

目录

目录	I
第一章 PWM模块	1
第一节 PWM模块介绍	1
第二节 PWM寄存器简介	2
2.1 PWME寄存器	2
2.2 PWMPOL寄存器	2
2.3 PWMCLK寄存器	3
2.4 PWMPRCLK寄存器	3
2.5 PWMCAE寄存器	4
2.6 PWMCTL寄存器	4
2.7 PWMSCLA寄存器	5
2.8 PWMCNTx寄存器	5
2.9 PWMPERx寄存器	6
2.10 PWMDTYx寄存器	7
第三节 PWM应用实例	8
3.1 周期和占空比计算举例	8
3.2 PWM初始化步骤总结	9
3.3 PWM应用实例	9
第二章 ECT模块	12
第一节 ECT模块介绍	12
1.1 简述	12
1.2 特征	12
1.3 运行模式	12
1.4 ECT的组成与工作模式	13
1.5 ECT的工作过程与设置	15
第二节 ECT寄存器简介	19
2.1 IC / OC选择寄存器(TIOS)	19
2.2 输出比较通道 7 屏蔽寄存器(OC7M)	20
2.3 输出比较通道 7 数据寄存器(OC7D)	20
2.4 定时器核心寄存器(TCNT)	21
2.5 计时器系统控制寄存器 1 (TSCR1)	21
2.6 计时器溢出绑定寄存器 1 (TTOV)	22
2.7 控制寄存器 (TCTL1-TCTL4)	23
2.8 计时器中断使能寄存器 (TIE)	24
2.9 计时器系统控制寄存器 2(TSCR2)	24
2.10 主定时器中断标志寄存器(TFLG1、TFLG2)	25
2.11 IC / OC寄存器(TC0-TC7)	26
2.12 脉冲累加器A控制寄存器(PACTL)	27
2.13 脉冲累加器A标志寄存器(PAFLG)	28
2.14 脉冲累加寄存器(PACN3、PACN2、PACN1、PACN0)	29

2.15	模数递减计数器控制寄存器(MCCTL).....	30
2.16	输入脉冲累加器控制寄存器(ICPAR)	31
2.17	输入覆盖控制寄存器(ICOVW).....	31
2.18	输入系统控制寄存器(ICSYS).....	32
2.19	脉冲累加器B控制寄存器(PBCTL)	33
2.20	脉冲累加器B标志寄存器(PBFLG)	34
2.21	脉冲累加器保持寄存器 (PA3H-PA0H)	34
2.22	模数递减计数器工作寄存器(MCCNT).....	35
2.23	IC保持寄存器 (TC0H-TC3H)	35
第三节	ECT应用实例	37
3.1	定时器编程步骤.....	37
3.2	输入捕捉IC:	37
3.3	通道 6 输出比较.....	38
3.4	通道 7 输出比较.....	39
3.5	模数递减计数器.....	40
第三章	SCI模块	42
第一节	SCI寄存器简介	42
1.1	波特率控制寄存器(SCIBDH、SCIBDL)	42
1.2	控制寄存器 1(SCICR1).....	43
1.3	控制寄存器 2(SCICR2).....	44
1.4	状态寄存器 1 (SCISR1)	45
1.5	状态寄存器 2(SCISR2)	47
1.6	数据寄存器(SCIDRH、SCIDRL)	47
第二节	SCI应用示例	48
第四章	SPI模块.....	53
第一节	SPI模块介绍.....	53
1.1	SPI的功能特点	53
1.2	SPI的组成与工作设置	54
第二节	SPI寄存器简介	60
2.1	SPI控制寄存器 1(SPICR1)	60
2.2	SPI控制寄存器 2(SPICR2)	62
2.3	SPI波特率选择寄存器	62
2.4	SPI状态寄存器	63
2.5	SPI数据寄存器.....	64
第三节	SPI应用实例	64
第五章	A/D转换模块.....	67
第一节	A/D模块介绍.....	67
1.1	A/D转换原理.....	67
1.2	A/D转换原理的应用前景	67
1.3	A/D转换模块.....	67
1.4	功能结构图.....	68
1.5	HCS12A/D特点	68
第二节	A/D寄存器简介	69
2.1	控制寄存器 2 (ATDCTL2)	69

2.2	控制寄存器 3 (ATDCTL3)	71
2.3	控制寄存器 4 (ATDCTL4)	71
2.4	控制寄存器 5 (ATDCTL5)	73
第三节	A/D应用示例	74
3.1	编程步骤	74
3.2	A/D程序示例—单通道查询	74
3.3	A/D程序示例—滤波	75
3.4	A/D程序示例—定时采样	76
第六章	EEPROM模块	79
第一节	EEPROM模块介绍	79
1.1	EEPROM功能	79
1.2	EEPROM结构	79
1.3	EEPROM特点	80
第二节	EEPROM寄存器简介	80
2.1	时钟分频寄存器ECLKDIV	80
2.2	配置寄存器ECNFG	81
2.3	保护寄存器EPROT	81
2.4	状态寄存器ESTAT	82
2.5	命令寄存器ECMD	84
第三节	EEPROM应用实例	85
3.1	EEPROM的写入操作	85
3.2	EEPROM的擦除操作	85
3.3	EEPROM示例程序	86
第七章	FLASH模块	91
第一节	FLASH模块介绍	91
1.1	FLASH功能	91
1.2	FLASH结构	91
1.3	FLASH特点	92
第二节	FLASH寄存器简介	92
2.1	时钟分频寄存器FCLKDIV	93
2.2	配置寄存器 FCNFG	94
2.3	安全寄存器 FSEC	94
2.4	保护寄存器 FPROT	95
2.5	状态寄存器 FSTAT 状态寄存器	96
2.6	命令寄存器 FCMD	98
第三节	FLASH应用实例	98
3.1	FLASH的写入操作	98
3.2	FLASH的擦除操作	99
3.3	FLASH的擦写操作注意事项	100
3.4	FLASH示例程序	100
第八章	CodeWarrior IDE 12 应用	103

第一章 PWM模块

第一节 PWM模块介绍

PWM 调制波有 8 个输出通道，每一个输出通道都可以独立的进行输出。每一个输出通道都有一个精确的计数器（计算脉冲的个数），一个周期控制寄存器和两个可供选择的时钟源。每一个 PWM 输出通道都能调制出占空比从 0—100% 变化的波形。

PWM 的主要特点有：

- 1、它有 8 个独立的输出通道，并且通过编程可控制其输出波形的周期。
- 2、每一个输出通道都有一个精确的计数器。
- 3、每一个通道的 PWM 输出使能都可以由编程来控制。
- 4、PWM 输出波形的翻转控制可以通过编程来实现。
- 5、周期和脉宽可以被双缓冲。当通道关闭或 PWM 计数器为 0 时，改变周期和脉宽才起作用。
- 6、8 字节或 16 字节的通道协议。
- 7、有 4 个时钟源可供选择（A、SA、B、SB），他们提供了一个宽范围的时钟频率。
- 8、通过编程可以实现希望的时钟周期。
- 9、具有遇到紧急情况关闭程序的功能。
- 10、每一个通道都可以通过编程实现左对齐输出还是居中对齐输出。

第二节 PWM寄存器简介

2.1 PWME寄存器

PWME 寄存器每一位如图 2 所示：

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	PWME7	PWME6	PWME5	PWME4	PWME3	PWME2	PWME1	PWME0
Write:								
Reset:	0	0	0	0	0	0	0	0

图 2 PWME 寄存器

每一个 PWM 的输出通道都有一个使能位 PWME_x。它是用来启动和关闭波形输出的。当任意的 PWME_x 位置 1，则相关的 PWM 输出通道就立刻可用。然而实际的 PWM 波形的输出还取决于时钟源。

此寄存器在任何时间都是可读、可写的，复位时全置 0。

用法： PWME7=1； 7 通道可对外输出波形。

PWME7=0； 7 通道不能对外输出波形。

注意：在通道使能后所输出的第一个波形可能是不规则的。当输出通道工作在串联模式时（PWMCTL 寄存器中的 CON_{xx} 位被设置），那么使能相应的 16 位 PWM 输出通道是由 PWME_x 的低电平位控制的（详情见 PWMCTL 寄存器）。

2.2 PWMPOL寄存器

PWMPOL 寄存器每一位如图 3 所示：

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	PPOL7	PPOL6	PPOL5	PPOL4	PPOL3	PPOL2	PPOL1	PPOL0
Write:								
Reset:	0	0	0	0	0	0	0	0

图 3 PWMPOL 寄存器

每一个 PWM 输出通道的波形都可以选择是在高电平时翻转，还是在低电平时翻转。此功能就是由 PWMPOL 寄存器实现的。

此寄存器在任何时间都是可读、可写的，复位时全置 0。

用法：PWMPOL0=1； 0 通道对外输出波形先是高电平然后再变为低电平。

PWMPOL0=0; 0 通道对外输出波形先是低电平然后再变为高电平。

2.3 PWMCLK寄存器

PWMCLK 寄存器每一位如图 4 所示：

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	PCLK7	PCLK6	PCLK5	PCLK4	PCLK3	PCLK2	PCLK1	PCLK0
Write:								
Reset:	0	0	0	0	0	0	0	0

图 4 PWMCLK 寄存器 PWM时钟选择寄存器

每一个 PWM 输出通道都有两个时钟可供选择（A、SA 或 B、SB）。0、1、4、5 通道可选用 A、SA 时钟，2、3、6、7 通道可选用 B、SB 通道。此寄存器在任何时间都是可读、可写的，复位时全置 0。应当注意的是，如果当一个 PWM 输出波形正在产生时，时钟改变，这时就会产生一个平头的或线形脉冲。

此寄存器在任何时间都是可读、可写的，复位时全置 0。

用法： PCLK1=1; 1 通道的时钟源设为 SA。

PCLK1=0; 1 通道的时钟源设为 A。

2.4 PWMPRCLK寄存器

PWMPRCLK 寄存器每一位如图 5 所示：

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	0	PCKB2	PCKB1	PCKB0	0	PCKA2	PCKA1	PCKA0
Write:								
Reset:	0	0	0	0	0	0	0	0


 = Reserved or unimplemented

图 5 PWMPRCLK 寄存器

PWMPRCLK 寄存器是单独用来给时钟源 A、B 进行预分频的。

PCKB2—PCKB0 是对 B 时钟源进行预分频。PCKA2 —PCKA0 是对 A 时钟源进行预分频。这 6 位可以随时被读、被写。复位时置 0。

其 A 时钟设置分频值如图 6 所示：

其 B 时钟设置分频值如图 7 所示：

PCKB2	PCKB1	PCKB0	Value of Clock B
0	0	0	E
0	0	1	E / 2
0	1	0	E / 4
0	1	1	E / 8
1	0	0	E / 16
1	0	1	E / 32
1	1	0	E / 64
1	1	1	E / 128

图 6 时钟 A 预分频选择值

PCKA2	PCKA1	PCKA0	Value of Clock A
0	0	0	E
0	0	1	E / 2
0	1	0	E / 4
0	1	1	E / 8
1	0	0	E / 16
1	0	1	E / 32
1	1	0	E / 64
1	1	1	E / 128

图 7 时钟 B 预分频选择值

2.5 PWMCAE寄存器

PWMCAE 寄存器每一位如图 8 所示：

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	CAE7	CAE6	CAE5	CAE4	CAE3	CAE2	CAE1	CAE0
Write:								
Reset:	0	0	0	0	0	0	0	0

图 8 PWMCAE 寄存器

PWMCAE 寄存器包含 8 个控制位来对每个 PWM 通道设置左对齐输出或居中对齐输出。如果 CAEx 置为 1，则为居中对齐输出。如果置为 0，则为左对齐输出。应当注意的是，只有输出通道被关闭后才能对其进行设置，即通道被激活后不能对其进行设置。

2.6 PWMCTL寄存器

PWMCTL 寄存器每一位如图 9 所示：

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	CON67	CON45	CON23	CON01	PSWAI	PFRZ	0	0
Write:								
Reset:	0	0	0	0	0	0	0	0

图 9 PWMCTL 寄存器

此寄存器的 2—7 位为可读、可写位。只有当相应的通道关闭后，才能改变这些控制字。

控制字介绍：

CON67=1；这时通道 6、7 就串联为同一个输出通道。此时只有 7 通道的控制字有用。例如：7 通道的 PWME 寄存器决定了他们的输出情况，7 通道的 PWMPOL 寄存器决定了他们是高电平翻转还是低电平翻转，7 通道的 PWMCLK 寄存器决定了他们两个的时钟源，7 通道的 PWMCAE 寄存器决定了他们是左对齐输出还是居中对齐输出等。

CON67=0；这时 6、7 通道分别作为独立输出通道对外输出。

CON45、CON23、CON01 的用法同 CON67 相似。设置此控制字的意义在于扩大了 PWM 对外输出脉冲的频率范围。

PSWAI=1；则 MCU 一旦处于等待状态，就会停止时钟的输入。这样就不会因时钟在空操作而费电。当它置为 0，则 MCU 就是处于等待状态，也允许时钟的输入。

2.7 PWMSCLA 寄存器

PWMSCLA 寄存器每一位如图 10 所示：

Address Offset:	\$0008							
	Bit 7	6	5	4	3	2	1	Bit 0
Read:	Bit 7	6	5	4	3	2	1	Bit 0
Write:	Bit 7	6	5	4	3	2	1	Bit 0
Reset:	0	0	0	0	0	0	0	0

图 10 PWMSCLA 寄存器

时钟 SA 是通过对 PWMSCLA 寄存器的设置来对 A 时钟进行分频而产生的。

其计算公式为：

$$\text{Clock SA} = \text{Clock A} / (2 * \text{PWMSCLA})$$

PWMSCLB 寄存器同 PWMSCLA 寄存器相似，时钟 SB 就是通过对 PWMSCLB 寄存器的设置来对 B 时钟进行分频而产生的。

其计算公式为：

$$\text{Clock SB} = \text{Clock B} / (2 * \text{PWMSCLB})$$

2.8 PWMCNTx 寄存器

PWMCNTx 寄存器共有 8 个，每一个通道都有一个。下面以 PWMCNT0 为例对 PWMCNTx 寄存器进行介绍。

PWMCNT0 寄存器每一位如图 11 所示：

Address	\$000C							PWMCNT0
Read:	Bit 7	6	5	4	3	2	1	Bit 0
Write:	0	0	0	0	0	0	0	0
Reset:	0	0	0	0	0	0	0	0

图 11 PWMCNT0 寄存器

计数器以所选时钟源的频率运行。计数器在任何时候都可以被读，而不影响计数，也不影响对 PWM 通道的操作。

任何值写入 PWMCNT0 寄存器都会导致计数器复位置 0，且其计数方向会被设置为向上计数，并且会立刻从缓冲器载入任务和周期值，并会根据翻转极性的设置来改变输出。当计数器达到计数值后，会自动清零。只有当通道使能后，计数器才开始计数。此寄存器随时都可以对其进行读、写操作。

2.9 PWMPERx寄存器

PWMPERx 寄存器共有 8 个，每一个通道都有一个这样的周期寄存器。这个寄存器的值就决定了相关 PWM 通道的周期。每一个通道的周期寄存器都是双缓冲的，因此如果当通道使能后，改变他们的值，将不会发生任何作用，除非当下列情况之一发生：

- *有效的周期结束。
- *对计数器进行写操作（计数器复位成 0）。
- *通道不可用（PWME_x=0）。

这样就会使 PWM 输出波形要么是新波形要么是旧波形，并不会在两者之间进行交替变换。如果通道不可用，那么对周期寄存器进行写操作，将会直接导致周期寄存器同缓冲器一起闭锁。图 12 所示的是 PWMPER0 寄存器。

Address	\$000C							PWMCNT0
Read:	Bit 7	6	5	4	3	2	1	Bit 0
Write:	0	0	0	0	0	0	0	0
Reset:	0	0	0	0	0	0	0	0

图 12 PWMPER0 寄存器。

周期的计算方法：

- 1) 当 CAEx=0 时，即进行左线性输出时：

$$\text{PWM}_x \text{ 周期} = \text{通道时钟周期} * \text{PWMPER}_x$$

- 2) 当 CAEx=1 时，即进行居中对齐输出时：

$$\text{PWM}_x \text{ 周期} = \text{通道时钟周期} * (2 * \text{PWMPER}_x)$$

2.10 PWMDTY_x寄存器

PWMDTY_x 寄存器也有 8 个，每一个通道都有一个这样的占空比常数寄存器。这个寄存器的值就决定了相关 PWM 通道输出波形的占空比。每一个通道的占空比寄存器都是双缓冲的，因此如果当通道被激活后，改变他们的值，将不会发生任何作用，除非当下列情况之一发生：

- *有效的周期结束。
- *对计数器进行写操作（计数器复位成 0）。
- *通道没有被激活（PWME_x=0）。

这样就会使 PWM 输出波形要么是新波形要么是旧波形，并不会在两者之间进行交替变换。如果通道没有被激活，那么对占空比常数寄存器进行写操作，将会直接导致周期寄存器同缓冲器一起闭锁。

当计数值与占空比常数 PWMDTY 相等时，则比较输出器有效，这时就会将触发器置位，然后 PWMCNT 继续计数，当计数值与周期常数 PWMPER 相等时，比较器输出有效，将触发器复位，同时也使 PWMCNT 复位，结束一个输出周期。

占空比的计算方法：

当 PPOL=0 时：

$$\text{占空比} = [(\text{PWMPER}_x - \text{PWMDTY}_x) / \text{PWMPER}_x] * 100\%$$

当 PPOL=1 时：

$$\text{占空比} = (\text{PWMDTY}_x / \text{PWMPER}_x) * 100\%$$

第三节 PWM应用实例

3.1 周期和占空比计算举例

例 1 设 CAEx=0; 即为左线形输出。此时, 设 E=10 MHz (100 ns), PPOLx=0, PWMPERx=4, PWMDTYx=1。

则 PWMx 输出频率=10 MHz/4=2.5 MHz

PWMx 输出周期=1/ (2.5 M) =400ns

PWMx 占空比= 3/4=25%

公式总结:

当为左线性输出时:

PWMx 输出频率=时钟频率/ PWMPERx

当 PPOLx=0 时

占空比=[(PWMPERx—PWMDTYx) / PWMPERx]*100%

当 PPOLx=1 时

占空比=[PWMDTYx/ PWMPERx]*100%

例 2 设 CAEx=1; 即为居中线性输出。此时, 设 E=10 MHz (100 ns), PPOLx=0, PWMPERx=4, PWMDTYx=1。

则 PWMx 输出频率=10 MHz/ (4*2) =1.25 MHz

PWMx 输出周期=1/ (1.25 M) =800ns

PWMx 占空比= 3/4=25%

公式总结:

当为居中线性输出时:

PWMx 输出频率=时钟频率/ (2* PWMPERx)

当 PPOLx=0 时

占空比=[(PWMPERx—PWMDTYx) / PWMPERx]*100%

当 PPOLx=1 时

$$\text{占空比} = [\text{PWMDTY}_x / \text{PWMPER}_x] * 100\%$$

应当注意的是，在对 PWMDTY_x 和 PWMPER_x 进行设置时，PWMPER_x 寄存器的值应当大于 PWMDTY_x 寄存器的值。

3.2 PWM初始化步骤总结

1、禁止 PWM Disable PWM

PWME=0;

2、选择时钟 Select clock (prescaler and scale) for the PWM

PWMPRCLK, PWMSCLA, PWMSCLB, PWMCLK

3、选择极性 Select polarity

PWMPOL

4、选择对齐模式 Select center or left aligned mode

PWMCAP

5、对占空比和周期编程 Program duty cycle and period

PWMDTY_x, PWMPER_x

6、使能 PWM 通道 Enable used PWM channels

PWME

3.3 PWM应用实例

例 1：输出占空比为 25% 的波形

(1) 实验设备：HCS12 编程器、开发板、示波器

(2) 软件程序设计：本例子是输出占空比为 25%，周期为 500Hz 的波形。JPb8 的 0 通道，开发板的地线和示波器相连。

(3) 源程序如下：

```
#include <hidef.h>          /* common defines and macros */
#include <mc9s12dp256.h>     /* derivative information */
#pragma LINK_INFO DERIVATIVE "mc9s12dp256b"
void main(void)
{
    PWME_PWME0=0;    //关闭 0 通道
```

```
PWMPRCLK=0X05; //对总线时钟进行预分频，总线时钟为 8M，分频后为 250K
PWMCLK_PCLK0=0; //设 A 为其时钟源
PWMSCLA=0X7D; //A 时钟为 2000Hz → SA时钟为1000HZ
PWMPOL_PPOL0=1; //上升沿翻转
PWMCAE_CAE0=0; //左对齐输出
PWMDTY0=0X01; //占空比为 25%的波形
PWMPER0=0X04; //输出为 500Hz 的波 → 输出为62.5KHZ的波
PWMCNT0=0X00; //0 通道计数器清 0
PWME_PWME0=1; //0 通道使能，0 通道为输出通道
}
```

例 2：步进电机的控制

(1) 实验设备：HCS12 编程器、开发板、步进电机、5804 芯片、稳压电源

(2) 功能实现：控制步进电机转动 90 度，按下中断后电机连续转动。

(3) 源程序如下：

```
#include <mc9s12dp256.h> /* derivative information */
#pragma LINK_INFO DERIVATIVE "mc9s12dp256b"
int i=0;
void main(void)
{
    DDRJ=0x00;
    PIEJ=0X03;
    EnableInterrupts; //中断设置
    PWME=0; //关闭所有通道
    PWMCTL_CON01=1; //对 PWM 控制寄存器进行设置,0,1 合为一个通道
    PWMCNT1=0X00; //0,1 通道计数器清 0
    PWMPOL_PPOL1=1; //上升沿翻转
    PWMCLK_PCLK1=0; //设 A 为其时钟源
    PWMPRCLK=0X00; //不对 A 时钟进行分频
    PWMCAE_CAE1=1; //中心对齐输出
    PWMPER1=0X20;
    PWMPER0=0x4e; //per 等于 40000，即 100 赫兹
    PWMDTY1=0X10;
    PWMDTY0=0X27; //dtty 等于 20000
    PWME_PWME1=1; //0, 1 通道使能
    for(i<=1854;)
    {
        if(PWMCNT0==0X4e)
            i++;
    } //步进电机转 90 度
    PWME=0; //0, 1 通道关闭
    while(1){;}
```

```
}  
#pragma CODE_SEG __NEAR_SEG NON_BANKED  
interrupt void man(void)  
{  
    if(PIFJ_PIFJ0==1)  
    {  
        PWMPER1=0X2a;  
        PWMPER0=0x68;//per 等于 26666，即 150 赫兹  
        PWMDTY1=0X15;  
        PWMDTY0=0X34;//dtty 等于 13333  
        PWME_PWME1=0xff;//0, 1 通道使能  
    }  
}
```

第二章 ECT模块

第一节 ECT模块介绍

1.1 简述

HC12 增强型捕捉计时器模块在 HC12 标准定时器的基础上增加了一些特点，用以扩展它的应用范围，特别是在汽车 ABS 方面。

基准计时器的核心仍然是一个 16 位的可编程计数器，其时钟源来自一个预分频器。该计时器可以被应用于多个方面，包括在对输入波形进行测量的同时产生一个输出波形。波形的脉宽可以在几微秒到数秒的范围内变化。

对计数寄存器或者输入捕捉/输出比较寄存器的访问可以发生在一个时钟周期内。对这些寄存器分别访问其高字节和低字节与将它们作为一个字进行访问所获得的结果肯定不同。

1.2 特征

- (1) 四个 IC 通道设置了 16 位保持寄存器，用于缓冲捕捉结果。
- (2) 四个 8 位脉冲累加器、四个与缓冲 IC 通道关联的 8 位保持寄存器。四个 8 位脉冲累加器通道可以级联，形成两个 16 位的脉冲累加器。
- (3) 具有 4 位定标器的 16 位递减模数计数器。
- (4) 四个可选的延迟计数器用于增强输入抗干扰能力。
- (5) 仅支持 IP 总线上的 16 位访问。

1.3 运行模式

停止：由于时钟停止，计时器和计数器均关闭。

冻结：计时器和计数器均保持运行，直到 TSCR(\$06)的 TSFRZ 位被置 1。

等待：计数器保持运行，直到 TSCR(\$06)的 TSWAI 位被置 1。

正常：计时器和计数器均保持运行，直到 TSCR(\$06)的 TEN 位和 MCCTL(\$26)的 MCEN 位被分别清 0。

ECT 增强模块的存储器图谱见表 55。每一个列出的寄存器地址是其地址的偏移量。总地址是 ECT 模块的基地址与偏移地址之和。

1.4 ECT的组成与工作模式

ECT 具有 8 个 IC / OC 通道、4 个 8 位或者 2 个 16 位的脉冲累加器(PAD 通道, 其 OC 部分与第 6 章的 TIM 模块相同, 但 IC 及 PAI 部分与 TIM 模块有一定区别, 其中 4 个 IC 通道与 TIM 模块相近, 当相关引脚出现预定动作时, 通过各自的捕捉寄存器 TCn 记录定时器的值; 另外四个 IC 通道, 除了捕捉寄存器 TCn, 还各有一个缓冲器 TCnH, 称为保持寄存器, 可以在不产生中断的前提下, 连续两次捕捉定时器的值。4 个 8 位的 PAI 通道 0-3 与 4 个缓冲 IC 通道 IC0-3 相关联, 并共享输入引脚 PORTT0-3。每一个脉冲累加器通道都拥有一个缓冲器 PACnH, 也称为保持寄存器, 可以在外部引脚出现预定动作时, 保存它的累加值。两对 8 位的脉冲累加器还可以通过级联形成 16 位的脉冲累加器 PACA、PACB。

16 位递减模数计数器(MDC)是 ECT 增设的, 它既是一个功能完善的定时器, 具有独立的可编程定标器、自动重装载和中断能力, 又可为 IC、PAI 寄存器向保持寄存器的传送提供定时控制信号。每当 MDC 回 0 时, 将在给定的时间段内控制 IC 和 PAI 寄存器的内容向各自的缓冲寄存器传输。当然, MDC 也可作为具有定时中断功能的独立时钟基准。

1.4.1 IC通道组

IC 通道组由四个标准的缓冲通道 IC0-IC3 和四个非缓冲通道 IC4-IC7 组成, 两部分的基本功能都是捕捉外部事件发生的时刻, 但是缓冲通道除了 IC / OC 寄存器 TCn 外, 还设有保持寄存器 TCnH, 此外还在入口设置了延迟计数器, 用来提高抗干扰能力。非缓冲通道没有保持寄存器, 入口也没有延迟计数器, 但每个通道入口设置了一个 2 输入端的多路器, 事件触发信号可以是来自本通道的输入引脚 PORTn, 也可以是来自其关联通道 PORT(n-4)的延迟计数器输出, 使用更加灵活。当延迟功能有效时, 输入引脚检测到一个有效的边沿后, 延迟计数器开始对 P 时钟(模块时钟)进行计数, 当到达设定的计数值后, 延迟计数器在其输出端有条件地产生一个脉冲, 这个条件就是延迟前后的引脚电平相反。这样可以避免对窄输入脉冲做出反应。延迟计数结束后, 计数器自动清除。输入信号两个有效边沿之间的持续时间必须大于设定的延迟时间。

在 ECT 中, 所有 IC 通道均设置了覆盖保护功能, 可以通过寄存器 IC0VW 设置是否允许某个通道用新的捕捉结果覆盖上一个结果。

对于缓冲的 IC 通道 PT0-PT3, 还具有锁存与队列两种工作方式。在锁存方式下, 每个有效的引脚事件只将自由定时器的值放入捕捉寄存器 TCn, 而 TCn 到保持寄存器 TCnH 的传送必须依赖递减模数计数器回 0 或者其他强制锁存命令才能实现, 这时 IC 的工作情形与第 6 章的 TIM 模块相似。在队列方式下(图 7-2), TCn 与 TCnH 形成了一个类似先进先出的队列, 每个捕捉结果从 TCn 进入, 然后随着下一个捕捉结果的到来移入 TCnH, 程序可以从 TCnH 取得结果, 然而这个队列是开放的, 即程序也可以直接从 TCn 取得捕捉结果。队列方式为 CPU 提供了充分的响应时间。

由于 PAC0-3 与 IC0-3 共享相同的引脚, 而且共享入口的逻辑, 因此在两种方式下, PAI 与 IC 都可以同时工作, 对同一引脚进行记录, 前者记录脉冲或者边沿的数量, 后者记录具体的时刻。

1.4.2 脉冲累加器

脉冲累加器由 4 个 8 位的通道 PAC0-PAC3 组成, 可以通过级联形成两个 16 位通道 PACA、PACB, 它可以统计输入引脚上出现的有效边沿的数量, 也可以统计有效电平出现的累计时间。各个通道的 8 位保持寄存器是与 4 个缓冲 IC 通道相关联的, 它们共享边沿检测与延迟电路。当 IC 工作在两种不同的队列方式时, PAC 保持寄存器也处于不同的工作状态, 在锁存方式下, PCnH 的加载依靠 MDC 计数器或者强制命令实现, 而在队列方式下, 则依靠 IC 通道的 TCnH 读命令。可见在 ECT 模块下, IC 与 PAI 的工作联系更加紧密。

此外, 脉冲累加器还有饱和记忆功能, 当 8 位的脉冲累加器计数超过 \$FFF 后记忆保持, 而不回滚到 0, 这时, 计数值 \$FFF 意味着计数已经达到或超过 255, 如不需要, 该功能可以关闭。这可以用来监视某个通道的计数值是否已经达到预定的目标值。

值得注意的是, PAC1、PAC0 级联后, 输入引脚为 PT0, 而 PAC3、PAC2 级联后, 输入引脚并不是 PT2, 而是 PT7, 这样可以与那些无 ECT 的 MCU 保持一致性。

1.4.3 模数递减计数器

16 位递减模数计数器(MDC)可以用作时钟基准，产生周期性的中断请求，也可用于将 IC 寄存器和脉冲累加器的值锁存到各自的保持寄存器中。锁存动作可以通过程序设定为周期性的或一次性的。MDC 的时钟频率可通过独立的定标器设定，内部设有定时常数寄存器，可以实现自动重装载，但 MDC 的常数寄存器与 MDC 计数器使用相同的地址，加载时通过特殊的时序实现。

每当 MDC 回 0 时，将在给定的时间段内控制贮和 PAI 寄存器的内容向各自的缓冲寄存器传输。反映了 MDC 在 IC、PAI 系统中的作用。

1.5 ECT的工作过程与设置

1.5.1 自由定时器、模数递减计数器与时钟频率设置

ECT 的自由定时器与 TIM 模块基本相同，惟一的区别是 TMSK2 中的 PR2-PR0 提供了 7 个预分频系数选项，包括 1、2、4、8、16、32、64、128，而原来的 TIM 只提供了前 6 个选项，这样定时器时钟频率可以更低，周期更长。

ECT 特有的模数递减计数器(MDC)逻辑结构，其核心为一个 16 位的递减计数器 MDC(\$37、\$36)，其外围配备了常数寄存器 MCCNT 和预分频器，分别为其提供定时常数和时钟信号。当寄存器 MCCTL 中的 MCEN=0 时，MDC 被复位成 \$FFFF，以避免在计数器启动的初期置位中断标志。将 MCEN 置 1，MDC 启动并从当前值开始对预分频器输出的时钟进行递减计数，分频系数为 1、4、8、16 可选，具体由 MCCTL 中的 MCPRI、MCPRO 确定(参看后面关于 MCCTL 的说明)。

MDC 有两种工作方式，由 MODMC 决定，当 MODMC=0 时，MDC 为单此计数方式，回到 \$0000 后停止，反之为循环工作方式。

无论在那种方式下，当 MDC 计数回 0 后，首先置位寄存器 MCFLG(\$27)的 MCZF 标志，若 MCCTL 中的中断允许位 MCZI=1，则向 CPU 发出中断请求，中断矢量为 \$FFCC、\$FFCD，向 MCFLG 的 MCZF 位写 1 将清除该标志。MDC 回 0 的另一个动作是向 IC、PAC 发出数据保持命令(参看 IC、PAC 部分)，这是 MDC 的重要任务之一。此外在连续方式即 MODMC=1 时，MDC 回 0 后还将自动从 MCCNT 加载定时常数，但在单次方式即 MODMC=0 时，MDC 回 0 后停止。

MCCNT 与 MDC 占用相同的 I/O 地址(\$37、\$36)，由控制位 RDMCL 决定每次读操作的具体访问对象。当 RDMCL=0/1 时，读操作分别返回 MDC 和 MCCNT 的当前值。对 MCCNT 的写操作要求在 MODMC=1 时进行，但 MDC 并不立即更新，必须等到计数器回 0 后重新加载，但向寄存器 MCCTL(\$26)中的 FLMC 位写 1 可以实现 MDC 的立即加载，同时还将复位预分频器。如果 MODMC: 0，对地址\$37、\$36 进行写入操作也将复位预分频器，并用写入值立即更新 MDC 计数器，然后开始一次递减计数，回到\$0000 后停止。可见在单次方式下可以直接指定定时常数，而在连续方式下则必须通过常数寄存器 MCCNT。

当寄存器 ICSYS(\$AB)中的 LATQ、BUFEN 均为 1 时，如果将\$0000 写入到 MCCNT 和模数计数器，输入捕捉和脉冲累加寄存器将被锁存。将\$0000 写入到 MCCNT 后，模数计数器将保持为 0，且不会将 MCFLG 中的 MCZF 标志置位。

1.5.2 IC通道组

IC 通道分为缓冲与非缓冲两类，缓冲类又具有锁存与队列两种工作方式，下面分别介绍其逻辑结构与寄存器的设置。

1. 缓冲的 IC 通道

缓冲的 IC 通道共有四个，即 PT0-3，其最大特点是在捕捉寄存器 TCn 的基础上，又增加了保持寄存器 TCnH，这样可以连续两次捕捉而不需要 CPU 干预。

单个缓冲 IC 通道主要由有效输入边沿检测、延迟滤波、捕捉与保持寄存器子系统等组成。当 TIOS 的 IOS0=0，即 PT0 设置为 IC 输入后，每当 PT0 引脚的逻辑电平发生变化时，内部边沿检测电路首先对其进行判别，判别的规则由寄存器 TCLA 中的 EDGOB、EDGOA 决定，判别的结果送到延迟计数器进行滤波处理。所谓滤波，即启动延迟计数器开始对 P 时钟(模块时钟)进行计数，当到达预先设定的计数值后，延迟计数器在其输出端有条件地产生一个脉冲，这个条件就是延迟前后的引脚电平相反。这样可以避免对窄输入脉冲做出反应。延迟时间由寄存器 DLYCT 中的 DLY1、DLY0 控制，四个选项分别为 0(旁路)、256、512、1024 个 P 时钟周期。延迟计数结束后，计数器自动清除，输入信号两个有效边沿之间的持续时间必须大于选定的延迟时间。

延迟后的有效信号送到捕捉逻辑，然后根据 ICOVW、ICSYS 等相关位的

设置决定是否捕捉 TCNT 的当前值以及是否处理保持寄存器 TCOH,此外还要决定是否设置中断请求标志——寄存器 TFLG1 中的 COF。一旦 COF 置位,如果 COI=1 且 CCR 中的 I=1,CPU 将会收到中断请求并予以响应,中断矢量为 \$FFEE、\$FFEF。通过寄存器 TFLG1 向 COF 写 1 将清除该标志位。对于锁存方式和队列方式,缓冲 IC 通道的捕捉条件和标志位的设置条件有所不同,下面将分别介绍。

(1)锁存方式。当设定为该方式时(LATQ=1),有效的引脚事件将主定时器值复制到寄存器 TC0。在下述情况下,TC0 还将被锁存到各自的保持寄存器中,这些情况包括模数计数器自然回 0、向模数计数器直接写 \$0000、向控制寄存器 MCCTL 中的 ICLAT 位直接写 1 等。若寄存器 ICOVW 中的 NOVW_x 位被清 0,而这时出现新的捕捉事件,IC 寄存器的内容将被新值覆盖。如果发生锁存操作,那么保持寄存器的内容也会被覆盖。若寄存器 ICOVW 中的 NOVW_x 位被置 1,那么捕捉寄存器和保持寄存器必须处于 清空状态才允许写入,否则不能写入。这样可以避免读取或者转移到保持寄存器前,TC0 被覆盖。

(2)队列方式(参看图 7-2)。当设定为该方式时(LATQ=0),有效的引脚事件将主定时器值复制到 IC 寄存器。寄存器 ICOVW 中的 NOVW_x 位被清 0,而这时出现新的捕捉事件,IC 寄存器的内容将被转移到保持寄存器。空出来的 IC 寄存器存放新的捕获值。若寄存器 ICOVW 中的 NOVW_x 位被置 1,那么捕捉寄存器和保持寄存器必须处于清空状态才允许写入,否则不能写入。在该方式下读保持寄存器 TC0H 会将相关的脉冲累加器值转移到对应的保持寄存器。在上述两种方式下,C3F-C0F 中断标志的建立条件如图 7-6 所示。

2. 非缓冲 IC 通道

这类通道与 TIM 模块相似,但是增加了覆盖控制位、入口处的延迟计数器及引入了双引脚控制(例如 PT0)。当 PT4 引脚出现有效事件时,主定时器的值可能被记录到 IC 寄存器 TC4,但是受覆盖控制寄存器 ICOVW 的影响,具体如下:

(1)如果输入覆盖控制寄存器 ICOVW 中的对应位 NOVW_x 为 0,当发生新的捕捉事件时,IC 寄存器将用新的时间值覆盖上次捕捉结果。

(2)如果上述控制位为 1,那么 IC 寄存器将不被覆盖,除非此时它处于清空状态。这可以阻止捕捉值在读取之前被覆盖掉。

1.5.3 脉冲累加器

ECT 的四个 8 位的脉冲累加器可以独立使用，也可级联使用，这样可以满足不同的分辨率和通道数量要求。具体取决于 PAEN、PBEN 以及 PAEN3-PAEN0 的设置。当 PAEN=PBEN=0 时，PAC3-PAC0 独立工作，当 PAENn=1 时，对应的 PACn 通道使能。当 PAEN=1 时，PAC3、PAC2 级联成 16 位通道 A，这时 PAEN3、PAEN2 无效，当 PBEN=1 时，PAC1、PAC0 级联成 16 位通道 B，这时 PAEN1、PAEN0 无效。级联后形成的通道 A 与 TIM 模块兼容，而通道 B 则功能相对简单。下面分别介绍级联前的四个 8 位通道 PAC3-0、级联后的 16 位通道 A 和通道 B 各自的逻辑结构与设置。

1、8 位通道 PAC3-PAC0

PAC3-PAC0 与 IC 联系十分密切，二者共享引脚、边沿检测和延迟计数电路，因此使用 PAC 通道时对应引脚必须通过寄存器 TIOS 设置为 IC 方式。

当 TIOS 中的 IOS0、IOS1 等于 0 时，PT0 / PT1 引脚信号进入边沿检测电路，有效边沿将被送到延迟计数部分进行窄脉冲消除，然后分别送到 PACN0 / PACN1，如果寄存器 ICPACR 中的 PAEN1、PAEN0 为 1，则 PACN1、PACN0 对输入进行计数。各通道的有效边沿可以通过寄存器 TCL4 单独设置，可选项包括切断、上升沿、下降沿和双向边沿，延迟时间则通过寄存器 DLYCT 设置，可选项包括 0、256、512、1024 个 P 时钟周期，但所有通道共用一个延迟时间，不能分别设置。PACN3 具有溢出中断功能，但 PACN2 没有，当 PACN3 从 \$FF 回滚到 \$00 时，寄存器 PBFLG 中的 PBOVF 标志置 1，如果寄存器 PBCTL 中的 PBOVI=1，同时 CPU 全局中断屏蔽位 I=1，将向 CPU 申请中断，中断矢量为 \$FFCA、\$FFCB。通过程序向 PBOVF 写 1 将清除该标志。

PAC3-PAC0 通道设置了饱和记忆功能，当 ICSYS 中的 PACMX=1 时，该功能启动，这时如果某个通道计数器 PACNx 计数达到 \$FF，那么 PACNx 将停止对后续事件的计数，而保持在 \$FF，因此发现某个 PACNx=\$FF 说明其计数结果大于或者等于 255。

当 ICSYS 中的 BUFEN=1 时，保持寄存器 PAIH、PAOH 有效，用来保存 PACN1、PACN0 的累计结果，传送控制信号由 MCCTL 中的 ICLAT 或递减计数器 MDC 发出，但是与 ICSYS 中的 LATQ 有关。当 BUFEN=0 时，保持寄存器无效。

(1)脉冲累加器的锁存方式;当 LATQ=1(锁存方式)时,模数递减计数器回 0、将\$0000 写入模数计数器或者强制锁存控制位 ICLAT 被置 1 三者之一发生时,脉冲累加器的值将被转移到与它对应的保持寄存器,其自身同时清 0。

(2)脉冲累加器的队列方式。当 LATQ=0(队列方式)时,只在读取对应的 IC 通道的保持寄存器(TCIH、TCOH)时,才会产生传送动作,其他的传送控制信号无效。

此外, PAC 设置了覆盖限制功能,当保持寄存器为空时,将允许进行结果传送,但当保持寄存器非空时则禁止覆盖。覆盖控制通过寄存器 ICSYS 实现。

对于 PAC2、PAC3,其逻辑结构与设置与 PAC0、PAC1 相似,只是涉及的某些寄存器不同,有关的设置这里不再赘述。

2、16 位通道 A

当寄存器 PACTL 中的 PAEN=1 时,通道 PAC3、PAC2 不复存在, PACN3、PACN2 级联成通道 A(PAI),但为了与 TIM 保持一致,PAI 仍然使用引脚 PT7,而不使用引脚 PT2,当然也不能使用 PT2 的边沿检测和延迟电路。该部分的工作过程与 TIM 的 PAI 完全相同。

3、16 位通道 B

当寄存器 PBCTL 中的 PBEN=1 时,通道 PAC1、PAC0 不复存在, PACN1、PACN0 级联成通道 B,它使用引脚 PT0,因此仍然可以使用 FT0 的边沿检测和延迟电路,但没有门控时间累加方式,此外该通道只有溢出中断,这与通道 A 均不同。通道 B 只有事件计数方式,其他与通道 A 类似,读者可参看第 6 章的有关部分,这里不再赘述。

第二节 ECT寄存器简介

2.1 IC / OC选择寄存器(TIOS)

寄存器偏移量: \$0000

Bit7	6	5	4	3	2	1	Bit0
IOS7	IOS6	IOS5	IOS4	IOS3	IOS2	IOS1	IOS0
复位后: 0	0	0	0	0	0	0	0

可在任何时候读或写。

TIOS 寄存器用于指定各个通道的功能,即工作于 IC 还是 OC 方式。当某个

位 $IOS_n=0$ 时, 对应的通道 n 为输入捕捉(IC)通道, 否则当 $IOS_n=1$ 时, 通道 n 为输出比较(OC)通道。其中的各位可以在任何时候写入或读出。上电后该寄存器默认为\$00, TSCR 中的 TEN 默认也为 0, 这时所有通道处于通用 I / O 方式, 将 TEN 置位后各个通道进入 IC 方式, 要将某些通道设置成 OC 方式, 必须对 TIOS 进行设置, 即将有关位置 1。设置成 OC 的通道其引脚具有降功率驱动功能, 设置成 IC 的通道具有内部上拉功能, 但上电后均处于关闭状态, 可以根据需要启用。

2.2 输出比较通道 7 屏蔽寄存器(OC7M)

寄存器偏移量: \$0002

Bit7	6	5	4	3	2	1	Bit0
OC7M7	OC7M6	OC7M5	OC7M4	OC7M3	OC7M2	OC7M1	OC7M0
复位后: 0	0	0	0	0	0	0	0

可在任何时候读或写。

前面已经说明, OC7 具有特殊地位, 它匹配时可以直接改变其他 7 个输出引脚的状态, 并覆盖各个引脚原来的匹配动作结果, 寄存器 OC7M 决定哪些通道将处于 OC7 的管理之下。OC7M 中的各位与 PORTT 口寄存器的各位一一对应。当通过 TIOS 将某个通道设定为输出比较时, 将 OC7M 中的相应位置 1, 对应的引脚就是输出状态, 与 DDR 中的对应位的状态无关。但 OC7M_n 并不改变 DDR 相应位的状态。

OC7M 具有更高的优先级, 它优于通过 TCTL1 和 TCTL2 寄存器中的 OM_n 和 OL_n 设定的引脚动作, 若 OC7M 中某个位置 1, 就会阻止相应引脚上由 OM 和 OL 设定的动作。

2.3 输出比较通道 7 数据寄存器(OC7D)

寄存器偏移量: \$0003

Bit7	6	5	4	3	2	1	Bit0
OC7D7	OC7D6	OC7D5	OC7D4	OC7D3	OC7D2	OC7D1	OC7D0
复位后: 0	0	0	0	0	0	0	0

可在任何时候读或写。

OC7M 对于其他 OC 输出引脚的管理限于将某个二进制值送到对应引脚, 这个值保存在寄存器 OC7D 中的对应位中。当 OC7 匹配成功后, 若某个 OC7M_n=1, 则内部逻辑将 OC7D_n 送到对应引脚。

OC7D 中的各位与 PORTT 口寄存器的各位一一对应。当通道 7 比较成功时，如果 OC7M 中的某个位为 1，OC7D 中的对应位将被输出到 PORTT 的对应引脚。

当 OC7M 中的某个位为 1 时，通道 7 匹配成功的动作如果与通道 6-0 的动作发生在同一个周期，前者将覆盖后者。因此各个通道的动作将依赖于 OC7D 中各个位的设置。

2.4 定时器核心寄存器(TCNT)

寄存器偏移量：\$0004-\$0005

Bit15	Bit14	Bit13	Bit12	Bit11	Bit10	Bit9	Bit8
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0

复位后：0 0 0 0 0 0 0 0

TCNT 是递增计数器，它不停地对内部时钟信号计数、程序可随时读取，但在普通模式下禁止写入。TCNT 应按字访问，分别访问高、低字节可能得到错误的结果。

在特殊模式下，TCNT 可写，但因为写操作与预分频器时钟不同步，TCNT 寄存器写入后，其第一个周期可能是一个不同的值。

2.5 计时器系统控制寄存器 1 (TSCR1)

寄存器偏移量：\$0006

Bit7	6	5	4	3	2	1	Bit0
TEN	TSWAI	TSFRZ	TFPCA				

复位后：0 0 0 0 0 0 0 0

可在任何时候读或写。

TSCR1 寄存器是定时器模块的总开关，它决定模块是否启动以及在中断等待、BDM 方式下的行为，还包括标志的管理方式。其各位的意义如下：

TEN：定时器使能位，此外它还控制定时器的时钟信号源。要使用定时器模块的 IC / OC 功能，必须将 TEN 置位。如果因为某种原因定时器没有使能，脉冲累加器也将得不到 ECLK / 64 时钟，因为 ECLK / 64 是由定时器的分频器产生的，这种情况下，脉冲累加器将不能进行引脚电平持续时间的累加。

0：主计时器、包括计数器均被禁止，有利于降低功耗。

1：定时器使能，正常工作。

TSWAI：等待模式下计时器关闭控制位。

注意定时器中断不能用于使 MCU 退出等待模式。

0: 在中断等待模式下允许 MCU 继续运行。

1: 当 MCU 进入中断等待模式时, 禁止计时器。

TSFRZ: 在冻结模式下计时器和计数器停止位。

0: 在冻结模式下允许计时器和计数器继续运行。

1: 在冻结模式下禁止计时器和计数器, 用于仿真调试。

注意: **TSFRZ** 不能停止脉冲累加。

TFFCA: 定时器标志快速清除选择位。

0: 定时器标志普通清除方式。

1: 对于 **TFLGI(\$0E)** 中的各位, 读输入捕捉寄存器或者写输出比较寄存器会自动清除相应的标志位 **CnF**。

对于 **TFLG2(\$0F)** 中的各位, 任何对 **TCNT** 寄存器(\$04、\$05) 的访问均会清除 **TOF** 标志; 任何对 **PACN3** 和 **PACN2** 寄存器 (\$22,\$23) 的访问都会清楚清除 **PAFLG** 寄存器(\$21) 中的 **PAOVF** 和 **PAIF** 位。任何对 **PACN1** 和 **PACN0** 寄存器 (\$24,\$25) 的访问都会清除 **PBFLG** 寄存器(\$21) 中的 **PBOVF** 位。这种方式的好处是削减了另外清除标志位的软件开销。此外, 必须特别注意避免对标志位的意外清除。

2.6 计时器溢出绑定寄存器 1 (TTOV)

寄存器偏移量: \$0007

Bit7	6	5	4	3	2	1	Bit0
TOV7	TOV6	TOV5	TOV4	TOV3	TOV2	TOV1	TOV0
复位后: 0	0	0	0	0	0	0	0

可在任何时候读或写。

TOVx: 溢出绑定管脚标志位。

TOVx 在输出比较时, 如果发现溢出, 则绑定该管脚。这一功能仅在输出比较模式下有效, 它的优先级高出强制输出比较, 而低于通道 7 的输出比较。

0: 输出比较的管脚发生溢出时绑定功能不发生作用。

1: 输出比较的管脚发生溢出时绑定功能使能。

2.7 控制寄存器 (TCTL1-TCTL4)

TCTL1 寄存器偏移量: \$0008

Bit7	6	5	4	3	2	1	Bit0
OM7	OL7	OM6	OL6	OM5	OL5	OM4	OL4
复位后: 0	0	0	0	0	0	0	0

TCTL2 寄存器偏移量: \$0009

Bit7	6	5	4	3	2	1	Bit0
OM3	OL3	OM2	OL2	OM1	OL1	OM0	OL0
复位后: 0	0	0	0	0	0	0	0

TCTL3 寄存器偏移量: \$000A

Bit7	6	5	4	3	2	1	Bit0
EDG7B	EDG7A	EDG6B	EDG6A	EDG5B	EDG5A	EDG4B	EDG4A
复位后: 0	0	0	0	0	0	0	0

TCTL4 寄存器偏移量: \$000B

Bit7	6	5	4	3	2	1	Bit0
EDG3B	EDG3A	EDG2B	EDG2A	EDG1B	EDG1A	EDG0B	EDG0A
复位后: 0	0	0	0	0	0	0	0

可在任何时候读或写。

TCTL1-TCTL4 分为两组, 分别对 IC 和 OC 电路进行设定, 每组 16 个二进制位, 每两个二进制位管理一个通道。其中 TCTL1、TCTL2 设定各个 OC 通道匹配时的动作, 包括切断 OC 与输出引脚的联系, 而 TCTL3、TCTL4 设定 IC 响应引脚的何种动作, 包括禁止 IC 的响应。各个控制位的作用如下:

OMn、OLn 分别设定输出方式和输出电平, 这 8 对控制位(OM7、OL7-OM0、OL0)编码后用于指定通道比较成功后的输出动作(参看表 7-3)。如果每对当中至少有一个为 1, 对应引脚就固定为相应通道的输出, 而与 DDRT 中的对应位无关。当二者同时为 0 时, OC 与输出引脚断开。

表 7-3 输出比较动作设置

OMn	OLn	动作
0	0	定时器与输出引脚断开
0	1	OCn 输出翻转
1	0	OCn 输出清 0
1	1	OCn 输出置 1

EDGnB、EDGnA 输入捕捉边沿控制位, 这 8 对控制位(EDG7B、EDG7A—EDG0B、EDG0A)对输入捕捉的边沿检测电路进行设置 (参看表 7-4)。当二者同时为 0 时, IC 与输入引脚断开。

表 7-4 输入捕捉边沿检测电路设置

EDGnB	EDGnA	边沿检测器电路设置
0	0	捕捉禁止
0	1	仅捕捉上升沿
1	0	仅捕捉下降沿
1	1	上升、下降沿均捕捉

注意：为了使 OMn、OLn 指定的引脚动作有效，OC7M 中的对应位必须清 0。若要使用 16 位脉冲累加器 A 和 B，并使它们分别独立于 IC / OC7 和 IC / OC0，必须设置对应的 IOSn：1、OMn=0、OLn=0，同时寄存器 OC7M 中的 C7M7、OC7M0 位必须清 0。

2.8 计时器中断使能寄存器 (TIE)

寄存器偏移量：\$000C

Bit7	6	5	4	3	2	1	Bit0
C7I	C6I	C6I	C4I	C3I	C2I	C1I	C0I
复位后：0	0	0	0	0	0	0	0

可在任何时候读或写。

TIE 寄存器中的位与状态寄存器 TFLG1 中的标志位相对应。如果将 TIE 中的某位清 0，相应的标志位就不能引起硬件中断。如果被置 1，相应的标志位就可以引起中断。

C7I-C0I:输入捕捉/输出比较 “x”中断使能。

2.9 计时器系统控制寄存器 2(TSCR2)

寄存器偏移量：\$000D

Bit7	6	5	4	3	2	1	Bit0
TOI	0	0	0	TCRE	PR2	PR1	PR0
复位后：0	0	0	0	0	0	0	0

可在任何时候读或写。

TOI:计时器溢出中断使能。

0：中断被禁止。

1：当 TOF 标志被置位时发出硬件中断请求。

TCRE：时钟计数器复位使能。

该位在通道 7 成功输出比较之后允许时钟计数器复位。该操作模式类似于递

增型计数器。

0: 计数器复位禁止, 计数器自由计数。

1: 通道 7 成功输出比较后计数器将被复位。

如果 $TC7=\$0000$ 并且 $TCRE=1$, $TCNT$ 将继续保持 $\$0000$ 。

如果 $TC7=\$FFFF$ 并且 $TCRE=1$, 当 $TCNT$ 从 $\$FFFF$ 到 $\$0000$ 之间被复位后 TOF 将永远不被置位。

$PR2, PR1, PR0$: 计数器预分频选择。

这三位所决定的分频因子如下表所示。

表 7-5 分频因子选择

PR2	PR1	PR0	Prescale Factor
0	0	0	1
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

新设定的分频因子不会立即起作用, 直到下一个触发沿到来那里所有预分频计数器值均为零。

2.10 主定时器中断标志寄存器(TFLG1、TFLG2)

TFLG1 寄存器偏移量: $\$000E$

Bit7	6	5	4	3	2	1	Bit0
C7F	C6F	C5F	C4F	C3F	C2F	C1F	C0F
复位后: 0	0	0	0	0	0	0	0

TFLG2 寄存器偏移量: $\$000F$

Bit7	6	5	4	3	2	1	Bit0
TOF	0	0	0	0	0	0	0
复位后: 0	0	0	0	0	0	0	0

所示的 TFLG1、TFLG2 为中断标志寄存器, 其中 TFLG1 对应 8 个 IC / OC 通道, 当某 $CnF=1$ 时说明对应的 IC / OC 通道有动作。TFLG2 只有一个标志位 TOF, 作为核心计数器的中断请求标志。当 $TOF=1$ 时说明核心计数器溢出。要清除某个标志位, 只需向该位写 1, 向某位写 0 不影响该位的状态。当 TSCR 中

的 TFFCA 位置位时，读 IC 通道或写 OC 通道 (\$10-\$1F)将自动清除该通道标志 CnF，对 TCNT 的任何访问将自动清除 TFLG2。

CnF: IC / OC 通道中断请求标志。

0: 上次清除标志以来，IC / OC 通道没有有效动作。

1: IC / OC 通道已经出现动作。将寄存器 ICSYS(\$2B)中的 TFMOD 位和 ICOVW 寄存器(\$2A)联合使用，可以使定时器在两次捕捉后才产生中断，而不是每次捕捉均产生动作。两次捕捉结果分别在捕捉和保持寄存器里面。

TOF: 定时器溢出标志，当 16 位自由定时器从 \$FFFF 回滚到 \$0000 时，该位置位。将 \$80 写入到 TFLG2 将自动清除该位。详见前面关于 TMSK2 中 TCRE 控制位的解释。

2.11 IC / OC寄存器(TC0-TC7)

每个 IC 或 OC 通道都设置有一个 16 位的寄存器，对于 IC(输入捕捉)通道，当通道的边沿探测器检测到由 EDGnA、EDGnB 指定的条件时，将自由定时器的值捕捉到寄存器 TCn，随后程序可以读取和处理；对于 OC(输出比较)通道，程序将预定的时刻写入到 TCn，当自由定时器的值与其相等时，触发由 OMn、OLn 所指定的输出动作。定时器模块共有 TC7-TC0 等 8 个 16 位 IC / OC 寄存器。

对于设定为 IC 的通道，相应通道写操作无意义。上述所有寄存器复位后为 \$0000。

TC0 寄存器偏移量: \$0010-\$0011

Bit15	Bit14	Bit13	Bit12	Bit11	Bit10	Bit9	Bit8
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0

复位后: 0 0 0 0 0 0 0 0

TC1 寄存器偏移量: \$0012-\$0013

Bit15	Bit14	Bit13	Bit12	Bit11	Bit10	Bit9	Bit8
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0

复位后: 0 0 0 0 0 0 0 0

TC2 寄存器偏移量: \$0014-\$0015

Bit15	Bit14	Bit13	Bit12	Bit11	Bit10	Bit9	Bit8
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0

复位后: 0 0 0 0 0 0 0 0

TC3 寄存器偏移量: \$0016-\$0017

Bit15	Bit14	Bit13	Bit12	Bit11	Bit10	Bit9	Bit8
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0

复位后: 0 0 0 0 0 0 0 0

TC4 寄存器偏移量: \$0018-\$0019

Bit15	Bit14	Bit13	Bit12	Bit11	Bit10	Bit9	Bit8
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0

复位后: 0 0 0 0 0 0 0 0

TC5 寄存器偏移量: \$001A-\$001B

Bit15	Bit14	Bit13	Bit12	Bit11	Bit10	Bit9	Bit8
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0

复位后: 0 0 0 0 0 0 0 0

TC6 寄存器偏移量: \$001C-\$001D

Bit15	Bit14	Bit13	Bit12	Bit11	Bit10	Bit9	Bit8
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0

复位后: 0 0 0 0 0 0 0 0

TC7 寄存器偏移量: \$001E-\$001F

Bit15	Bit14	Bit13	Bit12	Bit11	Bit10	Bit9	Bit8
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0

复位后: 0 0 0 0 0 0 0 0

2.12 脉冲累加器A控制寄存器(PACTL)

寄存器偏移量: \$0020

Bit7	6	5	4	3	2	1	Bit0
0	PAEN	PAMOD	PEDGE	CLK1	CLK0	PAOVI	PAI

复位后: 0 0 0 0 0 0 0 0

可在任何时候读或写。

寄存器 PACTL 决定了 PAI 的工作方式及其参数, 还包括 PAI 的启动、确定定时器系统的时钟信号源、中断管理等。其各位作用如下:

PAEN:脉冲累加系统使能位, 该位与 TEN 相互独立。PAEN=1 时, PAI 的输入部分才进行引脚信号检测, 同时 8 位脉冲累加器 PAC3、PAC2 被禁止。

0: PAI 系统禁止。

1: PAI 系统使能。

PAMOD: 脉冲累加器模式控制位。当 TEN=0 时, 没有 ECLK / 64 时钟, 如果启动 PAI,

它只能工作在事件计数方式。

0: 事件计数模式。

1: 门控时间累加模式。这时 TEN 必须为 1。

PEDGE: 脉冲累加器有效边沿设定位, 该位的作用效果与 PAMOD 的状态

有关。

(1)PAMOD: 0 (事件计数模式)时:

0: 对脉冲输入引脚(PT7 / PAI)的下降沿计数。

1: 对脉冲输入引脚(PT7 / PAI)的上升沿计数。

(2)PAMOD: 1 (门控时间累加模式)时:

0: 当脉冲输入引脚(PT7 / PAI)为高电平时, 允许 ECLK / 64 时钟脉冲计入脉冲累加器, 并在随后的输入弓 I 脚下降沿置位 PAIF 标志。

1: 当脉冲输入引脚(PT7 / PAI)为低电平时, 允许 ECLK / 64 时钟脉冲计入脉冲累加器, 并在随后的输入引脚上升沿置位 PAIF 标志。

CLKI、CLK0: 定时器模块时钟选择位, 如果脉冲累加器被禁止(PAEN: 0), 来自定时器的预分频器时钟总是作为定时计数器的时钟源。时钟源的改变在这两位写入后立即生效。其作用见表 7-6。

表 7-6 时钟选择

CLK1	CLK0	定时计数器时钟源
0	0	预分频时钟
0	1	PACLK
1	0	PACLK/256
1	1	PACLK/65536

PAOVI: 脉冲累加器溢出中断允许位。

0: 禁止溢出中断。

1: 当 PAOVF 置位时申请中断。

PAI: 脉冲累加器输入中断允许位。

0: 禁止输入中断

1: 当 PAIF 置位时申请中断。

2.13 脉冲累加器A标志寄存器(PAFLG)

寄存器偏移量: \$0021

Bit7	6	5	4	3	2	1	Bit0
0	0	0	0	0	0	PAOVF	PAIF
复位后: 0	0	0	0	0	0	0	0

可在任何时候读或写。

PAI 子系统可能产生两种中断，累加器溢出和事件中断。当 PACNT 溢出时，先置位 PAOVF，当 PAOVI=1 时将发出中断请求。而 PAIF 则在 PAI 检测到有效事件时置位，当 PAI=1 时还将发出中断请求。当 TSCR 寄存器中的 TFFCA=1 时，访问 PACNT 将清除 PAFLG 中的所有标志位。PAOVF、PAIF 位于寄存器 PAFLG 中，其意义如下：

PAOVF：脉冲累加器溢出标志位。

0：上次清除以来，PACNT 没有回滚到\$0000。

1：PACNT 从\$FFFF 回滚到\$0000。将\$02 写入到 PAFLG 将自动清除该位。

PAIF：脉冲累加器输入边沿有效标志位。

0：上次清除以来，PAI 引脚尚未检测到预期的有效边沿。

1：在输入引脚检测到有效边沿。在事件计数模式下，选定的有效边沿在计数的同时触发该位置 1；在门控时间累加模式下，输入引脚门控信号的后沿触发该位置 1，将\$01 写入到 PAFLG 将自动清除该位。

2.14 脉冲累加寄存器(PACN3、PACN2、PACN1、PACN0)

PACN3 寄存器偏移量：\$0022

Bit7	6	5	4	3	2	1	Bit0
------	---	---	---	---	---	---	------

复位后：0 0 0 0 0 0 0 0

PACN2 寄存器偏移量：\$0023

Bit7	6	5	4	3	2	1	Bit0
------	---	---	---	---	---	---	------

复位后：0 0 0 0 0 0 0 0

PACN1 寄存器偏移量：\$0024

Bit7	6	5	4	3	2	1	Bit0
------	---	---	---	---	---	---	------

复位后：0 0 0 0 0 0 0 0

PACN0 寄存器偏移量：\$0025

Bit7	6	5	4	3	2	1	Bit0
------	---	---	---	---	---	---	------

复位后：0 0 0 0 0 0 0 0

可在任何时候读或写。

PACN3、PACN2、PACN1、PACN0 是四个 8 位的寄存器，PAC3、PAC2 级连后形成 PACA，PAC1、PAC0 级连后形成 PACB，它们反映外部事件或者时间的累计结果，程序可以随时读取或通过写入设定初始值。当 PACA / PACB 允许 (PACTL: \$20 寄存器中的 PAEN=1 / PBCTL: \$30 寄存器中的 PBEN=1) 时，寄

寄存器 PACN3、PACN2 / PACN1、PACN0 分别是 PACA、PACB 的高、低位字节，当 PACN1 从\$FF 回滚到\$00，寄存器 PBFLG(\$31)中的 PBOVF 中断标志置 1。

正确的寄存器访问方式是在单个时钟周期内完成，即采用字读写指令。分开访问高 / 低位字节可能得到错误的结果。但如果只需要寄存器的高 8 位或低 8 位，则读操作不受限制，但写操作仍需要注意。

2.15 模数递减计数器控制寄存器(MCCTL)

寄存器偏移量：\$0026

Bit7	6	5	4	3	2	1	Bit0
MCZI	MODMC	RDMCL	ICLAT	FLMC	MCEN	MCPR1	MCPR0
复位后：0	0	0	0	0	0	0	0

可在任何时候读或写。

该寄存器主要控制 MDC 及相关操作。各位意义及作用如下：

MCZI： 模数计数器下溢出中断允许位。当 MDC 回 0 标志 MCZF 置位时，是否申请中断取决于 MCZI。

0：禁止中断。

1：允许中断。

MODMC： 模数方式允许位。该位设定 MDC 的工作方式，即选择单次或模数循环方式。

0： 单次计数方式，计数器从设定值递减到 0 后停止。

1： 模数计数方式，当计数器递减回 0 后，加载最新设定值，并开始新一轮计数过程。

注意：为保证正确运行，修改 MODMC 位之前，必须清除 MCEN 位使模数计数器复位到\$FFFF。

RDMCL： 模数读取选择位。该位决定读模数寄存器返回的内容。

0： 返回模数计数器的当前值。

1： 返回重新加载所用的常数。

ICLAT： 捕捉寄存器强制转移控制位。当输入捕捉工作在锁存方式时(ICSYS(\$2B)寄存器中的 LATQ 和 BUFEN 位置位)，向该位写 1 立即强制将捕捉寄存器 TC0-TC3 以及对应的 8 位脉冲累加器的内容转移到保持寄存器，同时相

关的脉冲累加器自动清 0。向该位写 0 无效，读该位总是返回 0。

FLMC：模数计数器强制加载控制位。该位只在模数递减计数器允许时 (MCEN=1) 有效，向该位写 1 将把模数常数寄存器的值加载到模数计数器，同时复位模数计数器的定标器。向该位写 0 无效。

当 MODMC=0 时，计数器开始计数，并在到达\$0000 后停止。读该位总是返回 0。

MCEN：模数递减计数器允许位。当 MCEN=0 时，计数器被预置为\$FFFF，这可以避免在计数器启动的初期出现中断标志。

0： 模数计数器禁止运行。

1： 模数计数器使能。

MCPR1、MCPR0：模数计数器定标器分频常数设定位。设定的分频常数不能立即起作用，必须等到模数计数器进行加载操作。当 MCPR1、MCPR0=00、01、10、11 时，对应的分频常数分别为 1、4、8、16。

2.16 输入脉冲累加器控制寄存器(ICPAR)

寄存器偏移量：\$0028

Bit7	6	5	4	3	2	1	Bit0
0	0	0	0	PA3EN	PA2EN	PA1EN	PA0EN
复位后：0	0	0	0	0	0	0	0

可在任何时候读或写。

8 位脉冲累加器 PAC3、PAC2 只有在 PACTL(\$20)中的 PAEN=0 时，才能被允许，若 PAEN=1，PA3EN 和 PA2EN 无效。同样，8 位脉冲累加器 PAC1、PAC0 只有在 PBCTL(\$30)中的 PBEN=0 时，才能被允许，若 PBEN=1，PA1EN 和 PA0EN 无效。当 PA_xEN=1 时，对应的 8 位脉冲累加器通道 x 使能，反之则禁止。

2.17 输入覆盖控制寄存器(ICOVW)

寄存器偏移量：\$002A

Bit7	6	5	4	3	2	1	Bit0
NOVW7	NOVW6	NOVW5	NOVW4	NOVW3	NOVW2	NOVW1	NOVW0
复位后：0	0	0	0	0	0	0	0

可在任何时候读或写。

该寄存器共有 8 个覆盖禁止位，对应 8 个 IC 通道，用来选择上次捕捉结果

是否允许被覆盖。如果某位 $NOVW_x=0$ ，新的捕捉事件或转移动作发生时，不论捕捉或保持寄存器是否为空，均直接保存新的结果；反之若 $NOVW_x=1$ ，对应的寄存器不允许被覆盖，但若处于空白状态则仍然允许覆盖。这样可以避免捕捉结果在被读取或者转移到保持寄存器前丢失。

2.18 输入系统控制寄存器(ICSYS)

寄存器偏移量：\$002B

Bit7	6	5	4	3	2	1	Bit0
SH37	SH26	SH15	SH04	TFMOD	PACMX	BUFEN	LATQ
复位后：0	0	0	0	0	0	0	0

可在任何时候读。

可能一次可写（test mode=0 时）但是当 test mode=1 时，写总是被允许的。

该寄存器用于对 IC 和 PAC 部分的工作进行设置与控制。其各位意义和作用如下：

SHxy: x、y 通道输入共享控制位。

0: x、y 通道正常工作，各自使用对应的引脚。

1: 通道 x 的边沿检测电路及延迟电路同时作用于 x、y 通道，即在 y 通道上也引发与 x 通道相同的动作。

TFMOD: 标志设置方式控制位，寄存器 ICSYS(\$2B)中的 TFMOD 位与寄存器 ICOVW(\$2A)一起使用可以使定时器的中断在捕捉到两个事件后才产生，而不是每次捕捉事件都产生中断，两次捕捉的结果分别存放在捕捉和保持寄存器中。如果在队列方式下将 TFMOD 置 1，同时 NOVW 置位且某通道 n 的捕捉和保持寄存器均为空白状态，那么第一个捕捉事件将首先用主定时器值更新对应的输入捕捉寄存器 TCn，下一个事件发生时，TCn 中的数值将首先被转移到保持寄存器 TCnH，然后更新 TCn。这时，CnF 中断标志才被置位。

在所有其他的输入捕捉情况下，PTn 引脚发生的有效外部事件将置位中断标志。

0: 当对应的输入引脚满足预定的捕捉条件时，寄存器 TFLG1(\$0E)中的标志 C3F-COF 将被相应置位。

1: 如果在队列方式下 (BUFEN=1 且 LATQ=0)，那么标志位 C3F-COF 只有在对应的保持寄存器被填充后才置位。如果没有启用队列方式，那么标志位

C3F-COF 的置位条件与 TFMOD: 0 时相同。

PACMX: 8 位脉冲累加器饱和控制位,即选择计数到\$FF 后是否保持不变。饱和功能允许时\$FF 就代表计数值已经达到 255 或更多。换言之,可记忆是否曾经达到\$FF。

0: 正常工作,计数达到\$FF 后,下一计数使其回到\$00。

1: 计数达到\$FF 时,停止加 1 操作,即进入饱和状态。

BUFEN: IC 缓冲允许位。

0: 禁止 IC 及 PAC 的保持寄存器。

1: IC 及 PAC 保持寄存器使能。锁存方式由控制位 LATQ 设定,当位 LATQ=1 时,向寄存器 MCCTL(\$26)中的 ICLAT 位写 1,将导致捕捉寄存器和脉冲累加器将其中数值转移到各自的保持寄存器。

LATQ: 输入锁存及队列方式允许位。当控制位 BUFEN=1 时,IC 和 PAC 的保持寄存器使能,反之不能通过 LATQ 进入锁存方式。当 LATQ、BUFFEN 同时为 1 时,向 MCCTL(\$26)中的 ICLAT 位写 1,IC 和 PAC 寄存器内容将转移到保持寄存器。

0: IC 工作在队列方式。当输入引脚满足预定的捕捉条件时,主定时器的值被记录到捕捉寄存器,当下一个捕捉事件出现时,捕捉寄存器将其内容转移到保持寄存器,以便存放新的捕获值。

1: IC 工作在锁存方式。当模数计数器自然回 0,或者向计数器 MCCNT 直接写\$0000 时,将进行锁存操作,结果,捕捉寄存器和脉冲累加器将其内容转移到保持寄存器,同时 8 位的脉冲累加器清 0。

2.19 脉冲累加器B控制寄存器(PBCTL)

寄存器偏移量: \$0030

Bit7	6	5	4	3	2	1	Bit0
0	PBEN	0	0	0	0	PBOVI	0
复位后: 0	0	0	0	0	0	0	0

可在任何时候读或写。

16 位的脉冲累加器 B(PACB)系两个 8 位的脉冲累加器 PAC1 和 PAC0 级连后形成的。当 PBEN 置 1 时, PACB 启动,它与 IC0 共享同一个输入引脚。

PBEN: 脉冲累加器 B 使能位, 它与 TEN 相互独立。只要 PBEN=1, 脉冲累加器 B 即可工作, 与定时器是否允许无关, 这与脉冲累加器 A 不同。

0: 脉冲累加器 B 禁止。8 位的 PAC1 和 PAC0 可通过各自的允许位(在寄存器 ICPACR 中, 地址\$28)使能。

1: 脉冲累加器 B 使能。实际上 PACB 为 PAC1、PAC0 级连形成, PAC1 为高位、而 PAC0 为低位字节, 这时寄存器 ICPACR 中的控制位 PAIEN、PAOEN 无效。

PBOVI: 脉冲累加器 B 溢出中断允许位。

0: 禁止 PBOVF 申请中断。

1: 当 PBOVF=1 时, 申请中断。

2.20 脉冲累加器B标志寄存器(PBFLG)

寄存器偏移量: \$0031

Bit7	6	5	4	3	2	1	Bit0
0	0	0	0	0	0	PBOVF	0
复位后: 0	0	0	0	0	0	0	0

可在任何时候读或写。

该寄存器只有一个标志位 PBOVF, 当 16 位的 PACB 从\$FFFF 回滚到\$0000, 或 8 位脉冲累加器 3(PAC1)从\$FF 回滚到\$00 时, 该位置 1。将\$01 写入 PBFLG 清除该位。当寄存器 TSCR(\$06)中的 TFFCA 位置位时, 访问 PACN1 和 PACN0 清除该标志。

2.21 脉冲累加器保持寄存器 (PA3H-PA0H)

PAC3H 寄存器偏移量: \$0032

Bit7	6	5	4	3	2	1	Bit0
复位后: 0	0	0	0	0	0	0	0

PA2H 寄存器偏移量: \$0033

Bit7	6	5	4	3	2	1	Bit0
复位后: 0	0	0	0	0	0	0	0

PA1H 寄存器偏移量: \$0034

Bit7	6	5	4	3	2	1	Bit0
复位后: 0	0	0	0	0	0	0	0

PA0H 寄存器偏移量: \$0035

Bit7	6	5	4	3	2	1	Bit0
------	---	---	---	---	---	---	------

复位后: 0 0 0 0 0 0 0 0

可在任何时候读。写不起作用。

当 ICPACR(\$28)中 PAnEN=1 时, 通道 PAn 使能, 寄存器 PA3H-PAOH 用于为 PAn 提供锁存功能。

2.22 模数递减计数器工作寄存器(MCCNT)

寄存器偏移量: \$0036-\$0037

Bit15	Bit14	Bit13	Bit12	Bit11	Bit10	Bit9	Bit8
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0

复位后: 1 1 1 1 1 1 1 1

MDC 启动后, MCCNT 用来对定标器输出的时钟进行递减计数, 也可作为定时基准。对该寄存器的访问应在一个时钟周期内完成, 即按字访问。对高低位字节分别访问可能得到错误的结果。如果 MCCTL 中的 RDMCL=0, 读 MCCNT 寄存器返回计数器的当前值; 反之如果 RDMCL=1, 返回每次重新加载所用的常数。

当寄存器 ICSYS(\$2B)中的 LATQ、BUFEN 均为 1 时, 如果将\$0000 写入到 MCCNT 和模数计数器, 输入捕捉和脉冲累加寄存器将被锁存。将\$0000 写入到 MCCNT 后, 模数计数器将保持为 0, 且不会将 MCFLG 中的 MCZF 标志置位。

如果模数方式允许(MODMC=1), 对该地址进行写操作将更新常数寄存器, 但计数器不会立即更新, 必须等到计数器回 0 后重新加载; 如果希望立即加载, 通过寄存器 MCCTL (\$26)中的 FLMC 位可以将新值立即加载到计数器。如果模数方式未允许(MODMC=0), 写该地址将清 0 定标器, 并用写入值立即更新计数器, 然后开始一次递减计数, 到 0 后停止。

2.23 IC保持寄存器 (TC0H-TC3H)

TC0H 寄存器偏移量: \$0038-\$0039

Bit15	Bit14	Bit13	Bit12	Bit11	Bit10	Bit9	Bit8
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0

复位后: 0 0 0 0 0 0 0 0

TC1H 寄存器偏移量: \$003A-\$003B

Bit15	Bit14	Bit13	Bit12	Bit11	Bit10	Bit9	Bit8
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0

复位后: 0 0 0 0 0 0 0 0

TC2H 寄存器偏移量: \$003C-\$003D

Bit15	Bit14	Bit13	Bit12	Bit11	Bit10	Bit9	Bit8
-------	-------	-------	-------	-------	-------	------	------

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
复位后: 0	0	0	0	0	0	0	0
TC3H 寄存器偏移量: \$003E-\$003F							
Bit15	Bit14	Bit13	Bit12	Bit11	Bit10	Bit9	Bit8
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
复位后: 0	0	0	0	0	0	0	0

这些寄存器用于为相应的捕捉寄存器 TC0-TC3 提供锁存功能。

定时器与 I / O 共享引脚的说明

定时器与 PORTT 共享 8 个 I/O 引脚, 复位后各个引脚默认为通用输入输出, 当定时器相关功能使能后, 将取代通用 I / O 功能。引脚 PT0-3 由 IC / OC、PAC 及通用 I/O 共享, 引脚 PT4-7 由 IC / OC 及通用 I / O 共享, 此外引脚 PT0、PT7 还供 16 位脉冲累加器 A、B 作为输入使用。其他请参看第 6 章 TIM 模块有关说明, 不再赘述。

不同模式下的定时器及模数计数器的运行状况

(1)停止(STOP): 因为 PCLK 和 ECLK 均已停止, 定时器和模数计数器关闭。

(2)调试方式(BDM): 只要不满足 TSBCK=1, 定时器保持运行。

(3)中断等待(WAIT): 只要不满足 TSWAI=1, 计数器保持运行。

(4)正常(Normal): 只要不满足 TEN=0 和 MCEN(MCCTL 中)=0, 定时器保持运行。

(5)禁止(TEN=0): 定时器和 MDC 停止, 但寄存器可访问。ECLK / 64 时钟被禁止。

(6)禁止(PAEN=0): 所有的脉冲累加器动作停止, 但寄存器可以访问。

(7)MCEN: 0: 模数计数器停止。

(8)PAEN: 1: 16 位脉冲累加器 A 激活。

(9)PAEN: 0: 8 位脉冲累加器 3 和 2 可以激活。

(10)PBEN: 1: 16 位脉冲累加器 B 激活。

(11)PBEN: 0: 8 位脉冲累加器 1 和 0 可以激活。

第三节 ECT应用实例

3.1 定时器编程步骤

1、初始化：

设定预分频系数，设定工作方式，定时器溢出中断使能，定时器使能

2、中断函数

用户自己的代码，清标志位

```
void ECT_Init(void)
{
    TSCR2_PR    = 7; //预分频系数为 8
    .....
    TSCR2_TOI    = 1; //定时器溢出中断使能
    TSCR1_TEN    = 1; //定时器使能
}
#pragma CODE_SEG __NEAR_SEG NON_BANKED
void TimerOverFlow(void)
{
    //用户自己的代码
    TFLG2_TOF    = 1; //清楚定时器溢出中断标志位
}
```

3.2 输入捕捉IC:

本试验的辅助设备有：信号发生器、示波器。

试验目的：通过连续记录输入信号的两个上升沿，用该程序可以计算出输入信号的频率；同时，利用脉冲累加器可以记录输入脉冲数。

```
#include <hidef.h>          /* common defines and macros */
#include <mc9s12dp256.h>     /* derivative information */
#pragma LINK_INFO DERIVATIVE "mc9s12dp256b"
int count=0;
float f;
double f1=2000000,first=0,second=0,n,N;
void main(void)
{
    DisableInterrupts;
    TSCR2=0X82;
    PACTL=0X20;
    TIOS=0XFE; //设定 pt0 输入捕捉口
```



```

    TCTL4=0X01;
    ICSYS=0X0A;
    PBCTL_PBEN=0X00;
    ICPAR=0X01;
    TIE=0X01;
    TSCR1=0X80;
    EnableInterrupts;
    for(;;)
    {
    }
}

#pragma CODE_SEG __NEAR_SEG NON_BANKED
interrupt void CH0IC(void)
{
    first=TC0H;
    second=TC0;
    n=count*65535+second-first;
    f=f1/n;
    N=PA0H;
    TFLG1=0X01;
    count=0;
}

interrupt void TOI(void)
{
    count++;
    TFLG2_TOF=1;
}

```

3.3 通道 6 输出比较

本试验的辅助设备有：示波器。

试验目的：利用定时器溢出中断产生脉冲周期，利用通道 6 输出比较产生不同脉宽的波形，但此方法产生的波形其周期不能改变。

```

double count,counter;
void main(void) {
    DisableInterrupts;
    TSCR2=0X82;
    TIOS=0XFE;//设定 pt6 输出捕捉口
    TCTL1=0X30;
    TC6=0X3333;
    TTOV=0X40;
    TIE=0X00;
    TSCR1=0X80;
    EnableInterrupts;
}

```

```

    for(;;)
    {
    }

    #pragma CODE_SEG __NEAR_SEG NON_BANKED
    interrupt void CH6OC(void)
    {
        counter++;
        if(counter==100)
            TC6=0X8888;
        if(counter==200)
            TC6=0XCCCC;
        if(counter==300)
            TC6=0XDEEE;
        TFLG2_TOF=1;
    }

```

3.4 通道 7 输出比较

本试验的辅助设备有：示波器。

试验目的：利用通道 7 输出比较中断产生脉冲周期，利用通道 6、4 输出比较产生不同脉宽的波形，此方法产生的波形既可以改变周期同时也能够改变脉宽。

```

#include <hdef.h>          /* common defines and macros */
#include <mc9s12dp256.h>    /* derivative information */
#pragma LINK_INFO DERIVATIVE "mc9s12dp256b"
double count,counter;
void main(void) {
    DisableInterrupts;
    TIOS=0XF0;//设定 pt6 输出捕捉口
    TSCR2=0X0a;
    TFLG1=0X00;
    TFLG2=0X00;
    TCTL1=0X62;
    ICSYS=0X0a;
    OC7M=0X50;
    OC7D=0X50;
    TC7=0X00C0;
    TC6=0X0060;
    TC4=0X0099;
    TIE=0X00;
    TSCR1=0X80;
    EnableInterrupts;

```

```

        for(;;)
        {; }
    }

```

3.5 模数递减计数器

本试验的辅助设备有：连接导线若干。

试验目的：利用模数递减计数器回零中断，控制五个小灯依次点亮，同时点亮的時間间隔由慢到快。

```

#include <hidef.h>          /* common defines and macros */
#include <mc9s12dp256.h>    /* derivative information */
#pragma LINK_INFO DERIVATIVE "mc9s12dp256b"
long counter=0;
void main(void)
{
    DisableInterrupts;
    DDRB=0XFF;
    PORTB=0X00;
    MCCTL_MCEN=0;
    MCCTL_MCZI=1;
    MCCTL_MODMC=1;
    MCCTL_MCPR1=1;
    MCCTL_MCPR0=1;
    MCCNT=0XAAAA;
    TSCR1=0X80;
    TSCR2=0X00;
    MCCTL_MCEN=1;
    EnableInterrupts;
    for(;;)
    {; }
}

#pragma CODE_SEG __NEAR_SEG NON_BANKED
interrupt void MDC(void)
{
    counter++;
    if(counter<=10)
    PORTB=0X01;
    if((counter>10)&&(counter<=20))
    {
        PORTB=0X02;
        MCCNT=0X2222;
    }
    if((counter>20)&&(counter<=30))

```

```
{  
    PORTB=0X04;  
    MCCNT=0X4444;  
}  
if((counter>30)&&(counter<=40))  
{  
    PORTB=0X08;  
    MCCNT=0X8888;  
}  
if((counter>40)&&(counter<=50))  
{  
    PORTB=0X10;  
    MCCNT=0XD888;  
}  
if(counter>50)  
    counter=0;  
MCFLG_MCZF=1;  
}
```

第三章 SCI模块

第一节 SCI寄存器简介

SCI 是一种采用 NRZ 格式的异步串行通信接口，它内置独立的波特率产生电路和 SCI 收发器，可以选择发送 8 或 9 个数据位(其中一位可以指定为奇或偶校验位)。

SCI 是全双工异步串行通信接口，主要用于 MCU 与其他计算机或设备之间的通信，几个独立的 MCU 也能通过 SCI 实现串行通信，形成网络。

MC12 里有两个 SCI (SCI0 和 SCI1)。设计 SCI 串口通信程序，主要是掌握八个寄存器，设置好初始化。

1.1 波特率控制寄存器(SCIBDH、SCIBDL)

SCIBDH 和 SCIBDL 一起构成了一个 16 位的波特率控制寄存器。SBR12~SBR0 为波特率常数，见表 9-1。

表 9-1 波特率控制寄存器

寄存器名: SCIBDH、SCIBDL;地址:\$00C0、\$00C1;复位默认值: 00000000B、00000100B 读操作: 任意; 写操作: SBR12~SBR0 任意, 低位字节写入后生效。SBR13~SBR15 未定义									
位		B7	B6	B5	B4	B3	B2	B1	B0
SCIBDH	读写	未定义	未定义	未定义	SBR12	SBR11	SBR10	SBR9	SBR8
SCIBDL	读写	SBR7	SBR6	SBR5	SBR4	SBR3	SBR2	SBR1	SBR0

SCI 的波特率由下述公式决定:

波特率: $MCLK / (16 \times BR)$

或 BR: $MCLK / (16 \times \text{波特率})$

其中 BR 是波特率常数, 设定时写入到 SBR12~SBR0。复位后, 在 SCICR2 寄存器中的 TE、RE 位第一次置 1 前, 波特率发生器是关闭的, 而且, 当 SBR12~

表 9-2 常用波特率及波特率常数

波特率		110	300	600	1200	2400	4800	9600	14400	19200	38400
波特率因子	P=4MHz	2273	833	417	208	104	52	26	17	13	—
	P=8MHz	4545	1667	833	417	208	104	52	35	26	13

SBR0=0 时，波特率发生器也会被关闭。

通常选用波特率为 9600。

BTST、BSPL 和 BRLD 保留用于检测功能。

1.2 控制寄存器 1(SCICR1)

SCI 的工作方式主要由该寄存器设置，包括回送、单线等方式选择，帧格式、唤醒、空闲检测类型以及奇偶校验等，见表 9-3。其各位意义如下：

表 9-3 SCI 控制寄存器 1

寄存器名：SCICR1；地址：\$00C2；复位默认值：00000000B；读操作：任意；写操作：任意								
位	B7	B6	B5	B4	B3	B2	B1	B0
读写	LOOPS	SCISWAI	RSRC	M	WAKE	ILT	PE	PT

LOOPS: SCI 回送模式 / 单线模式允许位，接收器的输入由 RSRC 位选择确定，发送器的输出受相应的 DDRS 位(S 口 I / O 方向控制位)控制。要使用回送或者单线模式，发送、接收器必须同时允许工作。在 LOOPS=1 期间，如果与 TxD 引脚对应的方向控制位被置 1，那么 TxD 引脚就输出 SCI 的信号；如果方向控制位被清 0，那么这时若 RSRC=0，TxD 引脚就变成高电平(空闲状态)，反之若 RSRC=1，TxD 引脚就变成高阻态。

0: SCI 发送和接收部分正常工作。

1: SCI 接收部分与 RxD 引脚断开，空出来的 RxD 引脚可以用作通用 I / O。

SCISWAI: 等待模式下 SCI 停止位

0: 在等待模式下允许 SCI

1: 在等待模式下禁止 SCI

RSRC: 接收器信号源选择位，当 LOOPS=1 时，RSRC 决定接收器的内部反馈信号路径。

0: 接收器的输入在内部连接到发送器输出(并非 TxD 引脚)。

1: 接收器的输入连接到 TxD 引脚。

M: 方式选择位(选择字符帧格式)。

0: 1 个起始位，8 个数据位，1 个停止位。

1: 1 个起始位，8 个数据位，第 9 个数据位，1 个停止位。

WAKE: 唤醒选择位。

0: 介质空闲唤醒。

1: 地址标志(最后一个数据位为 1)唤醒。

ILT: 空闲检测方式选择位，该位在 SCI 接收器可以使用的两种空闲检测方式中选择一种。

0: 快速检测，SCI 在一个帧的开始位后立即开始对“1”计数，因此，停止位以及停止位前面的任何“1”均被计算在内，这样可以提前检测到空闲状态。

1: 保守检测，SCI 在停止位后才开始对“1”计数，因此最后一个字节的停止位以及该位以前的各个为“1”的位，对检测的时间长短无影响。

PE: 奇偶校验允许位。

0: 禁止奇偶校验。

1: 允许奇偶校验。

PT: 奇 / 偶校验选择位，如果奇偶校验允许，该位决定收发器使用奇校验还是偶校验。如果选择偶校验，当数据中有偶数个“1”时，校验位置 0，否则校验位置 1。

0: 选择偶校验。

1: 选择奇校验。

1.3 控制寄存器 2(SCICR2)

该寄存器主要控制收发器的使能、中断允许及其他辅助操作，见表 9-4。其各位意义如下：

表 9-4 SCI 控制寄存器 2

寄存器名: SCICR2;地址: \$00C3;复位默认值: 00000000B;读操作:任意值;写操作: 任意								
位	B7	B6	B5	B4	B3	B2	B1	B0
读写	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK

TIE: 发送中断允许位，清 0 时禁止 TDRE 产生中断，若置 1 则允许 TDRE 位置 1 时产生 SCI 中断请求。

TCIE: 发送结束中断允许位，清 0 时禁止 TC 产生中断，若置 1 则允许 TC 位置 1 时产生 SCI 中断请求。

RIE: 接收中断允许位，清 0 时禁止 RDRF 和 OR 产生中断，若置 1 则允许 RDRF 或 OR 置 1 时产生 SCI 中断请求。

ILIE: 空闲中断允许位，清 0 时禁止 IDLE 产生中断，若置 1 则允许 IDLE 位置 1 时产生 SCI 中断请求。

TE: 发送允许位。该位由 0 置 1 时可用来发送空闲报头。

0: 发送器禁止。

1: 允许 SCI 发送部分工作，TxD 引脚(PS1 / PS3)用于发送。

RE: 接收允许位。

0: 接收器禁止。

1: 允许 SCI 接收器工作。 **RWU:** 接收器唤醒控制位。

RWU: 接收器唤醒控制位

0: SCI 接收器正常工作。

1: 允许唤醒功能，禁止接收器中断。通常，硬件通过自动清除该位来唤醒接收器。

SBK: 中止符发送允许位。只要该位保持为 1，发送器就不停地发出“0”；如果变为 0，当前的全“0”帧发送结束后，TxD 引脚将变成空闲状态。如果 SBK 开关一次，发送器将只发出 10(11)个“0”，然后复原，处于空闲或发送数据状态。

0: 中止符产生器关闭。

1: 产生中止符，至少 10 或 11 个连续的“0”。

1.4 状态寄存器 1 (SCISR1)

该寄存器反映 SCI 运行过程中的各种状态，包括发送器和接收器的状态信息和接收器的出错信息，见表 9-5。

表 9-5 SCI 状态寄存器

寄存器名: SCISR1;地址: \$00C4;复位默认值: 11000000B;读操作: 任意;写操作: 无意义								
位	B7	B6	B5	B4	B3	B2	B1	B0
读	TDRE	TC	RDRF	IDLE	OR	NF	FE	PR

当 SCI 硬件处于某些状态时，SCISR1 中的对应位置位，并通过特定的确认动作清除。先后对 SCISR1 和 SCIDRL 进行读操作将清除与接收有关的标志位

(RDRF、IDLE、OR、NF、FE 和 PF)，读 SCISR1 然后写 SCIDRL 将清除与发送有关的标志位(TDRE 和 TC)。

TDRE: 发送保持器空标志位。发送前必须读 SCISR1，并确认 TDRE=1，然后将新的数据写入发送保持器以开始发送过程。复位后该位为 1。

0: SC0DR 处于忙状态。

1: 发送保持器的数据已被传送到发送移位器，这时可以向发送保持器写入新的数据

TC: 发送结束标志。该位在发送器空闲(无发送动作)时置位。读 SCISR1，然后写 SCIDR 将清除该位。

0: 发送器忙。

1: 发送器空闲。

RDRF: 接收数据就绪标志。当收到的字符已经在 SCIDR 中就绪时，RDRF 置 1，顺次读取 SCISR1 和 SCIDR 将会自动清除 RDRF。该位被清除后，必须等到 RxD 线变为活动，然后重新变成空闲以后，IDLE 位才会被再次置 1。

0: SCIDR 空。

1: SCIDR 中数据已就绪。

IDLE: 空闲标志。检测到接收器 RxD 端空闲(收到 10 或者 11 个以上连续的“1”)。当 RWU 位为 1 时，空闲状态不会使该位置 1。该位被清除后，必须等到 RDRF 置位(RxD 线变为活动，然后重新变成空闲)，IDLE 位才会被再次置 1。

0: RxD 线活动。

1: RxD 线空闲。

OR: 重叠错误标志。如果接收数据寄存器中的数据尚未读取(RDRF=1)，接受移位寄存器又准备向其传送新的数据，则称为重叠错误，该位被置 1。必须清除该位，才能使新的数据进入接收数据寄存器。

0: 无重叠。

1: 出现重叠错误。

NF: 噪声错误标志。噪声错误出现时，该位与 RDRF 在同一个周期内置位，但如果同时或已经出现重叠错误，该位不置位。

0: 采样结果一致。

1: 在起始位、数据位或停止位接收期间检测到噪声。

FE: 帧格式错误。如果在应该出现停止位的时刻，检测到 0，则该位置位。顺次读取寄存器 **SCISR1** 和 **SCIDR** 将清除 FE 标志。

0: 检测到停止位。

1: 在预期的停止位处检测到 0。

PF: 奇偶错误标志。指示收到数据的奇偶性与校验位是否一致。奇偶校验允许 (PE=1) 时，该标志才有意义。所要求的奇偶性由 **SC0CR1** 中的 PT 位决定。

0: 奇偶校验正确。

1: 奇偶校验错误。

1.5 状态寄存器 2(SCISR2)

表 9-6 SCI 状态寄存器 2

寄存器名: SCISR2;地址: \$00C5;复位默认值: 00000000B;读操作: 任意;写操作: 写 RAF 无意义								
位	B7	B6	B5	B4	B3	B2	B1	B0
读写	未定义	未定义	未定义	未定义	未定义	BRK13	TXDIR	RAF

BRK13: 中止符长度控制位。

0: 中止符长度为 10 或 11 位。

1: 中止符长度为 13 或 14 位。

TXDIR: 单线模式下发送管脚数据方向控制位。

0: 单线模式下 TxD 脚用于输入。

1: 单线模式下 TxD 脚用于输出。

RAF: 接收器活动标志位。反映接收器是否处于活动状态。在搜索起始位的 RT1 期间该位置 1，当接收器检测到空闲状态或者出现一个伪起始位（通常由于噪声或波特率匹配错误引起）时，该位清 0。该位由接收器前端控制，

1.6 数据寄存器(SCIDRH、SCIDRL)

SCI 内部分别设有发送和接收两个数据寄存器，其低位都通过 **SCIDRL** 访问，读操作返回接收数据寄存器 **RDR** 的内容，写操作数据置入发送数据寄存器

TDR。当 $M=1$ 即运行在 9 位数据模式时，**SCIDRL** 和 **SCIDRH** 形成 9 位的 **SCI** 数据字，这时必须先写入 **SCIDRH**，以便与低位字节(**SCIDRL**)一起进入发送移位器。如果 $M=0$ 即 **SCI** 只用于 7 位或 8 位的数据传送，可以只访问 **SCIDRL**。当 $PE=1$ 即奇偶校验允许时，奇偶校验位由硬件负责，无需软件干预，见表 9-7。

表 9-7 SCI 数据寄存器

寄存器名: SCIDRH、SCIDRL;地址: \$00C6、\$00C7;复位默认值: 00000000B、00000000B; 读操作: 任意;写操作: 任意但写 R8 无意义。									
位		B7	B6	B5	B4	B3	B2	B1	B0
SCIBDH	读	R8	T8	0	0	0	0	0	0
	写	未定义		未定义	未定义	未定义	未定义	未定义	未定义
SCIBDL		R7T7	R6T6	R5T5	R4T4	R3T3	R2T2	R1T1	R0T0

R8: 接收到的位 8，该位写操作无效。当 **SCI** 设置成 9 位数据运行模式时，该位是从串行数据流中接收到的第 9 位。

T8: 发送位 8，任何时候可写。当 **SCI** 设置成 9 位数据模式时，该位是送到串行数据流的第 9 位。该位不必为每个数据重新设置，每次发送可重复使用。

R7T7-ROT0: 收 / 发数据位 7-0，读操作返回只读寄存器 **RDR** 的内容，写操作写入只写寄存器 **TDR**。

第二节 SCI应用示例

下面看看实验:

1、本实验需软件程序“串口调试助手 V2.0”。

2、实验步骤:

1) 保证单片机 5V 电源、串口通讯线与仿真板的连接无误;

2) 上电后建立新建文件，在 **main.c** 中输入下面程序并写如 **flash**;

3) 点击图标运行后，打开“助手”，在下面的发送框中输入任意数字或字符，然后选择自动或手动发送。

3、试验结果

在“助手”下边的发送框中输入任意数字或字符，然后发送，接收到数据后，上面的显示区就会将其以 **ASCII** 码或十六进制的形式（可选）显示出来。

4、源程序

```
#include <hidef.h> /* common defines and macros */
```

```

#include <mc9s12dp256.h>      /* derivative information */
#pragma LINK_INFO DERIVATIVE "mc9s12dp256b"
    byte sci_data;
//发送子函数
void SCI_Transmit(byte data){
while(!SCI0SR1_TDRE); //SC0DR 处于忙状态，等待。
SCI0DRL=data;
}
//接收子函数
void SCI_Receive(byte *data)
{
*data=SCI0DRL;
}
void main(void)
{ SCI0BDL=0x34; //总线为 8M，波特率为 9600
  SCI0CR2=0X2C; //允许中断，允许发送，允许接受
  while(1)
  {
    while(!SCI0SR1_RDRF); //SCIDRL 为空，等待数据就绪。
    SCI_Receive(&sci_data);
    SCI_Transmit(sci_data);
  }
}

```

下面的源程序是利用 SCI 做的一个足球答题系统，感兴趣的同学可以看一下：

```

#include <hidef.h>          /* common defines and macros */
#include <mc9s12dp256.h>    /* derivative information */
#pragma LINK_INFO DERIVATIVE "mc9s12dp256b"
byte sci_data;
//中断初始化
void SCI_Init(void) {
SCI0BDL=0x34;
SCI0CR2=0X2C;
}
//发送子函数
void SCI_Transmit(byte data){
while(!SCI0SR1_TDRE) ;
SCI0DRL=data;
}
//接收子函数
void SCI_Receive(byte *data){
*data=SCI0DRL;
}
//特定输出子函数
void printf(char *str){

```

```
while(*str!='\r'){
SCI_Transmit(*str);
*str++;
}
}
N0Choose(byte data){
switch(data){
    case '1':
        NO1();
        break;
    case '2':
        NO2();
        break ;
    case '3':
        NO3();
        break;
    case '4':
        NO4();
        break;
    case '5':
        NO5();
        break;

    default:
        break;
}
}
void main(void) {
SCI_Init();
printf("welcome to lipu's football-quiz system!\n\r");
printf("choose the problem number(1to5)\n\r");
while(1){
while(!SCI0SR1_RDRF);
SCI_Receive(&sci_data);
SCI_Transmit(sci_data);
N0Choose(sci_data);
}
}
void Right(void){
printf("\nyou are RIGHT.\nchoose the next question\n\r");}
void Wrong(void){
printf("\nyou are WRONG.\nchoose the next question\n\r");}
int NO1(void){
printf(".which country is the champion of World Cup at 2006?\n\r") ;
```

```
printf("A:Brazil B:Italy\n\r");
while(!SCI0SR1_RDRF);
SCI_Receive(&sci_data);
SCI_Transmit(sci_data);
switch(sci_data){
    case 'A':
        Wrong();
        break;
    case 'B':
        Right();
        break;
    default:
        break;
}
}
int NO2(void){
printf(".which country have the most champions of World Cup?\n\r") ;
printf("A:Brazil B:Italy\n\r");
while(!SCI0SR1_RDRF);
SCI_Receive(&sci_data);
SCI_Transmit(sci_data);
switch(sci_data){
    case 'B':
        Wrong();
        break;
    case 'A':
        Right();
        break;
    default:
        break;
}
}
int NO3(void){
printf(".which club is the champion of Spanish Prinera Divison at 06-07\n\r") ;
printf("A:Barcelona B:Real Madrid\n\r");
while(!SCI0SR1_RDRF);
SCI_Receive(&sci_data);
SCI_Transmit(sci_data);
switch(sci_data){
    case 'A':
        Wrong();
        break;
    case 'B':
        Right();
```

```
    break;
    default:
    break;
}
}
int NO4(void){
printf(".which club is the champion of Italian Serie A at 06-07\n\r") ;
printf("A:Inter Milan   B:AC.Milan\n\r");
while(!SCI0SR1_RDRF);
SCI_Receive(&sci_data);
SCI_Transmit(sci_data);
switch(sci_data){
    case 'B':
        Wrong();
        break;
    case 'A':
        Right();
        break;
    default:
        break;
}
}
int NO5(void){
printf(".who is the FIFA World Player at 2006\n\r") ;
printf("A:Henry   B:Ronaldiaho\n\r");
while(!SCI0SR1_RDRF);
SCI_Receive(&sci_data);
SCI_Transmit(sci_data);
switch(sci_data){
    case 'B':
        Wrong();
        break;
    case 'A':
        Right();
        break;
    default:
        break;
}
}
```

第四章 SPI模块

第一节 SPI模块介绍

1.1 SPI的功能特点

串行设备接口 SPI 主要用于同步串行通信,它使 MCU 具备了与外围设备以及其他微处理器进行同步通信的能力,也能够多主系统中实现处理器间的通信。可连接的设备包括简单的移位寄存器(例如 74LS165 用作并行输入口,或 74LS164 用作并行输出口等)、LCD 显示驱动器或 A/D 转换器的接口。MCU 可选择 8 种不同的位传送频率、两种不同的时钟极性、相位和位传送顺序,因此可直接与各个厂家生产的多种标准的串行外围接口器件连接。在串行外围接口中,数据和时钟线是分开的,在 SPI 格式中,时钟不包括在数据流中,它必须是另一个独立的信号线。MC9S12DP256 的 SPI 可定义为主机或从机方式,主要特性如下:

- (1)全双工、三线同步传送。
- (2)单个数据引脚的双向传送方式。
- (3)主机或从机工作方式。
- (4)每一晶体频率下可通过程序选择八种不同的主机位传送频率。
- (5)主机位传送频率最大 4MHz,当 MCU 总线频率=8MHz 时最小为 31.25kHz。
- (6)从机位传送频率最大 4MHz,允许频率范围为 0-4MHz。
- (7)可编程设置位时钟极性、相位和数据位传送顺序,即可选高位在前或低位在前。
- (8)发送完成中断标志。
- (9)多主机系统控制冲突保护中断标志。
- (10)写冲突标志保护。
- (11)可方便地与各种简单扩展器件接口,如 PLL、D/A、锁存器、LCD 显示驱动器等。

SPI 系统可在软件控制下构成各种不同复杂程度的系统,例如:

- (1)一个主 MCU 和一个或者几个从 MCU。
- (2)几个 MCU 相互联接构成多主机的系统(全分布式系统)。
- (3)一个主 MCU 和一个或者几个从 I / O 设备。
- (4)一个其他主微机系统和一个或者几个从 MCU。

1.2 SPI的组成与工作设置

(一)SPI 的结构与工作过程

1. SPI 的结构组成

SPI 系统主要由 8 位移位寄存器、时钟控制逻辑、引脚控制逻辑、SPI 控制逻辑和分频器以及波特率寄存器 SPIBR、状态寄存器 SPISR、控制寄存器 1SPICR1、控制寄存器 2SPICR2、数据寄存器 SPIDR 等 5 个寄存器组成,如图 1 所示。

SPI 的核心是一个 8 位移位寄存器。发送或接收时,引脚的数据流在时钟信号 SCK 的作用下移出或移入该寄存器。该寄存器对用户透明,CPU 通过 SPIDR 与其联系。发送时将数据写入 SPIDR 将直接进入移位寄存器,即所谓单缓冲;而接受到的数据则通过读数据缓冲器到达 SPIDR,因此称为双缓冲,这样可以保证移位寄存器正在接收时,CPU 能正确读取上一个收到的字节。

时钟逻辑主要用于为移位寄存器提供工作时钟。在主模式下,内部波特率产生逻辑为其提供时钟源,同时还通过 SCK 引脚输出到外部从器件;在从模式下,外部主器件通过 SCK 引脚提供时钟源。分频器的时钟频率通过对 SPIBD 的设定来选择,因而可以得到各种位传送速率。时钟控制逻辑将产生各种极性和相位的位时钟(SCK)信号,还可以选择传送时数据高位在先还是低位在先,以适应各种不同的外围器件。

控制寄存器 SPICR1、SPICR2 和 SPIBR 用来设置 SPI 的工作方式,包括主 / 从模式、数据位顺序、时钟极性、相位、波特率以及引脚逻辑等参数。状态寄存器 SPISR 保存 SPI 的工作状态,包括传送结束、写冲突和模式错误等。

SPI0 与 PS 口共享 PS7-PS4 引脚, SPI1 与 PP 口共享 PP7-PP4 引脚, SPI2 与 PP 口共享 PP3-PP0 引脚。当 SPI 系统使能时,四个引脚一般由通用 I/O 变为 SPI 的有关引脚(\overline{SS} 、SCK、MISO、MOSI)。但当 SPI 工作在双向模式时,个别引脚仍可用作通用 I/O。

若 $MSTR=1$ ，SPI 处于主机方式，其 \overline{SS} 引脚可选三种功能。

当 $DDS7=SSOE=0$ 时， \overline{SS} 引脚输出从机选择信号， SCK 输出 SPI 工作时钟。若要选中外部从机， \overline{SS} 引脚必须输出逻辑 0，外部从机在 SCK 的作用下接收或发送数据。当 \overline{SS} 输出逻辑 1 时，从器件被禁止。每一次传送时， \overline{SS} 输出自动变低以选中外部器件，在传送中间的空闲状态回到高电平以释放外部器件。

当 $DDS7=SSOE=1$ 时， \overline{SS} 引脚为 SPI 系统的工作方式冲突错误检测输入，正常情况下保持高电平。在分布式多主机系统中，多个器件均可能作为主机，但是同一时刻只能有一个主机。当已有一个器件作为主机并正在选择从器件时， \overline{SS} 引脚被其拉低到逻辑 0，这时若 MCU 将自身 SPI 设为主机，即置 $MSTR=1$ ，SPI 将会发现 \overline{SS} 已经为逻辑 0，表示已经存在主机，于是发生了工作方式冲突错误，SPI 自动置位 $MODF$ 标志，并在 $SPIE=1$ 时申请中断，通知 CPU 处理。

当 $DDS7=1$ 、 $SSOE=0$ 时， \overline{SS} 引脚为通用输出引脚，与 SPI 系统无关。

当 $DDS7=0$ 、 $SSOE=1$ 时， \overline{SS} 引脚为系统保留功能，用户不能使用。

(2) 串行数据线(MISO、MOSI)。MISO 为主入/从出线，MOSI 为主出 / 从入线，它们用于串行接收和发送数据；数据高低位传送顺序由 $LSBF$ 选择。每个 MOSI、MISO 引脚都是双向引脚，当 $SPICR$ 寄存器的 $MSTR$ 位由软件置 1 时，SPI 设置为主机方式，其 MISO、MOSI 分别是数据输入 / 输出线，这时主机从它的 MOSI 脚输出数据，从它的 MISO 引脚输入数据。当器件设置为从机方式时，其 MISO 变为数据输出线，而 MOSI 成为数据输入线。在多主机系统中，无论主从，所有 SCK 引脚、MISO 引脚和 MOSI 分别联在一起。时钟信号由主 SPI 的 SCK 引脚输出，送到所有的从 SPI 的 SCK 引脚，主 SPI 的数据从其 MOSI 引脚输出，送到所有从 SPI 的 MOSI 引脚；被选中的从 SPI 的数据从其 MISO 输出，送到主 SPI 的 MISO 引脚。

(3) 串行时钟(SCK)。在 SPI 设置为主机方式时， SCK 信号来自内部 MCU 时钟，并从其 SCK 引脚输出。在主机起动一次传送时，自动在 SCK 端产生 8 个 SCK 时钟脉冲。对于从机，只有其 \overline{SS} 引脚为逻辑低电平时，才接收 SCK 时钟信号，并与 SCK 同步发送或接收数据。无论是主 SPI 或从 SPI，都是在 SCK 信号的一个跳变沿进行数据移位，在数据稳定后的另一个跳变沿进行采样。主机设备的 $SPIBR$ 选择时钟频率，与从机无关。主机和从机的 SPI 数据与位时钟之间的传送定时关系必须相同(由 $SPICR1$ 的 $CPOL$ 和 $CPHA$ 位控制)。



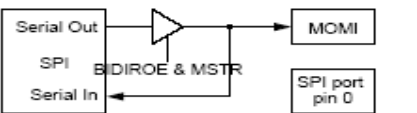

另外，数据方向寄存器的 $DDRS4$ - $DDRS6$ 分别影响 MISO、MOSI 和 SCK 。

若某一引脚作为输出时，应使 **DDRS** 对应位置 1，若作为输入，则不受 **DDRS** 位影响。

3. SPI 的双向模式(MOMI 或 SISO)

当控制寄存器 **SPOCR2** 中的 **SPC0=1** 时，SPI 进入一种双向模式，事实上 SPI 原来就是双向的，因此这里的双向模式确切地说是单线双向模式。在该模式下，SPI 使用单一引脚与外部器件接口，具体引脚由 **MSTR** 控制位决定，在主模式下 **MOSI** 成为双向引脚 **MOMI**，而在从模式下 **MISO** 成为双向引脚 **SISO**，引脚数据方向取决于相应的 **DDRS** 位。如表 1 所示。在使用双向模式的系统中，所有需要双向传输的器件均必须工作在双向模式下。在双向模式下，无论 SPI 作为主器件或从器件，均有一引脚重新获得通用 I/O 功能。

表 1 普通模式与双向模式

When SPE = 1	Master Mode MSTR = 1	Slave Mode MSTR = 0
Normal Mode SPC0 = 0		
Bidirectional Mode SPC0 = 1		

4. SPI 数据与位时钟的各种时序关系

两器件之间传送信息，它们的时序必须一致。为了能与各种标准外围设备接口，MCU 提供了四种数据和位时钟的时序关系，由 **CPOL** 和 **CPHA** 控制位选择。将 **CPOL** 置 1 相当于在时钟信号中串入一个反相器。除此以外，SPI 还可以选择数据位的传送顺序。时序相同时，将主机和从机的 **MISO**、**MOSI**、**SCK** 分别直接相联，即可构成一个主从系统。

图 2、图 3 分别表示了 **CPHA** 为 0、1 时，单个字节数据传送的时序。由图可见，每个字节需要 8 个时钟周期，传送期间从器件选中信号 \overline{SS} 必须有效。这里以 **CPOL=0**、**LSBF=0** 为例，主机发送数据为例，对两个时序图简单说明如下。

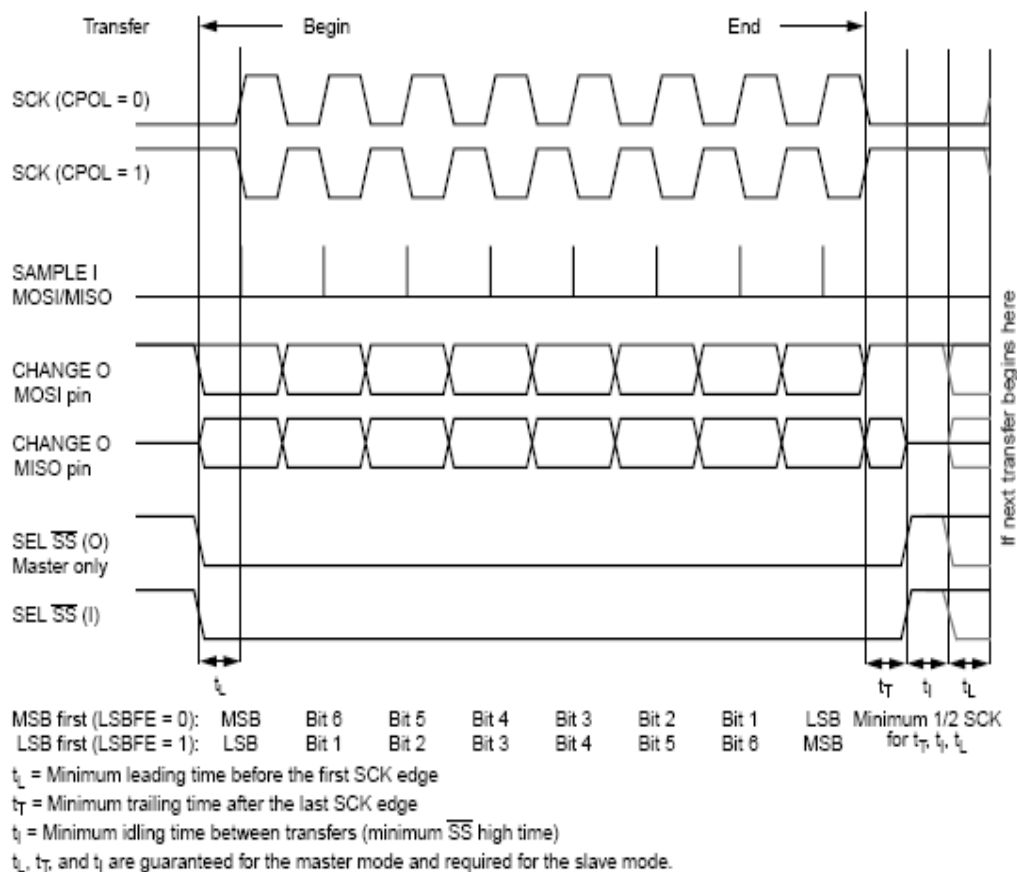


图 2 CHPA=0 时的数据位与时钟时间关系

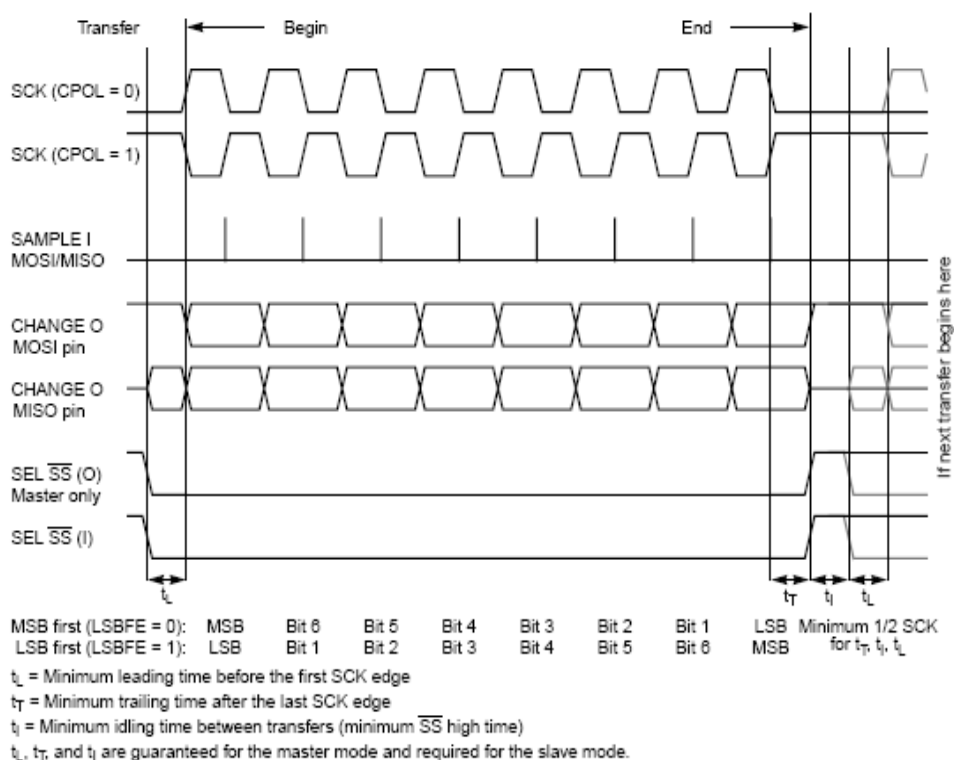


图 3 CHPA=1 时的数据位与时钟时间关系

在图 2 中, 主机的 \overline{SS} 输出逻辑 0, 选中从器件, 同时 MOSI 输出数据最高位, 经过半个时钟周期后, 在 SCK 的上升沿, 从器件采样 MOSI 线, 保存结果。在 SCK 的下降沿, 主机输出下一个数据位, 在下一个 SCK 的上升沿, 从器件又采样 MOSI 线, 保存结果, 如此循环八次, 数据传送结束, 然后 \overline{SS} 再延迟半个时钟周期后释放, 回到高电平, 等待半个周期后可以开始下一次传送过程。

在图 3 中, \overline{SS} 有效后, 时钟信号没有立即输出, 因此传送过程并没有真正开始, 而是半个周期后, 时钟和数据同时输出, 然后在时钟的下降沿从器件采样 MOSI 线, 保存结果, 其他与图 9-5 类似。

5. SPI 的工作过程

系统工作时, 主机发送数据的同时也接收从机来的数据, 从机接收的同时也向主机发送数据, 这个过程就好像分别位于主从器件的两个 8 位寄存器被“连接”在一起形成了一个分布式的 16 位寄存器。当进行数据传输时, 该寄存器在主器件时钟 SCK 控制下, 连续循环移动 8 位, 于是数据在主从器件之间实现了数据的有效交换。对于主器件来讲, 写入 SPIDR 寄存器的数据成为送往从器件的输出数据, 经传输操作后, 从 SPIDR 读出的数据则是来自从器件的输入数据。

当 SPI 控制寄存器 SPICR1 中的 SPE=1 时, SPI 系统使能。SPICR1 中的相位控制位 CPHA 和极性控制位 CPOL 用于决定当前的 SPI 系统使用的时钟格式。CPOL 只选择正向还是反向时钟, 而 CPHA 则通过选择是否将时钟移相 180 度, 来协调两种不同的协议。

在主机方式中, CPU 将数据写入 SPIDR 时(写入 SPIDR 的数据立即装入 8 位移位寄存器中), 起动一个传送过程。SCK 脚输出 8 个位传送时钟, MOSI 脚串行移位输出数据到从机 SPI。同时, 数据也由从 SPI 通过 MISO 脚串行移位输入到这个 8 位移位寄存器中。在传送的第 8 次移位后, 数据并行传送到 SPIDR, 此后, CPU 可读出它的内容。每完成一次数据传送, SPRF 状态位置位, 读 SPISR(SPRF 为 1 时), 访问 SPIDR, 将 SPIF 位清 0。

在从机方式中, CPU 可先向 SPIDR 写入需传送到主机的数据, 该数据从内部总线并行装入 8 位移位寄存器中。与主机不同, 该写入不起动传送, 从机需等待主机的信号。当 CPHA 为 0 时, \overline{SS} 脚的逻辑低电平, 使数据的最高位输出, 在 SCK 的第一个跳变沿采样输入, 在另一个跳变沿将数据的次高位输出, 当 CPHA=1 且 $\overline{SS}=0$ 时, SCK 引脚的第一个跳变沿启动从机移位, 将数据的最高位输出, 在下一个跳变沿采样输入, 这使从机与主机同步。这时, 来自主机的数据

由从机的 MOSI 脚串行输入，并移入 8 位移位寄存器，同时原来 8 位移位寄存器的数据由从机的 MISO 脚串行输出至主机。在 8 次数据移位后，接收的数据并行传送到 SPIDR 中，等待 CPU 读出。同主机一样，每完成一次数据的传送，SPRF 位置 1，这时读 SPSR 随后访问 SPIDR 将 SPRF 位清 0。

主机可发可收，从机可收可发。主机从机之间的数据传送是双向的。对于主机发、从机收的情况，主机由写入 SPIDR 起动一次发送过程。对于主机收、从机发的场合，也由主机写入 SPIDR 起动一次传送过程，不过写入 SPIDR 的数据与传送无关。

正在传送时，若写入 SPIDR，将引起写冲突错误。对于主机，将打断数据的传送，对从机若写入 SPIDR，对数据的传送无影响。

(二)波特率设置

P 时钟经过一组级联的分频器形成 SPI 时钟，分频系数由 SPIBR 寄存器中的 SPPR2~SPPR0 和 SPR2~SPR0 共六位控制。波特率分频因子的表达式为：

$$\text{BaudRateDivisor} = (SPPR + 1) * 2^{(SPR + 1)}$$

该波特率产生器必须满足两个条件才能激活：一是 SPI 为主器件，二是串行传输正在进行，否则处于关闭状态以降低功率消耗。详见波特率寄存器 SPIBR 的说明。

(三)中断管理

SPI 设置了两种中断，一是数据传送结束中断，另一个是模式冲突中断，二者受同一个中断允许位 SPIE 的控制，每次 SPIF 或 MODF 状态标志置位时，若 CCR 中的全局中断允许位 I=1，则向 CPU 发出硬件中断请求。读 SPOSR 寄存器、然后读或写 SPI 数据寄存器将清除标志位 SPIF，读 SPOSR 寄存器、然后写 SPI 数据寄存器将清除标志位 MODF。

第二节 SPI寄存器简介

2.1 SPI控制寄存器 1(SPICR1)

SPI 的工作方式主要由该寄存器设置，包括主从方式、单线双向模式选择，时钟及位顺序等，甚至包括引脚工作特性的设置，见表 2。其各位意义如下：

表 2 SPI 控制寄存器 1

寄存器名: SPICR1;地址: \$00D0;复位默认值: 00000100B;读操作: 任意;写操作:任意。								
位	B7	B6	B5	B4	B3	B2	B1	B0
读写	SPIE	SPE	SPTIE	MSTR	CPOL	CPHA	SSOE	LSBF

SPIE: SPI 中断允许位。

0: 禁止 SPI 中断。

1: 每次 SPRF 或 MODF 状态标志置位时发出硬件中断请求。

SPE: SPI 系统允许位。

当 MODF=1 时, SPE 读取结果总是 0, SPOCR1 的写操作指令必须作为模式故障恢复序列的一部分嵌入其中。

0: SPI 内部硬件完成初始化, 但 SPI 系统处于低功耗的禁止状态。

1: SPI 使能。

SPTIE:

0: SPTEF 中断不被允许。

1: SPTEF 中断允许。

MSTR: 主、从模式选择位, 用于设定本机 SPI 以主器件还是从器件身份出现。

0: 从模式。

1: 主模式。

CPOL、CPHA: SPI 时钟极性、相位选择位, 这两位用来指定 SPI 的时钟格式。当无传输动作且 CPOL=0 时, 主器件的 SCK 引脚处于低电平, 而如果 CPOL=1, SCK 则闲置在高电平。参看图 9-6、图 9-7。

SSOE: 从器件选中输出信号(\overline{SS})允许位, \overline{SS} 输出功能只有在主模式下通过置位 SSOE 和 DDRS7 实现。

0: 禁止 \overline{SS} 输出。

1: 允许 \overline{SS} 输出, 但同时 DDRS7 必须为 1。

LSBF: SPI 数据位传输顺序选择位, 通常要求 LSBF=0, 即传输过程高位在先。该位只决定传输过程中各位的先后顺序, 不影响数据位在寄存器中的顺序, 因此

读写操作正常进行，即高位(MSB)在第 7 位(BIT7)。对于仅由 MC9S12DP256 构成的互连系统，只要各个 SPI 的 LSBF 相同，对传输结果无影响，但在与外围设备或器件连接时，SPI 必须根据该设备或器件所要求的位顺序正确设置 LSBF。

0: 数据传输高位(MSB)在先。

1: 数据传输低位(LSB)在先。

2.2 SPI控制寄存器 2(SPICR2)

表 3 SPI 控制寄存器 2

寄存器名: SPICR2;地址: \$00D1;复位默认值: 00001000B;读操作: 任意;写操作: 任意。								
位	B7	B6	B5	B4	B3	B2	B1	B0
读写	未定义	未定义	未定义	MODFEN	BIDIROE	未定义	SPISWAI	SPC0

MODFEN: 模式错误使能位

0: 禁止模式错误标志位置位。

1: 允许模式错误标志位置位。

BIDIROE: 双向模式下输出缓冲使能位

0: 双向模式下禁止输出缓冲。

1: 双向模式下允许输出缓冲。

SPISWAI: 等待模式下 SPI 工作方式

0: 等待模式下停止 SPI 时钟。

1: 等待模式下 SPI 时钟正常工作。

SPC0: 串行引脚控制，该位与 MSTR 位一起决定串行引脚功能设置。见表 3

2.3 SPI波特率选择寄存器

该寄存器只有六个有效位 SPPR2~SPPR0 和 SPR2~SPR0，用来确定 SPI 系统工作时钟 SCK 的频率，即波特率。复位默认值为 0。参看表 9-11。

表 4 SPI 波特率选择寄存器

寄存器名: SPIBDR;地址: \$00D2;复位默认值为: 00000000\$;读操作: 任意;写操作: 任意。								
位	B7	B6	B5	B4	B3	B2	B1	B0
读写	未定义	SPPR2	SPPR1	SPPR0	未定义	SPR2	SPR1	SPR0

SPPR2-SPPR0:波特率预选位;

SPR2-SPR0:波特率选择位。

波特率分频因子表达式为：

$$\text{BaudRateDivisor} = (SPPR + 1) * 2^{(SPR + 1)}$$

2.4 SPI状态寄存器

该寄存器反映 SPI 的工作状态，其中包括传输结束、写冲突和模式故障三个标志位，程序可以检查各位的状态，也可以通过特定的寄存器访问序列将标志位清 0，见表 9—12。其各位意义如下：

表 5 SPI 状态寄存器

寄存器名：SPISR;地址：\$00D2;复位默认值：00100000B;读操作：任意;写操作：无意义。								
位	B7	B6	B5	B4	B3	B2	B1	B0
读	SPIF	未定义	SPTEF	MODF	未定义	未定义	未定义	未定义

SPIF： SPI 中断请求位，在数据传输过程中，SPRF 在第 8 个 SCK 周期后置位，通过读 SPISR 寄存器并随后读或写数据寄存器 SPIDR 清 0。

0：传输正在进行或者根本没有传输。

1：一次传输已经结束。

SPTEF： 当 SPIDR 中的值送入移位寄存器中是，该位置 1。同时如果 SPTIE 位为 1，则向 CPU 发出中断请求。读 SPISR 然后写 SPIDR 将清除该位。

0：SPI 数据寄存器非空。

1：SPI 数据寄存器空。

MODF： SPI 模式错误中断状态位。当 MSTR=1 时，如果从选择引脚 \overline{SS} 在外部被拉低成逻辑 0，该位由 SPI 硬件自动置 1。这时本机 SPI 已经不能成功设定为主机，显然这种情况在正常情况下是不允许的。当 DDRS7=1 时，PS7 是通用输出或 \overline{SS} 输出引脚，而不是专用于 SPI 系统的 \overline{SS} 输入引脚，在这种特殊情况下，模式故障功能被禁止，MODF 保持为 0。读 SPOSR 随后写入 SPOCR1 将清 0 该位。

0：无异常。

1：系统中已经出现另一个主机，并正在选中从器件。

2.5 SPI数据寄存器

表 6 SPI 数据寄存器

寄存器名: SPIDR;地址: \$00D5;复位默认值;读操作: 任意, 一般在读 SPIF 位后, 写操作: 任意, 读 SPTEF 后								
位	B7	B6	B5	B4	B3	B2	B1	B0
读写	D7	D6	D5	D4	D3	D2	D1	D0

该 8 位寄存器是 SPI 数据寄存器, 具有输入、输出双重功能, 见表 6。对该寄存器进行读操作时所访问的输入部分是双缓冲的, 但写操作则直接将数据送到串行移位器。注意: 某些简单的从器件可能只有发送功能, 只向主器件发送数据而不向主器件请求数据, 例如并行输入串行输出的 TTL 逻辑电路; 或者只有接收功能, 只从主器件接收而不返回数据, 例如串行输入并行输出的 TTL 逻辑电路。由于 SPI 的传输是一次交换过程, 对于只发送数据的从器件, 交换前主机可向 SPODR 写入任何数据; 对于只接收数据的从器件, 交换后主机 SPODR 中的数据无意义。

第三节 SPI应用实例

利用单片机与显示驱动芯片 MC14489 的 SPI 通讯实现计时器功能

(1) 实验设备: HCS12 编程器、开发板、MC14489 驱动芯片、数码管

(2) 软件程序设计: 串型外部通讯是单片机与外界设备联系的重要方式。本例子利用 HCS12 单片机与 MC14489 芯片构成的具有串型通讯功能的电路, 实现单片机与外部设备的串口通讯。并且利用程序的相应算法实现计时器的功能。

(3) 源程序

```
#include <hdef.h>          /* common defines and macros */
#include <mc9s12dp256.h>    /* derivative information */
#pragma LINK_INFO DERIVATIVE "mc9s12dp256b"
/*声明变量*/
int n1,n2,n3,n4,n5,j=0;
word t1,t2,time;
byte mode=0;
/*SPI 初始化*/
void SPI (void) {
    SPIOCR1=0b01010000;
    DDRS=0x60; //S5, S6 为输出, 其余为输入
    SPIOBR=0x12;
    SPIOCR2=0x00;
```

```

DDRT=0X01;//t0 为输出，其余为输入
PTT=0X01;//T0=1
}
/*键盘中断初始化*/
void PortInit(void){
    DDRJ=0x00;    //j 脚数据方向寄存器为输入
    PIEJ=0X03;    //j 脚中断允许寄存器
}
/*延时程序*/
void delay(word time){
    for(t2=0;t2<time;t2++){
        for(t1=0;t1<100;t1++){
        }
    }
}
/*将单字节数据写入 14489 的配置寄存器*/
void DanZiJie(void) {
    PTT=0X00;
    delay(1);
    SPI0DR=0b00000001;
    while(!SPI0SR_SPTEF){;}
    delay(3);
    PTT=0X01;
}
/*发三字节数据*/
void SanZiJie (void) {
    PTT=0X00;
    delay(1);
    SPI0DR=0x80+n1;
    while(!SPI0SR_SPTEF){;}
    SPI0DR=0x00+n2*16+n3;
    while(!SPI0SR_SPTEF){;}
    SPI0DR=0x00+n4*16+n5;
    while(!SPI0SR_SPTEF){;}
    delay(3);
    PTT=0X01;
}
/*主程序*/
void main(void) {
    SPI();
    DanZiJie();
    PortInit();
    EnableInterrupts;
    while(1){

```

```
n1=j/10000;
n2=j/1000%10;
n3=j/100%10;
n4=j/10%10;
n5=j%10;
j++;
SanZiJie();
delay(15600);
}
}
/*键盘中断子程序*/
#pragma CODE_SEG __NEAR_SEG NON_BANKED
interrupt void PortJISR(void){
    switch(PIFJ){    //J 脚中断标志寄存器
        case 0x01: mode=0;  PIFJ_PIFJ0=1;
            for (;;) {
            }
            break;
        }
    }
}
```

(4)调试结果

程序运行，计时器开始工作，本计时器的工作范围（1s—99999s）。按下 Key1 计时停止，记录时间，按下 Reset 键，计时重新开始。

第五章 A/D转换模块

第一节 A/D模块介绍

1.1 A/D转换原理

模拟信号依次通过抽样和保持（S/H）电路和模拟转换器（A/D）后转换为数字格式。

抽样和保持电路以均匀间隔对模拟信号进行抽样，并且在每个抽样运算后在足够的时间内保持抽样值恒定，以保证输出值可以被 A/D 转换器精确转换。

下一步是通过模数转换器将抽样和保持电路的输出转换为数字形式。模数转换器的输出通常表示为二进制编码的形式。

转换精度由分辨率来表示，它由离散级数量决定。比如，对一个以二进制形式编码的长度为 N 位的长的输出，有效地离散级数量是 2 的 N 次方，分辨率为离散数量级的倒数。

1.2 A/D转换原理的应用前景

A/D 转换器是模拟信号源与计算机或其他数字系统之间联系的桥梁，它的任务是将连续变化的模拟信号转换为数字信号，以便计算机或数字系统进行处理、存储、控制和显示。在工业控制和数据采集及许多其他领域中，A/D 转换器是不可缺少的重要组成部分。

1.3 A/D转换模块

12 中 A/D 转换共有两个方块，每个方块各有 8 个输入通道，使用时应以标头 ATD0 或 ATD1 标识。

1.4 功能结构图

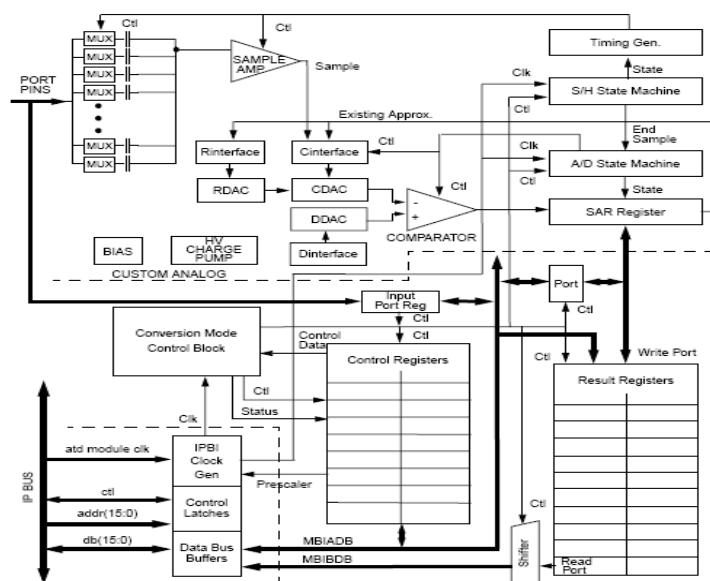


图 2.1 功能结构图

图中所示的是 A/D 模块的功能结构，这个功能模块被虚线划分成为图示所示的虚线所隔离的三个部分：IP 总线接口、转换模式控制/寄存器列表，自定义模拟量。

IP 总线接口负责该模块与总线的连接，实现 A/D 模块和通用 I/O 的目的，还起到分频的作用；

转换模式控制寄存器列表中有控制该模块的所有的寄存器，执行左右对齐运行和连续扫描。

自定义模拟量负责实现模拟量到数字量的转换。包括了执行一次简单转换所需的模拟量和数字量。

1.5 HCS12A/D特点

8/10 位精度；7 us, 10-位单次转换时间.；采样缓冲放大器；可编程采样时间；左/右对齐，有符号/无符号结果数据；外部触发控制；转换完成中断；模拟输入 8 通道复用；模拟/数字输入引脚复用；1 到 8 转换序列长度；连续转换模式；多通道扫描方式。

ATD 模块有模拟量前端、模拟量转换、控制部分及结果存储等四部分组成。其中模拟前端包括多路转换开关、采样缓冲器、放大器等，结果存储部分主要有 8 个 16 位的存储器和反映工作状态的若干标志位。

第二节 A/D寄存器简介

要完成 ATD 转换的特定功能，必须对相关寄存器有所了解。在 12ATD 模块中，ATDCTL2,3,4,5 为常用的控制寄存器，ATDSTAT0, 1 为常用的两个状态控制器，ATDDR0-7 为八个结果寄存器。

ATD 工作时，由 CPU 发出启动命令，然后经采样、模数转换，最后将结果保存到相应的寄存器。

ATD 每次启动要进行若干个扫描循环，每个扫描循环称为一个转换序列，每个转换序列能包含 1-8 最多 8 次转换，由控制寄存器 ATDCTL3 中的 S8C/S4C/S2C/S1C 等位来决定。

这些转换序列可以针对某个单一通道，也可以针对几个相邻通道，每个通道可以使外部模拟输入，也可以是参考电压或其他保留信号，ATD 可支持多种不同的通道信号组合。每次转换包括哪些通道由 ATDCTL5 中的 CC、CB、CA 决定的。

对单一通道连续进行多次转换有利于实现滤波，一次转换多个通道则可以通过一次启动命令快速浏览多个信号，中间无需 CPU 干涉，节约了 CPU 时间。

ATD 的运行方式分为单次和连续运行两种。

在单次方式下，每个转换序列完成后，寄存器 ATDSTAT 中的 SCF 置位，然后 ATD 模块暂停。

在连续方式下，转换以转换序列为单位连续进行，当第一个转换序列完成后，SCF 置位，同时 ATD 模块开始下一个转换序列。

在上述两种方式下，每个通道的转换结果进入到对应结果寄存器后，寄存器 ATDSTAT1 种对应的 CCF 位置 1，对存放转换结果的寄存器进行读操作后，CCF 位将自动清 0。转换过程的启动是通过写入寄存器 5ATDCTL5 实现的。

ATD 转换所需要的时钟周期数是固定不变的，但是采样时间和时钟频率可以通过 ATDCTL4 在一定范围内选择，因此转换时间也可以选择。

2.1 控制寄存器 2（ATDCTL2）

	Bit 15	14	13	12	11	10	9	Bit 8
	ADPU	AFFC	AWAI	ETRIGLE	ETRIGP	ETRIGE	ASCIE	ASCIF
Reset:	0	0	0	0	0	0	0	0

ADPU - A/D 电源使能/禁止

1 = A/D 模块上电

0 = 禁止 A/D，以减少功耗

AFFC - A/D 快速转换完成标志位清零

1 = 快速标志位清零顺序 每次读取结果寄存器自动清零

0 = 正常标志位清零顺序 需要手动对状态标志位清零

AWAI - A/D 等待模式

1 = 等待模式下，转换

0 = 等待模式下，禁止转换

ASCIE - A/D 顺序完成中断使能

ETRIGLE	ETRIGP	ETRIGE	SCAN	描述
x	x	0	0	忽略外部触发，执行一次转换后停止
x	x	0	1	忽略外部触发，执行连续转换后
0	0	1	X	下降沿触发，每次触发，执行一次转换
0	1	1	X	上升沿触发，每次触发，执行一次转换
1	0	1	X	低电平触发，每次触发，执行连续转换
1	1	1	X	高电平触发，每次触发，执行连续转换

2.2 控制寄存器 3 (ATDCTL3)

	Bit 7	6	5	4	3	2	1	Bit 0
	0	S8C	S4C	S2C	S1C	FIFO	FRZ1	FRZ0
Reset:	0	0	1	0	0	0	0	0

转换序列长度

S8C	S4C	S2C	S1C	Number of Conversions per Sequence
0	0	0	0	8
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	X	X	X	8

FIFO – 结果寄存器 FIFO

1 = 结果寄存器映射到转换序列

0 = 结果寄存器没有映射到转换序列

FRZ	Response
00	Ignore IFREEZE
01	Reserved 预留保留
10	Finish conversion, then freeze
11	Freeze Immediately

2.3 控制寄存器 4 (ATDCTL4)

Address Offset: \$0004

	Bit 15	14	13	12	11	10	9	Bit 8
	SRES8	SMP1	SMP0	PRS4	PRS3	PRS2	PRS1	PRS0
Reset:	0	0	0	0	0	1	0	1

Handwritten notes: A bracket under bits 15-12 is labeled "10位". A bracket under bits 15-8 is labeled "8位".

SRES8 - A/D 精度选择

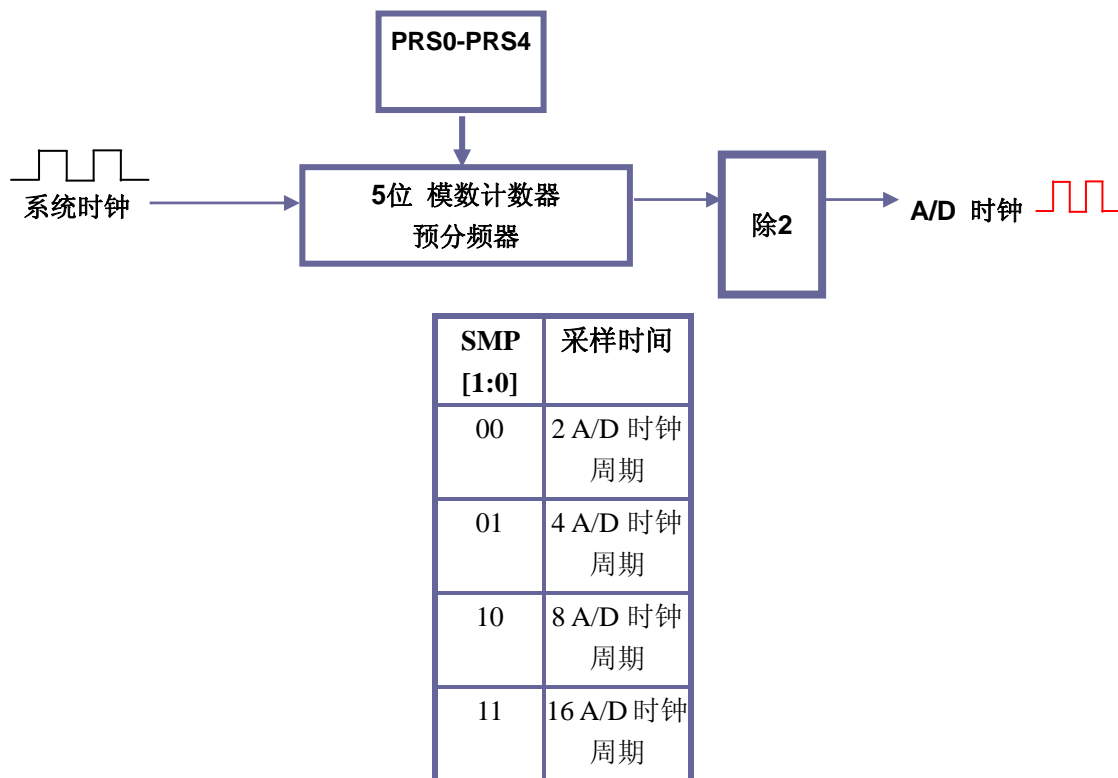
1 = 8 位

~~0 = 10 位~~

5 位 模数计数器预分频器

- 由 A/D 控制寄存器中的 PRS[4:0]控制
- 分频系数从 2 到 64
- 如果 PRS[4:0] = 0, 预分频不起作用

注: 设置 PRS[4:0]时, A/D Clock 不能大于 2 MHz.



转换时间计算:

转换时间计算举例: (假设 2MHZ A/D 时钟频率)

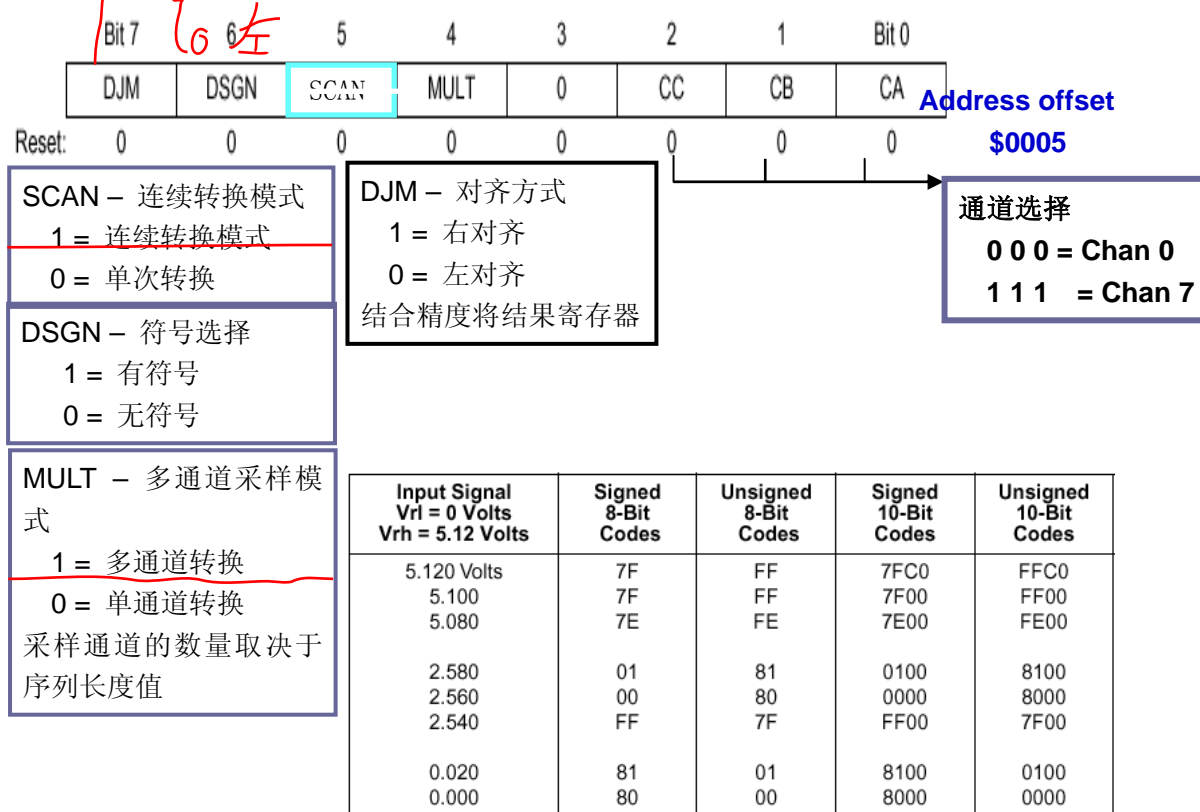
例 1:

$$\begin{aligned}
 \text{转换时间} &= \text{Initial Sample Time} + \text{Programmed Sample Time} + \text{Resolution Period} \\
 &= 2 + 2 + 10 = 14 \text{ A/D Clocks} \\
 &= 7\mu\text{Sec}
 \end{aligned}$$

例 2:

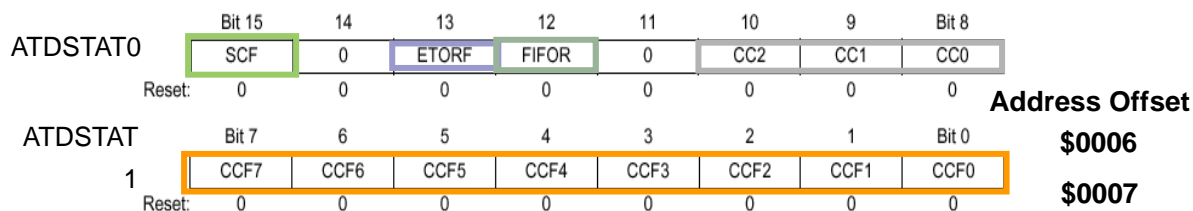
$$\begin{aligned}
 \text{转换时间} &= \text{Initial Sample Time} + \text{Programmed Sample Time} + \text{Resolution Period} \\
 &= 2 + 16 + 10 = 28 \text{ A/D Clocks}
 \end{aligned}$$

2.4 控制寄存器 5 (ATDCTL5)



注意：对这个寄存器写操作时，将会中断当前的转换，然后重新启动新的转换序列

(5) 状态寄存器



SCF – 转换序列完成标志

- 在单次转换模式时，当转换完成后置位 (SCAN = 0)

在连续转换模式时，当第一次转换完成后置位 (SCAN = 1).

当 (AFFC = 0) , 写 1 清零.

ETORF - 外部触发覆盖标志

- 如果在转换过程中高/低电平出现, 置位 FIFOR
- 当结果寄存器在读出之前已经被写入时, 置位 (CCF 没有清零)

CC[2:0]转换计数器

3-位计数器指向下一个将要转换的通道

CCF7 - CCF0 - 独立通道转换完成标志位

每个相应的通道转换结束后置位, 当相应的 A/D 结果寄存器被读出时, 清零, 注意当 AFFC 位不同时情况

第三节 A/D应用示例

3.1 编程步骤

要让 ATD 开始转换工作, 必须经过以下三个步骤:

1. 将 ADPU 置 1, 使 ATD 启动;

2. 按照要求对转换为数、扫描方式、采样时间、时钟频率及标志检查等方式进行设置;

3. 发出启动命令;

如果上电默认状态即能满足工作要求, 那么只要将 ADPU 置 1, 然后通过控制寄存器发出转换命令, 即可实现转换。

3.2 A/D程序示例—单通道查询

```
#include <hidef.h>          /* common defines and macros */
#include <mc9s12dp256.h>      /* derivative information */
#pragma LINK_INFO DERIVATIVE "mc9s12dp256b"
int ADValue;
void AD_Init(void) //初始化
{
    ATD0CTL2=0xc0; //定义寄存器 2 快速清 0
    ATD0CTL3=0x20; //定义寄存器 3 4个
    ATD0CTL4=0xc3; //定义寄存器 4 8个时钟 4u
    ATD0CTL5=0xa6; //定义寄存器 5
```

```

ATD0DIEN=0x00; // 禁止数字输入
}
int AD_GetValue(void) //读取 AD 转换结果
{
    ADValue = ATD0DR0; //读结果寄存器
}
void main(void)
{
    AD_Init(); //AD 初始化
    DDRB = 0xFF; //portB 均为输出通道
    PORTB = 0x00;
    for(;;)
    {
        while(!(ATD0STAT1&0x01)); //等待转换结束
        AD_GetValue(); //读取转换结果
        PORTB = (int)ADValue; //在 B 口显示转换值
    }
}

```

3.3 A/D程序示例—滤波

12 中 A/D 系统启动一次 A/D 转换,能对同一模拟量输入连续执行多次转换。利用此功能,使用下面的程序,可在一次采样中执行四取二数字滤波,从而可滤除干扰,得到精度较高的 A/D 转换结果。

```

#include <hidef.h> /* common defines and macros */
#include <mc9s12dp256.h> /* derivative information */
#pragma LINK_INFO DERIVATIVE "mc9s12dp256b"
static long ADValue0, ADValue1, ADValue2, ADValue3;
long a,b,c,d;
long m,n,sum;
long max();
long min();
void AD_Init(void) //初始化
{
    ATD0CTL2=0xc0; //定义寄存器 2
    ATD0CTL3=0x20; //定义寄存器 3
    ATD0CTL4=0xc3; //定义寄存器 4
    ATD0CTL5=0x86; //定义寄存器 5
    ATD0DIEN=0x00; // 禁止数字输入
}
void AD_GetValue0(void) //读取 AD 转换结果
{
    ADValue0 = ATD0DR0; //读结果寄存器 0
}

```

```

    a=ADValue0;
}
void AD_GetValue1(void) //读取 AD 转换结果
{
    ADValue1 = ATD0DR1; //读结果寄存器 1
    b=ADValue1;
}
void AD_GetValue2(void) //读取 AD 转换结果
{
    ADValue2 = ATD0DR2; //读结果寄存器 2
    c=ADValue2;
}
void AD_GetValue3(void) //读取 AD 转换结果
{
    ADValue3 = ATD0DR3; //读结果寄存器 3
    d=ADValue3;
}
void main(void)
{
    AD_Init(); //AD 初始化
    DDRB = 0xFF;
    PORTB = 0x00;
    for(;;)
    {
        while(!(ATD0STAT1&0x01)){;} //等待转换结束
        AD_GetValue0(); //读取转换结果
        while(!(ATD0STAT1&0x02)){;} //等待转换结束
        AD_GetValue1(); //读取转换结果
        while(!(ATD0STAT1&0x04)){;} //等待转换结束
        AD_GetValue2(); //读取转换结果
        while(!(ATD0STAT1&0x08)){;} //等待转换结束
        AD_GetValue3(); //读取转换结果
        m=max(a,b,c,d);
        n=min(a,b,c,d);
        sum=(a+b+c+d-m-n)/2;
    }
}

```

3.4 A/D程序示例—定时采样

在实际应用中，经常需每隔一段时间对某个模拟量进行一次采样。为实现定时 A/D 采样，可使用 12 中的输出比较功能来控制定时，其定时时间间隔可设定。在发生定时器输出比较中断时，进行启动 A/D 转换。然后等待 A/D 转换完成，

读入数据。为了不浪费 CPU 时间去等待转换的完成，可使用中断方式。

本程序有两部分组成：一为初始化和启动程序，另一位定时器输出比较中断处理程序。CCF 为标志字节，它的最高位为 1，表示完成一次 A/D 转换，由程序可读入的值，并应清零该位。TOF 为定时器控制标志位，=0 表示为处于启动 A/D 状态；=1 表示为等待定时时间到，以便下次采样。

```

#include <hidef.h>          /* common defines and macros */
#include <mc9s12dp256.h>     /* derivative information */
#pragma LINK_INFO DERIVATIVE "mc9s12dp256b"

int ADValue0;
unsigned int i=0;
int counter=0;
void AD_Init(void) //初始化
{
    ATD0CTL2=0xc0; //定义寄存器 2
    ATD0CTL3=0x08; //定义寄存器 3
    ATD0CTL4=0xc3; //定义寄存器 4
    ATD0CTL5=0xa0; //定义寄存器 5
    ATD0DIEN=0x00; // 禁止数字输入
}
void AD_GetValue0(void) //读取 AD 转换结果
{
    ADValue0 = ATD0DR0; //读结果寄存器 0
}
void main(void)
{
    TSCR2_PR   = 7; //prescale factor is 8, bus clock/128=8Mhz/8
    TSCR2_TOI  = 1; //timer overflow interrupt enable
    TSCR1_TEN  = 1; //timer enable
    AD_Init(); //AD 初始化
    DDRB = 0xFF;
    PORTB = 0x00;
    EnableInterrupts;
    for(;;) {
    }
}
#pragma CODE_SEG NON_BANKED
#pragma TRAP_PROC
void Int_TimerOverFlow(void)
{
    {
        counter++;
        if(counter==10)

```



```
{
    while(!(ATD0STAT1&0x01)){;}    //等待转换结束
    AD_GetValue0();                //读取转换结果
    PORTB = (int)ADValue0;          //在 B 口显示转换值
    for(i=0;i<=1000;i++);    //delay
    counter=0;
}
}
TFLG2_TOF  = 1;  //clear timer overflow flag
}
#pragma CODE_SEG DEFAULT
```

第六章 EEPROM模块

MC9S12DP256B 内部同时集成了两种非易失存储器: FLASH 和 EEPROM, 前者用于存储程序, 后者可用来保存组态、设置等关键数据, 它们均为防止误操作而设置了各种保护措施。本章主要介绍 EEPROM 的功能、特点及使用, 包括擦除与写入步骤。

第一节 EEPROM模块介绍

1.1 EEPROM功能

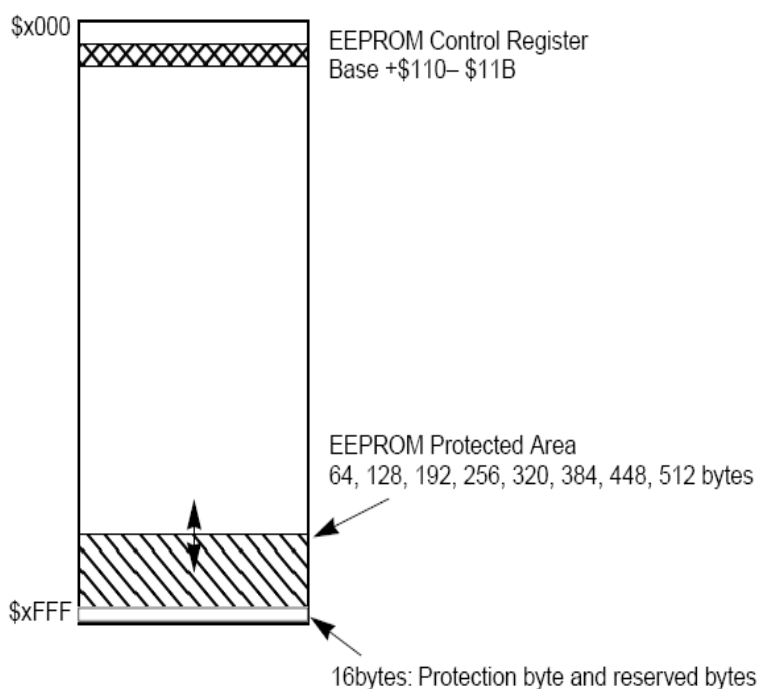
MC9S12DP256 内部集成了 4KB 的 EEPROM 存储器, 具有单块和整块擦除、编程、灵活保护和安全功能、快速区域擦除和字编程模式特点, 规范字访问可在单总线周期内完成。EEPROM 是一种非易失性存储介质, 具有稳定、保密性好的特点, 在系统掉电后, EEPROM 中的内容仍能可靠保持不变, 可以用来保存一些短时间不变的内容(如环境参数、产品序列等)。EEPROM 可以同 RAM 一样地读, 但要向 EEPROM 写入需要一定的时序和花费比 RAM 多的时间, 即需要一段特定的程序来写 EEPROM。

1.2 EEPROM结构

\$0000~\$0FFF 是 4KB 为 EEPROM 的存储空间, 复位时默认的 EEPROM 地址是从 0 开始的, 而单片机各 I/O 寄存器的地址已经占用了从 0 地址开始的 1KB 空间, 故 EEPROM 中开始的 1KB 空间(\$0000---\$03FF)看不到了, 所以实际上用户可以访问的是 \$0400---\$0FFF 的 3KB 的存储空间。

EEPROM 设有保护机制, 用户可以通过编程 \$0FFD, 在普通模式下设定需要的保护空间, 保护空间可在 64B、128B、192B、256B、320B、384B、448B、512B 中进行选择, 相应的保护区在 \$0XXX---\$0FFFF。由于设定保护地址所访问的地址也在所保护的区域内, 所以一旦用户在普通模式下对 \$0FFD 的内容进行了编程, 就无法再在此模式下修改保护区的内容, 从而也就达到了写保护或禁止局部擦除的效果。要改变保护区的设定, 需要单片机工作在“特殊模式”下, 访问 EPROT 寄存器, 而一旦在特殊模式下取消了保密属性, 该区的内容也就被擦除

了。



1.3 EEPROM特点

- ①单电源供电擦写
- ②自动编程和擦除算法
- ③命令执行完可产生中断
- ④快速扇区擦除和字编程操作
- ⑤灵活的保护机制，避免意外编程和擦除

第二节 EEPROM寄存器简介

2.1 时钟分频寄存器ECLKDIV

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	EDIVLD	PRDIV8	EDIV5	EDIV4	EDIV3	EDIV2	EDIV1	EDIV0
Write:								
Reset:	0	0	0	0	0	0	0	0

 = Reserved or unimplemented

EDIVLD: EEPROM 时钟分频加载。ECLKDIV 寄存器被写入时此位置“1”;

若此位置“0”，FCLKDIV 寄存器从最后复位后没有写入。在没有写入此寄存器之前，擦写和编程 eeprom 将引起进入错误和命令不被执行。

1: 最后复位后寄存器写入。

0: 寄存器没有写入。

PRDIV8: 8 分频控制。

0 - 晶振时钟为 EEPROM 时钟分配器的时钟源

1 - 晶振时钟除以 8 后，为 EEPROM 时钟分配器的时钟源

EDIV[5:0] 时钟分配器位

PRDIV8 和 EDIV[5: 0]联合用于将外部晶振频率分频为 150~200KHZ。


注意：在对 EEPROM 进行操作之前，必须对时钟进行初始化

这个频率用于驱动 EEPROM 擦写和编程。外部晶振频率大于 12.8MHZ，PRDIV8 置 1 位，CLK=OSCCLK/8。反之，PRDIV8 置零，CLK=OSCCLK。

$ECK = CLK / (EDIV[5: 0] + 1)$ 。

2.2 配置寄存器ECNFG

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	CBEIE	CCIE	0	0	0	0	0	0
Write:								
Reset:	0	0	0	0	0	0	0	0

 = Reserved or unimplemented

CBEIE: 命令缓冲区空中断允许位，

0: 命令缓冲区空中断关闭。


1: 命令缓冲区空中断允许。

CCIE — 指令完成中断使能，当所有指令被完成该位产生中断

0: 命令完成中断关闭。

2.3 保护寄存器EPROT

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	EPOPEN	F	F	F	EPDIS	EP2	EP1	EP0
Write:								
Reset:	F	F	F	F	F	F	F	F

 = Reserved or unimplemented

EPOPEN: 开启 EEPROM 编程/擦写

1: 全部 EEPROM 或是部分 EEPROM 可以擦写;

0: 所有 EEPROM 块保护。

EPDIS: EEPROM 部分保护位

1: 不作部分保护 EEPROM。

0: EEPROM 部分保护。


EP[2:0]——EEPROM 保护地址范围

Table 37 Address Range Protection

EP[2:0]	Protected Address Range	Protected Size
000	\$_FC0 - \$_FFF	64 bytes
001	\$_F80 - \$_FFF	128 bytes
010	\$_F40 - \$_FFF	192 bytes
011	\$_F00 - \$_FFF	256 bytes
100	\$_EC0 - \$_FFF	320 bytes
101	\$_E80 - \$_FFF	384 bytes
110	\$_E40 - \$_FFF	448 bytes
111	\$_E00 - \$_FFF	512 bytes

2.4 状态寄存器 ESTAT

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	CBEIF	CCIF	PVIOL	ACCERR	0	BLANK	0	0
Write:								
Reset:	1	1	0	0	0	0	0	0

 = Reserved or unimplemented

CBEIF: 命令缓冲区空中断标志, 表示地址、数据命令缓冲去空, 新的命令序列开始, 标志写“1”清零, 通过标志清零, 发出命令序列。写“0”将取消没

有发出的命令序列，并 ACCERR 标志置“1”。

1: 缓冲准备好接受新的命令。

0: 缓冲器满

CCIF— 命令完成中断标志，表明不再有未决的指令。该标志通过硬件写入来清除，写操作不起作用。

1: 所有命令完成。

0: 命令正在进行。

PVIOL: 保护冲突标志。表示在保护地址空间内进行了擦写或编程操作，在此标志置位同时，后续的擦写或编程操作无法进行。标志写“1”清零，写“0”无影响。在 **CBEIF** 清零后或者当 **CCIF** 清零时，写入新的有效命令可以使保护错误标志清零。

1: 保护冲突已发生。

0: 无故障。

BLANK — 空白校验标志

表明在响应一条擦除校验指令后 **EEPROM** 块已被完全地擦除

该标志位通过写“1”被清除。

1 = **EEPROM** 被完全擦除。

0 = **EEPROM** 未被完全擦除

ACCERR— 存取错误标志

表明在编程或擦除时序中有错误。该标志位通过写“1”来清除。

1 = 存取错误已出现

0 = 指令时序或执行被成功完成

ACCERR: 进入错误标志。表示非法进入 **EEPROM** 块或者与定义命令序列冲突。以下命令序列会引起 **ACCERR** 置位:

1 在初始化 **ECLKDIV** 之前，向 **EEPROM** 地址写值;


2 向 **EEPROM** 地址写入错误的字或字节;

3 在 **CBEIF** 未置 1 时，向 **EEPROM** 地址写值;

- 4 在执行前一个编程或擦除命令之前，向 EEPROM 地址写入第二个字
- 5 在向 EEPROM 地址空间写入字后，接着写其它 EEPROM 寄存器，没有写 ECMD 寄存器；
- 6 在执行前一个写好的命令之前，向 ECMD 寄存器写入第二个命令；
- 7 在对有保护的 EEPROM 空间，向 ECMD 寄存器写入整块擦除命令；
- 8 在对有保护的 EEPROM 空间，向 ECMD 寄存器写入部分擦除命令；
- 9 在写完命令寄存器后，接着写其它 EEPROM 寄存器，而没有写 ESTAT 寄存器；
- 10 在命令行列已开始或擦/写命令已执行的同时，从 EEPROM 块读取。这样读到无效数据。

2.5 命令寄存器ECMD

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	0			0	0		0	
Write:		ERASE	PROG			ERVER		MASS
Reset:	0	0	0	0	0	0	0	0

 = Reserved or unimplemented

ERASE: EEPROM 擦除

0: 无意义

1: 在 MASS=0 时，区域擦除；在 MASS=1 时，区域擦除；

PROG: EEPROM 编程

0: 无意义。

1: EEPROM 编程。

ERVER: 擦除校验：

0: 无意义。

1: 区域擦除后擦除校验。

MASS: 全局铲除

0: 部分擦除

1: 全部擦除

注意：只有在 ERASE 和 MASS 同时置 1 时，才能实现全部擦除。

有关寄存器 ECMD 的命令字的设置：

Table 39 Valid Commands

Command	Meaning	Remarks
\$20	Memory Program	Program 1 word (2 bytes)
\$40	Sector Erase	Erase 4 bytes
\$41	Mass Erase	Erase all 4k bytes
\$05	Erase Verify	Verify all 4k bytes are erased
other	Illegal	Generate an access error

第三节 EEPROM应用实例

3.1 EEPROM的写入操作

EEPROM 的写入操作可以按照以下步骤进行：

1. ECLKDIV 寄存器初始化，确定分频因子
2. 检查 EEPROM 时钟分频寄存器(ECLKDIV_EDIVLD)是否已经设置
3. 检查命令缓冲区(ESTAT_CBEIF)是否为空
4. 对读写错误(ESTAT_ACCER)标志位清零(写 1 清零)
5. 对保护区编程错误(ESTAT_PVIOL)标志位清零(写 1 清零)
6. 检查保护寄存器(EPROT_EPOPEN)是否允许编程/写
7. 将编程的数据赋值给要编程的 EEPROM 地址
8. 对 EEPROM 命令寄存器(ECMD)写入编程命令(0x20)
9. 对命令缓冲区空标志位(ESTAT_CBEIF)清零(写 1 清零)
10. 等待，直到命令完成标志位(ESTAT_CCIF)置位

3.2 EEPROM的擦除操作

1. ECLKDIV 寄存器初始化，确定分频因子
2. 检查 EEPROM 时钟分频寄存器(ECLKDIV_EDIVLD)是否已经设置

- 3.检查命令缓冲区(ESTAT_CBEIF)是否为空
- 4.对读写错误(ESTAT_ACCER)标志位清零(写 1 清零)
- 5.对保护区编程错误(ESTAT_PVIOL)标志位清零(写 1 清零)
- 6.检查保护寄存器(EPROT_EPOPEN)是否允许编程/写
7. 在要擦除的 EEPROM 的段地址内写入任意数据
8. 对 EEPROM 命令寄存器(ECMD)写入段擦除命令(0x40)
9. 对命令缓冲区空标志位(ESTAT_CBEIF)清零(写 1 清零)
10. 等待, 直到命令完成标志位(ESTAT_CCIF)置位

3.3 EEPROM示例程序

```

#include <hidef.h>          /* common defines and macros */
#include <mc9s12dp256.h>    /* derivative information */
#pragma LINK_INFO DERIVATIVE "mc9s12dp256b"
#define portb (*((volatile unsigned char*)(0x0001)))
#define ddrb  (*((volatile unsigned char*)(0x0003)))
#define eckdiv  (*((volatile unsigned char*)(0x0110)))
#define ecnfg  (*((volatile unsigned char*)(0x0113)))
#define eprot  (*((volatile unsigned char*)(0x0114)))
#define estat  (*((volatile unsigned char*)(0x0115)))
#define ecmd   (*((volatile unsigned char*)(0x0116)))
#define any    (*((volatile unsigned char*)(erase)))
#define protection  (*((volatile unsigned char*)(0xffd)))
unsigned char *erase;
unsigned char *eeaddr ;
unsigned char i;
unsigned int j=0;
byte StateFlag=1,mode=0;
void PortInit(void)
{
    DDRB=0XFF;
    DDRJ=0x00;
    PIEJ=0X03;
}
void program(void)
{
    eeaddr=(volatile unsigned char*)(0x800);
    if (StateFlag==1)
    {

```

```

while(eeaddr<=(volatile unsigned char*)(0x810))
{
    while(!(eclkdiv&0x80))
    {}
    while(!(estat&0x80))
    {}
    while(!(eprot&0x80))
    {}
    while (!SCI0SR1_RDRF) ;
    SCI_Receive(&i);
    *eeaddr=i;
    SCI_Transmit(*eeaddr);
    *eeaddr++ ;
    ecmd=0x20;
    estat|=0x80;
    estat=0x30;
    while(!(estat&0x40))
    {}
    while(!(estat&0x80))
    {}
}
portb=0x0f;
}

void eraseblock(void)
{
    erase=(volatile unsigned char*)(0x800);
    if (StateFlag==1)
    {
        eclkdiv=0x4a;
        if(j==0) //通过串口判断是部分擦除,还是全部擦除.
        {
            while (erase<(volatile unsigned char*)(0x810))
            {
                while(!(eclkdiv&0x80)) {}
                while(!(estat&0x80)) {}
                estat=0x30;
                while(!(eprot&0x80)) {}
                any=0xff;
                ecmd=0x40;
                estat|=0x80;
                estat=0x30;
                while(!(estat&0x40)) {}
                erase+=4;
            }
        }
    }
}

```

```

        while(!(estat&0x80)) {}

    }
    portb=0x03;
}
if(j==1) //通过串口判断是部分擦除,还是全部擦除.
{
    while(!(eclkdiv&0x80)) {}
    while(!(estat&0x80)) {}
    estat=0X30;
    while(!(eprot&0x80)) {}
    any=0x00;
    ecmd=0x41;
    estat|=0x80;
    while(!(estat&0x40)) {}
    portb=0x0c;
}
}
}
void protect(void)
{
    if (StateFlag==1)
    {
        protection=64;
    }
}
void EraseVerify(void)
{
    while(!(eclkdiv&0x80)) {}
    while(!(estat&0x80)) {}
    ESTAT= 0x30;
    any = 0x00;
    FCMD= 0x05;
    ESTAT= 0x80;
    while (ESTAT_CCIF == 0);
    if(ESTAT_BLANK==0)
    {
        portb=0X01;
    }
    else
    {
        portb=0x04;
    }
}
}

```

```

void main(void)
{
    SCI_Init();
    EnableInterrupts;
    PortInit();
    for(;;)
    {
        switch(mode)
        {
            case 0 :
            {
                StateFlag=1;
                eraseblock();
                EraseVerify();
                program();
                break;
            }
            case 1 :
            {
                StateFlag=1;
                protect();
                break;
            }
        }
    }
}

#pragma CODE_SEG __NEAR_SEG NON_BANKED
interrupt void PortJISR(void){
    switch(PIFJ){
        case 0x01: mode=0;  PIFJ_PIFJ0=1; break;
        case 0x02: mode=1;  PIFJ_PIFJ1=1; break;
    }
    StateFlag=0;
}

//初始化
int SCI_Init(void) {
    SCI0BDL=0x34;
    SCI0CR2=0X2C;
}

//发送子函数
int SCI_Transmit(byte data){
    while(!SCI0SR1_TDRE) ;
    SCI0DRL=data;
}

```

```
//接收子函数  
int SCI_Receive(byte *data){  
    *data=SCI0DRL;  
}
```

第七章 FLASH模块

MC9S12DP256B 内部同时集成了两种非易失存储器: FLASH 和 EEPROM, 前者用于存储程序, 后者可用来保存组态、设置等关键数据, 它们均为防止误操作而设置了各种保护措施。本章主要介绍 flash 的功能、特点及使用, 包括擦除与写入步骤。

第一节 FLASH模块介绍

1.1 FLASH功能

MC9S12DP256 内部集成了 256KB 的 FLASH 存储器, 具有单块和整块擦除、编程、灵活保护和功能、快速区域擦除和字编程模式特点, 规范字访问可在单总线周期内完成。可以存放整个程序或者需要高速访问或经常运行的程序代码, 如操作系统核心或标准子程序库、数据表格等。FLASH 模块对于单芯片应用系统是理想的程序存放处, 它允许在现场进行代码更新。

Flash 是一种非易失性存储介质, 读取它的内容同 RAM 的读取一样方便, 而对它的写操作却比 EPROM 还要快。同时, 在系统掉电后, Flash 中的内容仍能可靠保持不变。Flash 的主要优点是结构简单、集成密度大、成本低。由于 Flash 可以局部擦除, 且写入、擦除次数可达数十万次以上, 从而使开发微控制器不再需要昂贵的仿真器。

1.2 FLASH结构

\$4000~4FFFF 是 48KB 为 FLASH 的存储空间, 分成 3 个 16KB 空间如图 1-1 所示。最后 16KB 空间的最后 256B, 即\$FF00~\$FFFF 是中断向量表空间。由于 FLASH 模块的地址范围超过 STAR12(16 位)最大地址 64KB 空间, 因此模块采用从一个\$8000~\$BFFF 16KB 地址窗口分配地址, 其余的地址位分配在分页寄存器 PPAGE。256KB 的 FLASH 由 4 个 64KB 块组成, 如表 3-1 所示, 每块 64KB FLASH 按照 16 位二进制方式组织, 支持按字节或字方式访问。字节和规则字的访问可在单总线周期内完成, 非规则字访问需两个总线 256KB 的 FLASH 被分成 4 个 64KB 块, 每一块又可以继续细分为 16KB 一页, 共 16 页。每一页再细分为 4KB、

2KB、1KB、或者 512B 一段。

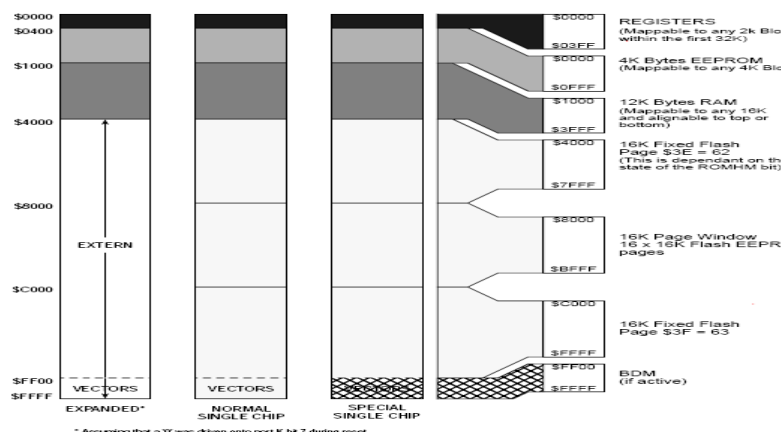


图 1-1 MC9S12DP256B 置位后的内存

1.3 FLASH特点

- 自动编程和擦除算法
- 快速扇区擦除和字编程操作
- 指令完成时中断
- 2 段命令流水线，快速实现多个字的编程
- 灵活的保护机制，避免意外编程和擦除
- 预防非法访问的加密特征

第二节 FLASH寄存器简介

FLASH 寄存器如表 2-1 所示

名称	寄存器	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
时钟分频寄存器	FCLKDIV	FDIVLD	PRDIV8	FDIV5	FDIV4	FDIV3	FDIV2	FDIV1	FDIV0
加密寄存器	FSEC	KEYEN1	KEYEN0	NV5	NV4	NV3	NV2	SEC1	SEC0
配置寄存器	FCNFG	CBEIE	CCIE	KEYACC	0	0	0	0	0
保护寄存器	FPROT	FPOPEN	NV6	FPHDIS	FPHS1	FPHS0	FPLDIS	FPLS1	FPLS0
状态寄存器	FSTAT	CBEIF	CCIF	PVIOL	ACCERR	0	BLANK	FAIL	DONE
命令寄存器	FCMD	0	CMDB6	CMDB5	0	0	CMDB2	0	CMDB0
地址寄存器	FADDRH	0	FABHI						
地址寄存器	FADDRLO	FABLO							
数据寄存器	FDATAHI	FDHI							
数据寄存器	FDATALO	FDLO							

2.1 时钟分频寄存器FCLKDIV

寄存器	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
FCLKDIV	FDIVLD	PRDIV8	FDIV5	FDIV4	FDIV3	FDIV2	FDIV1	FDIV0

FDIVLD: FLASH 时钟分频加载。FCLKDIV 寄存器被写入时此位置“1”；若此位置“0”，FCLKDIV 寄存器从最后复位后没有写入。在没有写入此寄存器之前，擦写和编程 FLASH 将引起进入错误和命令不被执行

0 - FCLKDIV 寄存器没有被写过

1 - FCLKDIV 寄存器从最近一次复位起已经被写过

PRDIV8 使能时钟 8 分频

0 - 晶振时钟为 FLASH 时钟分配器的时钟源

1 - 晶振时钟除以 8 后，为 FLASH 时钟分配器的时钟源

FDIV[5:0] 时钟分配器位

—PRDIV8 和 FDIV[5:0]的设置,必须满足 FLASH 时钟在 150 kHz – 200 kHz 范围内。最

大的分频系数为 512。在对 FLASH 进行操作之前，必须对时钟进行初始化，否则接下来的任何操作将是无效的。

2.2 配置寄存器 FCNFG

	7	6	5	4	3	2	1	0
	CBEIE	CCIE	KEYACC	0	0	0	0	0
Reset	0	0	0	0	0	0	0	0

CBEIE — 指令缓冲区空中断使能

该位在空地址，数据和指令缓冲区的情况下使能中断。

1 = CBEIF 标志被置位产生中断请求

0 = 指令缓冲区空中断禁止

CCIE — 指令完成中断使能

当所有指令被完成该位产生中断

1 = CCIF 标志被置位产生中断请求

0 = 指令完成中断禁止

KEYACC — 使能安全密钥写入

1 = 写入 flash 模块被解释成打开后门的密钥。

0 = Flash 写入后编程或擦除过程的开始

2.3 安全寄存器 FSEC

寄存器	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
FSEC	KEYEN	NV6	NV5	NV4	NV3	NV2	SEC01	SEC00

KEYEN: 安全门码控制。

1: 允许通过 BDM 或外部总线接口开启安全门。

0: 禁止通过 BDM 或外部总线接口开启安全门。

当 KEYEN 置 1 时，用户可以通过以下步骤避开安全检查

FCNFG 寄存器中 KEYACC 置 1。

通过安全门比较码地址向 FLASH 写四个正确 16 位字。

将 FCNFG 寄存器中 KEYACC 置 0。

如果全部四个字与 FALSH 内容相符，强制通过 SEC[1: 0]处于非安全状态，使 MCU 处于非安全状态。

6. 如果全部四个字与 FALSH 内容不相符，MCU 处于安全状态，向 CPU 发回安全冲突信号。

NV[6: 2]: 非冲突标志位。这 5 位是用户可以用做非冲突标志。

SEC0[1: 0] 存储安全位。这 2 位决定设备的安全状态。如表 2-2 所示

表 2-3 SEC0[1: 0] 与安全状态关系

SEC0[1: 0]	00	01	10	11
描述	安全	安全	非安全	安全

2.4 保护寄存器 FPROT

	7	6	5	4	3	2	1	0
	FPOPEN	NV6	FPHDIS	FPHS1	FPHS0	FPLDIS	FPLS1	FPLS0
Reset	F	F	F	F	F	F	F	F

FPOPEN—FLASH 编程或擦除保护

1 = FPHDIS 和 FPLDIS 定义的 FLASH 地址区域不被保护

0 =

FPHDIS—Flash 保护高地址范围禁止

该位用来决定在 Flash 块地址映射的较高末端是否存在一个被保护的区域

1 = 保护禁止

0 = 保护使能

FPHS[1:0]—Flash 保护高地址范围

这两位用来决定保护区域的范围。

FPLDIS—Flash 保护较低的地址范围禁止

该位用来决定在 Flash 块地址映射的较低末端是否存在一个被保护的区域

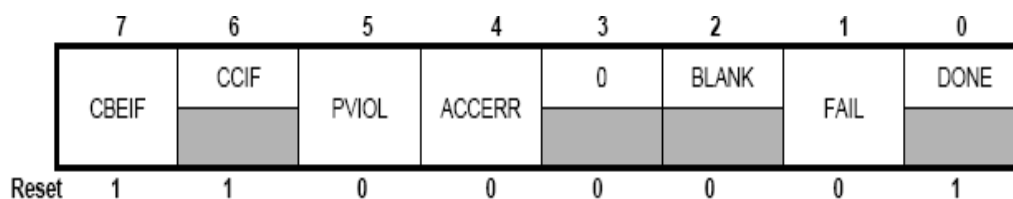
1 = 保护禁止

0 = 保护使能

FPLS[1:0]—Flash 保护低地址范围

这两位用来决定保护区域的范围

2.5 状态寄存器 FSTAT 状态寄存器



CBEIF— 指令缓冲区空中断标志

表明地址，数据和指令缓冲区为空因此一个新的指令序列可以开始。

该标志通过写入“1”清除。指令序列通过清除该位来开始。

写入“0”可异常中止一个尚未开始的指令序列且置 ACCERR 标志

1 = 缓冲区准备接收一个新指令

0 = 缓冲区满

CCIF— 指令完成中断标志

表明不再有未决的指令。该标志通过硬件写入来清除

写操作不起作用。

1 = 所有指令都被完成

0 = 指令进行中

PVIOL— 保护错误标志

表明试图对被保护的存储器区域中的一个地址编程或擦除。

若该位被置 1 则并行的编程或擦除指令不能被执行。该标志通过写入“1”被清除，也可通过在 CBEIF 被清零后或当 CCIF 被清除时写一条新的，有效的指令来清除

1 = 保护系统侵犯已出项

0 = 无故障

ACCERR— 存取错误标志

表明在编程或擦除时序中有一个错误。该标志位通过写“1”来清除。

1 = 存取错误已出现

0 = 指令时序或执行被成功完成

ACCERR: 进入错误标志。表示非法进入 FLASH 块或者与定义命令序列冲突。以下命令序列会引起 **ACCERR** 置位:

- 1.在初始化 **FCLDIV** 之前, 向 **FLASH** 地址写值;
- 2.当分页寄存器没有选择 **16KBFLASH** 块, 向\$8000~\$BFFF 地址空间的 **FLASH** 写值;
- 3.向\$4000~\$7FFF 地址空间的 **FLASH** 写值或在 **BKSEL[1: 0]**没选 **FALSH 0** 下, 向\$C000~\$FFFF 地址空间的 **FLASH** 写值;
- 向 **FLASH** 地址空间写入错误的字或字节;
- 4.**CBEIF** 未置 1 时, 向 **FLASH** 地址写值;
- 5.在执行前一个编程或擦除命令之前, 向 **FLASH** 地址空间写入第二个字;
- 6.在向 **FLASH** 地址空间写入字之后, 接着写其他 **FLASH** 寄存器, 没有写 **FCMD** 寄存器;
- 7.在执行完前一个写好的命令之前, 向 **FCMD** 寄存器写入第二个命令;
- 8.在对有保护 **FLASH** 空间, 向 **FCMD** 寄存器写入整块擦除命令;
- 9.在对有保护 **FLASH** 空间, 向 **FCMD** 寄存器写入部分擦除命令;
- 10.在写完命令寄存器后, 接着写其他 **FLASH** 寄存器, 没有写 **FSTAT** 寄存器;
- 11.在命令序列已开始或擦除/编程命令以执行的同时, 从 **FLASH** 块读取。这样读到无效数据, 但能读其他没有编程的 **FLASH** 块。

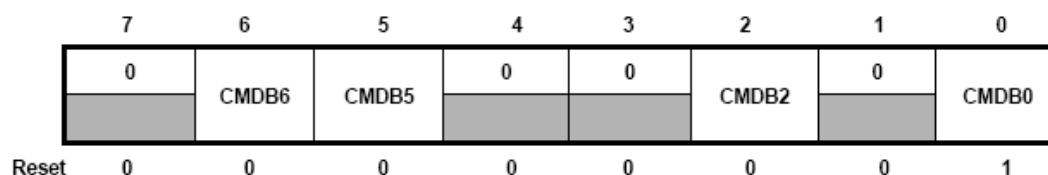
BLANK: **FLASH** 块校验标志, 表示响应的擦除—校验命令时 **FLASH** 块全部擦除, 标志写“1”清零, 写“0”无影响。

1: FLASH 块全部擦除

0: FLASH 块没有全部擦除

BITS[1: 0] 这两位用于工厂测试预留，用户无法使用。

2.6 命令寄存器 FCMD



ERASE: FLASH 擦除

0: 无意义

1: 在 MASS=0 时，区域擦除；在 MASS=1 时，区域擦除；

PROG: FLASH 编程

0: 无意义。

1: FLASH 编程。

ERVER: 擦除校验:

0: 无意义。

1: 区域擦除后擦除校验。

MASS: 全局铲除

0: 部分擦除

1: 全部擦除。

第三节 FLASH应用实例

3.1 FLASH的写入操作

FLASH 的写入操作可以按照以下步骤进行:

1. FCLKDIV 寄存器初始化，确定分频因子

2. 设置 FCNFG 寄存器和 PPAGE 寄存器，用于管理命令序列。写 FCNFG 寄

寄存器中的 BKSEL[1; 0]位, 选择要擦除或编程的 FALKSH 块状的相应寄存器。如果编程在 8000~BFFF 地址范围内, 向 (内核) 寄存器 (X030) 写入值选择要编程的某一 16KB 页。当编程在 4000~7FFF 或 C000~FFFF 地址范围, 没有必要写分页寄存器。

3. 检查 FLASH 时钟分频寄存器(FCLKDIV_FDIVLD)是否已经设置
4. 检查命令缓冲区(FSTAT_CBEIF)是否为空
5. 对读写错误(FSTAT_ACCER)标志位清零(写 1 清零)
6. 对保护区编程错误(FSTAT_PVIOL)标志位清零(写 1 清零)
7. 将编程的数据赋值给要编程的 FLASH 地址
8. 对 FLASH 命令寄存器(FCMD)写入编程命令(0x20)
9. 对命令缓冲区空标志位(FSTAT_CBEIF)清零(写 1 清零)
10. 等待, 直到命令缓冲区标志位(FSTAT_CBEIF)置位
11. 检查是否还有新的数据需要写入到 FLASH, 如有转到第 7 步
12. 等待, 直到命令完成标志位(FSTAT_CCIF)置位

3.2 FLASH的擦除操作

1. FCLKDIV 寄存器初始化, 确定分频因子

2. 设置 FCNFG 寄存器和 PPAGE 寄存器, 用于管理命令序列。写 FCNFG 寄存器中的 BKSEL[1; 0]位, 选择要擦除或编程的 FALKSH 块状的相应寄存器。如果编程在 8000~BFFF 地址范围内, 向 (内核) 寄存器 (X030) 写入值选择要编程的某一 16KB 页。当编程在 4000~7FFF 或 C000~FFFF 地址范围, 没有必要写分页寄存器。

3. 检查 FLASH 时钟分频寄存器(FCLKDIV_FDIVLD)是否已经设置
4. 检查命令缓冲区(FSTAT_CBEIF)是否为空
5. 对读写错误(FSTAT_ACCER)标志位清零
6. 对保护区编程错误(FSTAT_PVIOL)标志位清零
7. 在要擦除的 FLASH 的段地址内写入任意数据
8. 对 FLASH 命令寄存器(FCMD)写入段擦除命令(0x40)

9. 对命令缓冲区空标志位(FSTAT_CBEIF)清零

10. 等待, 直到命令完成标志位(FSTAT_CCIF)置位

3.3 FLASH的擦写操作注意事项

a、由于在 FLASH 的擦除和写入过程中, FLASH 是不能读的, 故擦除和写入 FLASH 的程序要放在 RAM 中, 换言之, 在 FLASH 的擦除或写入之前, 要把擦除或写入的可执行代码复制到 RAM 中去, 并让程序在 RAM 中执行。这段程序的编译是在 FLASH 中的程序段复制到 RAM 中的数据段, 不论这段程序复制到 RAM 中的哪一段, 即无论程序的起始地址在哪里, 程序都能正确执行。这就要求生成的程序和装载地址无关, 程序可以重载。

b、系统初始化后一定要严格按照上述步骤规范设置, 这样避免出现逻辑错误也便于检查错误。此外, 在编程选择写入或者擦除的范围时, 一定要明确所选区域是否需要设置分页寄存器。

3.4 FLASH示例程序

```
#include <hidef.h>          /* common defines and macros */
#include <mc9s12dp256.h>     /* derivative information */
#pragma LINK_INFO DERIVATIVE "mc9s12dp256b"
#define portb (*((volatile unsigned char*)(0x0001)))
#define ddrb  (*((volatile unsigned char*)(0x0003)))
#define ppage (*((volatile unsigned char*)(0x0030)))
#define fclkdiv (*((volatile unsigned char*)(0x0100)))
#define fcnfg  (*((volatile unsigned char*)(0x0103)))
#define fprot  (*((volatile unsigned char*)(0x0104)))
#define fstat  (*((volatile unsigned char*)(0x0105)))
#define fcmd   (*((volatile unsigned char*)(0x0106)))
#define any    (*((volatile unsigned char*)(erase)))
#define a      (*((volatile unsigned char*)(0x2000)))
#define EraseInSector 0x40;
#define ProgramInAlignedWord 0x20;
unsigned char *flashaddr;
unsigned char *progaddr;
unsigned char *erase,*base,*end1,*end2;
unsigned int i;
/*flash 擦除函数*/
void eraseflash ()
{
    while(erase<=((volatile unsigned char*)(end1))
```

```

{
    while(!(fclkdiv&0x80))    //时钟是否加载
        {}
    while(!(fstat&0x80))    //命令缓冲区是否为空
        {}
    while(!(fprot&0x80))    //flash 是否允许写入或者擦除
        {}
    any=0xff;
    fcmd=EraseInSector;    //扇区擦除命令，每次擦除 512B
    fstat|=0x80;    //准备接收新的命令
    while(!(fstat&0x40))    //判断命令是否完成，没有则等待
        {}
    erase+=512;    //擦除地址递加
}
}

/*flash 写入函数*/
void writeflash ()
{
    while(flashaddr<=(volatile unsigned char*)(end2))
    {
        while(!(fclkdiv&0x80))
            {}
        while(!(fstat&0x80))
            {}
        while(!(fprot&0x80))
            {}
        while(!SCI0SR1_RDRF);    //等待接收数据
        SCI_Receive(&a);
        *flashaddr=*progaddr;
        if(SCI0SR1_TDRE)
        SCI_Transmit(*flashaddr);
        flashaddr++;
        progaddr++;
    FCMD=ProgramInAlignedWord;
        FSTAT|=0x80;
        while(!(FSTAT&0x40))
            {}
    }
}

/*flash 初始化函数*/
void flash_init ()
{
    fclkdiv=0x4a;    //时钟分频
    FCNFG=0x00;    /*选择 flash 模块 0*/

```



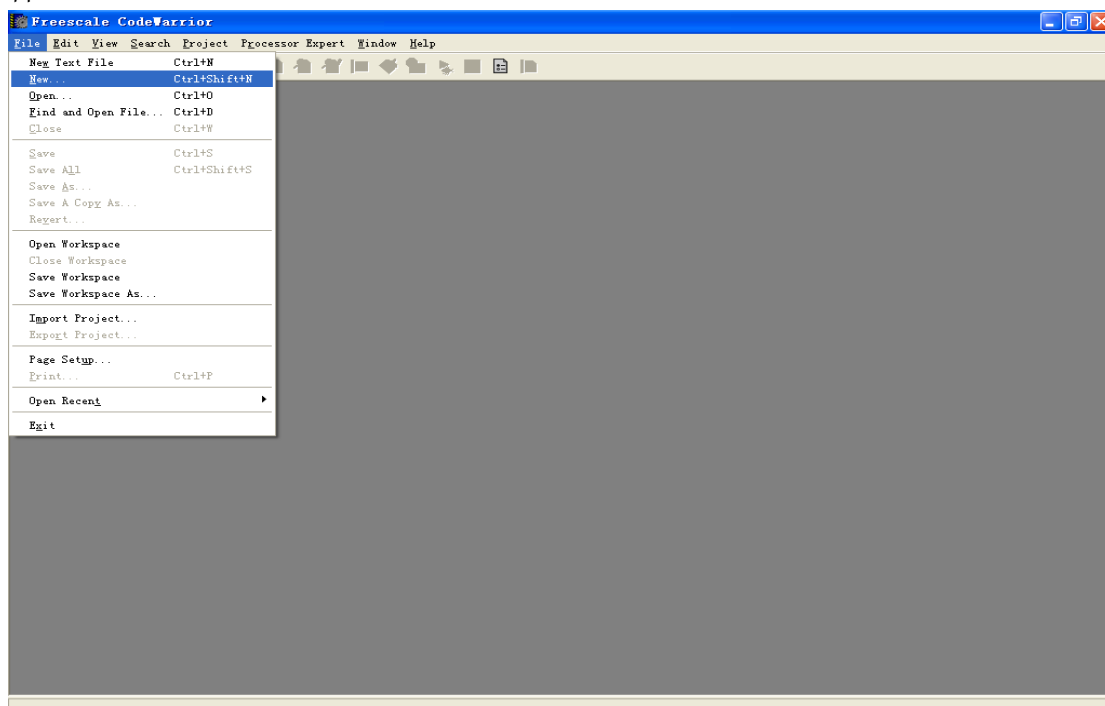
```

        FSTAT=0X30;    //清除上次的错误标示位
    // PPAGE=0x30;    /*分页寄存器设置，选择 30 页
    }
    // 串口初始化函数
int SCI_Init(void) {
    SCI0BDL=0x34;
    SCI0CR2=0X2C;
    }
    //发送子函数
int SCI_Transmit(byte data){
    while(!SCI0SR1_TDRE);
        SCI0DRL=data;
    }
    //接收子函数
int SCI_Receive(byte *data){
    *data=SCI0DRL;
    }
void main(void)
{
    ddrb=0xff;
    erase=(volatile unsigned char*)(0x4000);
    end1= (volatile unsigned char*)(0xbfff);
    end2= (volatile unsigned char*)(0x5000);
    flashaddr=(volatile unsigned char*)(0x4000);
    progaddr=(volatile unsigned char*)(0x2000);
    flash_init () ;//flash 初始化函数调用
    portb=0xf0;
    SCI_Init();    // 串口初始化函数调用
    eraseflash () ;    //flash 擦除函数
    for(i=0;i<=10000;i++)
        portb=0x0f;    //演示擦除完毕
    writeflash ();    // flash 写入函数
        while(1) {}
    }

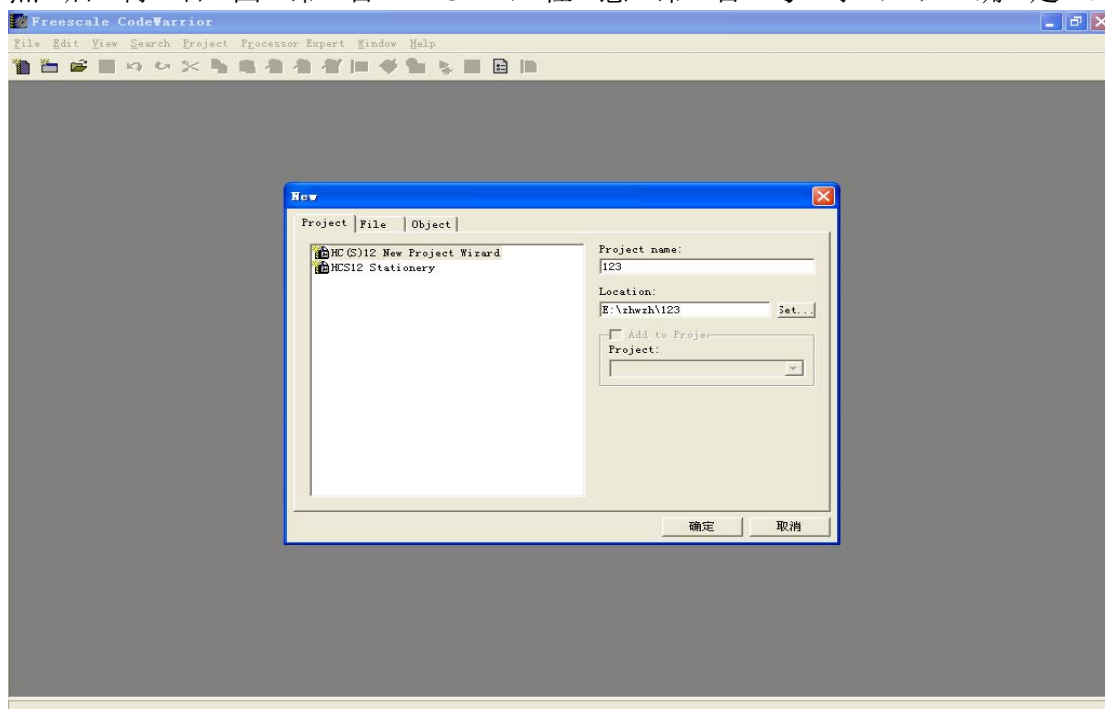
```

第八章 CodeWarrior IDE 12 应用

第一步：双击桌面 CodeWarrior IDE 12 图标，进入 CodeWarrior IDE 12 运用界面。在打开的界面中点击菜单 File，在其下拉菜单中点击 New，新建数据库文件



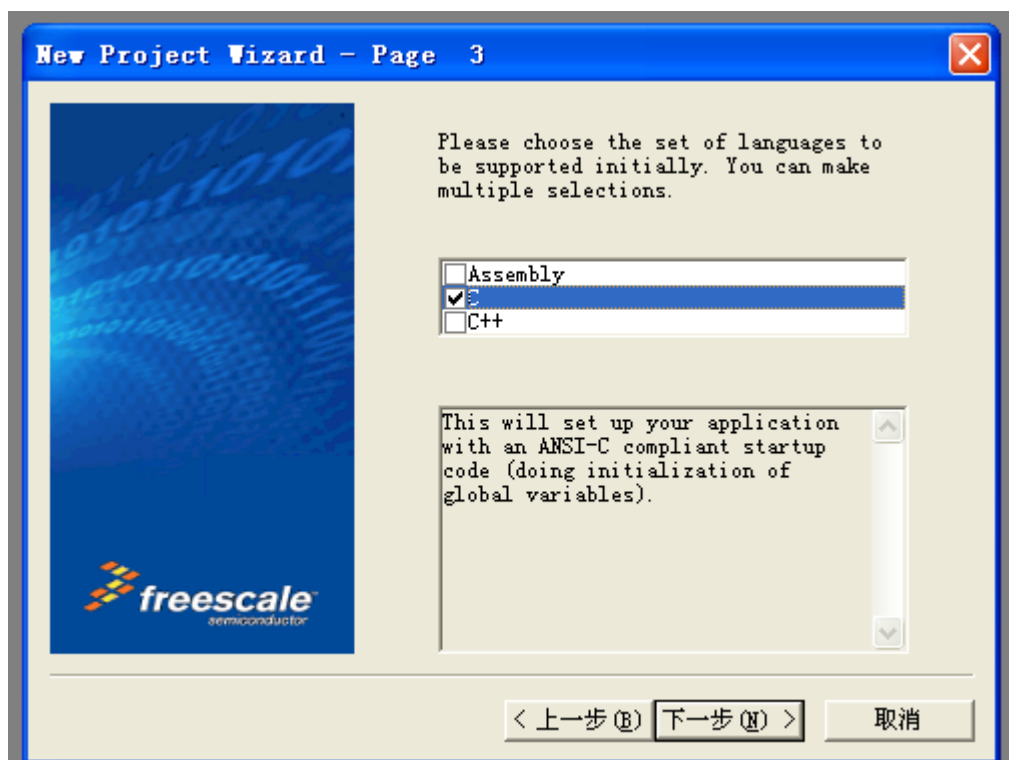
第二步：新建一个数据库，如图选择第一行（HC(s)12New Project Wizard），然后再右面命名 123（任意命名均可），确定。



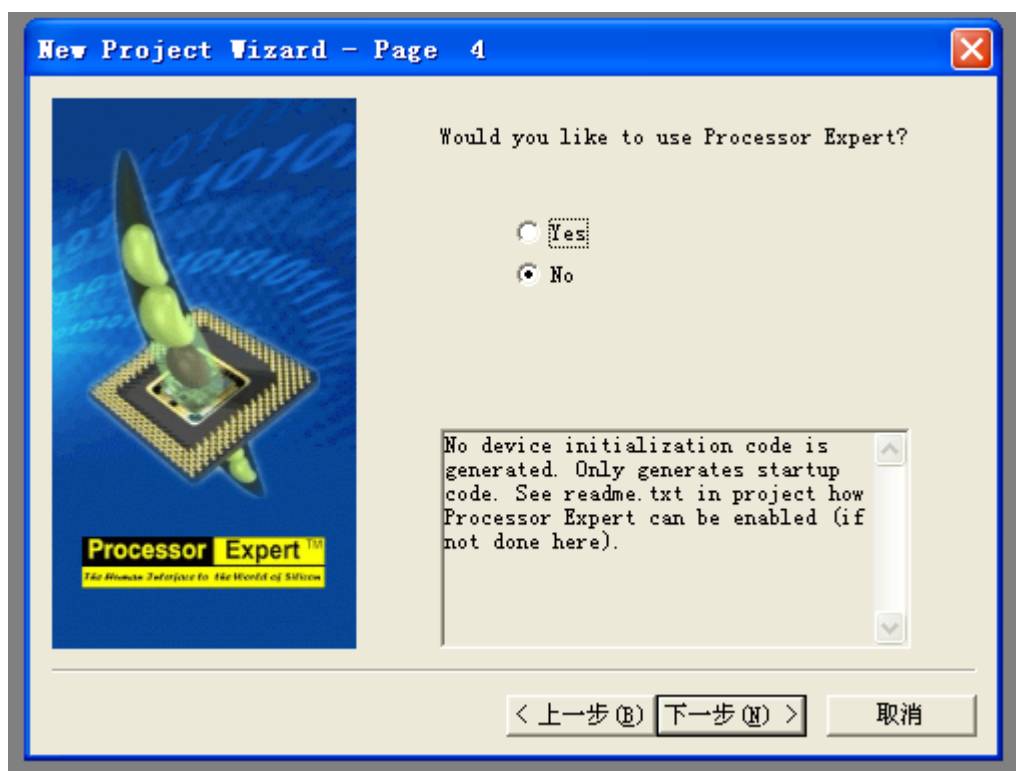
第三步：进入欢迎界面，点击下一步。然后选择芯片型号 MC9S12DP256B，点击下一步。程序一定要与单片机型号一直，否则在硬件调试中会出现错误。



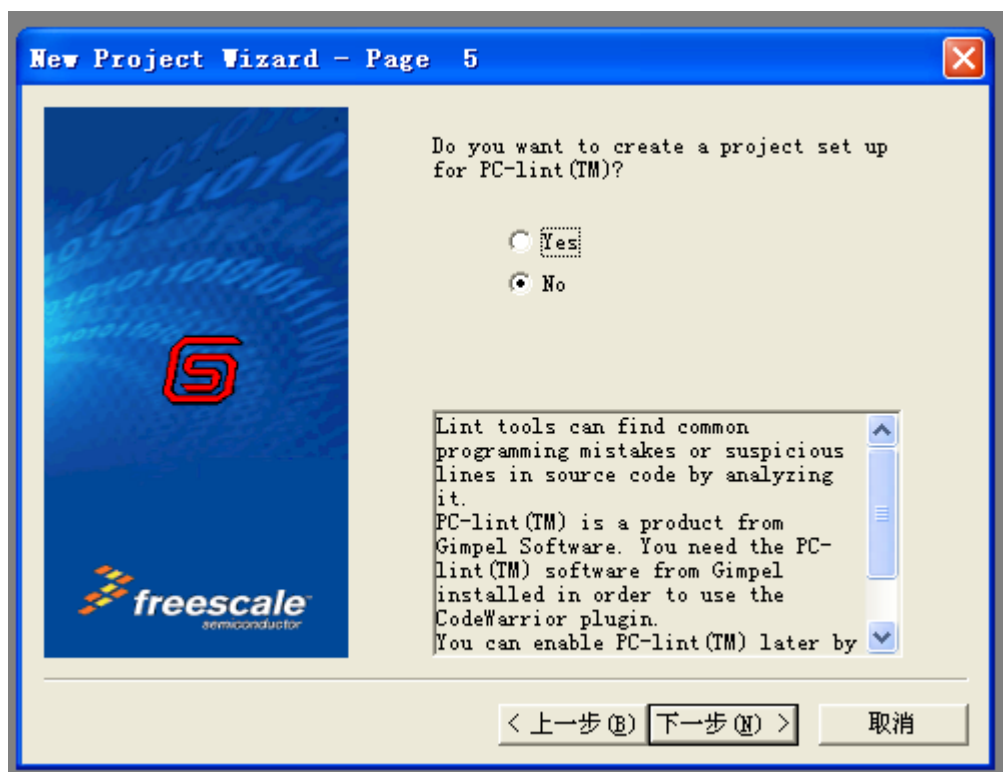
第四步：选择使用的语言，在此选择 C 语言，点击“下一步”。



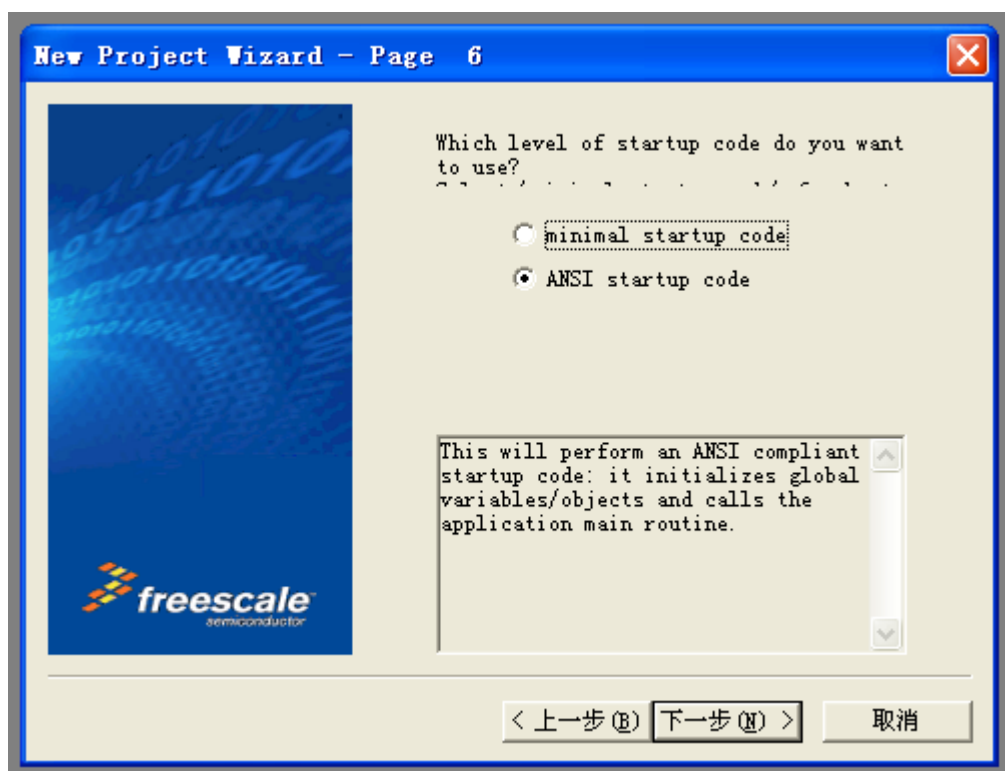
第五步：选择调试专家，在此选择“NO”，点击“下一步”。



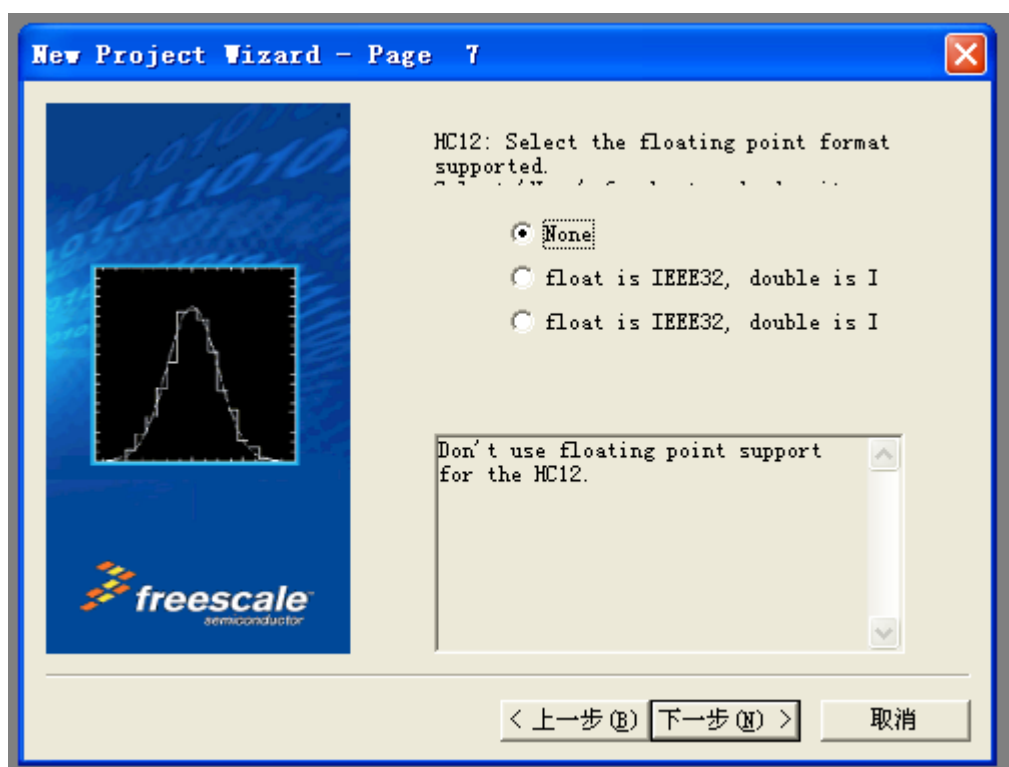
第六步：界面如下图所示，选择“NO”，点击“下一步”。



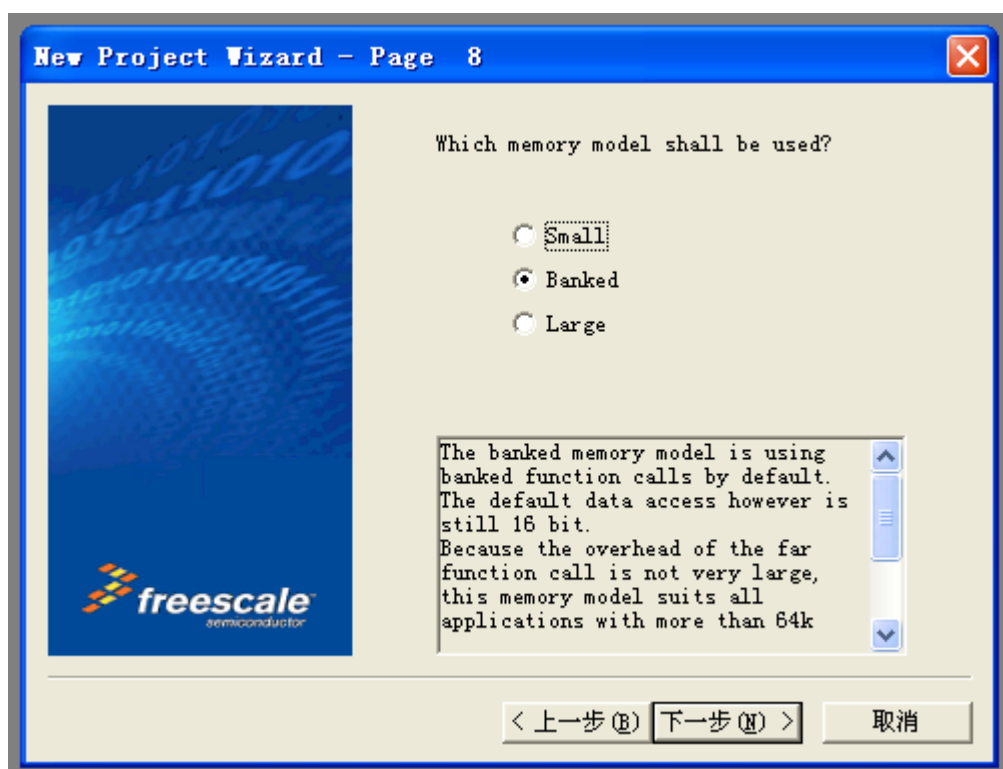
第七步：界面如下图所示，选择”ANSI startup code”，点击“下一步”。



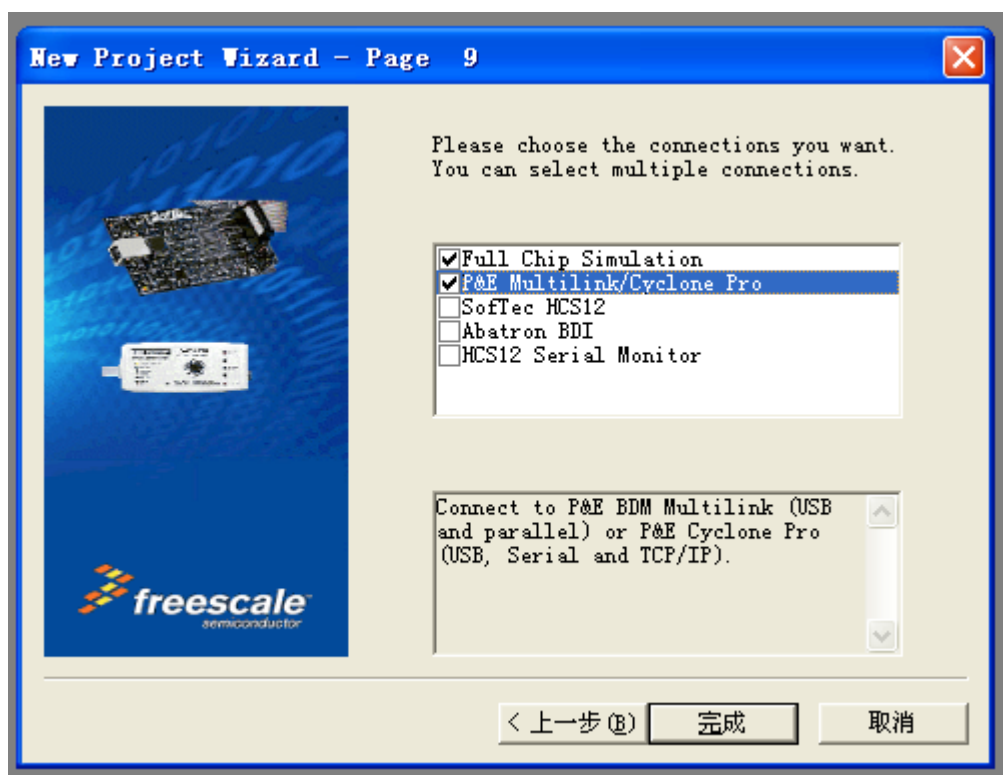
第八步：选择有无浮点格式，根据自己情况而定，本程序在此选择“None”，点击“下一步”。



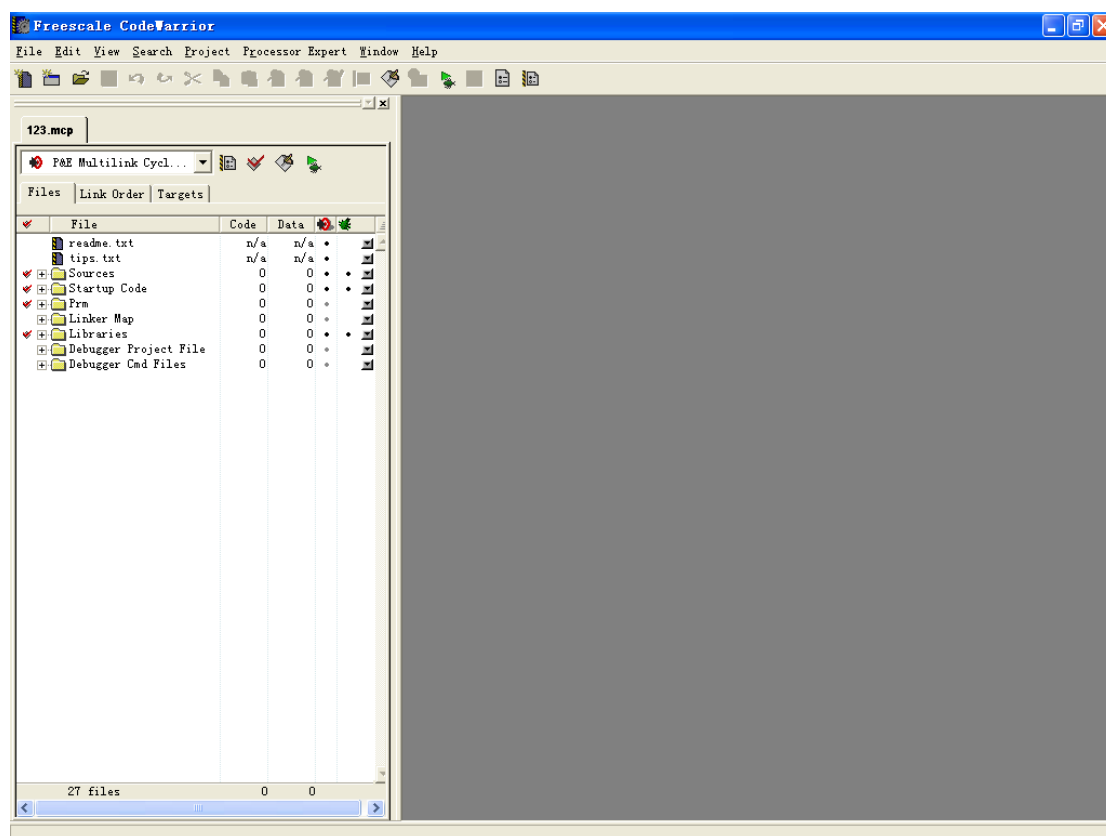
第九步：界面出现“选择存储模式？”，选择 Banked，点击下一步。



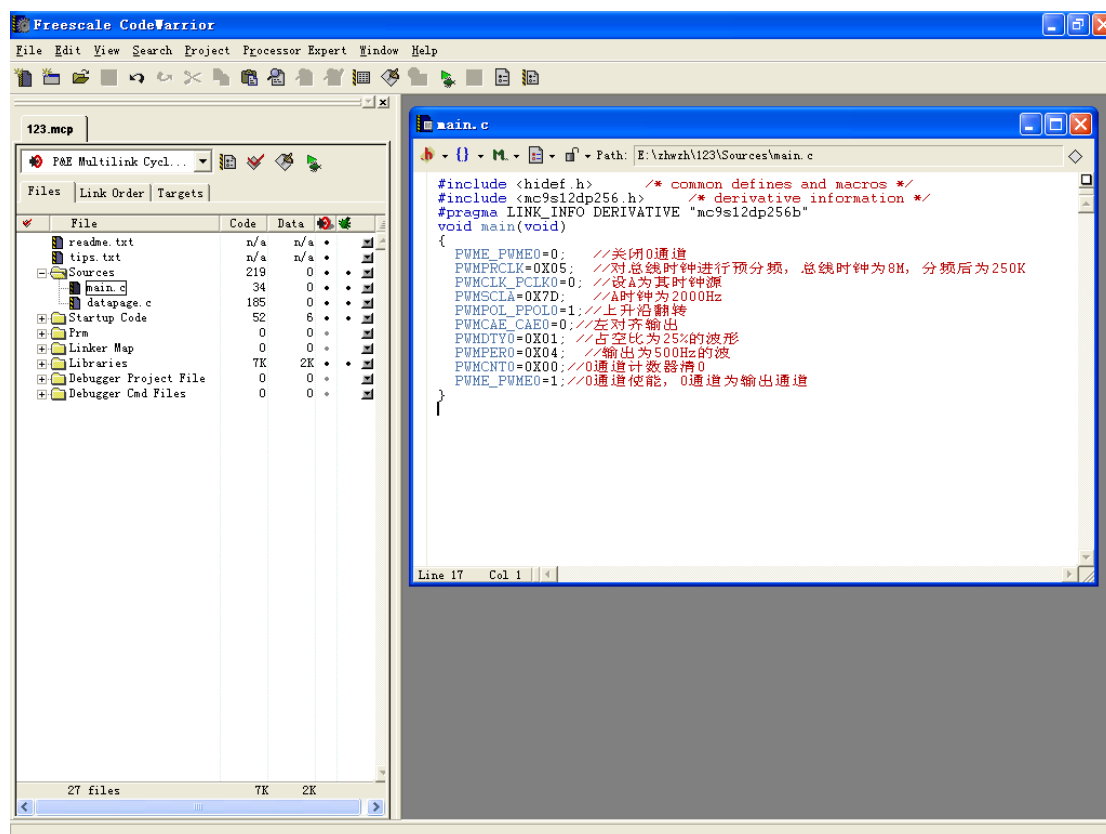
第十步：界面出现“选择硬件连接电缆型号？”，前两项全选中，点击“完成”。



第十一步：进入如下界面，点击左侧 Sources 文件前面的加号，选择主程序中的 Main.c，双击左键进入。



第十二步：显示界面如下，其中右侧为 Main.c 的编辑环境，可以在此输入你的单片机程序，也可以删除、修改或拷贝你前面编辑完成的程序。




例如当输入以下源程序：


```

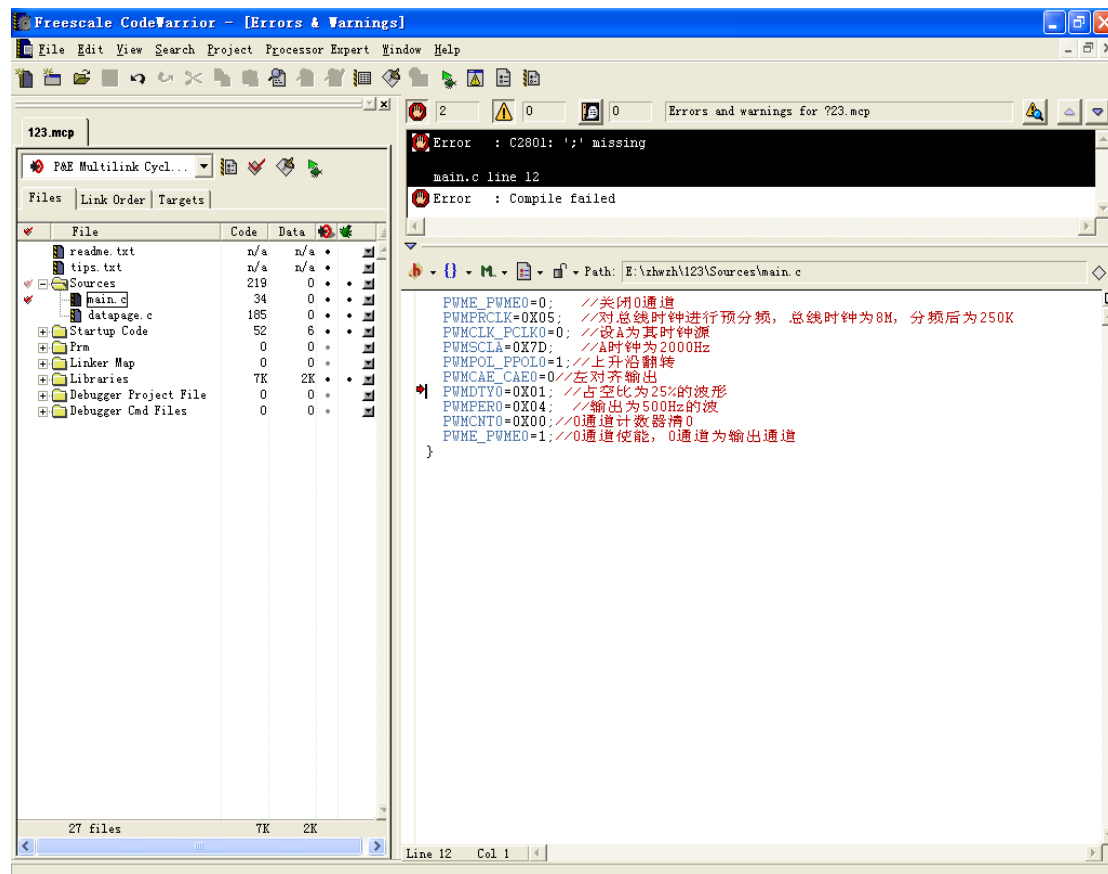
#include <hidef.h>          /* common defines and macros */
#include <mc9s12dp256.h>    /* derivative information */
#pragma LINK_INFO DERIVATIVE "mc9s12dp256b"

void main(void)
{
    PWME_PWME0=0;    //关闭 0 通道
    PWMPRCLK=0X05;    //对总线时钟进行预分频，总线时钟为 8M，分频后为 250K
    PWMCLK_PCLK0=0;    //设 A 为其时钟源
    PWMSCLA=0X7D;    //A 时钟为 2000Hz
    PWMPOL_PPOL0=1;    //上升沿翻转
    PWMCAE_CAE0=0;    //左对齐输出
    PWMDTY0=0X01;    //占空比为 25% 的波形
    PWMPER0=0X04;    //输出为 500Hz 的波
    PWMCNT0=0X00;    //0 通道计数器清 0
    PWME_PWME0=1;    //0 通道使能，0 通道为输出通道
}


```

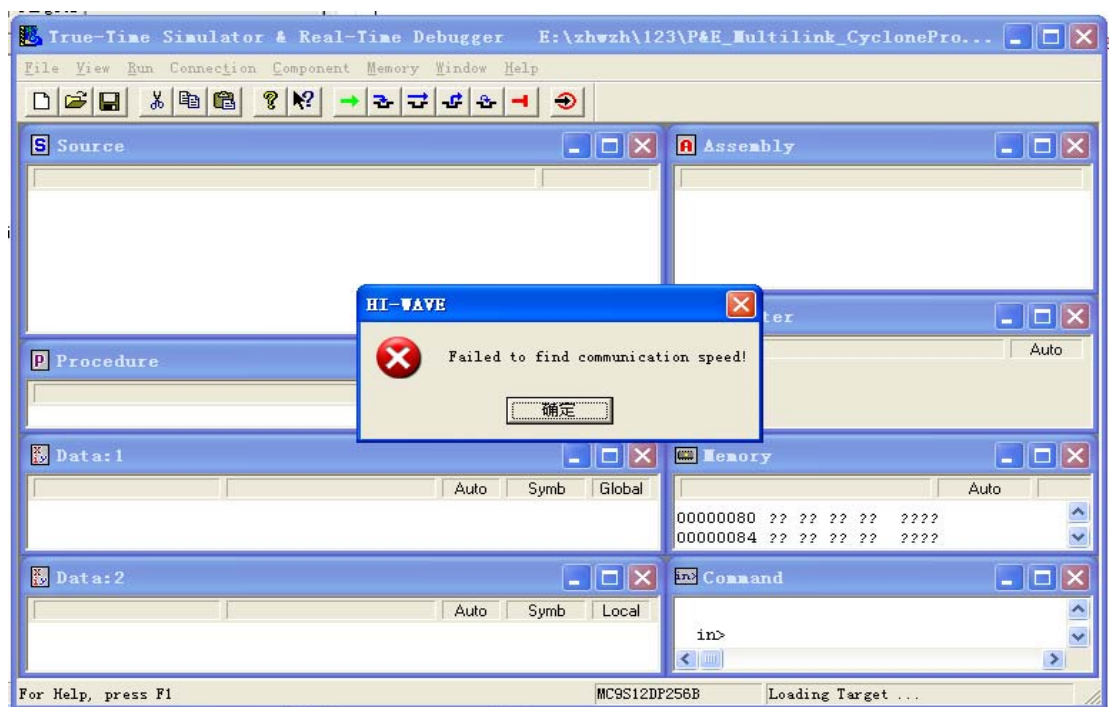
第十三步：程序输入以后，点击工具栏的  图标（make）检查程序是否有错，

如果有错，会在编辑框上方提示错误警告，以  为标志，程序中会随之用红色箭头在程序中标出出错位置。例如下图中的程序有两个错误（第 12 行少了一个分号，导致无法编译通过）。检查程序并修改，直到没有错误为止。

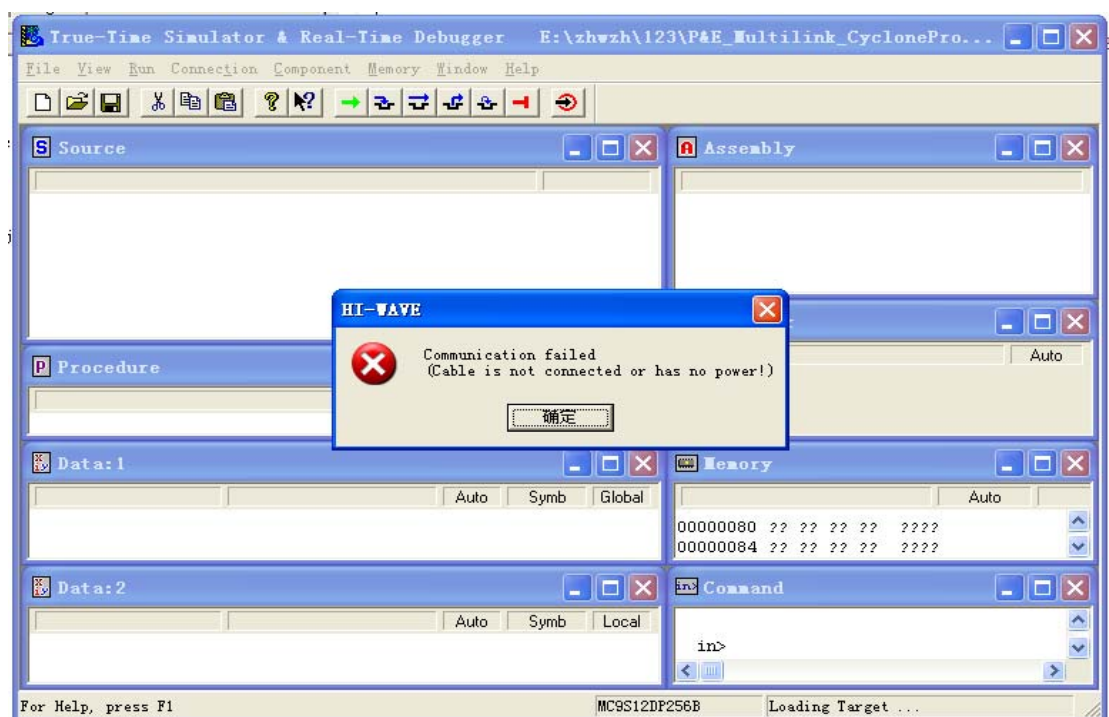


前面介绍的属于程序在计算机中的编译运行，不需要开发工具，适用于初试开发和检查程序有无语法错误。下面介绍采用开发板的实际调试过程：

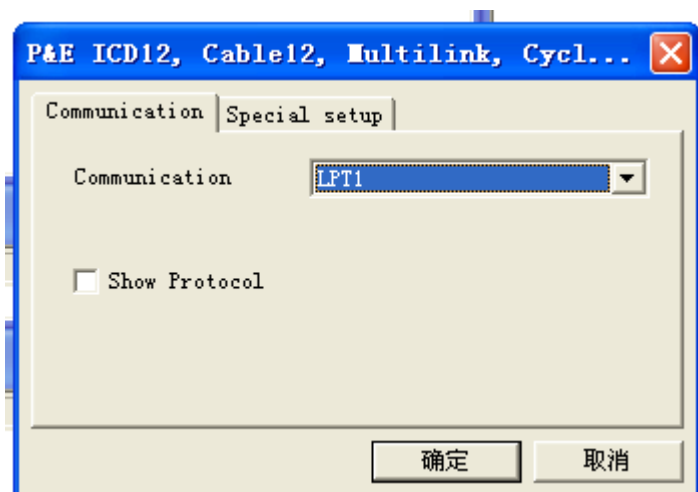
第一步：在上述单片机程序编译的基础上，将编程器与仿真试验板连接好，编程器下载口用数据线同计算机相连，试验板与 5V 直流电源相连。点击  图标开始调试（debug），进入调试仿真状态后会出现如下界面。



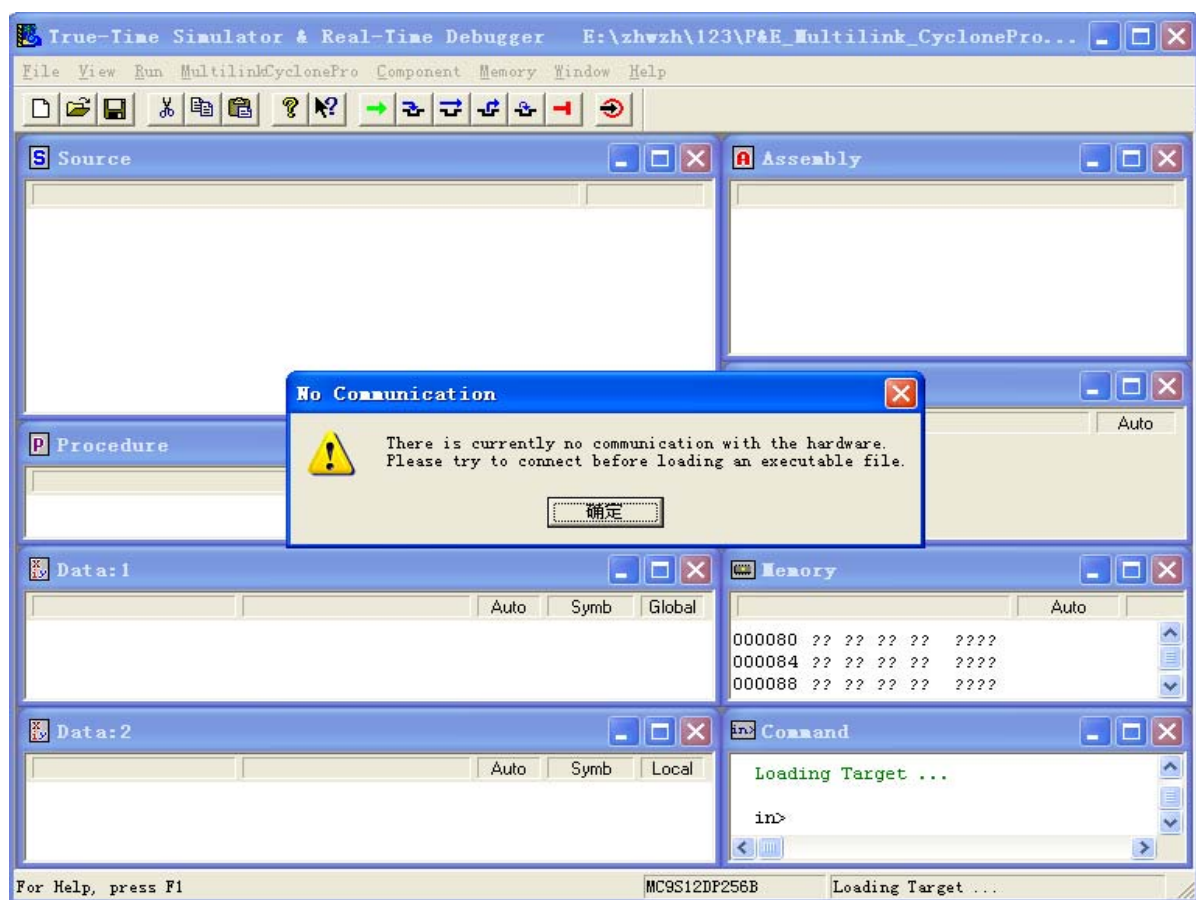
第二步：在上面的界面中点击“确定”以后。会出现下面界面“连接失败”，仍点击“确定”。



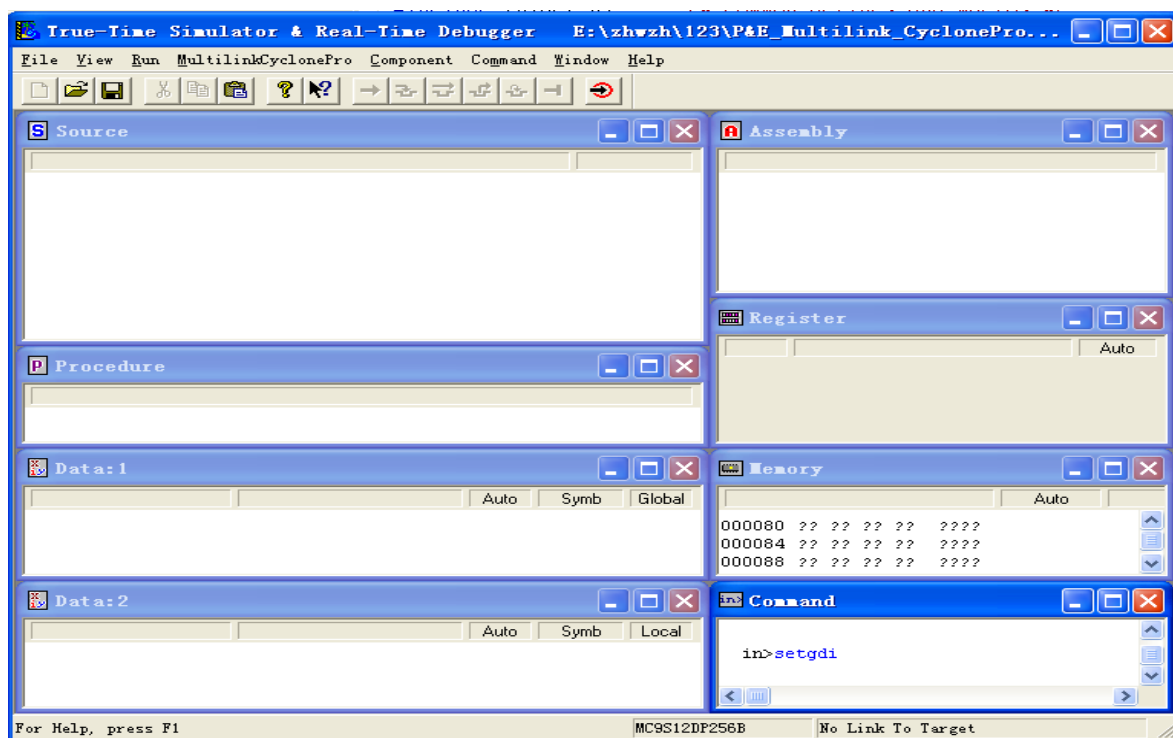
第三步：在上面的界面单击“确定”以后，会出现如下界面单击“取消”



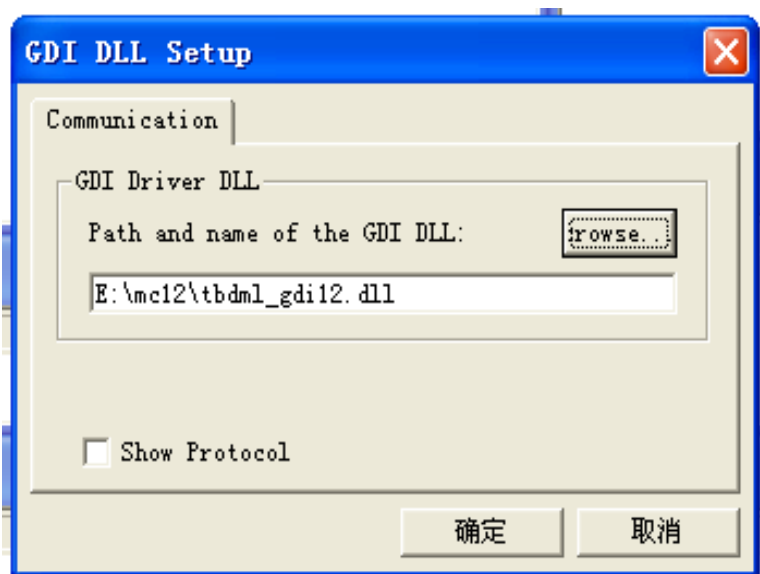
第四步：在单击“取消”以后，出现“硬件连接”失败界面如下图所示，单击确定。



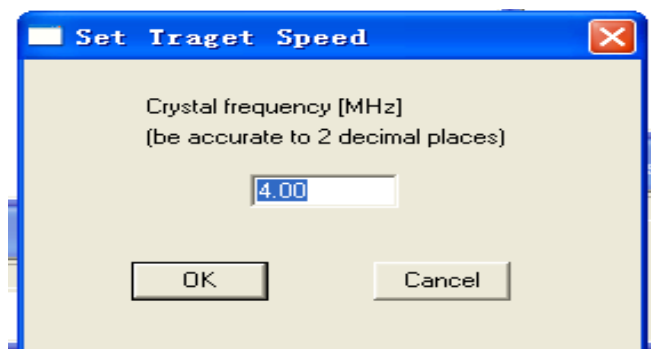
第五步：在上面的界面单击“确定”以后，会发现光标在 COMMAND 窗口 in> 后闪动，输入字母 set gdi(注意 set 和 gdi 之间有一个空格)后单击“回车键”。



第六步：单击 Browse 按钮，在安装的软件文件夹 mc12 中找到指定文件 tbdml_gdi12.dll，然后单击“确定”。



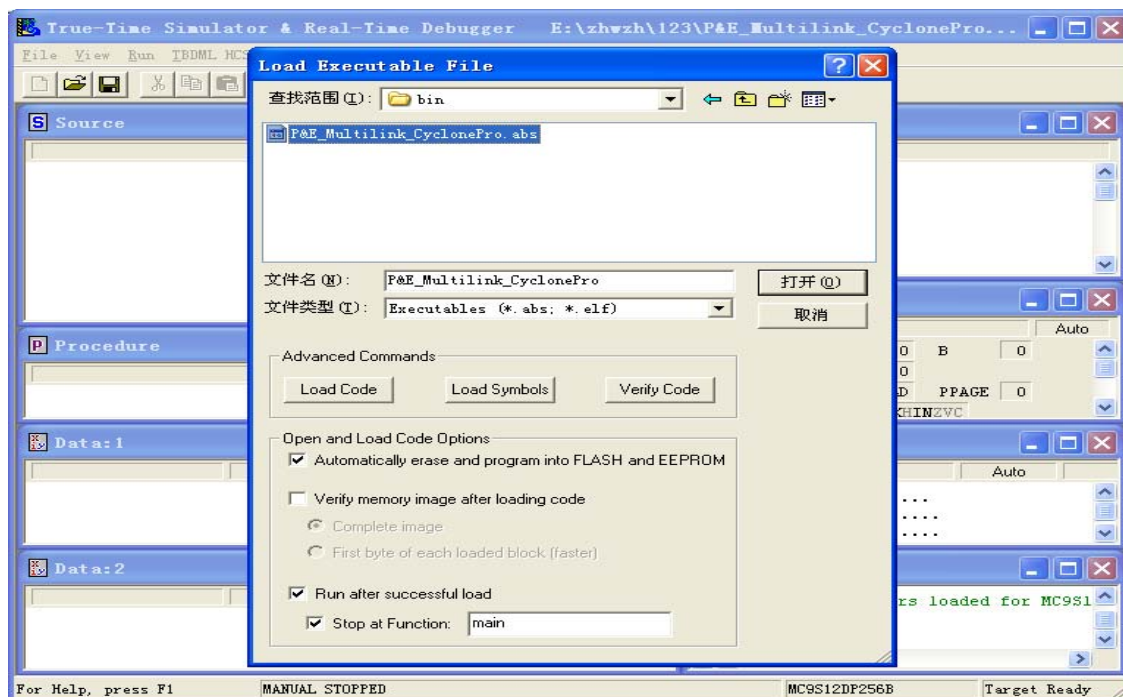
第七步：在上面的界面单击“确定”以后，在图示窗口输入晶振频率数值 16:00 后单击“OK”。



第八步：在上面的界面单击“确定”以后，在如下界面中单击“OK”。



第九步：在 TBDML HCS12 中单击 Load，然后在自己建立的文件夹里找到 bin 文件夹，单击 bin 文件夹然后再双击其中的第一个文件即可将源程序下载到 flash 中。



第十步：单击图示按钮即可执行所编程序功能。

