

嵌入式系统设计及应用报告

班级：自动化 42 姓名：曹琳阁 学号：2140504027
班级：自动化 42 姓名：林素濠 学号：2140301044

实验项目 1：学习 RENESAS RL78/G13 嵌入式微控制器开发环境（第 1 次实验完成，共 4 学时）

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。

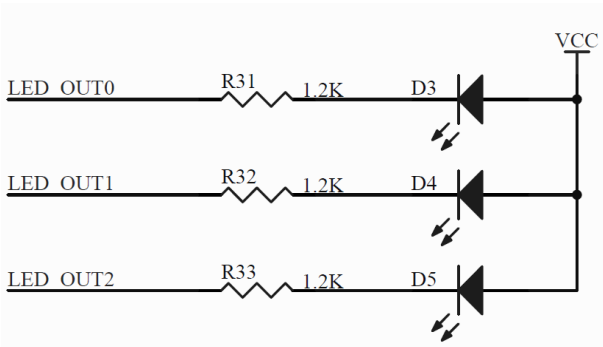
实验基本要求：能够熟练使用 RENESAS 嵌入式设备的集成编译环境 CubeSuite Plus 和仿真调试工具 EZ-CUBE。

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：1) 学习视频资料和“RL78 G13 Demo 板使用指南.pdf”，创建新工程，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；

学习资料：视频资料、例程 Sample_Timer & LED_100LG、文档 RL78 G13 Demo 板使用指南（二、CS+ 开发环境简介及使用说明）。

硬件设计



如图，Led 灯的引脚为 P41，P42，P43，引脚拉低灯亮。
设置 Port4 的 P41，P42，P43 为输出。

P41	<input type="radio"/> Unused	<input type="radio"/> In	<input checked="" type="radio"/> Out	<input type="checkbox"/> Pull-up	<input checked="" type="checkbox"/> 1
P42	<input type="radio"/> Unused	<input type="radio"/> In	<input checked="" type="radio"/> Out	<input type="checkbox"/> Pull-up	<input checked="" type="checkbox"/> 1
P43	<input type="radio"/> Unused	<input type="radio"/> In	<input checked="" type="radio"/> Out	<input type="checkbox"/> Pull-up	<input checked="" type="checkbox"/> 1

使用定时器通道 0，设置时间间隔为 500ms，使能中断，优先级默认。

Channel 0

Interval timer

General setting	Channel 0	Channel 1	Channel 2	Channel 3	Channel 4	Channel 5	Channel 6	Channel 7
Interval timer setting								
Interval value (16 bits)		500		ms		(Actual value: 500)		
<input type="checkbox"/> Generates INTTM00 when counting is started								
Interrupt setting								
<input checked="" type="checkbox"/> End of timer channel 0 count, generate an interrupt (INTTM00)								
Priority		Low						

软件设计

在定时器的中断处理函数中依次点亮一个 led 灯即可。

```
__interrupt static void r_tau0_channel0_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    static char i = 0;
    switch(i)
    {
        case 0:
            P4.1 = 0;
            P4.2 = 1;
            P4.3 = 1;
            break;

        case 1:
            P4.1 = 1;
            P4.2 = 0;
            P4.3 = 1;
            break;

        case 2:
            P4.1 = 1;
            P4.2 = 1;
            P4.3 = 0;
            break;
    }
    i = (i + 1) % 3;
    /* End user code. Do not edit comment generated here */
}
```

另外用延时函数替代定时器，延时函数做空循环即可，如下：

```
int i, j;
for(i = 0; i < 10000; i++)
{
    for(j = 0; j < 10000; j++);
}
```

实验项目 2：处理器接口模块设计实验（第 2、3、4、5 次实验完成，共 16 学时）

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。

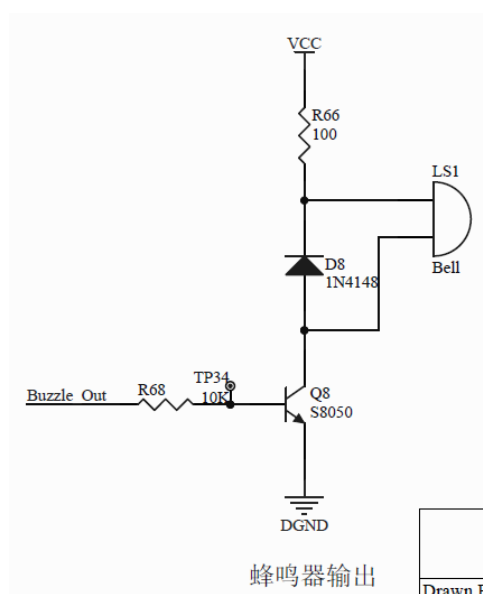
实验内容提要：能够编写嵌入式 C 语言并完成：蜂鸣器按照某种规律工作；7 段 LED 显示、LCD128*64 显示；键盘输入；定时中断的功能。

具体任务：2) 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）

学习资料：例程 **Sample_Buzzle_100LG**。第 1 种实现方式，通过系统参数配置改变频率，通过延时改变节奏；第 2 种实现方式，通过循环和延时（或者定时器）找到音乐声调对应的频率。

硬件设计

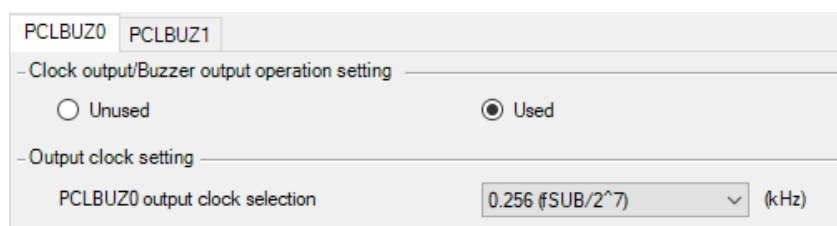
蜂鸣器连接的引脚为 P140，利用一个三极管工作在开关状态提供驱动。这里输出不同频率的方波，蜂鸣器发出不同频率的声音。



Buzzer Output 0 模块的输出引脚为 P140，刚好跟蜂鸣器的引脚对应。

64 P140/PCLBUZO/INTP6

Buzzer Output 0 模块设置好输出频率即可。更改频率在这里更改即可。



软件设计

在主函数中调用 `R_PCLBUZ0_Start()` 函数开启蜂鸣器输出，调用 `R_PCLBUZ0_Stop()` 关闭蜂鸣器输出。实际上板上为有源蜂鸣器，只要 P140 输出高电平($P14.0 = 1$)，蜂鸣器即响，输出低电平($P14.0 = 0$)，蜂鸣器即不响。

3) 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。

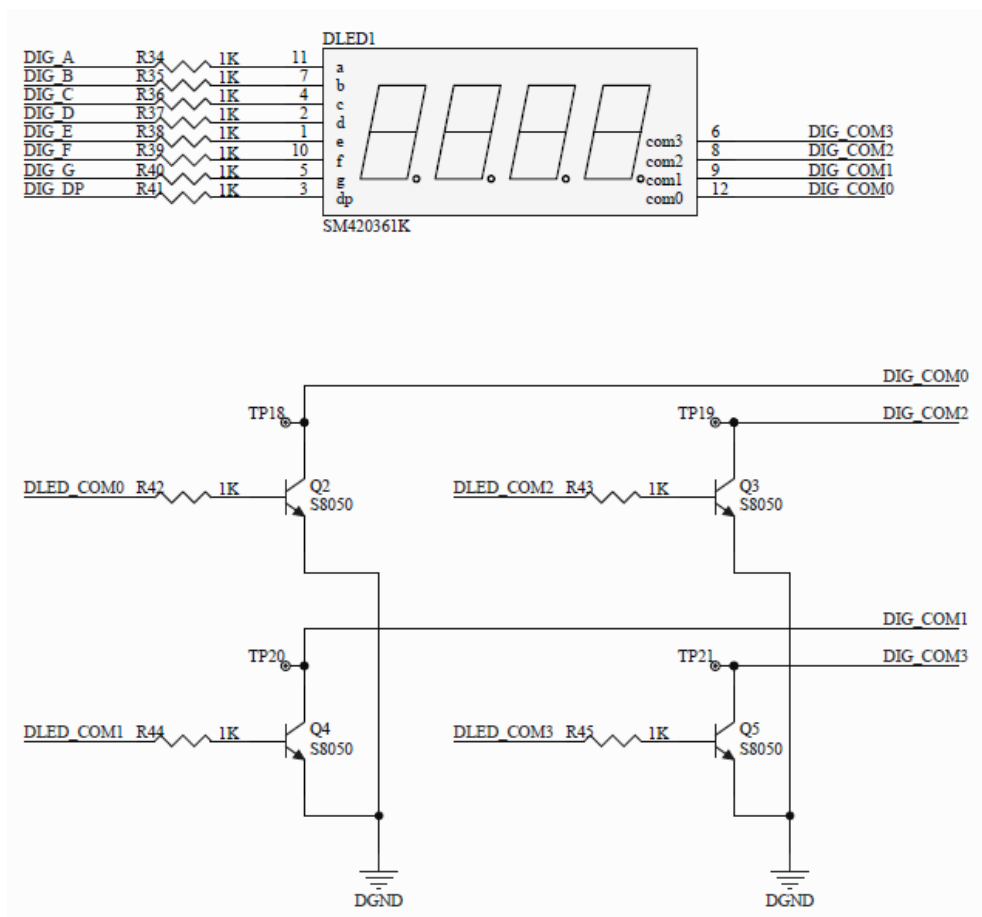
学习资料：例程 `Sample_Key&LED_100LG`，重点是数码管的控制和显示部分，数字 0-20 可以通过定时器或循环延时生成。

硬件设计

如下图，四位八段数码管有八个段选信号，和四个位选信号。

该数码管为共阴极数码管。位选信号选择点亮哪一个八段数码管，拉低点亮。位选信号通过三极管驱动，基极为高电平三极管导通，位选信号接地，对应数码管点亮。

段选信号对应数码管的八段，高电平点亮，不同亮灭组合控制显示不同的字符。这里段选信号直接由单片机驱动，没外加驱动，增加了单片机的供电压力，不是一个明智的硬件设计。



段选信号(P10-P17)和片选信号(P52-P55)都是驱动信号，都设置为输出。

Port0	Port1	Port2	Port3	Port4	Port5	Port6	Port7	Port12	Port13	Port14
P10										
<input type="radio"/> Unused	<input type="radio"/> In	<input checked="" type="radio"/> Out	<input type="checkbox"/> Pull-up	<input type="checkbox"/> TTL buffer	<input type="checkbox"/> N-ch	<input type="checkbox"/> 1				
P11										
<input type="radio"/> Unused	<input type="radio"/> In	<input checked="" type="radio"/> Out	<input type="checkbox"/> Pull-up	<input type="checkbox"/> TTL buffer	<input type="checkbox"/> N-ch	<input type="checkbox"/> 1				
P12										
<input type="radio"/> Unused	<input type="radio"/> In	<input checked="" type="radio"/> Out	<input type="checkbox"/> Pull-up		<input type="checkbox"/> N-ch	<input type="checkbox"/> 1				
P13										
<input type="radio"/> Unused	<input type="radio"/> In	<input checked="" type="radio"/> Out	<input type="checkbox"/> Pull-up	<input type="checkbox"/> TTL buffer	<input type="checkbox"/> N-ch	<input type="checkbox"/> 1				
P14										
<input type="radio"/> Unused	<input type="radio"/> In	<input checked="" type="radio"/> Out	<input type="checkbox"/> Pull-up	<input type="checkbox"/> TTL buffer	<input type="checkbox"/> N-ch	<input type="checkbox"/> 1				
P15										
<input type="radio"/> Unused	<input type="radio"/> In	<input checked="" type="radio"/> Out	<input type="checkbox"/> Pull-up	<input type="checkbox"/> TTL buffer	<input type="checkbox"/> N-ch	<input type="checkbox"/> 1				
P16										
<input type="radio"/> Unused	<input type="radio"/> In	<input checked="" type="radio"/> Out	<input type="checkbox"/> Pull-up	<input type="checkbox"/> TTL buffer		<input type="checkbox"/> 1				
P17										
<input type="radio"/> Unused	<input type="radio"/> In	<input checked="" type="radio"/> Out	<input type="checkbox"/> Pull-up	<input type="checkbox"/> TTL buffer	<input type="checkbox"/> N-ch	<input type="checkbox"/> 1				

Port0	Port1	Port2	Port3	Port4	Port5	Port6	Port7	Port12	Port13	Port14
P50										
<input checked="" type="radio"/> Unused	<input type="radio"/> In	<input type="radio"/> Out	<input type="checkbox"/> Pull-up		<input type="checkbox"/> N-ch	<input type="checkbox"/> 1				
P51										
<input checked="" type="radio"/> Unused	<input type="radio"/> In	<input type="radio"/> Out	<input type="checkbox"/> Pull-up		<input type="checkbox"/> 1					
P52										
<input type="radio"/> Unused	<input type="radio"/> In	<input checked="" type="radio"/> Out	<input type="checkbox"/> Pull-up		<input type="checkbox"/> 1					
P53										
<input type="radio"/> Unused	<input type="radio"/> In	<input checked="" type="radio"/> Out	<input type="checkbox"/> Pull-up		<input type="checkbox"/> 1					
P54										
<input type="radio"/> Unused	<input type="radio"/> In	<input checked="" type="radio"/> Out	<input type="checkbox"/> Pull-up		<input type="checkbox"/> 1					
P55										
<input type="radio"/> Unused	<input type="radio"/> In	<input checked="" type="radio"/> Out	<input type="checkbox"/> Pull-up	<input type="checkbox"/> TTL buffer	<input type="checkbox"/> N-ch	<input type="checkbox"/> 1				

软件设计

由硬件可知，不可能同时让四个数码管都显示不同的字符，所以只能通过扫描的方式依次让每一个数码管显示一个数字。利用 led 的余晖让数码管看似在同时显示。
利用定时器实现数码管的显示。每隔 4ms（250Hz）点亮一个数码管。

General setting	Channel 0	Channel 1	Channel 2	Channel 3	Channel 4	Channel 5	Channel 6	Channel 7
Interval timer setting								
Interval value (16 bits)		<input type="text" value="4"/>		ms		(Actual value: 4)		
<input type="checkbox"/> Generates INTTM00 when counting is started								
Interrupt setting								
<input checked="" type="checkbox"/> End of timer channel 0 count, generate an interrupt (INTTM00)								
Priority		Low						

定时器中断函数一次点亮一个数码管，代码如下：

```
//定时器0 为数码管刷新定时器
```

```
static char order = 0;
```

```
//先关闭数码管，准备好显示数据，再开相应的数码管
```

```

DIGITAL_LED_D1 = 0;
DIGITAL_LED_D2 = 0;
DIGITAL_LED_D3 = 0;
DIGITAL_LED_D4 = 0;

switch(order)
{
    case 0:
        DIGITAL_PORT = Getcode(digitalLEDBit[0]);
        DIGITAL_LED_D1 = 0;
        DIGITAL_LED_D2 = 0;
        DIGITAL_LED_D3 = 0;
        DIGITAL_LED_D4 = 1;
        break;

    case 1:
        DIGITAL_PORT = Getcode(digitalLEDBit[1]);
        DIGITAL_LED_D1 = 0;
        DIGITAL_LED_D2 = 0;
        DIGITAL_LED_D3 = 1;
        DIGITAL_LED_D4 = 0;
        break;

    case 2:
        DIGITAL_PORT = Getcode(digitalLEDBit[2]);
        DIGITAL_LED_D1 = 0;
        DIGITAL_LED_D2 = 1;
        DIGITAL_LED_D3 = 0;
        DIGITAL_LED_D4 = 0;
        break;

    case 3:
        DIGITAL_PORT = Getcode(digitalLEDBit[3]);
        DIGITAL_LED_D1 = 1;
        DIGITAL_LED_D2 = 0;
        DIGITAL_LED_D3 = 0;
        DIGITAL_LED_D4 = 0;
        break;
}

order = (order + 1) % 4;

```

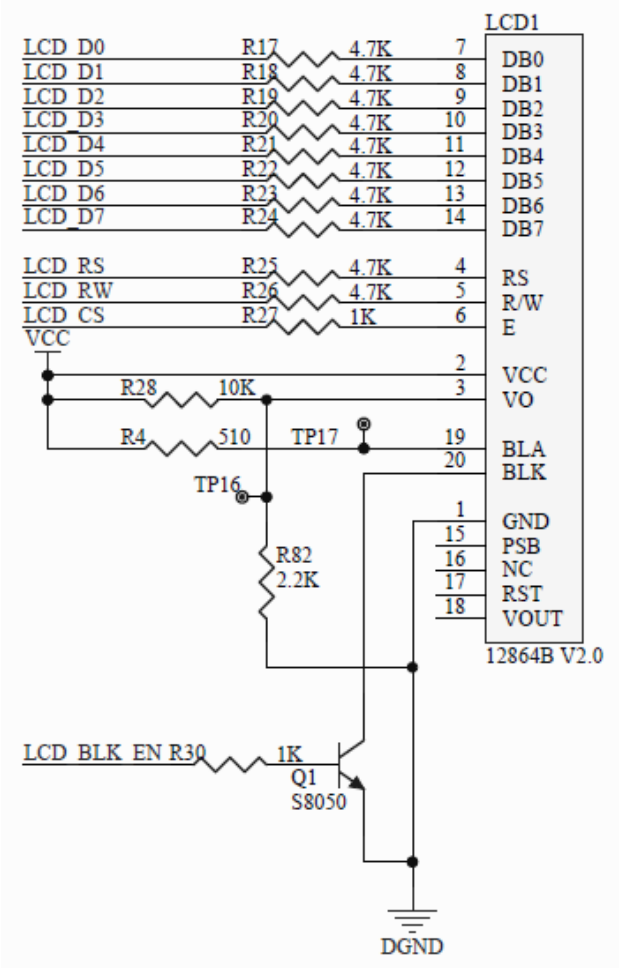
其中 `digitalLEDBit[]` 数组保存要显示的数字，`Getcode()` 函数返回对应数字的编码。
最后打开定时器即可。

4) 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。

学习资料：例程 Sample_LCD_100LG 和 RL78_FunctionDemoCodenew。

硬件设计



如上图，LCD 用到的引脚有：

- P2.0-P2.7，并行传输 8 位数据
- P14.6，为命令、数据选择引脚
- P14.7，为读、写选择引脚
- P14.1，为片选信号引脚
- P13.0，为背光电源引脚，通过三极管提供驱动。

以上所有引脚直接在代码生成器中设置为输出即可。这里不再赘述。该设置只是初始化的设置。

在 LCD 显示过程中有一个查询 LCD 是否忙的操作，需要临时更改数据端口 P2.0-P2.7 为输入，更改相应寄存器即可，如下：

```
#define DB2_PIN_OUT()    (PM2 = 0x00)
#define DB2_PIN_IN()    (PM2 = 0xff)
```

软件设计

已有提供一些 LCD 驱动函数，其中：

LCD 初始化函数主要进行一些初始化设置。

LCD 显示字符函数如下：

```
void lcd_display(unsigned char pos, unsigned char *str)
```

传入要显示的位置和字符串的首地址/指针即可。例如：

```
lcd_display(0, "display");
```

显示汉字的注意事项如下：

CGRAM 字型與中文字形之編碼只可出現在每一 Address counter 的起始位置(參考 Table 4)

80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
H	L	H	L	H	L	H	L	H	L	H	L	H	L	H	L
S	i	t	r	o	n	i	x		S	T	7	9	2	0	
矽	創	電	子	.	.	中	文	編	碼	(正	確)		
矽	創	電	子	.	.	.	中	文	編	碼					

Table 4

錯誤填入中文碼位置

没有现成的图片显示函数，下面简要说明如何显示图片。

GDRAM 介绍如下：

繪圖 RAM (GDRAM)

繪圖顯示 RAM 提供 64x32 個位元組的記憶空間(由擴充指令設定繪圖 RAM 位址)，最多可以控制 256x64 點的二維繪圖緩衝空間，在更改繪圖 RAM 時，由擴充指令設定 GDRAM 位址先設垂直位址再設水平位址(連續寫入兩個位元組的資料來完成垂直與水平的座標位址)，再寫入兩個 8 位元的資料到繪圖 RAM，而位址計數器 (AC) 會自動加一，整個寫入繪圖 RAM 的步驟如下：

1. 先將垂直的位元組座標 (Y) 寫入繪圖 RAM 位址。
2. 再將的水平座標 (X) 寫入繪圖 RAM 位址。
3. 將 D15~D8 寫入到 RAM 中(寫入第一個 Bytes)。
4. 將 D7~D0 寫入到 RAM 中(寫入第二個 Bytes)。

繪圖顯示的記憶體對應分佈請參考 Table-8。

根据 LCD 驱动芯片的数据手册，LCD 显示图片需要对 LCD 的 GDRAM/图像数据 RAM 进行读写，并让 GDRAM 的数据显示到屏幕上。

GDRAM 的写步骤为：

- 1 设置行地址
- 2 设置列地址
- 3 写数据高八位
- 4 写数据低八位

其中设置 GDRAM 的地址指令为扩充指令，需要把指令集从基本指令集更改为扩充指令集。

GDRAM 地址与物理屏幕的映射关系为：

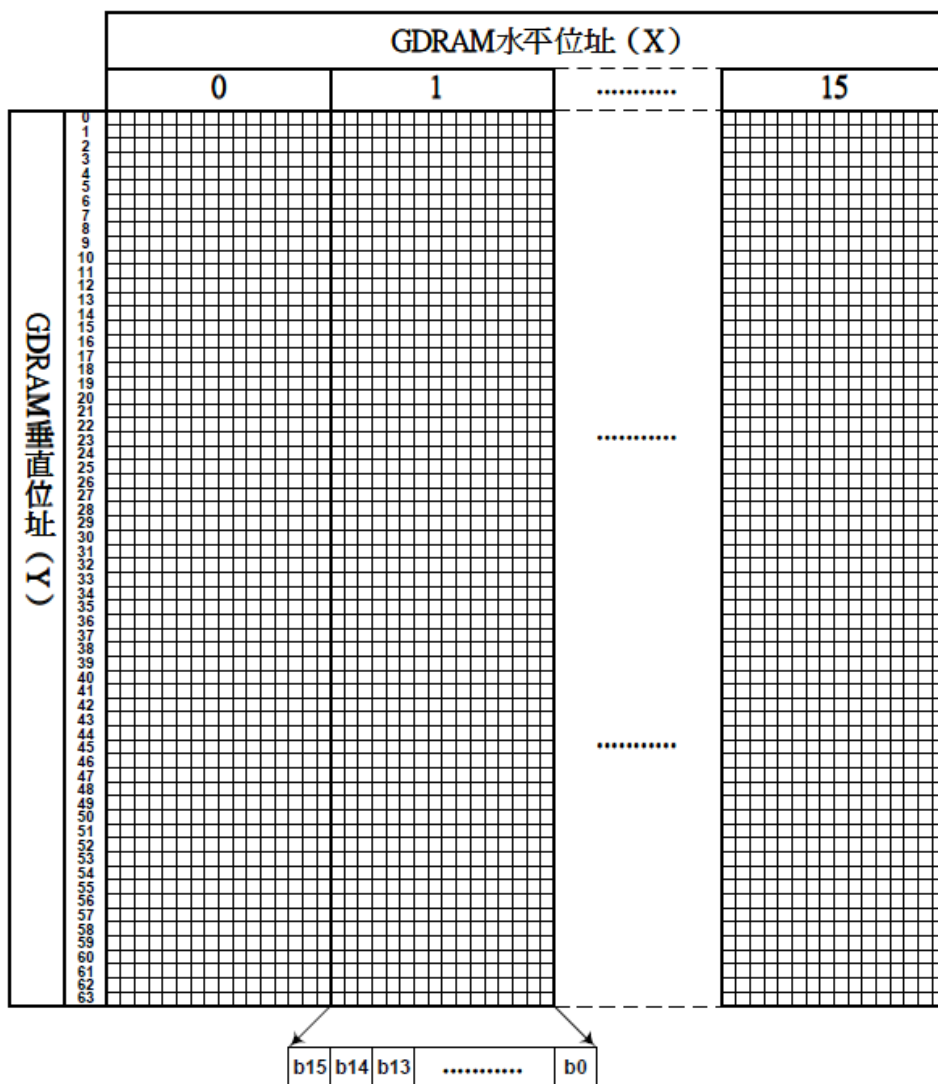


Table 8 GDRAM座標位址與資料排列順序對照表

设置 GDRAM 地址的具体指令内容为:

設定繪圖 RAM 位址	0	0	1	0	0	0	AC3	AC2	AC1	AC0	設定 GDRAM 位址到位址計數器 (AC) 先設垂直位址再設水平位址(連續寫入兩個位元組的資料來完成垂直與水平的座標位址) 垂直位址範圍 AC6...AC0 水平位址範圍 AC3...AC0	72 us
----------------	---	---	---	---	---	---	-----	-----	-----	-----	---	-------

所以显示一幅图片的具体操作为:

- 打开绘图显示
- 设置扩充指令集
- 依次把每个像素点的值写入 GDRAM(如前所述, 先写地址, 再写数据)
- 恢复基本指令集

具体函数如下, 传入要显示图片数组的首地址/指针即可。(这里把图片显示在大小为 64*64 的左半屏上, 右半屏填充空白)。

```
void LcdFill_Image(unsigned char * image)
{
```

```

unsigned char x, y, ii;
unsigned short cursor = 0;

#if DBUS_BITS == 4
    lcd_write(0x26, 0); //4 位并口, 扩充指令 绘图显示开
#else
    //8 位并口, 扩充指令 绘图显示开
    //先改变G, 再改变RE
    lcd_write(0x32, 0);
    lcd_write(0x36, 0);
#endif

//先画前两行, 再画后两行
for(ii = 0; ii < 9; ii += 8)
{
    //依次画0-31 行
    for(y = 0; y < 0x20; y++)
    {
        //依次画4*16(两字节)个像素点
        for(x = 0; x < 4; x++)
        {
            //在扩充指令集下设定 GDRAM 地址
            //行地址
            lcd_write(y+0x80, 0);
            //列地址
            lcd_write(x+0x80+ii, 0);

            //在扩充指令集下可直接对 GDRAM 进行写操作
            //写数据 D15 - D8
            lcd_write(image[cursor++], 1);
            //写数据 D7 - D0
            lcd_write(image[cursor++], 1);
        }

        //依次画4*16(两字节)个像素点
        for(x = 4; x < 8; x++)
        {
            lcd_write(y+0x80, 0);
            lcd_write(x+0x80+ii, 0);
            lcd_write(0x00, 1);
            lcd_write(0x00, 1);
        }
    }
}

#endif DBUS_BITS == 4

```

```

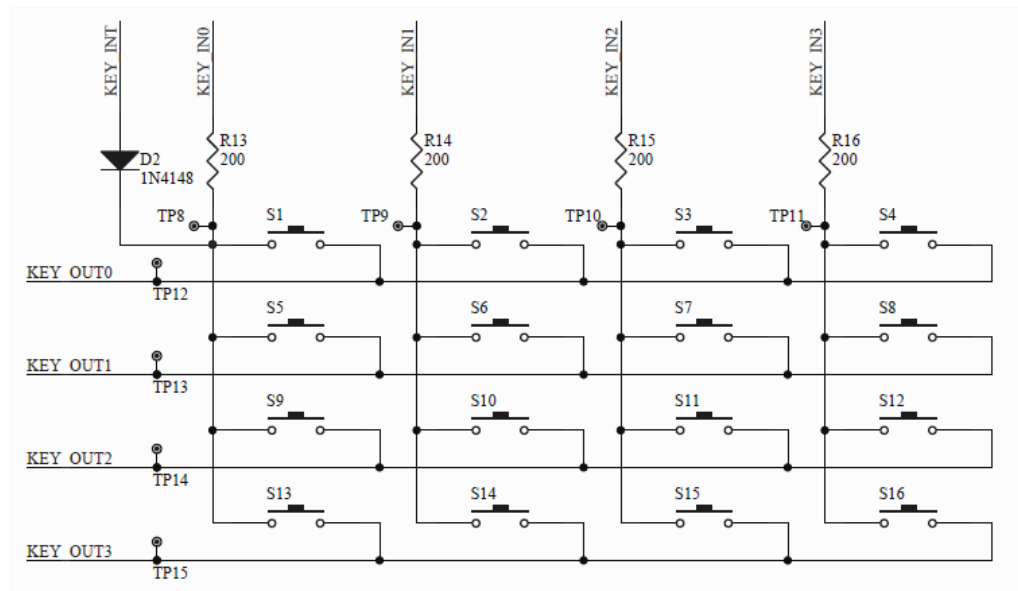
    lcd_write(0x20,0);
    #else
    //绘图开, 回到基本指令集
    //先改变G, 再改变RE
    lcd_write(0x36,0);
    lcd_write(0x32,0);
    #endif
}

```

5) 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 +、-、×、÷、= 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。

学习资料：例程 Sample_Key&LED_100LG 和 Sample_LCD_100LG，重点是键盘定义、单次扫描和液晶屏显示控制。

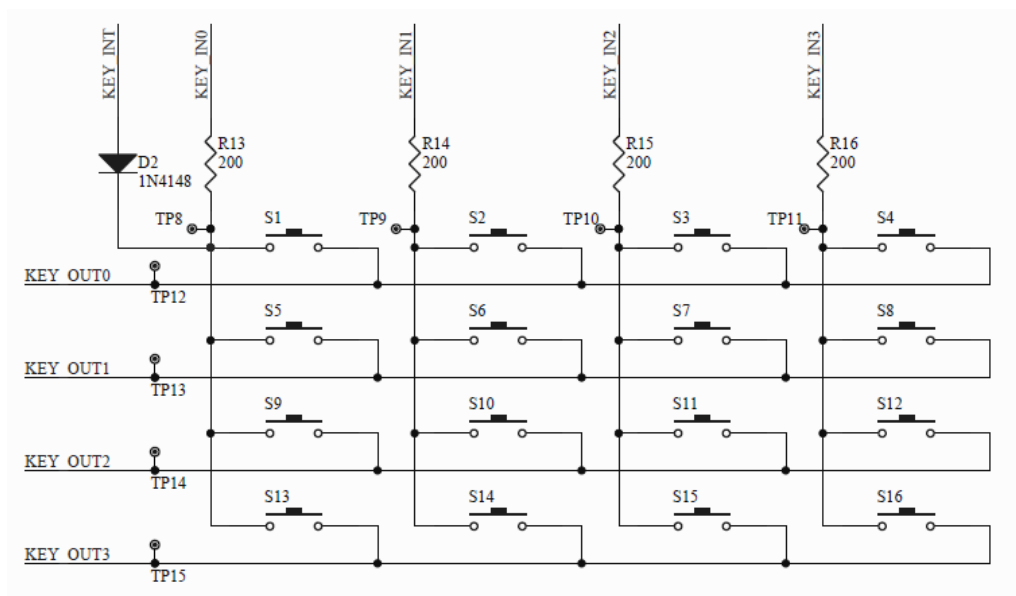
硬件设计



键盘电路图如上，其中需要把横向(P70-P73)设置为输出，纵向(P74-P77)设置为输入。这里硬件没有提供输入端口的上部上拉，所以需要设置单片机内部上拉，以提供稳定的输入端口状态，否则无法检测键盘。如下：

Port0	Port1	Port2	Port3	Port4	Port5	Port6	Port7	Port12	Port13	Port14
P70										
<input type="radio"/> Unused	<input type="radio"/> In	<input checked="" type="radio"/> Out	<input type="checkbox"/> Pull-up							<input type="checkbox"/> 1
P71										
<input type="radio"/> Unused	<input type="radio"/> In	<input checked="" type="radio"/> Out	<input type="checkbox"/> Pull-up					<input type="checkbox"/> N-ch		<input type="checkbox"/> 1
P72										
<input type="radio"/> Unused	<input type="radio"/> In	<input checked="" type="radio"/> Out	<input type="checkbox"/> Pull-up							<input type="checkbox"/> 1
P73										
<input type="radio"/> Unused	<input type="radio"/> In	<input checked="" type="radio"/> Out	<input type="checkbox"/> Pull-up							<input type="checkbox"/> 1
P74										
<input type="radio"/> Unused	<input checked="" type="radio"/> In	<input type="radio"/> Out	<input checked="" type="checkbox"/> Pull-up					<input type="checkbox"/> N-ch		<input type="checkbox"/> 1
P75										
<input type="radio"/> Unused	<input checked="" type="radio"/> In	<input type="radio"/> Out	<input checked="" type="checkbox"/> Pull-up							<input type="checkbox"/> 1
P76										
<input type="radio"/> Unused	<input checked="" type="radio"/> In	<input type="radio"/> Out	<input checked="" type="checkbox"/> Pull-up							<input type="checkbox"/> 1
P77										
<input type="radio"/> Unused	<input checked="" type="radio"/> In	<input type="radio"/> Out	<input checked="" type="checkbox"/> Pull-up							<input type="checkbox"/> 1

软件设计



再看一下电路图，这里说明如何扫描第一行的四个按键，其他行以此类推。

设置第一行输出低电平，其他三行输出高电平。此时第一行的 S1 按键若按下，则 IN0 被拉低，而 S5，S9，S13 若按下，IN0 均为高，不影响 S1 的检测。所以 IN0 为低对应 S1 被按下，IN1 为低对应 S2 被按下，依次类推。

下面为扫描第一行的程序：

```
//扫描第一行
KEY_PORT = 0xff;
P7.3 = 0;
keyStatus = KEY_PORT & 0xf0;

if(keyStatus != 0xf0)
{
```

```

//消除抖动
keyScanDelay(20);

keyStatus = KEY_PORT & 0xf0;
if(keyStatus != 0xf0)
{
    keyStatus = KEY_PORT & 0xf0;

    switch(keyStatus)
    {
        case 0xe0: keyboardNum = 1; break;
        case 0xd0: keyboardNum = 5; break;
        case 0xb0: keyboardNum = 9; break;
        case 0x70: keyboardNum = 13; break;
    }
}
}

```

在 lcd 上显示键盘按下的符号，只需把该符号添加到一个字符串中，然后在 lcd 中显示该字符串即可，程序如下：

```

char buffer[32];
char cursor = 0;

while(1)
{
    buffer[cursor++] = getInput();
    buffer[cursor] = '\0';

    lcd_display(0, buffer);
}

```

实验项目 3：综合设计实验（第 6、7、8 次实验完成，共 12 学时）

实验目的： 较全面掌握 RL78/G13 系列嵌入式微控制器的程序设计技术。

实验基本要求： 利用基础模块设计实验的积累，完成较为复杂的综合实验任务。

实验内容提要： 实现具有简单人机界面的加、减、乘、除计算器；分别采用定时中断和定时器，设计秒表和倒计时表；设计实现电子日历和时钟。要求：a) 用三个不同的工程分别实现上述功能；b) 建立一个工程，设计一个菜单，完成上述所有功能。

具体任务： 6) 制作简易计算器，在第 5 个实验基础上，实现具有简单人机界面的加、减、乘、除计算，并在 LCD 上显示输入内容及计算结果；

硬件设计

无需额外设计。

软件设计

利用状态机实现。

对于一个输入，状态机的响应包括对该输入的响应，状态转移。

这里只设计一级加减乘除运算。

总共设计了四个状态，分别为：

1. 输入第一个数的整数部分
2. 输入第一个数的小数部分
3. 输入第二个数的整数部分
4. 输入第二个数的小数部分

状态转移关系为：

输入'.': 1->2

输入'+','-', '*', '/': 1->3

输入'+','-', '*', '/': 2->3

输入'.': 3->4

输入 '=': 3->0

输入 '=': 4->0

程序如下：

//计算器部分，使用状态机实现

```
void dealWithNum(char num)
{
    //16 个有效数字+1 个'\0'
    static char bufferInput[17];
    static char bufferOutput[17];
    static char bufferInputIndex = 0;
    static char bufferOutputIndex = 0;

    static unsigned char status = 0;

    static float operand1 = 0, operand2 = 0;
    static float factor = 0.1;

    static char operatorNum = '+';

    float result = 0;
    int tempResult1, tempResult2;

    switch(status)
    {
        //输入第一个数的整数部分
        case 0:
            if((num >= '0') && (num <= '9'))
```

```

{
    //数字处理
    operand1 = operand1 * 10 + (num - '0');

    //display
    bufferInput[bufferInputIndex++] = num;
    bufferInput[bufferInputIndex] = '\0';
    //sprintf(bufferLine1, "%d", (short)operand1);
}
else if(num == '.')
{
    //小数点处理
    //设置好乘数因子
    factor = 0.1;
    //进入状态1
    status = 1;

    //display
    bufferInput[bufferInputIndex++] = num;
    bufferInput[bufferInputIndex] = '\0';
    //sprintf(bufferLine1, "%d.", (short)operand1);
}
else if((num == '+') || (num == '-') || (num == '*') || (num == '/'))
{
    //若没有输入数字,直接输入/,则退出
    if((operand1 == 0) &&(num == '/'))
    {
        exitFlag = 1;
    }
    //加减乘除处理
    //记录运算方式
    operatorNum = num;
    //进入状态2
    status = 2;

    //display
    bufferInput[bufferInputIndex++] = num;
    bufferInput[bufferInputIndex] = '\0';
    //sprintf(bufferLine1, "%d%c", (short)operand1, operatorChar[operatorNum - 11]);
}
else
{
    //等号处理
    //默认不处理
}
}

```

```
display(2, "          ");
display(2, bufferInput);
display(3, "          ");
//display(3, bufferOutput);
```

```
break;
```

```
//输入第一个数的小数部分，如果有的话
```

```
case 1:
```

```
if((num >= '0') && (num <= '9'))
{
```

```
    //数字处理
```

```
    operand1 += (num - '0') * factor;
```

```
    factor *= 0.1;
```

```
    //display
```

```
    bufferInput[bufferInputIndex++] = num;
```

```
    bufferInput[bufferInputIndex] = '\0';
```

```
    //sprintf(bufferLine1, "%f", operand1);
```

```
}
```

```
else if(num == '.')
```

```
{
```

```
    //小数点处理
```

```
    //出错，默认不处理
```

```
}
```

```
else if((num == '+') || (num == '-') || (num == '*') || (num == '/'))
```

```
{
```

```
    //加减乘除处理
```

```
    //记录运算方式
```

```
    operatorNum = num;
```

```
    //进入状态2
```

```
    status = 2;
```

```
    //display
```

```
    bufferInput[bufferInputIndex++] = num;
```

```
    bufferInput[bufferInputIndex] = '\0';
```

```
    //sprintf(bufferLine1, "%d%c", (short)operand1, operatorChar[operatorNum - 11]);
```

```
}
```

```
else
```

```
{
```

```
    //等号处理
```

```
    //默认不处理
```

```
}
```

```
display(2, "          ");
```

```
display(2, bufferInput);
```



```

        display(3, "          ");
        //display(3, bufferOutput);

        break;

//输入第二个数的整数部分
case 2:
    if((num >= '0') && (num <= '9'))
    {
        //数字处理
        operand2 = operand2 * 10 + (num - '0');

        //display
        bufferInput[bufferInputIndex++] = num;
        bufferInput[bufferInputIndex] = '\0';
        //sprintf(bufferLine1, "%f%c%d", operand1, operatorChar[operatorNum],
(short)operand2);
    }
    else if(num == '.')
    {
        //小数点处理
        //设置好乘数因子
        factor = 0.1;
        //进入状态3
        status = 3;

        //display
        bufferInput[bufferInputIndex++] = num;
        bufferInput[bufferInputIndex] = '\0';
        //sprintf(bufferLine1, "%f%c%d.", operand1, operatorChar[operatorNum],
(short)operand2);

    }
    else if((num == '+') || (num == '-') || (num == '*') || (num == '/'))
    {
        //加减乘除处理
        //出错, 这里默认不处理(默认只执行一次运算)
    }
    else if(num == '=')
    {
        //等号处理
        //计算结果
        switch(operatorNum)
        {
            case '+':
                // +

```

```

        result = operand1 + operand2;
        break;

    case '-':
        // -
        result = operand1 - operand2;
        break;

    case '*':
        // *
        result = operand1 * operand2;
        break;

    case '/':
        // /
        //默认不处理operand2=0的情况
        if(operand2 != 0)
        {
            result = operand1 / operand2;
        }
        break;
}

//该函数对浮点数格式化出现问题
//sprintf(bufferOutput, "%f", result);
tempResult1 = result;
tempResult2 = (result - tempResult1) * 10000;
sprintf(bufferOutput, "%d.%04d", tempResult1, tempResult2);

//复位
status = 0;
operand1 = 0;
operand2 = 0;
bufferInputIndex = 0;
}

display(2, "          ");
display(2, bufferInput);
display(3, "          ");
//最后一步才显示结果
if(status == 0)
{
    display(3, bufferOutput);
}

break;

```

//输入第二个数的小数部分, 如果有的话

case 3:

```
if((num >= '0') && (num <= '9'))
```

```
{
```

```
    //数字处理
```

```
    operand2 += (num - '0') * factor;
```

```
    factor *= 0.1;
```

```
    bufferInput[bufferInputIndex++] = num;
```

```
    bufferInput[bufferInputIndex] = '\0';
```

```
    //sprintf(bufferLine1, "%f%c%f", operand1, operatorChar[operatorNum], operand2);
```

```
}
```

```
else if(num == '.')
```

```
{
```

```
    //小数点处理
```

```
    //出错, 默认不处理
```

```
}
```

```
else if((num == '+') || (num == '-') || (num == '*') || (num == '/'))
```

```
{
```

```
    //加减乘除处理
```

```
    //出错, 这里默认不处理(默认只执行一次运算)
```

```
}
```

```
else if(num == '=')
```

```
{
```

```
    //等号处理
```

```
    //计算结果
```

```
    switch(operatorNum)
```

```
    {
```

```
        case '+':
```

```
            // +
```

```
            result = operand1 + operand2;
```

```
            break;
```

```
        case '-':
```

```
            // -
```

```
            result = operand1 - operand2;
```

```
            break;
```

```
        case '*':
```

```
            // *
```

```
            result = operand1 * operand2;
```

```
            break;
```

```
        case '/':
```

```
            // /
```

```

        //默认不处理operand2=0的情况
        if(operand2 != 0)
        {
            result = operand1 / operand2;
        }
        break;
    }

    //该函数对浮点数格式化出现问题
    //sprintf(bufferOutput, "%f", result);
    tempResult1 = result;
    tempResult2 = (result - tempResult1) * 10000;
    sprintf(bufferOutput, "%d.%04d", tempResult1, tempResult2);

    //复位
    status = 0;
    operand1 = 0;
    operand2 = 0;
    bufferInputIndex = 0;

}

display(2, "          ");
display(2, bufferInput);
display(3, "          ");
//最后一步才显示结果
if(status == 0)
{
    display(3, bufferOutput);
}

break;
}
}

```

7) 制作秒表和倒计时表，通过按键操作、通过 LCD 显示，倒计时结束时蜂鸣器发出提示音；要求：通过键盘实现“启动”、“停止”、“数值设定”等功能，在液晶屏上有“2-4 位数字显示”。

硬件设计

无需额外设计。

软件设计

秒表：利用一个定时器实现，定时间隔为 10ms，打开定时器后，时间依次累加即可。

中断函数如下：

```
__interrupt static void r_tau0_channel4_interrupt(void)
{
    tenMS1 += 2;
    if(tenMS1 == 100)
    {
        tenMS1 = 0;
        second1++;
        if(second1 == 60)
        {
            second1 = 0;
            minute1++;
        }
    }

    sprintf(buffer1, "---%02d:%02d:%02d---", minute1, second1, tenMS1);
    display(3, buffer1);
}
```

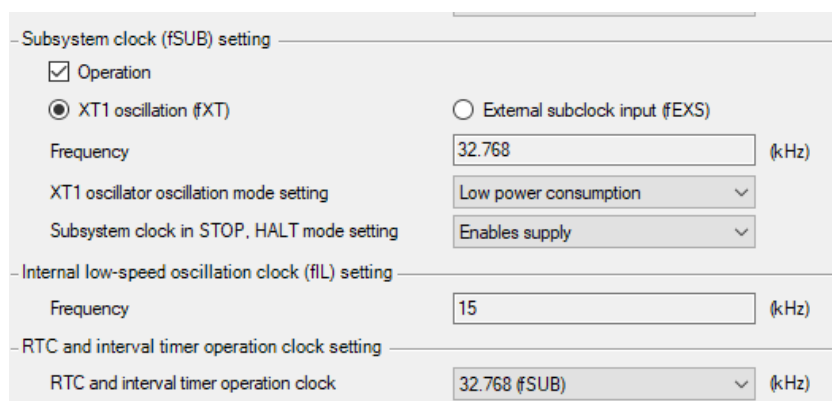
倒计时设置好时间后，打开定时器，时间依次递减，与倒计时类似，不再赘述。

8) 制作电子日历和时钟，在液晶屏上显示当前日期年月日、在液晶屏（或数码管）上显示当前时钟，能够通过键盘设定年月日和时间，数值能够自动正确累加。

硬件设计

使用单片机的 RTC 外设。

要使用 RTC 的时间日期累加功能，需要使用子系统时钟(32.768kHz)，需要启动该时钟，并且选择 RTC 的时钟源为该时钟。



The screenshot shows the 'Subsystem clock (fSUB) setting' section of the STM32CubeMX configuration tool. It includes the following settings:

- Operation:** ☒ Operation
- XT1 oscillation (fXT):** ☒ XT1 oscillation (fXT)
- External subclock input (fEXS):** ☐ External subclock input (fEXS)
- Frequency:** 32.768 (kHz)
- XT1 oscillator oscillation mode setting:** Low power consumption
- Subsystem clock in STOP, HALT mode setting:** Enables supply

Below this, the 'Internal low-speed oscillation clock (fIL) setting' is shown with a frequency of 15 (kHz).

At the bottom, the 'RTC and interval timer operation clock setting' is shown with the clock source set to '32.768 (fSUB)' (kHz).

设置时间，使能周期为 1s 的中断（在该中断中刷新 lcd 显示，这里忽略闹钟的使用）如下：


Real-time clock operation setting

☐ Unused ☒ Used

Real-time clock setting

Hour-system selection: 24-hour

☒ Set real-time clock initial value: 17-09-28 00:00:00 (星期四)

☐ Enable output of RTC1HZ pin (1 Hz) 

Alarm detection function setting

☒ Use alarm detection function

☒ Set alarm initial value

Week day

<input type="checkbox"/> Sunday	<input checked="" type="checkbox"/> Monday	<input checked="" type="checkbox"/> Tuesday	<input checked="" type="checkbox"/> Wednesday
<input checked="" type="checkbox"/> Thursday	<input checked="" type="checkbox"/> Friday	<input type="checkbox"/> Saturday	

Hour:Minute: 00:00

Interrupt setting

☒ Used as constant-period interrupt function (INTRTC): Once per 1 s

☒ Used as alarm interrupt function (INTRTC):

Priority: Low

软件设计

这里介绍几个使用到的软件提供的函数。

RTC 模块初始化

```
void R_RTC_Create(void);
```

开始计时

```
void R_RTC_Start(void);
```

获取时间日期寄存器的值

```
MD_STATUS R_RTC_Get_CounterValue(rtc_counter_value_t * const counter_read_val);
```

该函数传入一个结构体指针，结果存储在该结构体中，结构体定义如下：

```
typedef struct
{
    uint8_t sec;
    uint8_t min;
    uint8_t hour;
    uint8_t day;
    uint8_t week;
    uint8_t month;
    uint8_t year;
} rtc_counter_value_t;
```

设置时间日期寄存器的值

```
MD_STATUS R_RTC_Set_CounterValue(rtc_counter_value_t counter_write_val);
```

传入一个结构体，结构体为要设置的各个日期时间寄存器的值。

打开周期中断

```
MD_STATUS R_RTC_Set_ConstPeriodInterruptOn(rtc_int_period_t period);
```

输入为一个枚举类型的变量，这里选择一秒。

```
typedef enum
```

```

{
    HALFSEC = 1U,
    ONESEC,
    ONEMIN,
    ONEHOUR,
    ONEDAY,
    ONEMONTH
}rtc_int_period_t;

```

在周期中断中读取寄存器的值，并且显示到 lcd 屏上，如下：

```

static void r_rtc_callback_constperiod(void)
{
    /* Start user code. Do not edit comment generated here */

    R_RTC_Get_CounterValue(&rtcCounter);

    //显示方式
    //"09-28 18:30:30"
    //BCD 码转换
    sprintf(rtcBuffer, "%02d-%02d %02d:%02d:%02d",
        10 * (rtcCounter.month >> 4) + (rtcCounter.month & 0x0f),
        10 * (rtcCounter.day >> 4) + (rtcCounter.day & 0x0f),
        10 * (rtcCounter.hour >> 4) + (rtcCounter.hour & 0x0f),
        10 * (rtcCounter.min >> 4) + (rtcCounter.min & 0x0f),
        10 * (rtcCounter.sec >> 4) + (rtcCounter.sec & 0x0f)
    );

    display(0, rtcBuffer);

    /* End user code. Do not edit comment generated here */
}

```