第5章　定时/计数器的 C 语言程序设计

5.1 定时/计数器的结构

5.1.1 时基信号发生器

时间基准信号，简称时基信号，来自于 32768Hz 实时时钟，通过频率选择组合而成。时基信号发生器的选频逻辑 TMB1，为 TimerA 的时钟源 B 提供各种频率选择信号，并为中断系统提供中断源(IRQ6)信号。此外，时基信号发生器还可以通过分频产生 2Hz、4Hz、1024Hz、2048Hz 以及 4096Hz 的时基信号，为中断系统提供各种实时中断源(IRQ4 和 IRQ5)信号。时基信号发生器的结构如图 5.1 所示。

时基信号发生器通过对 P_Timebase_Setup(写)($700EH)单元的编程写入来进行选频操作。

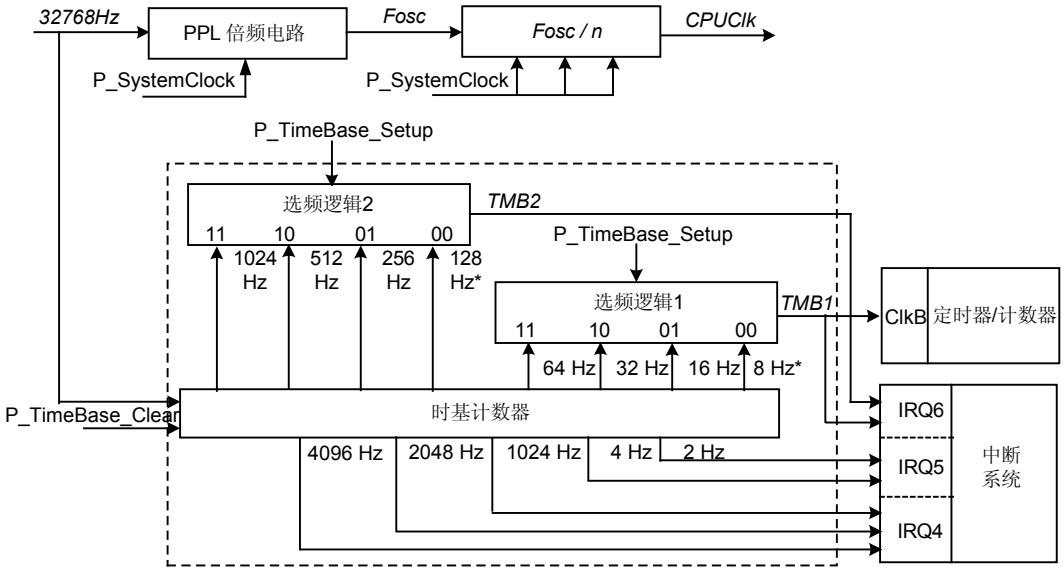

图5.1　时基信号发生器的结构

5.1.2 定时器/计数器

SPCE061A 提供了两个 16 位的定时/计数器：TimerA 和 TimerB。TimerA 为通用计数器；TimerB 为多功能计数器。TimerA 的时钟源由时钟源 A 和时钟源 B 进行“与”操作而形成；TimerB 的时钟源仅为时钟源 A。TimerA 的结构如图 5.2 所示，TimerB 的结构如图 5.3 所示。

定时器发生溢出后会产生一个溢出信号(TAOUT/TBOUT)。一方面，它会作为定时器中断信号传输给 CPU 中断系统；另一方面，它又会作为 4 位计数器计数的时钟源信号，输出一个具有 4 位可调的脉宽调制占空比输出信号 APWMO 或 BPWMO(分别从 IOB8 和 IOB9 输出)，可用来

控制马达或其它一些设备的速度。此外，定时器溢出信号还可以用于触发 ADC 输入的自动转换过程和 DAC 输出的数据锁存。

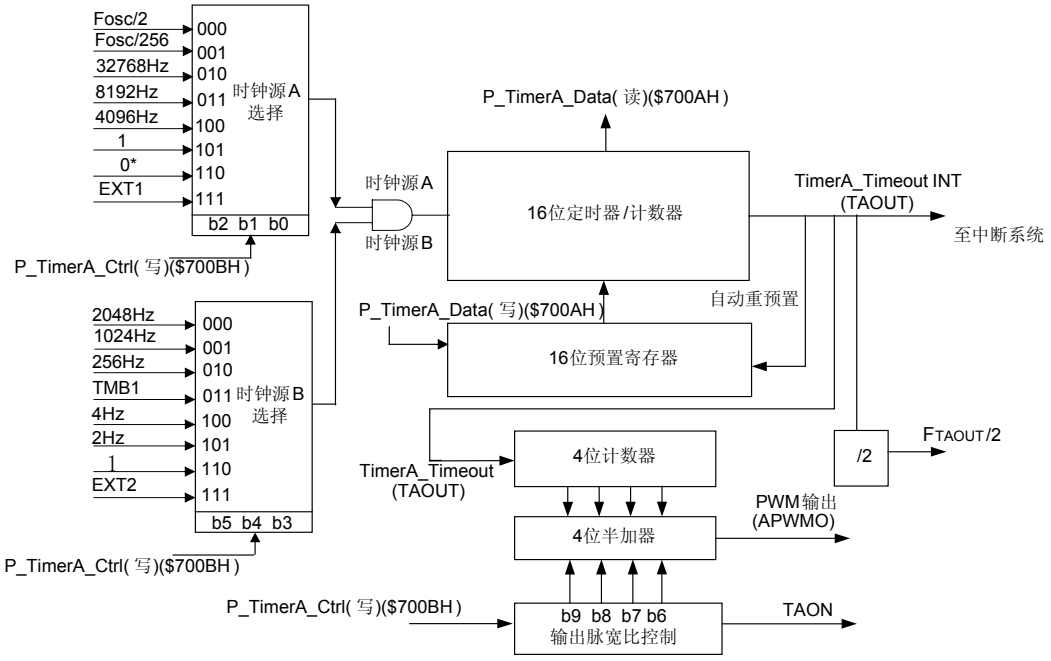

图5.2 TimerA 结构

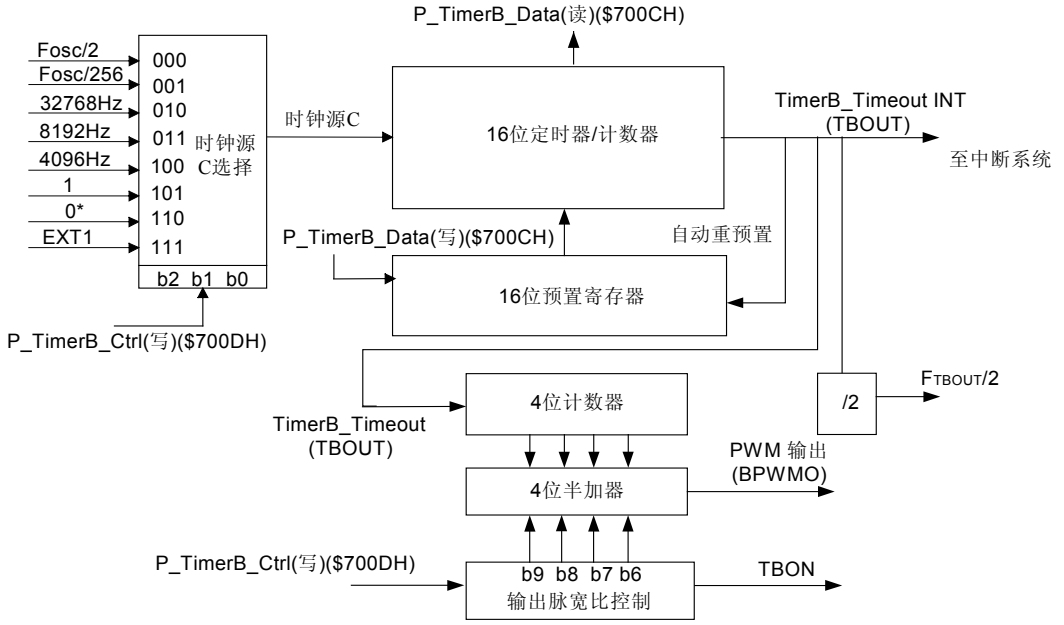

图5.3 TimerB 结构

向定时器的P_TimerA_Data(读/写)($700AH)单元或P_TimerB_Data(读/写)($700CH)单元写入一个计数值 N 后，选择一个合适的时钟源，定时器/计数器将在所选的时钟频率下开始以递增方式计数 N，N+1，N+2，…0xFFFE，0xFFFF。当计数达到 0xFFFF 后，定时器/计数器溢出，产生中断请求信号，被 CPU 响应后送入中断控制器进行处理。同时，N 值将被重新载入定时器/计数器并重新开始计数。

时钟源 A 和时钟源 C 是高频时钟源（从图中可以看出，时钟源 A 和时钟源 C 的结构是相同的），来自带锁相环的晶体振荡器输出 Fosc；时钟源 B 的频率来自 32768Hz 实时时钟系统，也就是说，时钟源 B 可以作为精确的计时器。例如，2Hz 定时器可以作为实时时钟的时钟源。

时钟源 A 和时钟源 B 的组合，为 TimerA 提供了多种计数速度。若以 ClkA 作为门控信号，‘1’表示允许时钟源 B 信号通过，而‘0’则表示禁止时钟源 B 信号通过。例如，如果时钟源 A 为“1”，TimerA 时钟频率将取决于时钟源 B；如果时钟源 A 为“0”，将停止 TimerA 的计数。EXT1 和 EXT2 为外部时钟源。

5.2 定时/计数器控制的寄存器

P_Timebase_Setup(写)($700EH)

时基信号发生器通过对 P_Timebase_Setup(写)($700EH)单元（如表 5.1 所示）的编程写入来进行选频操作。

表5.1 P_Timebase_Setup 单元

b15- b4	B3	b2	b1	b0
---	TMB2 选频逻辑		TMB1 选频逻辑	

表5.2 选频逻辑

b3	b2	TMB2	b1	b0	TMB1
0	0	128Hz*	0	0	8Hz**
0	1	256Hz	0	1	16Hz
1	0	512Hz	1	0	32Hz
1	1	1024Hz	1	1	64Hz
*：默认的 TMB2 输出频率为 128Hz			**：默认的 TMB1 输出频率为 8Hz		

P_Timebase_Clear(写)($700FH)

P_Timebase_Clear (写)($700FH)单元是控制端口，设置该单元可以完成时基计数器复位和时间校准。向该单元写入任意数值后，时基计数器将被置为“0”，以此可对时基信号发生器进行精确的时间校准。

P_TimerA_Data(读/写)($700AH)

TimerA 的数据单元，用于向 16 位预置寄存器写入数据(预置计数初值)或从其中读取数据。在写入数值以后，计数器便会在所选择的频率下进行加一计数，直至计数到 0xFFFF 产生溢出。溢出后 P_TimerA_Data 中的值将会被重置，再以置入的值继续加一计数。

P_TimerA_Ctrl(写)($700BH)

TimerA 的控制单元如表 5.3 所示。用户可以通过设置该单元的第 0~5 位来选择 TimerA 的时钟源(时钟源 A、B)。设置该单元的第 6~9 位（如表 2.13 所示），TimerA 将输出不同频率的脉宽调制信号，即对脉宽占空比输出 APWMO 进行控制。

表5.3 P_TimerA_Ctrl 单元

b15 – b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
---	占空比的设置（表 5.4）				时钟源 B 选择位(表 5.6)			时钟源 A 选择位(表 5.5)		

表5.4 设置 B6-B9 位

b9	b8	b7	b6	脉宽占空比(APWMO)	TAON[1]
0	0	0	0	关断	0
0	0	0	1	1/16	1
0	0	1	0	2/16	1
0	0	1	1	3/16	1
0	1	0	0	4/16	1
0	1	0	1	5/16	1
0	1	1	0	6/16	1
0	1	1	1	7/16	1
1	0	0	0	8/16	1
1	0	0	1	9/16	1
1	0	1	0	10/16	1
1	0	1	1	11/16	1
1	1	0	0	12/16	1
1	1	0	1	13/16	1
1	1	1	0	14/16	1
1	1	1	1	TAOUT[2]触发信号	1

注：

[1]：TAON 是 TimerA(APWMO)的脉宽调制信号输出允许位，默认值为“0”，当 TimerA 的第 6~9 位不全为零时 TAON=1；

[2]：TAOUT 是 TimerA 的溢出信号，当 TimerA 的计数从 N 达到 0xFFFF 后(用户通过设置 P_TimerA_Data 单元指定 N 值)，发生计数溢出。产生的溢出信号可以作为 TimerA 的中断信号被送至中断控制系统；同时 N 值将被重新载入预置寄存器，使 Timer 重新开始计数。TAOUT 触发信号(TAOUT/2)的占空比为 50%，频率为 $F_{TAOUT}/2$，其它信号的频率为 $F_{TAOUT}/16$。请参考 TimerA 结构图。

表5.5 设置 b0－b2 位

b2	b1	b0	时钟源 A 的频率
0	0	0	Fosc/2
0	0	1	Fosc/256
0	1	0	32768Hz
0	1	1	8192Hz
1	0	0	4096Hz
1	0	1	1
1	1	0	0
1	1	1	EXT1

表5.6 设置 b3－b5 位

b5	b4	b3	时钟源 B 的频率
0	0	0	2048Hz
0	0	1	1024Hz
0	1	0	256Hz

0	1	1	TMB1
1	0	0	4Hz
1	0	1	2Hz
1	1	0	1
1	1	1	EXT2

注：

若以 ClkA 作为门控信号，‘1’表示允许时钟源 B 信号通过，而‘0’则表示禁止时钟源 B 信号通过而停止 TimerA 的计数。如果时钟源 A 为‘1’，TimerA 时钟频率将取决于时钟源 B；如果时钟源 A 为‘0’，将停止 TimerA 的计数。

P_TimerB_Data(读/写)($700CH)

TimerB 的数据单元，用于向 16 位预置寄存器写入数据(预置计数初值)或从其中读取数据。写入数据后，计数器就会以设定的数值往上累加直至溢出。计数初值的计算方法和 TimerA 相同。

P_TimerB_Ctrl(写)($700DH)

TimerB 的控制单元(如表 5.7 所示)。用户可以通过设置该单元的第 0~2 位来选择 TimerB 的时钟源。设置第 6~9 位，TimerB 将输出不同频率的脉宽调制信号，即对脉宽占空比输出 BPWMO 进行控制。

表5.7 设置 P_TimerB_Ctrl 单元

b15 - b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
---	Output_pulse_ctrl				---			时钟源 C 选择位		

表5.8 占空比的设置

b9	b8	b7	b6	脉宽占空比(BPWMO)	TBON[1]
0	0	0	0	关断	0
0	0	0	1	1/16	1
0	0	1	0	2/16	1
0	0	1	1	3/16	1
0	1	0	0	4/16	1
0	1	0	1	5/16	1
0	1	1	0	6/16	1
0	1	1	1	7/16	1
1	0	0	0	8/16	1
1	0	0	1	9/16	1
1	0	1	0	10/16	1
1	0	1	1	11/16	1
1	1	0	0	12/16	1
1	1	0	1	13/16	1
1	1	1	0	14/16	1
1	1	1	1	TBOUT[2] 触发信号	1

注：

[1]：TBON 是 TimerB(BPWMO)的脉宽调制信号输出允许位，默认值为‘0’；

[2]：TBOUT 是 TimerB 的溢出信号，当 TimerB 的计数从 N 达到 0xFFFF 后(用户通过设置 P_TimerB_Data 单元指定 N 值)，发生计数溢出。产生的溢出信号可以作为 TimerB 的中断信号被送至中断控制系统；同时 N 值将被重新载入预置寄存器，使 Timer 重新开始计数。TBOUT 触发信号(TBOUT/2)的占空比为 50%，频率为 $F_{TBOUT}/2$，其它输入信号的频率为 $F_{TBOUT}/16$。

5.3 定时/计数器设置的 C 函数

SPCE061.lib 中提供了相应的 API 函数如下所示：

函数原型

void Set_TimerA_Data(unsigned int);
void Set_TimerB_Data(unsigned int);

功能说明　预置计数器初值

用法

Set_TimerA_Data(TimerA_Data);
Set_TimerB_Data(TimerB_Data);

参数

TimerA_Data = 0xFFFF – (Source A & Source B Frequency) / Desired Frequency
TimerB_Data = 0xFFFF – (Source C Frequency) / Desired Frequency

函数原型

unsigned int Get_TimerA_Data(void);
unsigned int Get_TimerB_Data(void);

功能说明　读计数器的值

用法

TimerA_Data = Get_TimerA_Data();
TimerB_Data = Get_TimerB_Data();

函数原型

void Set_TimerA_Ctrl(unsigned int);
void Set_TimerB_Ctrl(unsigned int);

功能说明　选择时钟源频率和占空比

用法

Set_TimerA_Ctrl(TimerA_Ctrl_Data);
Set_TimerB_Ctrl(TimerB_Ctrl_Data);

参数

TimerA_Ctrl_Data = Source A + Source B Frequency + Duty Cycle
TimerB_Ctrl_Data = Source C Frequency + Duty Cycle

Default Value →
TimerA_Ctrl_Data　= C_TimerADefault

TimerB_Ctrl_Data　= C_TimerBDefault

Source A Frequency →

C_SourceA_Fosc2	→	Fosc / 2
C_SourceA_Fosc256	→	Fosc / 256
C_SourceA_32768Hz	→	32768 Hz
C_SourceA_8192Hz	→	8192 Hz
C_SourceA_4096Hz	→	4096 Hz
C_SourceA_1	→	1
C_SourceA_0	→	0
C_SourceA_Ext1	→	EXT1　(IOB2)

Source B Frequency →

C_SourceB_2048Hz	→	2048 Hz
C_SourceB_1024Hz	→	1024 Hz
C_SourceB_256Hz	→	256 Hz
C_SourceB_TMB1	→	TMB1
C_SourceB_4Hz	→	4 Hz
C_SourceB_2Hz	→	2 Hz
C_SourceB_1	→	1
C_SourceB_Ext2	→	EXT2　(IOB3)

Source C Frequency →

C_SourceC_Fosc2	→	Fosc / 2
C_SourceC_Fosc256	→	Fosc / 256
C_SourceC_32768Hz	→	32768 Hz
C_SourceC_8192Hz	→	8192 Hz
C_SourceC_4096Hz	→	4096 Hz
C_SourceC_1	→	1
C_SourceC_0	→	0
C_SourceC_Ext1	→	EXT1　(IOB2)

函数原型

void Set_TimeBase(unsigned int);

功能说明　时基频率选择

用法

Set_TimeBase (TimerBase_Data);

参数

TimerBase_Data = TMB1 + TMB2 Frequency

Default Value →

TimerBase_Data　= C_TimeBaseDefault

TMB1 Frequency →

C_TMB1_8Hz	→ 8 Hz
C_TMB1_16Hz	→ 16 Hz
C_TMB1_32Hz	→ 32 Hz
C_TMB1_64Hz	→ 64 Hz

TMB2 Frequency →

C_TMB2_128Hz	→ 128 Hz
C_TMB2_256Hz	→ 256 Hz
C_TMB2_512Hz	→ 512 Hz
C_TMB2_1024Hz	→ 1024 Hz

函数原型

void Clear_TimeBase(void);

功能说明　时基计数器复位

用法

Clear_TimeBase();

5.4 定时/计数器的应用实例

5.4.1 时基频率的选择

例 5.1

选择 8Hz 的时基频率，进入中断后 IOB 口交替输出高低电平。通过示波器可观察到 IOB 口输出为方波。本例程由主程序 main.c 和中断服务程序 ISR.c 组成，其中与中断有关的部分请参见第六章中断的内容。

以下为 main.c 的代码：

```
#include  "SPCE061.H"
main()
{
    asm("INT OFF");

    *P_IOB_Dir=0xFFFF;              //output
    *P_IOB_Attrib=0xFFFF;
    *P_IOB_Data=0xFFFF;

    *P_TimeBase_Setup=0;            //timebase=8Hz
    *P_INT_Ctrl=C_IRQ6_TMB2;        //Setup interrupt

    __asm("INT IRQ");

    while(1)
```

```
        *P_Watchdog_Clear = C_WDTCLR;
}
```

以下为 ISR.c 的代码：

```
#include  "SPCE061.H"
unsigned int g_uiOutput=0xffff;
void IRQ6(void) __attribute__ ((ISR));
void IRQ6(void)
{
    unsigned int i;
    g_uiOutput=~g_uiOutput;          //reverse
    if(!(*P_INT_Ctrl&0x0001))
    {   //IRQ_TMB2
        *P_INT_Clear=0x0001;         //clear INT flag
    }
    else
    {   //IRQ_TMB1
        *P_IOB_Data=g_uiOutput;      //output
        i=100;
        while(i--);                  //delay
        *P_INT_Clear=0x0002;         //clear INT flag
    }
}
```

5.4.2 用 TimerA 产生方波

例 5.2

本例通过 TimerA 计数，每隔 0.5 秒，从 IOB 口相应的输出高或低电平，最终输出连续 1Hz 方波。

上文讲过，给 TimerA 预置一个初值，它会在此基础上进行加一计数，加到 0xffff 时会产生溢出，同时产生中断请求信号，送入中断控制器进行处理。给本例跟上例不同，上例的输出是在中断服务子程序中进行的，而本例没有中断服务程序可以响应，本例只是用到了 P_INT_Ctrl 寄存器的一个标志位而已。

那么，应该给 TimerA 预置一个多大的数呢？这里有一个公式

TimerA_Data = 0xFFFF – (Source A & Source B Frequency) / Desired Frequency

定时 0.5 秒，我们利用时钟源 B，选择 1024Hz 的频率。这样我们就可以得到应该预置的初值为 TimerA_Data＝0xFFFF –1024/2＝0xFDFF。

本例的程序代码如下：

```
#include  "SPCE061.H"
main()
{
    unsigned int uiOutput;
    uiOutput=0x0000;
```

```
    *P_IOB_Dir=0xFFFF;
    *P_IOB_Attrib=0xFFFF;
    *P_IOB_Data=0x0000;

    *P_INT_Ctrl=C_IRQ1_TMA;
    asm("INT OFF");

    *P_TimerA_Ctrl=C_SourceA_1+C_SourceB_1024Hz;    //TimerA:1024Hz
    *P_TimerA_Data=0xFDFF;                          //0.5 Second
    while(1)
    {
        *P_Watchdog_Clear = C_WDTCLR;
        if(*P_INT_Ctrl&C_IRQ1_TMA)
        {
            *P_INT_Clear=C_IRQ1_TMA;                //clear INT flag
            uiOutput ^= 0xFFFF;                     //reverse
            *P_IOB_Data=uiOutput;                   //output rectangle
        }
    }
}
```