第 2-16 讲：可编程计数器阵列 PCA 定时器/计数器

1. 学习目的

1. 掌握 STC8A8K64D4 系列 PCA 可编程计数器阵列的原理。
2. 掌握 4 个 PCA 外设相关寄存器配置及程序设计。

2. PCA 概述

PCA 全称是可编程计数器阵列（Programable Counter Array），其中 P 表示可以编程控制、C 表示计数器、A 表示阵列，即有多路通道。他和我们前面学习的定时/计数器类似，对于初学者，可以把他理解为一个功能更加强大的定时/计数器。

PCA 可以根据不同的应用需求进行编程，具有较高的灵活性和可靠性，因此他在工业自动化、仪器仪表等领域得到广泛的应用。

STC8A8K64D4 系列单片机内部集成了 4 组可编程计数器阵列模块，可用于软件定时器、外部脉冲捕获、高速脉冲输出和 PWM 脉宽调制输出，其结构图如下所示。

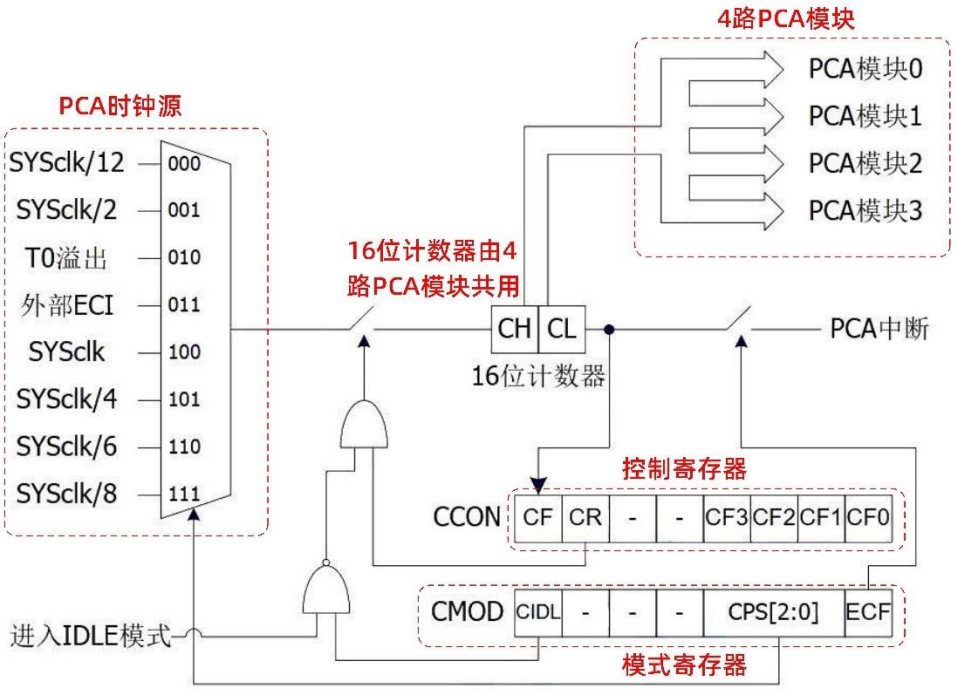

图 1：PCA 结构图

PCA 外设包含 4 路 PCA 模块，这 4 路 PCA 模块使用同一个可配置时钟源的 16 位计数器， 并且他们使用的模式寄存器 CCON 和控制寄存器 CMOD 也是同一个，我们在使用时应注意这一点，因为，对他们的配置会同时对 4 个 PCA 模块有效。

除了共有的配置，每个 PCA 模块还有一个独立的配置寄存器“PCA 模块模式控制寄存器（CCAPMn，n=0~4）”，用于配置 PCA 模块的比较、捕获、高速脉冲输出和 PWM 脉宽

调制输出，其中 PCA 模块工作于 PWM 模式时，还有一个寄存器“PCA 模块 PWM 模式控制寄存器（PCA_PWMn）”用于配置 PWM 的位宽。

PCA 模块的工作模式有 4 种：软件定时器、外部脉冲捕获、高速脉冲输出和 PWM 脉宽调制输出。PCA 模块工作于捕获、高速脉冲输出和 PWM 时，需要通过“P_SW1”寄存器配置关联的引脚，用于连接输入的脉冲、输出高速脉冲和PWM。工作于软件定时器模式时，是没有关联引脚的。

3. STC8A8K64D4 的 PCA 应用步骤

3.1. 软件定时器

PCA 模块工作于软件定时器时的应用流程如下图所示。

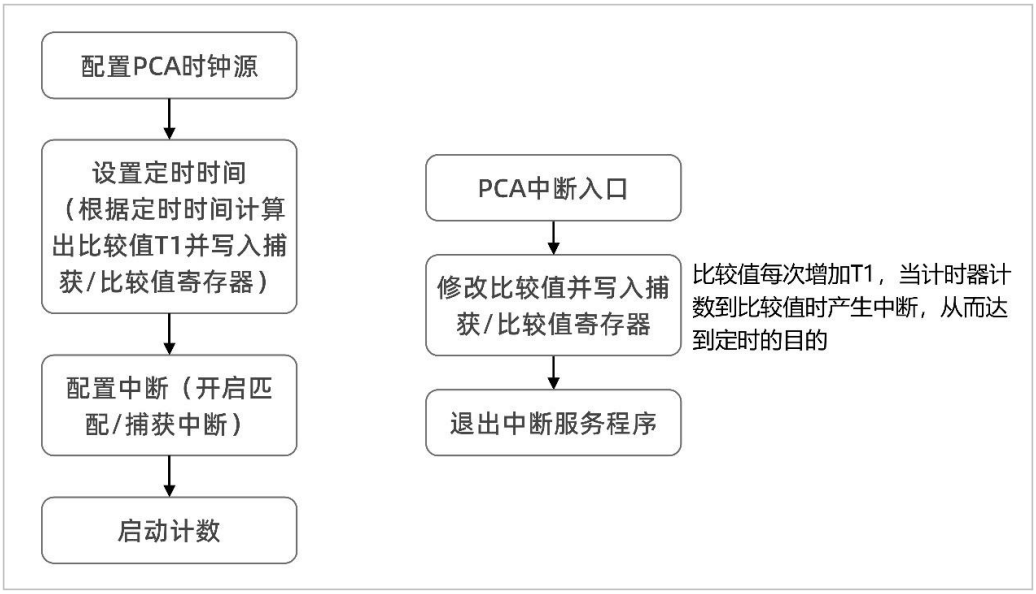

图 2：PCA 工作于软件定时器模式时的应用流程

3.1.1. 配置 PCA 时钟源

PCA 时钟源由“PCA 模式寄存器（CMOD）”中的 CPS[2:0]位配置，此外，还需配置 CIDL 位用于确定计数器在空闲模式下是否计数，如下所示。

PCA 模式寄存器（CMOD）：

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CMOD	D9H	CIDL	-	-	-	CPS[2:0]			ECF
		空闲模式下是否计数				时钟源配置位			

- CIDL：空闲模式下是否停止 PCA 计数。
  - 0：空闲模式下 PCA 继续计数。
  - 1：空闲模式下 PCA 停止计数。
- CPS[2:0]：PCA 计数脉冲源选择位，可配置项如下表所示。

表 1：PCA 时钟源

CPS[2:0]	PCA 的输入时钟源
000	系统时钟/12
001	系统时钟/2
010	定时器 0 的溢出脉冲
011	ECI 脚的外部输入时钟
100	系统时钟
101	系统时钟/4
110	系统时钟/6
111	系统时钟/8

3.1.2. 设置定时时间

PCA 应用于软件定时器时和定时器类似，都是通过对时钟的计数达到定时的目的，所不同的是定时器设置了计数初值后，就不用软件干预了，每次溢出后硬件都会自动重装计数初值，而 PCA 软件定时器则需要软件干预。PCA 软件定时器没有自动重装的机制，每次匹配中断后，都需要修改匹配值（在当前匹配值的基础上加上定时时间对应的计数值）。

使用 PCA 软件定时器时，“PCA 计数器寄存器（CL，CH）”的值通常设置为 0，然后根据定时时间、PCA 时钟源计算出比较值。PCA 启动计数后，计数器的值向上递增，当计数器的值和比较值匹配时产生匹配中断，因此，对于 PCA 软件定时器来说，需要计算的是比较值。

- 比较值计算示例如下（定时时间 10ms）：

  系统时钟：24MHz。

  PCA 时钟源：系统时钟/12 = 2Mhz，即一个时钟的时间是：1/2000000 秒 = 0.5us。

  比较值 = 10000us/0.5us = 20000，对应的 16 进制为 0x4E20。因此，比较寄存器初次赋值为（CCAP0L = 0x20; CCAP0H = 0x4E;），之后，在 PCA 中断服务函数中，每次加上这个数值后再对比较寄存器赋值。

3.1.3. 中断配置

中断配置包括中断的开启/关闭和中断优先级的配置。

PCA 计数器溢出中断 PCA 的 4 个模块各有一个匹配/捕获中断，中断的开启和关闭由中断使能寄存器 IE 的位 1（ET0）和位 3（ET1）控制，如下图所示。

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CMOD	D9H	CIDL	-	-	-	CPS[2:0]			ECF

PCA计数器溢出中断允许位

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CCAPM0	DAH	-	ECOM0	CCAPP0	CCAPN0	MAT0	TOG0	PWM0	ECCF0
CCAPM1	DBH	-	ECOM1	CCAPP1	CCAPN1	MAT1	TOG1	PWM1	ECCF1
CCAPM2	DCH	-	ECOM2	CCAPP2	CCAPN2	MAT2	TOG2	PWM2	ECCF2
CCAPM3	FD54H	-	ECOM3	CCAPP3	CCAPN3	MAT3	TOG3	PWM3	ECCF3

PCA模块匹配/捕获中断允许位

- ECF：PCA 计数器溢出中断允许位。
  - 0：禁止 PCA 计数器溢出中断。
  - 1：使能 PCA 计数器溢出中断。
- ECCFn：允许 PCA 模块 n 的匹配/捕获中断
  - 0：禁止 PCA 模块 n 的匹配/捕获中断。
  - 1：使能 PCA 模块 n 的匹配/捕获中断。

PCA 的中断优先级由 IP 和 IPH 寄存器位 7 的组合配置，如下图所示。

PCA中断优先级配置位

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IP	B8H	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0
IPH	B7H	PPCAH	PLVDH	PADCH	PSH	PT1H	PX1H	PT0H	PX0H
IP2	B5H	-	PI2C	PCMP	PX4	PPWMFD	PPWM	PSPI	PS2

图 6：PCA 中断优先级配置

- PPCAH，PPCA：CCP/PCA/PWM 中断优先级控制位
  - 00： CCP/PCA/PWM 中断优先级为 0 级（最低级）。
  - 01： CCP/PCA/PWM 中断优先级为 1 级（较低级）。
  - 10： CCP/PCA/PWM 中断优先级为 2 级（较高级）。
  - 11： CCP/PCA/PWM 中断优先级为 3 级（最高级）。

✍ **中断优先级配置示例**：配置 PCA 中断优先级为最高优先级 3

```
IP  |= 0x80;  //中断优先级配置为最高优先级：PPCAH = 1，PPCA = 1
IPH |= 0x80;
```

✧ **注意事项**：PCA 开启中断的情况下，还需要开启总中断“EA=1”，中断才能起作用。通常，我们会在主函数“main()”中开启总中断，这是因为我们开发的程序中可能会使用到多个中断，在主函数中开启一次即可。

3.1.4. 启动和停止 PCA 计数器

PCA 是通过置位“PCA 控制寄存器（CCON）”中的“CR”位启动 PCA 计数器、通过清零“CR”位停止 PCA 计数器的，如下所示。

PCA 控制寄存器（CCON）：

PCA计数器允许控制位

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CCON	D8H	CF	CR	-	-	CCF3	CCF2	CCF1	CCF0

- CR：PCA 计数器允许控制位。
  - 0：停止 PCA 计数。
  - 1：启动 PCA 计数。

3.2. 输出高速脉冲

PCA 工作于输出高速脉冲模式时和软件定时器类似，区别是高速脉冲模式需要配置 PCA 模块的功能引脚，用于输出脉冲。

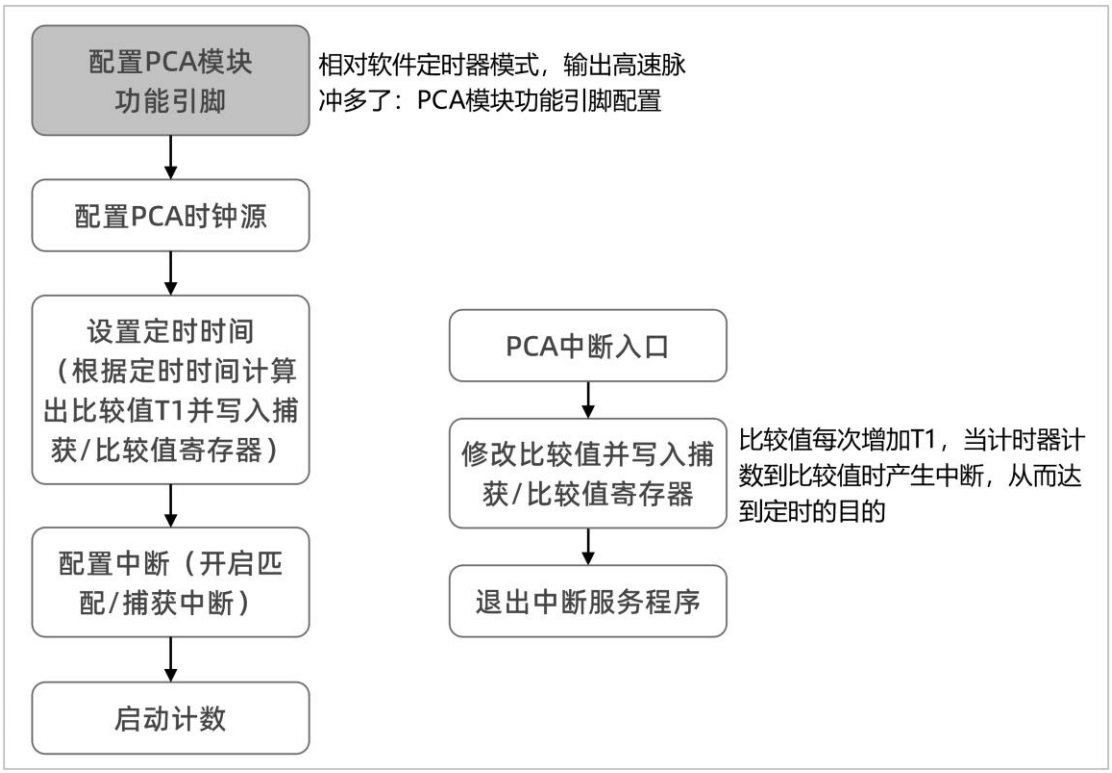

图 3：PCA 工作于输出高速脉冲模式时的应用流程

这里的定时时间决定了脉冲的频率/周期，定时时间计算比较值的方式和软件定时器一样。比较值写入捕获值/比较值寄存器并启动 PCA 计数后，当 PCA 计数器的计数值与 PCA 模块捕获值/比较值寄存器的值相匹配时，PCA 模块关联的引脚输出将发生翻转。需要注意的是：一个周期需要 2 倍的定时时间（输出方波），如下图所示。

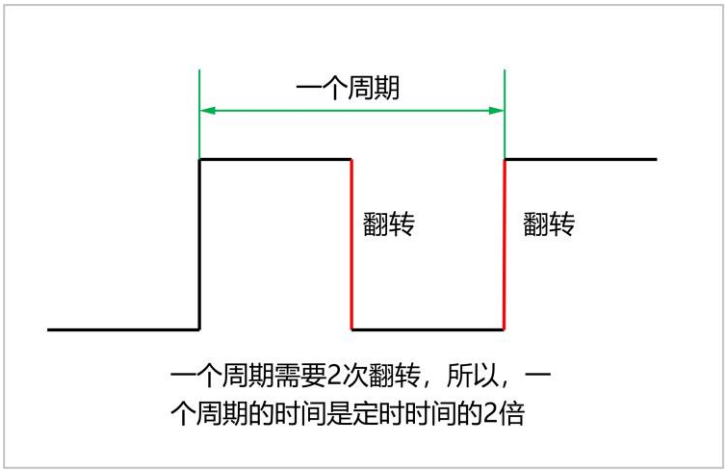

图 4：输出脉冲的周期

3.3. 捕获测量脉冲宽度

PCA 模块工作于捕获模式时，对模块的关联引脚输入跳变进行采样。当采样到有效跳变时，PCA 控制器立即将 PCA 计数器 CH 和 CL 中的计数值装载到模块的捕获寄存器中 CCAPnL 和 CCAPnH，同时将 CCON 寄存器中相应的 CCFn 置 1。若 CCAPMn 中的 ECCFn 位被设置为 1，将产生中断。

中断服务程序中，读出 PCA 计数值并保存，以便于计算两次上升沿或两次下降沿之间计数值的差，由这个差值即可计算出两次上升沿或两次下降沿之间的时间间隔。PCA 工作于脉宽测量模式时的应用流程如下所示。

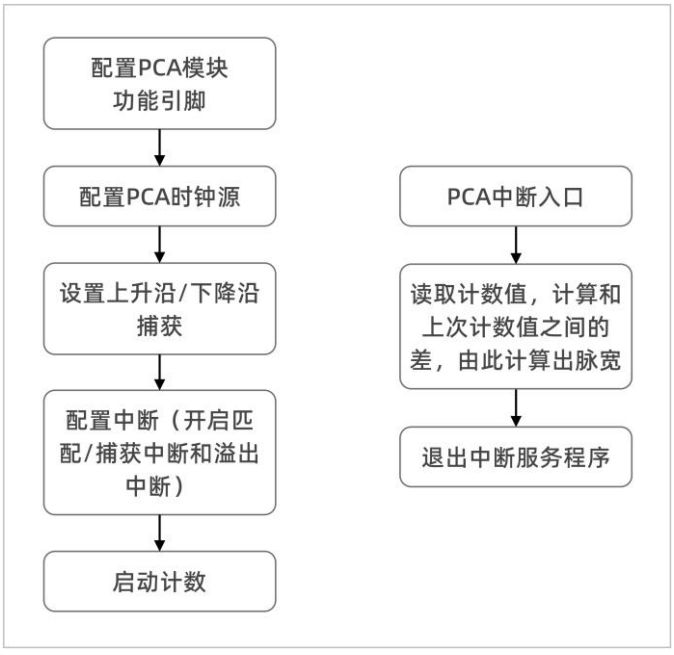

图 5：PCA 工作于脉宽测量模式时的应用流程

捕获脉宽需要注意以下两个方面：

1. PCA 工作于脉宽测量模式时只能设置为捕获上升沿或者捕获下降沿，不能设置同时捕

获上升沿和下降沿。

2. PCA 计数器溢出。

■ **示例**：测量一个周期为 10ms 的方波，PCA 配置为上升沿捕获、系统时钟 24MHz，PCA 时钟源为系统时钟/12（即一个计数值对应的时间是：0.5us），示意图如下所示。

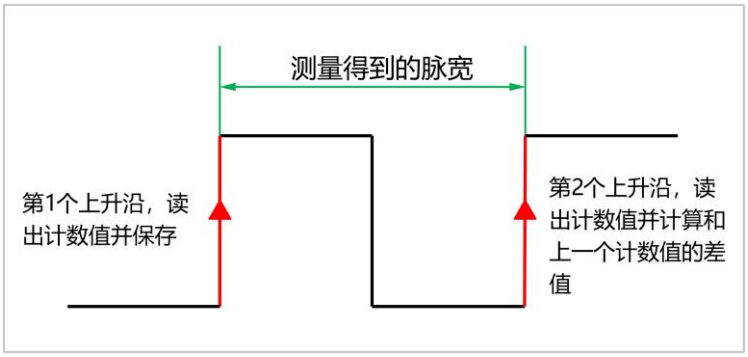

图 6：输出脉冲的周期

第一个上升沿到来后，PCA 进入中断，在中断服务函数中读出 PCA 计数器的值，存储到变量“cnt_value_last”。第二个上升沿到来后，同样在中断服务函数中读出 PCA 计数器的值，存储到变量“cnt_value_current”，求出变量“cnt_value_current”和“cnt_value_last”的差值，再乘以一个计数值对应的时间即为测量的脉宽。

在计算两次上升沿之间的计数值的时候需要考虑 PCA 计数器的溢出问题，通常，我们会开启 PCA 的溢出中断用来跟踪 PCA 计数器的溢出情况，详细的处理方式见实验程序。

3.4. 输出 PWM

PCA 模块工作于 PWM 模式时，可配置为 6/7/8/10 位 PWM。PCA 计数器启动后，PCA 计数器的值向上递增，当 PCA 计数器的值小于设置的比较值时，配置的功能引脚输出低电平；当 PCA 计数器的值大于等于设置的比较值时，配置的功能引脚输出高电平；当 PCA 计数器溢出后，比较值自动重装，由此，实现无需软件干预的 PWM 输出。

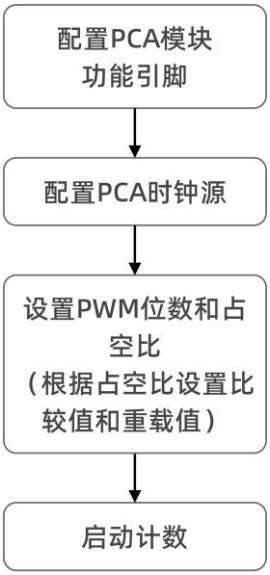

图 7：PCA 工作于 PWM 模式时的应用流程

PCA 工作于 PWM 模式时的时钟源和功能引脚配置和其他模式一样，本节中我们重点关注 PWM 位数、频率（周期）和占空的配置。

PWM 的位数由“PCA 模块 PWM 模式控制寄存器（PCA_PWMn）”中的 CPS[2:0]位配置，如下图所示。

PCA 模块 PWM 模式控制寄存器（PCA_PWMn）：

PCA模块n的PWM位数配置位

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PCA_PWM0	F2H	EBS0[1:0]		XCCAP0H[1:0]		XCCAP0L[1:0]		EPC0H	EPC0L
PCA_PWM1	F3H	EBS1[1:0]		XCCAP1H[1:0]		XCCAP1L[1:0]		EPC1H	EPC1L
PCA_PWM2	F4H	EBS2[1:0]		XCCAP2H[1:0]		XCCAP2L[1:0]		EPC2H	EPC2L
PCA_PWM3	FD57H	EBS3[1:0]		XCCAP3H[1:0]		XCCAP3L[1:0]		EPC3H	EPC3L

PWM 可配置为 6、7、8 或 10 位，PWM 配置的位数不同，他的重载值和比较值的组成也是不一样的，如下图所示。

EBSn[1:0]	PWM 位数	重载值	比较值
00	8 位 PWM	{EPCnH, CCAPnH[7:0]}	{EPCnL, CCAPnL[7:0]}
01	7 位 PWM	{EPCnH, CCAPnH[6:0]}	{EPCnL, CCAPnL[6:0]}
10	6 位 PWM	{EPCnH, CCAPnH[5:0]}	{EPCnL, CCAPnL[5:0]}
11	10 位 PWM	{EPCnH, XCCAPnH[1:0], CCAPnH[7:0]}	{EPCnL, XCCAPnL[1:0], CCAPnL[7:0]}

1. PWM 频率（周期）

PCA 工作于 PWM 模式时，PWM 的频率由 PWM 配置的位数和时钟源决定，另外注意 PCA 的 4 个通道共用同一个时钟源，因此，PCA 的各个模块工作于 PWM 模式时，不能单独设置各个模块的 PWM 频率。

PWM 频率计算公式如下，其中 n 是 PWM 的位数，可取的值为 6、7、8、10：

$$PWM\,频率 = \frac{PCA\,时钟源频率}{2^n}$$

✧ 注：下列情况 PWM 输出固定电平：

- 当 EPCnH=0，XCCAP0H=0 及 CCAPnH=00H 时， PWM 固定输出高。
- 当 EPCnH=1，XCCAP0H=3 及 CCAPnH=FFH 时， PWM 固定输出低。

2. PWM 占空比

PWM 频率确定后，PWM 的位数、比较值和重装值的位数也就确定了，同时，一个 PWM 周期包含多少个计数也确定了，因此，设置比较值和重装值即可设置一个 PWM 周期内高电平的时间，即 PWM 的占空比。（PCA 计数器的初值通常在初始化时设置为 0，计数器启动后向上计数，当 PCA 计数器的值小于设置的比较值时，配置的功能引脚输出低电平；当 PCA 计数器的值大于等于设置的比较值时，配置的功能引脚输出高电平）。

■ 示例：6 位 PWM，占空比：50%

PWM 为 6 位时，一个周期包含的计数值为 64，如果要占空比为 50%，把比较值和重装值设置为 64 的一半即可，即 32，对应的 16 进制为 0x20。

```
//PWM 占空比为 50%
CCAP0L = 0x20;          //PCA 模块 0 的 PWM 比较值
CCAP0H = 0x20;          //PCA 模块 0 的 PWM 重装值
```

■ 示例：6 位 PWM，占空比：20%

```
//PWM 占空比为 25%
CCAP0L = 0x10;          //PCA 模块 0 的 PWM 比较值
CCAP0H = 0x10;          //PCA 模块 0 的 PWM 重装值
```

4. 软件设计

4.1. 软件定时器实验

✧ 注：本节的实验是在“实验 2-1-3：流水灯（自编驱动文件方式）”的基础上修改，本节对应的实验源码是：“实验 2-16-1：PCA 软件定时器实验”。

4.1.1. 实验内容

使用 PCA 模块 0 实现 200ms 的软件定时器，定时时间到了之后翻转指示灯 D1 的状态，即驱动指示灯 D1 以 200ms 的间隔闪烁。PCA 配置如下：

- 系统时钟：24MHz。
- PCA 时钟源：系统时钟/12 = 2Mhz。
- 中断：关闭计数器溢出中断，使能 PCA 模块 0 的匹配/捕获中断，中断优先级设置为 2（较高级）。

4.1.2. 代码编写

1. 新建一个名称为“pca.c”的文件及其头文件“pca.h”并保存到工程的“Source”文件夹，并将“pca.c”加入到 Keil 工程中的“SOURCE”组。
2. 引用头文件

因为在“main.c”文件中使用了“pca.c”文件中的函数，所以需要引用下面的头文件“pca.h”。

代码清单：引用头文件

```
1.  //引用 pca 的头文件
2.  #include    "pca.h"
```

3. PCA 初始化和启动

配置 PCA 时钟源为系统时钟的 1/12，本例中使用的系统时钟是 24MHz，即 PCA 时钟源为 2MHz。PCA 计数值初值设置为 0，之后将定时 10ms 的比较值写入比较寄存器并开启匹配/捕获中断。因为 PCA 软件定时器是通过修改比较值实现的，所以，我们定义一个变量“counter_val”用于存储比较器，每次写入比较值后，“counter_val”的值增加 10ms 对应的

计数值，以备下一次修改比较值时使用。

配置完成后，启动 PCA 计数器，当计数器的计数值等于比较值时，触发匹配/比较中断，代码清单如下。

代码清单：初始化 PCA

```
/*******************************************************************************
 * 描   述 : PCA 初始化，初始化 PCA 模块 0 用于软件定时器，定时时间 10ms
 * 参   数 : 无
 * 返回值 : 无
 *******************************************************************************/
void pca_init(void)
{
    CCON = 0x00;        //CF、CR、CCF1、CCF0~CCF3 位均清零
    /*--------------------PCA 模式寄存器 CMOD 配置---------------------------
      位 7  位 6  位 5  位 4   位 3~位 1    位 0
      CIDL   x    x    x    CPS[2~0]   BCF
       0     x    x    x      000       0
      CIDL=0：空闲模式下仍然计数
      CPS[2~0]=000：PCA 时钟源选择:系统时钟/12
      BCF=0：关闭 PCA 计数器溢出中断
    ------------------------------END-----------------------------------*/
    CMOD = 0x00;
    CL = 0x00;          //PCA 计数器赋初值
    CH = 0x00;          //PCA 计数器赋初值

    IP &= ~0x02;        //中断优先级配置为 2（较高优先级）
    IPH |= 0x02;

    /*------------------PCA 模块 0 模式控制寄存器 CCAPM0 配置------------------
      位 7  位 6     位 5    位 4      位 3    位 2   位 1    位 0
       x   ECOM0  CCAPP0  CCAPN0   MAT0   TOG0  PWM0   ECCF0
       x    1      0       0       1      0     0      1
      ECOM0=1：允许 PCA 模块 0 的比较功能
      CCAPP0=0：关闭 PCA 模块 0 的上升沿捕获
      CCAPN0=0：关闭 PCA 模块 0 的下降沿捕获
      MAT0=1：允许 PCA 模块 0 的匹配功能
      TOG0=0：关闭 PCA 模块 0 的高速脉冲输出功能
      PWM0=0：关闭 PCA 模块 0 的脉宽调制输出功能
      ECCF0=1：允许 PCA 模块 0 的匹配/捕获中断
    ------------------------------END-----------------------------------*/
    CCAPM0 = 0x49;
    //写入比较值，注意：必须先写 CCAP0L，再写 CCAP0H
    CCAP0L = (u8)CLK_SOURCE_2MHZ_10MS;        //PCA 比较值寄存器赋初值
```

```
    CCAP0H = (u8)(CLK_SOURCE_2MHZ_10MS>>8);    //PCA 比较值寄存器赋初值

    counter_val = CLK_SOURCE_2MHZ_10MS * 2;    //下一次写入比较寄存器的数值
    CR = 1;                                    //启动 PCA 计数器阵列计数
}
```

4. PCA 中断服务函数

因为 PCA 配置的定时时间是 10ms，为了达到 200ms 的定时，我们定义一个变量“count”来记录 Timer0 的溢出次数，当“count”的值等于 20 的时候，表示达到 200ms，此时，翻转指示灯 D1 的状态，从而实现驱动 D1 以 200ms 间隔闪烁的目的。

代码清单：PCA 中断服务函数

```
/*********************************************************************************************
* 描  述 : PCA 中断服务程序
* 参  数 : 无
* 返回值 : 无
*********************************************************************************************/
void pca_isr (void) interrupt 7
{
  CCF0 = 0;                                //清 PCA 模块 0 匹配/捕获中断标志
  //写入比较值，注意：必须先写 CCAP0L，再写 CCAP0H
  CCAP0L = (u8)counter_val;                //PCA 比较值寄存器赋值
  CCAP0H = (u8)(counter_val >> 8);
  counter_val += CLK_SOURCE_2MHZ_10MS; //因为计数器的计数值不会清零，所以通过修改比较值达到定时的目的

  cnt++;                                   //每 10ms 执行一次 cnt 加一的操作
  if(cnt == 20)                            //cnt 加到 20 后，时间为 200ms
  {
    led_toggle(LED_1);                     //翻转用户指示灯 D1 状态
    cnt = 0;
  }
}
```

5. 主函数

主函数中初始化指示灯和 PCA 并开启总中断，PCA 计数启动后，即会按照配置的时间触发中断，代码清单如下。

代码清单：主函数

```
/*********************************************************************************************
* 描  述 : 主函数
* 参  数 : 无
* 返回值 : int 类型
*********************************************************************************************/
int main(void)
{
    P2M1 &= 0x3F;   P2M0 &= 0x3F; //设置 P2.6~P2.7 为准双向口（指示灯 D1 和 D2）
```

```
9.      pca_init();                 //初始化 PCA
10.     EA = 1;                     //使能总中断
11.
12.     while(1)
13.     {
14.     }
15. }
```

4.1.3. 硬件连接

本实验需要使用 LED 指示灯 D1，按照下图所示短接指示灯和按键的跳线帽。

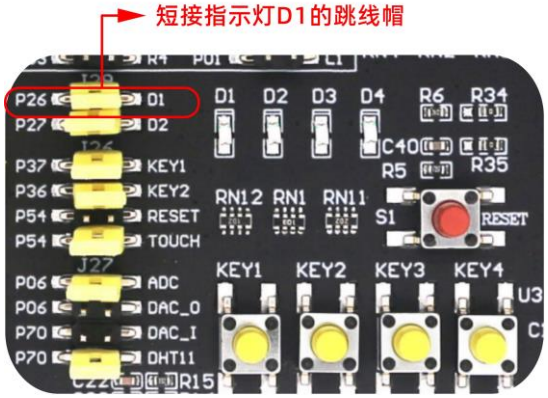

图 8：跳线帽短接

4.1.4. 实验步骤

1) 解压“…\第 3 部分：配套例程源码”目录下的压缩文件“实验 2-16-1：PCA 软件定时器实验”，将解压后得到的文件夹拷贝到合适的目录，如“D\STC8”（这样做的目的是为了防止中文路径或者工程存放的路径过深导致打开工程出现问题）。
2) 双击“…\pca_timer\project”目录下的工程文件“pca_timer.uvproj”。
3) 点击编译按钮编译工程，编译成功后生成的 HEX 文件“pca_timer.hex”位于工程的“…\pca_timer\Project\Object”目录下。
4) 打开 STC-ISP 软件下载程序，下载使用内部 IRC 时钟，IRC 频率选择：24MHz。
5) 程序运行后，可以观察到用户指示灯 D1 以 200mS 间隔闪烁。

4.2. 输出高速脉冲实验

✧ 注：本节的实验是在“实验 2-16-1：PCA 软件定时器实验”的基础上修改，本节对应的实验源码是：“实验 2-16-2：PCA 输出高速脉冲实验”。

4.2.1. 实验内容

使用 PCA 模块 0 输出 100KHz 的高速脉冲， PCA 配置如下：

- PCA 模块 0 功能引脚：P1.7。
- 系统时钟：24MHz。
- PCA 时钟源：系统时钟= 24Mhz。
- 中断：关闭计数器溢出中断，使能 PCA 模块 0 的匹配/捕获中断，中断优先级设置为 2

（较高级）。

- 脉冲频率 100KHz，对应的比较值为：24000000L/2/100000 = 120。

4.2.2. 代码编写

1. 引用头文件

因为在“main.c”文件中使用了“pca.c”文件中的函数，所以需要引用下面的头文件“pca.h”。

代码清单：引用头文件

```
1.  //引用 pca 的头文件
2.  #include  " pca.h"
```

2. PCA 初始化

本例中输出的脉冲频率较高，因此 PCA 时钟源配置为系统时钟 24MHz，PCA 模块 0 的功能引脚为 P1.7，即通过引脚 P1.7 输出高速脉冲，代码清单如下。

代码清单：PCA 初始化

```
1.  /*******************************************************************************
2.   * 描   述 : PCA 初始化
3.   * 参   数 : 无
4.   * 返回值 : 无
5.   *******************************************************************************/
6.  void pca_init(void)
7.  {
8.      CCON = 0x00;        //CF、CR、CCF1、CCF0 位均清零
9.      P_SW1 &= ~0x30;     //PCA 模块 0 功能引脚选择 P1.7
10.
11.     /*--------------------PCA 模式寄存器 CMOD 配置---------------------------
12.       位 7   位 6   位 5   位 4    位 3~位 1     位 0
13.      CIDL    x     x     x     CPS[2~0]    BCF
14.       0      x     x     x       100        0
15.
16.       CIDL=0：空闲模式下仍然计数
17.       CPS[2~0]=100：PCA 时钟源选择:系统时钟
18.       BCF=0：关闭 PCA 计数器溢出中断
19.     ------------------------------END-----------------------------------*/
20.     CMOD = 0x08;
21.
22.     CL = 0x00;          //PCA 计数器赋初值
23.     CH = 0x00;          //PCA 计数器赋初值
24.
25.     IP &= ~0x02;        //中断优先级配置为 2（较高优先级）
26.     IPH |= 0x02;
27.     /*------------------PCA 模块 0 模式控制寄存器 CCAPM0 配置------------------
28.      位 7    位 6     位 5     位 4      位 3     位 2    位 1     位 0
```

```
29.       x    ECOM0    CCAPP0   CCAPN0     MAT0    TOG0   PWM0    ECCF0
30.       x      1        0        0         1       1      0       1
31.
32.      ECOM0=1：运行 PCA 模块 0 的比较功能
33.      CCAPP0=0：关闭 PCA 模块 0 的上升沿捕获
34.      CCAPN0=0：关闭 PCA 模块 0 的下降沿捕获
35.      MAT0=1：允许 PCA 模块 0 的匹配功能
36.      TOG0=1：开启 PCA 模块 0 的高速脉冲输出功能
37.      PWM0=0：关闭 PCA 模块 0 的脉宽调制输出功能
38.      ECCF0=1：允许 PCA 模块 0 的匹配/捕获中断
39.     ----------------------------END----------------------------------*/
40.     CCAPM0 = 0x4D;
41.
42.     //PCA 模块 0 初始化部分
43.     counter_val = T100KHZ;            //counter_val 赋值
44.     //写入比较值，注意：必须先写 CCAP0L，再写 CCAP0H
45.     CCAP0L = (u8)counter_val;         //PCA 比较值寄存器赋初值
46.     CCAP0H = (u8)(counter_val >> 8); //PCA 比较值寄存器赋初值
47.     counter_val = T100KHZ * 2;        //下一次写入比较寄存器的数值
48.     CR = 1;                           //启动 PCA 计数器阵列计数
49. }
```

3. PCA 中断服务函数

PCA 工作于输出高速脉冲模式时，输出引脚的翻转是由硬件自动完成的，因此，中断服务函数中处理比较值即可，代码清单如下。

代码清单：PCA 中断服务函数

```
1.  /*********************************************************************
2.    * 描  述 : PCA 中断服务程序
3.    * 参  数 : 无
4.    * 返回值 : 无
5.    *********************************************************************/
6.  void pca_isr (void) interrupt 7
7.  {
8.     CCF0 = 0;                          //将 PCA 计数器阵列溢出标志位软件清零
9.     //写入比较值，注意：必须先写 CCAP0L，再写 CCAP0H
10.    CCAP0L = (u8)counter_val;          //PCA 比较值寄存器赋值
11.    CCAP0H = (u8)(counter_val >> 8);
12.    counter_val += T100KHZ;            //因为计数器的计数值不会清零，所以通过修改比较值达到定时的目的
13. }
```

4. 主函数

主函数中初始化指示灯和 PCA 并开启总中断，PCA 计数启动后，即会按照配置的时间触发中断，代码清单如下。

代码清单：主函数

```
/*************************************************************************
 * 描  述 : 主函数
 * 参  数 : 无
 * 返回值 : int 类型
 *************************************************************************/
int main(void)
{
  P1M1 &= 0x7F;   P1M0 &= 0x7F; //设置 P1.7 为准双向口
  pca_init();                   //PCA 初始化
  EA = 1;                       //使能总中断

  while(1)
  {
  }
}
```

4.2.3. 硬件连接

本实验通过 P1.7 输出 100KHz 高速脉冲，P1.7 引脚的位置如下图所示。

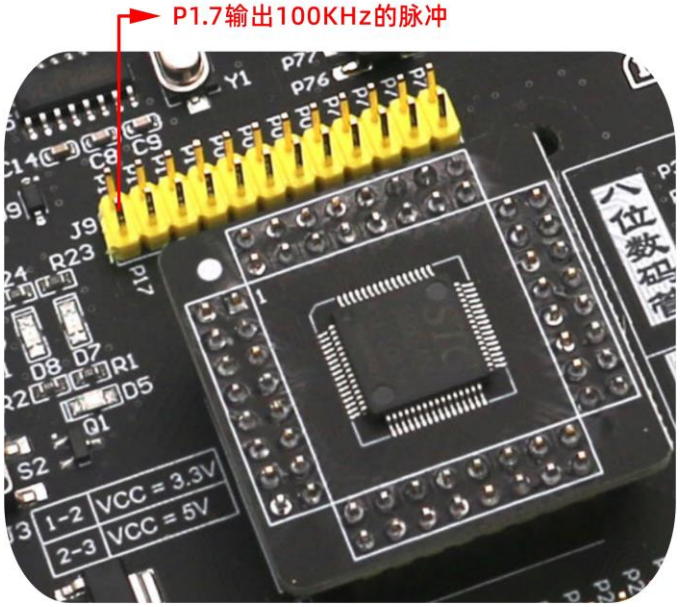

图 9：硬件连接

4.2.4. 实验步骤

1) 解压“…\第 3 部分：配套例程源码”目录下的压缩文件“实验 2-16-2：PCA 输出高速脉冲实验”，将解压后得到的文件夹拷贝到合适的目录，如“D\STC8”（这样做的目的是为了防止中文路径或者工程存放的路径过深导致打开工程出现问题）。
2) 双击“…\pca_pulse\project”目录下的工程文件“pca_pulse.uvproj”。
3) 点击编译按钮编译工程，编译成功后生成的 HEX 文件“pca_pulse.hex”位于工程的“…\pca_pulse\Project\Object”目录下。

4) 打开 STC-ISP 软件下载程序，下载使用内部 IRC 时钟，IRC 频率选择：24MHz。

5) 程序运行后，用示波器测量 P1.7，可以观察 P1.7 输出 100KHz 的方波。

4.3. 测量脉冲宽度实验

✧ 注：本节的实验是在“实验 2-10-1：Timer0 定时驱动 LED 闪烁”的基础上修改，本节对应的实验源码是：“实验 2-16-3：PCA 测量脉冲宽度实验”。

4.3.1. 实验内容

为了方便实验，本例使用 Timer0 定时 200ms，Timer0 中断服务函数中翻转 P2.6 的状态，即通过 P2.6 输出方波周期为 400ms 的方波。

使用 PCA 模块 0 捕获 P2.6 输出的方波， PCA 配置如下：

- PCA 模块 0 功能引脚：P1.7。
- 系统时钟：24MHz。
- PCA 时钟源：系统时钟= 24Mhz。
- 上升沿捕获。
- 中断：开启计数器溢出中断和 PCA 模块 0 的匹配/捕获中断，中断优先级设置为 2（较高级）。

每次测量后，通过串口输出测量得到的计数值和计数值对应的时间（单位毫秒）。

✧ 说明：关于 Timer0 部分的程序，读者可以阅读《第 2-10 讲：定时器和计数器》，这里不再赘述。

4.3.2. 代码编写

1. 新建一个名称为“pca.c”的文件及其头文件“pca.h”并保存到工程的“Source”文件夹，并将“pca.c”加入到 Keil 工程中的“SOURCE”组。
2. 引用头文件

因为在“main.c”文件中使用了“pca.c”文件中的函数，所以需要引用下面的头文件“pca.h”。

代码清单：引用头文件

```
3.  //引用 pca 的头文件
4.  #include  "pca.h"
```

3. PCA 初始化

本例配置 PCA 模块 0 功能引脚为 P1.7，测量由 Timer0 产生的周期为 400ms 的方波，PCA 模块 0 的时钟源为系统时钟的 1/12，上升沿捕获，代码清单如下。

代码清单：PCA 初始化和启动

```
1.  /*************************************************************************
2.  * 描  述 : PCA 初始化
3.  * 入  参 : 无
4.  * 返回值 : 无
5.  *************************************************************************/
6.  void pca_init(void)
7.  {
```

```
    CCON = 0x00;          //CF、CR、CCF1、CCF0 位均清零

    P_SW1 &= ~0x30;       //PCA 模块 0 功能引脚选择 P1.7
    /*--------------------PCA 模式寄存器 CMOD 配置---------------------------
     位 7   位 6   位 5   位 4    位 3~位 1     位 0
     CIDL    x     x     x     CPS[2~0]     BCF
      0      x     x     x       100         0

     CIDL=0：空闲模式下仍然计数
     CPS[2~0]=000：PCA 时钟源选择:系统时钟/12
     BCF=1：开启 PCA 计数器溢出中断
    ----------------------------END-----------------------------------*/
    CMOD = 0x01;
    CL = 0x00;            //PCA 计数器赋初值
    CH = 0x00;            //PCA 计数器赋初值
    IP &= ~0x02;          //中断优先级配置为 2（较高优先级）
    IPH |= 0x02;
    /*------------------PCA 模块 0 模式控制寄存器 CCAPM0 配置------------------
     位 7    位 6     位 5     位 4      位 3     位 2    位 1     位 0
      x    ECOM0   CCAPP0   CCAPN0    MAT0    TOG0   PWM0    ECCF0
      x      0       1        0         0       0      0       1

      ECOM0=0：关闭 PCA 模块 0 的比较功能
      CCAPP0=1：开启 PCA 模块 0 的上升沿捕获
      CCAPN0=0：关闭 PCA 模块 0 的下降沿捕获
      MAT0=0：关闭 PCA 模块 0 的匹配功能
      TOG0=0：关闭 PCA 模块 0 的高速脉冲输出功能
      PWM0=0：关闭 PCA 模块 0 的脉宽调制输出功能
      ECCF0=1：允许 PCA 模块 0 的匹配/捕获中断
    ----------------------------END-----------------------------------*/
    CCAPM0 = 0x21;
    CCAP0L = 0x00;              //PCA 比较值寄存器赋初值
    CCAP0H = 0x00;              //PCA 比较值寄存器赋初值
    CR = 1;                     //启动 PCA 计数器阵列计数
}
```

4. 测量脉宽

为了便于计算脉宽，我们定义了如下 3 个变量分别用于保存两次捕获时读取的 PCA 计数值和他们的差值。这里，我们定义的变量是 32 位的，这是为了处理 PCA 计数器的溢出。PCA 计数器是 16 位的，当其溢出后会从 0 开始计数，因此，溢出后将变量“ticks_current”的值加上 0x00010000，再减去上次捕获时保存的计数值即可得到正确的差值。

代码清单：PCA 初始化和启动

```
volatile u32 tick_diff;     //存储两次捕获之间的计数值的差值
```

```
volatile u32 ticks_last;   //记录上一次的捕获值
volatile u32 ticks_current;//记录本次的捕获值
```

P1.7 上出现上升沿或者 PCA 计数器溢出会触发 PCA 中断，PCA 中断服务程序中读取 PCA 计数值，并计算差值，之后将测量完成标志“flag”置位，代码清单如下。

代码清单：PCA 中断服务程序

```
/*********************************************************************
 * 描  述 : PCA 中断服务程序
 * 参  数 : 无
 * 返回值 : 无
 *********************************************************************/
void pca_isr (void) interrupt 7
{
    if(CF)                                  //PCA 计数器溢出
    {
        CF = 0;                             //清零 PCA 计数器溢出标志位
        ticks_current += 0x00010000;        //当前计数值变量增加一个溢出的计数值
    }
    if(CCF0)                                //是 PCA 模块 0 匹配/捕获中断标志
    {
        CCF0 = 0;                           //清零 PCA 模块 0 匹配/捕获中断标志位
        ((u8 *)&ticks_current)[3] = CCAP0L;     //16 位计数值保存到 ticks_current 的低 16 位
        ((u8 *)&ticks_current)[2] = CCAP0H;
        tick_diff = ticks_current - ticks_last;   //length 保存的即为捕获的脉冲宽度量化值
        ticks_last = ticks_current & 0x0000FFFF;  //备份上一次的捕获值，备份时去掉溢出部分
        ticks_current = 0;                  //当前计数值变量清零
        flag = 1;                           //测量完成标志置位
    }
}
```

5. 串口打印测量结果

测量完成后，通过串口打印出计数差值极其对应的时间，代码清单如下。

代码清单：打印测量结果

```
/*********************************************************************
 * 描  述 : 串口打印计数值和脉宽
 * 参  数 : 无
 * 返回值 : 无
 *********************************************************************/
void print_pw(void)
{
    float time_ms;
    if(flag)                                //测量完成，打印测量结果
    {
```

```
        printf("counter value: %lu \r\n",length);
        //PCA 时钟源使用的是：系统时钟（24MHz）/12 = 2MHz，脉宽对应的时间(秒)=length/2000000
        //这里打印的是毫秒，所以除以 2000
        time_ms = (float)length / 2000;
        printf("pulse width: %.2f ms\r\n",time_ms);
        flag = 0;
    }
}
```

6. 主函数

主函数中配置引脚并调用 Timer0 和 PCA 初始化函数，完成对 Timer0 和 PCA 的初始化和启动。之后在主循环里面调用测量结果打印函数，该函数中会检查测量完成标志，如测量完成标志置位，测打印测量结果，代码清单如下。

代码清单：主函数

```
/*************************************************************************
 * 描  述 : 主函数
 * 参  数 : 无
 * 返回值 : int 类型
 *************************************************************************/
int main(void)
{
    P1M1 &= 0x7F;   P1M0 &= 0x7F;     //设置 P1.7 为准双向口
    P2M1 &= 0x3F;   P2M0 &= 0x3F;     //设置 P2.6~P2.7 为准双向口
    P3M1 &= 0xFE;   P3M0 &= 0xFE;     //设置 P3.0 为准双向口（串口 1 的 RxD）
    P3M1 &= 0xFD;   P3M0 |= 0x02;     //设置 P3.1 为推挽输出（串口 1 的 TxD）

    uart1_init();                 //串口 1 初始化
    timer0_init();                //定时器 0 初始化
    pca_init();
    timer0_start();               //启动定时器 0
    EA = 1;                       //使能总中断

    while(1)
    {
        print_pw();     //查询测量是否完成，完成后串口打印测量结果
    }
}
```

4.3.3. 硬件连接

本实验需要用拔掉指示灯 D1 的跳线帽，然后用杜邦线将 P26 连接到 P17，如下图所示。

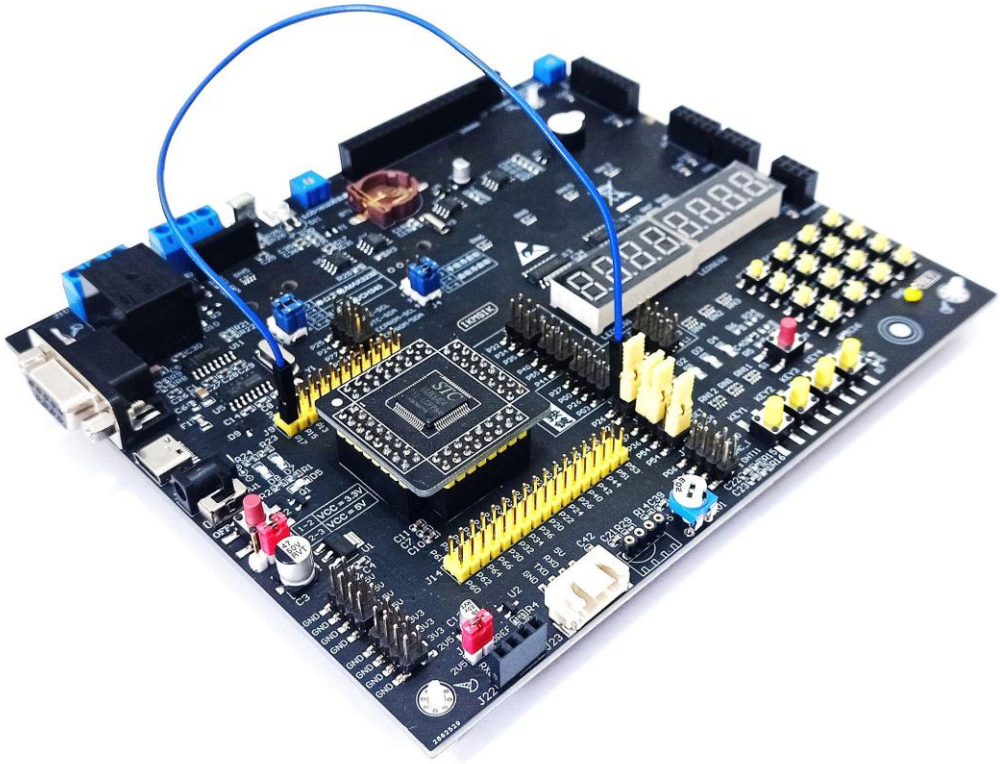

图 10：硬件连接

4.3.4. 实验步骤

1) 解压“…\第 3 部分：配套例程源码”目录下的压缩文件“实验 2-16-3：PCA 测量脉冲宽度实验”，将解压后得到的文件夹拷贝到合适的目录，如“D\STC8”（这样做的目的是为了防止中文路径或者工程存放的路径过深导致打开工程出现问题）。
2) 双击“…\pca_capture\project”目录下的工程文件“pca_capture.uvproj”。
3) 点击编译按钮编译工程，编译成功后生成的 HEX 文件“pca_capture.hex”位于工程的“…\pca_capture\Project\Object”目录下。
4) 打开 STC-ISP 软件下载程序，下载使用内部 IRC 时钟，IRC 频率选择：24MHz。
5) 程序运行后，可以在串口调试助手接收窗口观察到脉宽测量结果，如下图所示。

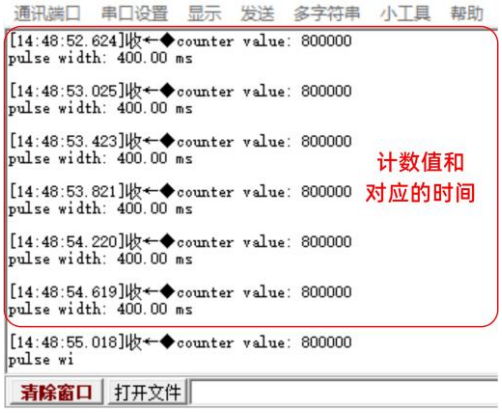

图 11：串口输出的测量的计数值和对应的时间

4.4. 输出 PWM 实验

✧ 注：本节的实验是在“实验 2-16-3：测量脉冲宽度实验”的基础上修改，本节对应的实验源码是：“实验 2-16-4：PCA 输出 PWM 实验”。

4.4.1. 实验内容

配置 PCA 模块 0 工作于 PWM 模式， PCA 配置如下：

- 系统时钟：24MHz。
- PCA 时钟源：系统时钟= 24Mhz。
- 功能引脚：P1.7。
- 中断：不开启中断。
- PWM 位数：6 位。
- PWM 频率：375KHz。
- PWM 占空比：25%。

4.4.2. 代码编写

1. 新建一个名称为“pca.c”的文件及其头文件“pca.h”并保存到工程的“Source”文件夹，并将“pca.c”加入到 Keil 工程中的“SOURCE”组。
2. 引用头文件

因为在“pca.c”文件中使用了“pca.c”文件中的函数，所以需要引用下面的头文件“pca.h”。

代码清单：引用头文件

```
1.  //引用 pca 的头文件
2.  #include   " pca.h"
```

3. PCA 初始化

本例中，我们配置 PCA 模块 0 工作于 PWM 模式，输出频率为 375KHz、占空比为 25%的波形，代码清单如下。

代码清单：PCA 初始化

```
1.   /*******************************************************************************
2.   * 描   述 : PCA 初始化
3.   * 入   参 : 无
4.   * 返回值 : 无
5.   *******************************************************************************/
6.  void pca_init(void)
7.  {
8.     CCON = 0x00;         //CF、CR、CCF1、CCF0 位均清零
9.     P_SW1 &= ~0x30;      //PCA 模块 0 功能引脚选择 P1.7
10.    /*--------------------PCA 模式寄存器 CMOD 配置---------------------------
11.     位 7  位 6  位 5  位 4   位 3~位 1    位 0
12.     CIDL  x    x    x     CPS[2~0]    BCF
13.      0    x    x    x      100         0
14.
```

```
15.      CIDL=0：空闲模式下仍然计数
16.      CPS[2~0]=100：PCA 时钟源选择:系统时钟
17.      BCF=0：关闭 PCA 计数器溢出中断
18.     ------------------------------END-----------------------------------*/
19.     CMOD = 0x08;
20.
21.     CL = 0x00;           //PCA 计数器赋初值
22.     CH = 0x00;           //PCA 计数器赋初值
23.     /*-----------------PCA 模块 0 模式控制寄存器 CCAPM0 配置-----------------
24.      位 7    位 6    位 5    位 4    位 3    位 2    位 1    位 0
25.       x   ECOM0  CCAPP0  CCAPN0   MAT0   TOG0  PWM0   ECCF0
26.         x     1      0      0       0      0     1      0
27.      ECOM0=1：允许 PCA 模块 0 的比较功能
28.      CCAPP0=0：关闭 PCA 模块 0 的上升沿捕获
29.      CCAPN0=0：关闭 PCA 模块 0 的下降沿捕获
30.      MAT0=0：关闭 PCA 模块 0 的匹配功能
31.      TOG0=0：关闭 PCA 模块 0 的高速脉冲输出功能
32.      PWM0=1：开启 PCA 模块 0 的脉宽调制输出功能
33.      ECCF0=0：关闭 PCA 模块 0 的匹配/捕获中断
34.     ------------------------------END-----------------------------------*/
35.     CCAPM0 = 0x42;
36.     //PCA 模块 0 工作于 6 位 PWM，由此可以计算出 PWM 频率=24Mhz/64 = 375KHz
37.     PCA_PWM0 &= 0xBF;
38.     PCA_PWM0 |= 0x80;
39.     //PWM 占空比为 50%
40.     //CCAP0L = 0x20;          //PCA 模块 0 的 PWM 比较值
41.     //CCAP0H = 0x20;          //PCA 模块 0 的 PWM 重装值
42.
43.     //PWM 占空比为 25%
44.     CCAP0L = 0x10;          //PCA 模块 0 的 PWM 比较值
45.     CCAP0H = 0x10;          //PCA 模块 0 的 PWM 重装值
46.     CR = 1;                 //启动 PCA 计数器阵列计数
47. }
```

4. 主函数

主函数中初始化 PCA 及其功能引脚，PCA 计数启动后，PWM 输出完全由硬件完成，无需软件干预，代码清单如下。

代码清单：主函数

```
1.  /*************************************************************************************
2.  * 描  述 : 主函数
3.  * 参  数 : 无
4.  * 返回值 : int 类型
5.  *************************************************************************************/
```

```
int main(void)
{
    P1M1 &= 0x7F;   P1M0 &= 0x7F;     //设置 P1.7 为准双向口
    pca_init();                       //初始化并启动 PCA 计数

    while(1)
    {
    }
}
```

4.4.3. 硬件连接

本实验通过 P1.7 输出频率为 375KHz、占空比为 25%的 PWM 波形，P1.7 引脚的位置如下图所示。

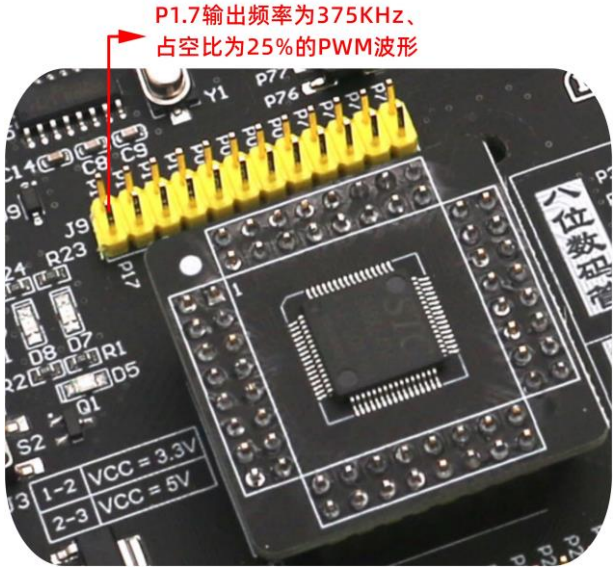

图 12：硬件连接

4.4.4. 实验步骤

1) 解压“…\第 3 部分：配套例程源码”目录下的压缩文件“实验 2-16-4：PCA 输出 PWM 实验”，将解压后得到的文件夹拷贝到合适的目录，如“D\STC8”（这样做的目的是为了防止中文路径或者工程存放的路径过深导致打开工程出现问题）。
2) 双击“…\pca_pwm\project”目录下的工程文件“pca_pwm.uvproj”。
3) 点击编译按钮编译工程，编译成功后生成的 HEX 文件“pca_pwm.hex”位于工程的“…\pca_pwm\Project\Object”目录下。
4) 打开 STC-ISP 软件下载程序，下载使用内部 IRC 时钟，IRC 频率选择：24MHz。
5) 程序运行后，用示波器测量 P1.7，可以观察 P1.7 输出频率为 375KHz、占空比为 25%的 PWM 波形。