嵌入式系统设计专题实验指导书

——基于瑞萨RL78G13学习板的设计与应用

[](http://pic.sogou.com/javascript:void(0))

刘美兰 刘瑞玲

2017年12月

目 录

1. CubeSuite Plus软件安装
2. 基于瑞萨RL78G13学习板的实验内容

**第一章 CubeSuite Plus软件安装**

**瑞萨RL78G13学习板程序安装调试注意事项：**

一、软件安装：安装开发环境(CubesuitePlus)，在安装此开发环境软件前，请按如下步骤操作：

1、请在光盘中“CubesuitePlus”文件中，选择安装微软公司的“NET Framework 4”

软件包；

2、请选择安装“Microsoft Visual C++ 2010 SP1”软件包（安装前卸载电脑上的Microsoft Visual C++ 2010）；

3、请选择安装“CubeSuitePlus\_Package\_V20000a”软件包，即开始安装CubesuitePlus工作。

注意：软件所在路径中不能有中文。

二、软件编程和调试：打开开发环境（CubesuitePlus）的界面，开始新建代码项目及调试操作，具体详情， 请查阅《RL78\_G13 DEMO BOARD 使用指南》中“开发环境及开发工具使用说明”一章的说明；

三、 简易Ez-cube仿真板的使用：

1、固件更新：

（**1**） 驱动程序安装

如果没有安装用于EZ-CUBE 的USB 驱动程序，请在光盘“仿真板 USB DRIVER”文件夹中，选择适合你当前操作系统的驱动文件，完成驱动文件安装。

（**2**） **USB** 连接线

在电脑上首次使用EZ-CUBE，将EZ-CUBE 通过USB 线缆连接到主机，不要将**EZ-CUBE** 与目标板相连，连接成功后状态指示灯显示为红色。

（**3**） **EZ-CUBE** 固件更新程序的启动

启动EZ-CUBE 固件更新程序【QBEZUTL.exe】，如下所示。

QBEZUTL.exe 版本为V1.14 及以上版本



（**4**） 选择固件

点击菜单栏中的[…]按钮，选择固件 RL78-OCD-FW(except G10).hex or RL78G10-OCD-FW.hex。

（**5**） 更新固件

点击[Start]按钮，开始更新固件。固件更新完成，显示以下对话框。



（6） **EZ-CUBE** 固件更新程序的关闭：固件更新完成，点击[EXIT]按钮，退出固件更新程序。

（**7**） 拔除**USB** 线缆：从EZ-CUBE 或主机上拔除USB 线缆，再次连接时需等待15s。

（8）为了防止电脑USB接口损坏，当前demo板最大工作电流为200mA（标称工作电流 < 100mA），如需更大电流进行的相关验证，请通过USB线外接5V电源适配器。

2、把简易Ez-cube仿真板插入到RL78/G13 Demo板上“Ez Emulator”所示的接口上，按3-2-7-8-1的顺序连接，4、5、6闲置，然后通过USB线连接简易Ez-cube仿真板到电脑上，即可通过CubesuitPlus进行在线调试。

3、简易Ez-cube仿真板连接和拔出先后顺序：为了防止Ez-cube仿真板损坏，编译调试工程前按2连接好

4、**EZ-CUBE** 开关设置 RL78/G12 (30pin), G13,G14, I1A

SW-1 开关: 请选择″M2″.

SW-2 开关: 请选择″ Int. Clock ″.

SW-3 开关: 请选择″Debug Mode″.

SW-4 开关: 请根据实际目标设备使用选择.

SW-5 开关: 请选择″Other ″.

**第二章 基于瑞萨RL78G13学习板的实验内容**

**实验项目1：学习RENESAS RL78/G13嵌入式微控制器开发环境**

**实验目的：**

掌握RL78/G13的集成编译环境CubeSuite Plus和仿真调试工具EZ-CUBE的使用方法。

**实验基本要求：**

能够熟练使用RENESAS嵌入式设备的集成编译环境CubeSuite Plus和仿真调试工具EZ-CUBE。

**实验内容提要：**

熟悉RENESAS 嵌入式设备的编程环境CubeSuite Plus和调试工具EZ-CUBE，能够根据实验要求在编程环境下创建相应的工程项目，包括文件定义、变量定义、程序结构设计、算法实现等；在EZ-CUBE环境下，掌握程序的调试步骤以及排除程序中的错误等。

**具体任务：**

**一、任务一：学习视频资料和“RL78 G13 Demo板使用指南.pdf”，创建新工程：**

RL78 系列 MCU 产品是瑞萨（Renesas）电子新一代微控制器系列，为 16 位 CISC 硬件架构设计。其中RL78/13为通用产品，具有集成 Timers, SIO, 8/10-bit ADC, DMAC ,POC/LVI, data flash 等外设。所使用的MCU为R5F100LG型。

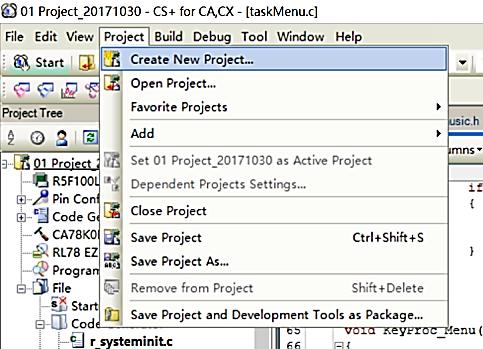
Renesas公司为其下属产品推出了基于.NetFramework4.0框架的Cubesuite+集成开发环境，并在其中有代码生成器 (Code Generator) 供使用。同时该公司也提供了相配套的调试器Ezcube，可以在线调试，简化了开发流程。

1、环境搭建

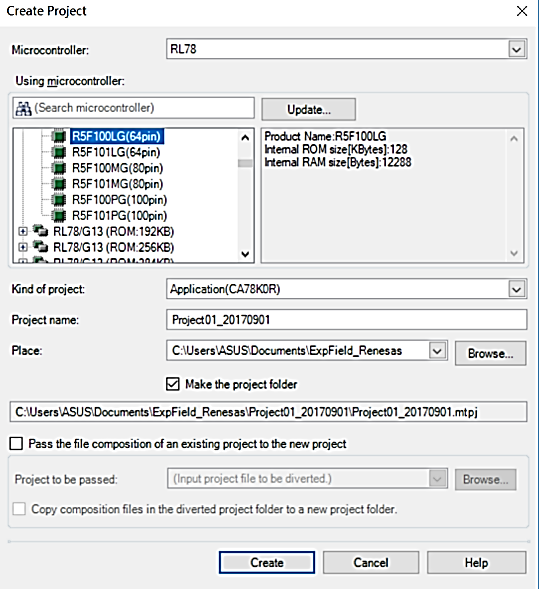
主要有两步：首先是安装.NetFramework4.0软件框架以及Microsoft Visual C++ 2010 SP1，并安装Cubesuite+集成开发环境。然后是安装Ezcube调试器的驱动。

2、新建工程

打开集成开发环境，新建工程。

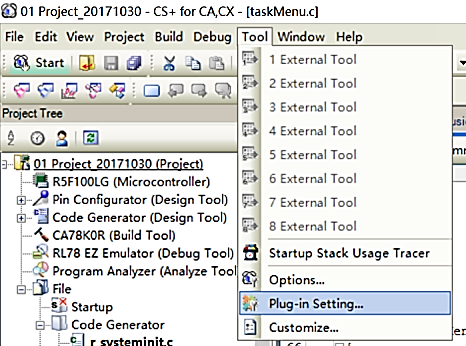


选择开发板对应的器件型号，并选择工程路径，给工程起个名字。

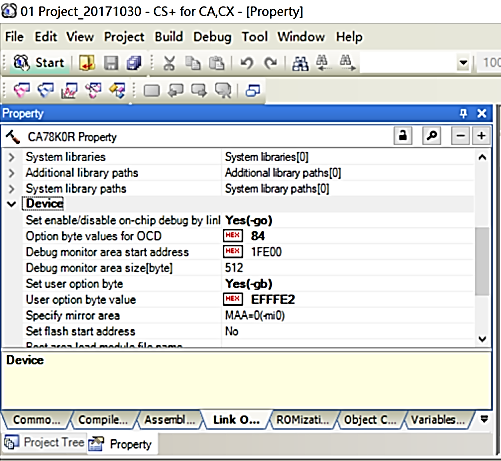


工程就建好了。不过还有一些有关开发板通用的设置，方便以后编程下载调试用。

（1）开启代码生成器功能：

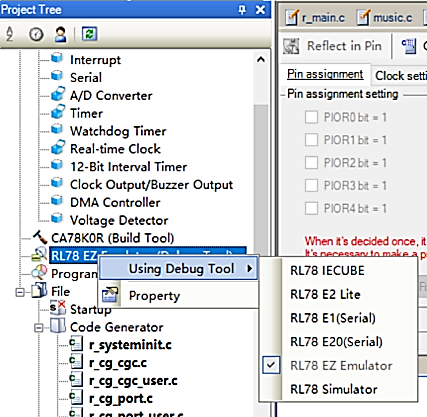
 

（2）进行编译器再设置，选择“CA78K0R(Debug Tool)”项双击，对“Link options”中相关项值进行设置。



（3）选择调试方式：

在右边Project Tree中如图位置右击选择。新工程默认是要进行软件仿真的，不过我们选择使用调试器，这样就可以进行硬件在线调试了。



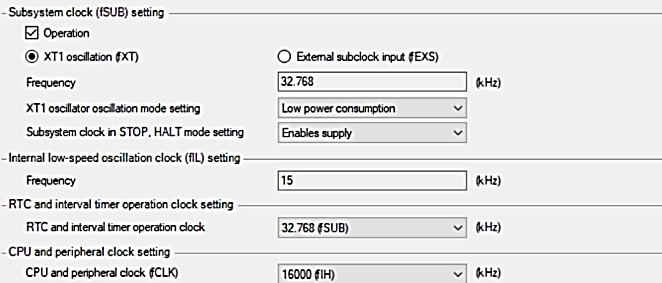
**3、程序编制调试的注意事项：**

对于一些有关单片机的操作大部分可以通过代码生成器实现，我们可以把更多的精力放在设计控制逻辑上。代码生成器的使用相当便利，而且可以自动生成一些接口函数供使用。主要需要注意的是：

（1）开发板上使用的晶振是16MHz。



并且使用实时时钟时需要提供开启副时钟系统32.168MHz



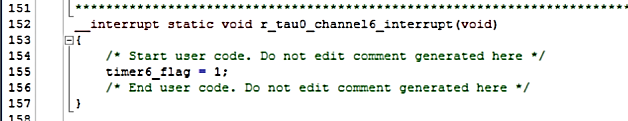
（2）需要启用片上调试



（3）对于看门狗定时器，如有需要则用之，但一定要记得喂狗，不然系统则会不停复位，不能正常工作。如果代码不是特别复杂，可以暂时先禁用之（不过不是一个好习惯）。



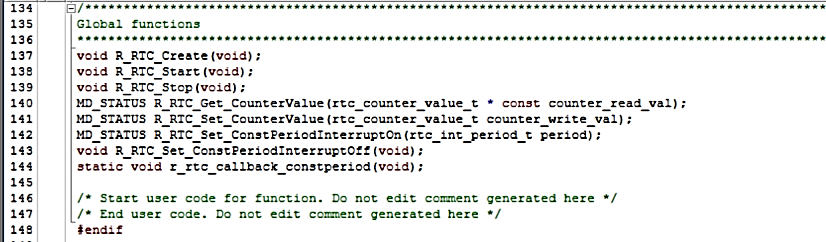
（4）用户自己写的代码一定要写在指定区域，不然在更改代码生成器设置时会自动清除注释之外的用户代码！



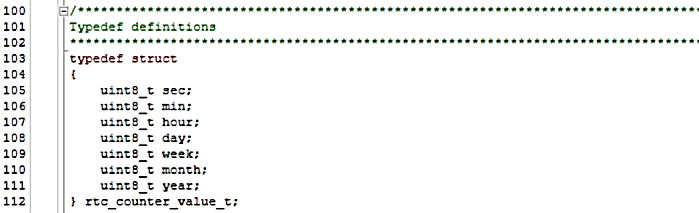
（5）关于中文注释，由于转码问题有可能会导致之后的汇编错误，导致程序出现一系列不可控的错误。一个可见的征兆是在编译过后原本的非代码区域（比如空行，注释）也有汇编语句。解决办法是在每个中文注释的结尾处添加一个空格。

（6）函数调用

对于一些常见的外设操作，可以使用由代码生成器生成的函数实现。



对于常见的函数参数以及用到的结构体在头文件中也有定义，建议使用。



要大致了解各函数的功能，各个模块的初始化函数在初始化的过程中就已经自动执行了，而对于其他功能需要记得调用，比如说定时器模块需要用户自行开启/关闭。

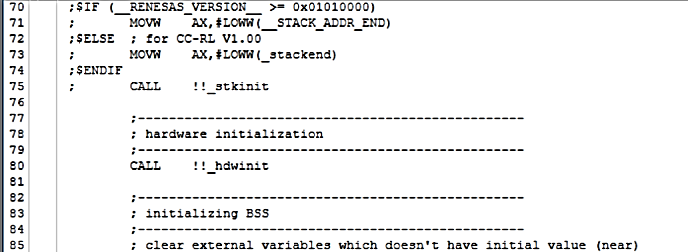
（7）启动顺序

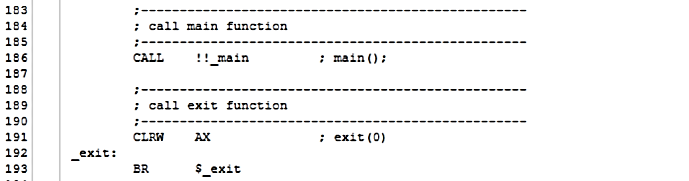
在代码生成后程序就已经写好的初始化的部分，有必要了解下启动的流程，避免一些不易察觉的错误。

留意以下路径：

C:\ProgramFiles(x86)\RenesasElectronics\CS+\CACX\Device\RL78\PG\Generate\RESETPRG

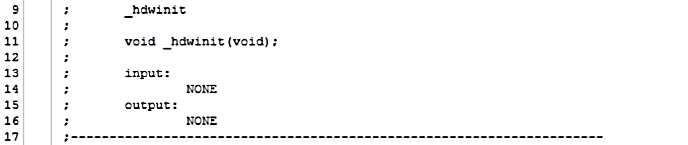
其中的cstart.asm是启动文件，注意如图位置：





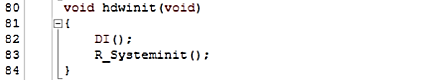
执行顺序：stkinit->hdwinit->main

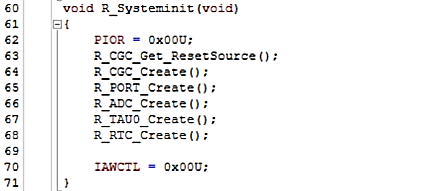
其中stkinit在stkinit.asm中已经实现，hdwinit在hdwinit.asm中的操作是调用工程中自动生成的hdwinit()，main就是我们需要实现的main()函数。



再看一下工程中自动实现的部分：

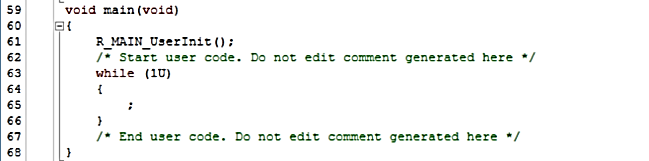
r\_systeminit.c中：





可见各个模块的初始化Create函数已经自动调用了。DI()是关中断的操作。

r\_main.c中：



其中EI()是使能中断的操作。

至此可以总结一下启动顺序：

* 初始化栈stkinit
* 硬件初始化hdwinit(): i. DI()禁用中断

ii.调用R\_SystemInit(): 各个外设的初始化Create()函数。

* 开始执行main(): i.调用R\_Main\_UserInit(): EI()使能中断

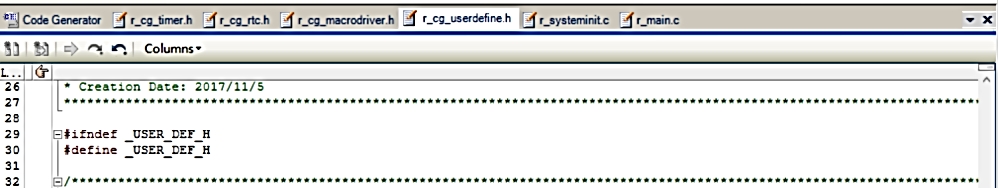
ii.执行用户代码。。。

如果人为的把hdwinit()在main()中又调用了一遍，中断功能不能使用。

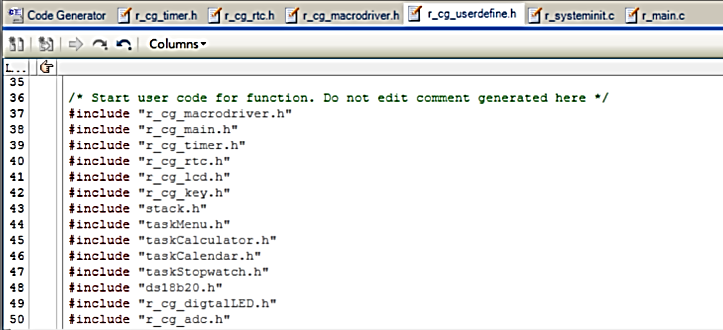
（8）包含结构

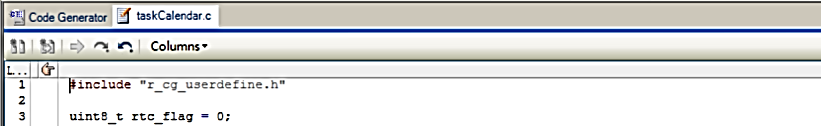
当工程较大时，头文件数目众多，难免出现包含的错误。通常有可能一个包含错误，在各个文件链接时都报一遍错，于是会出现几百个错误的情况，但其实错误只有一处。一种较为规范而易于管理的包含结构如下：

1. 所有的头文件用#ifndef…#define…#endif防止重复包含。



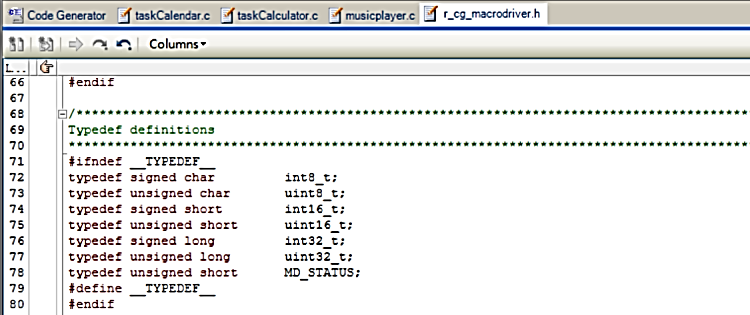
1. 所有要用到的头文件都包含在r\_cd\_userdefine.h中，而所有其他源文件均只包含r\_cd\_userdefine.h



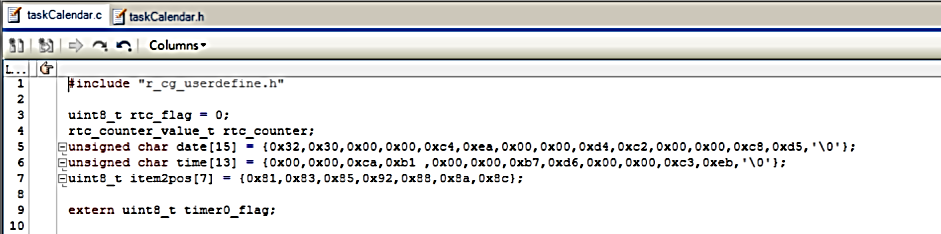


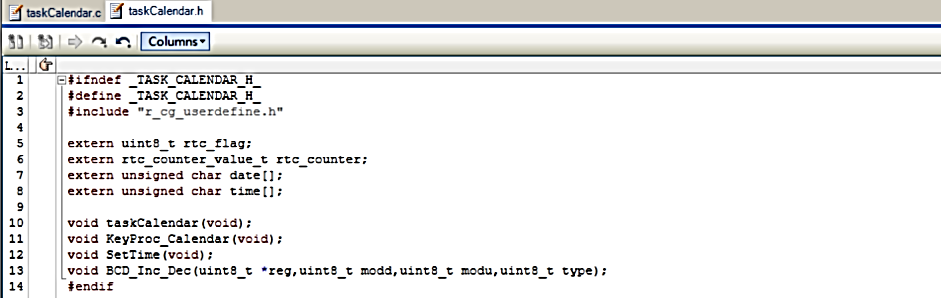


c. r\_cd\_userdefine.h中包含顺序一定将r\_cd\_macrodriver.h放在最前！因为r\_cd\_macrodriver.h中重命名了数据类型，而这些重命名的数据类型在后面其他头文件中会用到。



d. 全局变量在各个源文件(.c)中定义，初始化，在对应的头文件中做全局声明。这样只需一次声明，而可以在全工程中使用全局变量。

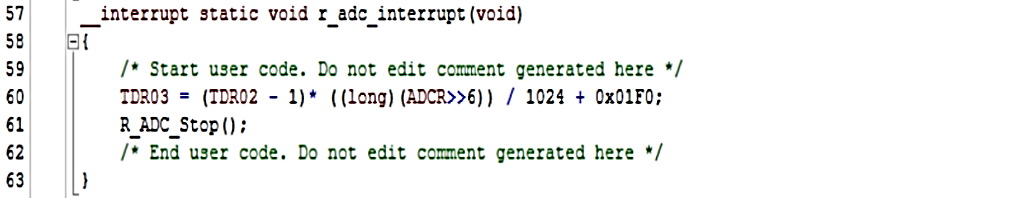




（9）中断使用：

一些常见功能（键盘，音乐播放，显示）的实现可以体现出一般嵌入式系统的软实时特性。对于实时性有严格要求的系统宜采用操作系统，而实时性不太严格的系统（即所谓软实时系统）则可以通过一些编程架构实现。这些编程方法都需要合理使用中断。

一个总体的原则是如果中断处理程序不太复杂，耗时不多的话，可以直接在ISR中处理，如图：

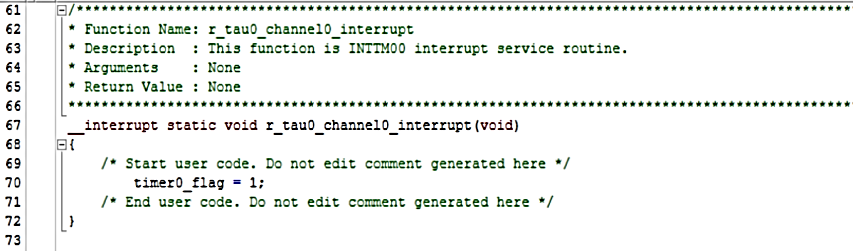


如果中断处理程序需要占用一定量的CPU资源的话，则尽量不要直接在ISR中处理，而要换一种架构。（原因是如果一个中断出现频率较高而又有较高优先级，中断处理程序直接在ISR中处理有可能是CPU一直被该程序占据，其他代码没有机会执行，相应功能当然就不能用了。）

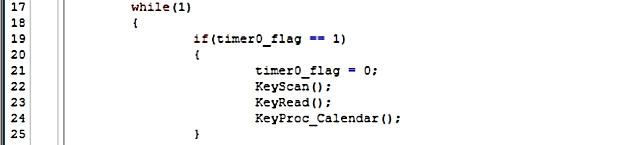
较为易用的一种如下：

比如定时器中断，这里需要用到一个标志变量flag，默认为0

在定时器中断的ISR中仅需要将其置1。



而较繁杂的中断处理程序在主循环while(1)中实现：



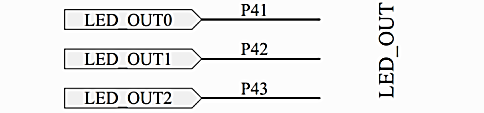
这样的好处是可以保证每个功能模块都有机会被执行，于是各个功能都可以正常运行。实验效果也很明显，这样的架构可以大大改善卡停的现象。然而也有缺点，就是在需要多个主循环时每个循环都需要添加如此一段代码，使代码很冗长。

了解了以上程序编制中的注意事项就可以规避掉日后的程序编制中七成的错误，也就可以把更多的精力放在程序设计上了。

1. **任务二：跑马灯**

**分别用Delay（）函数和Timer模块控制实现跑马灯功能，使二极管D3、D4、D5能够按照一定的规律循环点亮或熄灭**

1. 硬件连接

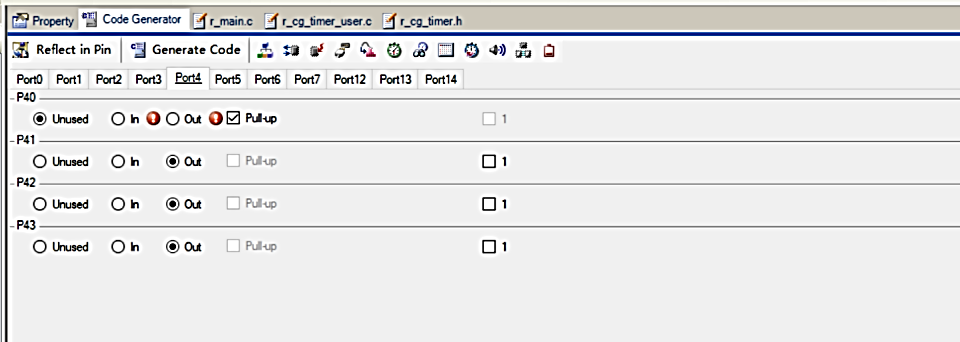


由原理图可知P41、P42、P43连接LED灯。

1. 程序设计

要使灯轮流亮灭，需要用到两个外设：GPIO输出电平，定时器Timer控制时间。

GPIO配置P41、P42、P43为输出模式。



设置Timer0为Interval Timer模式，设置周期1秒。

每过一秒，触发一次中断，timer0\_flag变化一次，while中相应代码段执行一次，用来控制哪一位亮的count就变化一次。Count有三个状态，对应P4三个灯亮的状态。每次执行while中相应代码段，根据count不同的状态，P4输出相应的电平，达到灯亮灭状况随时间改变的目的。

注意使用定时器前要先开启定时器R\_TAU0\_Channel0\_Start()。

**实验项目2：处理器接口模块设计实验**

**实验目的：**掌握处理器各接口模块的使用和编程方法。

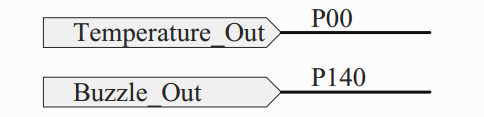
**实验基本要求：**通过对各接口模块参考示例程序的学习，了解其工作原理，并能根据实验要求修改或扩充示例程序及功能。

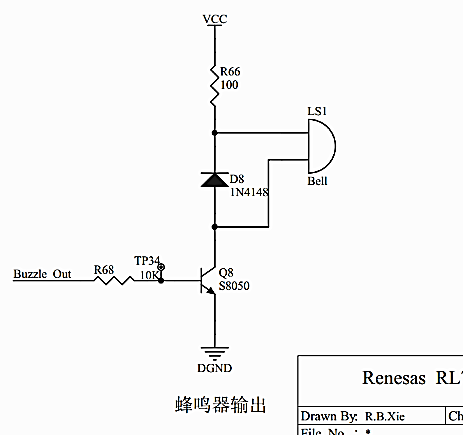
**实验内容提要：**能够编写嵌入式C语言并完成：蜂鸣器按照某种规律工作；7段LED显示、LCD128\*64显示；键盘输入；定时中断的功能。

**具体任务：**

**一、任务一：学习Buzzle模块的调用和编程，能够重建该工程，并控制蜂鸣器发出不同节奏和频率的声音。（注意：要保证蜂鸣器能正常输出声音，输出频率设置无需太高，1KHz～10KHZ 即可）**

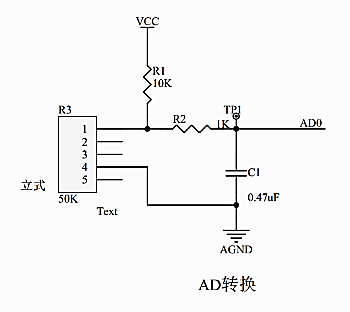
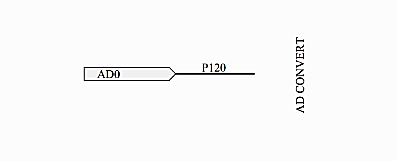
1、硬件连接





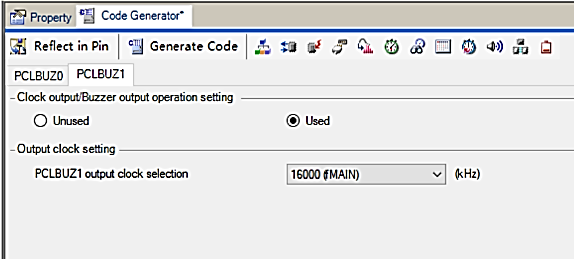
P14.0负责控制蜂鸣器输出。P14.0为1则Q8导通蜂鸣器响，P14.0为0则Q8关断蜂鸣器不响。

另外还要探究一下方波占空比对蜂鸣器发声的影响。这里要用到开发板上的电位器。

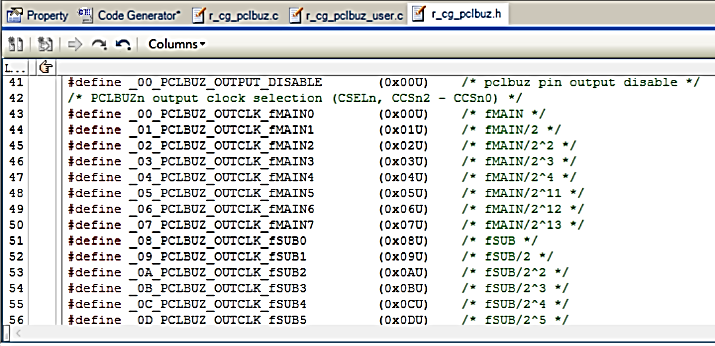


2、软件设计

微控制器及代码生成器本身提供了蜂鸣器的驱动：



但由其结构知该驱动只能发出有限的几个频率控制信号：





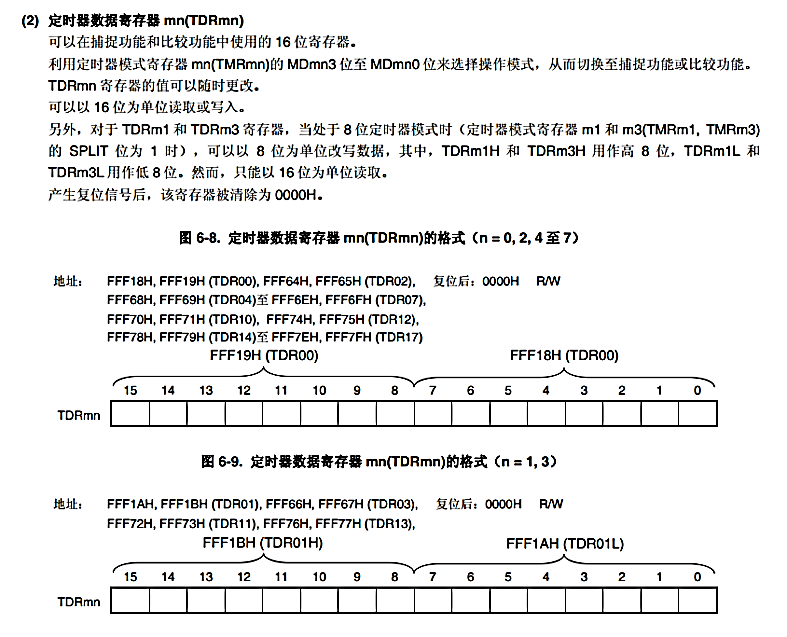
其实这个驱动本质是将时钟信号分频后输出。如果只是让蜂鸣器响起来，那么就已经足够了。如果要控制音调的话，那么需要用别的方式。

这里将采用定时器的方式来控制频率。总体思想仍然是P14.0输出一个方波，不过方波周期和定时器周期成正比（所采用的操作是每次定时器中断时将P14.0输出取反，这样每两个定时器的周期是一个输出方波的周期。这时P14.0是被当做普通GPIO口操作的。），然后实时控制定时器的装载值来改变输出方波的周期，实现频率控制。

具体实现会有如下几个问题：

1. 怎样控制定时器的频率？

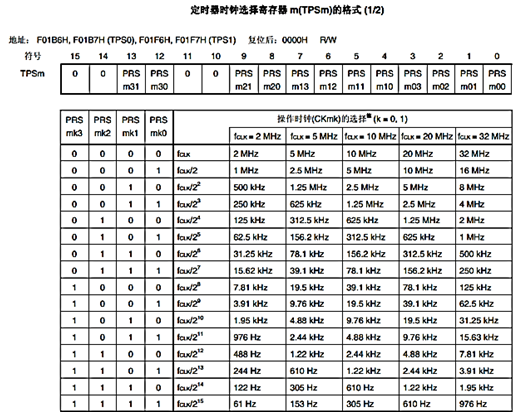
改变定时器的计数装载值。注意Interval Timer模式下默认是递减技术。



1. 给数据寄存器写入某一个值，他会输出多少的频率呢？

要解决这个问题，可以先参考代码生成器是怎样做的。先使用代码生成器配置一个定时器周期为1秒，观察一下：



具体的周期算法：主时钟晶振16MHz，根据TPS0中数据先做预256分频成62500Hz，然后计数值为0xF423=62499，共62500个状态，于是输出频率为1Hz。

总结一下，主时钟晶振（16MHz）先根据TPSm寄存器预分频得到计数的频率，再根据数据寄存器TDRmn得到定时器中断的频率。

1. 想让蜂鸣器唱歌，那么不同音调应该让P14.0输出频率为多少呢？

这个问题需要查找资料了，看看不同音调对应的频率是多少，再根据找到的频率算出对应的定时器装载值。（资料来源百度文库）

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 16M直接分频，数多少下？ | | | | | | | | | | | |
| 低音 | 频率/Hz | 十进制 | Hex | 中音 | 频率/Hz | 十进制 | Hex | 高音 | 频率/Hz | 十进制 | Hex |
| 1 | 262 | 61069 | EE8C | 1 | 523 | 30593 | 7780 | 1 | 1046 | 15296 | 3BC0 |
| 1# | 277 | 57762 | E1A1 | 1# | 554 | 28881 | 70D0 | 1# | 1109 | 14427 | 385B |
| 2 | 294 | 54422 | D495 | 2 | 587 | 27257 | 6A79 | 2 | 1175 | 13617 | 3531 |
| 2# | 311 | 51447 | C8F6 | 2# | 622 | 25723 | 647B | 2# | 1245 | 12851 | 3233 |
| 3 | 330 | 48485 | BD64 | 3 | 659 | 24279 | 5ED7 | 3 | 1318 | 12140 | 2F6B |
| 4 | 349 | 45845 | B315 | 4 | 698 | 22923 | 598A | 4 | 1397 | 11453 | 2CBD |
| 4# | 370 | 43243 | A8EB | 4# | 740 | 21622 | 5475 | 4# | 1480 | 10811 | 2A3A |
| 5 | 392 | 40816 | 9F70 | 5 | 784 | 20408 | 4FB8 | 5 | 1568 | 10204 | 27DC |
| 5# | 415 | 38554 | 969A | 5# | 831 | 19254 | 4B35 | 5# | 1661 | 9633 | 25A0 |
| 6 | 440 | 36364 | 8E0B | 6 | 880 | 18182 | 4705 | 6 | 1760 | 9091 | 2382 |
| 6# | 466 | 34335 | 861E | 6# | 932 | 17167 | 430F | 6# | 1865 | 8579 | 2183 |
| 7 | 494 | 32389 | 7E84 | 7 | 988 | 16194 | 3F42 | 7 | 1976 | 8097 | 1FA1 |

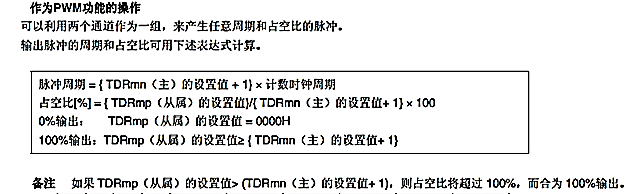
实践效果表明直接按照上表去做频率还是较低，把每个音调高两个八度则更适合蜂鸣器。每个音调高八度对应频率乘两倍。

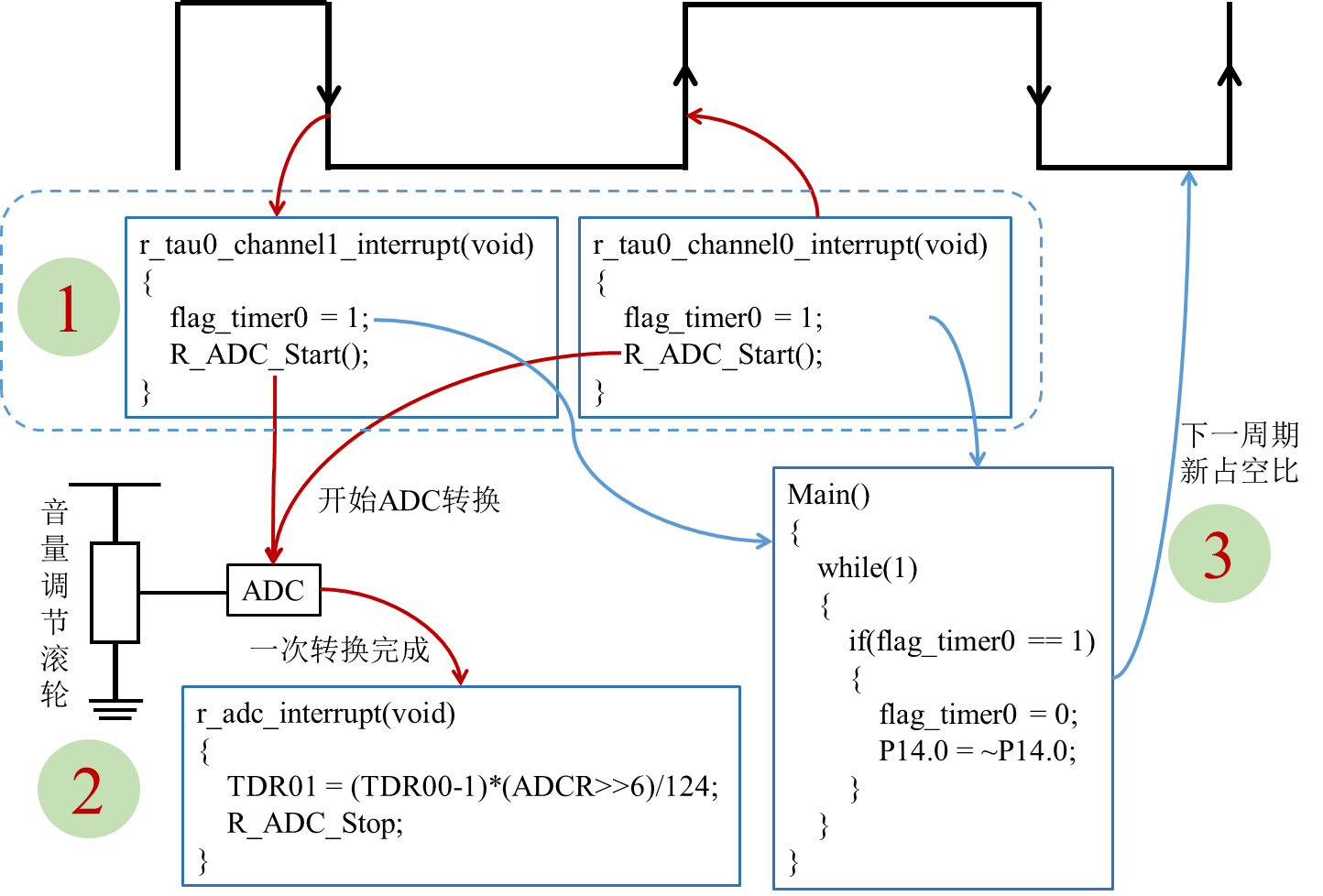
解决了以上问题就可以让蜂鸣器输出某一个音调了。那么怎样在音调之间切换变化呢？可以再开设一个定时器(不排除有效率更高的做法)，固定频率切换。如果某个音调要多唱几拍，那就连续几个都是同一个音调。空拍则取消P14.0的输出状态（修改模式寄存器TMRmn的值）。

实际使用是可以将其封装一下做成一个播放器，使用的时候可以调用开始和结束的函数（实际操作就是开启、关闭两个定时器）。具体的乐谱以数组指针的形式传入函数中，演奏相应的乐曲。乐谱的格式如下：1234567表示七个音调，-1~-7表示低八度，11~17表示高八度，空拍以0表示，乐曲结束处加以END（可宏定义其值为20）。

另外还有一个问题：方波的占空比对声音有什么影响？实际实验的结果是：不同的占空比音质不同。同一频率的方波输出，有的占空比好听，有的就刺耳。实验的方法如下：

这里需要用两个定时器配合的脉宽调制模式实现不同的占空比。

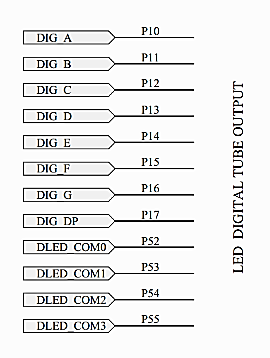


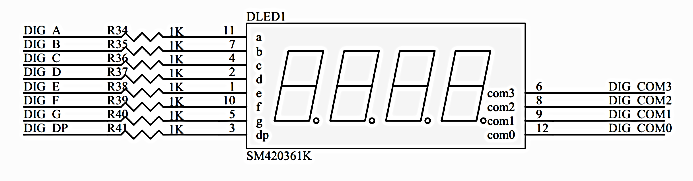


另外不同播放速度下最合适的占空比是不同的。这样就可以用滚轮手动调节占空比，以使播放的效果最佳。

**二、任务二：学习数码管（数字LED）模块的使用和编程，能够实现从0-20顺序循环显示。**

1、硬件连接





开发板上使用的是共阴极数码管，写0则不亮，写1则亮。

2、软件设计

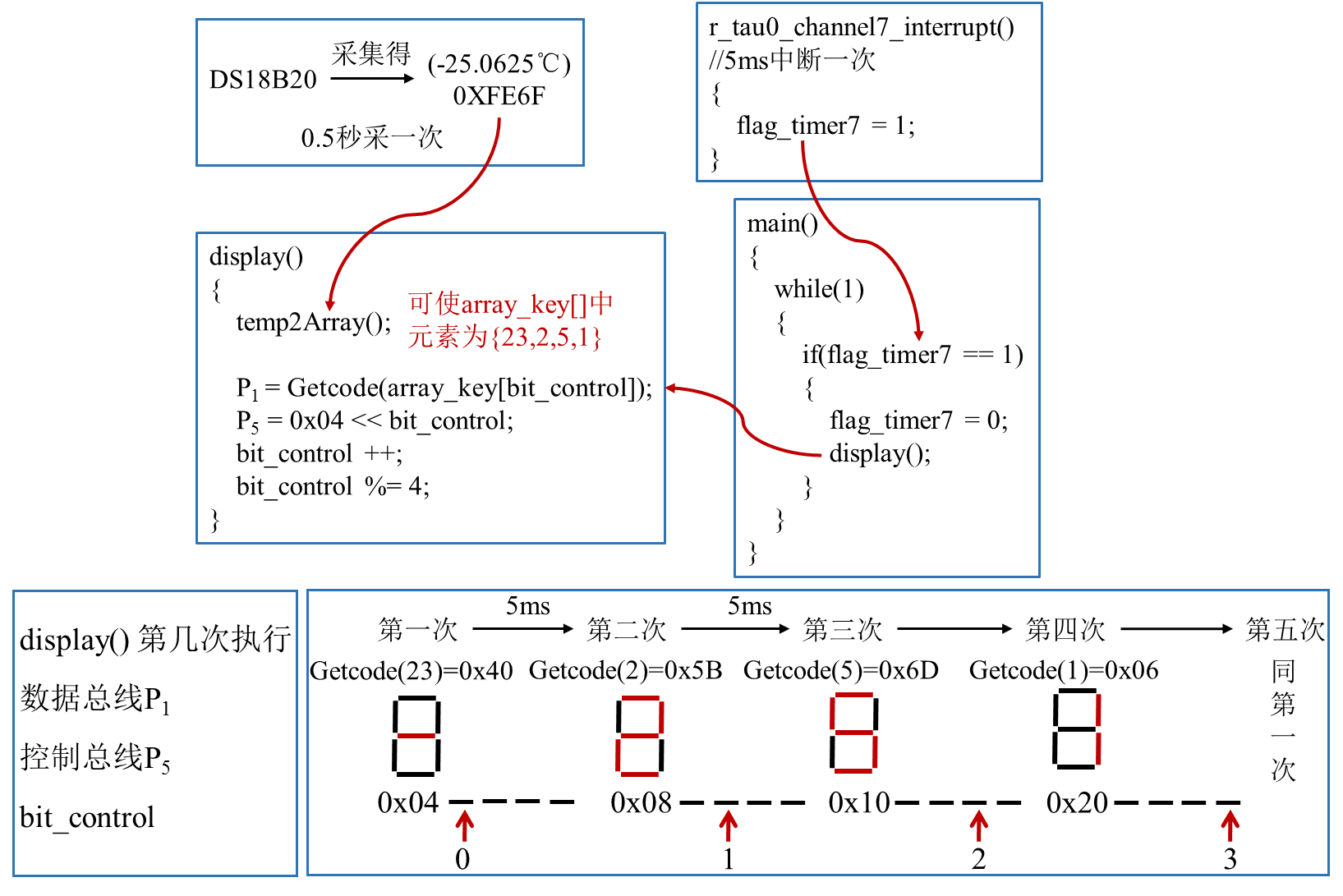
数码管的动态显示原理。

由原理图可见一根四位数码管有八根数据线，四根控制线。四根数据线分别控制每一个显示位有没有显示内容，八根数据线同时控制四位的显示内容。每一位上有七段数码管是显示数字的，一段是显示小数点的，八根数据总线恰好分别与之对应，控制每一段亮还是不亮。

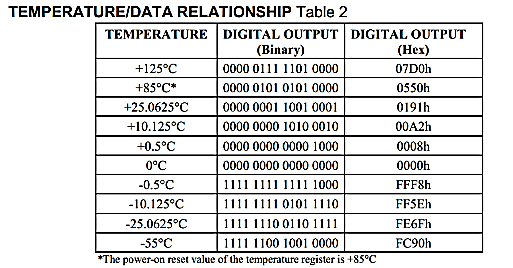
要想让四位上同时显示不同的内容，具体做法如下：分别每个时刻只让一位有显示内容，此时数据总线上为当前位应该显示的内容，下一个时刻就只让下一位上有显示，此时数据总线上为下一位应该显示的内容。如此四位轮流显示，数据总线上依次为某位上该显示的内容，循环往复，当循环的时间足够快时，就看起来好像四位同时显示不同的内容了。

另外直接如上显示会发现显示较为模糊，即所谓”消影”问题。办法是在下一位显示之前先将前一位显示关闭即可解决。

以下是软件编制流程，是按照以上原理编制的。

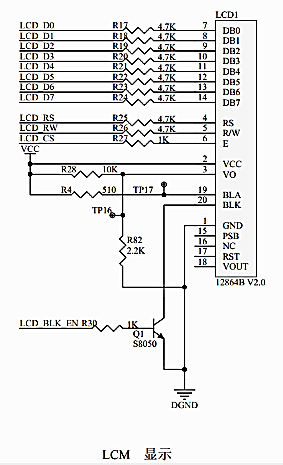
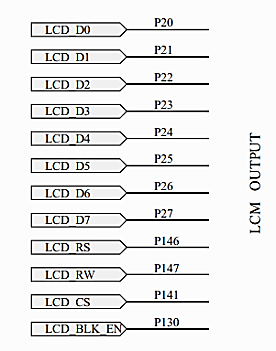


DS18B20数据手册：



**三、任务三：学习LCD液晶屏模块的使用和编程，能够实现2-3个静态画面循环显示，含中文、英文、数字等符号，能够通过变量控制LCD显示内容和位置，能够显示一个简单的图形图片。**

1、硬件连接



2、软件编制

实现这些功能的关键在于12864液晶驱动(ST7920)的使用上。能够实现图片、文字、符号的显示后，就不难实现循环显示之类的功能（后者只是多加了一些逻辑上的操作，较易设置）。

（1）液晶屏显示原理

主要部分：DDRAM,GDRAM,CGRAM,CGROM,HCGROM,IRAM,BF,AC（共8个）

BF表示当前液晶是否在工作，决定用户能否对液晶进行读写操作。

AC是个寄存器，记录当前光标所在位置。可以有自增功能。

实际用户要使用的：画图像就对GDRAM操作。在其上对应位置写入表示各个像素点亮不亮的字节。

写字符就对DDRAM操作。对应位置上写入ASCII码，驱动芯片自动从HCGROM中

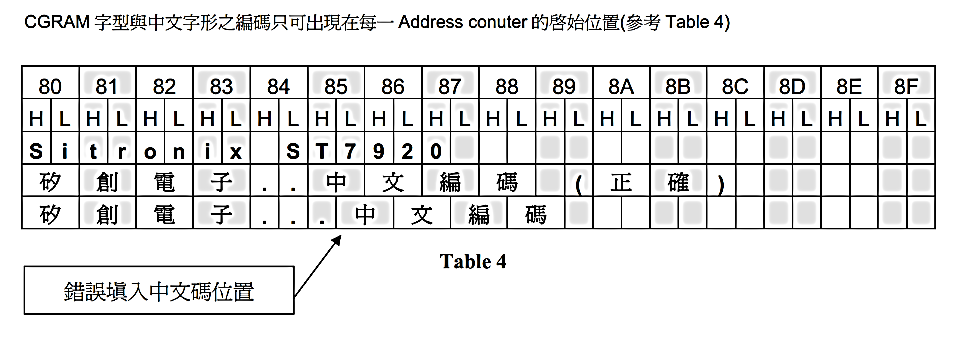
找到相应字符并显示。对应位置上写入GB2312码，驱动芯片自动从CGROM中找到相应字符并显示。(简体中文)对应位置上写入CGRAM地址，驱动芯片自动从CGRAM中找到相

应用户自订字符并显示(暂不用)。IRAM貌似市场上的12864没有该项功能，暂不用。

（2）一般地址寄存器AC自不自增，自增还是自减是需要配置的。在官方样例的初始化函数中是将其配置成自增的，之后我们沿用这一配置。

（3）显示图片：着重注意一是对于图像的取码方式（在用取模软件时），二是要注意在GDRAM上作画时AC指标是八个点横向自增的。

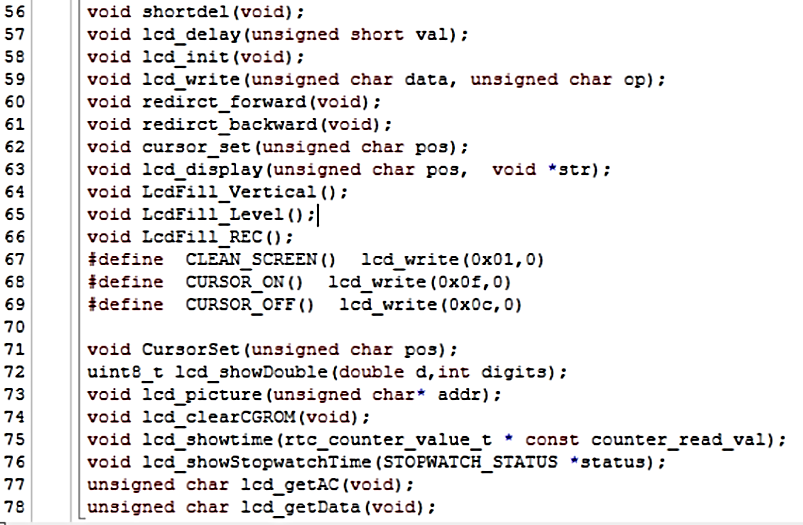
（4）中文字符，可以考虑直接查找gb2312码使用，另外注意书写位置，前后空格是偶数。

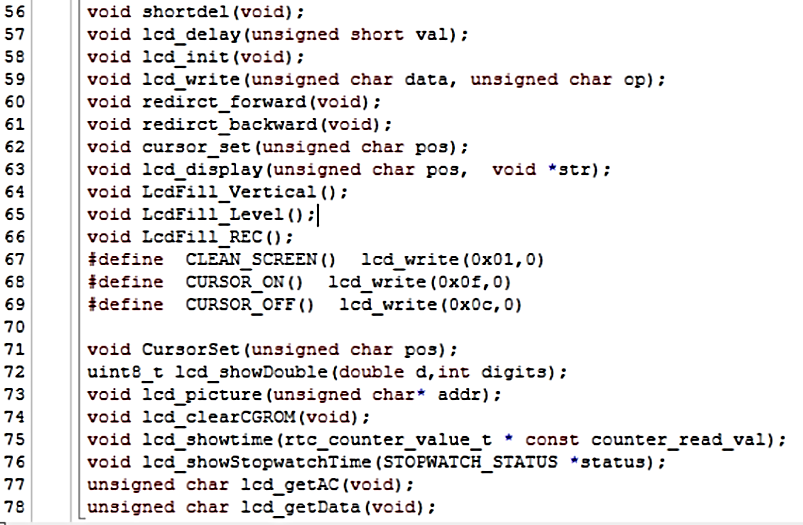


（5）有关位置读取。直接读取AC寄存器（这里编制了lcd\_getAC(void)供使用，返回值为uint8\_t 类型的数据），获得的是0x00，0x01…类似的值，而在使用指令改变AC值时，需要用0x80，0x81…类似的值。注意区别。



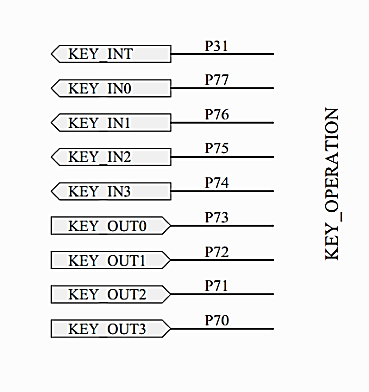
然后根据原理就可以来编写驱动程序。需要说明的是一些基本的操作如写指令，读数据都是按照手册上的时序来写的，各个不同版本的程序差别并不大（主要在于IO口操作，管脚宏定义以及延时的控制上面），这里直接使用样例中的代码。而在液晶基本原理熟悉后用户自己可以根据需求用这些基本操作来编制自己的驱动程序，本程序中的r\_cg\_lcd.c中也有相当一部分比例自己根据需要做出的函数。同时样例代码的个别之处也是有问题地，根据原理在发现后不难即使改正。

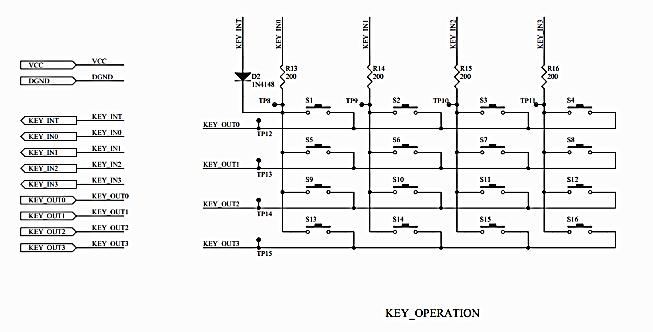




**四、任务四：学习键盘模块的使用和编程，能够定义键盘功能，例如数字0~9，符号“＋”、“－”、“×”、“÷”、“＝”、删除和小数点等。在键盘按下后能够在LCD屏幕上依次显示输入的内容，能够在数码管上显示定义的数字，实现简单计算器和日历功能；数码管上显示秒表。**

1、硬件连接





2、软件编制

样例代码给出的是行列扫描的办法，对于一般的按键消抖用的是延时的办法，这样缺点主要在于在延时的时候其实大大浪费了CPU资源，同时其他任务不能使用CPU，易造成时序上的混乱。这里在网上找到一种更加稳定高效，且易于扩展的办法。

下面稍作介绍。其实这个办法的本质还是行列扫秒，只不过这个扫描是固定时间间隔的，比如说本例中100ms扫描一次，这个定时扫描的间隔本身就已经有消抖的作用了。每次扫描所需要做的只是改变一些所记录的键盘状态：

volatile unsigned char CF[4]; 记录当前哪一行及哪一列有按下

volatile unsigned char Cont[4]; 记录各行有无键按下，是不是第一次按下

之后根据这些数据结构中的内容，加以逻辑判断，便不难解决三个问题：

1. 当前有无按键按下

for(k = 0;k<4;k++)

{

if(Cont[k] != 0)

{

…

}

}

可在第一次按下按键时检测到，并执行相应操作。

1. 如果有按下，那么是第一次按下，还是之前按下现在没有松开？

if((KeyVal != 0x00) && (CF[0]|CF[1]|CF[2]|CF[3]))

{

…

}

其中(KeyVal != 0x00)说明此时有按键按下，(CF[0]|CF[1]|CF[2]|CF[3])说明是第一次按下。

1. 如果有按下，那么按下的是哪个键？

for(k = 0;k<4;k++)

{

if(Cont[k] != 0)

{

KeyVal += ((0x10<<k)+Cont[k]);

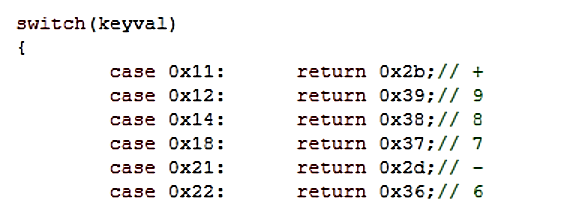
KeyCount++;

}

}

KeyVal = KeyVal2No(KeyVal);

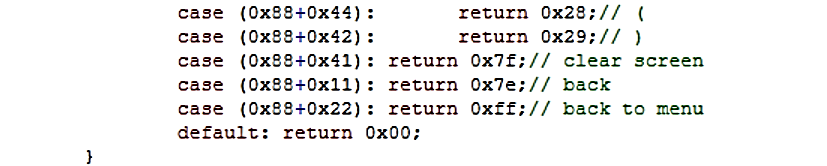
其中KeyVal先按上述算法得到一值，再通过函数KeyVal2No()重新映射到一个更加易用的键值上。



以上举例，这样的键值更利于使用(这里为方便在液晶上直接显示)。

另外这样的算法还有一个好处，就是十分容易扩展出不同的按键类型识别，比如说单击、双击、组合键、长按、短按等，实现一部分的组合键，扩展键盘的功能。

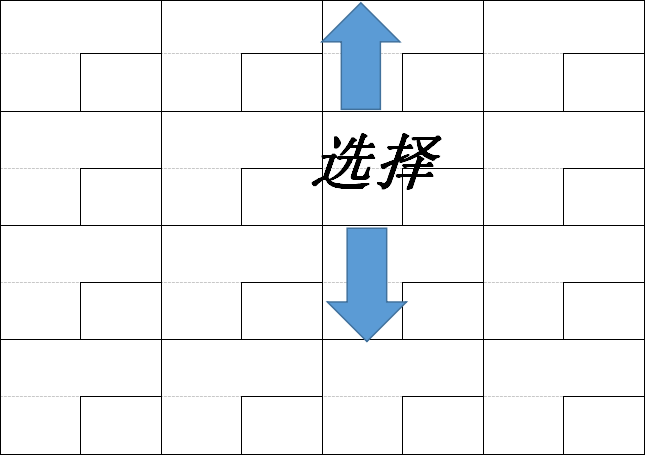
举例如下：



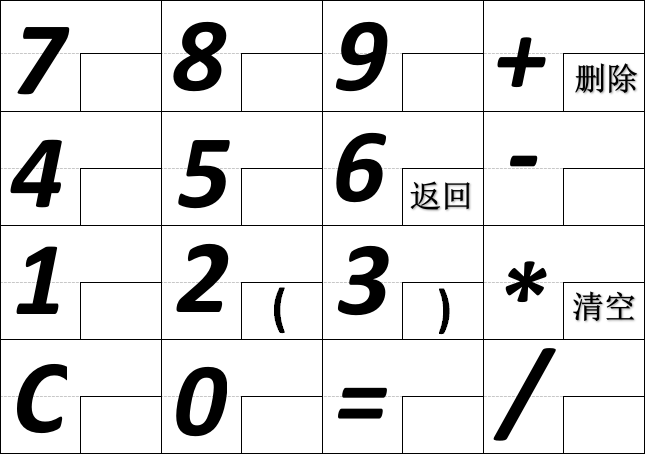
改变键值映射便可实现组合键的识别,但需注意的是，利用本程序的方法对于同一行和同一列的组合键识别方法和上述不同，也不难实现。

（4）键位设置说明书

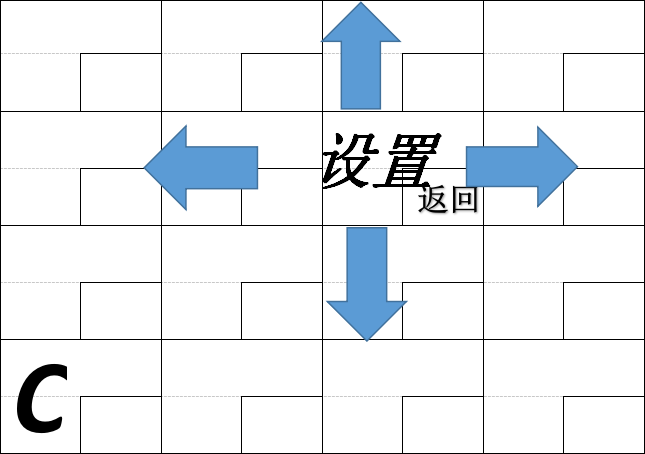
菜单界面：



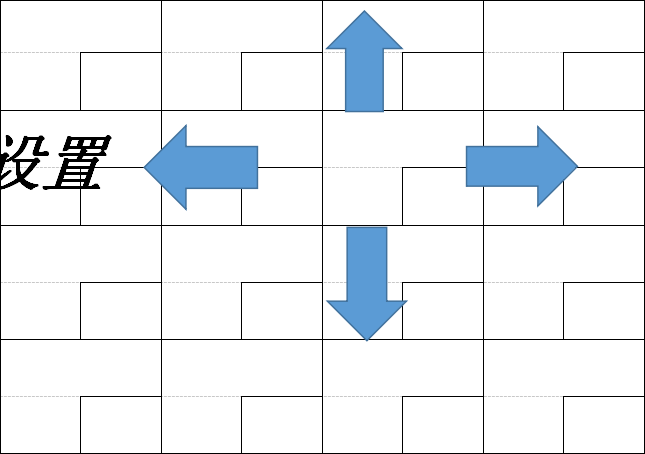
计算器界面：



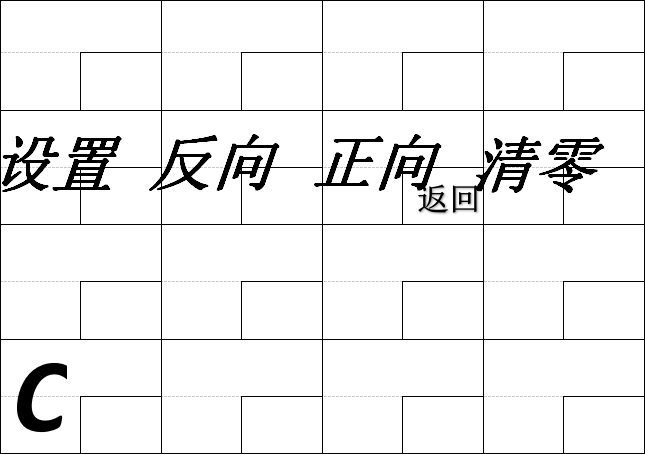
日历界面：



秒表界面（设置计数时间界面）：



**秒表界面（记时界面）：**



注：

A:键盘单个点击时的效果；B:键盘在点击”C”时候组合键位的效果。