PWM 脉冲宽度调制

1. 实验目的

- 掌握 STC15W4K32S4 系列 PWM 脉冲宽度调制的原理。
- 掌握 6 个 PWM 外设相关寄存器配置及程序设计。

2. 实验内容

- 编写程序实现 PWM2 和 PWM3 呼吸灯的程序设计。
- 编写程序实现 PWM4 和 PWM5 呼吸灯的程序设计。
- 编写程序实现 PWM6 和 PWM7 呼吸灯的程序设计。

3. 硬件设计

3.1. PWM 脉冲宽度调制介绍

PWM (全称是 Pulse Width Modulation)脉冲宽度调制是利用微处理器的数字输出来对模拟电路进行控制的一种非常有效的技术，广泛应用于从测量、通信到功率控制与变换的许多领域中。

下面以一个简单有趣的例子和大家分析 PWM 的作用。下图是使用 9V 电池来给一个电灯供电，这是个模拟电路，电路使用开关来控制电灯工作状态。

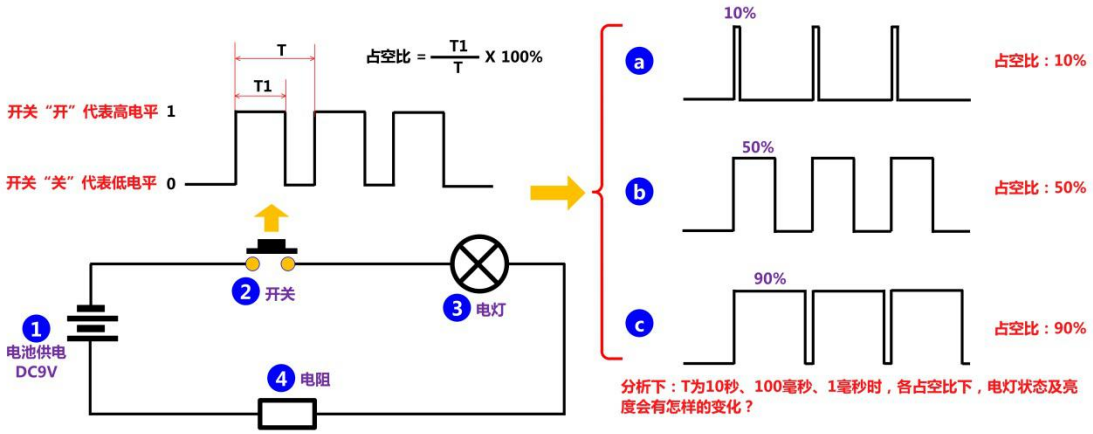


图 1：PWM 控制模拟电路示意图

■ 开关状态与电灯工作状态之间的关系：

根据常识我们知道开关按下，则电灯两端有压降，电灯会被点亮；如果开关松开，则电灯两端没有压降，形成不了回路，电灯就是灭的。那请分析下下面几种情况：

1） 假如开关按下去 5 秒、之后松开 5 秒，如此反复，则实验现象必然是灯亮 5 秒，然后熄灭 5 秒，不停闪烁下去。

2） 假如开关按下去 1 秒、之后松开 1 秒，如此反复，则实验现象也会是灯亮 1 秒，然

后熄灭 1 秒，不停闪烁下去。

3） 假如开关按下去 1 毫秒、之后松开 1 毫秒，那实现现象会是灯亮 1 毫秒，然后熄灭 1 毫秒，如此反复闪烁下去吗？

4） 假如开关按下去 0.1 毫秒，之后松开 0.9 毫秒，那实现现象又会是什么？会和开关按下去 0.9 毫秒、之后松开 0.1 毫秒一样吗？

分析：

1） 当开关按下去和松开的时间比较长时，比如 1 秒或 5 秒，人眼是很容易分辨出灯被点亮或是熄灭了的，所以看到的就是灯闪烁的现象。

2） 当开关按下去 1 毫秒、之后松开 1 毫秒，实际灯确实是被点亮 1 毫秒、然后熄灭 1 毫秒的，但人眼已经分别不出来了。（一般来说，人眼对于每 11 毫秒闪烁一次约 83 赫兹基本感觉不到，每 13 毫秒闪烁一次约 66 赫兹轻微频闪。）

3） 既然人眼分辨不出，那开关按下去 1 毫秒、之后松开 1 毫秒的实验现象会是什么，答案是看到灯始终是亮的，只是这个亮度没有开关一直按下去那么亮。根据上图中的分析，易知此时是占空比为 50%的情况，可等效为 4.5V 电池给电灯供电，那电灯的亮度自然会有所下降。

4） 同理，在开关按下去 0.1 毫秒，之后松开 0.9 毫秒时，人眼看到的还是灯一直亮的状态，只是此时占空比是 10%，等效于 0.9V 电池给电灯供电，亮度会很暗。而当开关按下去 0.9 毫秒、之后松开 0.1 毫秒时，占空比为 90%，可等效于 8.1V 电池给电灯供电，电灯亮度只略低于 9V 电池直接供电的亮度。

5） 细心的朋友会发现，开关按下去 1 秒、松开 1 秒，容易实现。那开关按下去 1 毫秒、松开 1 毫秒怎么实现？所以上面说的是假设，实际使用中，开关会是一个开关电路，触发信号是单片机提供的 PWM 信号，该 PWM 的周期和占空比，单片机都可以精确给出。

✧ 注：呼吸灯的原理，如果我们控制 PWM 输出频率达到 83HZ 以上，那么 PWM 引脚连接的指示灯是看不到闪烁的，是一种常亮的状态，但亮度又不像直接给个低电平那么亮，如果频率不变，不断控制占空比，那么被点亮的指示灯的亮度就是不断变化的，像“呼吸”一样。

■ 关于频率和占空比的理解：

上面的实例中已经在使用频率和占空比的知识点，频率是相对于信号的周期而言，通过波形知道了周期就可以算出频率，频率是周期的倒数。而占空比是指高电平的时间占整个周期的比例。

1） 实现呼吸灯的效果，对输出的 PWM 信号频率有要求，该 PWM 频率越高，效果越好，完成一个调制周期的时间越短。但 PWM 的频率越高，对硬件的要求也越苛刻。

2） PWM 信号可以模拟产生也可以控制 PWM 外设寄存器实现，一般模拟产生会占用 CPU 较多资源，尤其高速 PWM 信号生成必须借助 PWM 外设来实现。

3） 脉冲宽度调制的宽，不是物体的宽度，而是高电平（有效电平）信号在一个调制周期中持续的时间长短，他可以用占空比去衡量，占空比越大，脉冲宽度越宽。

■ 使用 PWM 信号控制模拟电路的优势：

根据上面的实例，我们知道可以通过 PWM 信号的频率和占空比控制电灯的状态和亮度。那么使用 PWM 信号控制模拟电路的优势有哪些呢？下面分析：

1） 控制灯的亮度，使用纯模拟电路可以实现，比如改变电池的输出电压或者改变电阻的阻值。但这两个参数改动起来都比较困难，即使可以还有两个不能忽视的问题，一个是电池电压会随着电量减少而不断变化，一个是模拟电路容易随时间漂移而难以调节。

2） 从功耗上面考虑，一般模拟电路还会存在发热严重的问题，其功耗相对于工作元件两端电压与电流的乘积成正比。

3） 抗干扰性上，模拟电路还可能对噪声很敏感，任何扰动或噪声都肯定会改变电流值的大小。而 PWM 信号是数字信号，外部噪声很难改变逻辑 0 或逻辑 1 的信息。

■ PWM 脉冲宽度调制还有哪些应用呢？

1） 电机驱动：PWM 被称为“开关驱动装置”，PWM 信号的高低电平可控制电机是否通电，电机通电，就会加速；电机断电，就会减速甚至停止。只要 PWM 信号频率达到一定值，改变 PWM 占空比可实现对电机速度的控制。

2） 开关电源：PWM 开关型稳压电路是在控制电路输出频率不变的情况下，通过电压反馈调整其占空比，从而达到稳定输出电压的目的。

3） 电池充电：在镍氢电池智能充电器中采用的脉宽 PWM 法，他是把每一脉冲宽度均相等的脉冲列作为 PWM 波形，通过改变脉冲列的周期可以调频，改变脉冲的宽度或占空比可以调压，采用适当控制方法即可使电压与频率协调变化。实现通过调整 PWM 的周期、PWM 的占空比而达到控制充电电流的目的。

4） D/A 转换：PWM 高频输出后加 RC 滤波电路，通过改变 PWM 信号的占空比实现输出不同电压值的目的。

5） 逆变电路：目前中小功率的逆变电路几乎都采用 PWM 技术，逆变电路是 PWM 控制技术极为重要的应用场合。

✧ 占空比的设置：理论上说，通过控制占空比（为 0%或者 100%）可以实现输出纯粹的低电平或者高电平，但实际使用要结合所选单片机特点进行设计。

3.2. STC15W4K32S4 系列单片机 PWM 介绍

STC15W4K32S4 系列单片机集成了一组增强型 PWM 波形发生器，该 PWM 波形发生器内部有一个 15 位的 PWM 计数器供 6 路相互独立的 PWM 使用，即 PWM2、PWM3、PWM4、PWM5、PWM6 和 PWM7。另外，用户可设置每路 PWM 的初始电平，也可以通过对每路 PWM 的两个控制波形翻转的计数器的配置实现每路 PWM 高低电平宽度的调节。

由于 6 路 PWM 是各自独立的，且每路 PWM 的初始状态可以自行设定，所以用户可以将其中的任意两路配合起来使用，从而实现互补对称输出及死区控制等特殊用途。

STC15W4K32S4 系列单片机 PWM 波形发生器还设计了对外部异常事件进行监控的功能，可用于紧急关闭 PWM 输出，避免出现严重后果。

STC15W4K32S4 系列单片机每一路 PWM 都有 2 个 IO 引脚供选择使用，如下表。

表 1：单片机 PWM 输出控制引脚分配

PWMx	对应 IO 口	功能描述	说明	备注
PWM2	P3.7	PWM2 输出控制引脚	非独立 GPIO	蜂鸣器电路
PWM2_2	P2.7	PWM2 输出控制引脚	非独立 GPIO	4*4 矩阵电路
PWM3	P2.1	PWM3 输出控制引脚	非独立 GPIO	LCD 屏接口
PWM3_2	P4.5	PWM3 输出控制引脚	非独立 GPIO	4*4 矩阵电路
PWM4	P2.2	PWM4 输出控制引脚	非独立 GPIO	外部存储器电路
PWM4_2	P4.4	PWM4 输出控制引脚	非独立 GPIO	4*4 矩阵电路
PWM5	P2.3	PWM5 输出控制引脚	非独立 GPIO	LCD 屏接口
PWM5_2	P4.2	PWM5 输出控制引脚	非独立 GPIO	W5500 模块接口
PWM6	P1.6	PWM6 输出控制引脚	非独立 GPIO	预留外部晶振接口
PWM6_2	P0.7	PWM6 输出控制引脚	非独立 GPIO	用户 LED 指示灯
PWM7	P1.7	PWM7 输出控制引脚	非独立 GPIO	预留外部晶振接口
PWM7_2	P0.6	PWM7 输出控制引脚	非独立 GPIO	用户 LED 指示灯

✧ 注：独立 GPIO 表示开发板没有其他的电路使用这个 GPIO，非独立 GPIO 说明开发板有其他电路用到了该 GPIO。针对非独立 GPIO 使用时需特别注意。

STC15W4K32S4 系列单片机 PWM 外设的理解主要是对其 PWM 波形发生器的内部结构框图的解析，下面给出该结构图。

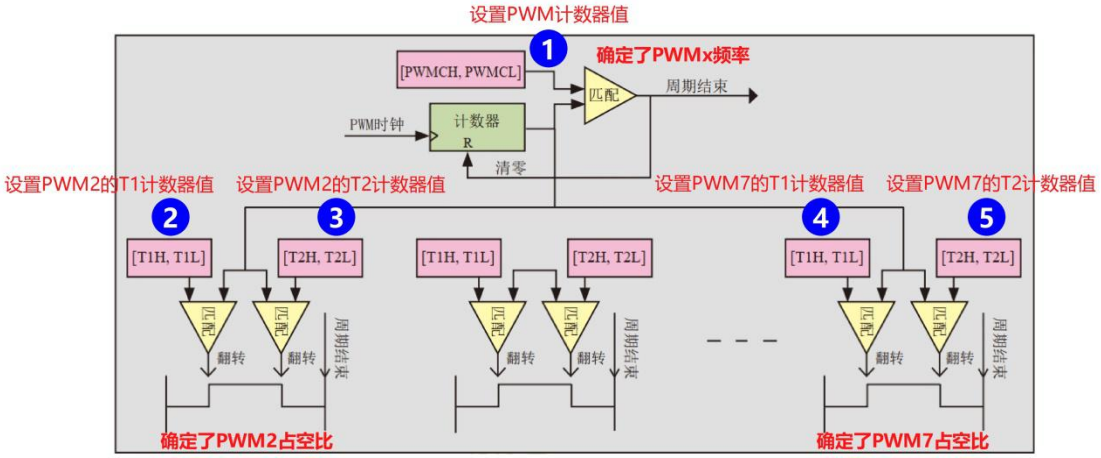


图 2：PWM 波形发生器内部结构示意图

✧ 注：从该图不难看出，虽然 6 路 PWM 相互独立，但因为共用同一个计数器，所以 6 路 PWM 若多个同时使用则其频率都是确定一样的。

3.3. PWM 配置步骤

针对 STC15W4K32S4 系列单片机有 6 路 PWM 外设，软件的配置过程如下：

图 3：PWM 软件配置步骤

✧ 注：实验例程即是按照上述配置步骤操作寄存器的相关位，后有详述。

4. 软件设计

4.1. PWM 寄存器汇集

STC15W4K32S4 系列单片机操作 PWM 时会用到 37 个寄存器，如下表所示：

表 2：STC15W4K32S4 系列 PWM 使用寄存器汇总

序号	寄存器名	读/写	功能描述
1	PWMCFG	读/写	PWM 配置寄存器。
2	PWMCR	读/写	PWM 控制寄存器。
3	PWMIF	读/写	PWM 中断标志寄存器。
4	PWMFDCR	读/写	PWM 外部异常控制寄存器。

5	PWMCH	读/写	PWM 计数器高 8 位寄存器。
6	PWMCL	读/写	PWM 计数器低 8 位寄存器。
7	PWMCKS	读/写	PWM 时钟选择寄存器。
8	PWM2T1H	读/写	PWM2T1 计数器高 8 位寄存器。
9	PWM2T1L	读/写	PWM2T1 计数器低 8 位寄存器。
10	PWM2T2H	读/写	PWM2T2 计数器高 8 位寄存器。
11	PWM2T2L	读/写	PWM2T2 计数器低 8 位寄存器。
12	PWM2CR	读/写	PWM2 控制寄存器。
13	PWM3T1H	读/写	PWM3T1 计数器高 8 位寄存器。
14	PWM3T1L	读/写	PWM3T1 计数器低 8 位寄存器。
15	PWM3T2H	读/写	PWM3T2 计数器高 8 位寄存器。
16	PWM3T2L	读/写	PWM3T2 计数器低 8 位寄存器。
17	PWM3CR	读/写	PWM3 控制寄存器。
18	PWM4T1H	读/写	PWM4T1 计数器高 8 位寄存器。
19	PWM4T1L	读/写	PWM4T1 计数器低 8 位寄存器。
20	PWM4T2H	读/写	PWM4T2 计数器高 8 位寄存器。
21	PWM4T2L	读/写	PWM4T2 计数器低 8 位寄存器。
22	PWM4CR	读/写	PWM4 控制寄存器。
23	PWM5T1H	读/写	PWM5T1 计数器高 8 位寄存器。
24	PWM5T1L	读/写	PWM5T1 计数器低 8 位寄存器。
25	PWM5T2H	读/写	PWM5T2 计数器高 8 位寄存器。
26	PWM5T2L	读/写	PWM5T2 计数器低 8 位寄存器。
27	PWM5CR	读/写	PWM5 控制寄存器。
28	PWM6T1H	读/写	PWM6T1 计数器高 8 位寄存器。
29	PWM6T1L	读/写	PWM6T1 计数器低 8 位寄存器。
30	PWM6T2H	读/写	PWM6T2 计数器高 8 位寄存器。
31	PWM6T2L	读/写	PWM6T2 计数器低 8 位寄存器。
32	PWM6CR	读/写	PWM6 控制寄存器。

33	PWM7T1H	读/写	PWM7T1 计数器高 8 位寄存器。
34	PWM7T1L	读/写	PWM7T1 计数器低 8 位寄存器。
35	PWM7T2H	读/写	PWM7T2 计数器高 8 位寄存器。
36	PWM7T2L	读/写	PWM7T2 计数器低 8 位寄存器。
37	PWM7CR	读/写	PWM7 控制寄存器。

✧ 注：上述寄存器的 6 路相互独立的 PWM 都有计数器 T1、T2 和控制寄存器，只需把一路 PWM 对应的寄存器搞清楚就可以了。

4.2. 寄存器解析

4.2.1. PWM 配置寄存器 PWMCFG

PWM 配置寄存器 PWMCFG 的 CBTADC 位为 PWM 计数器归零时触发 ADC 转换，该 CBTADC 位应用于比较特殊的场合，一般配置为 0 即可。关于 C2INI~C7INI 位的操作：6 路 PWM 输出口的初始电平设置，根据应用需要配置即可。

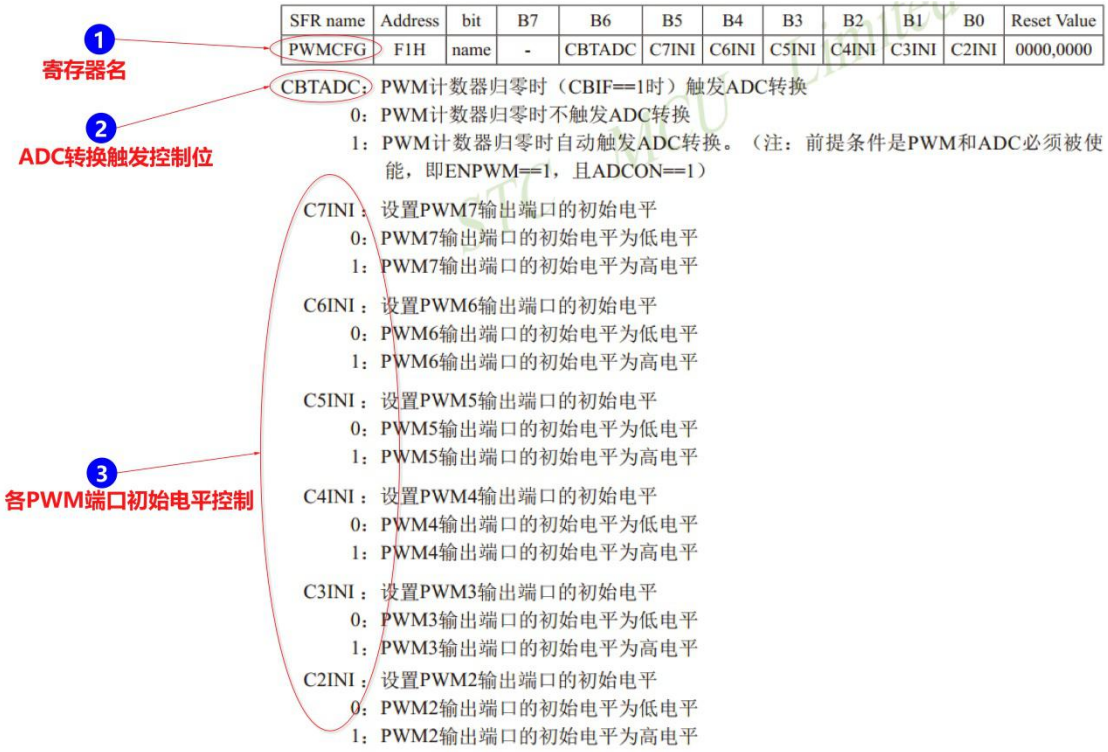
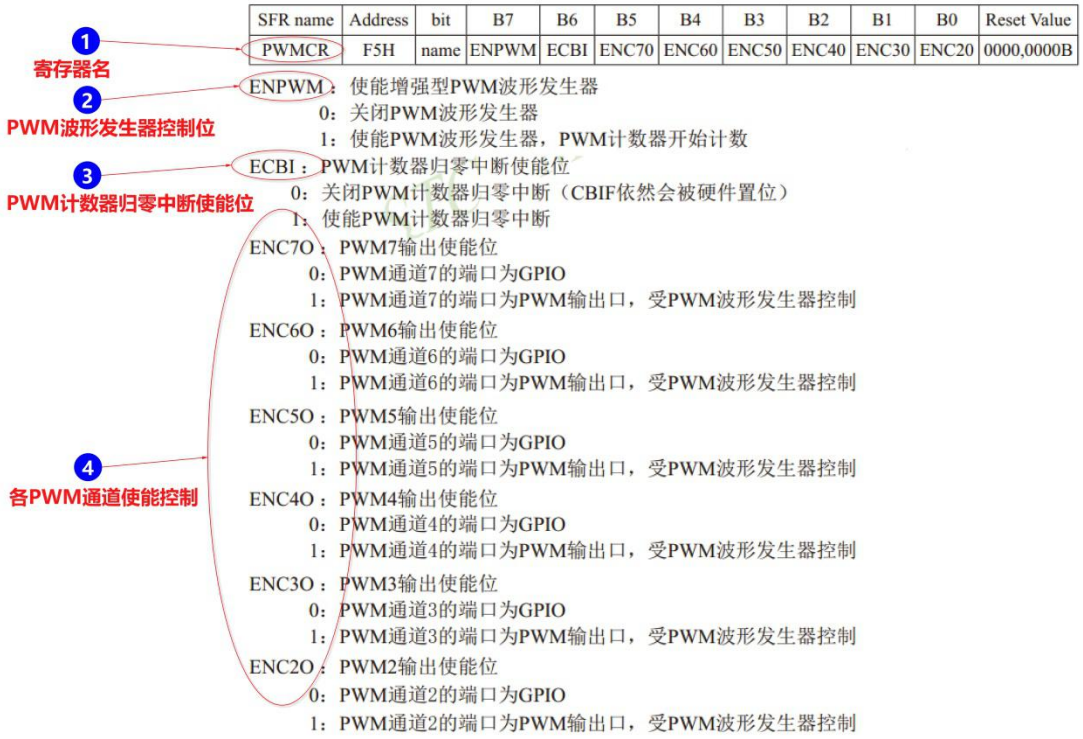


图 4：PWM 配置寄存器

4.2.2. PWM 控制寄存器 PWMCR

PWM 控制寄存器 PWMCR 的 ENPWM 位为 PWM 波形发生器使能位，在对相应 PWM 口配置完成后需使能该位。PWMCR 寄存器的 ECBI 位用于使能 PWM 计数器归零中断。关于 ENC2O~ENC7O 位的操作：关联相应 PWM 通道端口为 PWM 输出口，受 PWM 波形发

生器控制。



SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWMCR	F5H	name	ENPWM	ECBI	ENC70	ENC60	ENC50	ENC40	ENC30	ENC20	0000,0000B

ENPWM：使能增强型PWM波形发生器
0：关闭PWM波形发生器
1：使能PWM波形发生器，PWM计数器开始计数
ECBI：PWM计数器归零中断使能位
0：关闭PWM计数器归零中断（CBIF依然会被硬件置位）
1：使能PWM计数器归零中断
ENC7O：PWM7输出使能位
0：PWM通道7的端口为GPIO
1：PWM通道7的端口为PWM输出口，受PWM波形发生器控制
ENC6O：PWM6输出使能位
0：PWM通道6的端口为GPIO
1：PWM通道6的端口为PWM输出口，受PWM波形发生器控制
ENC5O：PWM5输出使能位
0：PWM通道5的端口为GPIO
1：PWM通道5的端口为PWM输出口，受PWM波形发生器控制
ENC4O：PWM4输出使能位
0：PWM通道4的端口为GPIO
1：PWM通道4的端口为PWM输出口，受PWM波形发生器控制
ENC3O：PWM3输出使能位
0：PWM通道3的端口为GPIO
1：PWM通道3的端口为PWM输出口，受PWM波形发生器控制
ENC2O：PWM2输出使能位
0：PWM通道2的端口为GPIO
1：PWM通道2的端口为PWM输出口，受PWM波形发生器控制

图 5：PWM 控制寄存器

4.2.3. PWM 中断标志寄存器

PWM 中断标志寄存器 PWMIF 的 CBIF 位为 PWM 计数器归零中断标志位。CBIF 位在当 PWM 计数器归零时，硬件自动置 1，需软件清零。关于 PWMIF 寄存器 C2IF~C7IF 位的操作： PWM 相应通道的中断标志位。C2IF~C7IF 位在当 PWM 发生翻转时，硬件自动置 1，需软件清零。

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWMIF	F6H	name	-	CBIF	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	x000,0000B

CBIF：PWM计数器归零中断标志位
当PWM计数器归零时，硬件自动将此位置1。当ECBI==1时，程序会跳转到相应中断入口执行中断服务程序。需要软件清零。

C7IF：第7通道的PWM中断标志位
可设置在翻转点1和翻转点2触发C7IF（详见EC7T1SI和EC7T2SI）。当PWM发生翻转时，硬件自动将此位置1。当EPWM7I==1时，程序会跳转到相应中断入口执行中断服务程序。需要软件清零。

C6IF：第6通道的PWM中断标志位
可设置在翻转点1和翻转点2触发C6IF（详见EC6T1SI和EC6T2SI）。当PWM发生翻转时，硬件自动将此位置1。当EPWM6I==1时，程序会跳转到相应中断入口执行中断服务程序。需要软件清零。

C5IF：第5通道的PWM中断标志位
可设置在翻转点1和翻转点2触发C5IF（详见EC5T1SI和EC5T2SI）。当PWM发生翻转时，硬件自动将此位置1。当EPWM5I==1时，程序会跳转到相应中断入口执行中断服务程序。需要软件清零。

C4IF：第4通道的PWM中断标志位
可设置在翻转点1和翻转点2触发C4IF（详见EC4T1SI和EC4T2SI）。当PWM发生翻转时，硬件自动将此位置1。当EPWM4I==1时，程序会跳转到相应中断入口执行中断服务程序。需要软件清零。

C3IF：第3通道的PWM中断标志位
可设置在翻转点1和翻转点2触发C3IF（详见EC3T1SI和EC3T2SI）。当PWM发生翻转时，硬件自动将此位置1。当EPWM3I==1时，程序会跳转到相应中断入口执行中断服务程序。需要软件清零。

C2IF：第2通道的PWM中断标志位
可设置在翻转点1和翻转点2触发C2IF（详见EC2T1SI和EC2T2SI）。当PWM发生翻转时，硬件自动将此位置1。当EPWM2I==1时，程序会跳转到相应中断入口执行中断服务程序。需要软件清零。

图 6：PWM 中断标志寄存器

4.2.4. PWM 外部异常控制寄存器

PWM 外部异常控制寄存器 PWMFDCR 的 ENFD 位为 PWM 外部异常检测功能控制位，需要用到 PWM 外部异常检测功能时置 1。PWMFDCR 寄存器的 EFDI 位为 PWM 异常检测中断使能位。该寄存器其他位请参考下图。

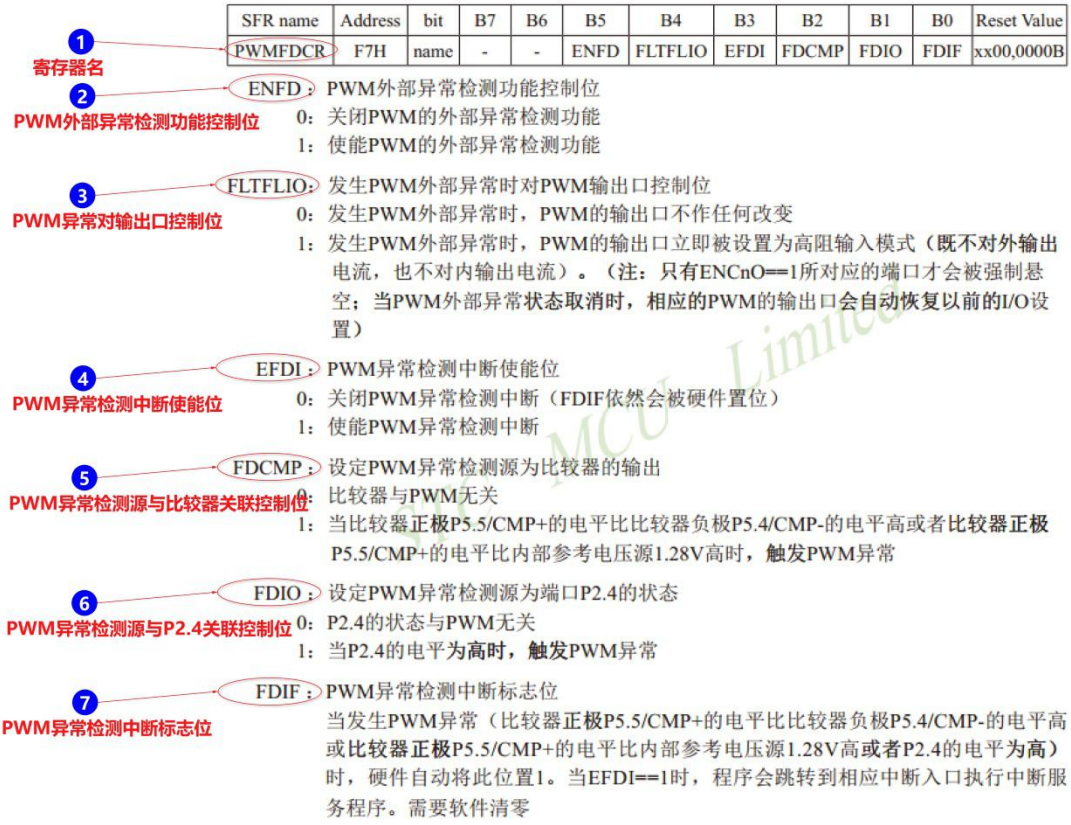


图 7：PWM 外部异常控制寄存器

✧ 注： PWMFDCR 寄存器是对 PWM 外部异常控制有需求时使用，一般应用可对相应位设置为 0 即可。

4.2.5. PWM 时钟选择寄存器

PWM 时钟选择寄存器 PWMCKS 的 SELT2 位为 PWM 时钟源选择位，该时钟源选择位为 0 时，即选择 PWM 时钟源为系统时钟提供，那么设置 PS[3:0]才有意义。PWMCKS 寄存器的 PS[3:0]位用于对系统时钟分频后提供 PWM 时钟源。

图 8：辅助寄存器 2

✧ 注：PWMCKS 是在扩展 RAM 区，这就要求在访问这些寄存器时需将 P_SW2 寄存器的 EAXSFR 位置 1。

4.2.6. PWM2 翻转寄存器

PWM2 有 2 个 15 位的翻转计数器 PWM2T1 和 PWM2T2，因为是 8 位单片机，寄存器都是 8 位的，所以 PWM2 实际拥有 4 个用于翻转的寄存器，分别是 PWM2T1H、PWM2T1L、

PWM2T2H 和 PWM2T2L。

PWM2的第一次翻转计数器的高字节：PWM2T1H

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM2T1H	FF00H (XSFR)	name	-	PWM2T1H[14:8]							x000,0000B

PWM2的第一次翻转计数器的低字节：PWM2T1L

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM2T1L	FF01H (XSFR)	name	PWM2T1L[7:0]								0000,0000B

PWM2的第二次翻转计数器的高字节：PWM2T2H

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM2T2H	FF02H (XSFR)	name	-	PWM2T2H[14:8]							x000,0000B

PWM2的第二次翻转计数器的低字节：PWM2T2L

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM2T2L	FF03H (XSFR)	name	PWM2T2L[7:0]								0000,0000B

图 9：PWM2 翻转寄存器

4.2.7. PWM2 控制寄存器

PWM2 控制寄存器 PWM2CR 的 PWM2_PS 位为 PWM2 输出引脚选择（P2.7 和 P3.7 引脚选择）。PWM2CR 寄存器的 PWM2I 位的操作，置 1 可保证 C2IF 被硬件置 1 时，程序进入相应中断入口。关于 EC2T2SI 和 EC2T1SI 位的操作，置 1 使能 T1 或 T2 翻转时中断，切记此处 T1 或 T2 不是单片机本身的定时器 T1 或 T2，而是 PWM2 里面的翻转计数器。

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM2CR	FF04H (XSFR)	name	-	-	-	-	PWM2_PS	EPWM2I	EC2T2SI	EC2T1SI	xxxx,0000B

1 寄存器名

2 PWM2输出引脚选择位

PWM2_PS：PWM2输出管脚选择位
0：PWM2的输出管脚为PWM2：P3.7
1：PWM2的输出管脚为PWM2_2：P2.7

3 PWM2中断使能控制位

EPWM2I：PWM2中断使能控制位
0：关闭PWM2中断
1：使能PWM2中断，当C2IF被硬件置1时，程序将跳转到相应中断入口执行中断服务程序。

4 PWM2的T2翻转中断控制位

EC2T2SI：PWM2的T2匹配发生波形翻转时的中断控制位
0：关闭T2翻转时中断
1：使能T2翻转时中断，当PWM波形发生器内部计数值与T2计数器所设定的值相匹配时，PWM的波形发生翻转，同时硬件将C2IF置1，此时若EPWM2I==1，则程序将跳转到相应中断入口执行中断服务程序。

5 PWM2的T1翻转中断控制位

EC2T1SI：PWM2的T1匹配发生波形翻转时的中断控制位
0：关闭T1翻转时中断
1：使能T1翻转时中断，当PWM波形发生器内部计数值与T1计数器所设定的值相匹配时，PWM的波形发生翻转，同时硬件将C2IF置1，此时若EPWM2I==1，则程序将跳转到相应中断入口执行中断服务程序。

图 10：PWM2 控制寄存器

4.3. PWM2 和 PWM3 呼吸灯实验

✧ 注：本节的实验源码是在“实验 2-4-1：外部中断 0（下降沿中断方式）”的基础上修改。本节对应的实验源码是：“实验 2-9-1：PWM2 和 PWM3 呼吸灯实验”。

4.3.1. 工程需要用到的 c 文件

本例需要用到的 c 文件如下表所示，工程需要添加下表中的 c 文件。

表 3：实验需要用到的 c 文件

序号	文件名	后缀	功能描述
1	pwm	.c	外部 PWM 有关的用户自定义函数。
2	delay	.c	包含用户自定义延时函数。

4.3.2. 头文件引用和路径设置

■ 需要引用的头文件

```
1. #include "delay.h"
2. #include "pwm.h"
```

■ 需要包含的头文件路径

本例需要包含的头文件路径如下表：

表 4：头文件包含路径

序号	路径	描述
1	..\ Source	pwm.h 和 delay.h 头文件在该路径，所以要包含。
2	..\User	15W4KxxS4.h 头文件在该路径，所以要包含。

MDK 中点击魔术棒，打开工程配置窗口，按照下图所示添加头文件包含路径。

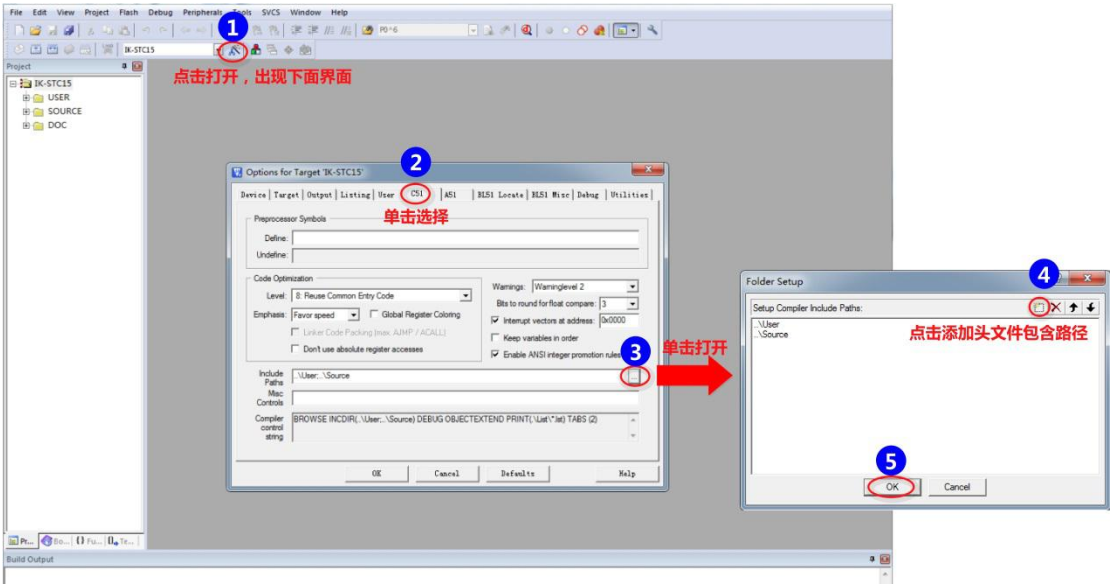


图 11：添加头文件包含路径

4.3.3. 编写代码

首先，在 pwm.c 文件中编写 PWM2 和 PWM3 的初始化函数 PWM2PWM3_Configuration，代码如下。

程序清单：PWM2 和 PWM3 初始化函数

```
1.  /*************************************************************************
2.  功能描述：对 PWM2 和 PWM3 进行初始化
3.  入口参数：无
4.  返回值：无
5.   *************************************************************************/
6.  void PWM2PWM3_Configuration(void)
7.  {
8.      PWMCFG &= 0xBF;                 //将 CBTADC 位置 0，即 PWM 计数器归零时不触发 ADC 转换
9.      PWMIF &= 0xBF;                  //将 CBIF 位置 0，PWM 计数器归零中断标志位，需软件清零
10.
11.     P_SW2 |= 0x80;          //将 EAXSFR 位置 1， 以访问 PWM 在扩展 RAM 区的特殊功能寄存器
12.     //对 PWM2 的初始化部分
13.     PWM2CR |= 0x08;                 //将 PWM2_PS 位置 1， 选择 PWM2 的输出引脚是 P2.7
14.     PWMCR |= 0x01;        //将 ENC2O 位置 1，PWM2 的端口为 PWM 输出口，受 PWM 波形发生器控制
15.     PWMCFG &= 0xFE;                 //将 C2INI 位置 0，设置 PWM2 输出端口的初始电平为低电平
16.     PWMIF &= 0xFE;                    //将 C2IF 位置 0， PWM2 中断标志位， 需软件清零
17.     PWM2CR |= 0x04;                 //将 EPWM2I 位置 1， 使能 PWM2 中断
18.     PWM2CR &= 0xFD;                 //将 EC2T2SI 位置 0， 关闭 T2 翻转时中断
19.     PWM2CR &= 0xFE;                 //将 EC2T1SI 位置 0， 关闭 T1 翻转时中断
20.     //对 PWM3 的初始化部分
21.     PWM3CR |= 0x08;                 //将 PWM3_PS 位置 1， 选择 PWM3 的输出引脚是 P4.5
22.     PWMCR |= 0x02;        //将 ENC3O 位置 1，PWM3 的端口为 PWM 输出口，受 PWM 波形发生器控制
```

```
    PWMCFG &= 0xFD;         //将 C3INI 位置 0，设置 PWM3 输出端口的初始电平为低电平
    PWMIF &= 0xFD;              //将 C3IF 位置 0，PWM3 中断标志位，需软件清零
    PWM3CR |= 0x04;             //将 EPWM3I 位置 1，使能 PWM3 中断
    PWM3CR &= 0xFD;             //将 EC3T2SI 位置 0，关闭 T2 翻转时中断
    PWM3CR &= 0xFE;             //将 EC3T1SI 位置 0，关闭 T1 翻转时中断
    //对 PWM2 和 PWM3 翻转计数器赋初值
    PWM2T1 =1;                  //赋值 PWM2 第一次翻转计数器值
    PWM2T2 = 0x00FA;            //赋值 PWM2 第二次翻转计数器值
    PWM3T1 =1;                  //赋值 PWM3 第一次翻转计数器值
    PWM3T2 = 0x00FA;            //赋值 PWM3 第二次翻转计数器值

  //对 PWM 波形发生器时钟源进行初始化
    PWMCKS |= 0x10;             //将 SELT2 位置 1，PWM 时钟源为定时器 2 溢出脉冲
    PWMC = 0x00FA;              //PWM 计数器赋值（同时对 PWMCH 和 PWMCL 进行了赋值）
    AUXR |= 0x04;                 //定时器 2 时钟为 Fosc,即 1T
    T2L = 0xE0;                 //设定定时初值
    T2H = 0xFE;                 //设定定时初值
    AUXR |= 0x10;               //启动定时器 2

    P_SW2 &= 0x7F;              //将 EAXSFR 位置 0，恢复访问 XRAM

    //PWM 外部异常控制寄存器的操作
    PWMFDCR &= 0xDF;              //将 ENFD 位置 0，关闭 PWM 外部异常检测功能
    PWMFDCR &= 0xF7;              //将 ENDI 位置 0，关闭 PWM 异常检测中断
    PWMFDCR &= 0xFB;              //将 FDCMP 位置 0，比较器与 PWM 无关
    PWMFDCR &= 0xFD;              //将 FDIO 位置 0，P2.4 的状态与 PWM 无关
    PWMFDCR &= 0xFE;              //将 FDIF 位置 0,PWM 异常检测中断标志位,需软件清零

    IP2 |= 0x40;                  //将 PPWM 位置 1，使能 PWM 中断为最高优先级中断
    //使能 PWM 波形发生器
    PWMCR |= 0x80;          //将 ENPWM 位置 1，使能 PWM 波形发生器，PWM 计数器开始计数
    PWMCR &= 0xBF;              //将 ECBI 位置 0，禁止 PWM 计数器归零中断
}
```

然后，编写 PWM 中断服务函数，一旦进入中断则软件清除相应中断标志位，代码如下。

程序清单：中断服务函数

```
/*********************************************************************
 * 描   述 : PWM 中断服务函数
 * 入   参 : 无
 * 返回值 : 无
 *********************************************************************/
void PWM(void) interrupt PWM_VECTOR using 1
```

```
{
    PWMIF &= 0xBF;              //将 CBIF 位置 0，PWM 计数器归零中断标志位，需软件清零
    PWMIF &= 0xFE;                 //将 C2IF 位置 0，PWM2 中断标志位，需软件清零
    PWMIF &= 0xFD;                 //将 C3IF 位置 0，PWM3 中断标志位，需软件清零
}
```

最后，在主函数中对用到的 PWM 口（P2.7 和 P4.5）进行模式配置，调用 PWM2 和 PWM3 初始化函数，开启总中断，在主循环中不断改变 PWM2 和 PWM3 翻转计数器 T1 和 T2 值实现对 PWM2 和 PWM3 输出信号占空比的调节。

代码清单：主函数

```
int main()
{
    uint8 flag=1;
    uint16 ledpwmval=0;
//////////////////////////////////////////////
//注意: STC15W4K32S4 系列的芯片,上电后所有与 PWM 相关的 IO 口均为
//       高阻态,需将这些口设置为准双向口或强推挽模式方可正常使用
//相关 IO: P0.6/P0.7/P1.6/P1.7/P2.1/P2.2
//         P2.3/P2.7/P3.7/P4.2/P4.4/P4.5
//////////////////////////////////////////////
    P2M1 &= 0x7F;   P2M0 &= 0x7F;      //设置 P2.7 为准双向口
    P4M1 &= 0xDF;   P4M0 &= 0xDF;      //设置 P4.5 为准双向口

    PWM2PWM3_Configuration();    //初始化 PWM2 和 PWM3 口
    EA = 1;                          //允许总中断

  while(1)
  {
        delay_ms(30);          //延迟每次指示灯亮度的时间，更方便观察实验现象

        if(flag)               //如果标识符为 1 则递增变量 ledpwmval
            ledpwmval++;
        else                   //如果标识符为 0 则递减变量 ledpwmval
            ledpwmval--;

        if(ledpwmval>248)      //如果变量 ledpwmval 递增到一定值则控制标识符为 0，以实现
ledpwmval 递减
            flag=0;
        if(ledpwmval==1)       //如果变量 ledpwmval 递减到一定值则控制标识符为 1，以实现
ledpwmval 递增
            flag=1;
```

```
30.
31.         P_SW2 |= 0x80;    //将 EAXSFR 位置 1，以访问 PWM 在扩展 RAM 区的特殊功能寄存器
32.         PWM2T1 =(uint16)ledpwmval;     //赋值 PWM2 第一次翻转计数器值（不断变化值）
33.         PWM2T2 = 0x00FA;               //赋值 PWM2 第二次翻转计数器值（定值）
34.         PWM3T1 =(uint16)ledpwmval;     //赋值 PWM3 第一次翻转计数器值（不断变化值）
35.         PWM3T2 = 0x00FA;               //赋值 PWM3 第二次翻转计数器值（定值）
36.         P_SW2 &= 0x7F;                   //将 EAXSFR 位置 0，恢复访问 XRAM
37.
38.     }
39. }
```

4.3.4. 硬件连接

本实验需要使用 P4.5 驱动蓝色指示灯 DS1、P2.7 驱动红色指示灯 DS2，因此需要按下图所示连接杜邦线和短路帽。

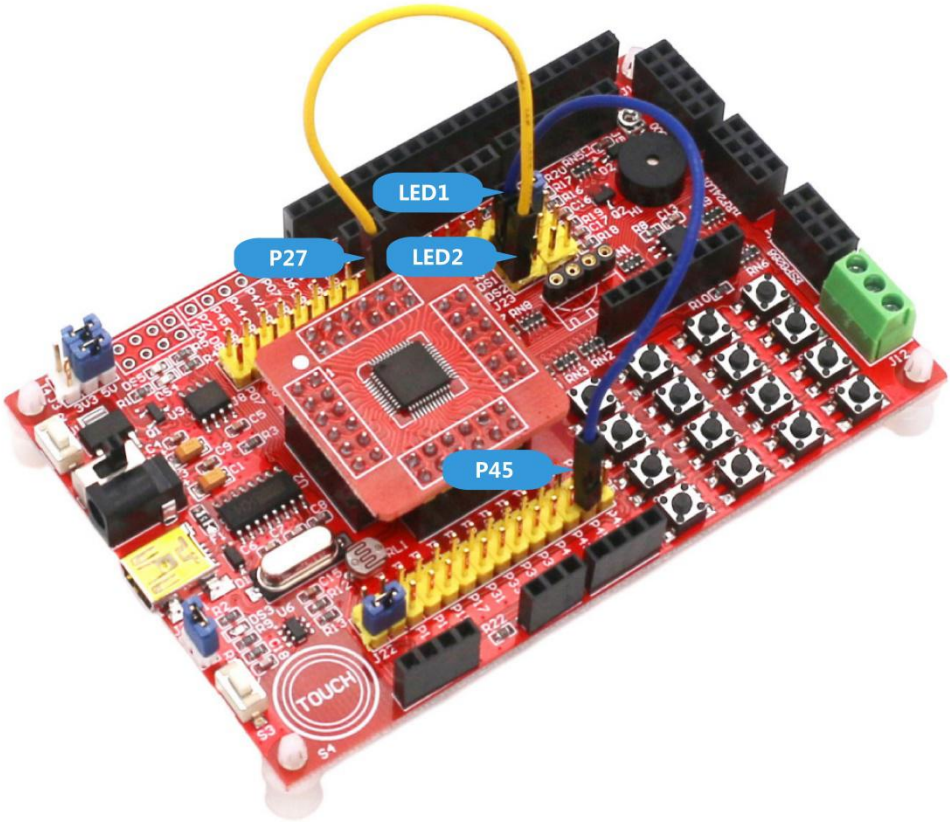


图 12：开发板连接图

4.3.5. 实验步骤

1. 解压“…\第 3 部分：配套例程源码\ 1 - 基础实验程序\”目录下的压缩文件“实验 2-9-1：PWM2 和 PWM3 呼吸灯实验”，将解压后得到的文件夹拷贝到合适的目录，如“D\STC15”。
2. 启动 Keil C51。

3. 在 Keil C51 中执行“Project→Open Project”打开“…\PWM2_PWM3\projec”目录下的工程“PWM2_PWM3.uvproj”。
4. 点击编译按钮编译工程。注意查看编译输出栏，观察编译的结果，如果有错误，修改程序，直到编译成功为止。编译后生成的 HEX 文件“PWM2_PWM3.hex”位于工程目录下的“Output”文件夹中。
5. 打开 STC-ISP 软件下载程序 。下载使用内部 IRC 时钟，IRC 频率选择为 11.0592MHZ。
6. 程序运行后，蓝色指示灯 DS1 和红色指示灯 DS2 亮度均会从暗变亮，再从亮变暗，如此循环不停。

4.4. PWM4 和 PWM5 呼吸灯实验

✧ 注：本节的实验源码是在“实验 2-9-1：PWM2 和 PWM3 呼吸灯实验”的基础上修改。本节对应的实验源码是：“实验 2-9-2：PWM4 和 PWM5 呼吸灯实验”。

4.4.1. 工程需要用到的 c 文件

本实验需要用到的头文件以及添加头文件包含路径的方法请参考“实验 2-9-1：PWM2 和 PWM3 呼吸灯实验”部分。

4.4.2. 编写代码

首先，在 pwm.c 文件中编写 PWM4 和 PWM5 的初始化函数 PWM4PWM5_Configuration，代码如下。

程序清单：PWM4 和 PWM5 初始化函数

```
1.  /*******************************************************************************
2.  功能描述：对 PWM4 和 PWM5 进行初始化
3.  入口参数：无
4.  返回值：无
5.   *******************************************************************************/
6.  void PWM4PWM5_Configuration(void)
7.  {
8.      PWMCFG &= 0xBF;                 //将 CBTADC 位置 0，即 PWM 计数器归零时不触发 ADC 转换
9.      PWMIF &= 0xBF;                  //将 CBIF 位置 0，PWM 计数器归零中断标志位，需软件清零
10.
11.     P_SW2 |= 0x80;          //将 EAXSFR 位置 1，以访问 PWM 在扩展 RAM 区的特殊功能寄存器
12.     //对 PWM4 的初始化部分
13.     PWM4CR |= 0x08;             //将 PWM4_PS 位置 1，选择 PWM4 的输出引脚是 P4.4
14.     PWMCR |= 0x04;      //将 ENC4O 位置 1，PWM4 的端口为 PWM 输出口，受 PWM 波形发生器控制
15.     PWMCFG &= 0xFB;           //将 C4INI 位置 0，设置 PWM4 输出端口的初始电平为低电平
16.     PWMIF &= 0xFB;                  //将 C4IF 位置 0，PWM4 中断标志位，需软件清零
17.     PWM4CR |= 0x04;             //将 EPWM4I 位置 1，使能 PWM4 中断
18.     PWM4CR &= 0xFD;             //将 EC4T2SI 位置 0，关闭 T2 翻转时中断
19.     PWM4CR &= 0xFE;             //将 EC4T1SI 位置 0，关闭 T1 翻转时中断
20.     //对 PWM5 的初始化部分
```

```
    PWM5CR |= 0x08;                 //将 PWM5_PS 位置 1，选择 PWM5 的输出引脚是 P4.2
    PWMCR |= 0x08;          //将 ENC5O 位置 1,PWM5 的端口为 PWM 输出口,受 PWM 波形发生器控制
    PWMCFG &= 0xF7;           //将 C5INI 位置 0，设置 PWM5 输出端口的初始电平为低电平
    PWMIF &= 0xF7;                  //将 C5IF 位置 0，PWM5 中断标志位，需软件清零
    PWM5CR |= 0x04;                 //将 EPWM5I 位置 1，使能 PWM5 中断
    PWM5CR &= 0xFD;                 //将 EC5T2SI 位置 0，关闭 T2 翻转时中断
    PWM5CR &= 0xFE;                 //将 EC5T1SI 位置 0，关闭 T1 翻转时中断
    //对 PWM6 和 PWM7 翻转计数器赋初值
    PWM4T1 =1;                      //赋值 PWM4 第一次翻转计数器值
    PWM4T2 = 0x00FA;                //赋值 PWM4 第二次翻转计数器值
    PWM5T1 =1;                      //赋值 PWM5 第一次翻转计数器值
    PWM5T2 = 0x00FA;                //赋值 PWM5 第二次翻转计数器值

  //对 PWM 波形发生器时钟源进行初始化
    PWMCKS |= 0x10;                 //将 SELT2 位置 1，PWM 时钟源为定时器 2 溢出脉冲
    PWMC = 0x00FA;                 //PWM 计数器赋值（同时对 PWMCH 和 PWMCL 进行了赋值）
    AUXR |= 0x04;                   //定时器 2 时钟为 Fosc,即 1T
    T2L = 0xE0;                     //设定定时初值
    T2H = 0xFE;                     //设定定时初值
    AUXR |= 0x10;                   //启动定时器 2
    P_SW2 &= 0x7F;                  //将 EAXSFR 位置 0，恢复访问 XRAM

    //PWM 外部异常控制寄存器的操作
    PWMFDCR &= 0xDF;                //将 ENFD 位置 0，关闭 PWM 外部异常检测功能
    PWMFDCR &= 0xF7;                //将 ENDI 位置 0，关闭 PWM 异常检测中断
    PWMFDCR &= 0xFB;                //将 FDCMP 位置 0，比较器与 PWM 无关
    PWMFDCR &= 0xFD;                //将 FDIO 位置 0，P2.4 的状态与 PWM 无关
    PWMFDCR &= 0xFE;                //将 FDIF 位置 0,PWM 异常检测中断标志位,需软件清零

    IP2 |= 0x40;                    //将 PPWM 位置 1，使能 PWM 中断为最高优先级中断
    //使能 PWM 波形发生器
    PWMCR |= 0x80;          //将 ENPWM 位置 1，使能 PWM 波形发生器，PWM 计数器开始计数
    PWMCR &= 0xBF;                  //将 ECBI 位置 0，禁止 PWM 计数器归零中断
}
```

然后，编写 PWM 中断服务函数，一旦进入中断则软件清除相应中断标志位，代码如下。

程序清单：中断服务函数

```
/*********************************************************************
 * 描   述 : PWM 中断服务函数
 * 入   参 : 无
 * 返回值 : 无
 *********************************************************************/
```

```
6.  void PWM(void) interrupt PWM_VECTOR using 1
7.  {
8.      PWMIF &= 0xBF;            //将 CBIF 位置 0，PWM 计数器归零中断标志位，需软件清零
9.      PWMIF &= 0xFB;                //将 C4IF 位置 0，PWM4 中断标志位，需软件清零
10.     PWMIF &= 0xF7;                //将 C5IF 位置 0，PWM5 中断标志位，需软件清零
11. }
```

最后，在主函数中对用到的 PWM 口（P4.2 和 P4.4）进行模式配置，调用 PWM4 和 PWM5 初始化函数，开启总中断，在主循环中不断改变 PWM4 和 PWM5 翻转计数器 T1 和 T2 值实现对 PWM4 和 PWM5 输出信号占空比的调节。

代码清单：主函数

```
1.  int main()
2.  {
3.      uint8 flag=1;
4.      uint16 ledpwmval=0;
5.  ////////////////////////////////////////////////
6.  //注意: STC15W4K32S4 系列的芯片,上电后所有与 PWM 相关的 IO 口均为
7.  //       高阻态,需将这些口设置为准双向口或强推挽模式方可正常使用
8.  //相关 IO: P0.6/P0.7/P1.6/P1.7/P2.1/P2.2
9.  //         P2.3/P2.7/P3.7/P4.2/P4.4/P4.5
10. ////////////////////////////////////////////////
11.     P4M1 &= 0xEB;   P4M0 &= 0xEB;      //设置 P4.2、P4.4 为准双向口
12.
13.     PWM4PWM5_Configuration();    //初始化 PWM4 和 PWM5 口
14.     EA = 1;                          //允许总中断
15.
16.   while(1)
17.   {
18.         delay_ms(30);          //延迟每次指示灯亮度的时间，更方便观察实验现象
19.
20.         if(flag)               //如果标识符为 1 则递增变量 ledpwmval
21.             ledpwmval++;
22.         else                   //如果标识符为 0 则递减变量 ledpwmval
23.             ledpwmval--;
24.
25.         if(ledpwmval>248)      //如果变量 ledpwmval 递增到一定值则控制标识符为 0，以实现
    ledpwmval 递减
26.             flag=0;
27.         if(ledpwmval==1)       //如果变量 ledpwmval 递减到一定值则控制标识符为 1，以实现
    ledpwmval 递增
28.             flag=1;
```

```
29.
30.         P_SW2 |= 0x80;   //将 EAXSFR 位置 1，以访问 PWM 在扩展 RAM 区的特殊功能寄存器
31.         PWM4T1 =(uint16)ledpwmval;     //赋值 PWM4 第一次翻转计数器值（不断变化值）
32.         PWM4T2 = 0x00FA;               //赋值 PWM4 第二次翻转计数器值（定值）
33.         PWM5T1 =(uint16)ledpwmval;     //赋值 PWM5 第一次翻转计数器值（不断变化值）
34.         PWM5T2 = 0x00FA;               //赋值 PWM5 第二次翻转计数器值（定值）
35.         P_SW2 &= 0x7F;                   //将 EAXSFR 位置 0，恢复访问 XRAM
36.
37.     }
38. }
```

4.4.3. 硬件连接

本实验需要使用 P4.2 驱动蓝色指示灯 DS1、P4.4 驱动红色指示灯 DS2，因此需要按下图所示连接杜邦线和短路帽。

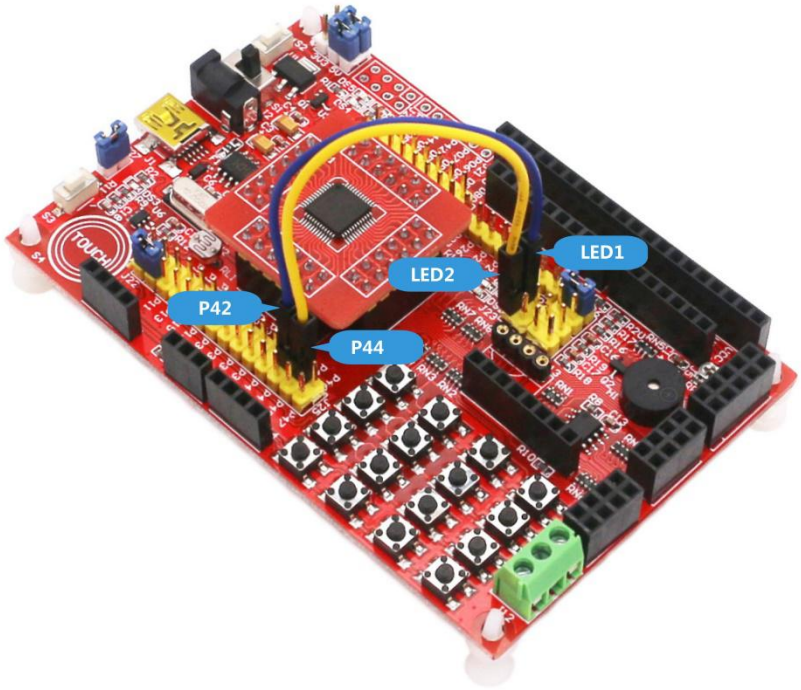


图 13：开发板连接图

4.4.4. 实验步骤

1. 解压“…\第 3 部分：配套例程源码\ 1 - 基础实验程序\”目录下的压缩文件“实验 2-9-2：PWM4 和 PWM5 呼吸灯实验”，将解压后得到的文件夹拷贝到合适的目录，如“D\STC15”。
2. 启动 Keil C51。
3. 在 Keil C51 中执行“Project→Open Project”打开“…\PWM4_PWM5\projec”目录下的工程“PWM4_PWM5.uvproj”。

4. 点击编译按钮编译工程。注意查看编译输出栏，观察编译的结果，如果有错误，修改程序，直到编译成功为止。编译后生成的 HEX 文件“PWM4_PWM5.hex”位于工程目录下的“Output”文件夹中。
5. 打开 STC-ISP 软件下载程序 。下载使用内部 IRC 时钟，IRC 频率选择为 11.0592MHZ。
6. 程序运行后，蓝色指示灯 DS1 和红色指示灯 DS2 亮度均会从暗变亮，再从亮变暗，如此循环不停。

4.5. PWM6 和 PWM7 呼吸灯实验

✧ 注：本节的实验源码是在“实验 2-9-1：PWM2 和 PWM3 呼吸灯实验”的基础上修改。本节对应的实验源码是：“实验 2-9-3：PWM6 和 PWM7 呼吸灯实验”。

4.5.1. 工程需要用到的 c 文件

本实验需要用到的头文件以及添加头文件包含路径的方法请参考“实验 2-9-1：PWM2 和 PWM3 呼吸灯实验”部分。

4.5.2. 编写代码

首先，在 pwm.c 文件中编写 PWM6 和 PWM7 的初始化函数 PWM6PWM7_Configuration，代码如下。

程序清单：PWM6 和 PWM7 初始化函数

```
1.  /*************************************************************************
2.  功能描述：对 PWM6 和 PWM7 进行初始化
3.  入口参数：无
4.  返回值：无
5.   *************************************************************************/
6.  void PWM6PWM7_Configuration(void)
7.  {
8.      PWMCFG &= 0xBF;                 //将 CBTADC 位置 0,即 PWM 计数器归零时不触发 ADC 转换
9.      PWMIF &= 0xBF;                  //将 CBIF 位置 0,PWM 计数器归零中断标志位,需软件清零
10.
11.     P_SW2 |= 0x80;        //将 EAXSFR 位置 1，以访问 PWM 在扩展 RAM 区的特殊功能寄存器
12.     //对 PWM6 的初始化部分
13.     PWM6CR |= 0x08;               //将 PWM6_PS 位置 1，选择 PWM6 的输出引脚是 P0.7
14.     PWMCR |= 0x10;      //将 ENC6O 位置 1,PWM6 的端口为 PWM 输出口,受 PWM 波形发生器控制
15.     PWMCFG &= 0xEF;             //将 C6INI 位置 0，设置 PWM6 输出端口的初始电平为低电平
16.     PWMIF &= 0xEF;                  //将 C6IF 位置 0，PWM6 中断标志位，需软件清零
17.     PWM6CR |= 0x04;               //将 EPWM6I 位置 1，使能 PWM6 中断
18.     PWM6CR &= 0xFD;               //将 EC6T2SI 位置 0，关闭 T2 翻转时中断
19.     PWM6CR &= 0xFE;               //将 EC6T1SI 位置 0，关闭 T1 翻转时中断
20.     //对 PWM7 的初始化部分
21.     PWM7CR |= 0x08;               //将 PWM7_PS 位置 1，选择 PWM7 的输出引脚是 P0.6
```

```
    PWMCR |= 0x20;      //将 ENC7O 位置 1,PWM5 的端口为 PWM 输出口,受 PWM 波形发生器控制
    PWMCFG &= 0xDF;           //将 C7INI 位置 0，设置 PWM7 输出端口的初始电平为低电平
    PWMIF &= 0xDF;              //将 C7IF 位置 0，PWM7 中断标志位，需软件清零
    PWM7CR |= 0x04;           //将 EPWM7I 位置 1，使能 PWM7 中断
    PWM7CR &= 0xFD;           //将 EC7T2SI 位置 0，关闭 T2 翻转时中断
    PWM7CR &= 0xFE;           //将 EC7T1SI 位置 0，关闭 T1 翻转时中断
    //对 PWM6 和 PWM7 翻转计数器赋初值
    PWM6T1 =1;                //赋值 PWM6 第一次翻转计数器值
    PWM6T2 = 0x00FA;          //赋值 PWM6 第二次翻转计数器值
    PWM7T1 =1;                //赋值 PWM7 第一次翻转计数器值
    PWM7T2 = 0x00FA;          //赋值 PWM7 第二次翻转计数器值

  //对 PWM 波形发生器时钟源进行初始化
    PWMCKS |= 0x10;             //将 SELT2 位置 1，PWM 时钟源为定时器 2 溢出脉冲
    PWMC = 0x00FA;            //PWM 计数器赋值（同时对 PWMCH 和 PWMCL 进行了赋值）
    AUXR |= 0x04;               //定时器 2 时钟为 Fosc,即 1T
    T2L = 0xE0;               //设定定时初值
    T2H = 0xFE;               //设定定时初值
    AUXR |= 0x10;             //启动定时器 2
    P_SW2 &= 0x7F;              //将 EAXSFR 位置 0，恢复访问 XRAM

    //PWM 外部异常控制寄存器的操作
    PWMFDCR &= 0xDF;              //将 ENFD 位置 0，关闭 PWM 外部异常检测功能
    PWMFDCR &= 0xF7;              //将 ENDI 位置 0，关闭 PWM 异常检测中断
    PWMFDCR &= 0xFB;              //将 FDCMP 位置 0，比较器与 PWM 无关
    PWMFDCR &= 0xFD;              //将 FDIO 位置 0，P2.4 的状态与 PWM 无关
    PWMFDCR &= 0xFE;              //将 FDIF 位置 0,PWM 异常检测中断标志位,需软件清零

    IP2 |= 0x40;                  //将 PPWM 位置 1，使能 PWM 中断为最高优先级中断
    //使能 PWM 波形发生器
    PWMCR |= 0x80;          //将 ENPWM 位置 1，使能 PWM 波形发生器，PWM 计数器开始计数
    PWMCR &= 0xBF;              //将 ECBI 位置 0，禁止 PWM 计数器归零中断
}
```

然后，编写 PWM 中断服务函数，一旦进入中断则软件清除相应中断标志位，代码如下。

程序清单：中断服务函数

```
/*************************************************************************
 * 描   述 : PWM 中断服务函数
 * 入   参 : 无
 * 返回值 : 无
 *************************************************************************/
void PWM(void) interrupt PWM_VECTOR using 1
```

```
{
    PWMIF &= 0xBF;              //将 CBIF 位置 0，PWM 计数器归零中断标志位，需软件清零
    PWMIF &= 0xEF;                  //将 C6IF 位置 0，PWM6 中断标志位，需软件清零
    PWMIF &= 0xDF;                  //将 C7IF 位置 0，PWM7 中断标志位，需软件清零
}
```

最后，在主函数中对用到的 PWM 口（P0.6 和 P07）进行模式配置，调用 PWM6 和 PWM7 初始化函数，开启总中断，在主循环中不断改变 PWM6 和 PWM7 翻转计数器 T1 和 T2 值实现对 PWM6 和 PWM7 输出信号占空比的调节。

代码清单：主函数

```
int main()
{
    uint8 flag=1;
    uint16 ledpwmval=0;
////////////////////////////////////////////////
//注意: STC15W4K32S4 系列的芯片,上电后所有与 PWM 相关的 IO 口均为
//       高阻态,需将这些口设置为准双向口或强推挽模式方可正常使用
//相关 IO: P0.6/P0.7/P1.6/P1.7/P2.1/P2.2
//         P2.3/P2.7/P3.7/P4.2/P4.4/P4.5
////////////////////////////////////////////////
    P0M1 &= 0x3F;   P0M0 &= 0x3F;     //设置 P0.6、P0.7 为准双向口

    PWM6PWM7_Configuration();   //初始化 PWM6 和 PWM7 口
    EA = 1;                         //允许总中断

  while(1)
  {
        delay_ms(30);         //延迟每次指示灯亮度的时间，更方便观察实验现象

        if(flag)              //如果标识符为 1 则递增变量 ledpwmval
            ledpwmval++;
        else                  //如果标识符为 0 则递减变量 ledpwmval
            ledpwmval--;

        if(ledpwmval>248)     //如果变量 ledpwmval 递增到一定值则控制标识符为 0，以实现
ledpwmval 递减
            flag=0;
        if(ledpwmval==1)      //如果变量 ledpwmval 递减到一定值则控制标识符为 1，以实现
ledpwmval 递增
            flag=1;

```

```
30.         P_SW2 |= 0x80;    //将 EAXSFR 位置 1，以访问 PWM 在扩展 RAM 区的特殊功能寄存器
31.         PWM6T1 =(uint16)ledpwmval;     //赋值 PWM6 第一次翻转计数器值（不断变化值）
32.         PWM6T2 = 0x00FA;               //赋值 PWM6 第二次翻转计数器值（定值）
33.         PWM7T1 =(uint16)ledpwmval;     //赋值 PWM7 第一次翻转计数器值（不断变化值）
34.         PWM7T2 = 0x00FA;               //赋值 PWM7 第二次翻转计数器值（定值）
35.         P_SW2 &= 0x7F;                   //将 EAXSFR 位置 0，恢复访问 XRAM
36.
37.     }
38. }
```

4.5.3. 硬件连接

本实验需要使用 P0.6 驱动蓝色指示灯 DS1、P0.7 驱动红色指示灯 DS2，因此需要按下图所示连接短路帽。

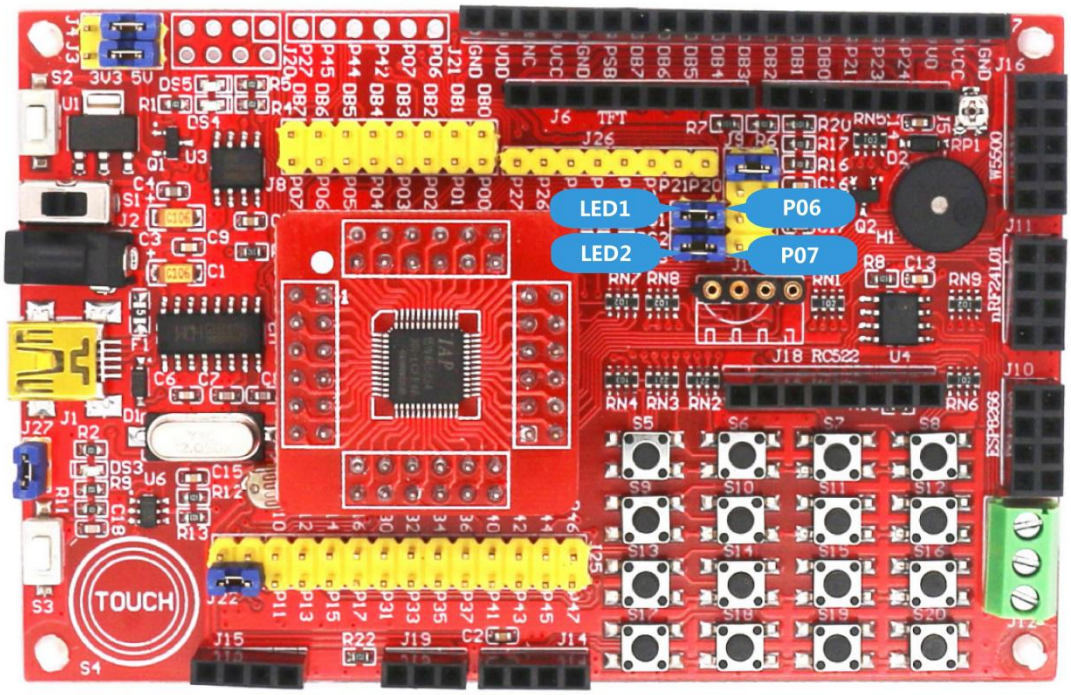


图 14：开发板连接图

4.5.4. 实验步骤

1. 解压“…\第 3 部分：配套例程源码\ 1 - 基础实验程序\”目录下的压缩文件“实验 2-9-3：PWM6 和 PWM7 呼吸灯实验”，将解压后得到的文件夹拷贝到合适的目录，如“D\STC15”。
2. 启动 Keil C51。
3. 在 Keil C51 中执行“Project→Open Project”打开“…\PWM6_PWM7\projec”目录下的工程“PWM6_PWM7.uvproj”。
4. 点击编译按钮编译工程。注意查看编译输出栏，观察编译的结果，如果有错误，修改程

序，直到编译成功为止。编译后生成的 HEX 文件“PWM6_PWM7.hex”位于工程目录下的“Output”文件夹中。

5. 打开 STC-ISP 软件下载程序 。下载使用内部 IRC 时钟，IRC 频率选择为 11.0592MHZ。
6. 程序运行后，蓝色指示灯 DS1 和红色指示灯 DS2 亮度均会从暗变亮，再从亮变暗，如此循环不停。