目录

目录 1

1 NBK 系列开发板简介 3

1.1 NBK 系列开发板概述 3

1.2 NBK1220 开发板硬件资源介绍 3

1.3 NBK-EBS001 触控扩展板硬件资源介绍 4

1.4 NBK-EBS002 基础扩展板硬件资源介绍 5

1.5 NBK-EBS003 IOT 扩展板硬件资源介绍 6

2 平台介绍 7

2.1 开发平台 KEIL C 7

2.2 烧录仿真工具 SC-LINK PRO 7

2.3 PC 端烧录软件 SOC Programming Tool 7

2.4 ISP 烧录工具 7

3 NBK 系列开发板电路图 10

3.1 NBK1220 开发板原理图 10

3.2 NBK-EBS001 开发板原理图 11

3.3 NBK-EBS002 开发板原理图 11

3.4 NBK-EBS003 开发板原理图 12

4 程序示例 13

4.1 硬件 LED 驱动数码管显示示例 13

4.1.1 总体描述 13

4.1.2 LED 驱动功能 13

4.1.3 在易码魔盒中的配置 13

4.1.4 函数接口及变量 18

4.1.5 程序流程图 19

4.1.6 LED 波形 19

4.2 软件 GPIO 实现数码管显示示例 20

4.2.1 总体描述 20

4.2.2 IO 主要功能 20

4.2.3 在易码魔盒中的配置 21
4.2.4 函数接口及变量 23
4.2.5 程序流程图 24

4.3 硬件 LCD 驱动 4 位 LCD 屏显示示例 25
4.3.1 总体描述 25
4.3.2 LCD 驱动功能 25
4.3.3 在易码魔盒中的配置 25
4.3.4 函数接口及变量 28
4.3.5 程序流程图 29
4.3.6 1/3bias LCD 波形 30

4.4 ADC 测量热敏电阻示例 31
4.4.1 总体描述 31
4.4.2 ADC 转换步骤 31
4.4.3 在易码魔盒中的配置 32
4.4.4 函数接口及变量 36
4.4.5 程序流程图 37

4.5 PWM 实现 RGB 灯珠呼吸灯示例 38
4.5.1 总体描述 38
4.5.2 PWM0 主要功能 38
4.5.3 PWM 结构框图 38
4.5.4 在易码魔盒中的配置 39
4.5.5 函数接口及变量 41
4.5.6 程序流程图 42
4.5.7 PWM 波形 43

4.6 硬件 TWI 驱动 OLED 显示示例 44
4.6.1 总体描述 44
4.6.2 OLED 驱动原理 44
4.6.3 TWI 主机操作步骤 45
4.6.4 在魔盒中的配置 45
4.6.5 函数接口及变量 48
4.6.6 程序流程图 48

4.7 定时器捕获和外部中断测周期脉宽示例 49
4.7.1 总体描述 49
4.7.2 定时器 2 主要功能 49
4.7.3 在易码魔盒中的配置 50
4.7.4 函数接口及变量 53
4.7.5 程序流程图 54

4.8 软件驱动 1/2bias LCD 屏示例 55
4.8.1 总体描述 55
4.8.2 外接电阻驱动原理图 55
4.8.3 在易码魔盒中的配置 56
4.8.4 函数接口及变量 59
4.8.5 程序流程图 60
4.8.6 1/2Bias LCD 波形 61

1 NBK 系列开发板简介

1.1 NBK 系列开发板概述

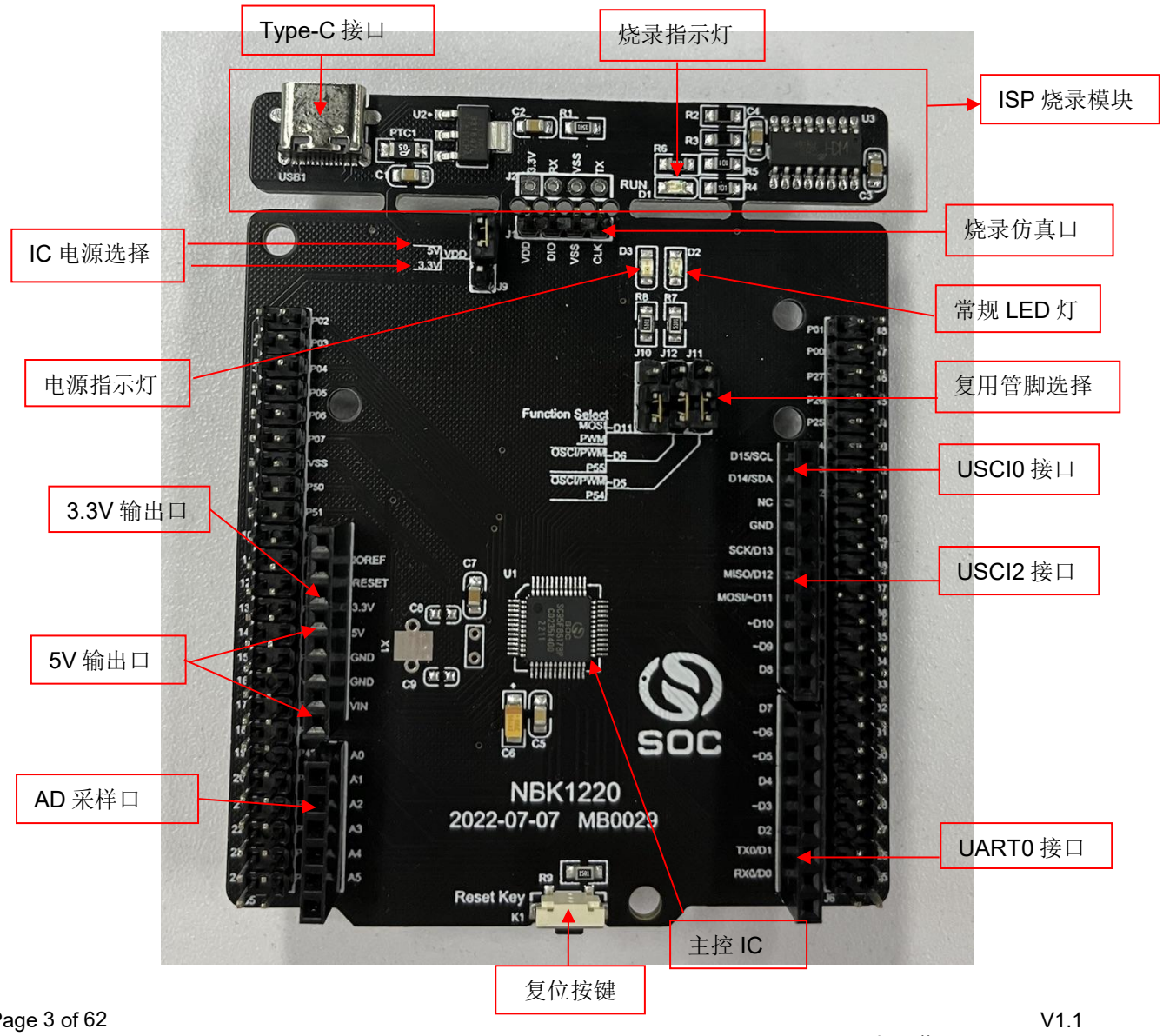
赛元 NBK 系列开发板，由核心开发板和外部扩展板组成。本开发板配备常用的单片机外围资源，自带调试下载接口，配合开发板提供的示例程序，可以让用户在最短的时间，熟悉并掌握赛元 MCU 相关的编程方法。本开发板非常适合初步接触赛元 MCU 的用户自学使用。

开发板功能如下表所示：

开发板	描述
NBK1220 核心板	**集成主控芯片和 ISP 烧录模块，引出所有引脚，并兼容 Arduino 接口**
NBK-EBS001 触控扩展板	**触控按键、滑条和滑轮三合一功能演示**
NBK-EBS002 基础功能扩展板	**扩展使用 PWM、LED、ADC、ACMP 等功能的器件，适合初学者学习**
NBK-EBS003 IOT 扩展板	**IOT 功能演示，可通过无线 bootload 升级芯片程序**

1.2 NBK1220 开发板硬件资源介绍

1) NBK1220 核心开发板资源简图。



NBK1220 核心板板载资源如下：

- CPU：SC95F8617B，工作电压为 2.0V~5.0V，ROM 为 64KB，RAM 为 4KB，系统时钟可选 32/16/4MHz
- 一组烧录仿真引脚
- 一个 Type-C 接口
- 一个复位按键
- 一个 LED 灯，光
- 一个电源指示灯，红光
- 一个 SPX1117-3.3 芯片，提供 3.3V 的稳压电源
- 一组 5V 电源供应口
- 一组 3.3V 电源供应口

1.3 NBK-EBS001 触控扩展板硬件资源介绍

2) NBK-EBS001 触控模块扩展板简图

NBK-EBS001 触控模块扩展板不能单独使用，需要配合开发板主板使用。



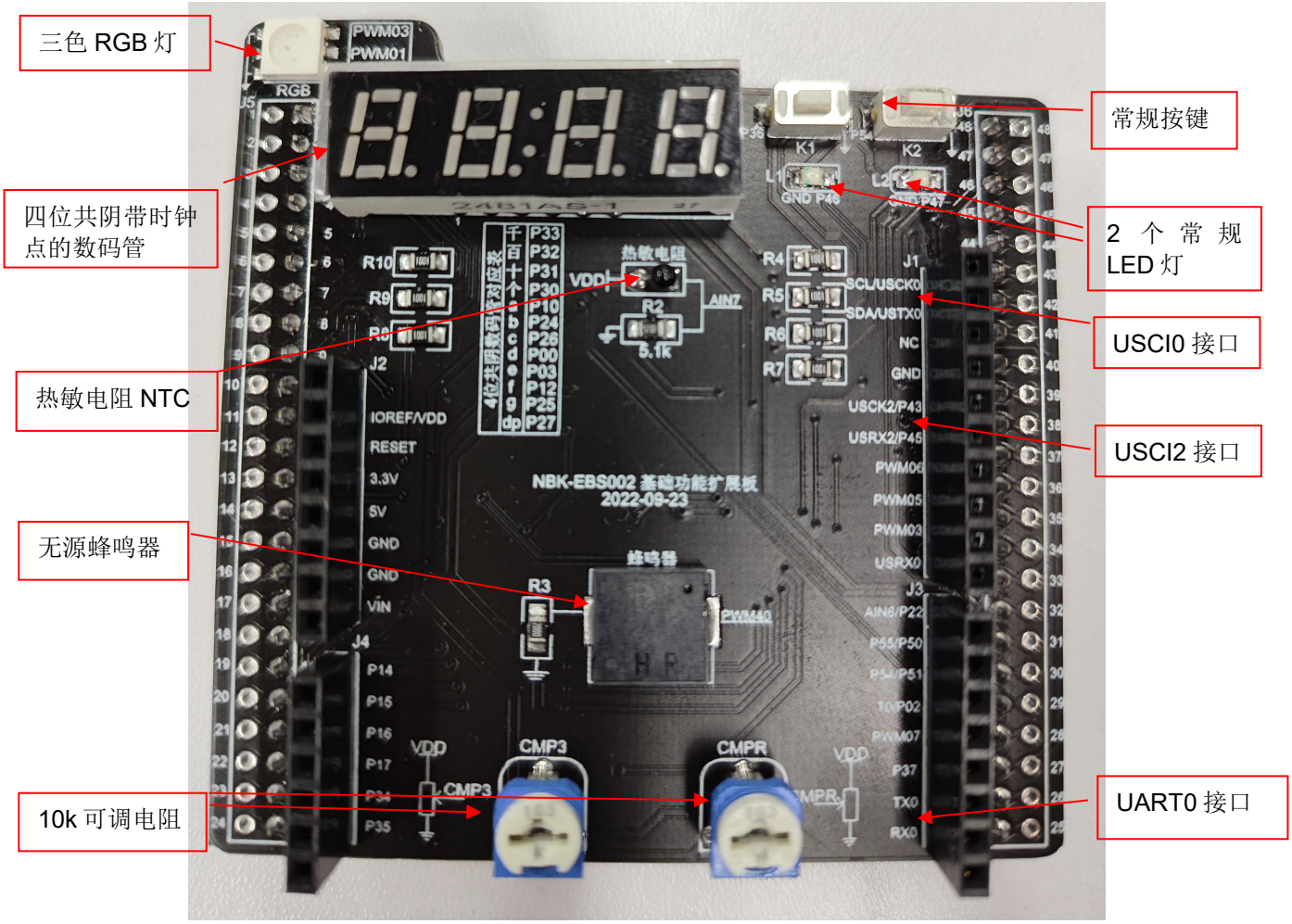
NBK-EBS001 触控板板载资源如下：

- 一个触控按键
- 一个触控滑条
- 一个触控滑轮
- 二十个 LED 灯，一个滑条有八个指示灯，一个滑轮有十二个指示灯。

1.4 NBK-EBS002 基础扩展板硬件资源介绍

3) NBK-EBS002 基础扩展板简图

NBK-EBS002 基础扩展板不能单独使用，需要配合开发板主板使用。



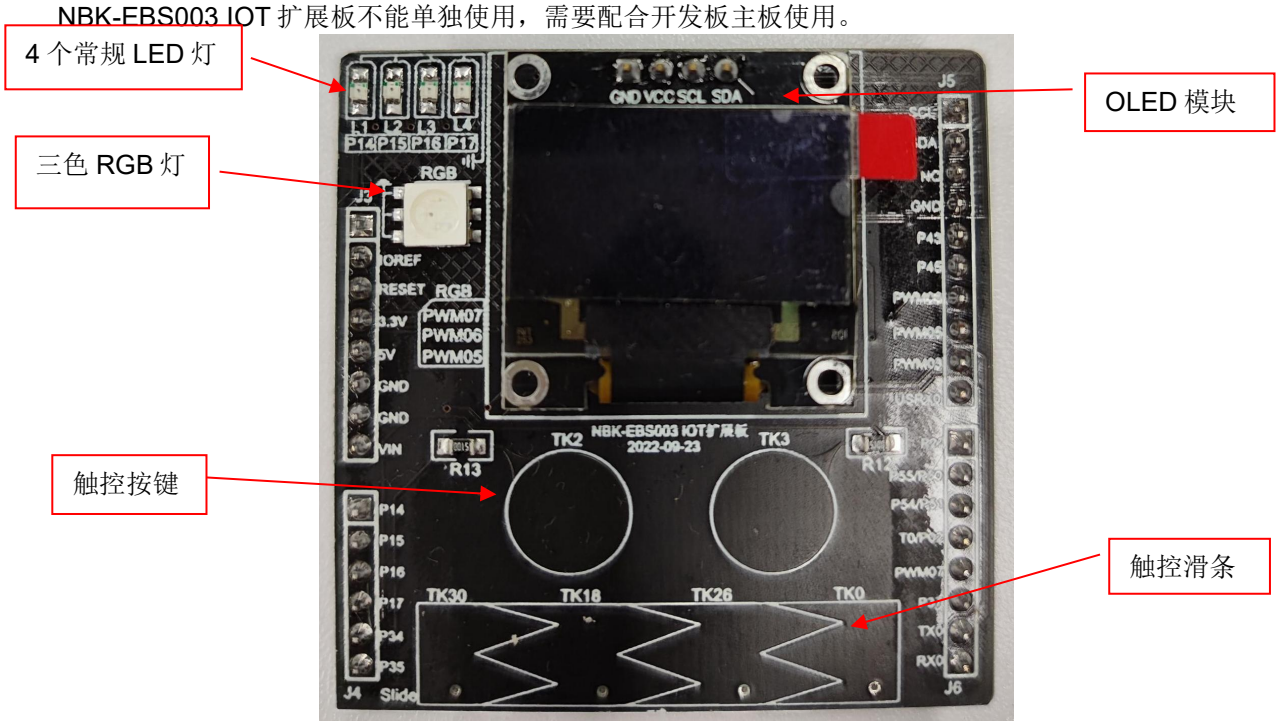
NBK-EBS002 基础扩展板板载资源如下：

- 一个三色 RGB 灯
- 一个四位共阴带时钟点的数码管
- 一个热敏电阻
- 两个常规按键
- 两个常规 LED 灯
- 一个无源蜂鸣器
- 两个 10k 可调电阻，可用于比较器和 ADC 采样。

1.5 NBK-EBS003 IOT 扩展板硬件资源介绍

4) NBK-EBS003 IOT 扩展板简图

NBK-EBS003 IOT 扩展板不能单独使用，需要配合开发板主板使用。



NBK-EBS003 IOT 扩展板板载资源如下：

- 四个常规 LED 灯
- 一个三色 RGB 灯
- 一个 TWI 通信 OLED 模块
- 两个触控按键
- 一个触控滑条
- 背部一个带 AT 固件的 EPS-12F wifi 模块。

2 平台介绍

2.1 开发平台 KEIL C

赛元 MCU，采用 Keil C 即 KEIL uVISION 平台来开发，支持汇编语言以及 C 语言编写。

KEIL uVISION，是众多单片机应用开发软件中最优秀的软件之一，它支持众多不同公司的 MCS51 架构的芯片甚至 ARM，它集编辑，编译，仿真等于一体，界面与常用的微软 VC++界面相似，界面简洁，易学易用，在调试程序，软件仿真方面也有很强大的功能。

有关 KEIL C 的使用，请参考赛元官网资料“赛元 LINK 系列量产开发工具使用手册”文档的第 5 章，有 Keil C 的安装及新建工程等使用说明。

2.2 烧录仿真工具 SC-LINK PRO

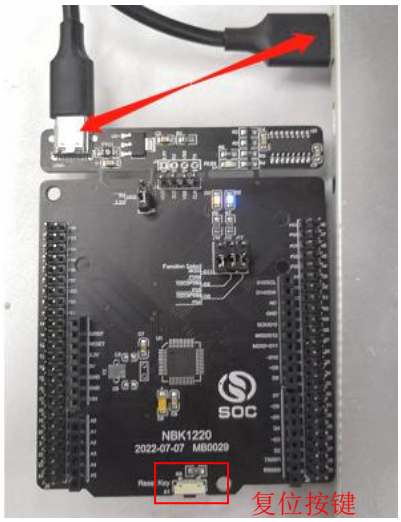
赛元目前使用的烧录工具为 SC-LINK PRO。烧录工具使用前请安装赛元仿真插件。SC-LINK PRO 适用于赛元 92F/93/95F 系列 IC 的脱机烧录、在线烧写、仿真以及 92F/93F/95F 系列触控 IC 的 Touch 调试。有关赛元烧录仿真工具的使用与仿真插件的安装，请参考赛元官网资料“赛元 LINK 系列量产开发工具使用手册”文档的第 2 章。

2.3 PC 端烧录软件 SOC PROGRAMMING TOOL

SOC Programming Tool 是赛元自主开发的全功能烧录软件，配合 SC LINK PRO 使用，支持编程、校验、查空、查看存储中的数据。关于 SOC PROGRAMMING TOOL 的安装步骤与使用说明请参考赛元官网资料“赛元 LINK 系列量产开发工具使用手册”文档的第 4 章。

2.4 ISP 烧录工具

1. 软硬件准备
   硬件：赛元 NBK1220 开发板，带 Type-C 公头的数据连接线
   软件：赛元 ISPTOOL，CH340 驱动
2. 使用 Type-C 数据线连接到 PC 上



3. 打开 ISPTOOL，点击“检测串口”，然后选择相应的端口号

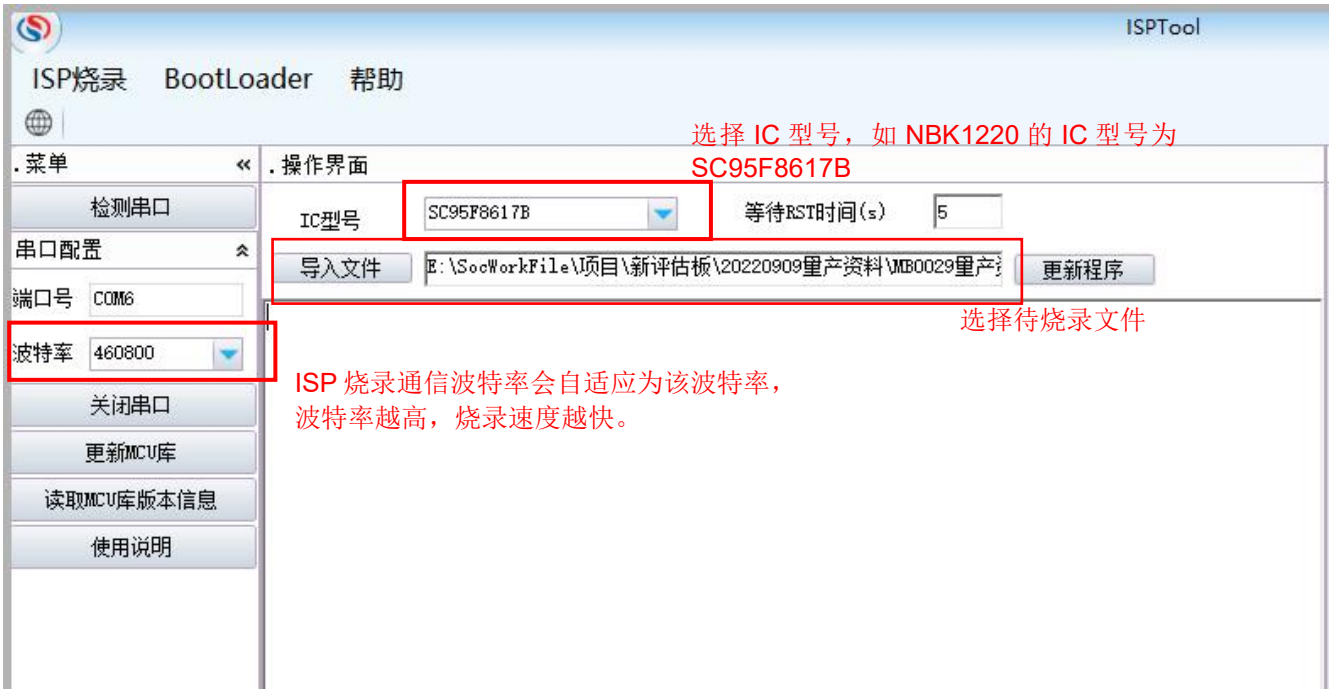


注：如果端口号显示为空，则是无法搜索到端口号，有以下两种情况导致：

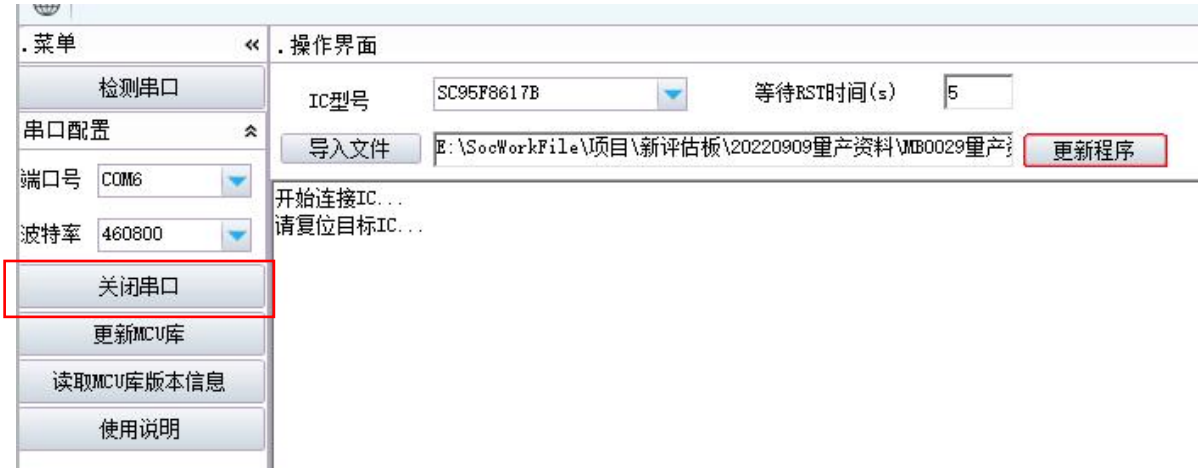
（1） 打开 windows 设备浏览器，未发现新增设备，可能是 Type-C 数据线连接不稳或数据线出现损坏，请重新插拔或更换数据线；

（2） 可以发现新增未知设备，则是 CH340 未能成功安装。

4. 然后选择 IC 型号、配置波特率和选择待烧录文件。



5. 打开串口，点击“更新程序”，此时信息框提示需要复位目标。参考步骤 2 的开发板图，开发板的按键就是复位按键，按下以进行芯片复位。

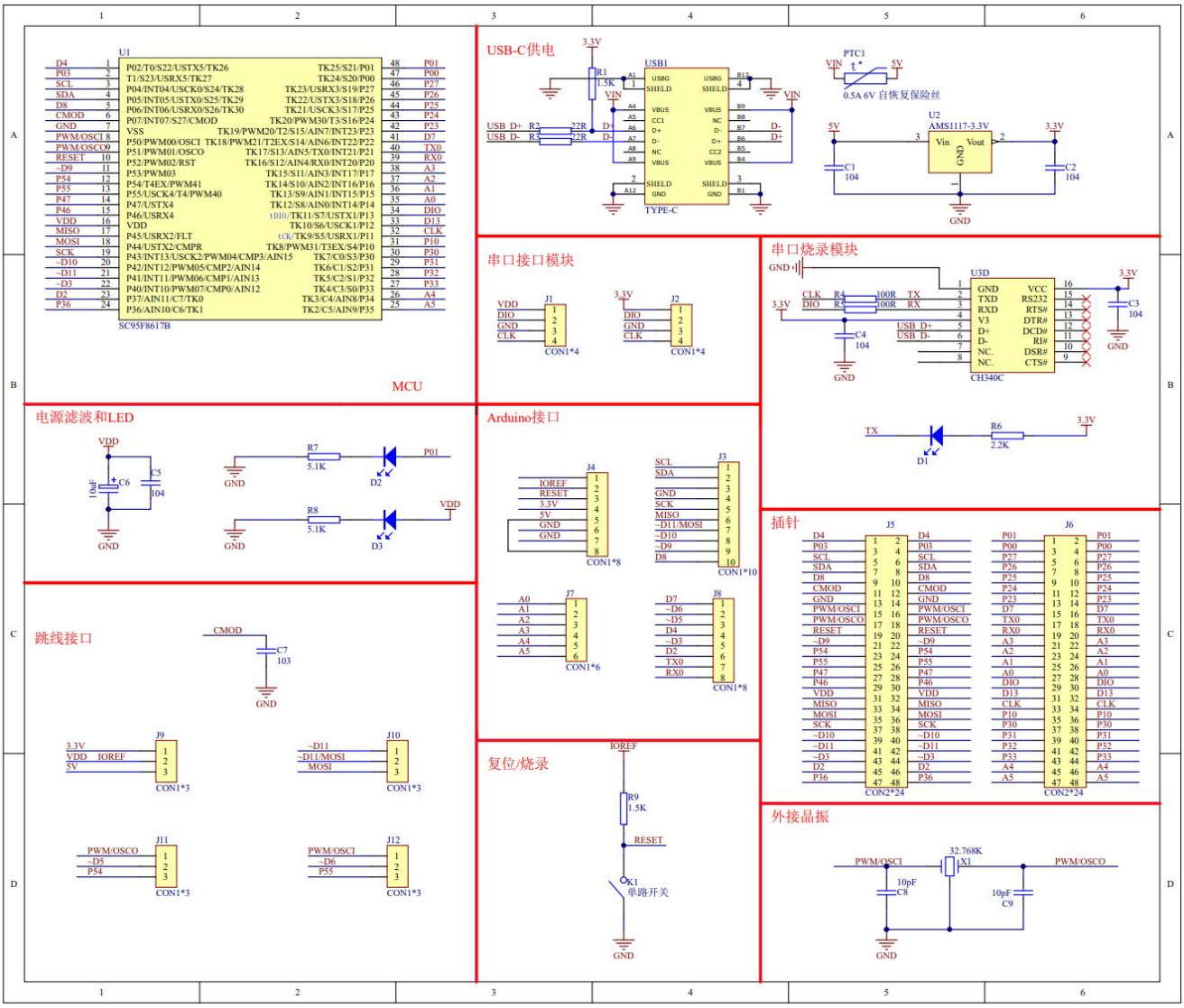


6. 等待烧录，烧录完成后界面如下；如果烧录失败，可以重试步骤 5.

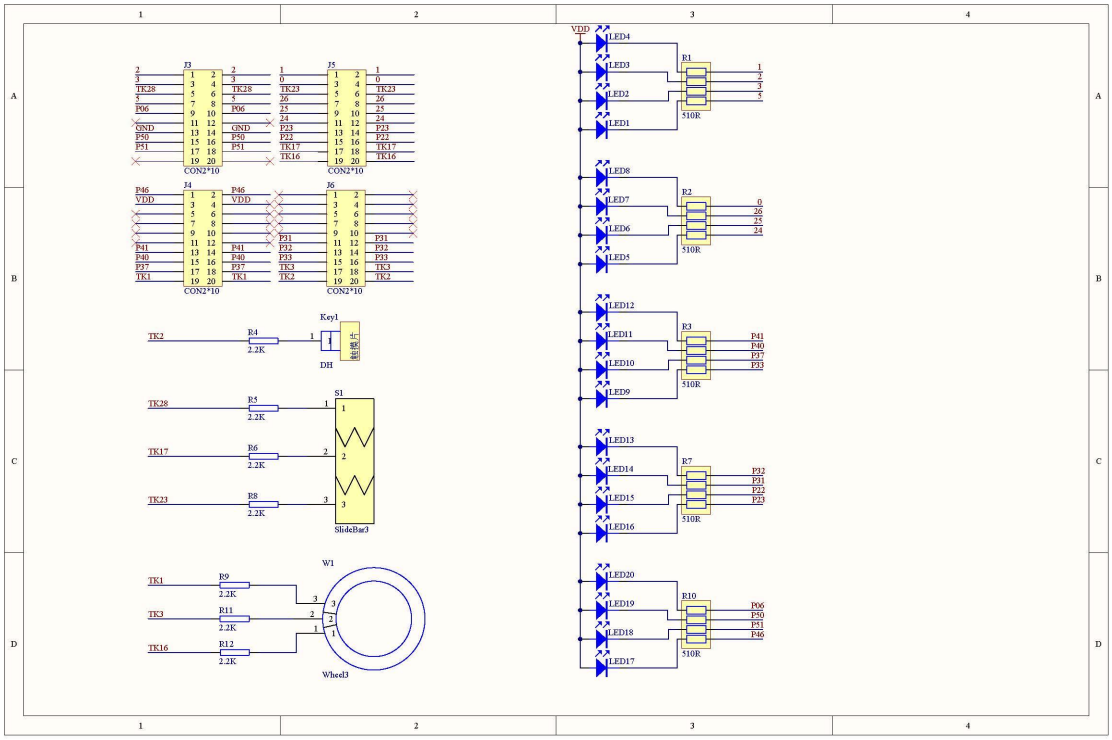


3 NBK 系列开发板电路图

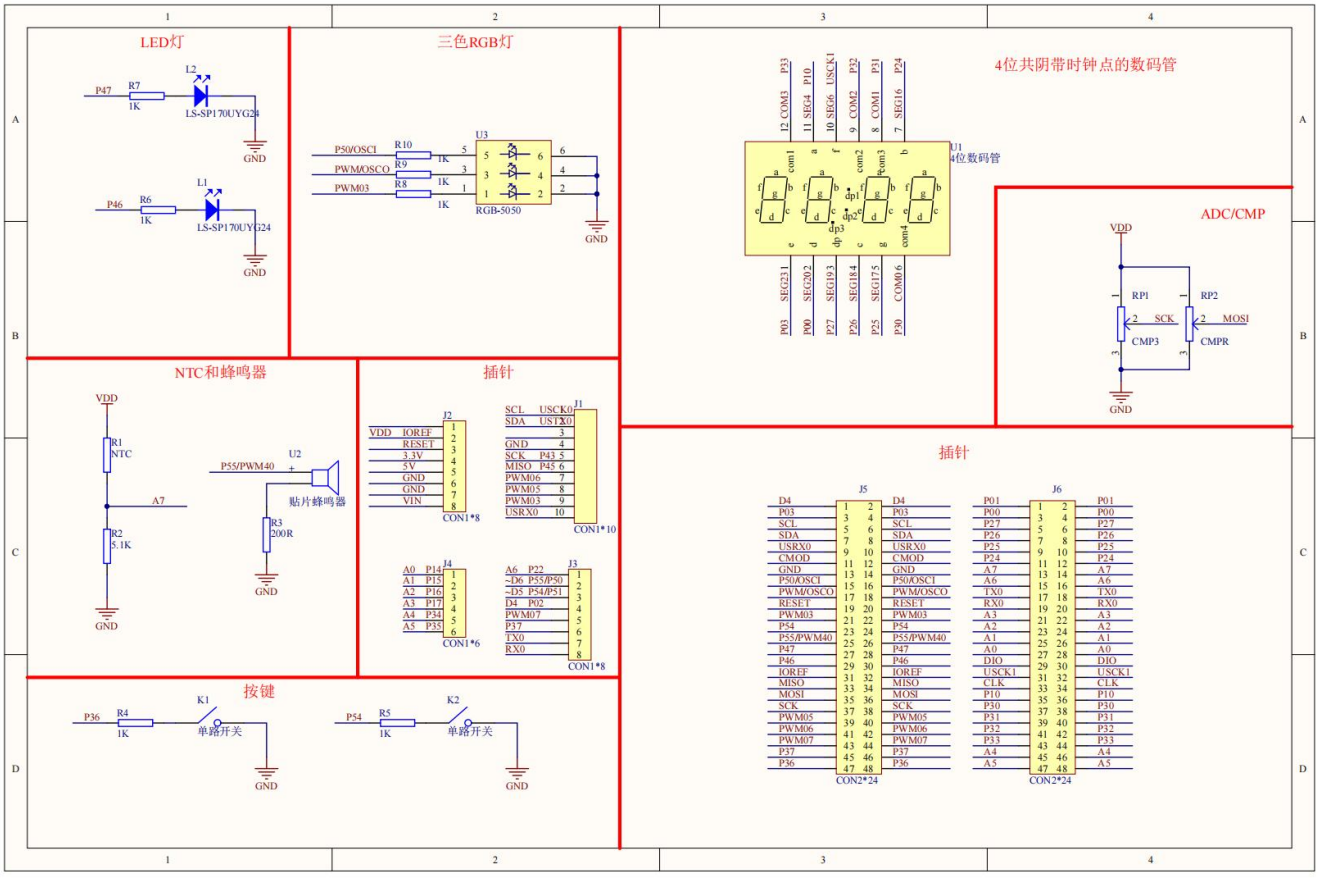
3.1 NBK1220 开发板原理图



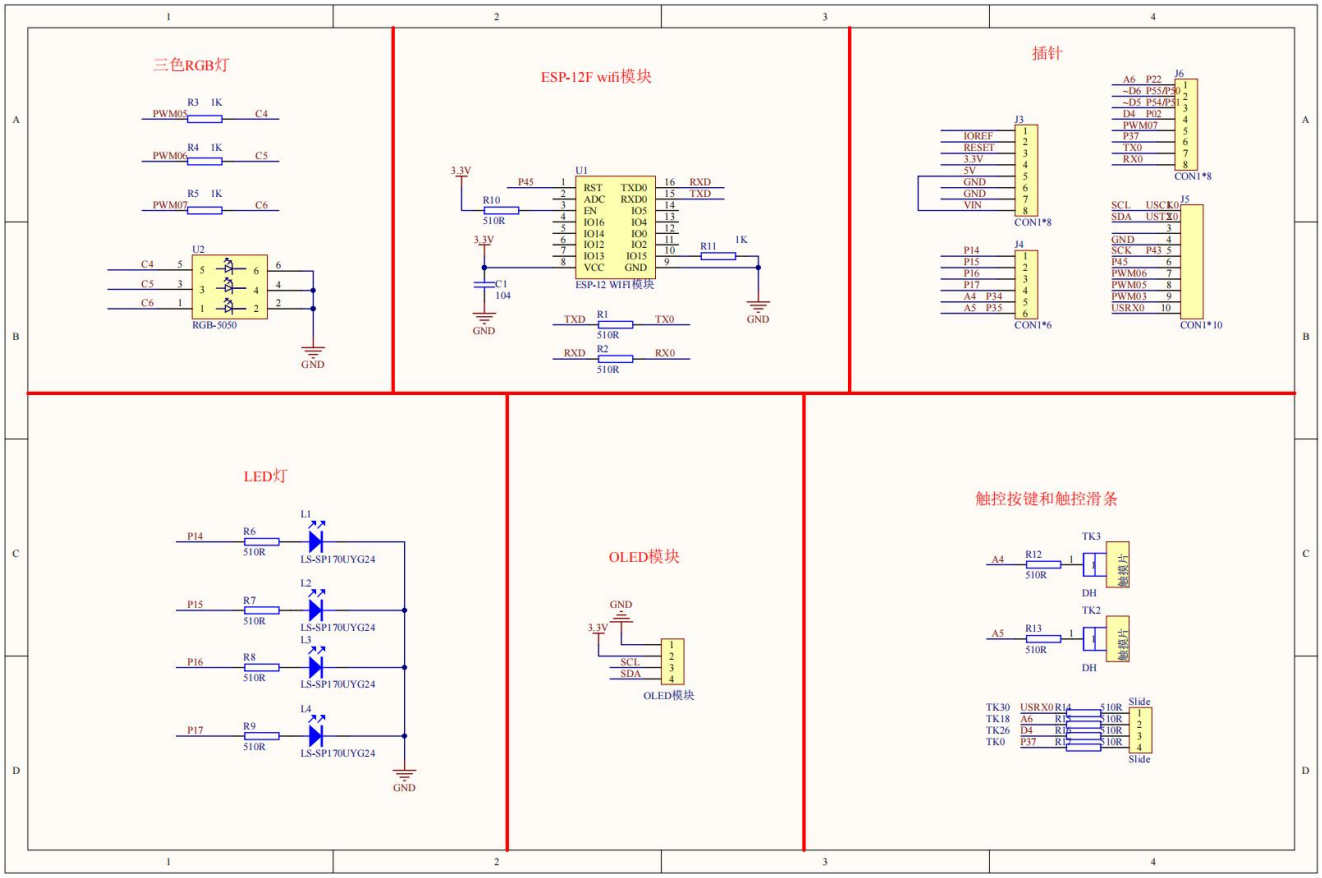
3.2 NBK-EBS001 开发板原理图



3.3 NBK-EBS002 开发板原理图



3.4 NBK-EBS003 开发板原理图



4 程序示例

4.1 硬件 LED 驱动数码管显示示例

4.1.1 总体描述

硬件 DDIC 扫描 LED 显示示例工程：DDIC_LEDDISPLAY_NBK1220_EBS002，可以通过魔盒例子工程打开并另存为用户自己的工程。

由 LED 组成的显示器是单片机应用系统中常用的输出设备。NBK1220 核心开发板上的主控芯片内部集成了硬件的 LCD/LED 显示驱动电路，可方便用户实现 LCD 和 LED 的显示驱动。其主要特点如下：

1．LCD 和 LED 显示驱动二选一，即某一时刻只能选用其中一种驱动功能。

2．LCD 和 LED 显示驱动共用相关 I/O 口和寄存器。

在 NBK-EBS002 基础功能扩展板上，集成有四位带时钟的共阴数码管，用户可以通过易码魔盒来配置 DDIC 资源，去开发和学习数码管的使用。LED 显示器的工作方式有两种： 静态显示方式和动态显示方式。静态显示的特点是每个数码管的段选必须接一个 8 位数据线来保持显示的字形码。它的缺点是占用 I/O 口较多，成本较高。所以，一般使用动态显示的方式。

动态显示的特点是将所有数码管的段选线并联在一起，由位选线控制是哪一位数码管有效。在动态扫描显示过程中轮流向各位数码管送出字形码和相应的位选，利用发光管的余辉和人眼视觉暂留作用，使人的感觉好像各位数码管同时都在显示。动态显示要比静态显示所用的 IO 口要少，这样可以节省 IC 的资源。主控芯片内部的 LED 显示驱动，可以硬件自动扫描，节省了程序的扫描时间，而且使用起来非常简单和快捷。

4.1.2 LED 驱动功能

LED 显示驱动功能如下：

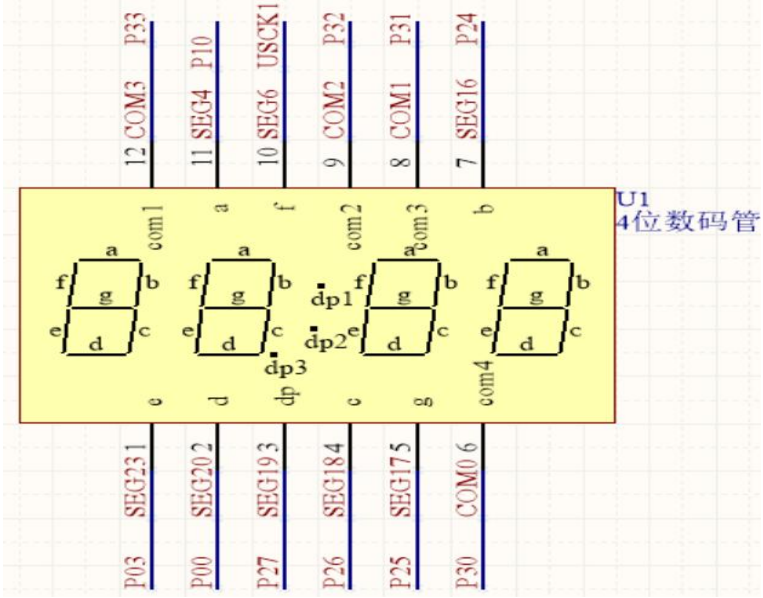
1．4 种显示驱动模式可选： 8 X 24、6 X 26、5 X 27、或 4 X28 段；

2．SEG 口驱动能力 4 级可选；

3．显示驱动电路可选择内建 32kHz LRC 或外部 32.768kHz 振荡器作为时钟源，帧频约为 64Hz。

4.1.3 在易码魔盒中的配置

在使用易码魔盒配置 DDIC 资源驱动 LED 时，首先要查看所驱动的 LED 显示器共有多少个 COM 和 SEG，根据 COM 和 SEG 的数量选择所需要的占空比。根据 NBK-EBS002 基础功能扩展板原理图可知，所用数码管有 4 个 COM 口和 8 个 SEG 口，因此在易码魔盒的芯片资源配置界面进行如图所示配置。





资源的配置步骤如下：

1．在列表勾选 DDIC 资源；
2．功能选择为 LED；
3．显示占空比控制选择为： 1/4 占空比；
4．复用管脚选择为： S4-27 为 SEG，C0-3 为 COM；
5．选择原理图对应的 COM 和 SEG 口。

其中，步骤 3 所配置的占空比要根据 LED 显示器使用的 COM 来选择，NBK1220 上的主控芯片一共有 8 个 COM 口，分别为 C0~C7，由 DDRCON 显示驱动控制寄存器的显示占空比控制位来选择使用多少 COM 口。NBK-EBS002 基础功能扩展板使用的 COM 为 C0~C3，共 4 个，根据 DDRCON 寄存器说明，选择 1/4 占空比即可。

DDRCON (93H) 显示驱动控制寄存器(读/写)

位编号	7	6	5	4	3	2	1	0
符号	DDRON	DMOD	DUTY[1:0]		VLCD[3:0]			
读/写	读/写	读/写	读/写	读/写	读/写	读/写	读/写	读/写
上电初始值	0	0	0	0	0	0	0	0

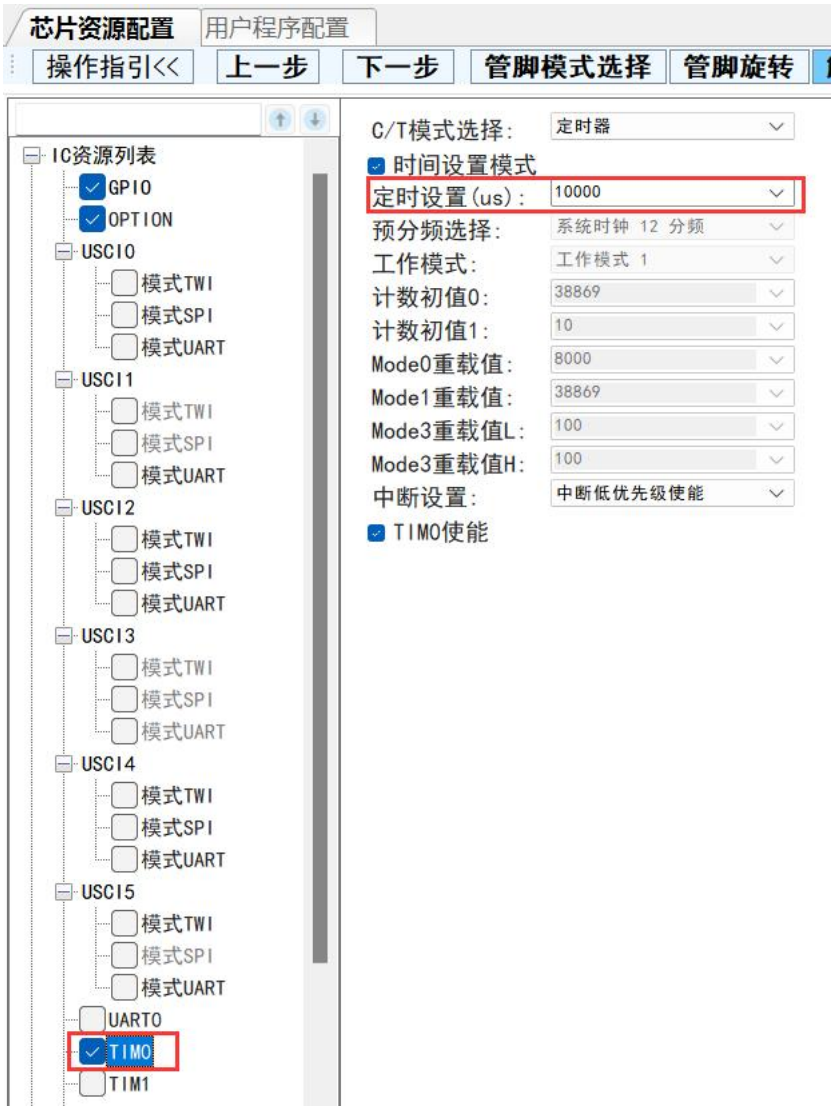
位编号	位符号	说明
7	**DDRON**	**LCD/LED 显示驱动使能控制** 0：显示驱动扫描关闭 1：显示驱动扫描打开
6	**DMOD**	**LCD/LED 显示驱动模式** 0：LCD 模式； 1：LED 模式
5~4	**DUTY[1:0]**	**LCD/LED 显示占空比控制** 00：1/8 占空比，S4~S27 为 segment，C0~C7 为 common； 01：1/6 占空比，S2~S27 为 segment，C2~C7 为 common； 10：1/5 占空比，S1~S27 为 segment，C3~C7 为 common； 11：1/4 占空比，S0~S27 为 segment，C4~C7 为 common，或 S4~S27 为 segment，C0~C3 为 common
3~0	**VLCD[3:0]**	**LCD 电压调节** $VLCD=V_{DD}*(17+VLCD[3:0])/32$

由于显示占空比控制选择 1/4 占空比，COM 有两种情况： C4~C7 或者 C0~C3，而板子上的 LED 显示器所用 COM 为 C0~C3，所以为了确定管脚配置，通过步骤 4，确定 C0~C3 为 COM 口。之后，再把 LED 显示器使用的 COM 口和 SEG 口选上，即步骤 5。

OTCON (8FH) 输出控制寄存器(读/写)

位编号	7	6	5	4	3	2	1	0
符号	USMD1[1:0]		USMD0[1:0]		VOIRS[1:0]		SCS	BIAS
读/写	读/写	读/写	读/写	读/写	读/写	读/写	读/写	读/写
上电初始值	0	0	0	0	0	0	0	0

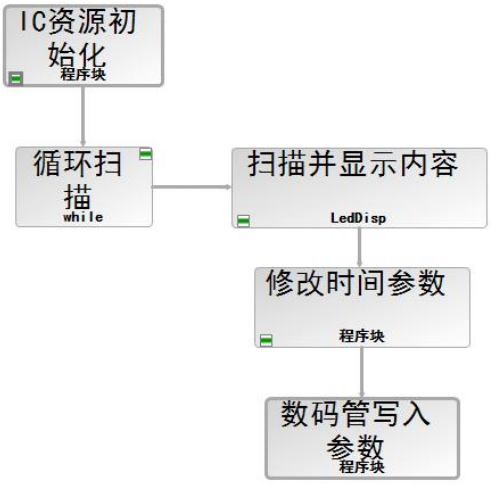
位编号	位符号	说明
3~2	**VOIRS[1:0]**	**LCD 电压输出口分压电阻选择（根据 LCD 屏大小选择适合的驱动）** 00：设定内部分压电阻总电阻值为 100kΩ 01：设定内部分压电阻总电阻值为 200kΩ 10：设定内部分压电阻总电阻值为 400kΩ 11：设定内部分压电阻总电阻值为 800kΩ 每次 Common 切换时，前 1/16 时间固定选择 100k 电阻，后 15/16 时间切换到 VORIS 选择的电阻值
1	**SCS**	**LCD/LED Segment/Common 复用管脚选择** 0：当设定为 1/4 占空比时，S0~S27 为 segment，C4~C7 为 common 1：当设定为 1/4 占空比时，S4~S27 为 segment，C0~C3 为 common
0	**BIAS**	**LCD 显示驱动偏置电压设置：** 0：1/4 偏置电压； 1：1/3 偏置电压



在 DDIC 资源配置好之后，再配置一个定时器资源，定时 10ms，操作如上，生成后魔盒会帮用户生成好定时器程序和计算好重载值。

当芯片资源配置完成后，即可在用户程序配置界面，开始图形化编程。

示例 使用 LED 硬件驱动电路，通过 Timer0 计时 1s，四位数码管上显示分、秒，从 00：00 到 59：59。






```
void main(void)
{
    /*<Generated by EasyCodeCube begin>*/
    /*<UserCodeStart>*//*<SinOne-Tag><3>*/
    SC_Init(); /*** MCU init***/
    /*<UserCodeEnd>*//*<SinOne-Tag><3>*/
    /*<UserCodeStart>*//*<SinOne-Tag><112>*/
    while(1)
    {
        /*<UserCodeStart>*//*<SinOne-Tag><31>*/
        LedDisp();
        /*<UserCodeEnd>*//*<SinOne-Tag><31>*/
        /*<UserCodeStart>*//*<SinOne-Tag><105>*/
        if(T0Flag1s)
        {
            T0Flag1s=0;
            Miao++;
            if(Miao==60)
            {
                Miao=0;
                Fen++;
                if(Fen==60)
                {
                    Fen=0;
                    Shi++;
                    if(Shi==24)
                    {
                        Shi=0;
                    }
                }
            }
        }
        /*<UserCodeEnd>*//*<SinOne-Tag><105>*/
        /*<UserCodeStart>*//*<SinOne-Tag><108>*/
        Led_DisplayData(Miao%10,Miao/10,Fen%10,Fen/10);
        //显示时钟点  DOTDISPLAY:开  DOTUNDISPLAY:关
        Led_DisplayDot(DOTUNDISPLAY,DOTUNDISPLAY,DOTDISPLAY,DOTUNDISPLAY);
        /*<UserCodeEnd>*//*<SinOne-Tag><108>*/
        /*<Begin-Inserted by EasyCodeCube for Condition>*/
    }
    /*<UserCodeEnd>*//*<SinOne-Tag><112>*/
    /*<Generated by EasyCodeCube end>*/
}
```

```
void LedDisp()
{
    /*<UserCodeStart>*//*<SinOne-Tag><24>*/
    uint8_t i;
    for(i=0;i<8;i++)
    {
        LedTemp[i] = 0;
    }
    for(i=0;i<4;i++)
    {
        LedSegData(LedCodeTab[LedDataTab[i]],LEDCOM0+i,DotTemp[i]);
    LCDRAM[4]  = LedTemp[0];
    LCDRAM[6]  = LedTemp[1];
    LCDRAM[16] = LedTemp[2];
    LCDRAM[17] = LedTemp[3];
    LCDRAM[18] = LedTemp[4];
    LCDRAM[19] = LedTemp[5];
    LCDRAM[20] = LedTemp[6];
    LCDRAM[23] = LedTemp[7];
    }

    /*<UserCodeEnd>*//*<SinOne-Tag><24>*/
}
void LedSegData(uint8_t LedData,LedSelCOM COMx,LedDotDisplay DotDisp)
{
    /*<UserCodeStart>*//*<SinOne-Tag><75>*/
    LedTemp[0]  |=  ((LedData>>0)&0x01)<<COMx;
    LedTemp[1]  |=  ((LedData>>1)&0x01)<<COMx;
    LedTemp[2]  |=  ((LedData>>2)&0x01)<<COMx;
    LedTemp[3]  |=  ((LedData>>3)&0x01)<<COMx;
    LedTemp[4]  |=  ((LedData>>4)&0x01)<<COMx;
    LedTemp[5]  |=  (((LedData | DotDisp)>>5)&0x01)<<COMx;
    LedTemp[6]  |=  ((LedData>>6)&0x01)<<COMx;
    LedTemp[7]  |=  ((LedData>>7)&0x01)<<COMx;

    /*<UserCodeEnd>*//*<SinOne-Tag><75>*/
}
```

4.1.4 函数接口及变量

图形化编程使用的具体函数和变量说明如下两表所示：

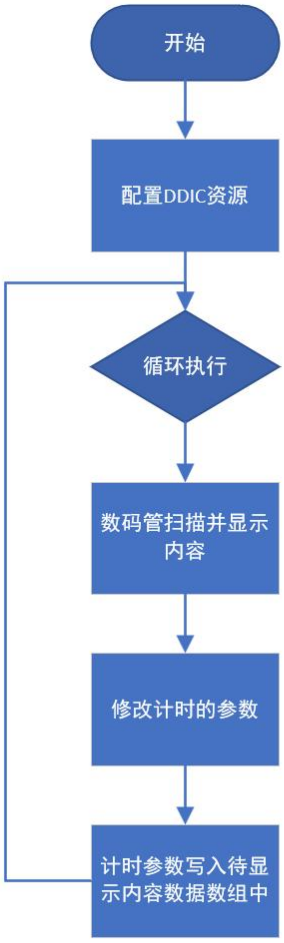
函数	函数功能	输入参数	返回值	备注
LedDisp()	LED 扫描显示	无	无	LCDRAM[] 地址必须跟使用的 COM 和 SEG 口相对应
Led_DisplayData(unsigned Int Num1,unsigned int Num2,unsigned int Num3, unsigned int Num4)	LED 显示的数据	Num1~Num4 ： C0~C3 要显示的数据	无	无
Led_DisplayDot(LedDotDisplay DotDisp1,LedDotDisplay DotDisp2,LedDotDisplay DotDisp3,LedDotDisplay DotDisp4)	LED 是否选择显示小数点	DotDisp1-4 ： 对应 C0-C3 的小数点 DOTDISPLAY: 开 DOTUNDISPLAY:关	无	无
LedSegData(uint8_t LedData, LedSelCOM COMx, LedDotDisplay DotDisp)	LED 显示数据转换	LedData ： 要写入 LED 的 8 位数据 COMx：选择的 COM 口（0-7） DotDisp：小数点显示位	无	将对应 COM 口的八位数数据存入数组

变量名	说明
LCDRAM[]	显示 RAM 配置，在 sc95f_ddic.c 中已有定义
LedCodeTab[]	数码管编码表
LedDataTab[]	4 个要写入数码管的数据
LedTemp[8]	8 个，1 位要写入 seg 口的数据
DotTemp[]	4 个选择是否要显示各位小数点的数

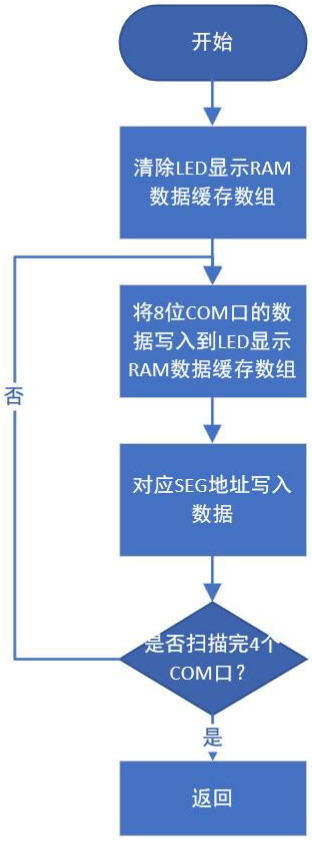
4.1.5 程序流程图

主程序和扫描函数的程序流程图如下：

其中，在扫描函数中，特别要将 COM 选择对，且 SEG 地址要对应上。

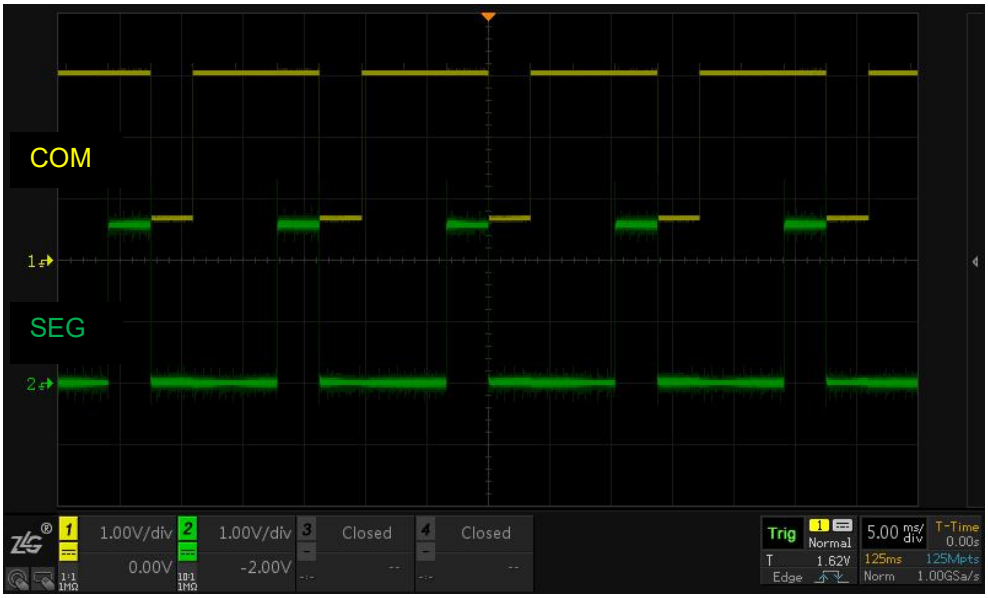


主程序



扫描函数

4.1.6 LED 波形



4.2 软件 GPIO 实现数码管显示示例

4.2.1 总体描述

GPIO 扫描 LED 显示示例：GPIO_LEDDISPLAY_NBK1220_EBS002，可以通过魔盒例子工程打开并另存为用户自己的工程。

LED 显示除了可以用芯片内部的 LED 驱动外，还可以用 I/O 进行动态扫描来显示。NBK1220 核心开发板上的主控芯片提供了多个可控制的双向 GPIO 端口，可以用输入输出控制寄存器用来控制各端口的输入输出状态，当端口作为输入时，每个 IO 端口带有由 PxPHy 控制的内部上拉电阻。每个 IO 同其他功能复用，其中 P3 可以通过设置输出 1/4V_{DD} 或 1/3V_{DD} 的电压，可用来作为 LCD 显示的 COM 驱动。I/O 端口在输入或输出状态下，从端口数据寄存器里读到的都是端口的实际状态值。

4.2.2 IO 主要功能

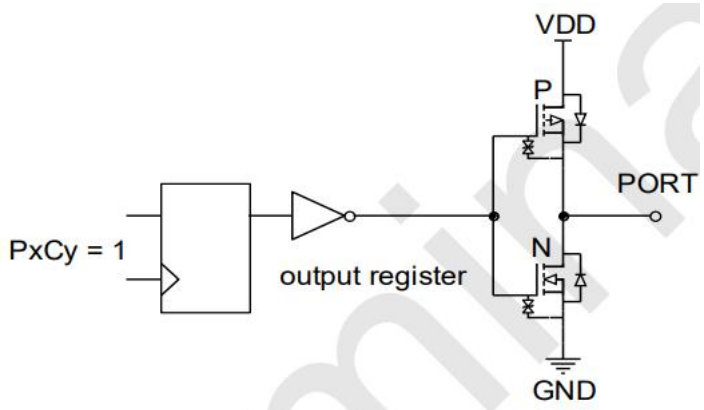
GPIO 的主要功能是输出和输入，其主要分为强推挽输出模式、带上拉的输入模式、高阻输入模式。

强推挽输出

在强推挽输出模式下，能够提供持续的大电流驱动：

1. 除 P04/P05/P06 之外的 IO 驱动能力为： 大于 10mA 的输出高，大于 50mA 的输出低。
2. P04/P05/P06 驱动能可达到： 大于 20mA 的输出高，大于 50mA 的输出低。

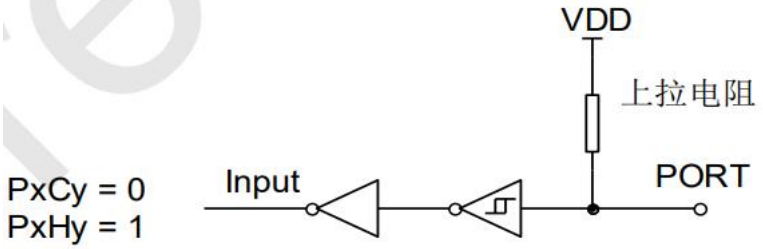
强推挽输出模式的端口结构示意图如下：



强推挽输出模式

带上拉的输入模式

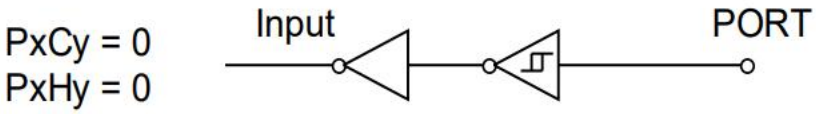
带上拉的输入模式下，输入口上恒定接一个上拉电阻，仅当输入口上电平被拉低时，才会检测到低电平信号。带上拉的输入模式的端口结构示意图如下：



带上拉的输入模式

高阻输入模式

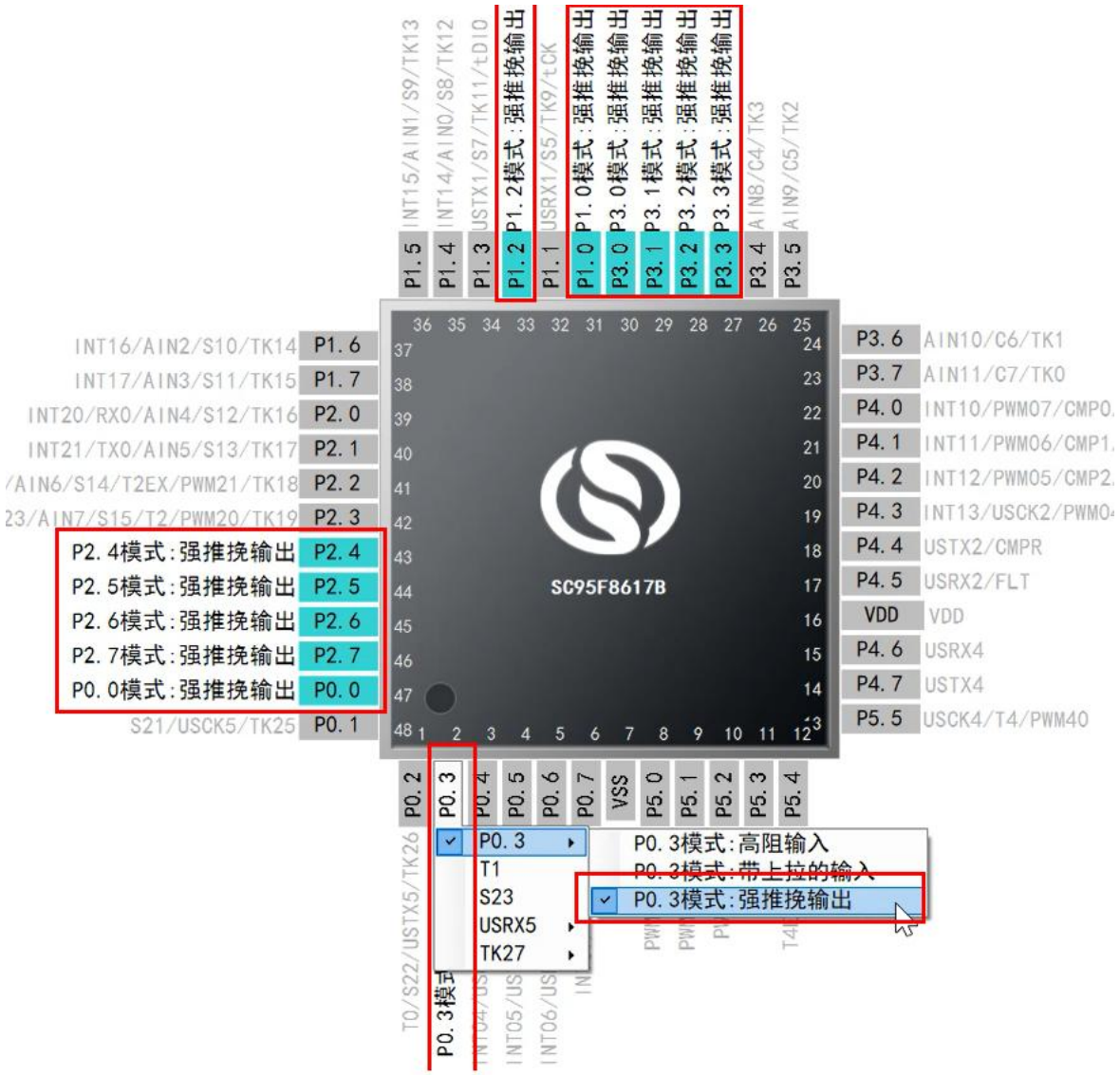
高阻输入模式的端口结构示意图如下：



高阻输入模式

4.2.3 在易码魔盒中的配置

在易码魔盒中，用户可以直接在芯片资源配置界面中，直接点击 LED 对应的引脚配置成强推挽输出，魔盒会生成对应引脚初始化的代码，不需要用户手动配置寄存器。



用户如需通过寄存器手动配置，可以通过配置 PxCON 和 PxPH 寄存器来设置输入输出。

P0CON (9AH) P0 口输入/输出控制寄存器(读/写)

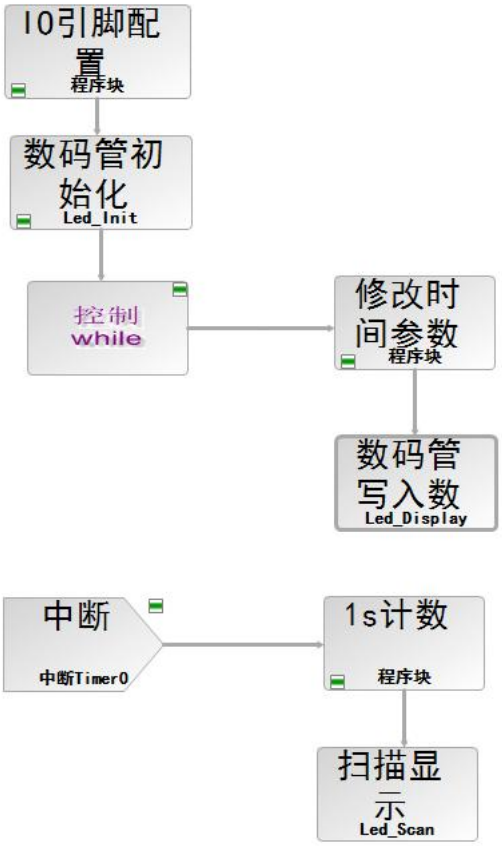
位编号	7	6	5	4	3	2	1	0
符号	P0C7	P0C6	P0C5	P0C4	P0C3	P0C2	P0C1	P0C0
读/写	读/写	读/写	读/写	读/写	读/写	读/写	读/写	读/写
上电初始值	0	0	0	0	0	0	0	0

P0PH (9BH) P0 口上拉电阻控制寄存器(读/写)

位编号	7	6	5	4	3	2	1	0
符号	P0H7	P0H6	P0H5	P0H4	P0H3	P0H2	P0H1	P0H0
读/写	读/写	读/写	读/写	读/写	读/写	读/写	读/写	读/写
上电初始值	0	0	0	0	0	0	0	0

当芯片资源配置完成后，即可在用户程序配置界面，开始图形化编程。

示例 使用 GPIO 动态扫描 LED，通过 Timer0 计时 1s，四位数码管上显示分、秒，从 00：00 到 59：59。




```
void main(void)
{
    /*<Generated by EasyCodeCube begin>*/
    /*<UserCodeStart>*//*<SinOne-Tag><3>*/
    SC_Init(); /*** MCU init***/
    /*<UserCodeEnd>*//*<SinOne-Tag><3>*/
    /*<UserCodeStart>*//*<SinOne-Tag><20>*/
    Led_Init();
    /*<UserCodeEnd>*//*<SinOne-Tag><20>*/
    /*<UserCodeStart>*//*<SinOne-Tag><21>*/
    while(1)
    {
        /*<UserCodeStart>*//*<SinOne-Tag><120>*/
        if(T0Flag1s)
        {
            T0Flag1s=0;
            Miao++;
            if(Miao==60)
            {
                Miao=0;
                Fen++;
                if(Fen==60)
                {
                    Fen=0;
                    Shi++;
                    if(Shi==24)
                    {
                        Shi=0;
                    }
                }
            }
        }
        /*<UserCodeEnd>*//*<SinOne-Tag><120>*/
        /*<UserCodeStart>*//*<SinOne-Tag><49>*/
        Led_Display( Miao%10 , Miao/10 , Fen%10 , Fen/10 , 0x04 );
        /*<UserCodeEnd>*//*<SinOne-Tag><49>*/
        /*<Begin-Inserted by EasyCodeCube for Condition>*/
    }
    /*<UserCodeEnd>*//*<SinOne-Tag><21>*/
    /*<Generated by EasyCodeCube end>*/
}
```

```
void Timer0Interrupt()      interrupt 1
{
    /*TIM0_it write here begin*/
    TIM0_ModelSetReloadCounter(33536);
    /*TIM0_it write here*/
    /*<Generated by EasyCodeCube begin>*/
    /*<UserCodeStart>*//*<SinOne-Tag><6>*/
    //Timer0Interrupt
    {
        /*<UserCodeStart>*//*<SinOne-Tag><127>*/
        T0FlaglsCount++;
        if(T0FlaglsCount >= 1000)
        {
            T0FlaglsCount = 0;
            T0Flagls = 1;
        }
        /*<UserCodeEnd>*//*<SinOne-Tag><127>*/
        /*<UserCodeStart>*//*<SinOne-Tag><46>*/
        Led_Scan();
        /*<UserCodeEnd>*//*<SinOne-Tag><46>*/
        /*<Begin-Inserted by EasyCodeCube for Condition>*/
    }
    /*<UserCodeEnd>*//*<SinOne-Tag><6>*/
    /*<Generated by EasyCodeCube end>*/
    /*Timer0Interrupt Flag Clear begin*/
    /*Timer0Interrupt Flag Clear end*/
}
void Led_Scan()
{
    /*<UserCodeStart>*//*<SinOne-Tag><36>*/
  static unsigned int i=0;
  LED_NT_COM1 = 1;  //首先关闭所有的COM口消影
  LED_NT_COM2 = 1;
  LED_NT_COM3 = 1;
  LED_NT_COM4 = 1;

  LED_NT_SEGA = (LedDataTab[i]&(0x01));
  LED_NT_SEGB = (LedDataTab[i]&(0x02))>>1;
  LED_NT_SEGC = (LedDataTab[i]&(0x04))>>2;
  LED_NT_SEGD = (LedDataTab[i]&(0x08))>>3;
  LED_NT_SEGE = (LedDataTab[i]&(0x10))>>4;
  LED_NT_SEGF = (LedDataTab[i]&(0x20))>>5;
  LED_NT_SEGG = (LedDataTab[i]&(0x40))>>6;
  LED_NT_SEGDP = (LedDotTab[i]&(0x01));

  GPIO_WriteLow(LED_COM_List[i]>>8,LED_COM_List[i]&0xff);  //打开COM口

  i++;
  if(i>=4) i=0;

    /*<UserCodeEnd>*//*<SinOne-Tag><36>*/
}
void Led_Display(unsigned int Com1,unsigned int Com2,unsigned int Com3,unsigned int Com4,unsigned int Dot)
{
    /*<UserCodeStart>*//*<SinOne-Tag><30>*/
  LedDataTab[0] = LedCodeTab[Com1];
  LedDataTab[1] = LedCodeTab[Com2];
  LedDataTab[2] = LedCodeTab[Com3];
  LedDataTab[3] = LedCodeTab[Com4];
  LedDotTab[0] = (Dot&0x01);
  LedDotTab[1] = (Dot&0x02)>>1;
  LedDotTab[2] = (Dot&0x04)>>2;
  LedDotTab[3] = (Dot&0x08)>>3;

    /*<UserCodeEnd>*//*<SinOne-Tag><30>*/
}
```

4.2.4 函数接口及变量

图形化编程使用的具体函数和变量说明如下两表所示：

函数	函数功能	输入参数	返回值	备注
Led_Scan ()	LED 扫描显示	无	无	放到定时器中断中，LED 引脚宏定义必须对应
Led_Display (unsigned Int Com1,unsigned int Com2,unsigned int Com3, unsigned int Com4,unsigned int Dot)	LED 显示的数据	Com1~Com4 ：C0~C3 要显示的数据 Dot： C0~C3 小数点位，低 4 位 bit 为 1，则显示	无	无
Led_Init()	LED 初始化	无	无	LED 先全部关闭

变量名	说明
LED_COM_List []	数码管 COM 口扫描列表
LedCodeTab[]	共阴数码管编码表
LedDataTab[]	4 个要写入数码管的数据
LedDotTab[4]	4 个小数点显示位的数据
LED_NT_COM1_INT	C0 引脚值宏定义，8-16 位代表 Px 口，7-0 位代表具体引脚位
LED_NT_COM2_INT	C1 引脚值宏定义，8-16 位代表 Px 口，7-0 位代表具体引脚位
LED_NT_COM3_INT	C2 引脚值宏定义，8-16 位代表 Px 口，7-0 位代表具体引脚位
LED_NT_COM4_INT	C3 引脚值宏定义，8-16 位代表 Px 口，7-0 位代表具体引脚位
LED_NT_SEGA	数码管 A 段对应引脚宏定义
LED_NT_SEGB	数码管 B 段对应引脚宏定义
LED_NT_SEGC	数码管 C 段对应引脚宏定义
LED_NT_SEGD	数码管 D 段对应引脚宏定义
LED_NT_SEGDP	数码管 DP 段对应引脚宏定义
LED_NT_SEGE	数码管 E 段对应引脚宏定义
LED_NT_SEGF	数码管 F 段对应引脚宏定义
LED_NT_SEGG	数码管 G 段对应引脚宏定义

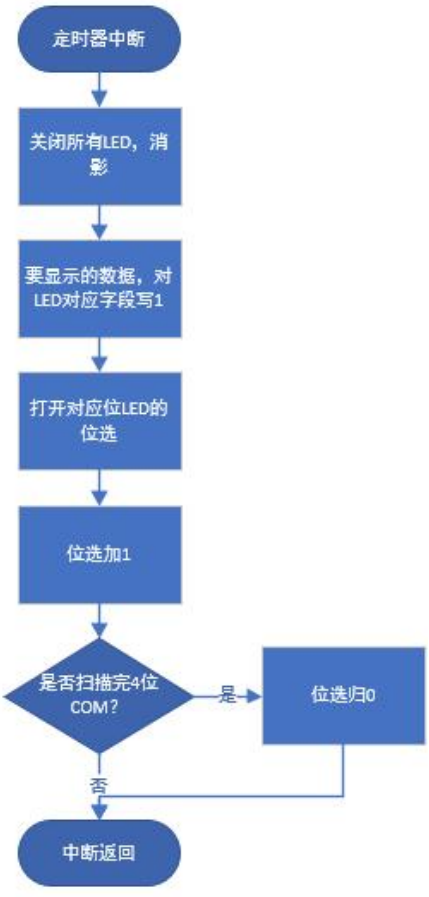
4.2.5 程序流程图

主程序和扫描函数的程序流程图如下：

其中，扫描函数要放到定时器中断中执行，LED 对应字段的宏定义和位选的宏定义要跟实际的引脚对应上。



主程序



定时器中断扫描

4.3 硬件 LCD 驱动 4 位 LCD 屏显示示例

4.3.1 总体描述

硬件 DDIC 扫描 4 位 LCD 显示示例：DDIC_LCDDISPLAY_NBK1220，可以通过魔盒例子工程打开并另存为用户自己的工程。

一般的段式 LCD 屏跟数码管一样多个液晶公用一个公共端 COM，另一端一般称之为 SEG，在 SEG 和 COM 上加上电压就可以“点亮”该段液晶。但是，与驱动 LED 不同的一点是 COM 口与 SEG 之间必须加上对称的交流电压，以保证加到 LCD 两端的交流电压平均值为零。过大的直流电压会使液晶材料迅速分解，大大缩短 LCD 的工作寿命，这也是 LCD 与段式 LED 最大的不同之处。与 LED 的显示类似，LCD 要显示出来有一定的门限电压，高于这个电压则会提高“亮度”，在 LCD 中一般称之为对比度，低于这个门限电压则不显示。这个们限电压在制作好的时候一般被称之为 BIAS(偏置)。通过芯片内置的 LCD 驱动和易码魔盒，我们可以很便捷的驱动 1/3bias 、1/4bias 的 LCD 屏。由于 EBS002、EBS003 板上未安装有段式 LCD 屏，如果有段式 LCD 屏显示需求的，请通过 NBK1220 核心板外接电路的方式实现。

4.3.2 LCD 驱动功能

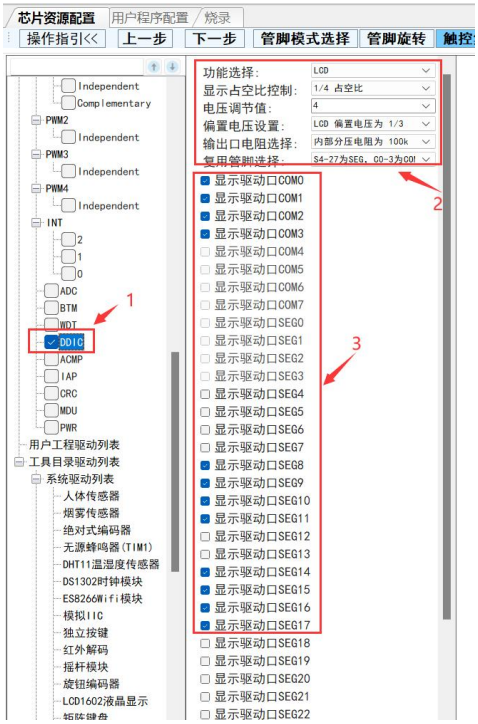
LCD 显示驱动功能如下：

1．4 种显示驱动模式可选:8X24、6X26、5X27、或 4X28 段；
2．2 种偏置方式可选: 1/4 Bias 和 1/3 Bias；
3．COM 口驱动能力 4 级可选；
4．显示驱动电路可选择内建 32kHz LRC 或外部 32.768 kHz 振荡器作为时钟源，帧频约为 64Hz。

注意：当用户所使用 LCD 屏的驱动帧频不是 64Hz 时，建议联系技术人员，咨询解决方案。

4.3.3 在易码魔盒中的配置

在使用易码魔盒配置 DDIC 资源驱动 LCD 时，首先要查看所驱动的 LCD 显示器使用的 DUTY、Bias、驱动电压是多少，然后根据所需的条件，在易码魔盒的芯片资源配置界面进行如图所示配置。



资源的配置步骤如下：

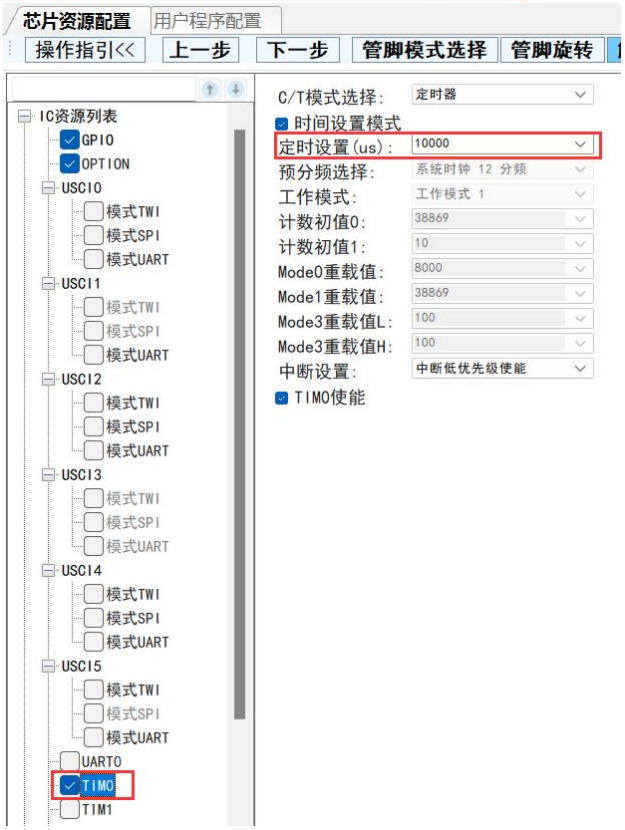
1．在列表勾选 DDIC 资源；
2．功能选择为 LCD；
3．显示占空比控制选择为： 1/4 占空比；
4．电压调节为 4；
5．偏置电压设置为 1/3。
6．输出口电阻选择为 100k；
7．复用管脚选择为 C0-3 为 COM；
8．选择 LCD 显示器对应的 COM 和 SEG 口。

其中，步骤 3 所配置的占空比要根据 LCD 显示器是多少位，用多少 COM 来决定，NBK1220 核心开发板主控芯片内部的 LCD 驱动一共有 8 个 COM 口，分别为 C0~C7，由 DDRCON 显示驱动控制寄存器的显示占空比控制位来选择使用多少 COM 口。如果使用的是 4 位 LCD 段码屏，根据 DDRCON 寄存器说明，选择 1/4 占空比即可，再根据实际使用的是哪几位 COM，来决定管脚复用的状态，即步骤 7。然后就是比较重要的步骤，电压调节，根据所用 LCD 的驱动电压来设置电压调节位，根据 VLCD 位的说明，当 VDD 为 5V 时，VLCD 为 4，输出的驱动电压为 3.3V，即步骤 4。然后，根据要使用 LCD 显示屏的需求，选择偏置电压和输出口电阻。最后，再把 LCD 显示器使用的 COM 口和 SEG 口选上，即步骤 8。

DDRCON (93H) 显示驱动控制寄存器(读/写)

位编号	7	6	5	4	3	2	1	0
符号	DDRON	DMOD	DUTY[1:0]		VLCD[3:0]			
读/写	读/写	读/写	读/写	读/写	读/写	读/写	读/写	读/写
上电初始值	0	0	0	0	0	0	0	0

位编号	位符号	说明
7	**DDRON**	**LCD/LED 显示驱动使能控制** 0：显示驱动扫描关闭 1：显示驱动扫描打开
6	**DMOD**	**LCD/LED 显示驱动模式** 0：LCD 模式； 1：LED 模式
5~4	**DUTY[1:0]**	**LCD/LED 显示占空比控制** 00：1/8 占空比，S4~S27 为 segment，C0~C7 为 common； 01：1/6 占空比，S2~S27 为 segment，C2~C7 为 common； 10：1/5 占空比，S1~S27 为 segment，C3~C7 为 common； 11：1/4 占空比，S0~S27 为 segment，C4~C7 为 common，或 S4~S27 为 segment，C0~C3 为 common
3~0	**VLCD[3:0]**	**LCD 电压调节** VLCD=V_{DD}*(17+VLCD[3:0])/32



在 DDIC 资源配置好之后，再配置一个定时器资源，定时 10ms，操作如上，生成后魔盒会帮用户生成好定时器程序和计算好重载值。

当芯片资源配置完成后，即可在用户程序配置界面，开始图形化编程。

示例 使用 LCD 硬件驱动电路，驱动条件为 1/4DUTY ， 1/3BIAS，3.3V LCD 显示屏，通过 Timer0 计时 1s，四位 LCD 显示屏上显示分、秒，从 00：00 到 59：59。




```
void main(void)
{
    /*<Generated by EasyCodeCube begin>*/
    /*<UserCodeStart>*//*<SinOne-Tag><3>*/
    SC_Init(); /*** MCU init***/
    /*<UserCodeEnd>*//*<SinOne-Tag><3>*/
    /*<UserCodeStart>*//*<SinOne-Tag><112>*/
    while(1)
    {
        /*<UserCodeStart>*//*<SinOne-Tag><31>*/
        LcdDisp();
        /*<UserCodeEnd>*//*<SinOne-Tag><31>*/
        /*<UserCodeStart>*//*<SinOne-Tag><105>*/
        if(T0Flag1s)
        {
            T0Flag1s=0;
            Miao++;
            if(Miao==60)
            {
                Miao=0;
                Fen++;
                if(Fen==60)
                {
                    Fen=0;
                    Shi++;
                    if(Shi==24)
                    {
                        Shi=0;
                    }
                }
            }
        }
        /*<UserCodeEnd>*//*<SinOne-Tag><105>*/
        /*<UserCodeStart>*//*<SinOne-Tag><108>*/
        Lcd_DisplayData(Fen/10,Fen%10,Miao/10,Miao%10);
        //显示时钟点  1:开  0:关
        Lcd_DisplayDot(1,0,0,0);
        /*<UserCodeEnd>*//*<SinOne-Tag><108>*/
        /*<Begin-Inserted by EasyCodeCube for Condition>*/
    }
    /*<UserCodeEnd>*//*<SinOne-Tag><112>*/
    /*<Generated by EasyCodeCube end>*/
}
```

```
void LcdDisp()
{
    /*<UserCodeStart>*//*<SinOne-Tag><238>*/
    uint8_t i;
    for(i=0;i<8;i++)
    {
        LcdTemp[i] = 0;
    }
    for(i=0;i<4;i++)
    {
        LcdSegData(LcdCodeTab[LcdDataTab[i]],DotTemp[i]);
        if(i==0)  //位1
        {
          LCDRAM[8]  = LcdTemp[0] | LcdTemp[5]| LcdTemp[4]| LcdTemp[7];
          LCDRAM[9]  = LcdTemp[1] | LcdTemp[2]| LcdTemp[3]| LcdTemp[6];
         }
        if(i==1)  //位2
        {
          LCDRAM[10] = LcdTemp[0] | LcdTemp[5]| LcdTemp[4]| LcdTemp[7];
          LCDRAM[11] = LcdTemp[1] | LcdTemp[2]| LcdTemp[3]| LcdTemp[6];
        }
        if(i==2)  //位3
        {
          LCDRAM[14] = LcdTemp[0] | LcdTemp[5]| LcdTemp[4]| LcdTemp[7];
          LCDRAM[15] = LcdTemp[1] | LcdTemp[2]| LcdTemp[3]| LcdTemp[6];
         }
        if(i==3)  //位4
        {
          LCDRAM[16] = LcdTemp[0] | LcdTemp[5]| LcdTemp[4]| LcdTemp[7];
          LCDRAM[17] = LcdTemp[1] | LcdTemp[2]| LcdTemp[3]| LcdTemp[6];
        }
    }

    /*<UserCodeEnd>*//*<SinOne-Tag><238>*/
}
  void LcdSegData(uint8_t LcdData,bit DotDisp)
 {
     /*<UserCodeStart>*//*<SinOne-Tag><235>*/
     LcdTemp[0]  =  ((LcdData>>0)&0x01)<<LCDCOM0;  //A
     LcdTemp[1]  =  ((LcdData>>1)&0x01)<<LCDCOM0;  //B
     LcdTemp[2]  =  ((LcdData>>2)&0x01)<<LCDCOM2;  //C
     LcdTemp[3]  =  ((LcdData>>3)&0x01)<<LCDCOM3;  //D
     LcdTemp[4]  =  ((LcdData>>4)&0x01)<<LCDCOM2;  //E
     LcdTemp[5]  =  ((LcdData>>5)&0x01)<<LCDCOM1;  //F
     LcdTemp[6]  =  ((LcdData>>6)&0x01)<<LCDCOM1;  //G
     LcdTemp[7]  =  (((LcdData>>7) | DotDisp)&0x01)<<LCDCOM3;  //DP

     /*<UserCodeEnd>*//*<SinOne-Tag><235>*/
 }
```

4.3.4 函数接口及变量

图形化编程使用的具体函数和变量说明如下两表所示：

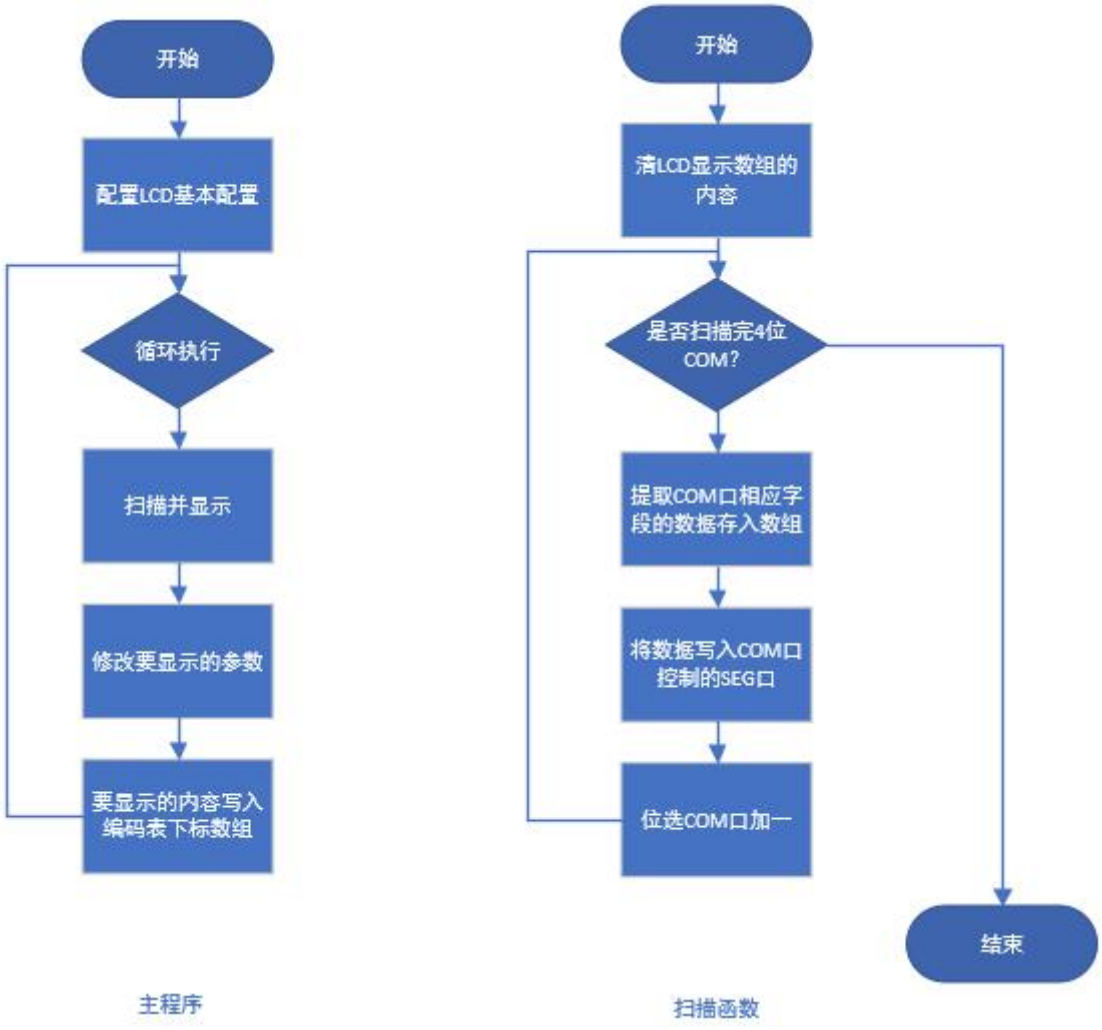
函数	函数功能	输入参数	返回值	备注
LcdDisp()	LCD 扫描显示	无	无	LCDRAM[] 中数字的地址必须跟使用的 COM 和 SEG 口相对应
Lcd_DisplayData(unsigned Int Num1,unsigned int Num2,unsigned int Num3, unsigned int Num4)	LCD 显示的数据	Num1~Num4 ：C0~C3 要显示的数据	无	无
Lcd_DisplayDot(bit DotDisp1,bit DotDisp2,bit DotDisp3,bit DotDisp4)	LCD 是否选择显示小数点	DotDisp1-4：对应 C0-C3 的小数点 1：开 0：关	无	无
LcdSegData(uint8_t LedData, bit DotDisp)	LCD 显示数据转换	LedData：要写入 LCD 的 8 位数据 DotDisp：小数点显示位	无	将对应 COM 口的 8 位数据存入数组

变量名	说明
LCDRAM[]	显示 RAM 配置，在 sc95f_ddic.c 中已有定义
LcdCodeTab[]	LCD 显示编码表
LcdDataTab[]	4 个要写入 LCD 显示屏的数据
LcdTemp[8]	8 个 1 位要写入 seg 口的数据
DotTemp[]	4 个选择是否要显示各位小数点的数

4.3.5 程序流程图

主程序和扫描函数的程序流程图如下：

其中，在扫描函数中，要将 SEG 口的地址对应上，且 COM 口控制的 SEG 口要与原理图上对应。



4.3.6 1/3BIAS LCD 波形



4.4 ADC 测量热敏电阻示例

4.4.1 总体描述

ADC 使用示例：ADC_DDIC_TEMPERATURE_NBK1220_EBS002，可以通过魔盒例子工程打开并另存为用户自己的工程。

NBK1220 核心开发板上的主控芯片内部集成有 17 路 12 位高精度 1M 高速 ADC，外部的 16 路 ADC 和 IO 口的其它功能复用，内部的一路可接至 1/4V_{DD}，配合内部 2.048V、1.024V 或 2.4V 参考电压用于测量 VDD 电压。1MHz 超高速采样时钟，采样到完成转换的总时间低至 2us。

ADC 参考电压可以有 4 种选择:

1．VDD 管脚(即直接是内部的 V_{DD});
2．内部 Regulator 输出的参考电压 2.048V。
3．内部 Regulator 输出的参考电压 1.024V。
4．内部 Regulator 输出的参考电压 2.4V。

在 NBK-EBS002 基础功能扩展板上，集成有热敏电阻 NTC ，该热敏电阻 25°C 时为 10K，用户可以通过易码魔盒来生成和编辑代码，进行 ADC 采样，测量温度。

4.4.2 ADC 转换步骤

用户实际进行 ADC 转换所需要的操作步骤如下:

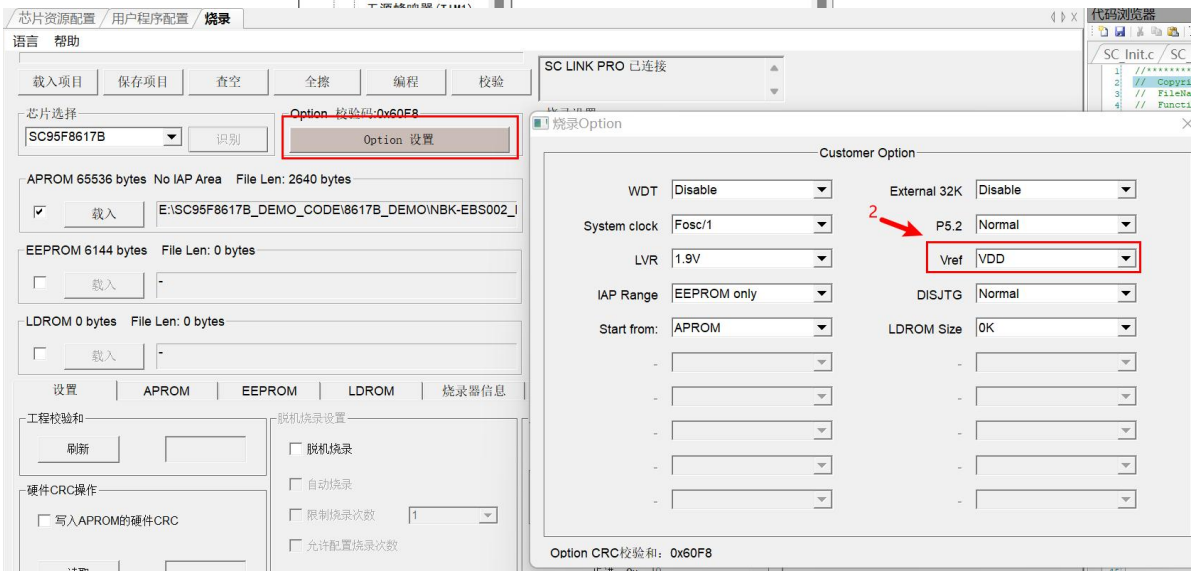
1．设定 ADC 输入管脚（设定 AINx 对应的位为 ADC 输入，通常 ADC 管脚会预先固定)；
2．设定 ADC 参考电压 Vref，设定 ADC 转换所用的频率；
3．开启 ADC 模块电源；
4．选择 ADC 输入通道（设置 ADCIS 位，选择 ADC 输入通道）；
5．启动 ADCS，转换开始；
6．等待 EOC/ADCIF=1，如果 ADC 中断使能，则 ADC 中断会产生，用户需要软件清 0 EOC/ADCIF 标志；
7．从 ADCVH、ADCVL 获得 12 位数据，先高位后低位，一次转换完成;
8．如果不换输入通道，则重复 5-7 的步骤，进行下一次转换。

注意：在设定 IE[6](EADC)前，使用者最好用软件先清除 EOC/ADCIF，并且在 ADC 中断服务程序执行完时，也清除该 EOC/ADCIF，以避免不断的产生 ADC 中断。

4.4.3 在易码魔盒中的配置

上述的 ADC 转换步骤，在魔盒中我们可以很方便的操作。





资源的配置步骤如下：

1. 资源在列表选中 ADC；
2. 选择 ADC 采样时间（根据需求选择）；
3. 选择 ADC 采样引脚；
4. 采样口输入选择 ADC 采样引脚；
5. 设置中断优先级和是否生成清除中断标志位语句（根据需求选择）；
6. 开启 ADC 功能开关；
7. 在烧录的 Option 设置中，选择 ADC 采样的参考电压。

用户如需在程序中，切换 ADC 采样输入，测量另一路 ADC，可以操作 ADC 的 BSP 控件，易码魔盒已封装好了相应的控件，也可以在程序中操作 ADCCON 寄存器，设置相应 ADC 采样输入，不过需要预先设置引脚为 ADC 引脚。

ADCCON (ADH) ADC 控制寄存器(读/写)

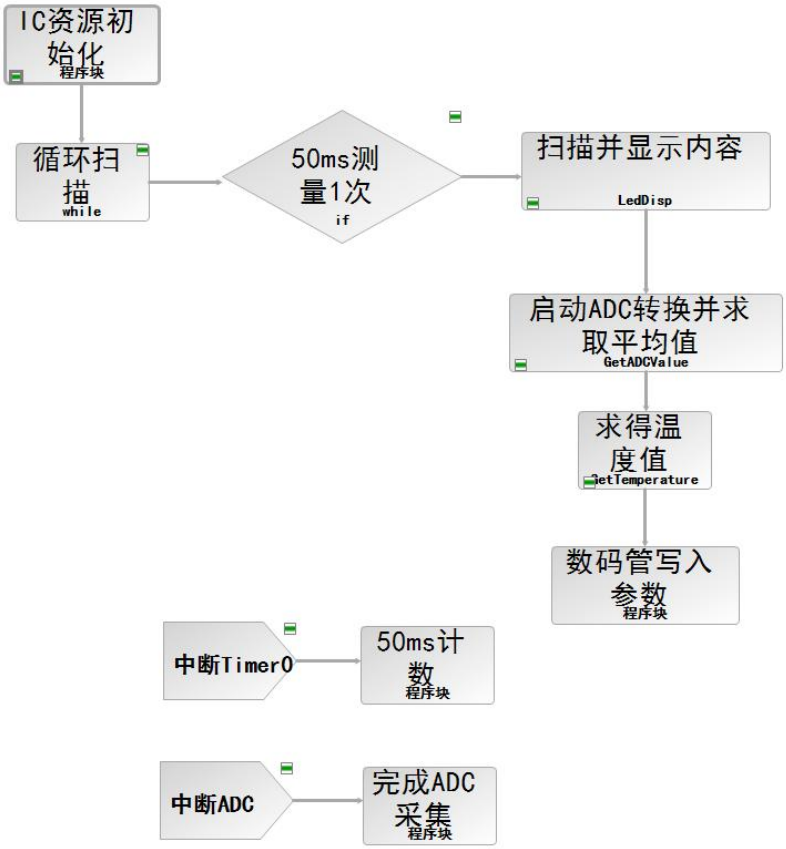
位编号	7	6	5	4	3	2	1	0
符号	ADCEN	ADCS	EOC/ADCIF	ADCIS[4:0]				
读/写	读/写	读/写	读/写	读/写	读/写	读/写	读/写	读/写
上电初始值	0	0	0	0	0	0	0	n

位编号	位符号	说明
7	**ADCEN**	启动 ADC 的电源 0：关闭 ADC 模块电源 1：开启 ADC 模块电源
6	**ADCS**	ADC 开始触发控制（ADC Start） 对此 bit 写 “1”, 开始做一次 ADC 的转换，即该位只是 ADC 转换的触发信号。此位只可写入 1 有效。 **注意：对 ADCS 写“1”后，到中断标志 EOC/ADCIF 置起前不要对 ADCCON 寄存器进行写操作**
5	**EOC /ADCIF**	转换完成/ADC 中断请求标志(End Of Conversion / ADC Interrupt Flag) 0：转换尚未完成 1：ADC 转换完成。需用户软件清除 ADC 转换完成标志 EOC：当使用者设定 ADCS 开始转换后, 此位会被硬件自动清除为 0； 当转换完成后, 此位会被硬件自动置为 1； ADC 中断请求标志 ADCIF： 此位同时也当作是 ADC 中断的中断请求标志，如果用户使能 ADC 中断， 那么在 ADC 的 中断 发生后, 用户必须用软件清除此位。
4~0	**ADCIS[4:0]**	ADC 输入通道选择(ADC Input Selector) 00000：选用 AIN0 为 ADC 的输入 00001：选用 AIN1 为 ADC 的输入 00010：选用 AIN2 为 ADC 的输入 00011：选用 AIN3 为 ADC 的输入 00100：选用 AIN4 为 ADC 的输入 00101：选用 AIN5 为 ADC 的输入 00110：选用 AIN6 为 ADC 的输入 00111：选用 AIN7 为 ADC 的输入 01000：选用 AIN8 为 ADC 的输入 01001：选用 AIN9 为 ADC 的输入 01010：选用 AIN10 为 ADC 的输入 01011：选用 AIN11 为 ADC 的输入 01100：选用 AIN12 为 ADC 的输入 01101：选用 AIN13 为 ADC 的输入 01110：选用 AIN14 为 ADC 的输入 01111：选用 AIN15 为 ADC 的输入 10000~11110：保留
		11111： ADC 输入为 1/4 V_{DD}, 可用于测量电源电压

Preliminary

当芯片资源配置完成后，即可在用户程序配置界面，开始图形化编程。

示例 使用 AIN7 采样热敏电阻 NTC 随温度变化的电压值，然后转化为温度值，在 4 位数码管上显示，温度测量范围为-15~85 度。




```
void main(void)
{
    /*<Generated by EasyCodeCube begin>*/
    /*<UserCodeStart>*//*<SinOne-Tag><3>*/
    SC_Init(); /*** MCU init***/
    /*<UserCodeEnd>*//*<SinOne-Tag><3>*/
    /*<UserCodeStart>*//*<SinOne-Tag><112>*/
    while(1)
    {
        /*<UserCodeStart>*//*<SinOne-Tag><165>*/
        if(T0Flag50ms)
        {
            T0Flag50ms=0;
            /*<UserCodeStart>*//*<SinOne-Tag><31>*/
            LedDisp();
            /*<UserCodeEnd>*//*<SinOne-Tag><31>*/
            /*<UserCodeStart>*//*<SinOne-Tag><162>*/
            GetADCValue();
            /*<UserCodeEnd>*//*<SinOne-Tag><162>*/
            /*<UserCodeStart>*//*<SinOne-Tag><163>*/
            GetTemperature();
            /*<UserCodeEnd>*//*<SinOne-Tag><163>*/
            /*<UserCodeStart>*//*<SinOne-Tag><108>*/
            Led_DisplayData(TEMP%10,TEMP/10,TempSign,16);
            //显示时钟点  DOTDISPLAY:开  DOTUNDISPLAY:关
            Led_DisplayDot(DOTUNDISPLAY,DOTUNDISPLAY,DOTUNDISPLAY,DOTUNDISPLAY);
            /*<UserCodeEnd>*//*<SinOne-Tag><108>*/
            /*<Begin-Inserted by EasyCodeCube for Condition>*/
        }
        /*<UserCodeEnd>*//*<SinOne-Tag><165>*/
        /*<Begin-Inserted by EasyCodeCube for Condition>*/
    }
    /*<UserCodeEnd>*//*<SinOne-Tag><112>*/
    /*<Generated by EasyCodeCube end>*/
}
```

```
void ADCInterrupt()          interrupt 6
{
    /*ADC_it write here begin*/
    /*ADC_it write here*/
    /*<Generated by EasyCodeCube begin>*/
    /*<UserCodeStart>*//*<SinOne-Tag><145>*/
    //ADCInterrupt
    {
        /*<UserCodeStart>*//*<SinOne-Tag><146>*/
        AdcFlag=1;
        /*<UserCodeEnd>*//*<SinOne-Tag><146>*/
        /*<Begin-Inserted by EasyCodeCube for Condition>*/
    }
    /*<UserCodeEnd>*//*<SinOne-Tag><145>*/
    /*<Generated by EasyCodeCube end>*/
    /*ADCInterrupt Flag Clear begin*/
    ADC_ClearFlag();
    /*ADCInterrupt Flag Clear end*/
}
void Timer0Interrupt()       interrupt 1
{
    /*TIM0_it write here begin*/
    TIM0_ModelSetReloadCounter(38869);
    /*TIM0_it write here*/
    /*<Generated by EasyCodeCube begin>*/
    /*<UserCodeStart>*//*<SinOne-Tag><124>*/
    //Timer0Interrupt
    {
        /*<UserCodeStart>*//*<SinOne-Tag><125>*/
        T0Flag50msCount++;
        if(T0Flag50msCount >= 5)
        {
            T0Flag50msCount = 0;
            T0Flag50ms = 1;
        }
        /*<UserCodeEnd>*//*<SinOne-Tag><125>*/
        /*<Begin-Inserted by EasyCodeCube for Condition>*/
    }
    /*<UserCodeEnd>*//*<SinOne-Tag><124>*/
    /*<Generated by EasyCodeCube end>*/
    /*Timer0Interrupt Flag Clear begin*/
    /*Timer0Interrupt Flag Clear end*/
}
void GetADCValue()
{
    /*<UserCodeStart>*//*<SinOne-Tag><150>*/
    static unsigned int i=0;
    ADC_StartConversion();  //启动ADC采样
    if(AdcFlag)
    {
      AdcFlag=0;
      ADC_Value = ADC_GetConversionValue();  //得ADC采样值
      if(ADC_Value >= value_max)
         value_max = ADC_Value;   //取一个最大值
      if(ADC_Value <= value_min)
         value_min = ADC_Value;   //取一个最小值
      ADC_ValueSum = ADC_ValueSum + ADC_Value;  //求和
      i++;
      }
    if(i==10)
    {
      ADC_AverageValue = (ADC_ValueSum-value_max-value_min)/8; //去掉最大最小值求平均值
      ADC_ValueSum=0;   //和清零，重测10次
      i=0;

      value_min=5000;
      value_max=0;
    }

    /*<UserCodeEnd>*//*<SinOne-Tag><150>*/
}
```

```
void GetTemperature()
{
    /*<UserCodeStart>*//*<SinOne-Tag><151>*/
    unsigned int i=0;
    for(i=0;i<100;i++)
    {
     if((ADC_AverageValue >= ADCValueToTemp[i]) && (ADC_AverageValue < ADCValueToTemp[i+1]))
      {
        TEMP = i;            //得转换值对应数组下标
        break;
       }
     }

    if(TEMP<15)
     {
        TempSign = 18;      //负温度显示'-'
        TEMP = 15 - TEMP;       //温度值
     }
     else
     {
        TempSign = 16;      //正温度不显示+
        TEMP =  TEMP-15;        //温度值
     }

    /*<UserCodeEnd>*//*<SinOne-Tag><151>*/
}
```

4.4.4 函数接口及变量

图形化编程使用的具体函数和变量说明如下两表所示：

函数	函数功能	输入参数	返回值	备注
LedDisp ()	LED 扫描显示	无	无	数码管显示程序，和 DDIC_LED 一致
GetADCValue()	得到 ADC 采集平均值		无	测十次 ADC 采样值后，去掉最高和最低值求平均值
GetTemperature()	将 ADC 采样值转为温度值	无	无	通过查表法，将 ADC 采样值与预先计算好标准值对比，求出温度值
ADC_StartConversion()	启动一次 ADC 转换	无	无	魔盒封装好的 BSP 接口
ADC_GetConversionValue()	获得一次 AD 转换数据	无	ADC 转换值	魔盒封装好的 BSP 接口

变量名	说明
ADCValueToTemp[100]	-15~84 度和 ADC 值的关系表
ADC_Value	ADC 采样值
ADC_ValueSum	ADC 采样值和
ADC_AverageValue	ADC 采样平均值
AdcFlag	完成 ADC 转换标志位
TEMP	温度值
TempSign	温度值+-号

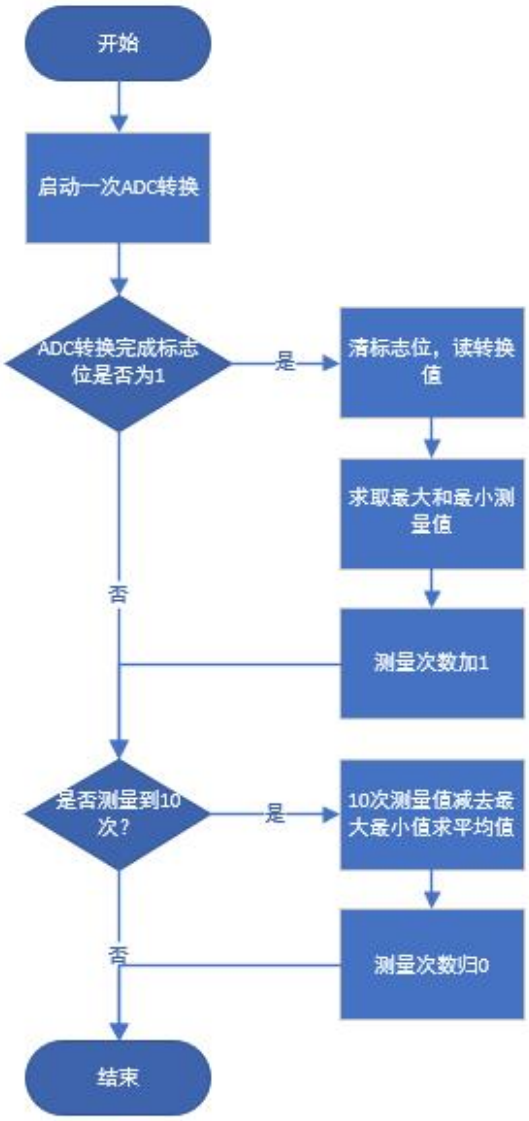
4.4.5 程序流程图

主程序和 ADC 测量函数的程序流程图如下：

ADC 采样温度，通过测量 10 次减去最大最小求平均值的方式，求取 ADC 转换值，然后通过查表法，对比已预先计算好的温度对应转换值，求出温度。ADC 转换完成标志位，需要在 ADC 中断服务函数中自行加入。



主程序



ADC测量函数

4.5 PWM 实现 RGB 灯珠呼吸灯示例

4.5.1 总体描述

PWM 示例：PWM_RGB_NBK1220_EBS002，PWM 驱动 RGB 灯珠实现混色呼吸灯，可以通过魔盒例子工程打开并另存为用户自己的工程。

除此之外，还可以打开示例: PWM_RGB_BREATHINGLIGHT_NBK1220_EBS002，该示例在上面示例的基础上做了一些优化，除了实现呼吸灯的功能之外，可以直接通过按键调节呼吸周期。

NBK1220 核心开发板上的主控芯片最多提供 14 路 PWM，这 14 路 PWM 分为两类：

1．多功能 PWM： 共 8 路，只有一组，即 PWM0，输出信号口为： PWM00~07；

2．常规 PWM：共 6 路，分为三组： PWM2、PWM3、PWM4。注意： **这三组 PWM 的周期寄存器分别与 Timer2，Timer3，Timer4 的 TLX 和 THX 共用，因此一旦用户使用了 PWM2、PWM3、PWM4 资源，就不能再更改 Timer2，Timer3，Timer4 的定时/计数值，否则会导致 PWM 周期输出异常！**

NBK-EBS002 基础功能板、NBK-EBS003 IOT 扩展版上的 RGB 灯珠使用了 PWM0 的资源，用户可以通过魔盒来配置 PWM0 功能，通过控制占空比，来控制颜色和亮度。EBS002、EBS003 使用 RGB 的示例的原理一致，**但是使用 EBS003 的 RGB 功能时请注意：把 NBK1220 底板上的 J10 跳线帽，从下方移到上方，也就是选择~D11 即 PWM06。**

4.5.2 PWM0 主要功能

PWM0 具有的功能如下：

1．16 位 PWM 精度；

2．输出波形可反向；

3．类型：可设为中心对齐型或边沿对齐型；

4．模式：可设为独立模式或互补模式；

a)独立模式下，8 路 PWM 周期相同，但每一路 PWM 输出波形的占空比单独可设置；

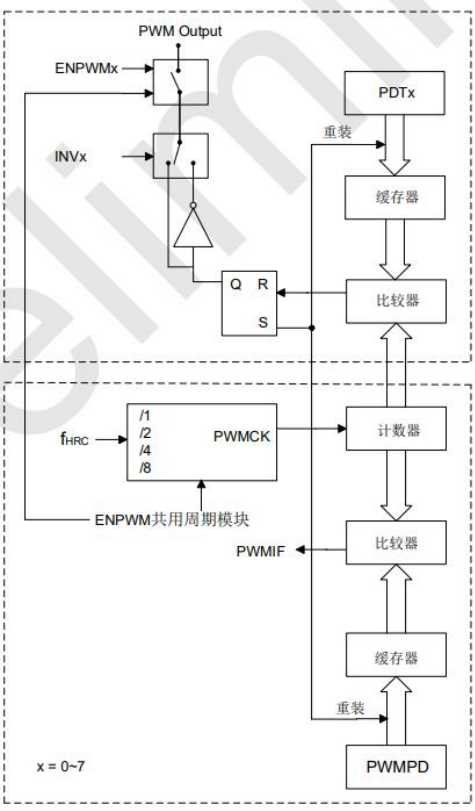
b)互补模式下可同时输出四组互补、带死区的 PWM 波形；

5．提供 1 个 PWM 溢出的中断；

6．支持故障检测机制。

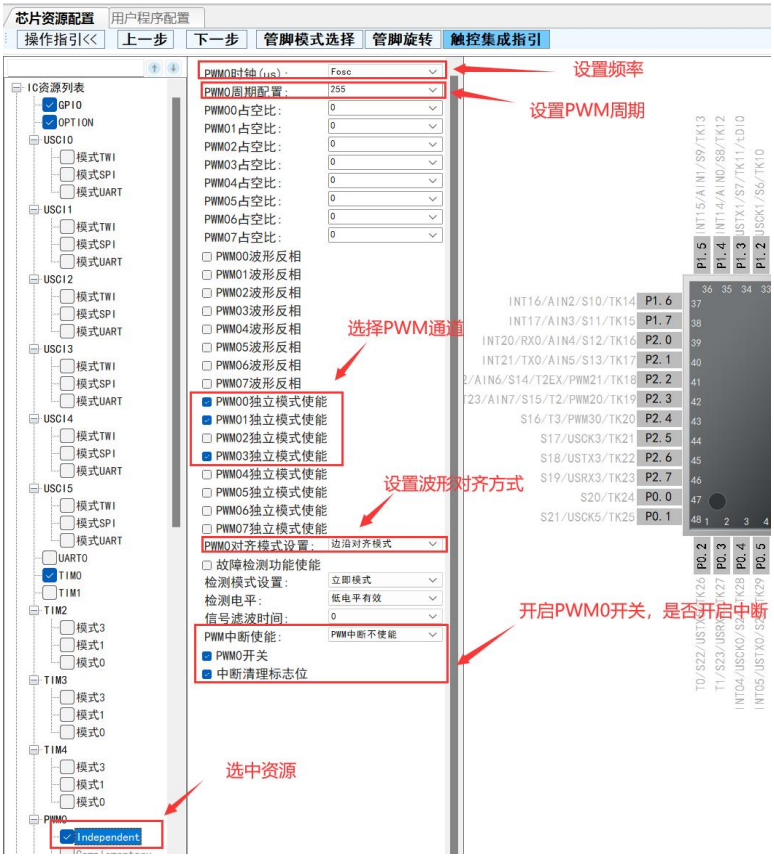
PWM 支持周期及占空比的调整，寄存器 PWMCFG、PWMCON0 和 PWMCON1 控制 PWM 的状态及周期，各路 PWM 的打开及输出波形占空比可单独调整。

4.5.3 PWM 结构框图



4.5.4 在易码魔盒中的配置

使用魔盒，用户可以快捷的配置 PWM 参数：



资源的配置步骤如下：

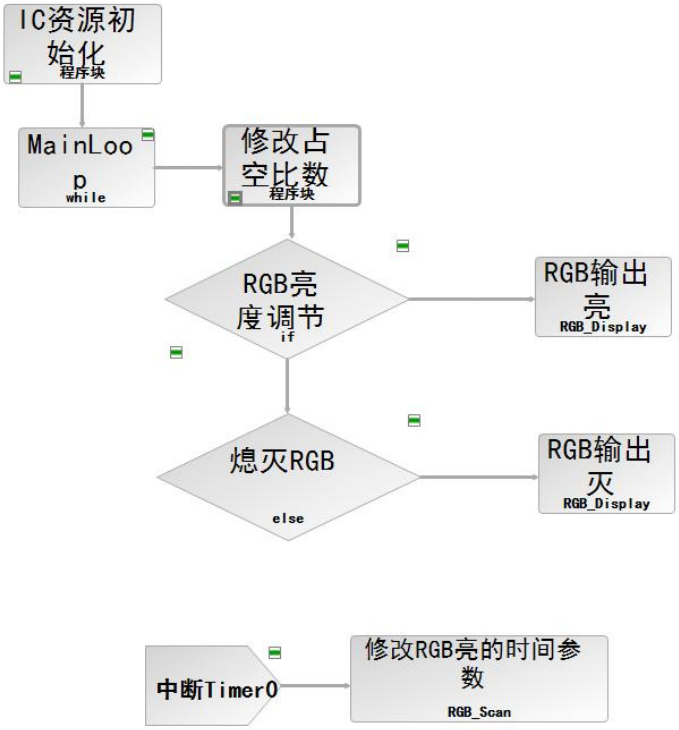
1. 资源在列表选中 PWM0 的 Independent 资源；
2. 设置 PWM0 时钟的频率和控制周期（占空比的调节范围：0-周期数）；
3. 选择要使用的 PWM 通道；
4. 设置 PWM0 输出波形的对齐方式（用户根据需求选择）；
5. 打开 PWM0 的开关，选择是否要打开 PWM 溢出中断，一般选择中断不使能。



完成 PWM 的配置后，还要配置一下定时器，在配置定时器时，要注意定时的时间不能太大，如果定时太大会出现比较明显的闪烁现象。

完成以上资源配置后，在程序中就可以调用魔盒封装好的 PWM 的 BSP 接口，直接控制占空比。

示例 RGB 灯珠混色呼吸灯范例。通过 PWM 控制占空比输出电压，定时器定时调节亮度，进行灯珠颜色转变。




```
void main(void)
{
    /*<Generated by EasyCodeCube begin>*/
    /*<UserCodeStart>*//*<SinOne-Tag><3>*/
    SC_Init(); /*** MCU init***/
    /*<UserCodeEnd>*//*<SinOne-Tag><3>*/
    /*<UserCodeStart>*//*<SinOne-Tag><4>*/
    /*****MainLoop*****/
    while(1)
    {
        /*<UserCodeStart>*//*<SinOne-Tag><124>*/
        PwmCount_Red = ((RGB_TAB[RGB_Num]&0xFF0000)>>16);   //颜色代码23-16
        PwmCount_Green = ((RGB_TAB[RGB_Num]&0x00FF00)>>8);  //颜色代码位15-8
        PwmCount_Blue = (RGB_TAB[RGB_Num]&0x0000FF);        //颜色代码位7-0
        /*<UserCodeEnd>*//*<SinOne-Tag><124>*/
        /*<UserCodeStart>*//*<SinOne-Tag><234>*/
        if(T0FlagCount<NUM)  //通过控制亮的时间控制亮度 NUM: 0-100
        {
            /*<UserCodeStart>*//*<SinOne-Tag><216>*/
            RGB_Display( PwmCount_Red , PwmCount_Green , PwmCount_Blue );
            /*<UserCodeEnd>*//*<SinOne-Tag><216>*/
            /*<Begin-Inserted by EasyCodeCube for Condition>*/
        }
        /*<UserCodeEnd>*//*<SinOne-Tag><234>*/
        /*<UserCodeStart>*//*<SinOne-Tag><235>*/
        else
        {
            /*<UserCodeStart>*//*<SinOne-Tag><246>*/
            RGB_Display( 0 , 0 , 0 );
            /*<UserCodeEnd>*//*<SinOne-Tag><246>*/
            /*<Begin-Inserted by EasyCodeCube for Condition>*/
        }
        /*<UserCodeEnd>*//*<SinOne-Tag><235>*/
        /*<Begin-Inserted by EasyCodeCube for Condition>*/
    }
    /*<UserCodeEnd>*//*<SinOne-Tag><4>*/
    /*<Generated by EasyCodeCube end>*/
}
```

```
void Timer0Interrupt()      interrupt 1
{
    /*TIM0_it write here begin*/
    TIM0_ModelSetReloadCounter(59136);
    /*TIM0_it write here*/
    /*<Generated by EasyCodeCube begin>*/
    /*<UserCodeStart>*//*<SinOne-Tag><129>*/
    //Timer0Interrupt
    {
        /*<UserCodeStart>*//*<SinOne-Tag><231>*/
        RGB_Scan();
        /*<UserCodeEnd>*//*<SinOne-Tag><231>*/
        /*<Begin-Inserted by EasyCodeCube for Condition>*/
    }
    /*<UserCodeEnd>*//*<SinOne-Tag><129>*/
    /*<Generated by EasyCodeCube end>*/
    /*Timer0Interrupt Flag Clear begin*/
    /*Timer0Interrupt Flag Clear end*/
}
void RGB_Scan()
{
    /*<UserCodeStart>*//*<SinOne-Tag><202>*/
    T0FlagCount++;
    if(T0FlagCount>=100)
    {
        T0FlagCount=0;
        if(NUM_Flag==0)
        {
            NUM++;
        }
        else
        {
            NUM--;
        }
        if(NUM==100) //到达亮度顶峰
        {
            NUM_Flag=1;
            RGB_NumFlag=1;
        }
        if(NUM==0)  //RGB熄灭
        {
            NUM_Flag=0;
        }
        if(RGB_NumFlag==1&&NUM==0)  //RGB亮灭之后颜色转换
        {
            RGB_NumFlag=0;
            RGB_Num++;
            if(RGB_Num==RGB_ColorNum) RGB_Num=0;  //共有7种颜色
        }
    }

    /*<UserCodeEnd>*//*<SinOne-Tag><202>*/
}
```

4.5.5 函数接口及变量

图形化编程使用的具体函数和变量说明如下两表所示：

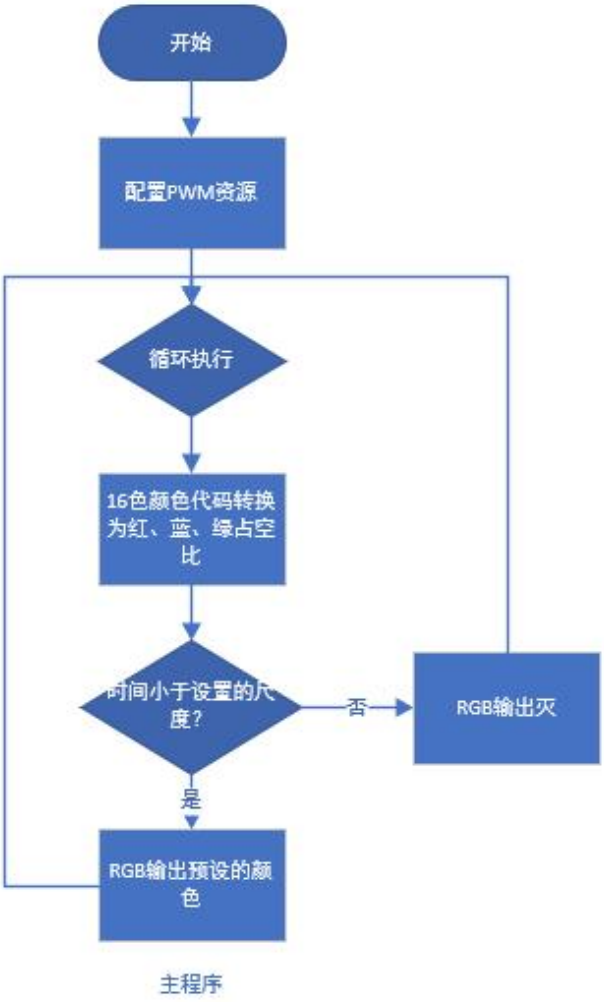
函数	函数功能	输入参数	返回值	备注
RGB_Display ()	改变 PWM 通道占空比	无	无	PWM 通道要对应上
RGB_Scan ()	修改 RGB 亮的时间参数	无	无	修改 Num 的值，在灯亮灭之后切换颜色数组下标
PWM_IndependentModeConfig(PWM_OutputPin_TypeDef PWM_OutputPin, uint16_t PWM_DutyCycle)	PWMx 独立工作模式配置函数	PWM_OutputPin：PWMx 独立通道选择 PWM_DutyCycle：PWM 占空比配置	无	魔盒封装好的 BSP 接口

变量名	说明
RGB_TAB[]	16 色颜色代码
NUM	亮度值，范围：0-100
PwmCount_Blue	蓝色占空比
PwmCount_Green	绿色占空比
PwmCount_Red	红色占空比
RGB_Num	颜色代码下标

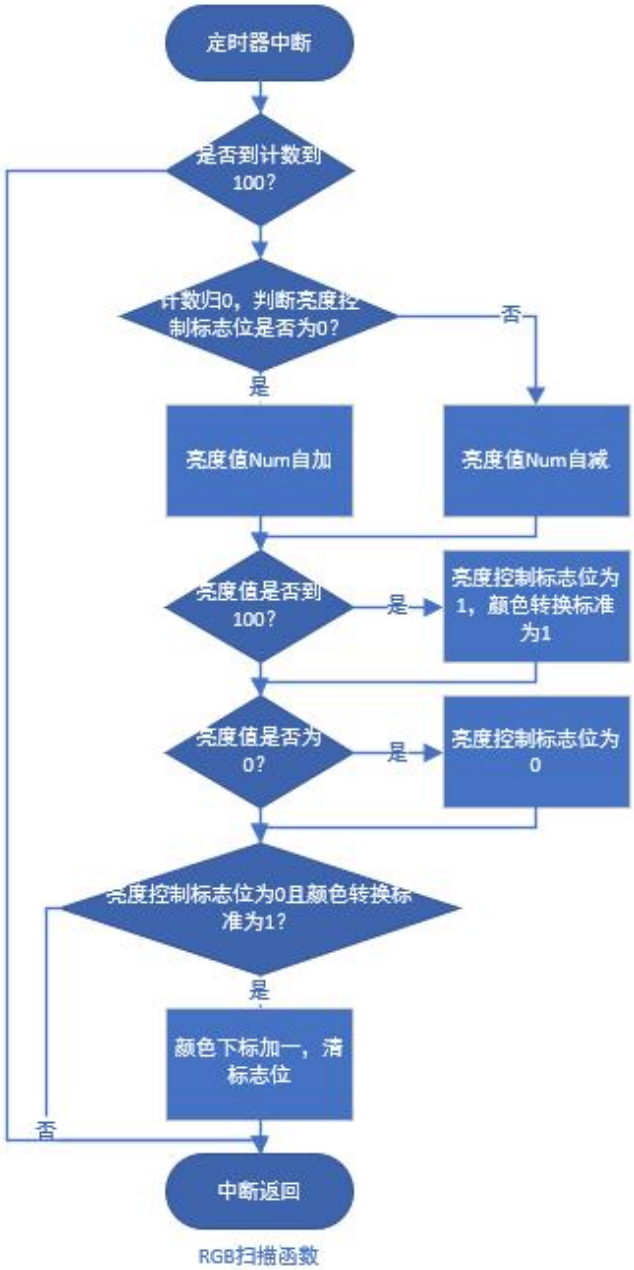
4.5.6 程序流程图

主程序和 RGB 扫描函数的程序流程图如下：

主程序输出控制占空比，扫描函数放在定时器中执行，通过控制 RGB 颜色在周期内亮灭的时间，来到达控制它亮度的目的。在移植程序时，要注意扫描函数占总体程序运行的时间，不然会出现比较明显的亮灭现象。



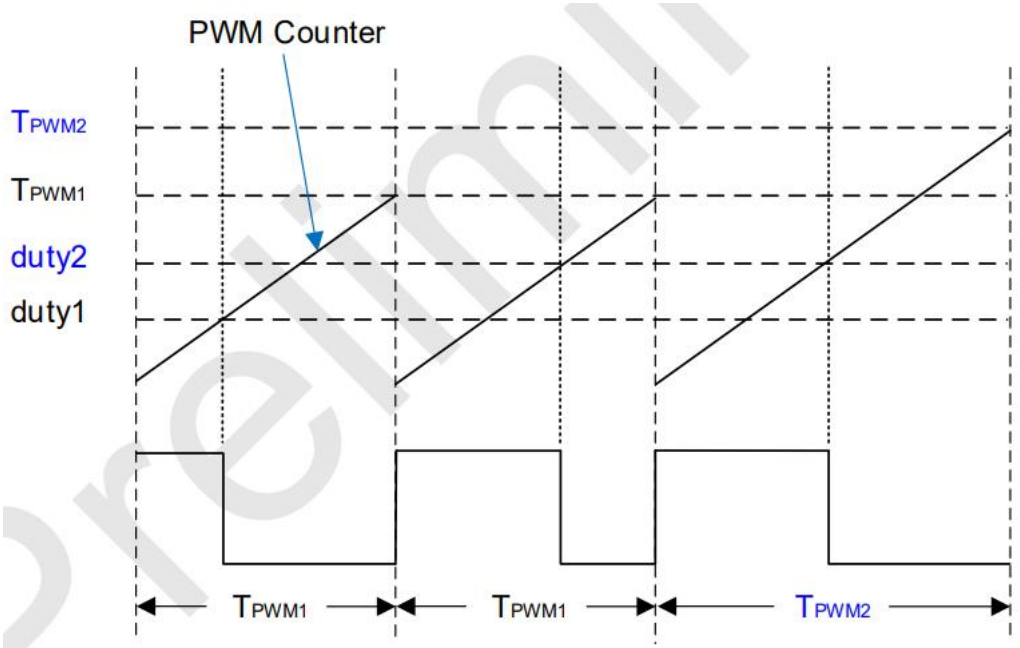
主程序



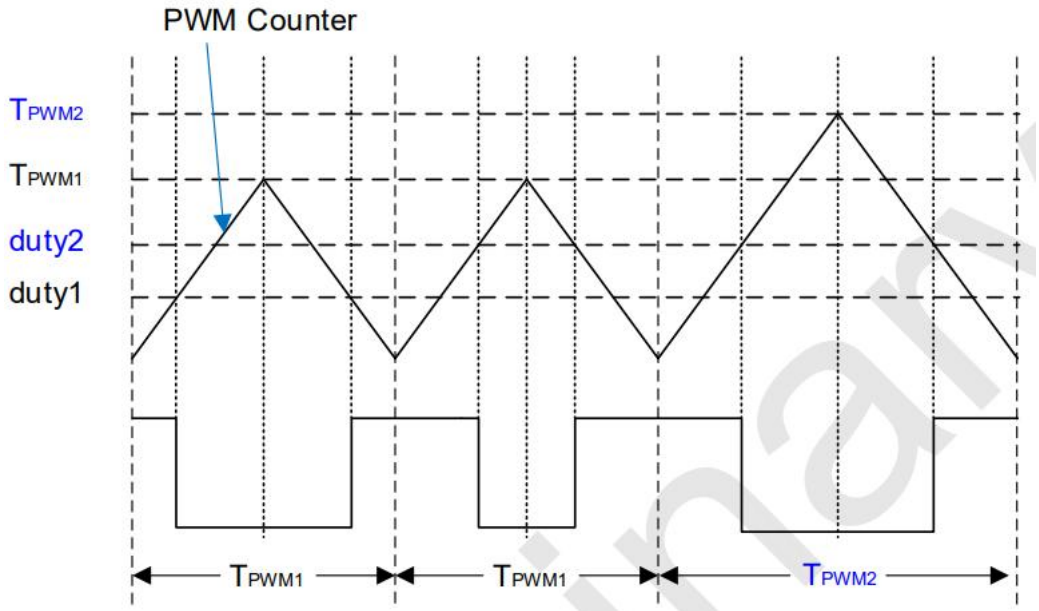
RGB扫描函数

4.5.7 PWM 波形

PWM 输出的波形一共有两种，分别是边沿对齐型和中心对齐型，用户可以根据需求，在魔盒资源配置界面自行选择。



边沿对齐的 PWM



中心对齐的 PWM

4.6 硬件 TWI 驱动 OLED 显示示例

4.6.1 总体描述

TWI 使用示例：USCI0_TWI_OLED_NBK1220_EBS003，可以通过魔盒例子工程打开并另存为用户自己的工程。

TWI 协议是嵌入式开发中常用的一种总线协议，使用方便，占用 IO 少，可以在同一个主机总线上挂多个从设备。NBK1220 核心开发板上的主控芯片有六组 UART/SPI/TWI 三选一通信口 USCI，用户可以通过配置 TMCON 寄存器的 USMDn 来配置 USCI 为 TWI 接口：

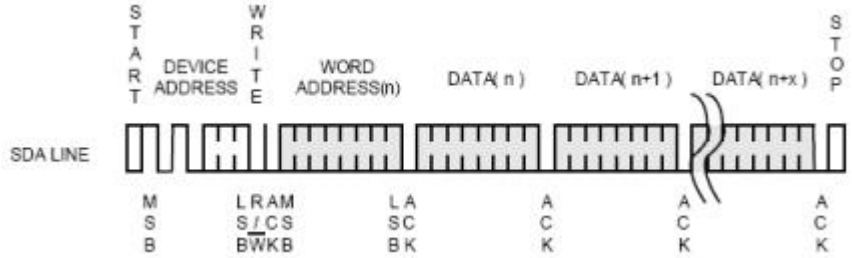
- USTXn 作为 SDA 信号
- USCKn 作为 CLK 信号

在 TW 通信时可根据应用需要设定为主机或从机模式。本示例使用模块为 4 脚 SSD1306 驱动 的 0.96 寸蓝光 OLED 模块，由于本模块仅需要主机 TWI 写功能，如用户需要了解 TWI 读操作或者 TWI 从机读写功能，请浏览魔盒中其他关于 TWI 通信的文档。

4.6.2 OLED 驱动原理

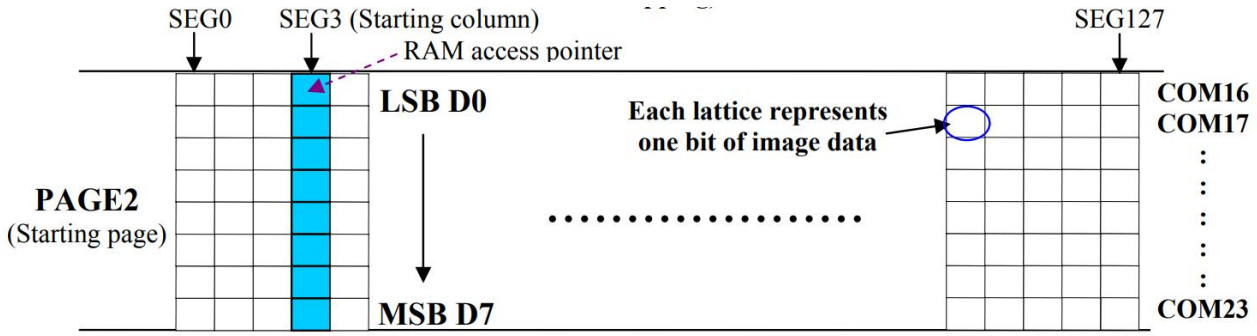
4 脚 0.96 寸 OLED 使用的是 TWI 协议进行通信，而且主要是写数据，要使用到写时序，时序如下：

写时序：首先发送启动信号，主机发送从设备地址+写命令，从机应答，应答成功，表示有这个设备，然后主机发送设备内部寄存器地址，此时不用再加写命令控制字，从机应答，应答成功，表示设备内有这个地址，主机写入数据，从机应答，是否继续发送，不发送的话，发送停止信号。



4 脚 0.96 寸 OLED 的显存地址与数据排列如下：

	COL0	COL 1	…..	COL 126	COL 127
PAGE0	→	→	→	→	→
PAGE1	→	→	→	→	→
:	:	:	:	:	:
PAGE6	→	→	→	→	→
PAGE7	→	→	→	→	→



整个屏幕水平方向划分为 8 个 page，垂直方向则是按像素划分为 128 column。每个 page-column 包含 8 个像素，通过一个十六进制数（其实就是一个字节,，8 个 bit）来控制，每个 bit 控制一个像素，数据的低位对应低坐标点。通过写命名，定位坐标点和字体的库，即可在相应的位置写入数据。

4.6.3 TWI 主机操作步骤

1．配置 USMDn[1:0]， 选择 TWI 模式；

2．配置 TWIn 控制寄存器 USnCON0：TWEN= 1，使能 TWI；

3．配置 TWIn 控制寄存器 USnCON1：配置 TWI 通信速率（TWCK[3:0]），将起始位 STA 置“1”；

4．配置 TWIn 地址寄存器 USnCON3：将“从机地址+读写位”写入 TWDAT，总线上发出地址帧；

5．如果主机接收数据， 则等待 USnCON0 中的中断标志位 TWIF 置 1。主机每接收到 8 位数据，中断标志位会被置 1，中断标志位需手动清零；

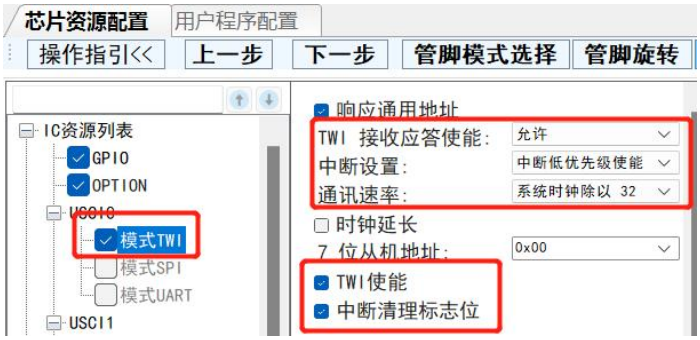
6．如果主机发送数据，则要将待发送的数据写进 TWDAT 中，TWI 会自动将数据发送出去。每发送 8 位，中断标志位 TWIF 就会被置 1。

7．数据接收发送完成，主机可发送停止条件（STO=1），主机状态切换为 000。 或发送重复起始信号，开始新一轮的数据传输。

注意：用户在使用中断的方式编辑代码使用硬件 TWI 硬件通信时，在主机发送启动条件前，请清除一次中断条件 TWIF 和用户自定义的中断标志位。

4.6.4 在魔盒中的配置

使用易码魔盒配置定时器 USCI0 的资源，在易码魔盒的芯片资源配置界面进行如图所示配置



资源配置步骤如下：

1． 在 IC 资源列表中，选中 USCI0 的 TWI 模式，开始配置；

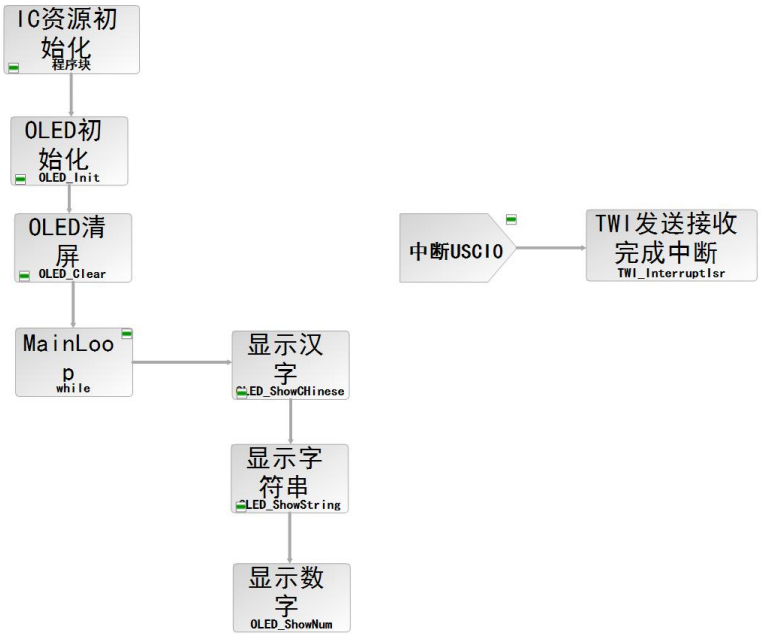
2． 配置 TWI 接收应答使能为允许（接收或者作为从机时，必须设置为允许）；

3． 中断优先级为低，通讯速率为 32；

4． 使能 TWI 和设置中断清理标志位。

上述配置中，通讯速率由于各器件响应主机信号速度的不同，所以不能设置太高，选择一个合适的速率即可。因为网上关于 0.96 寸 OLED 的资料较多，所以在本节中就不再全部举例说明该模块的函数使用说明，只列出使用本司芯片硬件 TWI 通信的基础函数和显示一个字符的函数，在实际使用 TWI 通信中推荐使用魔盒自带的 BSP 接口进行通信。

示例 使用芯片硬件 USCI0 的 TWI 资源，在 4 脚 OLED 模块上进行显示。



```
void main(void)
{
    /*<Generated by EasyCodeCube begin>*/
    /*<UserCodeStart>*//*<SinOne-Tag><3>*/
    SC_Init(); /*** MCU init***/
    /*<UserCodeEnd>*//*<SinOne-Tag><3>*/
    /*<UserCodeStart>*//*<SinOne-Tag><1801>*/
    OLED_Init();
    /*<UserCodeEnd>*//*<SinOne-Tag><1801>*/
    /*<UserCodeStart>*//*<SinOne-Tag><1802>*/
    OLED_Clear();
    /*<UserCodeEnd>*//*<SinOne-Tag><1802>*/
    /*<UserCodeStart>*//*<SinOne-Tag><4>*/
    /*****MainLoop*****/
    while(1)
    {
        /*<UserCodeStart>*//*<SinOne-Tag><1804>*/
        OLED_ShowCHinese( 20 , 0 , 0 );
        OLED_ShowCHinese( 38 , 0 , 1 );
        OLED_ShowCHinese( 56 , 0 , 2 );
        OLED_ShowCHinese( 74 , 0 , 3 );
        OLED_ShowCHinese( 92 , 0 , 4 );
        /*<UserCodeEnd>*//*<SinOne-Tag><1804>*/
        /*<UserCodeStart>*//*<SinOne-Tag><1805>*/
        OLED_ShowString(10,2,CharData,16);
        /*<UserCodeEnd>*//*<SinOne-Tag><1805>*/
        /*<UserCodeStart>*//*<SinOne-Tag><1806>*/
        OLED_ShowNum(40,4,202212,6,16);
        /*<UserCodeEnd>*//*<SinOne-Tag><1806>*/
        /*<Begin-Inserted by EasyCodeCube for Condition>*/
    }
    /*<UserCodeEnd>*//*<SinOne-Tag><4>*/
    /*<Generated by EasyCodeCube end>*/
}
```

```
////TWI中断服务函数
 void TWI_InterruptIsr()
{
    /*<UserCodeStart>*//*<SinOne-Tag><1371>*/
     if(US0CON0&0X40)     //TWI中断标志位
     {
      US0CON0 &= 0Xbf;     //将TWI清零
      TWI0Flag = 1;
      }

    /*<UserCodeEnd>*//*<SinOne-Tag><1371>*/
}
```

```
 bool TWI_Start()
{
    /*<UserCodeStart>*//*<SinOne-Tag><1331>*/
    bit Start_flag=1;
    unsigned int i=1000;
    US0CON1 |= 0x20;
    while(!TWI0Flag)    //等待启动信号结束
    {
        i--;
        if(i==0)
        {
            break;
        }
    }
    TWI0Flag=0;
    if(i==0)
    {
        Start_flag=0;
    }
    return     Start_flag;

    /*<UserCodeEnd>*//*<SinOne-Tag><1331>*/
}
////停止信号
 void TWI_Stop()
{
    /*<UserCodeStart>*//*<SinOne-Tag><1347>*/
    Delay(10);
    US0CON1 |= 0x10;
    Delay(10);

    /*<UserCodeEnd>*//*<SinOne-Tag><1347>*/
}
////等待应答
 bool TWI_WaitAck()
{
    /*<UserCodeStart>*//*<SinOne-Tag><1343>*/
    unsigned int i=1000;
    for(;i>0;i--)
    {
    if((US0CON1&0x80))
        {
      return 1;
        }
     }
     return 0;

    /*<UserCodeEnd>*//*<SinOne-Tag><1343>*/
}
```

```
////接收一个字节数据
 unsigned char TWI_ReceiveByte()
{
    /*<UserCodeStart>*//*<SinOne-Tag><1339>*/
  unsigned int i=1000;
  unsigned char TWI_ReceiveByte=0;
  while(!TWI0Flag)
    {
        i--;
        if(i==0)
        {
            break;
        }
    }
  TWI0Flag=0;
  TWI_ReceiveByte = US0CON3;
  return TWI_ReceiveByte;

    /*<UserCodeEnd>*//*<SinOne-Tag><1339>*/
}
////发送一个字节数据
 void TWI_SendByte(unsigned int SendByte)
{
    /*<UserCodeStart>*//*<SinOne-Tag><1335>*/
    unsigned int i=1000;
    US0CON3 = SendByte;
    while(!TWI0Flag)    //等待发送完成
    {
        i--;
        if(i==0)
        {
            break;
        }
     }
    TWI0Flag=0;

    /*<UserCodeEnd>*//*<SinOne-Tag><1335>*/
}
```

```
u8 TWI_WriteOneByte(u8 DeviceAddr,u8 DataAddr,u8 u8Data)
{
    /*<UserCodeStart>*//*<SinOne-Tag><1392>*/
      bit Error_Flag=0;
      US0CON0 &= 0xF7;    //AA=0;
      Delay(10);
      US0CON0 &= 0X7F;    //开关一次TWI，防止上一次传输影响
      US0CON0 |= 0x80;
      Delay(10);

      TWI0Flag=0;      //回复状态
      Error_Flag=TWI_Start();
      if(Error_Flag==0)
      {
        return Error_Flag;    //启动失败，退出
       }
      Error_Flag=0;
      TWI_SendByte(DeviceAddr);  //设备地址
      if(TWI_WaitAck()==1)
      {
         TWI_SendByte(DataAddr);    //数据地址
         if(TWI_WaitAck()==1)
         {
            TWI_SendByte(u8Data);
            if(TWI_WaitAck()==1)
            {
               Error_Flag=1;
            }
          }
       }
      TWI_Stop();
      return Error_Flag;

    /*<UserCodeEnd>*//*<SinOne-Tag><1392>*/
}
void OLED_ShowChar(u8 x,u8 y,u8 chr,u8 Char_Size)
{
    /*<UserCodeStart>*//*<SinOne-Tag><75>*/
      unsigned char c=0,i=0;
      c=chr-' ';//得到偏移后的值
      if(x>Max_Column-1)
      {
        x=0;
        y=y+2;
       }
      if(Char_Size ==16)
      {
          OLED_Set_Pos(x,y);
          for(i=0;i<8;i++)
          {
            TWI_WriteOneByte(0x78,0x40,F8X16[c*16+i]);
          }
          OLED_Set_Pos(x,y+1);
          for(i=0;i<8;i++)
          {
            TWI_WriteOneByte(0x78,0x40,F8X16[c*16+i+8]);
           }
      }
      else
      {
          OLED_Set_Pos(x,y);
          for(i=0;i<6;i++)
          {
            TWI_WriteOneByte(0x78,0x40,F6x8[c][i]);
           }
      }

    /*<UserCodeEnd>*//*<SinOne-Tag><75>*/
}
```

4.6.5 函数接口及变量

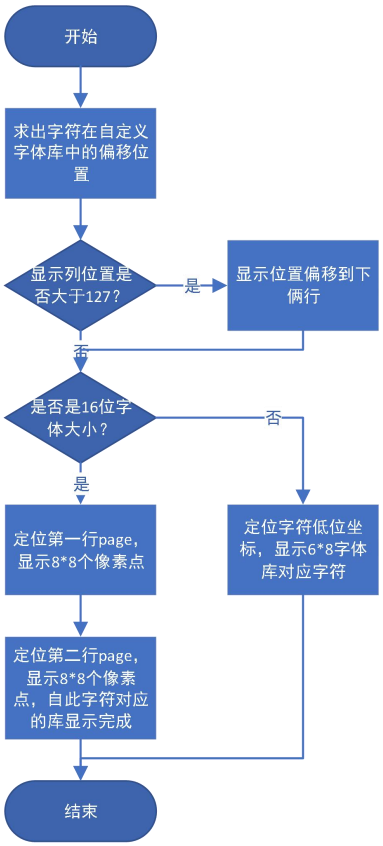
图形化编程使用的具体函数和变量说明如下两表所示：

函数	函数功能	输入参数	返回值	备注
TWI_InterruptIsr()	TWI 中断服务函数	无	无	放入 USCI 中断中执行
TWI_Start ()	启动信号	无	无	无
TWI_Stop ()	停止信号	无	无	无
TWI_WaitAck()	等待应答信号	无	无	无
TWI_SendByte (unsigned int SendByte)	发送一字节	无	无	无
TWI_ReceiveByte()	接收一字节	无	接收的数据	无
TWI_WriteOneByte (u8 DeviceAddr,u8 DataAddr,u8 u8Data)	向某一从设备易地址内写入一字节	无	发送状态	无
OLED_ShowChar(u8 x,u8 y,u8 chr,u8 Char_Size)	在 OLED 指定位置显示一个字符	x：列，0~127 y：行，0~7 chr：字符 Size：显示字体大小，16 或者其他（8*16 或者 6*8）	无	无

变量名	说明
TWI0Flag	TWI0 中断标志位
Max_Column	OLED 屏最大列数

4.6.6 程序流程图

下图为在 OLED 指定位置显示一个字符函数的程序流程图：



4.7 定时器捕获和外部中断测周期脉宽示例

4.7.1 总体描述

示例工程 TIMER_INT_CAPTUREMODE_NBK1220_EBS003，是外部中断+定时器捕获实现测周期和脉宽的例程，可以通过魔盒例子工程打开并另存为用户自己的工程。

对于控制系统来说，知道脉冲信号的周期和脉宽十分重要，本示例主要是利用芯片的定时器捕获功能和外部中断功能，测量脉冲信号的脉宽及周期。本司的芯片，主要有 5 个定时器，有捕获功能的定时器是 Timer2/3/4，其中 Timer2 有 4 种工作模式，Timer3 和 Timer4 有 3 种工作模式。这 3 个定时器是独立的，但是 Timer2/3/4 的控制寄存器共用同一组地址，用户可通过 TXINX[2:0]将 TimerX 寄存器组(TXCON / TXMOD / RCAPXL / RCAPXH / TLX / THX)指向 Timer2/3/4，从而实现一组寄存器配置三个独立 Timer 的功能。

注意：只有在 TXINX[2:0]配置成功后 TimerX 寄存器组才会指向用户指定的 Timer2/3/4，此时操作 TimeX 寄存器组才是对相应 Timer 的有效操作。

外部中断有 INT0~2，当外部中断口有中断条件发生时，外部中断就发生了。其中 INT0 和 INT1 会产生中断标志为 IE0/IE1，用户不需要处理，硬件会自动清除。INT0 有四个外部中断源，INT1 有八个外部中断源，INT2 有四个外部中断源，用户可以根据需要设成上沿、下沿或者双沿中断，可通过设置 SFR （INTxF 和 INTxR）来实现。用户可通过 IP 寄存器来设置每个中断的优先级级别。外部中断 INT0~2 还可以唤醒单片机的 STOP。

由于定时器捕获功能只有下降沿捕获，又为了节约引脚的使用，所以本示例主要是使用定时器 2 和外部中断 INT22 实现周期和脉宽的测量，定时器 2 的捕获引脚和外部中断 INT22 的引脚为同一个，这样只使用一个引脚便可以实现测量周期和脉宽的功能。本例程，主要使用 NBK1220 核心开发板测量，并在 EBS003 IOT 扩展版的 OLED 模块上显示。

4.7.2 定时器 2 主要功能

定时器 Timer2 具有计数方式和定时方式两种工作模式。特殊功能寄存器 TXCON 中有一个控制位 C/TX 来选择 T2 是定时器还是计数器。它们本质上都是一个加法计数器，只是计数的来源不同。定时器的来源为系统时钟或者其分频时钟，但计数器的来源为外部管脚的输入脉冲。TRX 是 T2/T3/T4 在定时器/计数器模式计数的开关控制，只有在 TRX=1 的时候，T2 才会被打开计数。

计数器模式下，T2 管脚上的每一个脉冲，T2 的计数值分别增加 1。

定时器模式下，可通过特殊功能寄存器 TXMOD.7(TXFD)来选择 T2 的计数来源是 fsys/12 或 fsys。

定时器/计数器 T2 有 4 种工作模式：

① 模式 0：16 位捕获模式
② 模式 1：16 位自动重载定时器模式
③ 模式 2：波特率发生器模式
④ 模式 3：可编程时钟输出模式

测量周期和脉宽主要是使用定时器 2 的模式 0，16 位捕获模式。

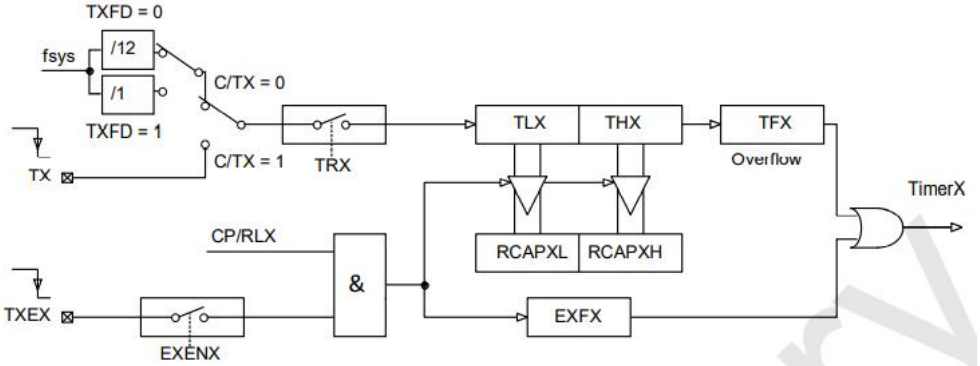
工作模式 0 说明：

配置 CP/RLX=1，将定时器 n (n=2~4)设置为 16 位捕获模式。

在捕获方式中，TXCON 的 EXENX 位有两个选项：

如果 EXENX=0，定时器 n 作为 16 位定时器或计数器，如果 ETn 被允许的话，定时器 n 能设置 TFX 溢出产生一个中断。

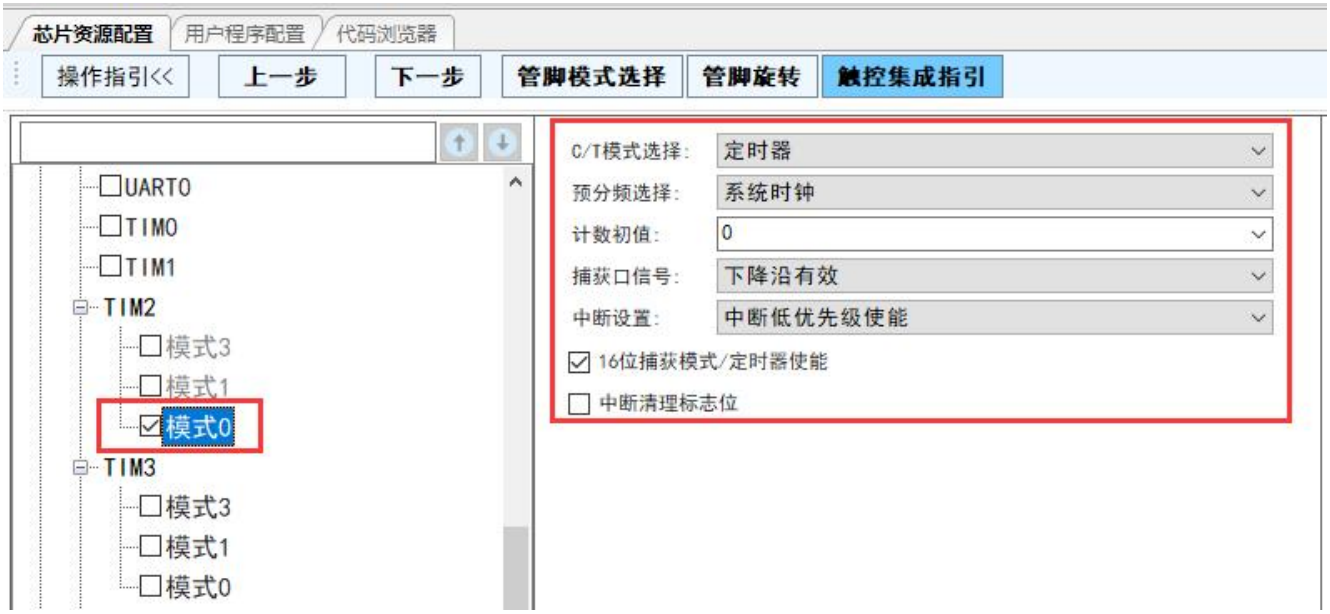
如果 EXENX=1，定时器 n 执行相同操作，但是在外部输入 TnEX 上的下降沿也能引起在 THX 和 TLX 中的当前值分别被捕获到 RCAPXH 和 RCAPXL 中，此外，在 TnEX 上的下降沿也能引起在 TXCON 中的 EXFX 被设置。如果 ETn 被允许，EXFX 位也像 TFX 一样也产生一个中断。



模式 0: 16 位捕获

4.7.3 在易码魔盒中的配置

使用易码魔盒配置定时器 TIM2 的资源，在易码魔盒的芯片资源配置界面进行如图所示配置，由于定时器 2 捕获模式和外部中断 INT22 配置冲突互斥，无法同时设置的缘故，所以不用在资源配置界面进行外部中断的配置，外部中断 INT22 的配置放到定时器捕获后的代码中进行。

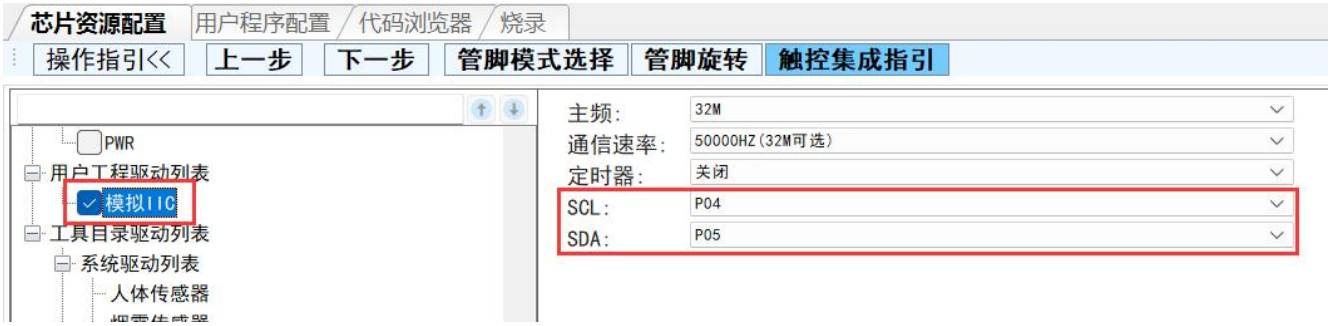


资源的配置步骤如下：

1. 在列表勾选 TIM2 的模式 0 资源；
2. 功能选择为定时器；
3. 预分频选择根据用户需求选择，一般最快选择系统时钟；
4. 计数初值设置为 0；
5. 捕获口信号设置为下降沿有效；
6. 中断设置为低优先级使能；
7. 勾选 16 位捕获功能模式/定时器使能，取消勾选中断清理标志位。

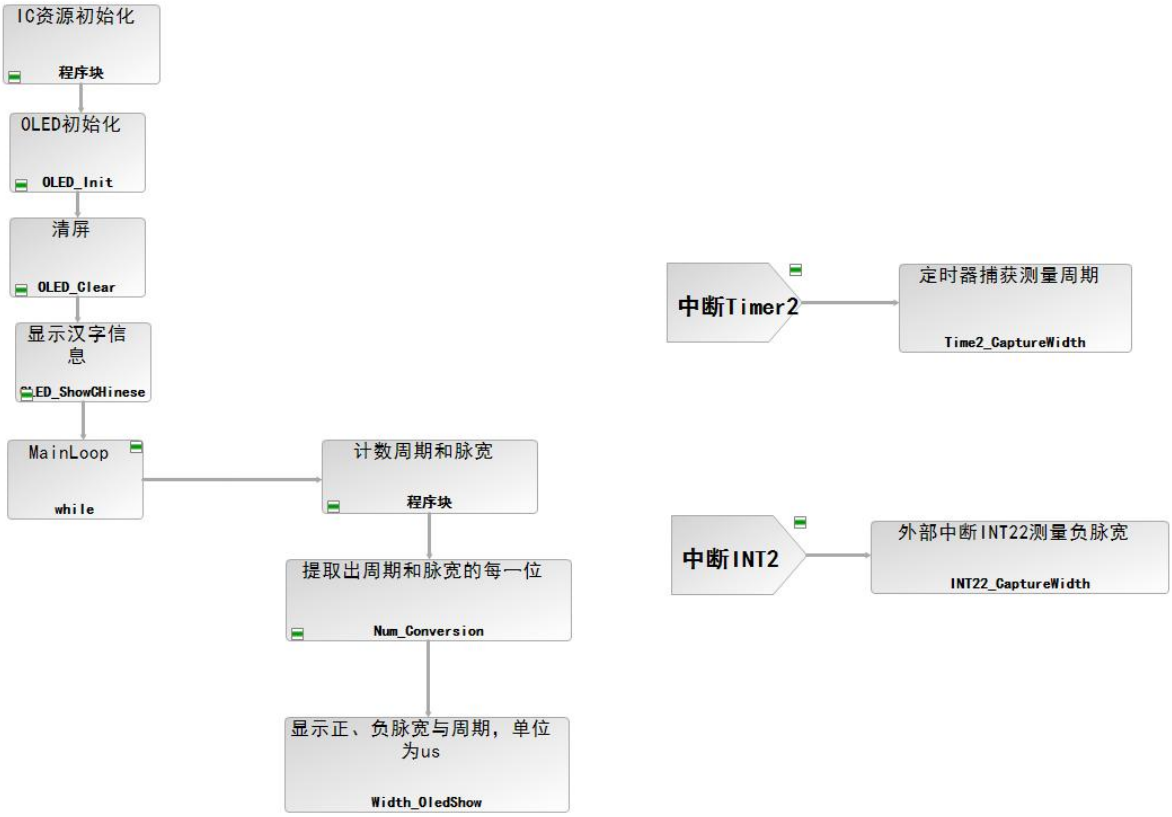
上述步骤中，步骤 3 预分频选择，用户可自行选择，选择后通过计算并更改宏定义 FSYSHRC 的值，即可正确计算时钟周期。步骤 6，中断设置为低优先级使能主要是为了防止在外部中断来时被定时器计数溢出中断打断，导致计算有误差。步骤 7，取消勾选中断清理标志位主要是为自行控制清理标志位，更好的控制定时器溢出中断和捕获中断同时来，给测量带来的误差。

在配置完 TIMER2 的资源后，还需要配置一下用于 OLED 显示的资源，在工具目录下导入模拟 IIC 到用户工程驱动列表，在界面中关闭定时器，选择 SCL 引脚为 P04，SDA 引脚为 P05 即可。



当芯片资源配置完成后，即可在用户程序配置界面，开始图形化编程。

示例 使用定时器 2 和外部中断测量脉宽和周期，在 OLED 显示屏上显示正脉宽、负脉宽、周期等信息，测量显示的单位为 us。




```
void main(void)
{
    /*<Generated by EasyCodeCube begin>*/
    /*<UserCodeStart>*//*<SinOne-Tag><3>*/
    SC_Init(); /*** MCU init***/
    /*<UserCodeEnd>*//*<SinOne-Tag><3>*/
    /*<UserCodeStart>*//*<SinOne-Tag><82>*/
    OLED_Init();
    /*<UserCodeEnd>*//*<SinOne-Tag><82>*/
    /*<UserCodeStart>*//*<SinOne-Tag><83>*/
    OLED_Clear();
    /*<UserCodeEnd>*//*<SinOne-Tag><83>*/
    /*<UserCodeStart>*//*<SinOne-Tag><383>*/
    OLED_ShowCHinese( 0 , 0 , 0 );
    OLED_ShowCHinese( 18 , 0 , 1 );
    OLED_ShowCHinese( 36 , 0 , 2 );
    OLED_ShowCHinese( 0 , 2 , 3 );
    OLED_ShowCHinese( 18 , 2 , 1 );
    OLED_ShowCHinese( 36 , 2 , 2 );
    OLED_ShowCHinese( 0 , 4 , 4 );
    OLED_ShowCHinese( 18 , 4 , 5 );
    /*<UserCodeEnd>*//*<SinOne-Tag><383>*/
    /*<UserCodeStart>*//*<SinOne-Tag><4>*/
    /*****MainLoop*****/
    while(1)
    {
        /*<UserCodeStart>*//*<SinOne-Tag><872>*/
        NegativeWidth=FSYSHRC*NegaWidth[TimeIndex];      //计算负脉宽
        Cycle=FSYSHRC*CycleWidth[TimeIndex];      //计算周期
        PositiveWidth=Cycle-NegativeWidth;      //计算正脉宽
        /*<UserCodeEnd>*//*<SinOne-Tag><872>*/
        /*<UserCodeStart>*//*<SinOne-Tag><835>*/
        Num_Conversion( PositiveWidth , NegativeWidth , Cycle );
        /*<UserCodeEnd>*//*<SinOne-Tag><835>*/
        /*<UserCodeStart>*//*<SinOne-Tag><843>*/
        Width_OledShow( PWidth , NWidth , Cyc );
        /*<UserCodeEnd>*//*<SinOne-Tag><843>*/
        /*<Begin-Inserted by EasyCodeCube for Condition>*/
    }
    /*<UserCodeEnd>*//*<SinOne-Tag><4>*/
    /*<Generated by EasyCodeCube end>*/
}
```

```
void Timer2Interrupt()      interrupt 5
{
    /*TIM2_it write here begin*/
    /*TIM2_it write here*/
    /*<Generated by EasyCodeCube begin>*/
    /*<UserCodeStart>*//*<SinOne-Tag><309>*/
    //Timer2Interrupt
    {
        /*<UserCodeStart>*//*<SinOne-Tag><861>*/
        Time2_CaptureWidth();
        /*<UserCodeEnd>*//*<SinOne-Tag><861>*/
        /*<Begin-Inserted by EasyCodeCube for Condition>*/
    }
    /*<UserCodeEnd>*//*<SinOne-Tag><309>*/
    /*<Generated by EasyCodeCube end>*/
    /*Timer2Interrupt Flag Clear begin*/
    /*Timer2Interrupt Flag Clear end*/
}
void INT2Interrupt()        interrupt 10
{
    /*INT2_it write here begin*/
    /*INT2_it write here*/
    /*<Generated by EasyCodeCube begin>*/
    /*<UserCodeStart>*//*<SinOne-Tag><310>*/
    //INT2Interrupt
    {
        /*<UserCodeStart>*//*<SinOne-Tag><867>*/
        INT22_CaptureWidth();
        /*<UserCodeEnd>*//*<SinOne-Tag><867>*/
        /*<Begin-Inserted by EasyCodeCube for Condition>*/
    }
    /*<UserCodeEnd>*//*<SinOne-Tag><310>*/
    /*<Generated by EasyCodeCube end>*/
    /*INT2Interrupt Flag Clear begin*/
    /*INT2Interrupt Flag Clear end*/
}
void Time2_CaptureWidth()
{
    /*<UserCodeStart>*//*<SinOne-Tag><855>*/
        Time_State=TXCON;    //首先保存定时器溢出和捕获标志位，防止连续来
        TXCON &= ~(0xC0);    //清标志位
        if(Time_State & 0x80)   //定时器溢出
        {
            Time_OverVaule++;  //定时器溢出次数
        }
        if((Time_State & 0x40))    //定时器捕获到事件
        {
            if(Measure_State==0)   //首次捕获到下降沿
            {
                count1=Time_OverVaule*0x10000+((RCAPXH<<8)+RCAPXL);
                Measure_State=1;
            }
            if(Measure_State==1)   //第二次及之后捕获到
            {
                count2=(Time_OverVaule&0x0000ffff)*0x10000;
                count2+=((RCAPXH<<8)+RCAPXL)&0x0000FFFF;
                CycleWidth[TimeIndex]=count2-count1;       //计算周期计数值
                count1=count2;  //更新上次捕获值后的计数值
                TimeIndex++;    //更新捕获计数值数组
                if(TimeIndex>=10)
                {
                    TimeIndex=0;
                }
            }
            TXCON &= ~(0x08);  //忽略定时器外部事件
            INT2R = 0x04;      //打开外部中断
            IE1 |=0x08;
            IP1 |=0x08;
            Time_State = 0;  //清保存的标志位
        }

    /*<UserCodeEnd>*//*<SinOne-Tag><855>*/
}
void INT22_CaptureWidth()
{
    /*<UserCodeStart>*//*<SinOne-Tag><865>*/
        count3=((THX<<8)+TLX)&0x0000FFFF;
        count3+=(Time_OverVaule&0x0000ffff)*0x10000;  //读计数值并把定时器溢出放在高16位
        NegaWidth[TimeIndex]=count3-count1;           //计数负脉宽计数值
        TXCON  |= 0x08;       //打开定时器捕获事件

    /*<UserCodeEnd>*//*<SinOne-Tag><865>*/
}
```

4.7.4 函数接口及变量

图形化编程使用的具体函数和变量说明如下两表所示：

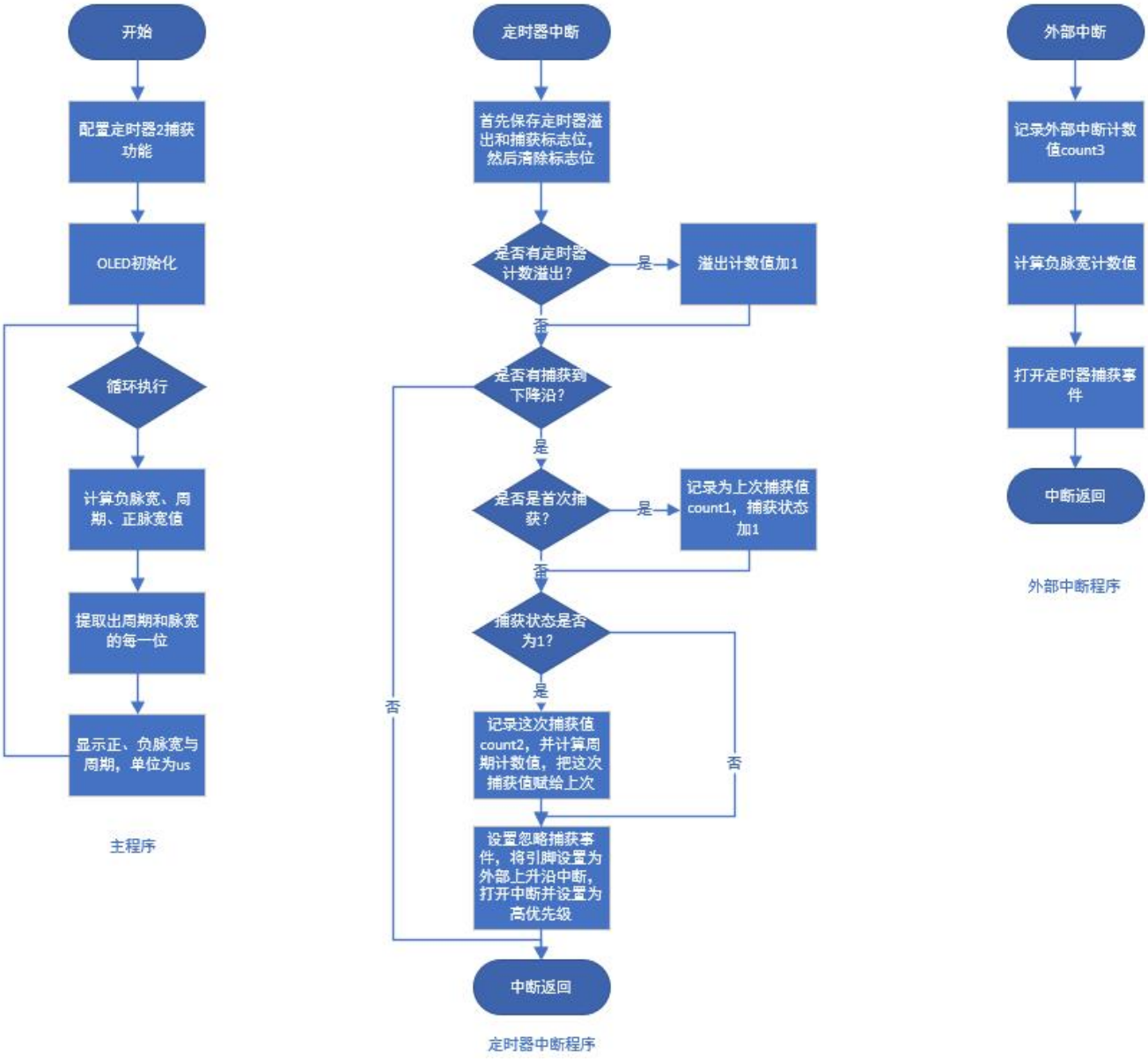
函数	函数功能	输入参数	返回值	备注
Time2_CaptureWidth ()	定时器 2 捕获测量周期函数	无	无	放到定时器 2 中断中执行
INT22_CaptureWidth()	外部中断 2 测量负脉宽函数	无	无	放到外部中断 2 中执行
Num_Conversion(unsigned long NumConver1,unsigned long NumConver2,unsigned long NumConver3)	提取出周期和脉宽的每一位	NumConver1：正脉宽 NumConver2：负脉宽 NumConver3：周期	无	将正脉宽、负脉宽、周期值的每一位提取出来，存入数组

变量名	说明
Time_State	保存定时器溢出和捕获标志位
Time_OverVaule	定时器溢出次数
Measure_State	0：首次捕获 1：第二次及之后的捕获
count1	上次捕获计数值，高 16 为溢出次数，低 16 位为捕获值
count2	这次捕获值计数值，高 16 为溢出次数，低 16 位为捕获值
count3	外部中断计数值，高 16 为溢出次数，低 16 位为捕获值
CycleWidth[]	存放周期计数值数组
NegaWidth[]	存放负脉宽计数值数组
TimeIndex	数组下标索引
FSYSHRC	时钟周期，1 计数值时间
NegativeWidth	负脉宽
PositiveWidth	正脉宽
Cycle	周期

4.7.5 程序流程图

主程序、定时器中断函数和外部中断函数的程序流程图如下：

其中，如果想要正确的计算周期和脉宽，要将宏定义 FSYSHRC 的值填对。



4.8 软件驱动 1/2BIAS LCD 屏示例

4.8.1 总体描述

GPIO 驱动 LCD 示例:GPIO_LCDDISPLAY_NBK1220，软件 GPIO 扫描实现 1/2Bias 的两位 LCD 显示，可以通过魔盒例子工程打开并另存为用户自己的工程。

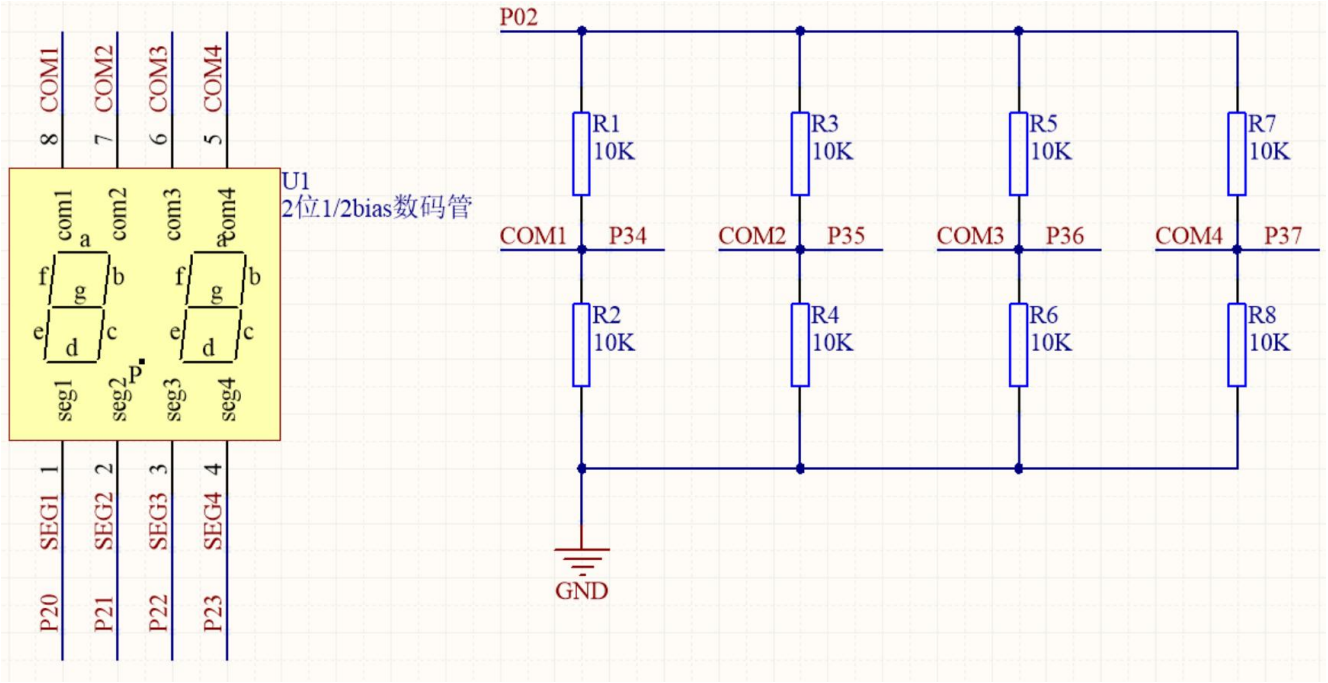
根据 LCD 的驱动原理可知，LCD 得交替加正、反向电压才会持续显示，如果长时间加恒定电压，会加速 LCD 的老化和损坏。LCD 显示器的对比度由 COM 脚上的电压值减去 SEG 脚上的电压值决定，当这个电压差大于 LCD 的饱和电压就能打开像素点。也就是 COM 与 SEG 有+VDD 或者-VDD 的电压差，所以要点亮某个像素点，只要将对应的 SEG 输出与 COM 相反的电压即可。

对于 1/2bias LCD 来说，假如 LCD 的显示电压是 3V，则 1/2bias 是 1.5V，也就是说在±3V 电压作用时，LCD 有显示；在±1.5V 及以下的电压作用时，LCD 没有显示。一般的单片机 IO 口不能直接输出半高电平（1.5V），但可以用相等的上下拉电阻实现。当 IO 口设置为高阻输入模式时，由于上下拉电阻的分压作用，则产生一个半高电平（1.5V）；当 IO 口设置为强推挽输出模式时，IO 输出 1，则产生一个高电平（3.3V），IO 输出 0，则引脚被拉到地产生一个低电平（0V）。如此，就可以产生驱动 1/2bias LCD 的波形。

以上，采用外接电阻驱动 1/2bias LCD 的方法基本上适用于所有有 IO 功能的 IC，除了以上的方法，92 系的一些 IC，有可以输出 1/2V_{DD} 的 IO，也可以用此类 IC 来驱动 1/2bias LCD。本示例主要说明的是用外接电阻的方式，驱动 1/2bias LCD。

4.8.2 外接电阻驱动原理图

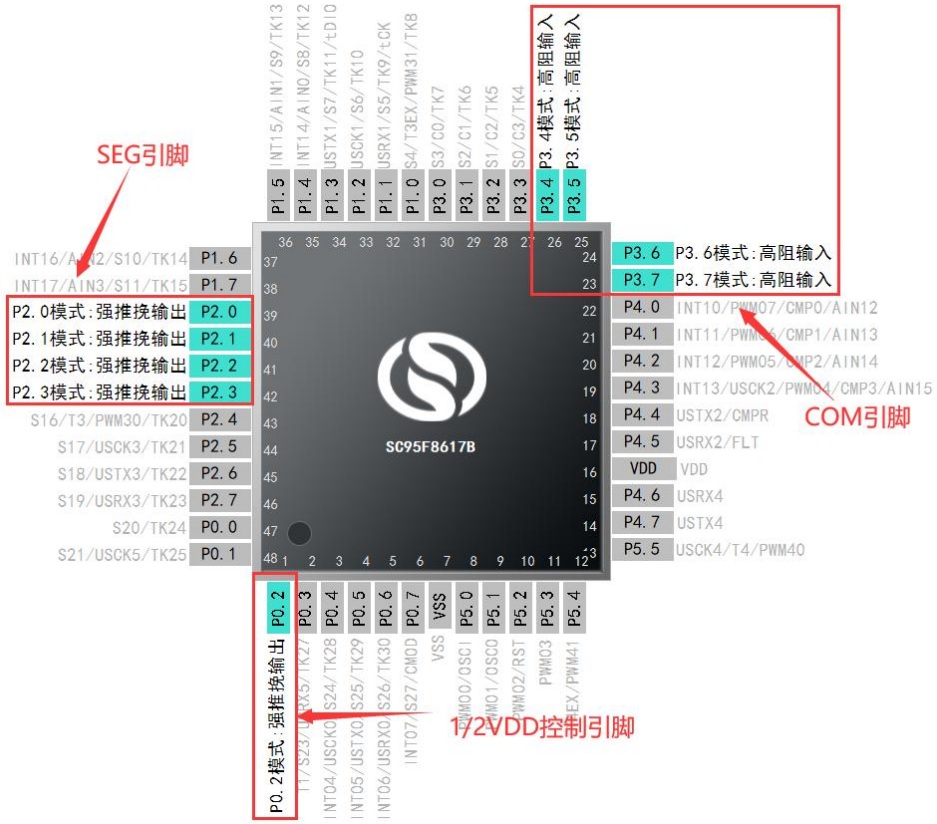
用外接电阻的方式驱动 1/2bias LCD 的电路原理图如下图所示，所用驱动条件为 3V 1/2bias 1/4duty 的两位 LCD 液晶显示屏，其中 P02 引脚用于控制输入 1/2V_{DD}，当需要进入低功耗模式时，可关闭该 IO 口；不需要进入低功耗，可直接用 VDD 接入。



PIN	1	2	3	4	5	6	7	8
COM1	1F	1A	2F	2A				COM1
COM2	1G	1B	2G	2B			COM2	
COM3	1E	1C	2E	2C		COM3		
COM4		1D	P	2D	COM4			

4.8.3 在易码魔盒中的配置

在易码魔盒中，用户可以直接在芯片资源配置界面中，直接点击 LCD 对应的引脚配置成强推挽输出和高阻输入模式，魔盒会生成对应引脚初始化的代码，不需要用户手动配置寄存器。

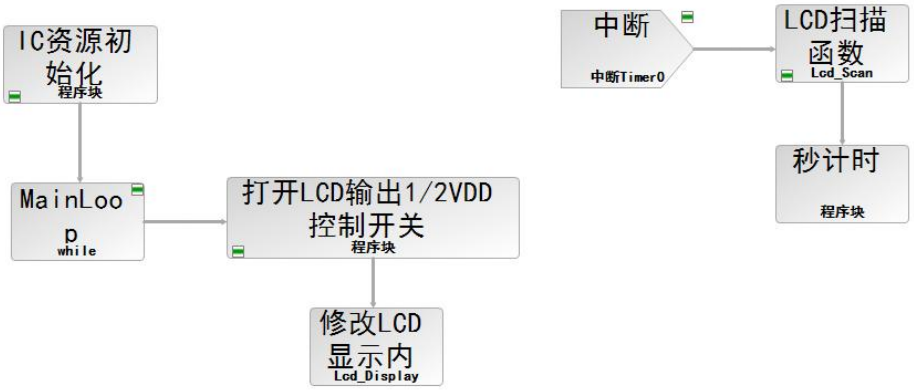


配置一个定时器，用于完成 LCD IO 口的扫描。



当芯片资源配置完成后，即可在用户程序配置界面，开始图形化编程。

示例 通过 GPIO 控制输入 LCD 的电压，在定时器中断中完成 IO 扫描，LED 显示计时，显示 10 秒：秒，范围 0.0 到 9.9。




```
void main(void)
{
    /*<Generated by EasyCodeCube begin>*/
    /*<UserCodeStart>*//*<SinOne-Tag><3>*/
    SC_Init(); /*** MCU init***/
    /*<UserCodeEnd>*//*<SinOne-Tag><3>*/
    /*<UserCodeStart>*//*<SinOne-Tag><4>*/
    /*****MainLoop*****/
    while(1)
    {
        /*<UserCodeStart>*//*<SinOne-Tag><314>*/
        P02=1;
        /*<UserCodeEnd>*//*<SinOne-Tag><314>*/
        /*<UserCodeStart>*//*<SinOne-Tag><210>*/
        Lcd_Display( Miao, HaoMiao , DISPLAYDOT );
        /*<UserCodeEnd>*//*<SinOne-Tag><210>*/
        /*<Begin-Inserted by EasyCodeCube for Condition>*/
    }
    /*<UserCodeEnd>*//*<SinOne-Tag><4>*/
    /*<Generated by EasyCodeCube end>*/
}
    void Timer0Interrupt()      interrupt 1
    {
        /*TIM0_it write here begin*/
        TIM0_ModelSetReloadCounter(1536);
        /*TIM0_it write here*/
        /*<Generated by EasyCodeCube begin>*/
        /*<UserCodeStart>*//*<SinOne-Tag><6>*/
        //Timer0Interrupt
        {
            /*<UserCodeStart>*//*<SinOne-Tag><188>*/
            Lcd_Scan();
            /*<UserCodeEnd>*//*<SinOne-Tag><188>*/
            /*<UserCodeStart>*//*<SinOne-Tag><253>*/
            T0Count++;
            if(T0Count>=500)
            {
                T0Count=0;
                HaoMiao++;
                if(HaoMiao>=10)
                {
                    HaoMiao=0;
                    Miao++;
                    if(Miao>=10)
                    {
                        Miao=0;
                    }
                }
            }
            /*<UserCodeEnd>*//*<SinOne-Tag><253>*/
            /*<Begin-Inserted by EasyCodeCube for Condition>*/
        }
        /*<UserCodeEnd>*//*<SinOne-Tag><6>*/
        /*<Generated by EasyCodeCube end>*/
        /*Timer0Interrupt Flag Clear begin*/
        /*Timer0Interrupt Flag Clear end*/
    }
```

```
void Lcd_Scan()
{
    /*<UserCodeStart>*//*<SinOne-Tag><185>*/
        P3CON &= ~0XF0;   //全部设为高阻，输出1/2VDD
        P3PH &= ~0XF0;
        switch(LCD_State)
        {

            case 0:
                COM1=1;             //先写1，防止脉冲干扰
                P3CON |= 0X10;      //COM1强推挽输出高，COM2、3、4高阻
                COM1=1;
                SEG1=((~(LcdDataTab[0]>>0))&0x01);        //写字段，COM为VDD，SEG为VSS时，像素点点亮
                SEG2=((~(LcdDataTab[1]>>0))&0x01);
                SEG3=((~(LcdDataTab[2]>>0))&0x01);
                SEG4=((~(LcdDataTab[3]>>0))&0x01);
                LCD_State=1;
                break;

            case 1:
                COM2=1;
                P3CON |= 0X20;      //COM2强推挽输出高，COM1、3、4高阻
                COM2=1;
                SEG1=((~(LcdDataTab[0]>>1))&0x01);        //写字段
                SEG2=((~(LcdDataTab[1]>>1))&0x01);
                SEG3=((~(LcdDataTab[2]>>1))&0x01);
                SEG4=((~(LcdDataTab[3]>>1))&0x01);
                LCD_State=2;
                break;

            case 2:
                COM3=1;
                P3CON |= 0X40;      //COM3强推挽输出高，COM1、2、4高阻
                COM3=1;
                SEG1=((~(LcdDataTab[0]>>2))&0x01);        //写字段
                SEG2=((~(LcdDataTab[1]>>2))&0x01);
                SEG3=((~(LcdDataTab[2]>>2))&0x01);
                SEG4=((~(LcdDataTab[3]>>2))&0x01);
                LCD_State=3;
                break;

            case 3:
                COM4=1;
                P3CON |= 0X80;      //COM4强推挽输出高，COM1、2、3高阻
                COM4=1;
                SEG1=((~(LcdDataTab[0]>>3))&0x01);        //写字段
                SEG2=((~(LcdDataTab[1]>>3))&0x01);
                SEG3=((~(LcdDataTab[2]>>3))&0x01);
                SEG4=((~(LcdDataTab[3]>>3))&0x01);
                LCD_State=4;
                break;

            case 4:
                COM1=0;           //先写0，防止脉冲干扰
                P3CON |= 0X10;      //COM1强推挽输出低，COM2、3、4高阻
                COM1=0;
                SEG1=(((LcdDataTab[0]>>0))&0x01);        //写字段，COM为VSS，SEG为VDD时，像素点点亮
                SEG2=(((LcdDataTab[1]>>0))&0x01);
                SEG3=(((LcdDataTab[2]>>0))&0x01);
                SEG4=(((LcdDataTab[3]>>0))&0x01);
                LCD_State=5;
                break;

            case 5:
                COM2=0;
                P3CON |= 0X20;      //COM2强推挽输出低，COM1、3、4高阻
                COM2=0;
                SEG1=(((LcdDataTab[0]>>1))&0x01);        //写字段
                SEG2=(((LcdDataTab[1]>>1))&0x01);
                SEG3=(((LcdDataTab[2]>>1))&0x01);
                SEG4=(((LcdDataTab[3]>>1))&0x01);
                LCD_State=6;
                break;

            case 6:
                COM3=0;
                P3CON |= 0X40;      //COM3强推挽输出低，COM1、2、4高阻
                COM3=0;
                SEG1=(((LcdDataTab[0]>>2))&0x01);        //写字段
                SEG2=(((LcdDataTab[1]>>2))&0x01);
                SEG3=(((LcdDataTab[2]>>2))&0x01);
                SEG4=(((LcdDataTab[3]>>2))&0x01);
                LCD_State=7;
                break;

            case 7:
                COM4=0;
                P3CON |= 0X80;      //COM4强推挽输出低，COM1、2、3高阻
                COM4=0;
                SEG1=(((LcdDataTab[0]>>3))&0x01);        //写字段
                SEG2=(((LcdDataTab[1]>>3))&0x01);
                SEG3=(((LcdDataTab[2]>>3))&0x01);
                SEG4=(((LcdDataTab[3]>>3))&0x01);
                LCD_State=0;
                break;
        }

    /*<UserCodeEnd>*//*<SinOne-Tag><185>*/
}
```

```
void Lcd_Display(uint8_t COMNUM1,uint8_t COMNUM2,bit DispDot)
{
    /*<UserCodeStart>*//*<SinOne-Tag><134>*/
    uint8_t i,temp=0;
    temp = COMNUM1;
    for(i=0;i<2;i++)
    {
     LcdTemp[0] = ((LcdCodeTab[temp]>>0)&0x01)<<0;    //A
     LcdTemp[1] = ((LcdCodeTab[temp]>>1)&0x01)<<1;    //B
     LcdTemp[2] = ((LcdCodeTab[temp]>>2)&0x01)<<2;    //C
     LcdTemp[3] = ((LcdCodeTab[temp]>>3)&0x01)<<3;    //D
     LcdTemp[4] = ((LcdCodeTab[temp]>>4)&0x01)<<2;    //E
     LcdTemp[5] = ((LcdCodeTab[temp]>>5)&0x01)<<0;    //F
     LcdTemp[6] = ((LcdCodeTab[temp]>>6)&0x01)<<1;    //G
     LcdTemp[7] = (((LcdCodeTab[temp]>>7) | DispDot)&0x01)<<3;   //DP
     if(i==0)
     {
       LcdDataTab[0] = (0x00 | LcdTemp[5] | LcdTemp[6] | LcdTemp[4] | LcdTemp[7]);   //1F、1G、1E、1P
       LcdDataTab[1] = (0x00 | LcdTemp[0] | LcdTemp[1] | LcdTemp[2] | LcdTemp[3]);   //1A、1B、1C、1D
       temp=COMNUM2;
      }
     if(i==1)
     {
       LcdDataTab[2] = (0x00 |LcdTemp[5] | LcdTemp[6] | LcdTemp[4] | LcdTemp[7]);   //2F、2G、2E、2P
       LcdDataTab[3] = (0x00 |LcdTemp[0] | LcdTemp[1] | LcdTemp[2] | LcdTemp[3]);   //2A、2B、2C、2D
     }

    }

    /*<UserCodeEnd>*//*<SinOne-Tag><134>*/
}
```

4.8.4 函数接口及变量

图形化编程使用的具体函数和变量说明如下两表所示：

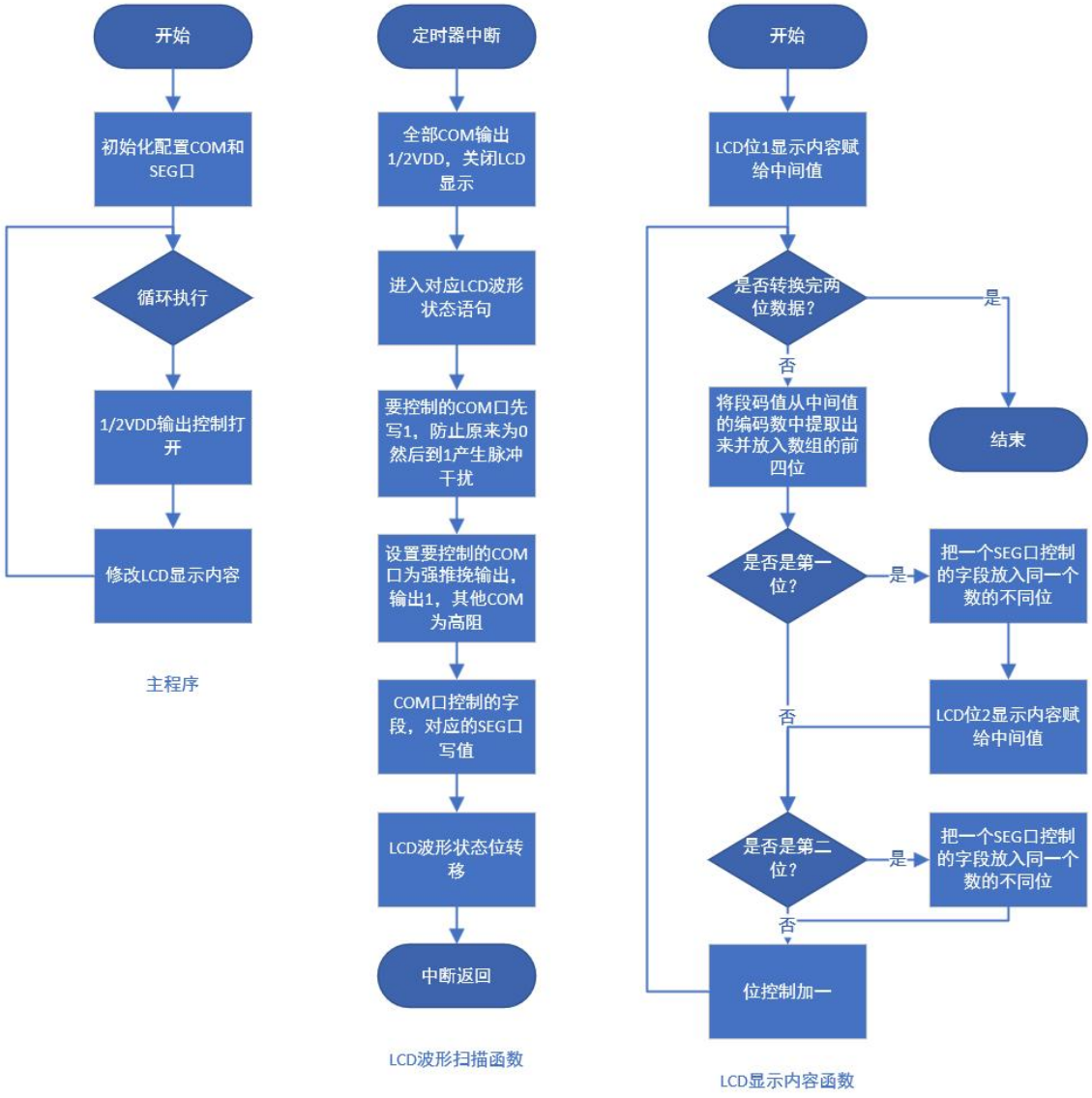
函数	函数功能	输入参数	返回值	备注
Lcd_Scan ()	1/2bias LCD 扫描函数	无	无	放到定时器中断中执行
Lcd_Display(uint8_t COMNUM1,uint8_t COMNUM2,bit DispDot)	LCD 显示内容	COMNUM1 ：LCD 位 1 要显示的数据 COMNUM2:LCD 位 2 要显示的内容 DispDot:LCD 是否要显示小数点	无	COM 口要跟控制的 SEG 口对应上

变量名	说明
LcdCodeTab[]	LCD 编码表
LcdDataTab[]	4 个 4 位段选的 LCD 数据
LcdTemp[8]	8 个 1 位 LCD 显示数据一段的值
LCD_State	LCD 波形的状态位
COM1	Com1 引脚的宏定义
COM2	Com2 引脚的宏定义
COM3	Com3 引脚的宏定义
COM4	Com4 引脚的宏定义
SEG1	Seg1 引脚的宏定义
SEG2	Seg2 引脚的宏定义
SEG3	Seg3 引脚的宏定义
SEG4	Seg4 引脚的宏定义

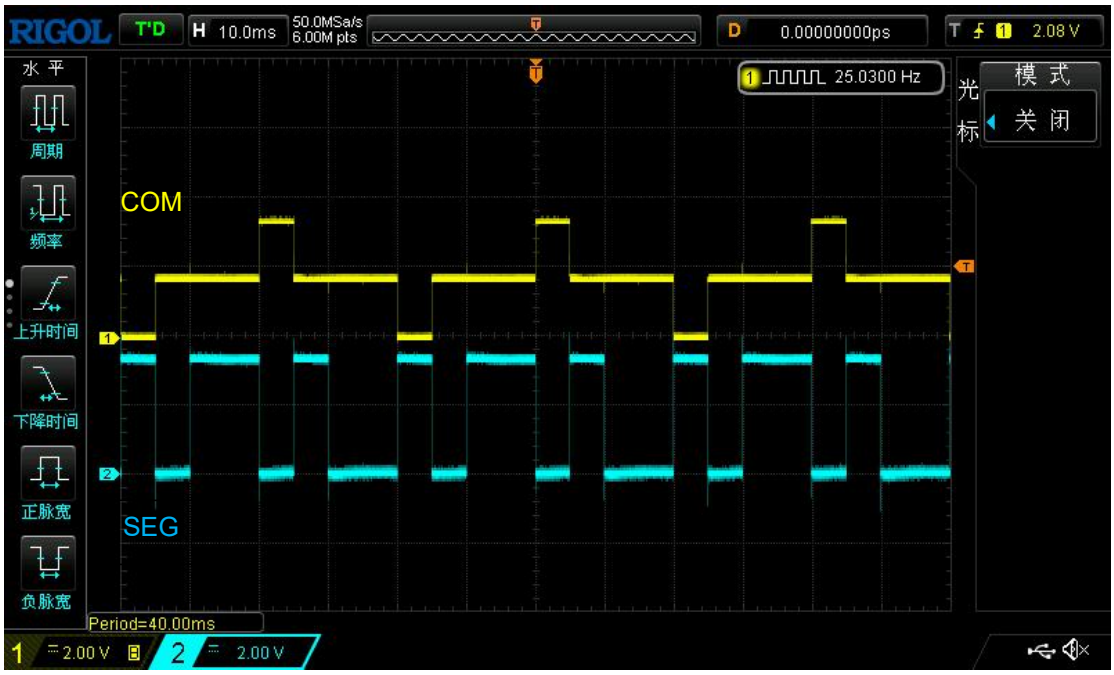
4.8.5 程序流程图

主程序、LCD 扫描函数和 LCD 显示内容的程序流程图如下：

其中，特别的在 LCD 显示内容函数中，要将 COM 控制的字段对应上。如果想要移植到 92 系有 1/2bias LCD 驱动的单片机，可以在扫描函数中，将其他 COM 设置为高阻输入模式改为设置该 IO 为 LCD 模式，具体为对 P0VO 寄存器进行操作，输出 1/2VDD 的电压。如果显示的内容有比较明显的残影，可以通过去降低供电电压的方式去解决。



4.8.6 1/2BIAS LCD 波形



规格更改记录

版本	记录	日期
V1.1	补充 EBS002&3 扩展板说明和各模块例程	2022 年 12 月
V1.0	初版	2022 年 10 月