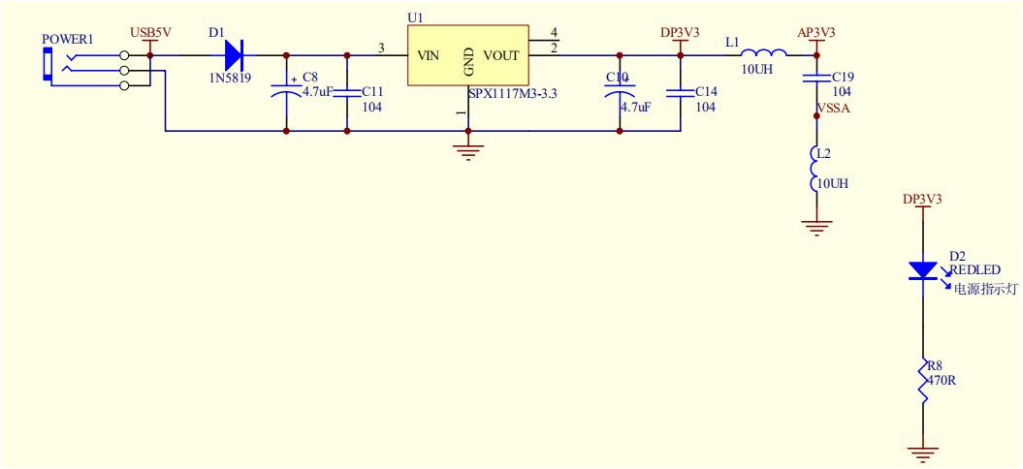

图 4　电源电路原理图

LPC2132 控制器需要双电源供电，1.8V 内核电压和 3.3V 功能外设电压，开发板的电源电路如图 4 所示。5V 电源由变压器或 USB 电源线输入，内部采用了二极管 1N5819 来稳流。电路采用 SPX1117 系列 LDO 芯片 SPX1117M-3.3 和 SPX1117M-1.8 将电压稳至 3.3V 和 1.8V，0 欧的电阻用来隔离数字电源和模拟电源、数字地和模拟地。SPX1117 系列 LDO 芯片是 EXAR 公司生产的低压差，其特点是输出电流大，输出电压精度高，稳定性高，宽电压输入（这里选择的是 5V 输入）。在电路输入输出端接一个 10μF 的电容，可改善瞬态响应和稳定性。

2、复位和 I2C 电路

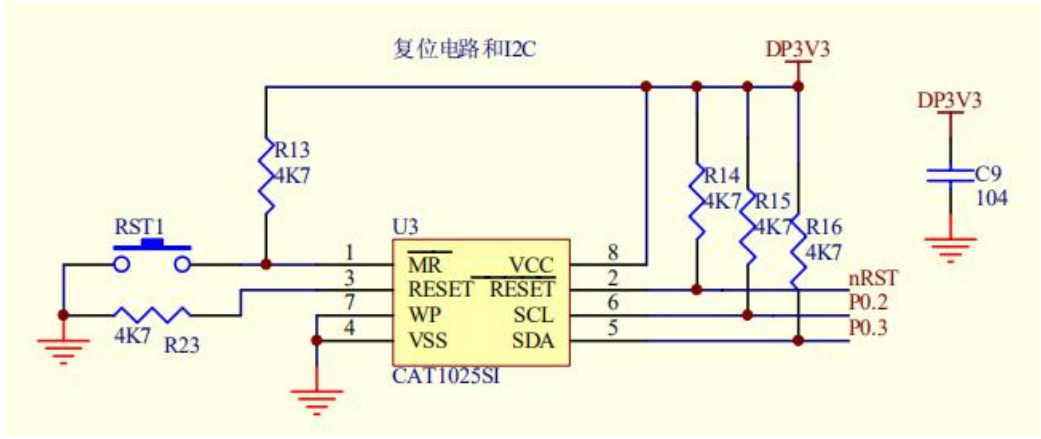

图 5　复位和 I2C 电路原理图

复位电路采用一颗 CAT1025 芯片，CAT1025 是基于微控制器系统的存储器和电源监控的完全解决方案。它们利用低功耗 CMOS 技术将 2K 位的串行 EEPOM 和用于掉电保护的系统电源监控电路集成在一块芯片内。存储器采用 400KHz 的 I2C 总线接口。

CAT1025 包含 2 个开漏输出:$\overline{RESET}$ 和 RESET 以及 1 个精确的 VCC 监测电路。当 VCC 低于复位门槛电压时，RESET 将变成低电平。CAT1025 还包含一个写保护输入(WP)，如果 WP 连接高电平，则写操作被禁止。

nRST 连接到 LPC2132 的复位引脚，当复位按键 RST1 按下时，CAT1025 的复位引脚输出低电平，使 LPC2132 复位。

3、时钟电路

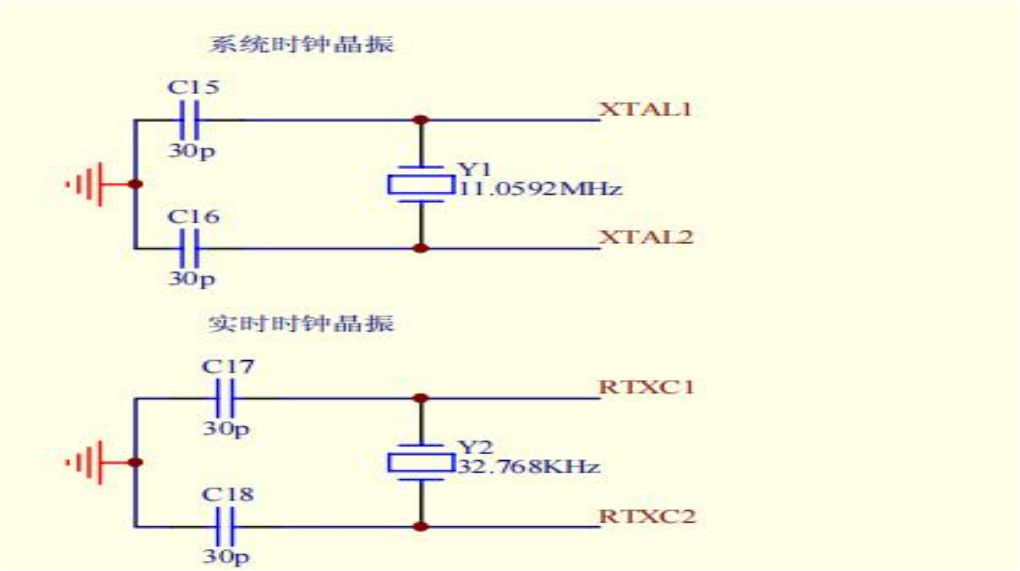

图 6　时钟电路原理图

LPC2132 微控制器可使用外部晶振或外部时钟源，内部 PLL 电路可调整

系统时钟，使系统运行速度更快（CPU 的操作频率最大可达 70MHz）。若不使用片内 PLL 功能及 ISP 下载功能，则外部晶振频率为 1～30MHz，外部时钟频率为 1～50MHz；若使用片内 PLL 功能或 ISP 下载功能，则外部晶振频率为 10～25MHz，外部时钟频率为 10～25MHz。

开发板使用外部晶振 11.0592MHz，实时时钟为 32.768KHz，电路原理如图 6 所示。用 11.0592MHz 的外部晶振使串口的波特率更精确，同时能支持 LPC2132 微控制器内部的 PLL 电路及 ISP（在系统编程）功能。

4、JTAG 接口电路

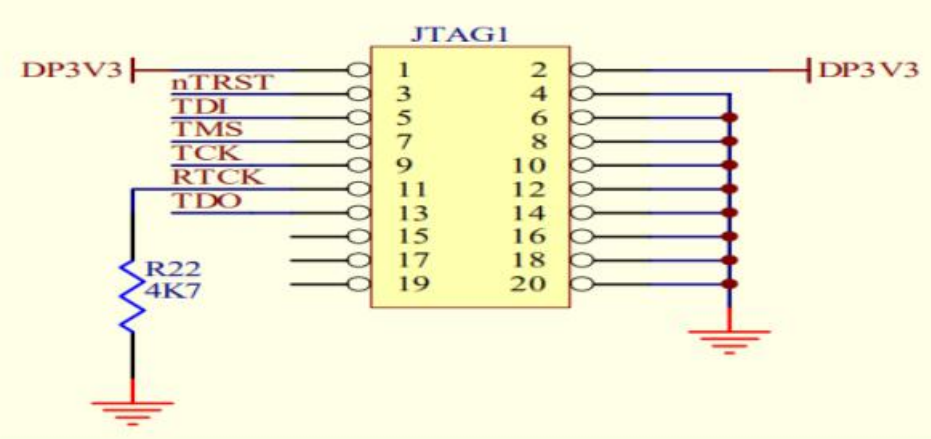

图 7　JTAG 接口电路原理图

JTAG 接口电路采用 ARM 公司提出的标准 20 脚 JTAG 仿真调试接口，JTAG 接口 LPC2132 引脚之间的连接如图 7 所示。

5、UART 接口电路

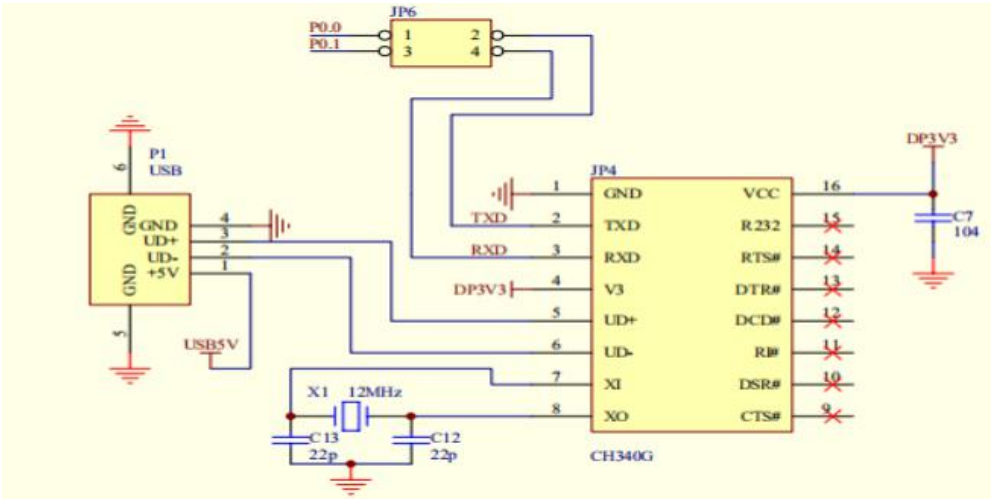

图 8　UART 接口电路原理图

CH340G 是一个 USB 转串口芯片，可以把电脑的 USB 口映射为串口使用。当使用串口电路进行 UART 调试的时候，需要将 JP6 短接，连通 P0.0

和 CH340G 的 TXD 口，连通 P0.1 和 CH340G 的 RXD 口。另一边 CH340G 的 X1 和 X0 接入了 X1-12MHz 的晶振。

6、按键与显示电路

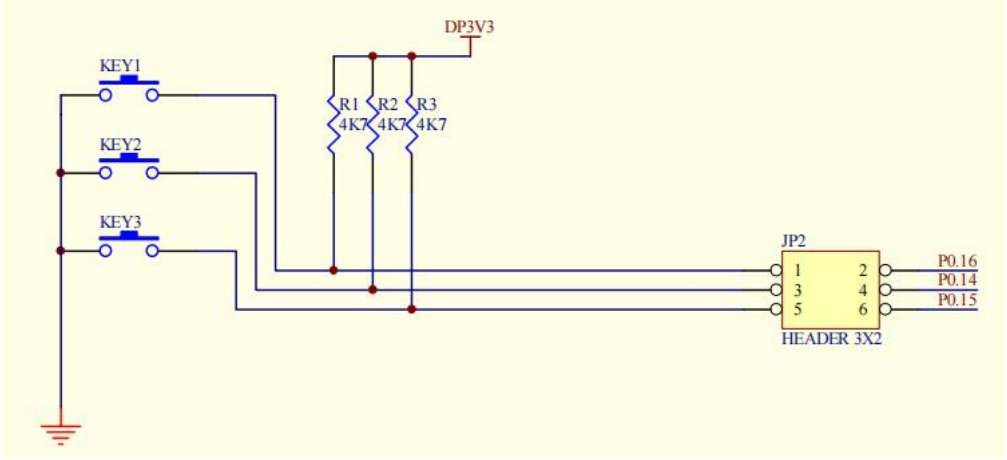

图 9　按键电路原理图

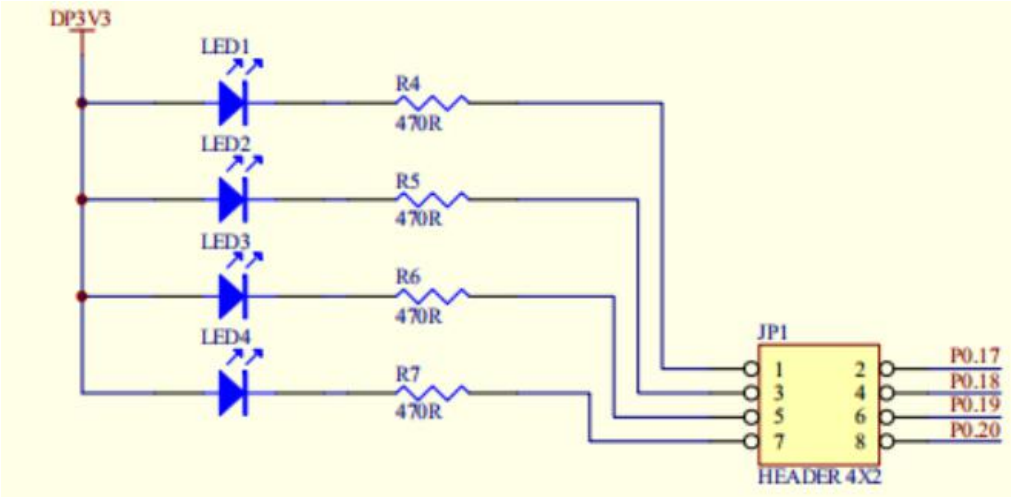

图 10　显示电路原理图

开发板具有 4 个按键、4 个 LED 灯。4 个 LED 灯一边已经与 DP3V3 连接，另一边经过 1 个 470R 的电阻后与 JP1 跳线相连，当 JP12 连通的时候，LED1 的一侧与芯片的 P0.17 口就会连通,P0.17 口输出低电平，LED1 就会点亮，如果 PO.17 输出高电平，LED1 就会熄灭。

4 个按键中复位键已经在前面介绍了，剩下三个按键 KEY1、KEY2、KEY3 一边接地，另一边接了 4K7 的电阻然后与正极相连,还接了 JP2 跳线组，如果 JP2 的 12 连通，那么 P0.16 就和 KEY1 连通，当按键按下的时候，P0.16 输出低电平，当按键没有按下的时候 P0.16 输出高电平，因为按键检测电路可以编写为检测是否有低电平来判断按键是否按下。

三、 开发板硬件安装调试过程

1、常见电子元件的识别与检测

（1）电阻：

识别：色环标注法。本次实验中使用的是五色环电阻，最后一环与前面四环距离较大。前三环为有效数字，第四环为倍率，第五环为误差。阻值=（第 1 色环数值*100+第 2 色环数值*10+第 3 色环数值）*第 4 色环代表的倍数。

表 1　色环电阻对照表

	有效数字	数量级	允许偏差
银	-	10^-2	10
金	-	10^-1	5
黑	0	10^0	-
棕	1	10^1	1
红	2	10^2	1
橙	3	10^3	-
黄	4	10^4	-
绿	5	10^5	0.5
蓝	6	10^6	0.25
紫	7	10^7	0.1
灰	8	10^8	0.05
白	9	10^9	-

检测：用万用表进行检测。先连接表笔，旋转万用表档位至欧姆档，选定电阻范围，在测量前将红笔和黑笔前端放在一起，使显示为 0；用表笔接触电阻两端，无正负之分，确保接触良好，读出万用表显示数据。

（2）电容：

识别：①直接标注：电容器电容量数值等参数直接标在电容器表面。②用数字标注容量：标在电容器表面上的是三位整数，其中第一、二位分别表示容量的有效数字，第三位数字表示容量的有效数字加零的个数。数码法表示电容量时，单位一律是 pF。

检测：用数字万用表检测电容器充放电现象：将数字万用表拨至适当的电阻档档位，万用表表笔分别接在被测电容 C 的两引脚上，这时屏幕显示值从“000”开始逐渐增加，直至屏幕显示“1”。然后将两表笔交换后再测，显示屏上瞬间显示出数据后立刻变为“1”，此时为电容器放电后再反向充电，证明电容器充放电正常。

（3）二极管：

识别：发光二极管的正负可从引脚的长短来判断，长脚为正,短脚为

负。也可以从内部看到，接触面小的为正，接触面大的为负。外边有切口的为负，另一边就为正。

检测：在接通电源后，将红黑表笔放置在两端，选择数字万用表的二级管测试，正向测出电压 3.3V 左右，反向显示为无穷大。

2、电路板焊接

（1）手工焊接方法及步骤：

① 整理好工作台，将烙铁、焊锡以及被焊工件准备好，保持烙铁以及被焊工件的干净，没有其他杂质影响焊接。

② 接通电源，加热烙铁。将烙铁接触焊接点，保持焊件的受热均匀。

③ 将焊锡放在工件上，熔化适量的焊锡，并且保证焊锡一定要润湿被焊工件表面和整个焊盘。

④ 等焊锡充满焊接点后，立即将焊锡沿着元件引线的方向向上提起焊锡。

⑤ 当焊锡的扩张范围达到要求后，快速沿着元件引线的方向向上拿开烙铁。

（2）注意事项：

① 保持烙铁头和焊接点的干净。烙铁长期使用会有一层黑色的杂质，影响焊接效果。

② 烙铁头与工件加热时，应该使其接触面积大一些，而不是在一个点上加热。

③ 焊接时要保持工件的稳定，焊接过程中不能产生较大的晃动。

④ 要移动工件时，要等融化的焊锡冷却凝固后再移动。

⑤ 添加焊锡时要适量，焊锡过少会导致虚焊；焊锡过多会影响装配。

3、电路板测试

（1）电源电路

将电源线和开发板的电源模块连接，观察电源指示灯是否点亮，如果点亮，使用万用表测量 U1 点电压，数值范围为：3.29V~3.31V，标准值为 3.30V，表明电源电路模块正常。

（2）复位电路

在按下复位按钮时，CAT1025 芯片复位引脚处电压会发生变化。

（3）时钟电路

使用示波器观察晶振引脚的波形，查看是否正常。

（4）显示电路

给开发板通电，然后依次给 JP1 的 1、3、5、7 接低电平，会发现 LED1-LED4 依次点亮，如果有不亮的，说明焊接有误，检查二极管的正负是否正确，用万用表检测是否短路。

（5）按键电路

给开发板供电，用万用表检测 KEY1-KEY3 的电压，正常范围应该是 3.28V-3.31V，当 KEY1 键按下的时候，JP2 的 1 电压应该为 0V；当 KEY2 键按下的时候，JP2 的 3 电压应该为 0V；当 KEY3 键按下的时候，JP2 的 4 电压应该为 0V。

（6）串口电路

先给开发板供电，用 USB 口与电脑连接，在设备管理器可以检测到串口的输入。

四、 遇到问题分析及硬件调试体会

1、问题分析与解决方法

在焊接完 USB 串口部分并进行测试的时候，可以在自己的电脑上进行串口发现，但在另一台电脑上发现不了串口。

发现问题后，首先排查了是否是没有驱动，然后两台电脑都重新安装了驱动，问题依然存在。开始检查焊接，没有发现有漏焊的引脚。问题定位在串口是否是接触不良。由于串口焊接时，部分引脚排布紧密，而且所占面积小，在第一次焊接时只用了一点点的焊锡进行焊接。然后重新将焊锡熔化一小滴在烙铁头，对每一个引脚都进行加固。然后经过测试，发现可以检测到串口，证实是串口接触不良。问题解决。

2、硬件调试体会

在进行底板焊接时，不能上来就开始焊，最首先的应该做好前期准备工作。包括焊接工具的准备，以及焊接工件的原理图。其次开始将元器件按照原理图插入相应的位置，并且保证准确无误。我和搭档在焊接每一个元器件的时候，进行了双重校验，两个人分别确认型号以及焊接的位置是否有误，比如 LED 灯的正负极、电容电阻的型号等。焊接时应该按照功能模块来进行，而且每焊接完一个模块都先进行校验，保证功能正常后才进行下一个模块的焊接。

在整个焊接和调试的过程中，需要有耐心和细心，整个过程都应该保持专注。正是因为我和搭档全程都在认真焊接和调试，基本上可以保证，当整个底板焊接完成后，没有出现其他重大错误及失误，也为了后续软件部分的顺利开发及调试打下了坚实基础。

第二周：软件编程与调试

一、 调试环境搭建

开发环境：Windows10_x86_64

IDE：MDK-ARM PLUS Version：5.36.0.0

1. 创建项目工程

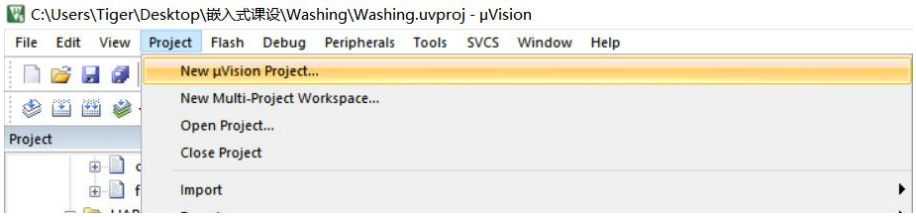

图 11　创建项目工程

在“Project”选项中点击“New µVision Project”，输入项目名即可。

2. 创建工程文件

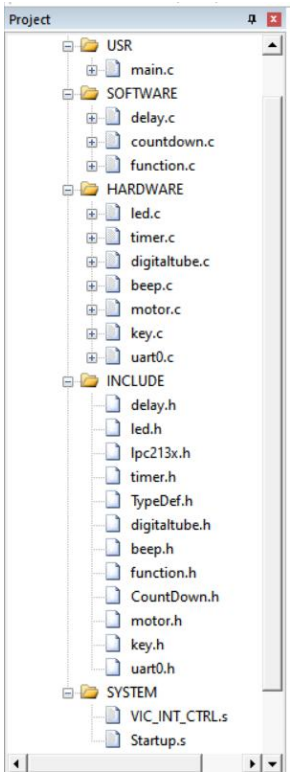

图 12　创建工程文件

然后在创建好的工程中创建目录，并添加文件。“SOFTWARE”中放置的是倒计时模块、软件延时模块、功能函数模块代码；“HARDWARE”中放置的是与外设连接的模块，包括控制 LED 灯、数码管、蜂鸣器、电机、串

口、定时器中断等代码；“INCLUDE”中放置的是各源代码对应的头文件；“USR”中放置主函数的入口。

3． 安装串口驱动

图 13　安装串口驱动

安装驱动时可能会出现安装失败的情况，这时先点击“卸载”，然后再点击安装即可。

4． 进行下板配置

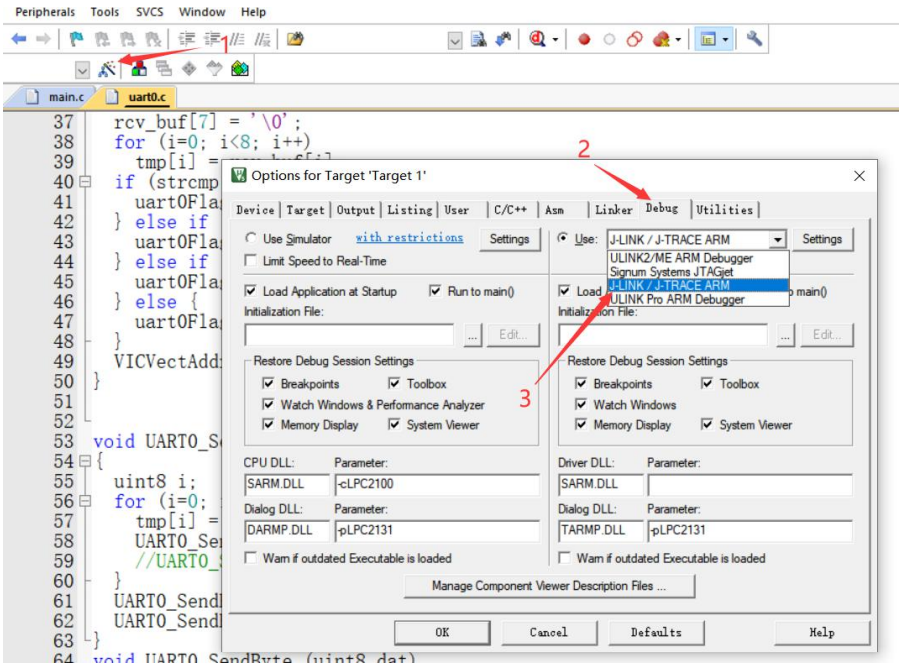

图 14　点击“Download”进行配置

点击“Download”图标，由于使用的调制器是 J-Link，在“Dubug”中选择“J-Link”。同时在“Device”中选择“LPC2132”。最后点击“OK”即可。

5． 编译工程，下板

图 15　点击“1”编译工程，“2”下板

先点击“Rebuild”，确保“0 error”后，再点击“Download”,即可下板成功。

二、完成实验内容

本人主要负责内容：编写基础实验代码并运行调试，在综合项目设计中负责框架总体设计、定时器中断模块、串口通信模块、PWM 电机控制模块。

1、GPIO 输入与输出

（1）实验目的

① 理解 ARM 芯片 GPIO 模块（部件）工作原理。

② 掌握具体的真实的 ARM 芯片（LPC2132）其 GPIO 模块。

③ 应用 GPIO 驱动数码管、电机。

④ 学会使用具体的开发板，查看其电路图，并编写程序对其进行控制。

（2）实验原理

LPC2132 芯片内部有 GPIO 模块部件，可以控制 GPIO 引脚的电平变化、也可以检测 GPIO 引脚的电平变化，通过芯片引脚供开发人员连接使用。LPC2132 控制器的引脚都具有多种功能，但是每个引脚在某一时刻只能选择一种功能，可以根据需要配置引脚功能，选择寄存器即可选择相应的功能。PINSEL0 和 PINSEL1 寄存器中的每两个位控制着一个引脚的功能，所以一个引脚最多可以有 4 种不同的功能选择。

表 2　GPIO 操作相关寄存器描述

通用名称	描述	访问类型	复位值
IOPIN	GPIO 引脚值寄存器，不管方向模式如何，引脚的当前状态都可以从该寄存器中读出。	只读	NA
IOSET	GPIO 输出置位寄存器。该寄存器控制引脚输出高电平，向某位写入 1 使对应引脚输出高电平，写入 0 无效。	读/置位	0x00000000
IOCLR	GPIO 输出置位寄存器。该寄存器控制引脚输出低电平，向某位写入 1 使对应引脚输出低电平，写入 0 无效。	只清零	0x00000000

IODIR	GPIO 方向控制寄存器。该寄存器单独控制每个 IO 口的方向，向某位写入 1 使对应引脚作为输出功能，写入 0 时作为输入功能。	读/写	0x00000000

（3）实验过程

在控制显示数码管，第一步要先分配 GPIO 引脚。本次实验提供的数码管是共阴极的，也就是 GPIO 输出高电平，数码管点亮。

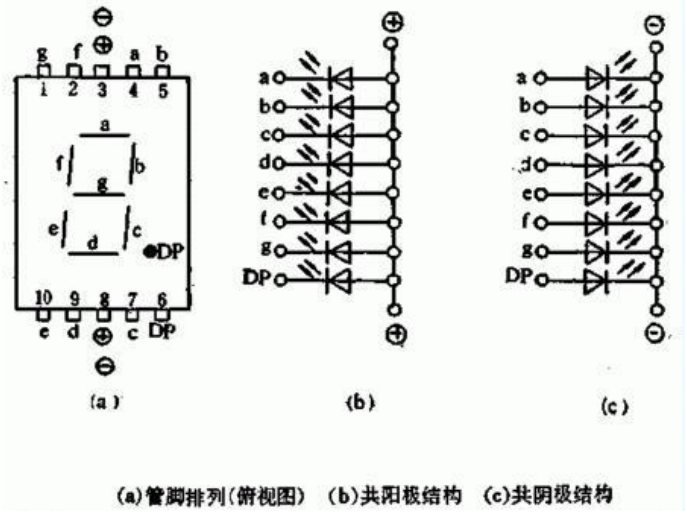

图 16　数码管原理图

这里数码管的引脚分配如表 5.2 所示：

表 3　数码管的引脚分配

数码管引脚	GPIO
a	P0.27
b	P0.25
c	P0.30
d	P0.28
e	P0.26
f	P0.29
g	P0.31

① 在“digitaltube.h”头文件中进行定义，代码如下：

```
#define DIGITALTUBE_A ((uint32)(1 << 27))    // P0.27 -- a
#define DIGITALTUBE_B ((uint32)(1 << 25))    // P0.25 -- b
#define DIGITALTUBE_C ((uint32)(1 << 30))    // P0.30 -- c
#define DIGITALTUBE_D ((uint32)(1 << 28))    // P0.28 -- d
```

```
#define DIGITALTUBE_E ((uint32)(1 << 26))    // P0.26 -- e
#define DIGITALTUBE_F ((uint32)(1 << 29))    // P0.29 -- f
#define DIGITALTUBE_G ((uint32)(1 << 31))    // P0.31 -- g

#define DIGITALTUBE_GPIO ((uint32)(0xFE000000))    // P[31:25]    数码管端口与 GPIO 的映射
#define DIGITALTUBE_0 ((uint32)(0x7E000000)) //数码管显示数字  0
#define DIGITALTUBE_1 ((uint32)(0x42000000)) //数码管显示数字  1
#define DIGITALTUBE_2 ((uint32)(0x9E000000)) //数码管显示数字  2
#define DIGITALTUBE_3 ((uint32)(0xDA000000)) //数码管显示数字  3
#define DIGITALTUBE_4 ((uint32)(0xE2000000)) //数码管显示数字  4
#define DIGITALTUBE_5 ((uint32)(0xF8000000)) //数码管显示数字  5
#define DIGITALTUBE_6 ((uint32)(0xFC000000)) //数码管显示数字  6
#define DIGITALTUBE_7 ((uint32)(0x4A000000)) //数码管显示数字  7
#define DIGITALTUBE_8 ((uint32)(0xFE000000)) //数码管显示数字  8
#define DIGITALTUBE_9 ((uint32)(0xFA000000)) //数码管显示数字  9

// 数码管显示数字 0
#define digitalTubeShow0() {\
    IO0CLR = DIGITALTUBE_GPIO;\
    IO0SET = DIGITALTUBE_0;\
}
// 数码管显示数字 1
#define digitalTubeShow1() {\
    IO0CLR = DIGITALTUBE_GPIO;\
    IO0SET = DIGITALTUBE_1;\
}
// 数码管显示数字 2
#define digitalTubeShow2() {\
    IO0CLR = DIGITALTUBE_GPIO;\
    IO0SET = DIGITALTUBE_2;\
}
// 数码管显示数字 3
#define digitalTubeShow3() {\
    IO0CLR = DIGITALTUBE_GPIO;\
```

```
    IO0SET = DIGITALTUBE_3;\
}
// 数码管显示数字 4
#define digitalTubeShow4() {\
    IO0CLR = DIGITALTUBE_GPIO;\
    IO0SET = DIGITALTUBE_4;\
}
// 数码管显示数字 5
#define digitalTubeShow5() {\
    IO0CLR = DIGITALTUBE_GPIO;\
    IO0SET = DIGITALTUBE_5;\
}
// 数码管显示数字 6
#define digitalTubeShow6() {\
    IO0CLR = DIGITALTUBE_GPIO;\
    IO0SET = DIGITALTUBE_6;\
}
// 数码管显示数字 7
#define digitalTubeShow7() {\
    IO0CLR = DIGITALTUBE_GPIO;\
    IO0SET = DIGITALTUBE_7;\
}
// 数码管显示数字 8
#define digitalTubeShow8() {\
    IO0CLR = DIGITALTUBE_GPIO;\
    IO0SET = DIGITALTUBE_8;\
}
// 数码管显示数字 9
#define digitalTubeShow9() {\
    IO0CLR = DIGITALTUBE_GPIO;\
    IO0SET = DIGITALTUBE_9;\
}

void digitalTubeInit(void);   // 数码管初始化
void digitalTubeShow(unsigned char);   // 数码管显示数字
```

② 在“digitaltube.c”文件中实现功能函数，代码如下：

```
#include "digitaltube.h"
void digitalTubeInit(void) {
  PINSEL1 &= 0x0003FFFF;
  IO0DIR |= DIGITALTUBE_GPIO;
  IO0CLR |= DIGITALTUBE_GPIO;
  return;
}
void digitalTubeShow(unsigned char num) {
  switch (num) {
        case 0: digitalTubeShow0(); break;
        case 1: digitalTubeShow1(); break;
        case 2: digitalTubeShow2(); break;
        case 3: digitalTubeShow3(); break;
        case 4: digitalTubeShow4(); break;
        case 5: digitalTubeShow5(); break;
        case 6: digitalTubeShow6(); break;
        case 7: digitalTubeShow7(); break;
        case 8: digitalTubeShow8(); break;
        case 9: digitalTubeShow9(); break;
  }
}
```

数码管显示数字的功能函数入口为 digitalTubeShow()函数，当传入一个参数 num，即可调用对应的显示函数，点亮对应的数码管引脚。

（4）问题分析与解决方法

问题：一开始对 IOSET 和 IOCLR 这两个寄存器的功能理解出现偏差，导致没有成功输出想要的结果。

解决方法：在确认 IOSET 读入 1 输出的是高电平、IOCLR 读入 1 输出的是低电平之后，就可以得到想要的结果。

2、定时器中断控制

（1）实验目的

① 了解有关定时器中断的相关应用。

② 了解 LPC2000 系列 ARM7 微控制器的定时器基本设置。

（2）实验原理

ARM 定时器/计数器结构原理图如图 17 所示，主要由：分配与计数模块、匹配控制模块和捕获控制模块三部分组成。与其他计算机外设一样，Timer 的各功能控制和工作状态标志，也都是通过对各相关寄存器编程实现。

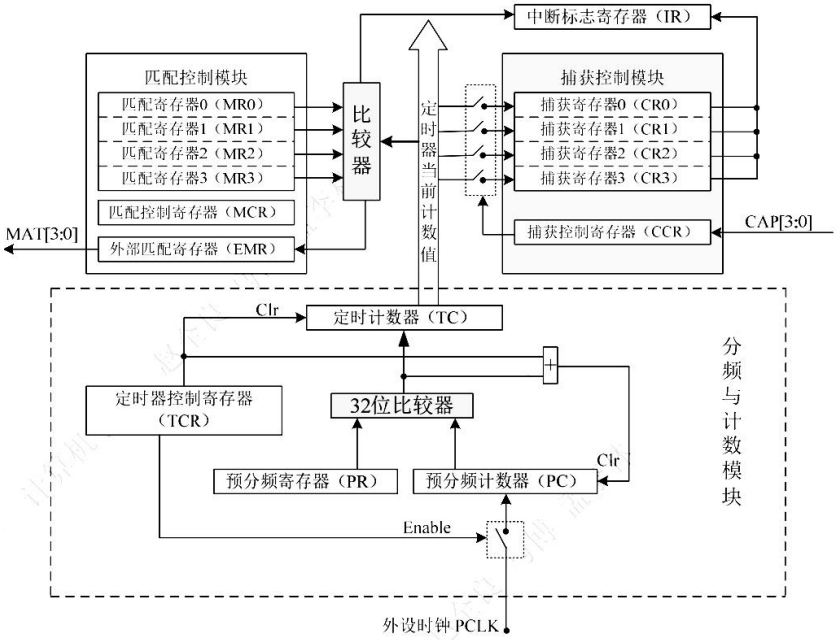

图 17　定时器/计数器结构原理图

每一个 Timer 中有 17 个寄存器，按寄存器对应的功能分为三类，用户通过对这 17 个寄存器的编程与访问，使用 Timer 资源的各项功能，Timer 寄存器说明如表 4 所示：

表 4　Timer 寄存器说明

分组	寄存器名称	寄存器说明
基础功能寄存器	IR	中断标志寄存器；8 个比特标示 8 个中断事件源的中断申请状态；写‘1’清 0 对应比特。
	TCR	定时器控制寄存器；用于控制 Timer 的使能与复位功能。
	TC	定时计数器；加 1 计数器，计数分频后的的脉冲数。
	PR	预分频寄存器；用于编程预分频值；

		电路按 PR+1 预分频。
	PC	预分频计数器；加 1 计数器，计数输入的脉冲数。
	CTCR	计数控制寄存器；用于选择 Timer 的“定时器模式”或“计数器模式”，并选择“计数器模式”下的信号方式和信号通道。
匹配功能寄存器	MCR	匹配控制寄存器；用于控制 MR0~MR3 四个通道匹配发生时是否产生中断申请、是否复位 TC 以及是否停止 TC 计数等。
	MRx	匹配寄存器 x(0-3)；用于预存匹配通道 x(0-3)的目标值。
	EMR	外部匹配寄存器；用于控制 4 个匹配通道匹配发生时所执行的操作。
捕获功能寄存器	CCR	捕获控制寄存器；用于控制 CR0~CR3 四个捕获通道的使能与否、捕获触发信号形式以及捕获事件发生时是否产生中断申请。
	CRx	捕获寄存器 x(0-3)；对应于 CAPn.x 捕获信道触发时，用于存储捕获时刻的 TC 计数当前值。

（3）实验过程

① 建立定时器中断的头文件“timer.h”：

```
#ifndef _TIMER_H
#define _TIMER_H
#include "TypeDef.h"
#include "led.h"

#define Fpclk 11059200 // 外设时钟频率  11.0592Mhz 晶振
void timer0Init(void);// 定时器初始化
void __irq IRQ_Timer0(void); // TC 的中断服务程序，MR0 和 MR1 与 TC 匹配时触发
extern  void IRQEnable(void); // 设置 CPSR 的 I 位开启 IRQ 中断
#endif
```

② 在“timer.c”中实现定时器中断功能：

```
#include "timer.h"
#include "CountDown.h"
```

```
void timer0Init(void){      // 定时器 0 初始化
    IRQEnable();            // 允许 IRQ 中断
    T0TCR = 0x3;            // 定时器 0 的 TC 和 PC 保持复位   11
    T0PR = 0;               // 预分频 PR 为 0，不分频
    T0MCR = 0x003;          // 设置 MRO 匹配时中断并复位 TC
    T0MR0 = Fpclk;          // 设置 TC 计数上限，当 TC 达到计数上限，则复位

    VICIntSelect &= 0xFFFFFFEF;   // 通道 4（Timer0 中断）设置为 IRQ 中断（对应位为 0）
    VICVectCntl0 = 0x20 | 0x04; // bit[5] 使能，bit[4:0] 通道号设置为 Timer0 的中断通道号 4
    VICVectAddr0 = (uint32)IRQ_Timer0;// 设置中断服务程序地址
    VICIntEnable = 1 << 4;  // 使能 Timer0 中断
    T0TCR = 0x1;   // 使能定时器 0，并清除复位
    return;
}

void __irq IRQ_Timer0(void){          //Timer0 中断服务程序
    countDownNum -= 1;   // 用于倒计时计数

    T0IR = 0x01;     // 清除 Timer0 中断标志寄存器 IR 的 bit[0]MR0
    VICVectAddr = 0x00; // 通知 VIC 中断处理结束
}
```

（4）问题分析与解决方法

问题：在实现定时器中断时，用示波器测量出比 1 秒要快。

解决方法：综合考虑是硬件结构本身的问题，可以适当地将 Fpclk 调大。

3、串口通信

（1）实验目的

① 了解串口通信原理。

② 掌握 UART 中断程序的设计。

③ 正确理解发送 FIFO 和接受 FIFO 的功能。

（2）实验原理

UART 是这样的一个硬件部件，在处理器对其编程设置了工作方式后，

UART 以寄存器接口的方式接受处理器并行（信息）数据，自动完成发送数据的组帧（字符帧），再将字符帧数据逐比特地传送出去。同时，也能逐比特地接收串行字符帧数据，并按编程设置的模式对收到的串行数据进行解帧，还原成并行信息数据，处理器以寄存器接口的方式从 UART 读取通信信息数据，如图 18 所示：

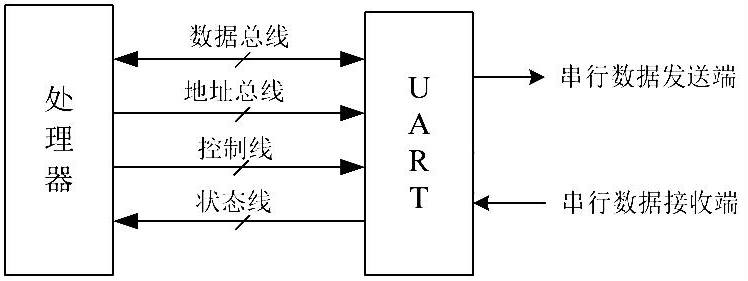

图 18　处理器与 UART

ARM 微控制器通用异步收发器（UART）主要由：串行数据发送模块、串行数据接收模块、波特率发生器模块、UART 控制与状态模块和中断控制逻辑模块等五部分组成，对于 UART1 还多了一个 Modem 控制模块。与其它计算机外设一样，UART 的各功能控制和工作状态标示，也都是通过相关的各个寄存器编程实现。

（3）实验过程

① 在头文件“uart0.h”中进行变量与函数的声明：

```
#ifndef UART0_H
#define UART0_H
#include "TypeDef.h"
#include "lpc213x.h"
volatile extern uint8 rcv_new;   // 接收数据标记
volatile extern char rcv_buf[8];   // 接收缓冲数据
volatile extern uint8 uart0Flag;
void uart0Init(void);   // 串口初始化
void __irq IRQ_Uart0(void);   // 中断服务程序
void UART0_SendBuf (void);   // 发送数据
void UART0_SendByte (uint8);
#endif
```

② 在“uart0.c”中进行功能函数的实现：、

```
#include "uart0.h"
#include "string.h"
volatile uint8 rcv_new = 0;
```

```
volatile char rcv_buf[8];
volatile uint8 uart0Flag = 0;
const char tmp1[8] = "Model_1";
const char tmp2[8] = "Model_2";
const char tmp3[8] = "Model_3";
char tmp[8];

void uart0Init() {
    PINSEL0 &= 0xFFFFFFF0;
    PINSEL0 |= 0x00000005;  // 设置 GPIO[1:0] 连接到 uart0
    U0LCR = 0x80;  // 通过波特率计算
    U0DLM = 0x00;
    U0DLL = 0x5A;
    U0LCR = 0x03;
    U0FCR = 0x81;  // 使能 FIFO，并设置触发点为 8 个字节
    U0IER = 0x01;  // 允许 RBR 中断，即接收中断
    VICIntEnable |= 1 << 0x06;  // 使能 UART0 中断, #UART0_INT = 6;
    VICIntSelect &= 0xFFFFFFBF; // 中断号 6
    VICVectCntl2 = 0x20 | 0x06; // uart0 分配到 IRQ slot0，即最高优先级
    VICVectAddr2 = (uint32)IRQ_Uart0;  // 中断服务程序
}
void __irq IRQ_Uart0() {
    uint8 i;
    if ((U0IIR & 0x0F) == 0x04)  {//IIR[0]==0,  有  UART_INT  ;
IIR[3:1]==010,UART 接收数据 INT
        rcv_new = 1;          // 设置接收到新的数据标志
        }
    for (i=0; i<7; i++)
        rcv_buf[i] = U0RBR;
    rcv_buf[7] = '\0';
    for (i=0; i<8; i++)
        tmp[i] = rcv_buf[i];
    if (strcmp(tmp, tmp1) == 0) {
        uart0Flag = 1;
    } else if (strcmp(tmp, tmp2) == 0) {
```

```
            uart0Flag = 2;
        } else if (strcmp(tmp, tmp3) == 0) {
            uart0Flag = 3;
        } else {
            uart0Flag = 0;
        }
        VICVectAddr = 0x00;                    // 中断处理结束
    }

    void UART0_SendBuf (void)
    {
        uint8 i;
        for (i=0; i<8; i++) {
            tmp[i] = rcv_buf[i];
            UART0_SendByte(tmp[i]);
        }
        UART0_SendByte(0x0d);
        UART0_SendByte(0x0a);
    }
    void UART0_SendByte (uint8 dat)
    {
        U0THR = dat;    // 要发送的数据
    }
```

③ 在 main.c 中调用 UART0_SendBuf()，使用上位机串口调试软件进行调试

（4）问题分析与解决方法

问题：进行 1 个字节传输的实验时，可以得到想要的结果。但进行 8 字节字符串传输的时候就会出现异常。

解决方法：理解 string.h 包下 strcmp()函数的用法，将串口发送来的数据暂存到一个 char tmp[8]中。同时要注意的是，要考虑接收到的数据是否带有结束符“\0”，如果没有带“\0”，遍历存储时要手动加上。

3、综合项目

（1）项目描述

项目名称：洗衣机控制系统。

洗衣机是一件常见的家用电器，为我们的生活带来了许多方便。我们这次的实验课题就是用现有的工具，模拟出家用洗衣机的功能，通过软硬件结合的方式，集成到一个系统中。我们设置了基础的三个功能：慢洗、快洗、脱水，还增加有串口，进行命令下发控制洗衣机的功能。

首先，系统刚开机时处于待机状态，第一个指示灯 LED0（红灯）常亮，数码管没有工作，以此表示当前系统处于待机状态。然后用户可以选择后续的三个按键 KEY1、KEY2、KEY3，分别代表慢洗、快洗、脱水三个功能。在按下任意一个功能按键时，洗衣机就会进入工作状态，待机指示灯（红灯）就会熄灭，对应的功能指示灯就会点亮，紧接着数码管会进入倒计时状态，电机会进行转动。待倒计时结束，即代表功能完成，蜂鸣器会发出滴滴的提示音，电机停止转动，功能指示灯会熄灭，待机指示灯点亮。洗衣机在一个工作状态工作时，按下其他工作状态的按键，是不能终止该工作状态的。除此之外，还可以通过串口下发命令，来控制洗衣机的工作状态。

（2）需求分析

① 硬件部分：

主要分成三个部分：状态反馈模块、输入模块、通讯模块。

状态反馈模块包括：4 个 LED 灯、1 个八段数码管、1 个蜂鸣器、1 个电机。

输入模块包括：4 个按键。

通讯模块包括：CH340。

② 软件部分：

首先用户能够进行功能选择，需要设计出一个能够兼容多种功能并作出反馈的程序框架，并将各个模块的驱动封装成 API 来供其他模块使用，系统状态反馈不由功能程序完成，全部由系统框架完成，此外任务完成后的返回也有系统完成后续处理。

其次是三个功能模式（快洗、慢洗、脱水），根据模式的不同，电机的转速和方向也有所不同。比如，快洗所需的时间比慢洗所需的时间要短，且快洗的转速更快；快洗和慢洗都是电机来回转动，脱水是电机单向转动。

最后是通过串口命令驱动，这里利用串口调试助手，在上位机上下发命令，直接控制洗衣机功能。

（3）方案设计

① 硬件选型

a. 电机

备选电机有普通的直流电机、步进电机、带功放与解码的电机。此次做的是洗衣机系统，对电机的精度要求不高，所以排除步进电机，考虑到开发板在使用 PWM 输出时，因为功率不够无法直接驱动普通直流电机，所以最终选择带功放与解码的电机。该电机可以直接用板子连接，并且可以控制旋转方向，可以使用 PWM 波控制转速。

b. 数码管

数码管有共阴极和共阳极之分，这里因为只有共阴极的，所以选用共阴的数码管，另外考虑到引脚资源节约和程序开发的容易程度，选择使用一位的共阴极数码管，虽然只能显示一位数字，但用于模拟完全够用。

② 方案设计

首先是系统框架的设计，因为要使用模式选择，所以采用 switch(flag) 的程序框架，读取用户的输入，更改 flag，通过 switch 的匹配进入相应功能。按键输入使用扫描的方式，因为在待机状态下系统主要任务就是获取用户的输入，所以使用扫描的形式满足任务需求且容易开发，节省系统资源。

其次，各功能模块以及各硬件模块的驱动 API 封装，比如数码管显示数字、蜂鸣器鸣叫的时长与次数、电机的转速控制等。以此方便快速开发与调用，以及后续的修改与调试。

最后是串口模块，开发板只需要配置好 UART 和进行控制信息处理，双方传输的信息不大（每次 8 字节）。可以通过自定义功能匹配字符串，将串口传输来的字符串与系统的功能字符串匹配，如果匹配上就执行相应的功能；如果都不匹配上，则都不执行。

③ 引脚分配

通过 LPC213x 的引脚功能手册，根据需求分析和方案设计，做出如下引脚分配，如表 5 所示：

表 5　引脚分配表

模块名称	模块引脚(编号)	LPC213x 引脚	所用功能
LED	LED0	P0.17	GPIO 输出
	LED1	P0.18	GPIO 输出
	LED2	P0.19	GPIO 输出

	LED3	P0.20	GPIO 输出
KEY	KEY1	P0.16	GPIO 输入
	KEY2	P0.14	GPIO 输入
	KEY3	P0.15	GPIO 输入
DigitalTube	a	P0.27	GPIO 输出
	b	P0.25	GPIO 输出
	c	P0.30	GPIO 输出
	d	P0.28	GPIO 输出
	e	P0.26	GPIO 输出
	f	P0.29	GPIO 输出
	g	P0.31	GPIO 输出
BEEP	VCC	P0.23	GPIO 输出
Motor	AIN	P0.7	PWM
	BIN	P0.8	GPIO 输出
UART	RX	P0.0	UART0-TX
	TX	P0.1	UART0-RX

各模块的连接原理图如图 19 所示：

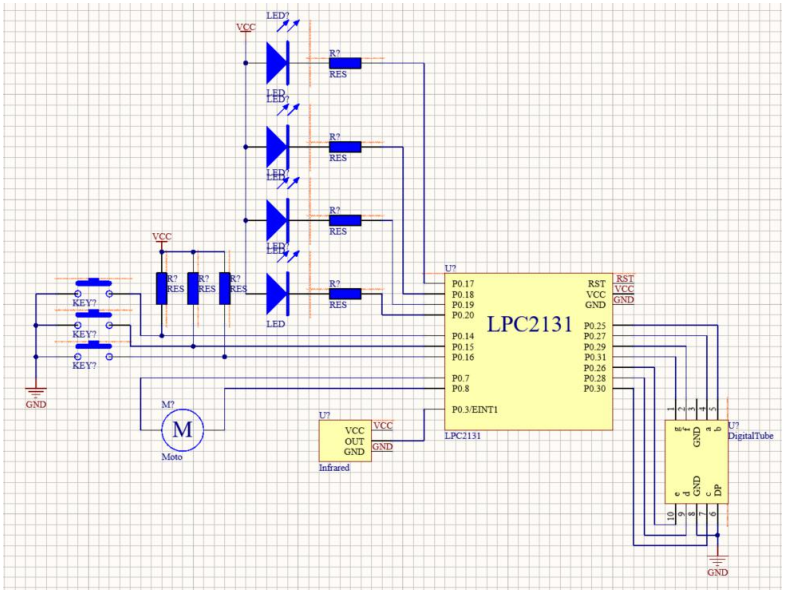

图 19　各模块连接原理图

系统结构图如图 20 所示：

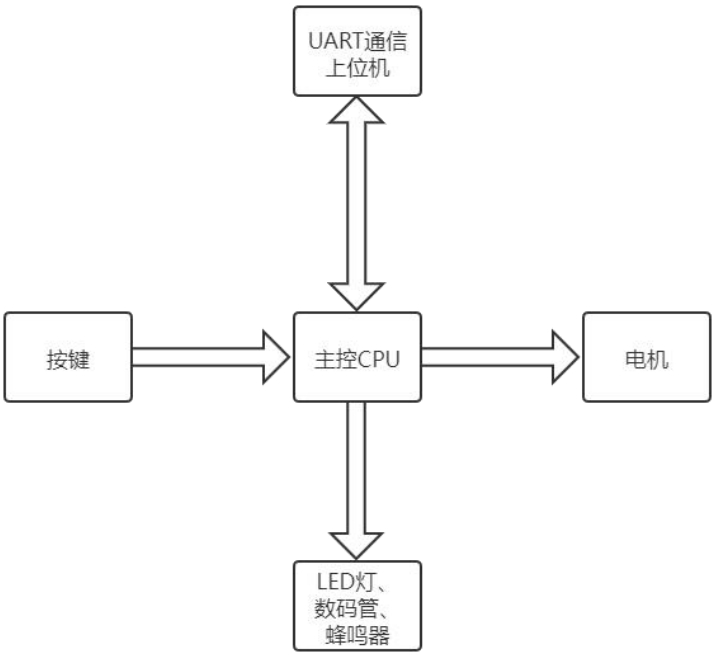

图 20　系统结构图

（4）功能实现

① 系统整体运行：接上电源，按下复位键（KEY0），进入待机状态。此时可以进行模式选择：如果按下 KEY1，就进入了慢洗模式；如果按下 KEY2，就进入了快洗模式；如果按下 KEY3，就进入了脱水模式。串口模块一直开启，如果从上位机发送“Model_1”，就进入了慢洗模式。

整个系统的流程图如图 21 所示：

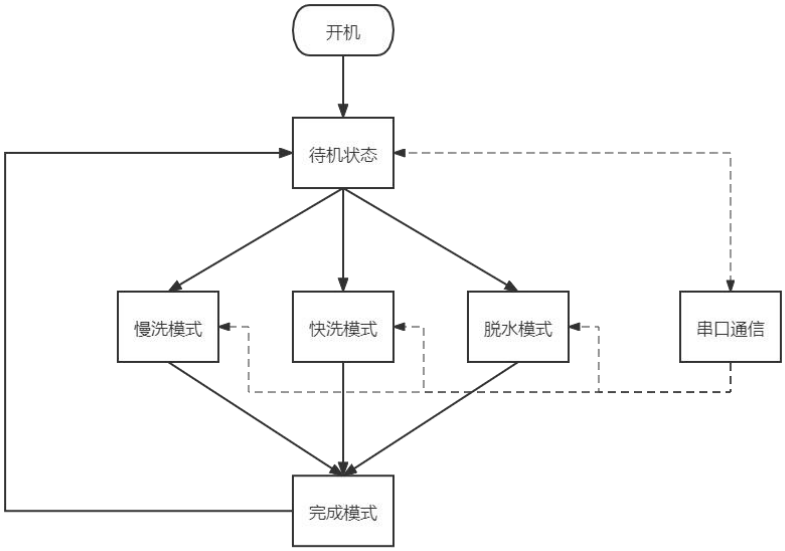

图 21　系统流程图

② 基础功能模块：主要是慢洗、快洗、脱水。先熄灭 LED0 待机指示灯，

然后点亮功能指示灯，配置 PWM 的占空比，控制电机的转速；同时数码管显示倒计时，并判断是否退出倒计时循环。在倒计时结束后，配置 PWM 的占空比为 0，停止电机，数码管显示 0，熄灭功能指示灯，回到待机状态。

如下为基础功能流程图：

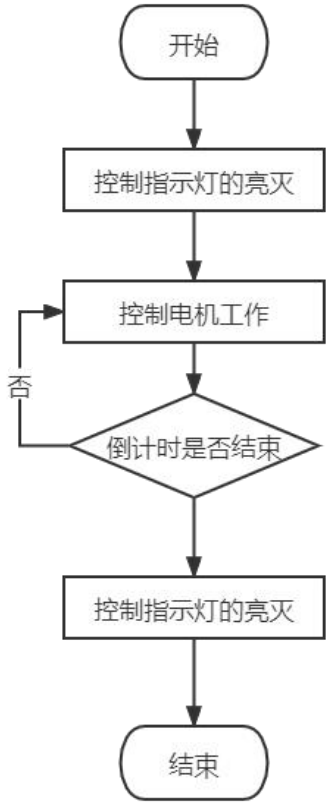

图 22　基础功能流程图

（5）系统测试

① 测试方案 1：在系统开始运行后，按下 KEY1 键，观测系统是否进入慢洗状态：红灯熄灭，绿灯点亮。数码管开始倒计时，电机开始来回转动。

测试结果与结论：在按下按键后，红灯熄灭，绿灯点亮，数码管开始倒计时，电机开始慢速来回转动。待倒计时完成后，蜂鸣器鸣叫，红灯点亮，绿灯熄灭，倒计时停止工作，系统进入待机状态。测试结果符合预期。

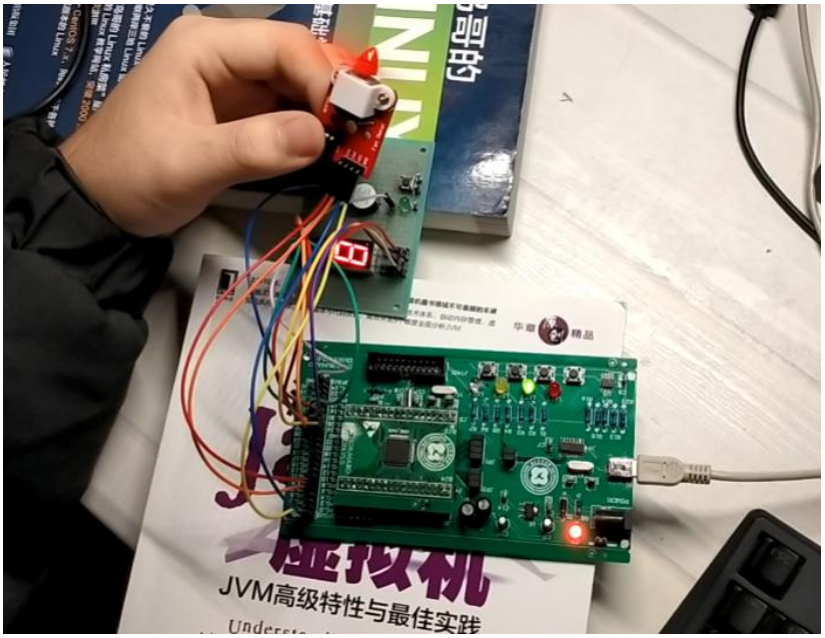

图 23　慢洗功能测试过程

② 测试方案 2：在系统开始运行后，按下 KEY2 键，观测系统是否进入快洗状态：红灯熄灭，黄灯点亮。数码管开始倒计时，电机来回转动。

测试结果与结论：在按下按键后，红灯熄灭，黄灯点亮，数码管开始倒计时，电机开始快速来回转动，且倒计时时间比慢洗要短。待倒计时完成后，蜂鸣器鸣叫，红灯点亮，黄灯熄灭，倒计时停止工作，系统进入待机状态。测试结果符合预期。

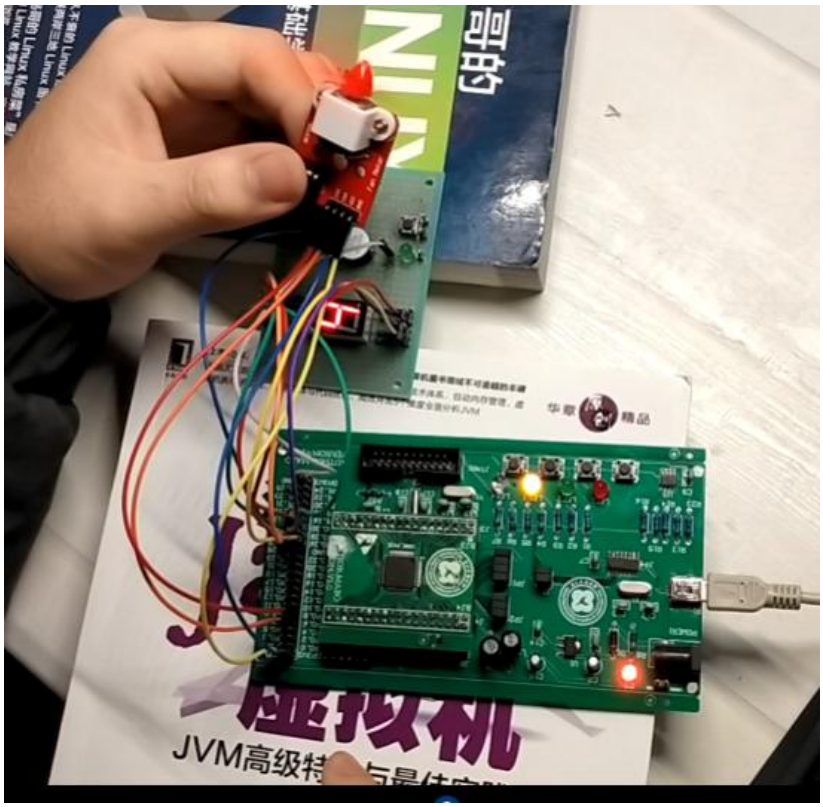

图 24　快洗功能测试过程

③ 测试方案 3：在系统开始运行后，按下 KEY3 键，观测系统是否进入脱水状态：红灯熄灭，蓝灯点亮。数码管开始倒计时，电机单向高速转动。

测试结果与结论：在按下按键后，红灯熄灭，蓝灯点亮，数码管开始倒计时，电机开始单向高速转动。待倒计时完成后，蜂鸣器鸣叫，红灯点亮，蓝灯熄灭，倒计时停止工作，系统进入待机状态。测试结果符合预期。

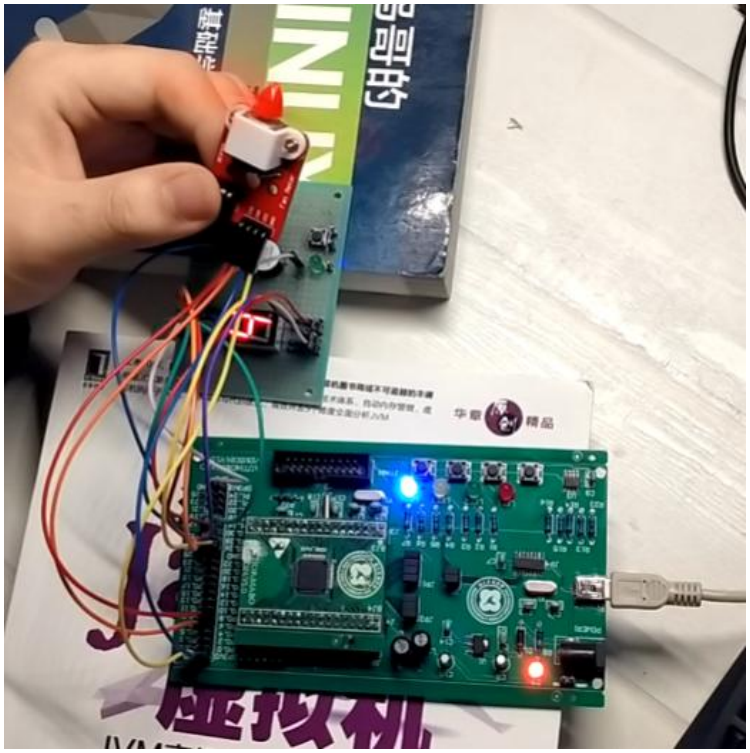

图 25　脱水功能测试过程

④ 测试方案 4：在系统开始运行后，在上位机的串口模块中发送命令，“Model_1”代表慢洗功能，“Model_2”代表快洗功能，“Model_3”代表脱水功能。

测试结果与结论：USB 插入后，上位机串口调试工具中应该可以检测到开发板的串口。然后向开发板中发送命令，当输入命令与功能匹配后，即可驱动开发板作出相应的动作，并且上位机中会反馈刚才调用的命令。经测试功能符合预期。

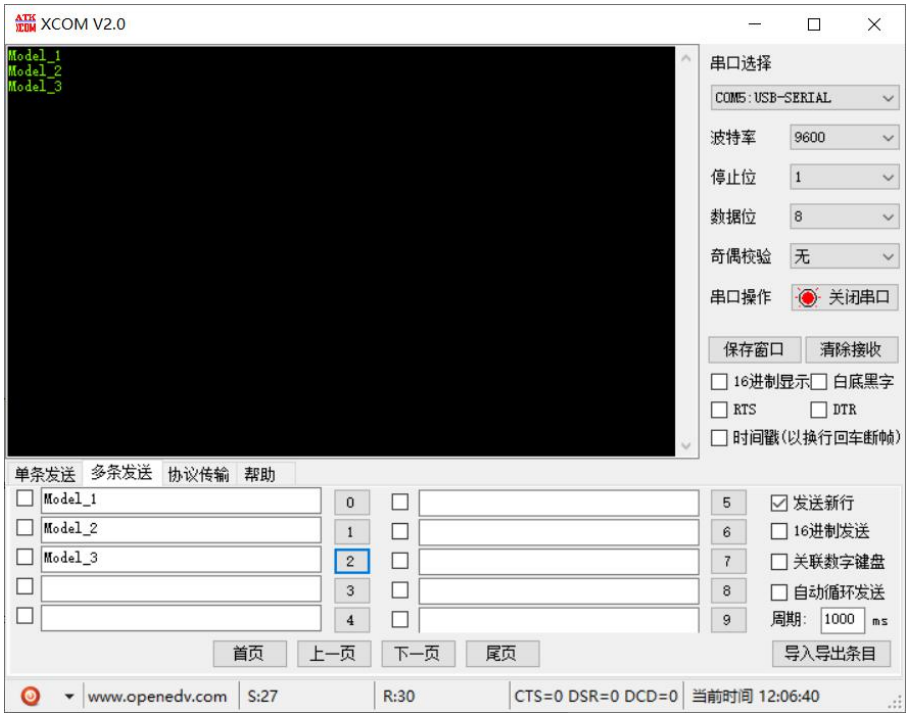

图 26　串口功能测试过程

（6）项目总结

问题：全局变量 countDownNum 用来初始化倒计时，在使用倒计时，会将其初始化成需要的值，比如说倒计时 9 秒。然后该全局变量在定时器中断服务程序中调用，每隔 1 秒进行减 1。但是出现了定时器中断服务程序无法修改的现象。

解决办法：需要加上 volatile 关键字。volatile 关键字是可以保证可见性的，在中断服务程序中修改了该变量，而编译器判断主函数里面没有修改了该变量，因此可能只执行一次从内存到某寄存器的读操作，然后每次只会从该寄存器中读取变量副本，使得中断程序的操作被短路。

三、总结与体会

1、实验完成情况

在综合实验中都利用到了 GPIO、定时器中断、串口通讯等技术。GPIO 用于输入输出，定时器中断用于倒计时模块，串口通讯用于上位机发送命令控制系统功能。

2、总结及体会

在本次实验中，学习了嵌入式开发的总体流程，对之前的原理有了更深入的了解和学习，巩固了先前学过的知识；同时也锻炼了工程实现能力。体验了系统级开发的完整流程，从确认题目，需求分析，器件选型，程序框架

的设计，方案的选择与敲定，最终得到一套具体的方案。然后对着芯片手册进行引脚功能分配和绘制原理图。按模块开发及调试，发现问题及时排查。最终将各功能模块组合到一起，联合调试，形成一个完整的系统。除此之外，还加强了软硬件协同开发的合作能力，和搭档在开发的过程中更是体会到了合作的重要性。

实验要求及评阅表

姓　名	周铭海	学　号	04181016

实验内容

硬件焊接与测试（第 15 周）

周一：讲解基本要求和实验内容，查阅学习资料、观看相关视频，学习相关基础知识。

周二：讲解焊接工具的使用方法和安全操作流程，练习焊接和解焊，熟练掌握焊接技术。

周三：讲解测试仪器、仪表的使用方法和测试步骤，JTAG 仿真器调试。

周四：根据板级电路的电路图等相关资料焊接主板和主要外设模块。

周五：利用相关仪器、仪表测试焊接电路板，开发板连接调试。

软件编程与调试（第 16 周）

周一： GPIO 输入与输出控制编码，定时器控制流水灯闪烁编码及调试。

周二：外部中断控制编码，UART 串行通信编码与调试。

周三：定时器与串行接口结合代码设计运行调试。

周四：综合项目设计与测试。

周五：所有编程代码综合调试并交付检查。

指导教师评语