



成 绩 评 定	
教 师 签 名	

四川大学电气信息学院 计算机应用设计（单片机） 实验报告

实验名称：实验一 中药熬煮温度控制系统

实验地点：高压楼 501 教室

年 级：2018

姓 名：许如玉

学 号：2018141442004

实验时间：2020 年 12 月 17 日

一、实验内容

1、设计一个中药熬煮温度控制系统，利用 AD 模块通过热电阻检测温度，每 20 秒采样一次温度值并将实时温度现实在数码管上。

2、温度控制要求，尽快升温到达设定的熬煮点温度(可通过键盘设置)，并保持该温度 15-20 分钟。加热的控制通过 PWM 输出控制固态继电器进行电炉加热功率的控制，加热持续时间显示在数码管上。

3、如果温度超过上限需要声光报警加热时间到也要声光提示，二者提示方式不同。利用发光二极管和蜂鸣器报警提示。

因为实际没有电炉，所以使用数码管、旋钮等模块进行模拟。实际的过程时间较长，模拟时为方便演示也进行了相应的缩短。

二、设计思路

1.计时模块：使用 PIT 模块进行硬件计时。

2.显示模块：使用数码管，左侧三个数码管进行加热时间的显示，显示时间和在 PIT 模块中进行记录；右侧三个数码管负责温度的显示，ADC 模块采样到数值根据一定数学关系，将采样值转换为温度值。

3.声光报警：使用蜂鸣器和 LED 灯。不同声音通过不同频率调整频率实现，不同的颜色借由 LED 达到，包括红、绿、蓝三种基础色和紫、黄、白三种合成颜色。

4.温度检测：本应使用温度传感器，但仅用单片机很难在短时间内提升太多温度，所以使用类似的旋钮进行温度的模拟。

5.加热：仅进行显示的模拟，通过数码管的数字显示和 LED 亮光来代表加热过程。加热到设定温度自动停止加热并声光显示、保持状态。

三、硬件系统构成

1.PIT 模块

2.LED 模块

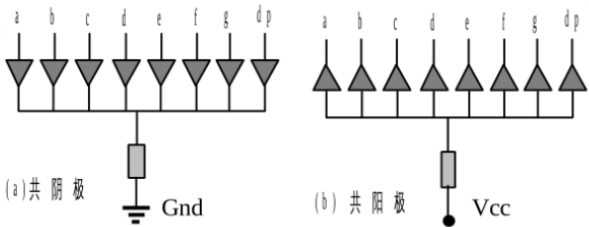
3.ADC 模块

4.按键



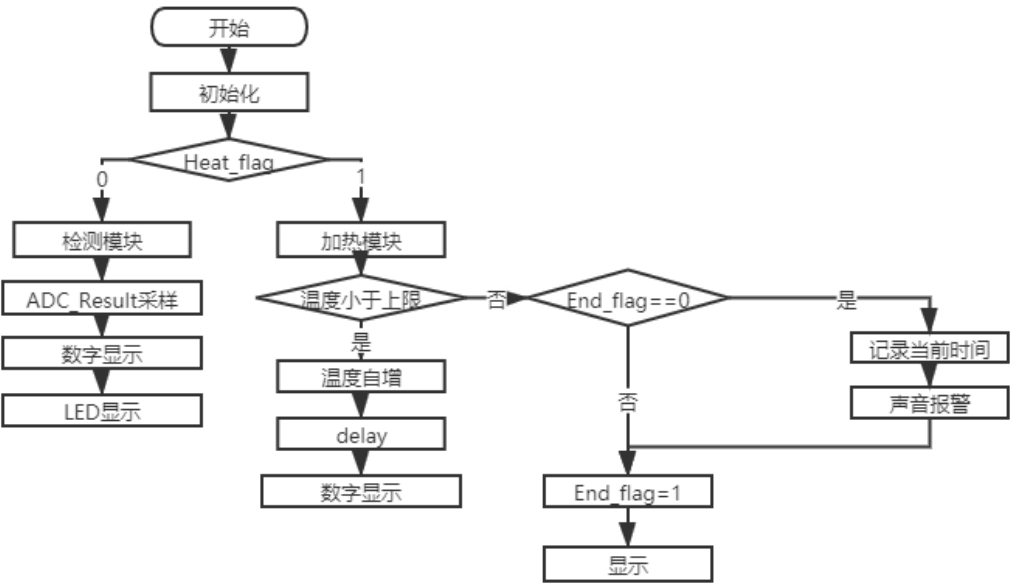
5.数码管

6.蜂鸣器



四、软件系统设计（包括程序流程图（加注释））

1. 加热、检测模块

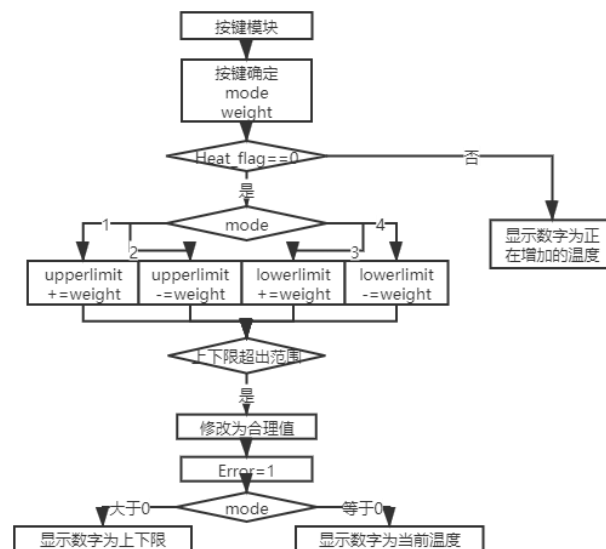


设置 Heat_flag 为模式旗帜，其值为 1 时代表加热模式，为 0 代表检测模式。

初始化后先进行 Heat_flag 的判断，进入对应模块。若在检测模块，则进行 ADC

模块采样，并进行数字显示和 LED 的显示（这里的判断在后面会详细介绍）。若在加热模块，则初始化 End_flag 为 0，先判断当前温度是否小于上限，如果是，则令温度自增，维持一小段时间，同时进行数字显示。如果温度大于等于预设上限，则判断 End_flag 是否为 0，如果是则进行声音报警，表示达到预设温度，报警结束后令 End_flag 为 1；如果为否，则维持 End_flag 为 1。但是两种结果都会进行 LED 的光显示和数码管的温度显示，这段维持的时间为七秒。End_flag 的意义在于，只有刚达到上限温度的时候才会进行声音的报警，不会在七秒内一直报警。

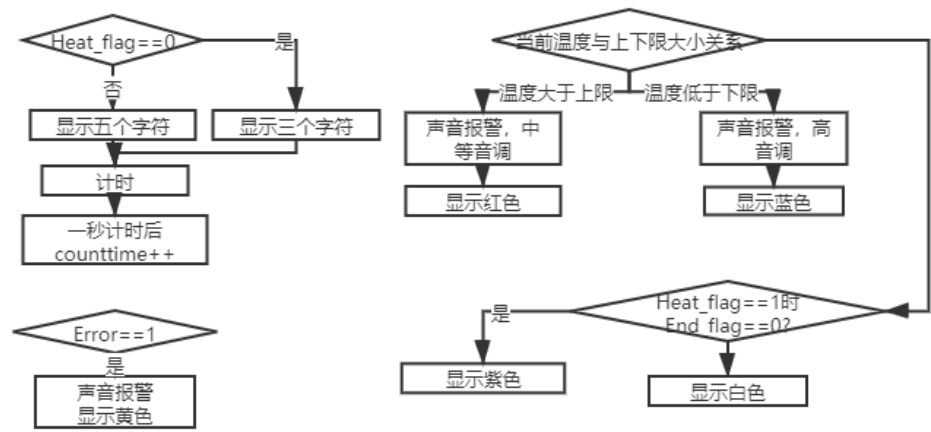
2. 上下限设置



上下限设置涉及到左右两个按键板块。都需要在 Heat_flag==0 时才可以正常执行。右侧板块负责 mode 的赋值，分别设置 mode 为 1、2、3、4，分别代表上限加、上限减、下限加、下限减，这部分的赋值在 PORTB\PORTE 模块进行。在主函数文件中令设一个用于计算上下限的函数，通过 mode 的值，对 upperlimit 和 lowerlimit 进行改变。左侧按键的前三个为权值，分别代表 1、5、10。每按一次，增或减该值。在 mode 改变的 porte 模块结尾会将权值赋值为 0，防止误增。如果上下限设置的不合理，会进行上下限报警（Error 赋值为 1），并自动设置为合理值。左侧的第四个按键按下后将 mode 赋值为 0，如果 mode 大于 0 时，数

码管显示会为当前正在是设置的上限或下限，如果 mode 等于 0，则数码管显示当前采样温度。

3. 显示 报警



如果 Heat_flag 为 0，则只显示温度，如果为 1，则也会显示加热时间，并且一秒计时后将计时变量 counttime 自增一。

报警系统为，如果温度大于上限，中等音调报警，显示红色，如果低于下限，则高音调报警，显示蓝色，如果处于上下限之间，则显示绿色。如果正在加热，则显示紫色，如果加热结束的维持时间，则显示白色。如果 Error 为 1，则声音报警，显示黄色，结束后 Error 重新设置为 0。

五、调试过程

1.时间调整：将要求的时间进行适当的缩小，采样时间为一秒，加热后维持时间为 7 秒。

2.加热速度：调节的加热的速度为速度较快，不会等待时间过长，同时也能够数码管上看见连续的温度变化。一开始没有设置中途退出加热的功能，后为更加仿真，加入功能。

3.上下限设置：考虑上下限的逻辑关系，如上限不可以低于下限，下限不能过低。

六、实验结果

按右侧键的第一个键，设置为上限增模式，此时数码管上显示上限，每按一次左侧第一个键，上限自增一，第二个则自增五，第三个自增十；右侧第二、三、四个键分别为上限减、下限加、下限减。如果设置的上限低于下限，则声音报警，LED 显示黄灯，自动调整当前设置的限值。如，下限 23，上限 26，如果按下右侧第二个键，数码管显示上限 26，再按一次左侧第二个，希望得到上限减 5 的效果，但是减去之后的值 21 会小于 23，故声音报警，LED 亮黄色光，较短时间后结束，并自动调整上限为 24，显示在数码管上。

按下左侧第四个键，则又显示采样温度。旋转旋钮，如果高于上限则 LED 亮红灯，并有声音报警；如果低于下限则 LED 亮蓝灯。

在显示当前温度的状态下，如果按下左侧第四个按键，则开始加热，LED 亮紫光，数码管左边显示加热时长，右侧显示当前温度。到达上限温度后响一声，LED 改亮白灯，数码管显示维持温度，此段维持时间为七秒。如果在加热过程中再次按下左侧第四个键，则停止加热，重新回到检测状态。

创新点：

- 1.上下限有范围规定，如果出错报警并自动修正
- 2.每一种状态都有 LED 不同颜色显示
- 3.加热可以中断、温度过低也会报警，更加模拟实际

七、实验分析（数据分析与讨论等）

实验结果符合预期，且三个创新点功能较为完备。总体运行良好。

各个固定的时间，如采样时间，基本符合预定，但是有时候会有一些小误差，但不影响总体功能。

八、实验心得

本次实验较为全面地总结应用了本课程所学内容，包括中断、数码管显

示等，通过本次实验我对之前所学有了一次大检测，察觉到了之前学习时一些不明白的知识点，并及时地查明。同时这次的实验也是对自己能力的一次综合考验，着其中对我而言最重要的就是不断精进、不断纠错并改正的过程。我在实验时遇见的一次次报错中，越来越了解查错、改错的方法。我感觉自己获益良多。



四川大学电气信息学院
计算机应用设计（单片机）
实验报告

实验名称： 实验二 牛奶包装系统

实验地点： 望江校区高压楼 501

年 级： 2018 级

姓 名： 汪雨甜

学 号： 2018141411218

实验时间： 2020 年 12 月 17 日

实验（2）牛奶包装系统

一、实验内容

1. 设计一个牛奶包装系统，对通过的牛奶盒进行计数。
2. 牛奶盒的计数值显示在两个数码管上。
3. 每次计满 24 盒，则通过串口输出一个数字量 0xFF 给外接的打包控制器，执行打包操作。
4. 打包机打包完成时发送给单片机 0x55，则单片机接收后给出声光提示。

二、设计思路

2.1 整体设计思路

我们的整体设计思路给予一个假设，即：牛奶包装系统的牛奶盒是均匀等间隔地通过计数器的。我们用 PIT 定时器模拟牛奶盒通过的间隔。

通过光照传感器和 pit 定时器实时对环境光照值进行采样，光照小于一定阈值代表有牛奶从传送带上经过，遮挡住光源，所以光照强度变低。此时记录下次数以及遮挡住光的时长，用于之后打包和判断牛奶是否正常。当计数值达到 24 时，计一个打包数并显示在数码管上，开始下一轮计数。当串口有”Pack”命令输入时，倒计时三秒钟，表示正在打包，随后 LED 白灯闪烁，蜂鸣器响起，程序结束。

2.2 分布设计思路

①利用光照传感器采集光照强度的变化模拟牛奶盒通过检测器的时刻。在 PIT 模块计时器的一个时刻光照强度低于某个设定值，等同于通过一个牛奶盒。

②每通过一个牛奶盒，数码管后三位将显示通过的牛奶盒的总数。当牛奶盒总数达到 24 时，包装数+1，显示在数码管的前三位。

③在打包前，通过串口输出一个数字量 “right” 给外接的打包控制器，表示执行打包操作。打包机打包完成时发送给单片机字符 “P”，则单片机接收后给出声光提示。

三、硬件系统构成

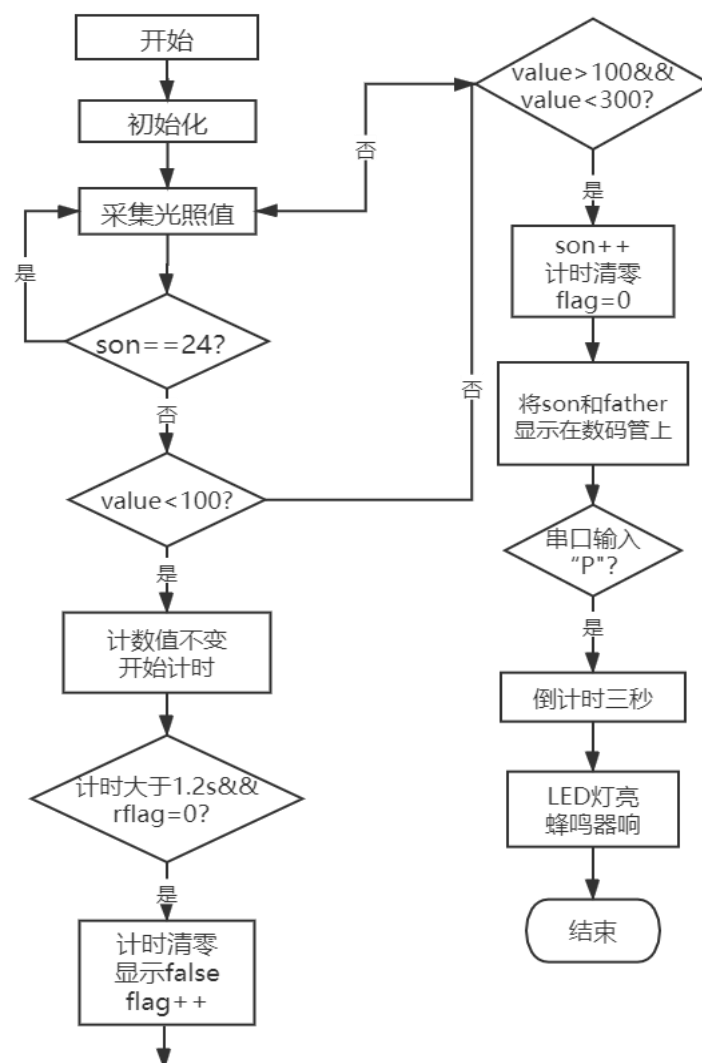
K20 芯片



K20 系列单片机有 144 引脚 LQFP、144 引脚 MAPBGA、80 引脚 LQFP 和 64 引脚 LQFP、32 引脚 QFN 等 12 种封装可供选择，片内集成了 SPI, I2C, UART, 高达 16 位 ADC, CMP, RTC, PIT, LPTMR, 16 位 FTM, PDB, DMA, 电容触摸控制器等多种外围设备。

上图为本次实验所使用的 80 引脚封装的单片机，具有 512KB 的 FLASH 空间和 128KB 的 SRAM 空间。实验中使用了 LED*4，数码管*6，定时器*1，计时器*1，光敏电阻*1，串口*1 等几个硬件模块。

四、 软件系统设计



五、 调试过程

1. 无法对牛奶的正确与否做出准确的判断

在改动过程中主要出现了下列三种错误：

- ① 运行结果与所期望结果相反，即在较短遮挡时间（正常）时显示 false，在较长遮挡时间（异常）时显示 right。
- ② 在出现 right 之后一直出现 false，陷入死循环
- ③ 串口没有输出

针对以上问题，我做出了如下的解决方案：

- ① 修改参数，发现是由于 pit 设置中断时间过短导致
- ② 检查程序逻辑，发现当程序运行入异常的判断后无法出来，因此设置一个标志位解决了这个问题

```
if((seconde>=40)&&(rfflag==0))
{
    seconde=0;
    UART0_SendString("\r\nfalse\r\n");
    rfflag++;
}

if(photo_value<=300&&photo_value>=100)//180
{
    son++;
    seconde=0;
    rfflag=0;
}
```

通过如上修改，得到了正确逻辑的结果。

- ③重启串口软件，更改代码会导致串口发生异常。

（2）倒计时参数出现问题

错误原因：在 main.c 和 kinetis_sysinit.c 中重复定义了数码管显示数组 num[11]

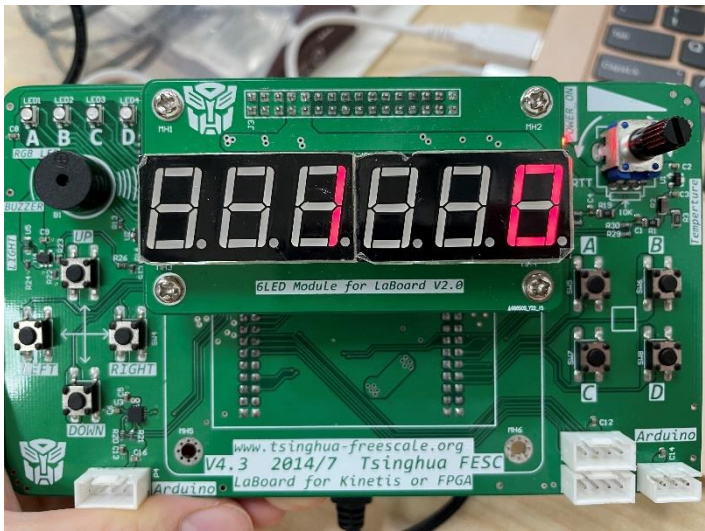
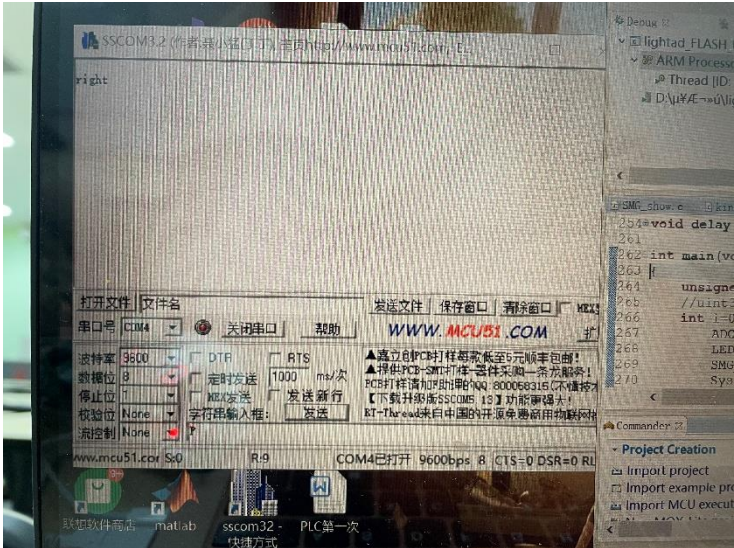
经过如下修改：

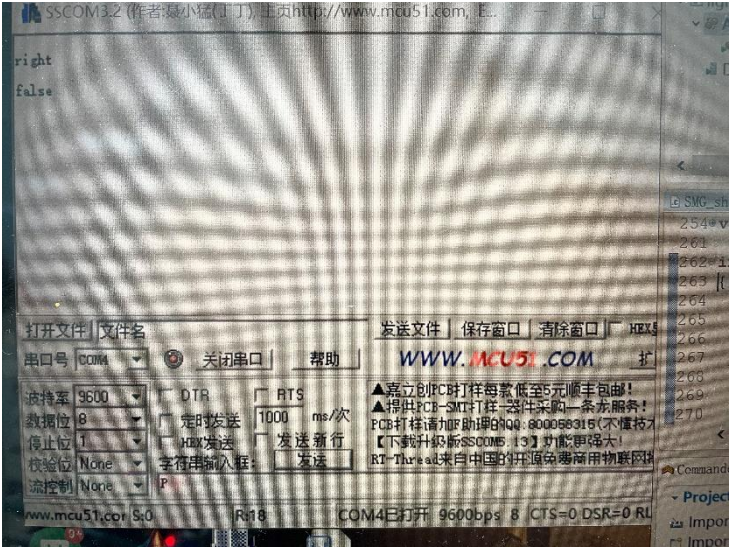

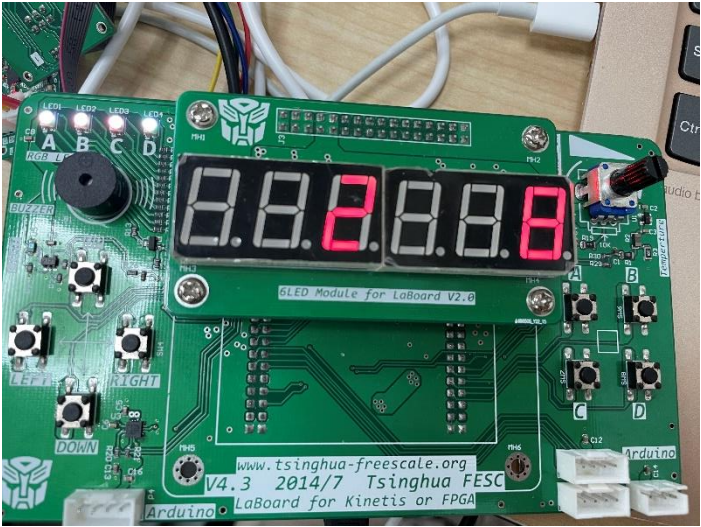
```
42 unsigned char daojishi[10]={0xA0,0xBE,0x62,0x2A,0x3C,0x29,0x21,0xBA,0x20,0x28};
```

成功解决了报错，程序得以运行。

六、实验结果

实验结果展示如下：

项目	图片
正常时满 24 串口显示” right”	
异常时满 24 串口显示” false”	

	
正常时满 24 发给串口信号”P”	
小灯亮，蜂鸣器响	

七、实验分析

7.1 创新点：

①在数码管上显示已经打包的个数

②打包时有倒数 3 秒计时

③可以判断牛奶通过打包系统时是否正常，正常时上位机显示 right，错误时上位机显示 false

7.2 函数分析

(1) 时间处理函数

```
92=void time_pro(void)
93 {
94     if(seconde==100) //将毫秒设置为秒
95     {
96         seconde=0;
97         second++;
98
99         if(second==60) //秒钟设为60进制
100        {
101            second=0;
102            minite++;
103
104            if(minite==60) //分钟设为60进制
105            {
106                minite=0;
107            }
108        }
109    }
110 }
```

原先设置 pit 定时器为 10ms 产生一次中断，而在中断子程序中遮挡时间的计时由 seconde 来控制，当 seconde 自增到 100 时，second 加 1 (1000ms=1s)，当 second 增到 60 时，minite 加 1 (60s=1m)，用这种方式完成了时间的转换。

(2) Pit 中断子程序

```

196 void pit_channel0_ISR(void)
197 {
198     //seconde++;
199     time_pro();
200     photo_value=ADC0_TR_DATA(); //光照值
201     if(son==5)
202     {
203         son=0; //计数
204         father++; //打包
205         //UART0_SendChar('-');
206         UART0_SendString("\r\nright\r\n");
207         //UART0_SendChar('\n');
208         seconde=0;
209     }
210     if(photo_value<100)
211     {
212         son=son+1-1;
213         seconde++;
214
215         if((seconde>=40) && (rfflag==0))
216         {
217             seconde=0;
218             UART0_SendString("\r\nfalse\r\n");
219             rfflag++;
220         }
221     }
222
223     if(photo_value<=300 && photo_value>=100) //180
224     {
225         son++;
226         seconde=0;
227         rfflag=0;
228     }
229
230
231
232
233     ADC_ledshow();
234     SMG_show_data(show_data);
235
236     /*Clear the flag of channel0,PIT*/
237     PIT_TFLG0|=PIT_TFLG_TIF_MASK;
238
239 }
240
241 /**
242 **=====
243 **  Reset handler
244 **=====
245 */

```

代码详情详见上述流程图。

八、实验心得

通过本次实验，我更加熟悉了 PIT 计时器的使用，加深了对数码管、LED 灯的显示的印象和理解。从设计函数再到编写每一个函数，到尝试运行结果，根据运行结果推测问题出现在哪里，从而对代码进行调整，在逐步增加需要的函数之后，对主要函数的编写也遇到了大量的问题。在格式上报的错已经越来越少，更多的是逻辑上的错，这说明我仍需要对代码的设计进行细化，因为 CW 无法对代码进行设断点排查错误的 debug，所以在调试程序的过程中一直在根据结果推断程序哪里出了问题，对机器的运行逻辑也了解的更加深刻，收获颇丰。



成绩 评定	
教师 签名	

四川大学电气信息学院 计算机应用设计（单片机） 实验报告

实验名称：_____ 实验三 智能循迹小车

实验地点：_____ 高压实验楼

年 级：_____ 2018 级

姓 名：_____ 朱琪霖

学 号：_____ 2018141241025

实验时间： 2020 年 12 月 17 日

一、 实验内容

实验要求:

- 1、设计一个智能循迹小车,通过安装在车前的红外对管进行跑道的检测,并跟踪跑道行进;
- 2、小车四轮使用直流减速电机进行驱动,用 PWM 波控制小车的左右轮电机进行小车速度与方向的控制
- 3、用两个发光二极管作为左右的转向灯,在小车转弯时点亮该侧的转弯灯;
- 4、其他创新功能。

模拟实验:

- 1、用 SW1-4 控制行进方向、用 SW5-8 控制左右转弯幅度 (两个档位);
- 2、用旋钮控制车轮速度;
- 3、用 LED 作为左右转向的转向灯,在小车转弯时点亮该侧的转弯灯;
- 4、其他创新功能:

①转弯幅度:左右转弯分别存在两个档位:小转弯和大转弯,两者区别体现在左右两轮的速度差值;

②速度显示:六个八段数码管三个为一组分别显示小车左右两轮的速度,速度由旋钮进行控制。当速度减小到零时进入停车挡;

③停止与倒车:停止时:数码管上显示两轮速度均为零,此时无论旋钮如何旋均不会有任何速度显示;倒车:当选择倒车档时,速度依然由旋钮控制,数码管上显示出的两轮速度为负值,区别于前进档位。

二、设计思路

本实验要完成智能循迹小车,需要实现频率采集与 A/D 转换、速度计算、LED 显示转弯等基本功能。从设计上可将其分为 PIT、LED、A/D、数码管初始化、LED 显示、速度计算、显示预备几个模块,设计思路框图如下图 3-1 所示:



图 3-1 设计思路框图

本实验以 MK20DN512 单片机为核心，在单片机内部完成数据的存储及处理功能，通过数字信号的转换及输入，再将数据存入存储芯片，在单片机进行数据处理后在对需要显示的数字信号进行译码显示在数码管显示器上。本实验完成了采集功能、存储功能、数据处理功能、测试数据显示功能，达到了设计的基本要求。

三、硬件系统构成

K20 Play board

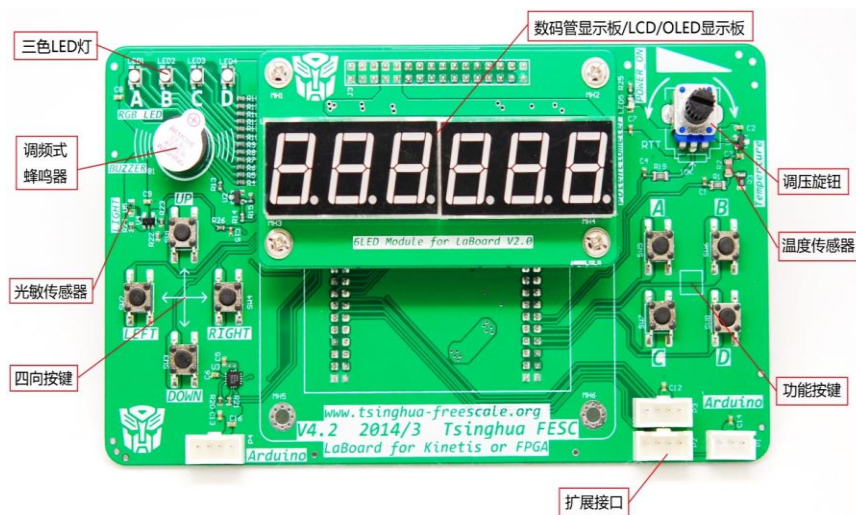


图 3-2 K20 Play board

1.LED 数码管显示

在单片机系统中，发光二极管（LED）常常作为重要的显示手段。由于 LED 显示器主要用于显示各种数字符号，故又称之为 LED 数码管，每个显示器还有

一个圆点型发光二极管，由于显示小数点。

2.A/D 转换

AD 模块具有初始化、采样、中值滤波、均值滤波等操作。按照构件的思想，可将它们封装成独立的功能函数。AD 构件包括头文件 `adc.h` 和 `adc.c` 文件。AD 构件头文件中主要包括相关宏定义、AD 的功能函数原型说明等内容。AD 构件程序文件的内容是给出 AD 各功能函数的实现过程。

3.按键

按键部分实现的主要原理是单片机读取与按键相连接的 I/O 口状态，来判定按键是否按下，达到系统参数设置的目的。键盘在单片机应用系统中的作用是实现数据输入。命令输入，是人工干预的主要手段。

四、软件系统设计（包括程序流程图（加注释））

主程序：首先是 PIT、LED、A/D、数码管各个模块初始化，然后对频率进行采样，设置通过旋钮的转动来控制小车左右两轮的速度，此时若速度直接减为零，则直接进入 `runmode=3`（即停车档位），否则则将设置速度代入速度计算模块，利用左右两轮的速度确定下 `show` 这个数组里的数值，最后进入到 LED 对转向的显示。

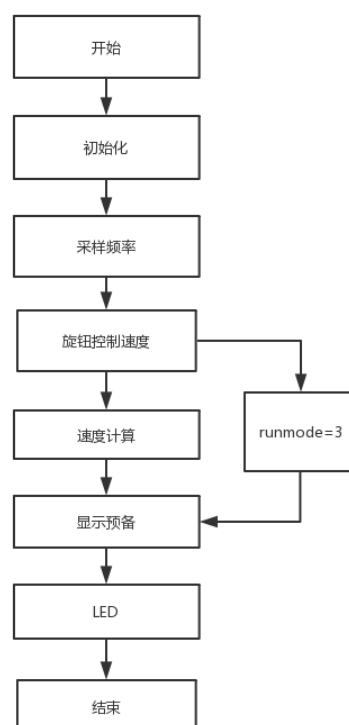


图 3-3 主程序流程图

runmode=1: 前进挡: 四个小灯全亮绿色

runmode=2: 转弯挡: turnmode=1: 第 2 个灯亮红色: $\text{leftspeed}=\text{speed}*0.6$,

$\text{rightspeed}=\text{speed}*1.4$

turnmode=2: 第 1 个灯亮红色: $\text{leftspeed}=\text{speed}*0.3$,

$\text{rightspeed}=\text{speed}*1.7$;

turnmode=3: 第 3 个灯亮红色: $\text{leftspeed}=\text{speed}*1.4$,

$\text{rightspeed}=\text{speed}*0.6$;

turnmode=4: 第 4 个灯亮红色: $\text{leftspeed}=\text{speed}*1.7$,

$\text{rightspeed}=\text{speed}*0.3$;

runmode=3: 停止挡: 四个小灯全亮红色

runmode=4: 倒车档: 四个小灯全亮白色

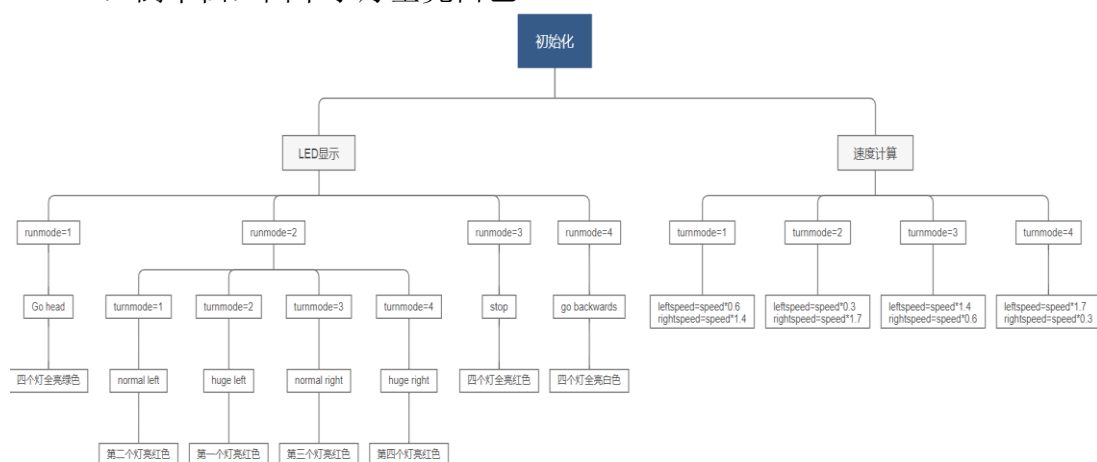


图 3-4 子程序流程图

五、调试过程

1. 硬件调试:

首先, 我们应该排除元器件失效问题。造成这类错误的原因有两个: 一是元器件买来时就已坏了; 另一个是由于焊接错误, 造成器件损坏。

其次, 排除电源故障问题。在通电前, 一定要检查电源电压的幅值和极性, 否则很容易造成集成块损坏。若有高压, 将会损坏单片机等, 有时会使应用系统中的集成块发热损坏。

本次实验的硬件调试顺序为先显示器后键盘。在显示器调试通过后，键盘调试就比较简单，完全可以借助于显示器，利用程序进行调试。利用开发装置对程序进行设置断点，通过断点可以检查程序在断点前后的键值变化，这样可知键盘工作是否正常。

2. 软件调试:

在确认硬件设备无误后，我们就可以开始利用 Codewarrior 对该实验进行调试：

代码如下：

```
* * main implementation: use this 'C' sample to create your own application
#include "derivative.h" /* include peripheral declarations */

#include "INKEY.h"

int turnmode=0;
int runmode=1;
int ADC_Result=0;
int AD_flag=1;
int speed=0;

int leftspeed=0;
int rightspeed=0;

extern int number;
extern int show[6]={0,0,0,0,0,0};
extern unsigned long int led[6];
extern unsigned char num[11];

void PIT_init(unsigned int number_ms)
{
    /*Turn on PIT clocks*/
    SIM_SCGC6|=SIM_SCGC6_PIT_MASK;

    /*Enable PIT Module*/
    PIT_MCR&=~(PIT_MCR_MDIS_MASK);

    /*Setup the channel0 of PIT*/
    PIT_LDVAL0=20000*number_ms;

    /*Enable the IRQ of channel0,PIT*/
    PIT_TCTRL0|=PIT_TCTRL_TIE_MASK;

    /*Running channel0,PIT*/
    PIT_TCTRL0|=PIT_TCTRL_TEN_MASK;
}
```

```

void LED_4_Init(void)
{
    SIM_SCGC5 |= 0X800;    // 使能PORTC时钟
    PORTC_PCR0 = 0x100;    // LED1红灯对应引脚设置为GPIO
    PORTC_PCR1 = 0x100;    // LED1绿灯对应引脚设置为GPIO
    PORTC_PCR2 = 0x100;    // LED1蓝灯对应引脚设置为GPIO
    PORTC_PCR3 = 0x100;    // LED2红灯对应引脚设置为GPIO
    PORTC_PCR4 = 0x100;    // LED2绿灯对应引脚设置为GPIO
    PORTC_PCR5 = 0x100;    // LED2蓝灯对应引脚设置为GPIO
    PORTC_PCR6 = 0x100;    // LED3红灯对应引脚设置为GPIO
    PORTC_PCR7 = 0x100;    // LED3绿灯对应引脚设置为GPIO
    PORTC_PCR8 = 0x100;    // LED3蓝灯对应引脚设置为GPIO
    PORTC_PCR9 = 0x100;    // LED4红灯对应引脚设置为GPIO
    PORTC_PCR10 = 0x100;   // LED4绿灯对应引脚设置为GPIO
    PORTC_PCR11 = 0x100;   // LED4蓝灯对应引脚设置为GPIO

    GPIOC_PDDR |= 0x0FFF;  // 对应端口设置为输出
    GPIOC_PDOR |= 0x0FFF;  // 四个小灯全灭
}

void LED_Dispatch_Init(void)
{
    SIM_SCGC5 |= (0x1000 | 0x0200);
    /*portA set to GPIO*/
    PORTA_PCR12 = 0x100;
    PORTA_PCR13 = 0x100;
    PORTA_PCR14 = 0x100;
    PORTA_PCR15 = 0x100;
    PORTA_PCR16 = 0x100;
    PORTA_PCR17 = 0x100;

    /*portD set to GPIO*/
    PORTD_PCR0 = 0x0100;
    PORTD_PCR1 = 0x0100;
    PORTD_PCR2 = 0x0100;
    PORTD_PCR3 = 0x0100;
    PORTD_PCR4 = 0x0100;
    PORTD_PCR5 = 0x0100;
    PORTD_PCR6 = 0x0100;
    PORTD_PCR7 = 0x0100;

    /*LED bit pin set to out*/
    GPIOD_PDDR |= 0x00FF;
    GPIOA_PDDR |= 0x03F000;
    /*LED control pin set to high ,LEDs are closed*/
    GPIOA_PDOR |= 0x03F000;

    /*LED bit pin set to high ,LED bit are all off, change GPIOD_PDOR register can change the number it shows*/
    GPIOD_PDOR = 0xFF;
}

void ADC_Init (void) {
    /*enable ADC0 clock*/
    SIM_SCGC3 |= SIM_SCGC3_ADC1_MASK;
    SIM_SCGC5 |= SIM_SCGC5_PORTE_MASK;
    /* Set pin0 of PORTE as analog function */
    PORTE_PCR0 = 0x0000;
    //PORTE_PCR0=PORT_PCR_MUX(0X0);
    //long sample time single-end 12bit conversion
    ADC1_CFG1 = 0X00000014;
    // ADC1_CFG1=ADC_CFG1_ADLSMP_MASK+ADC_CFG1_MODE(1);
    //ADxxxa channel select
    ADC1_CFG2 = 0X00000000;
    //default voltage reference Vrefh and Vrefl,software trigger
    ADC1_SC2 = 0X00000000;
    //continuous conversions
    ADC1_SC3 = 0X00000008;
    //ADC1_SC3=ADC_SC3_ADC0_MASK;
    //interrupt disable and select ADC0_SE4a channel as input
    ADC1_SC1A = 0X00000004;
    //ADC1_SC1A=ADC_SC1_ADCH(4);
}

```

```

void LED_Light(void)
{
    GPIOC_PDOR|=0x0FFF;    //四个小灯全灭
    if(runmode==1)
    {
        GPIOC_PDOR&=~(0x02|0x10|0x80|0x400);    //go ahead
    }
    else if(runmode==2)
    {
        if(turnmode==1)
            GPIOC_PDOR&=~(0x02|0x08|0x80|0x400);    //normal left
        else if(turnmode==2)
            GPIOC_PDOR&=~(0x01|0x10|0x80|0x400);    //huge left
        else if(turnmode==3)
            GPIOC_PDOR&=~(0x02|0x10|0x40|0x400);    //normal right
        else if(turnmode==4)
            GPIOC_PDOR&=~(0x02|0x10|0x80|0x200);    //huge right
        else
            GPIOC_PDOR&=~(0x02|0x10|0x80|0x400);
    }
    else if(runmode==3)
    {
        GPIOC_PDOR&=~(0x01|0x08|0x40|0x200);    //stop
    }
    else if(runmode==4)
    {
        GPIOC_PDOR&=~(0x07|0x38|0x1C0|0xE00);    //go backwards
    }
}

```

```

void SpeedCal(void)
{
    if(runmode==2)
    {
        if(turnmode==1)
        {
            leftspeed=speed*0.6;
            rightspeed=speed*1.4;
        }
        else if(turnmode==2)
        {
            leftspeed=speed*0.3;
            rightspeed=speed*1.7;
        }
        else if(turnmode==3)
        {
            leftspeed=speed*1.4;
            rightspeed=speed*0.6;
        }
        else if(turnmode==4)
        {
            leftspeed=speed*1.7;
            rightspeed=speed*0.3;
        }
        else
        {
            leftspeed=speed;
            rightspeed=speed;
        }
    }
    else if(runmode==1)
    {
        leftspeed=speed;
        rightspeed=speed;
    }
    else if(runmode==3)
    {
        speed=0;
        leftspeed=speed;
        rightspeed=speed;
    }
    else if(runmode==4)
    {
        leftspeed=speed/10;
        rightspeed=speed/10;
    }
}

```

```

void Num_Show(void)
{
    unsigned int i=0;
    for(i=0;i<3;i++)
    {
        show[i+3]=leftspeed%10;
        leftspeed/=10;
    }
    for(i=0;i<3;i++)
    {
        show[i]=rightspeed%10;
        rightspeed/=10;
    }
    if(runmode==4)
    {
        show[5]=10;           //负号
        show[2]=10;
    }
}

int main(void)
{
    PIT_init(0x01);           //1ms interval interrupt
    NVICICPR2=1<<(68%32);
    NVICISR2=1<<(68%32);
    LED_4_Init();
    LED_Disg_Init();
    KEY_Init();
    EN_Init();
    ADC_Init();

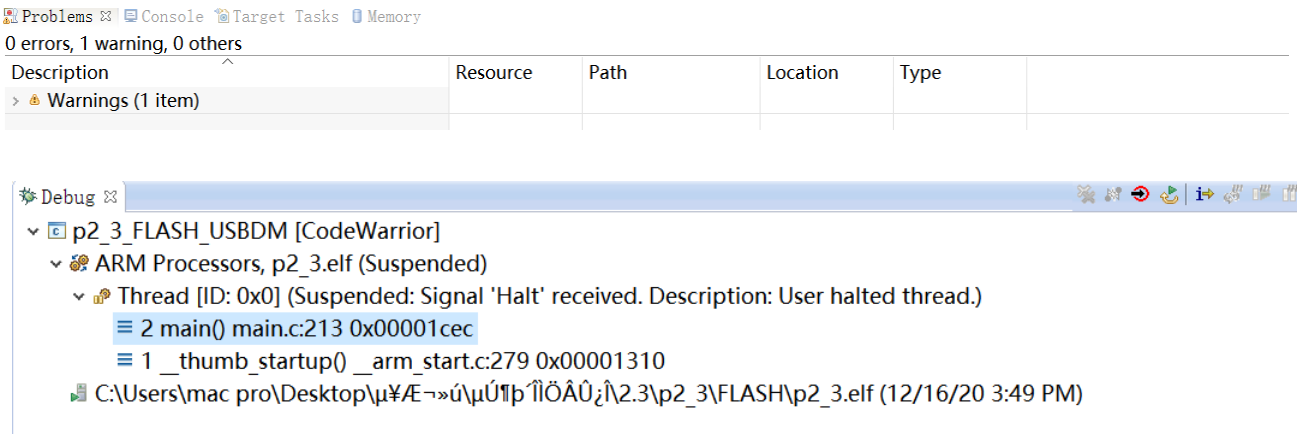
    for(;;)
    {
        if(AD_flag)
        {
            AD_flag=0;
            while( (ADC1_SC1A&ADC_SC1_COCO_MASK) !=ADC_SC1_COCO_MASK);

            ADC_Result=ADC1_RA;

            speed=(ADC_Result-120)/10;
            if(speed==0)
                runmode=3;
            //clear flag
            ADC1_SC1A&=~ADC_SC1_COCO_MASK;
            SpeedCal();
            Num_Show();
            LED_Light();
        }
    }
}

```


以下结果显示代码无误，调试成功：



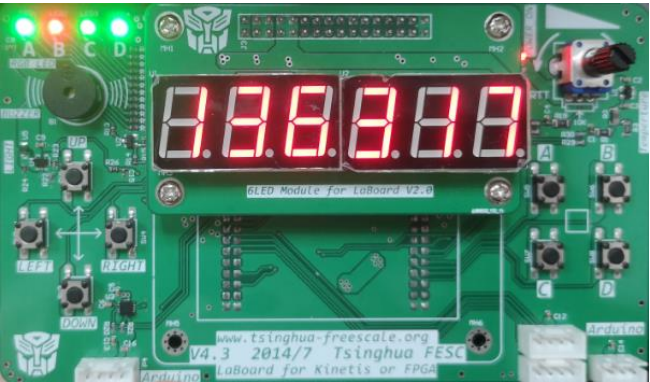
六、实验结果

1. 当按下 SW1 时，进入前进挡，四个小灯全亮绿色，数码管显示左右两轮的速度，可通过旋钮进行调节速度：

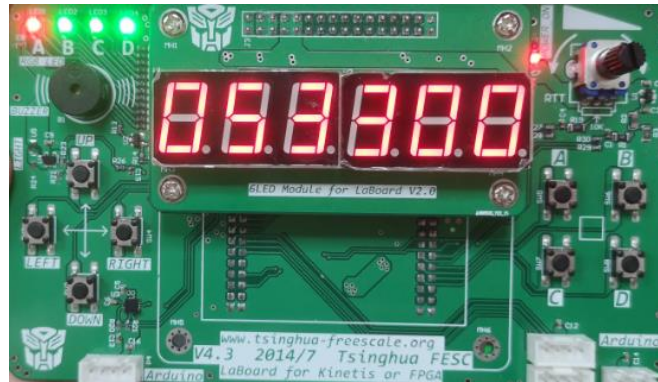


2. 当按下 SW2 时，进入转弯挡：

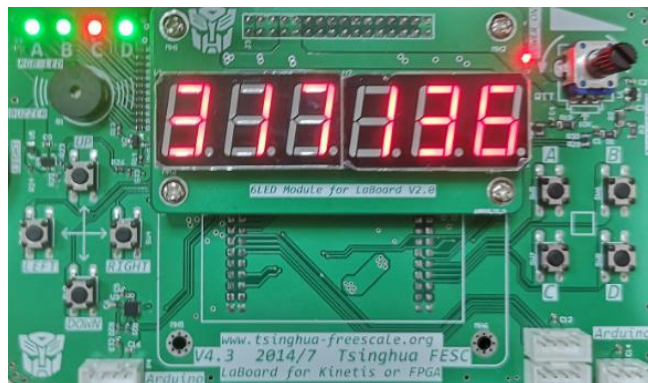
①当按下 A 时，为向左转小弯，第二个灯亮红色：



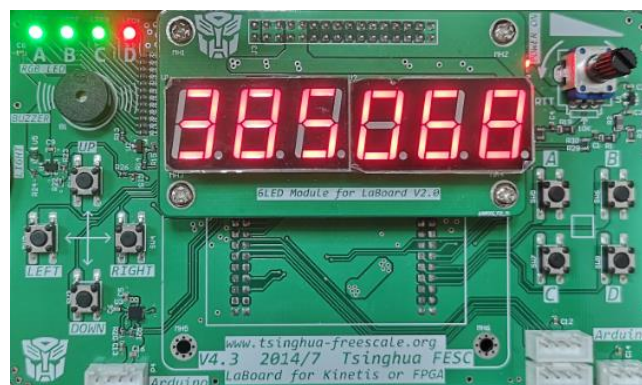
②当按下 B 时，为向左转大弯，第一个灯亮红色：



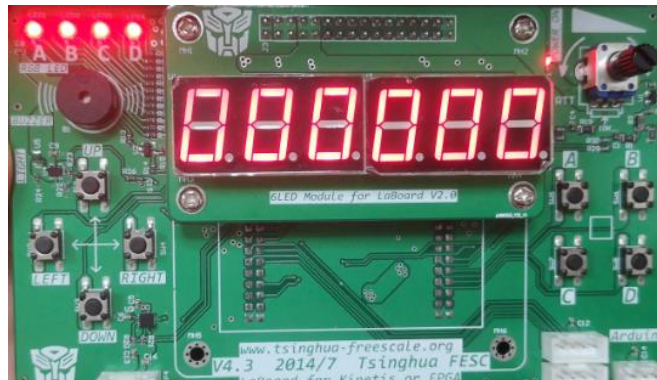
③当按下 C 时，为向右转小弯，第三个灯亮红色：



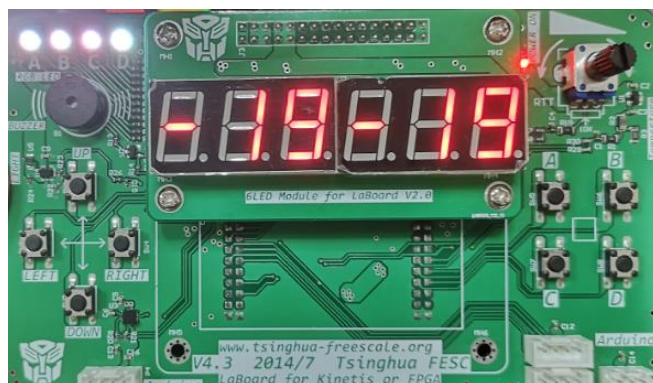
④当按下 D 时，为向右转大弯，第四个灯亮红色：



3. 当按下 SW3 时, 进入停车挡, 四个小灯全亮红色, 数码管显示两轮速度为零:



4. 当按下 SW4 时, 进入倒车档, 四个小灯全亮白色, 数码管显示数值为负, 区别于前进挡:



七、实验分析

```
void pit_channel0_ISR(void)
{
    GPIOA_PDOR|=0xFFFFF;
    GPIOB_PDOR=num[show[number]];
    GPIOA_PDOR&=~(led[number]);

    number++;
    flag_100ms++;

    if (number==6)
        number=0;

    if (flag_100ms==500)
    {
        flag_100ms=0;
        AD_flag=1;
    }
    /*Clear the flag of channel0,PIT*/
    PIT_TFLG0|=PIT_TFLG_TIF_MASK;
}
```

```
int main(void)
{
    PIT_init(0x01); //1ms interval interrupt
    NVICICPR2=1<<(60*32);
    NVICISR2=1<<(60*32);
    LED_4_Init();
    LED_Displ_Init();
    KEY_Init();
    EN_Init();
    ADC_Init();

    for(;;)
    {
        if (AD_flag)
        {
            AD_flag=0;
            while((ADC1_SCI&ADC_SCI_COCO_MASK)!=ADC_SCI_COCO_MASK);

            ADC_Result=ADC1_RA;

            speed=(ADC_Result-120)/10;
            if (speed==0)
                runmode=3;
            //clear flag
            ADC1_SCI&=~ADC_SCI_COCO_MASK;
            SpeedCal();
            Num_Show();
            LED_Light();
        }
    }
}
```

在主函数的主循环中, AD_flag 为采样频率, 先将其赋值为 0, 旋钮采样值赋

值给 ADC_Result，由此来控制速度，然后通过速度计算模块、显示预备模块以及 LED 显示模块实现功能。在 PIT 的中断模块中，设置采样周期为 500，即当 flag 从 1 加到 500 时，才重新对 AD_flag 进行赋值，以实现采样频率不同。

八、实验心得

本实验是关于智能循迹小车的设计，通过按键实现对各个模块的控制，用 LED 对小车转向灯进行模拟，利用旋钮控制速度，并且根据数据处理的结果，在数码管上显示。

关于本次实验，由于经验的不足和所掌握知识的限制，在实验中犯下了一些低级错误，通过翻查例程和询问同学方得解惑。从这次实验中，让我更深刻的体会到理论联系实际的重要性，只有多听多看多实践，才能更好的运用所学知识。这对于以后的学习工作都同样的重要，让我更好的意识到：实践方能出真知。



成绩 评定	
教师 签名	

四川大学电气信息学院 计算机应用设计（单片机） 实验报告

实验名称：_____实验四 智能洗衣机_____

实验地点：_____望江校区高压楼 501_____

年 级：_____2018 级_____

姓 名：_____赖梦婷_____

学 号：_____2018141442020_____

实验时间：_____2020 年 12 月 17 日_____

Lab Project4 智能洗衣机

一、实验内容

1. 通过 PWM 控制直流电机的转速，表征三种不同的洗衣方式：弱洗、强洗、漂洗；实现最长 10 分钟定时洗衣；
2. 用三个独立按键设置待洗衣物的不同洗涤方式：丝质衣服：漂洗 3 分钟；棉质衣服：弱洗 2 分钟，强洗 5 分钟，漂洗 3 分钟； 化纤衣服：强洗 4 分钟，漂洗 2 分钟。
3. 定时时间到蜂鸣器报警提示。

二、设计思路

本实验要完成智能洗衣机模拟，首先约定： 1，LED0 亮红灯表示洗衣机当前处于弱洗状态； LED1 亮绿灯表示洗衣机当前处于强洗状态； LED2 亮蓝灯表示洗衣机当前处于漂洗状态。2，单片机模拟过程中的 2 秒代表洗衣机实际洗涤过程的一分钟。

需要实现用三个 LED 灯指示三种洗涤状态、用按键控制模拟洗涤三种不同材质的衣服、用蜂鸣器指示洗衣过程结束等基本功能。本实验以 MK20DN512 单片机为核心，设计的重点主要是如何定时？如何定时控制三个 LED 灯的亮灭和颜色？如何将倒计时的数字显示在数码管上？由于软件延时将导致 cpu 的利用率不高，定时不够准确等问题，故本实验中使用 PIT 定时模块这一硬件定时功能。

整个程序从功能上可将其分为 PIT 计时，LED 灯显示、数码管显示，蜂鸣器报警三个部分进行设计，

PIT 定时功能的实现分为三步：SIM 使能相应的 PIT 时钟；初始化 PIT；编写中断服务子程序完成相应的功能；

控制 LED 灯颜色分为三步：将三个 LED 灯相应的端口设置为 GPIO 功能，并设置为输出；初始化；在中断服务子程序中编写控制的程序。

数码管显示分为三步：将数码管相应的端口设置为 GPIO 功能，并设置为输出；初始化；在 main 函数中编写控制的程序。

三、预计实现的结果

（单片机模拟过程中的 2 秒代表洗衣机实际洗涤过程的一分钟）

1，按 SW5 键，表示洗涤丝质衣服：LED2 亮蓝灯持续六秒后熄灭，随之蜂鸣器响，直到按下 SW8 键，蜂鸣器停止响且数码管上的数字归零。

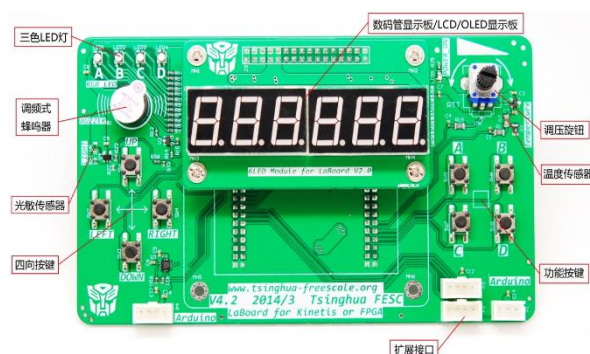
2，按 SW6 键，表示洗涤棉质衣服：LED0 亮红灯持续 4 秒后熄灭，随之 LED1 亮绿灯持续 10 秒后熄灭，然后 LED2 亮蓝灯持续 6 秒后熄灭，蜂鸣器响。直到按下 SW8 键，蜂鸣器停止响且数码管上的数字归零。

3，按 SW7 键，表示洗涤化纤衣服：LED1 亮绿灯持续 8 秒后熄灭，然后 LED2 亮蓝灯持续 4 秒后熄灭，蜂鸣器响。直到按下 SW8 键，蜂鸣器停止响且数码管上的数字归零。

4，蜂鸣器响表示当前洗衣状态结束，提醒人来拿衣服。洗涤不同材质的衣服结束的声音不一样。

四、硬件系统构成

K20 Play board



1.LED 数码管显示

在单片机系统中，发光二极管（LED）常常作为重要的显示手段。由于 LED 显示器主要用于显示各种数字符号，故又称之为 LED 数码管，每个显示器还有一个圆点型发光二极管，由于显示小数点。

2.PIT 模块

PIT 是周期中断定时器的名称，实际上 PIT 模块就是一个 24 位递减计数器，每过一个总线周期，计数器进行减一操作，当计数器减为 0 之后，触发中断，并再次自动载入初值。

3.蜂鸣器模块

蜂鸣器通过控制不同频率使其发出不同的音调。

五、软件系统设计（包括程序流程图（加注释））

首先，初始化将使用到的 LED 灯、数码管、PIT 模块（设置定时周期为 1s）、蜂鸣器；

然后，在 pit_channel0_ISR()和 main()两个函数里面实现控制代码：

程序流图 2-1：

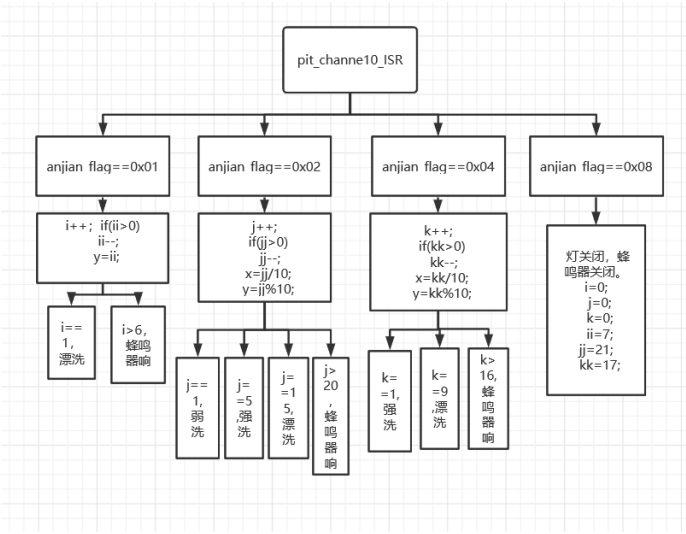


图 2-1 pit_channel0_ISR()main()

程序流图 2-2：

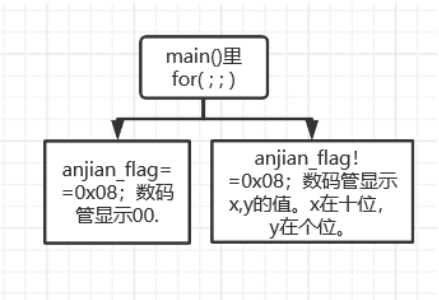


图 2-2 main()

六、调试过程

3. 硬件调试

经验证，本实验所使用的单片机功能均正常，故而不需特别调试。

4. 软件调试：

在调试阶段，出了几个错误，而后改正，一一将这些错误记录如下：

错误一： x,y 两个值不能同时显示，会有跳变。

改正： 将控制数码管显示的代码写在 `main()`里，问题得到解决。

错误二：没办法控制蜂鸣器停止响。

改正：把控制该功能的代码写在 `if` 语句之外。

错误三：洗涤过程结束倒计时还没进行完。

改正：调整相应的参数。

完整实验代码见程序文件中。

七、实验结果

完整的实验结果附有视频，这里只截取有代表性的洗涤棉质衣服的过程：

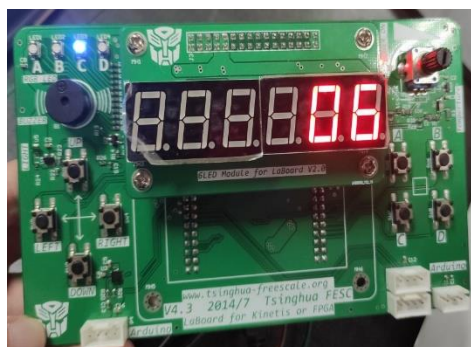
按下 SW6 键，先弱洗：



再强洗：



再漂洗：



洗涤完成蜂鸣器响：



按下 SW8 键蜂鸣器停止响：



八、实验分析

总体来说，本实验是一个比较综合的实验，重点在于如何设计模拟思路能完成实验要求并且使用简单结果明确。然后分模块编写代码，从实验结果来看，正确实现实验所要求的功能。

九、实验心得

单片机是自动化专业的一门重要的专业核心课程，该课程的应用性很强，因此，较多的实验对我们加深对理论知识的理解和应用是很有帮助的。

在从前学习 C 语言、数据结构等课程的时候，往往一头雾水，不知道学着有什么用。通过单片机这门课程，我们有了把所学知识用到实践的机会。理论联系实际的过程，也使得我们对所学知识的内容比如数电、微机原理等课程了解得更深入透彻了。该实验使用的模块较多，功能明确，做完这次实验，我把这学期所学的知识更好地整合了。

最后，感谢课堂中不厌其烦悉心教导的老师和助教老师，你们给我们学习单片机的道路上提供了很多帮助。感谢一起完成实验的组员，是我们友好的互相协作，共同努力，才令这次实验变得如此有趣、有意义。今后我们也将继续认真学习本课程，学习的路途还远，我们的脚步也不会停下。

实验分工：

实验一程序、ppt 及报告：许如玉

实验二程序、ppt 及报告：汪雨甜

实验三程序、ppt 及报告：朱琪霖

实验四程序、ppt 及报告：赖梦婷