

SH79F161B 用户指南

-带 10 位 ADC 的增强型 8051 微控制器



一、SH79F161B I/O 用户指南

1. 概述

SH79F161B 是一种高速高效率 8051 可兼容单片机。在同样振荡频率下，较之传统的 8051 芯片它有着运行更快速的优越特性。

SH79F161B 其 IO 特性为：

- ◆ 30个双向I/O端口
- ◆ I/O端口可与其它功能共享
- ◆ 4 个可选的开漏极 I/O 口

SH79F161B 提供 30 个位可编程双向 I/O 端口。端口数据在寄存器 Px 中。每个 I/O 口均有内部上拉电阻。端口控制寄存器 (PxCRy) 控制端口是作为输入或者输出。当端口作为输入时，每个 I/O 端口带有由 PxPCRy 控制的内部上拉电阻 (x = 0-3, y = 0-7)。

2. 控制寄存器

IO 模块使用的所有控制寄存器如下表所示：

类别	缩写符号	功能说明
模块控制	PxCRy (x=0-3, y=0-7)	IO 输入输出控制寄存器
	PxPCRy (x=0-3, y=0-7)	IO 上拉电阻控制寄存器
	P0OS	开漏极 I/O 控制寄存器
模块输出输入	Px.y (x=0-3, y=0-7)	IO 端口数据寄存器

针对每个寄存器的详细介绍，请参照“SH79F161B SPEC”

3. IO 设置

3.1. IO 输出设置

IO 作为输出模式，需要将其 PxCRy (x=0-3, y=0-7) 设置为 1，此时若设置 Px.y (x=0-3, y=0-7) 为 0，则 IO 输出低电平，低电平的值为 GND。若设置 Px.y (x=0-3, y=0-7) 为 1，则 IO 输出高电平，高电平的值为 VDD。

3.2. IO 输入设置

IO 作为输入模式，需要将其 PxCRy (x=0-3, y=0-7) 设置为 0，此时若设置 PxPCRy (x=0-3, y=0-7) 为 0，则 IO 处于输入 Floating 状态，若设置 PxPCRy (x=0-3, y=0-7) 为 1，则上拉电阻打开，IO 处于输入高的状态。IO 输入的高低之分是在 VDD/2 左右，具体参数请参考“SH79F161B SPEC”。

3.3. 开漏极 I/O 设置



SH79F161B 的 P0.2~P0.5 是可以设置为 N 沟道开漏输出或者 CMOS 推挽输出的,若用到此功能,请先将 P0CR2~P0CR5 设置为 1,输出状态,此时设置 P0OSx (x=2-5) 为 0, P0.2~P0.5 是 N 沟道开漏输出的,设置 P0OSx (x=2-5) 为 1, P0.2~P0.5 是 CMOS 推挽输出。若选择 P0.2~P0.5 为 N 沟道开漏输出模式,此时 P0.2~P0.5 是没有输出高的能力的,需要根据系统的需要外围加上上拉电阻,上拉电阻的阻值是实际应用而定。

4. 应用实例

4.1. 要求

将一个 0~5V 的三角波从 P3.0 输入, P3.0 的值通过 P1.0 输出

4.2. 例程

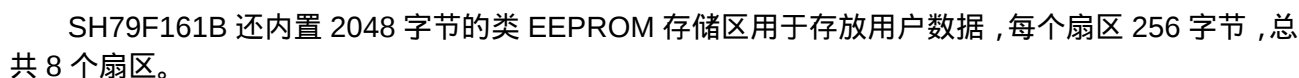
```
bit a ;
Void    main ( void )
{
    P3CR &= 0xFE;           //P3.0 输入
    P3PCR |= 0x01;          //P3.0~P3.5 上拉电阻打开
    P1CR |= 0x01;           // P1.0 输出
    while(1)
    {
        a = P3.0;
        P1.0 = a;           // P1.0 输出值为 P3.0 输入值
    }
}
```



1. 概述

SH79F161B 其 Flash&EEPROM 特性为：

- 类 EEPROM 区：至少 100,000 次





2. 控制寄存器

Flash&EEPROM 模块使用的所有控制寄存器如下表所示：

类别	缩写符号	功能说明
模块控制	XPAGE	编程用地址选择寄存器
	IB_OFFSET	编程用地址偏移寄存器
	IB_DATA	编程用数据寄存器
	IB_CON1	SSP 型选择寄存器
	IB_CON2	SSP 流程控制寄存器 1
	IB_CON3	SSP 流程控制寄存器 2
	IB_CON4	SSP 流程控制寄存器 3
	IB_CON5	SSP 流程控制寄存器 4
	FLASHCON	访问控制寄存器

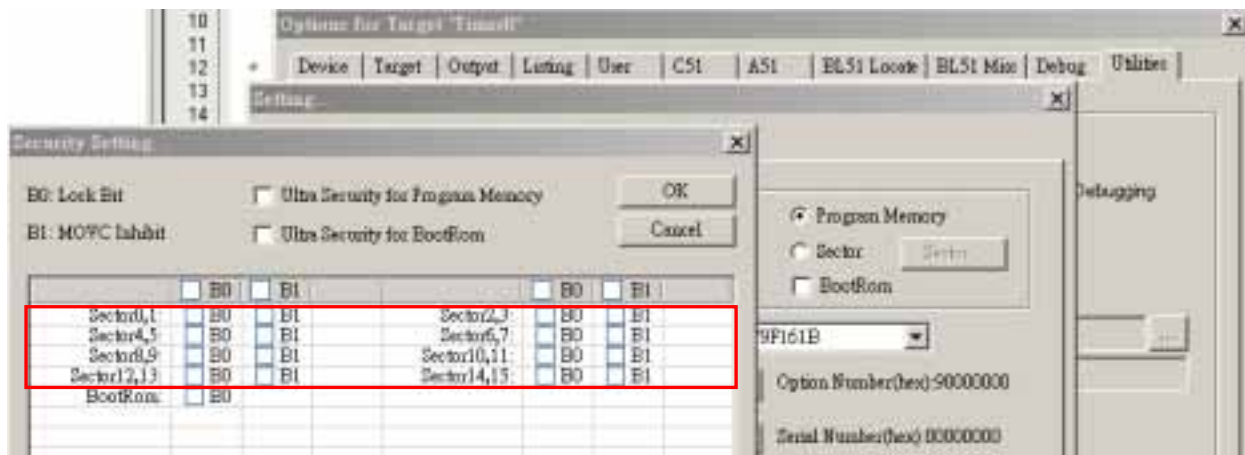
针对每个寄存器的详细介绍，请参照“SH79F161B SPEC”

3. SSP 编程设置

3.1. Flash SSP 编程设置

若使用 SSP 功能对 Flash 进行擦除或者编写，首先需要将 FLASHCON 的 FAC 位置 0，然后设置相应的 XPAGE、IB_OFFSET、IB_DATA（需要操作的地址和数据），再设置 IB_CON1，若 IB_CON1 为 0xE6，则相应的操作为扇区擦除，若 IB_CON1 为 0x6E，则相应的操作为存储单元编程。之后再设置 IB_CON2 为 0x05，IB_CON3 为 0x0A，IB_CON4 为 0x06，IB_CON5 为 0x09 完成 Flash 的整个操作。需要注意，SSP 扇区擦除操作不可选择 SSP 程序所在扇区作为操作对象。

特别地，若需要对特定扇区进行保护，防止 SSP 误操作，可在 Keil 中通过“Options->Utilities->Security”路径设置 Sector 加密。其中，B0 加密对应 Flash 代码保护模式 0；B1 加密对应 Flash 代码保护模式 1。Flash 代码保护模式详细介绍，请参照“SH79F161B SPEC”。





若使用 SSP 功能对 EEPROM 进行擦除或者编写，首先需要将 FLASHCON 的 FAC 位置 1，然后设置相应的 XPAGE，IB_OFFSET，IB_DATA（需要操作的地址和数据），再设置 IB_CON1，若 IB_CON1 为 0xE6，则相应的操作为扇区擦除，若 IB_CON1 为 0x6E，则相应的操作为存储单元编程。之后再设置 IB_CON2 为 0x05，IB_CON3 为 0x0A，IB_CON4 为 0x06，IB_CON5 为 0x09 完成 EEPROM 的整个操作。Flash 控制流程图如 3.3 所示。

```

graph TD
    Start([Start]) --> SetFlashCON[Set FLASHCON]
    SetFlashCON --> SetIB[Set IB_OFFSET<br/>Set XPAGE<br/>Set IB_DATA<br/>Set IB_CON1]
    SetIB --> S0((S0))
    S0 --> IB_CON2_5H[IB_CON2[3:0]≠5H]
    IB_CON2_5H --> S0
    IB_CON2_5H --> S1((S1))
    S1 --> SetIB_CON2[Set IB_CON2[3:0]=5H]
    SetIB_CON2 --> S1
    S1 --> IB_CON3_AH[IB_CON3≠AH]
    IB_CON3_AH --> S1
    IB_CON3_AH --> S2((S2))
    S2 --> SetIB_CON3[Set IB_CON3=AH]
    SetIB_CON3 --> S2
    S2 --> IB_CON4_9H[IB_CON4≠9H]
    IB_CON4_9H --> S2
    IB_CON4_9H --> S3((S3))
    S3 --> SetIB_CON4[Set IB_CON4=9H]
    SetIB_CON4 --> S3
    S3 --> IB_CON5_6H[IB_CON5≠6H]
    IB_CON5_6H --> S3
    IB_CON5_6H --> S4((S4))
    S4 --> SetIB_CON5[Set IB_CON5=6H]
    SetIB_CON5 --> S4
    S4 --> IB_CON1_6EH[IB_CON1=6EH<br/>&IB_CON2[3:0]=5H<br/>&IB_CON3=AH<br/>&IB_CON4=9H<br/>&IB_CON5=6H]
    IB_CON1_6EH --> Programming[Programming]
    Programming --> Reset[Reset IB_CON1-5]
    Reset --> S0
    S0 --> ELSE[ELSE]
    ELSE --> S0
    ELSE --> S1
    ELSE --> S2
    ELSE --> S3
    ELSE --> S4
    ELSE --> SectorErase[Sector Erase]
    SectorErase --> Reset
  
```

The flowchart illustrates the sequence for Sector Erase and Programming. It begins with setting the FLASHCON and then the IB parameters (IB_OFFSET, XPAGE, IB_DATA, IB_CON1). The process then enters a state machine with states S0 through S4. S0 checks IB_CON2[3:0] and either loops back or proceeds to S1. S1 checks IB_CON3 and either loops back or proceeds to S2. S2 checks IB_CON4 and either loops back or proceeds to S3. S3 checks IB_CON5 and either loops back or proceeds to S4. S4 sets IB_CON5=6H and then checks for the final configuration (IB_CON1=6EH, IB_CON2[3:0]=5H, IB_CON3=AH, IB_CON4=9H, IB_CON5=6H). If this configuration is met, it proceeds to Programming. If not, it branches to either Sector Erase (which resets IB_CON1-5 and loops back to S0) or to one of the other states (S0, S1, S2, S3, S4) based on the ELSE condition.



4. 应用实例

4.1. 要求

先将整个 Flash 区域整体擦除，再 sector0 写入 00，sector1 写入 01，依次类推至 sector15 写入 0F。

4.2. 例程

```
#include <sh79f161B.h>
#include <intrins.h>
#include <absacc.h>
#define uchar unsigned char
#define uint unsigned int
#define NOP _nop_();

uchar i,j,m,n;
uint code *p;
uchar count = 0;
uchar Ssp_flag;

void main (void)
{
    uchar temp;
    CLKCON = 0;
    m = 0;
    n = 0;
    Ssp_flag = 0xA5;
    /*****sector erase*****/
    EA = 0;
    for(i=0x00;i<0x40;i++)
    {
        FLASHCON = FLASHCON&0xFE;
        NOP
        EA = 0;
        IB_CON1 = 0xE6;
        IB_CON2 = 0x05;
        IB_CON3 = 0x0A;
        IB_CON4 = 0x09;
        if(!(Ssp_flag==0xA5)) goto Error;
    }
}
```



```
XPAGE = i&0xFC;
IB_CON5 = 0x06;
NOP
NOP
NOP
NOP
NOP
for(j=0x00;j<0xFF;j++)
{
    p = (uint)(i<<8)+j;
    temp = *p;
}
}
/*****sector write*****/
temp = 0x00;
for(m=0x04;m<0x40;m++)
{
    for(n=0x00;n<0xFF;n++)
    {
        temp = m;
        FLASHCON = FLASHCON&0xFE;
        NOP
        EA = 0;
        IB_OFFSET = n;
        IB_DATA = temp;
        IB_CON1 = 0x6E;
        IB_CON2 = 0x05;
        IB_CON3 = 0x0A;
        IB_CON4 = 0x09;
        if(!(Ssp_flag==0xA5)) goto Error;
        XPAGE = m;
        IB_CON5 = 0x06;
        NOP
        NOP
        NOP
        NOP
```



```
        NOP
    }
}
while(1);
Error:
    Ssp_flag = 0;
    IB_CON1 = 0x00;
    IB_CON2 = 0x00;
    IB_CON3 = 0x00;
    IB_CON4 = 0x00;
    IB_CON5 = 0x00;
    XPAGE = 0x00;
    FLASHCON = 0x00;
    EA = 0;
    while(1);
}
```



5. 编程注意事项

为确保顺利完成SSP编程，用户软件必须按以下步骤设置：

(1) 用于代码/数据编程：

1. 关闭中断；
2. 根据地址设置 XPAGE，IB_OFFSET；
3. 按编程需要，设置 IB_DATA；
4. 按照顺序设置 IB_CON1 - 5；
5. 添加 4 个 NOP 指令；
6. 开始编程，CPU 将进入 IDLE 模式；烧写完成后自动退出 IDLE 模式；
7. 如需继续写入数据，跳转至第 2 步；
8. XPAGE 寄存器清 0，恢复中断设置。

(2) 用于扇区擦除：

1. 关闭中断；
2. 按相应的扇区设置 XPAGE；
3. 按照顺序设置 IB_CON1 - 5；
4. 添加 4 个 NOP 指令；
5. 开始擦除，CPU 将进入 IDLE 模式；擦除完成后自动退出 IDLE 模式；
6. 如需要继续擦除数据，跳转至第 2 步；
7. XPAGE 寄存器清 0，恢复中断设置。

(3) 读取：

使用“MOVC A,@A+DPTR”或者“MOVC A,@A+PC”指令。

(4) 对于类 EEPROM 区域

对于类 EEPROM 的操作类似于 Flash 的操作，即类似上述(1)/(2)/(3)部分的描述。区别在于：

1. 在对类 EEPROM 进行擦除、写或读之前，应首先将 FLASHCON 寄存器的最低位 FAC 位置 1。
2. 类 EEPROM 的扇区为 256 字节，而不是 1024 字节

注意：

1. 系统时钟不得低于 200kHz 以确保 FLASH 的正常编程
2. 当不需对类 EEPROM 操作时，必须将 FAC 位清 0



6. FLASH/类 EEPROM 的烧写/擦除的超级抗干扰措施

- 1) 在程序下载时，选择代码选项中的“enable LVR function”。
- 2) 设置扇区时，对 XPAGE 写立即数。
- 3) 在调用烧写或擦除函数前，置一个标志，比如 0A5H；在烧写或擦除函数中判断此标志是否为 0A5H，否则清零此标志，退出函数。

下面类 EEPROM 示例仅供参考：

C 文件：

```
    uchar Ssp_flag;

    /*****SSP Erase*****/
    Ssp_flag = 0xA5;
    EA = 0;
    FLASHCON = 0x01;
    IB_CON1 = 0xE6;
    IB_CON2 = 0x05;
    IB_CON3 = 0x0A;
    IB_CON4 = 0x09;
    if(!(Ssp_flag==0xA5)) goto Error;
    XPAGE = 0x01;      //XPAGE 写立即数
    IB_CON5 = 0x06;
    NOP
    NOP
    NOP
    NOP

    /*****sector write*****/
    Ssp_flag = 0x5A;
    EA = 0;
    FLASHCON = 0x01;
    IB_DATA = data;      //data 为烧写数据
    IB_OFFSET = addr;    //addr 为烧写地址低位
    IB_CON1 = 0x6E;
    IB_CON2 = 0x05;
    IB_CON3 = 0x0A;
    IB_CON4 = 0x09;
```



```
if(!(Ssp_flag==0x5A)) goto Error;
XPAGE = 0x01;          //烧写地址高位，写立即数
IB_CON5 = 0x06;
NOP
NOP
NOP
NOP
while(1);
Error:
    Ssp_flag = 0;
    IB_CON1 = 0x00;
    IB_CON2 = 0x00;
    IB_CON3 = 0x00;
    IB_CON4 = 0x00;
    IB_CON5 = 0x00;
    XPAGE = 0x00;
    FLASHCON = 0x00;
    EA = 0;
    while(1);
```



三、SH79F161B Timer0/1 用户指南

1. 概述

SH79F161B 是一种高速高效率 8051 可兼容单片机。在同样振荡频率下，较之传统的 8051 芯片它有着运行更快速的优越特性。

SH79F161B 其 Timer0/1 特性为：

- ◆ 定时器0/1兼容标准的8051
- ◆ 定时器0/1增加了比较输出功能和时钟源分频功能

SH79F161B 的定时器 0/1 可配置为 13 位、16 位、8 位自动重载和双 8 位（仅 Timer0 支持双 8 位定时器方式）定时器/计数器四种方式。通过计数器/定时器方式寄存器（TMOD）的方式选择位 Mx1 - Mx0，选择定时器工作方式。

两个数据寄存器[THx & TLx (x = 0,1)]可作为一个 16 位寄存器来访问。它们由寄存器 TCON 和 TMOD 控制。IEN0 寄存器的 ET0 和 ET1 位置 1 能允许定时器 0 和定时器 1 中断。（详见 SH79F161B SPEC）

2. 控制寄存器

Timer0/1 模块使用的所有控制寄存器如下表所示：

类别	缩写符号	功能说明
模块控制	TCON	控制寄存器
	TCON1	控制寄存器 1
	TMOD	模式控制寄存器
	TL0	Timer0 低位计数器
	TH0	Timer0 高位计数器
	TL1	Timer1 低位计数器
	TH1	Timer1 高位计数器

针对每个寄存器的详细介绍，请参照“SH79F161B SPEC”

3. Timer0/1 设置

3.1. 方式 0 设置

在方式 0 中，定时器 x (x = 0,1) 为 13 位计数器/定时器。THx 寄存器存放 13 位计数器/定时器的高 8 位，TLx 存放低 5 位（TLx.4-TLx.0）。TLx 的高三位（TLx.7-TLx.5）是不确定的，在读取时应该被忽略。当 13 位定时器寄存器递增，溢出时，系统置起定时器溢出标志 TFX。如果定时器 x 中断被允许，将会产生一个中断。C/Tx 位选择计数器/定时器的时钟源。

如果 C/Tx = 1，定时器 x 输入引脚（Tx）的电平从高到低跳变，使定时器 x 数据寄存器加 1。如果 C/Tx = 0，选择系统时钟为定时器 x 的时钟源。

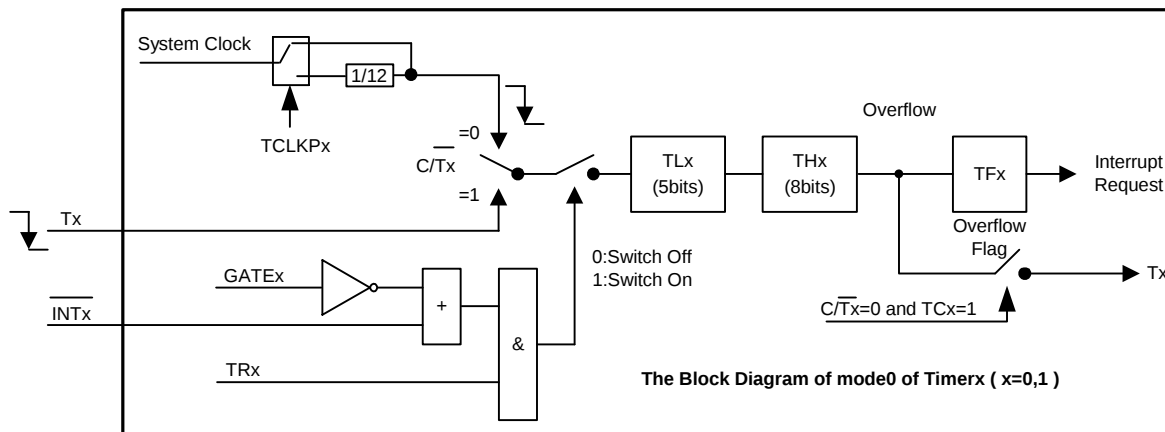
当 GATEx = 0 或 GATEx = 1 且输入信号 $\overline{\text{INTx}}$ 有效时，TRx 置 1 打开定时器。GATEx 置 1 允许定时器由外部输入信号 $\overline{\text{INTx}}$ 控制，便于测量 $\overline{\text{INTx}}$ 的正脉冲宽度。TRx 位置 1 不强行复位定时器，这意味着如果 TRx 置 1，定时器寄存器将从上次 TRx 清 0 时的值开始计数。所以在允许定时器之前，应该



设定定时器寄存器的初始值。

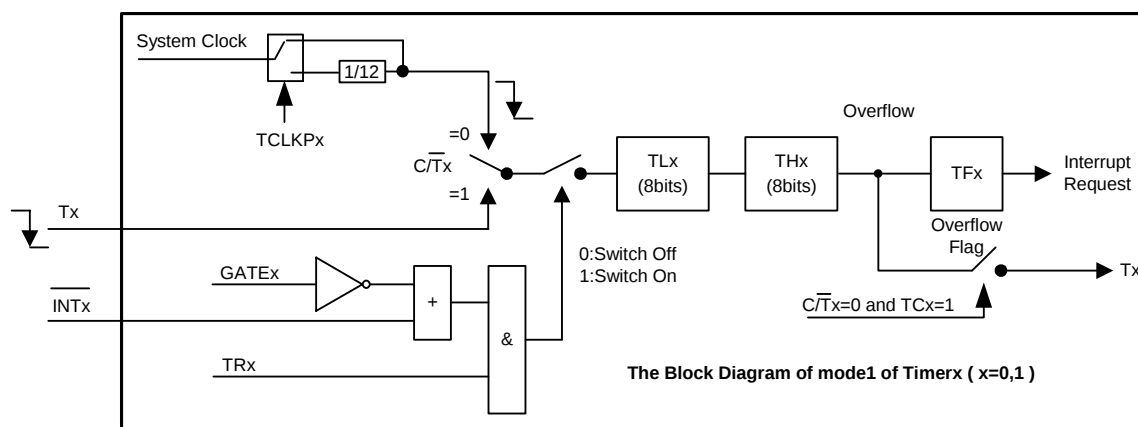
可配置寄存器 TCON1 中的 TCLKP_x ($x = 0, 1$) 位选择系统时钟或系统时钟的 1/12 作为定时器 x ($x = 0, 1$) 的时钟源。

当作为定时器应用时,可配置寄存器 TCON1 中的 TC0/1 位使定时器 0/1 溢出时 T0/T1 脚自动翻转。如果 TC0/1 被置 1, T0/T1 引脚自动设置为输出。



3.2. 方式 1 设置

除了使用 16 位定时器/计数器之外,方式 1 的运行与方式 0 一致。打开和配置计数器/定时器也如同方式 0。



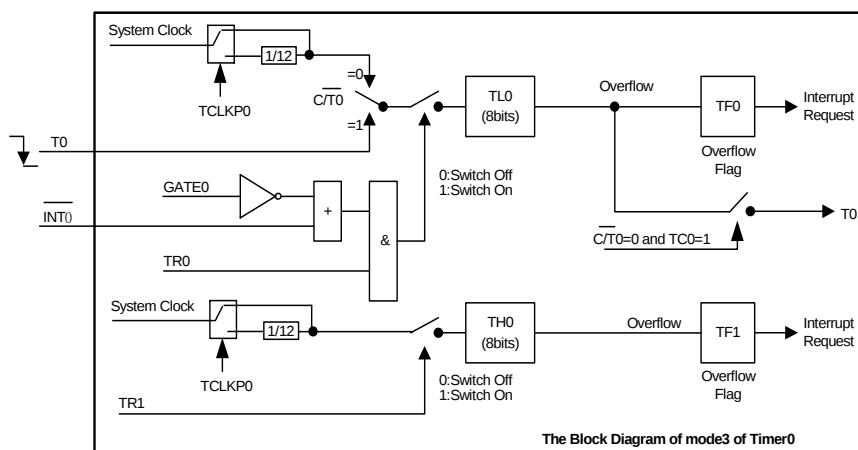
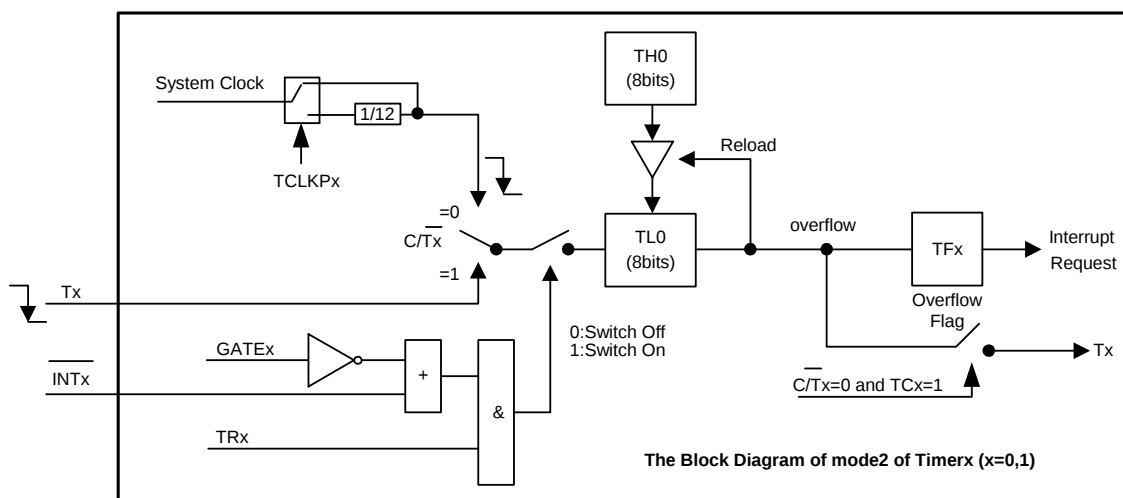
3.3. 方式 2 设置

方式 2 中,定时器 x 是 8 位自动重载计数器/定时器。TL_x 存放计数值,TH_x 存放重载值。当在 TL_x 中的计数器溢出至 0x00 时,置起定时器溢出标志 TF_x,寄存器 TH_x 的值被重载入寄存器 TL_x 中。如果定时器中断使能,当 TF_x 置 1 时将产生一个中断。而在 TH_x 中的重载值不会改变。在允许定时器正确计数开始之前,TL_x 必须初始化为所需的值。

除了自动重载功能外,方式 2 中的计数器/定时器的使能和配置与方式 1 和 0 是一致的。

可配置寄存器 TCON1 中的 TCLKP_x ($x = 0, 1$) 位选择系统时钟或系统时钟的 1/12 作为定时器 x ($x = 0, 1$) 的时钟源。

当作为定时器应用时,可配置寄存器 TCON1 中的 TC0/1 位使定时器 0/1 溢出时 T0/T1 脚自动翻转。如果 TC0/1 被置 1, T0/T1 引脚自动设置为输出。





4. 应用实例

4.1. 方式 0 例程

```
#include <sh79f161B.h>
#include <intrins.h>

void Port_init(void)
{
    P1 = 0x00;
    P1CR = 0x01; //P1.0 设置为输出口
}

void init_timer0()
{
    //13 位定时器方式
    IEN0 = 0x82;           //打开定时器 0 中断和总中断
    TMOD = 0x00;           //设置 Timer0 工作方式为方式 0
    TCON1 = 0x00;          //将系统时钟作为定时器 0 的时钟源
    TL0 = 0x11;            //设置 Timer0 定时器初值为 0x1B31
    TH0 = 0xD9;
    TR0 = 1;               //打开 Timer0
}

void main()
{
    Port_init();
    init_timer0();
    while(1);
}

void INT_TIMER0(void) interrupt 1
{
    TF0 = 0;
    P1_0 = ~P1_0;          //P1.0 的翻转周期为 200us , 即频率为 5kHz
    TL0 = 0x11;            //设置 Timer0 定时器初值为 0x1B31
    TH0 = 0xD9;
}
```



4.2. 方式 1 例程

```
#include <sh79f161B.h>
#include <intrins.h>

void Port_init(void)
{
    P1 = 0x00;
    P1CR = 0x01; //P1.0 设置为输出口
}

void init_timer0()
{
    //16 位定时器方式
    IEN0 = 0x82;           //打开定时器 0 中断和总中断
    TMOD = 0x01;           //设置 Timer0 工作方式方式 1
    TCON1 = 0x00;          //将系统时钟作为定时器 0 的时钟源
    TL0 = 0x31;            //设置 Timer0 定时器初值为 0xFB31
    TH0 = 0xFB;
    TR0 = 1;               //打开 Timer0
}

void main()
{
    Port_init();
    init_timer0();
    while(1);
}

void INT_TIMER0(void) interrupt 1
{
    TF0 = 0;
    P1_0 = ~P1_0;          //P1.0 的翻转周期为 200us，即频率为 5kHz
    TL0 = 0x31;            //设置 Timer0 定时器初值为 0xFB31
    TH0 = 0xFB;
}
```



4.3. 方式 2 例程

```
#include <sh79f161B.h>
#include <intrins.h>
void Port_init(void)
{
    P1 = 0x00;
    P1CR = 0x01; //P1.0 设置为输出口
}

void init_timer0()
{
    //8 位自动重载定时器方式
    IEN0 = 0x82;           //打开定时器 0 中断和总中断
    TMOD = 0x02;           //设置 Timer0 工作方式为方式 2
    TCON1 = 0x00;          //将系统时钟作为定时器 0 的时钟源
    TL0 = 0x09;            //设置 Timer0 定时器初值为 0x09，重载值 0x09
    TH0 = 0x09;
    TR0 = 1;               //打开 Timer0
}

void main()
{
    Port_init();
    init_timer0();
    while(1);
}

void INT_TIMER0(void) interrupt 1
{
    TF0 = 0;
    P1_0 = ~P1_0;          //P1.0 的翻转周期为 40us，即频率为 25kHz
}
```

4.4. 方式 3 例程

```
#include <sh79f161B.h>
#include <intrins.h>
void Port_init(void)
```



```
{
    P1 = 0x00;
    P1CR = 0x03; //P1.0 , P1.1 设置为输出口
}

void init_timer0()
{
    //双 8 位定时器方式
    IEN0 = 0x8A;           //打开定时器 0 中断，定时器 1 中断和总中断
    TMOD = 0x03;           //设置 Timer0 工作方式方式为方式 3
    TCON1 = 0x00;          //将系统时钟作为定时器 0 的时钟源
    TL0 = 0x09;             //设置第一个 8 位定时器初值为 0x09
    TH0 = 0x84;             //设置第二个 8 位定时器初值为 0x84
    TR0 = 1;               //启动第一个 8 位定时器
    TR1 = 1;               //启动第二个 8 位定时器
}

void main()
{
    Port_init();
    init_timer0();
    while(1);
}

void INT_TIMER0(void) interrupt 1
{
    TF0 = 0;
    P1_0 = ~P1_0;          //P1.0 的翻转周期为 50us，即频率为 20kHz
    TL0 = 0x09;            //设置第一个 8 位定时器初值为 0x09
}

void INT_TIMER1(void) interrupt 3
{
    TF1 = 0;
    P1_1 = ~P1_1;          //P1.1 的翻转周期为 25us，即频率为 40kHz
    TH0 = 0x84;            //设置第二个 8 位定时器初值为 0x84
}
```



四、SH79F161B Timer2 用户指南

1. 概述

SH79F161B 是一种高速高效率 8051 可兼容单片机。在同样振荡频率下，较之传统的 8051 芯片它有着运行更快速的优越特性。

SH79F161B 其 Timer2 特性为：

- ◆ 定时器2兼容标准的8052，且有递增递减计数和可编程输出功能

定时器 2 有 4 种工作方式：捕获/重载，带递增或递减计数器的自动重载方式，波特率发生器和可编程时钟输出。RCLK，TCLK 和 CP/RL2 的组合能选择这些方式。如下所示：

C/T2	T2OE	DCEN	TR2	CP/RL	RCLK	TCLK	方式	
X	0	X	1	1	0	0	0	16 位捕获
X	0	0	1	0	0	0	1	16 位自动重载定时器
X	0	1	1	0	0	0		
X	0	X	1	X	1	X	2	波特率发生器
					X	1		
0	1	X	1	X	0	0	3	只用于可编程时钟
					1	X	3	带波特率发生器的可编程时钟输出
					X	1		
1	1	X	1	X	X	X		不推荐使用
X	X	X	0	X	X	X	X	定时器 2 停止，T2EX 通路仍旧允许

两个数据寄存器（TH2 和 TL2）串联后可作为一个 16 位寄存器来访问，由寄存器 T2CON 和 T2MOD 控制。设置 IEN0 寄存器中的 ET2 位能允许定时器 2 中断。（详见 SH79F161B SPEC）

定时器 2 的 C/T2 选择系统时钟（定时器）或外部引脚 T2（计数器）作为定时器时钟输入。通过所选的引脚设置 TR2 允许定时器 2/计数器 2 数据寄存器计数。

2. 控制寄存器

Timer2 模块使用的所有控制寄存器如下表所示：

类别	缩写符号	功能说明
模块控制	T2CON	控制寄存器
	T2MOD	模式控制寄存器
	RCAP2L	重载/捕获数据低位
	RCAP2H	重载/捕获数据高位
	TL2	低位计数器
	TH2	高位计数器

针对每个寄存器的详细介绍，请参照“SH79F161B SPEC”

3. Timer2 设置

3.1. 方式 0 设置

Timer2 的方式 0 为 16 位捕获模式，需要先将 T2CON 中的 CP/RL 为置 1，然后再 T2CON 的 EXEN2 位有两个选项。



如果 EXEN2 = 0，定时器 2 作为 16 位定时器或计数器，如果 IET2 被允许的话，定时器 2 能设置 TF2 溢出产生一个中断。

如果 EXEN2 = 1，定时器 2 执行相同操作，但是在外部输入 T2EX 上的下降沿也能引起在 TH2 和 TL2 中的当前值分别被捕获到 RCAP2H 和 RCAP2L 中，此外，在 T2EX 上的下降沿也能引起在 T2CON 中的 EXF2 被设置。如果 IET2 被允许，EXF2 位也像 TF2 一样也产生一个中断。

可配置寄存器 T2MOD 中的 TCLKP2 位选择系统时钟或系统时钟的 1/12 作为定时器 2 的时钟源。

3.2. 方式 1 设置

Timer2 的方式 1 为 16 位自动重载定时器，需要先将 T2CON 中的 CP/RL 为置 0。

在 16 位自动重载方式下，定时器 2 可以被选为递增计数或递减计数。这个功能通过 T2MOD 中的 DCEN 位（递减计数允许）选择。系统复位后，DCEN 位复位值为 0，定时器 2 默认递增计数。当设置 DCEN 时，定时器 2 递增计数或递减计数取决于 T2EX 引脚上的电平。

当 DCEN = 0，通过在 T2CON 中的 EXEN2 位选择两个选项。

如果 EXEN2 = 0，定时器 2 递增到 0FFFFH，在溢出后置起 TF2 位，同时定时器自动将用户软件写好的寄存器 RCAP2H 和 RCAP2L 的 16 位值装入 TH2 和 TL2 寄存器。

如果 EXEN2 = 1，溢出或在外部输入 T2EX 上的下降沿都能触发一个 16 位重载，置起 EXF2 位。如果 IET2 被使能，TF2 和 EXF2 位都能产生一个中断。

可配置寄存器 T2MOD 中的 TCLKP2 位选择系统时钟或系统时钟的 1/12 作为定时器 2 的时钟源。

注意：当 EXEN2 = 1，C/T2 = 1（选择 T2 引脚时钟），且定时器 [TH2, TL2] 设置为 0FFFFH 时，T2 无外灌时钟，外部输入 T2EX 上的下降沿会触发 TF2 和 EXF2 全部置位。

设置 DCEN 位允许定时器 2 递增计数或递减计数。当 DCEN = 1 时，T2EX 引脚控制计数的方向，而 EXEN2 控制无效。

T2EX 置 1 可使定时器 2 递增计数。定时器向 0FFFFH 溢出，然后设置 TF2 位。溢出也能分别引起 RCAP2H 和 RCAP2L 上的 16 位值重载入定时器寄存器。

T2EX 清 0 可使定时器 2 递减计数。当 TH2 和 TL2 的值等于 RCAP2H 和 RCAP2L 的值时，定时器溢出。置起 TF2 位，同时 0FFFFH 重载入定时器寄存器。

无论定时器 2 溢出，EXF2 位都被用作结果的第 17 位。在此工作方式下，EXF2 不作为中断标志。

可配置寄存器 T2MOD 中的 TCLKP2 位选择系统时钟或系统时钟的 1/12 作为定时器 2 的时钟源。

3.3. 方式 2 设置

Timer2 的方式 2 是波特率发生器，此功能介绍请参照 Uart0 章节。

3.4. 方式 3 设置

Timer2 的方式 3 是可编程始终输出，T2 (P3.1) 可以编程输出 50% 的占空比时钟周期。清 C/T2 位和置 T2OE 位，使定时器 2 作为时钟发生器。TR2 位启动和中止定时器。

可配置寄存器 T2MOD 中的 TCLKP2 位选择系统时钟或系统时钟的 1/12 作为定时器 2 的时钟源。在这种方式中，T2 输出占空比为 50% 的时钟：

$$\text{Clock Out Frequency} = \frac{1}{2 \times 2} \times \frac{f_{\text{sys}}}{65536 - [\text{RCAP2H}, \text{RCAP2L}]} ; \text{TCLKP2} = 0$$

$$\text{Clock Out Frequency} = \frac{1}{2 \times 2 \times 12} \times \frac{f_{\text{sys}}}{65536 - [\text{RCAP2H}, \text{RCAP2L}]} ; \text{TCLKP2} = 1$$

定时器 2 溢出不产生中断，所以定时器 2 可以同时以相同频率用作波特率发生器和时钟输出。



4. 应用实例

4.1. 方式 0 例程

```
#include <sh79f161B.h>
#include <intrins.h>
#include <absacc.h>
unsigned int Val = 0;
void main (void)
{
    CLKCON = 0;
    P0CR = 0x02;           //P0.1 输出
    T2CON = 0x09;          //16 位捕获 T2EX 处的下降沿
    T2MOD = 0x00;          //Timer2 时钟采用系统时钟
    TL2=0x00;
    TH2=0x00;
    EA = 1;                //使能中断
    ET2 = 1;               //使能 Timer2 中断
    TR2 = 1;               //启动 Timer2
    TF2 = 0;               //TF 清零
    while(1);
}

void timer2_isr(void) interrupt 5
{
    if(TF2 == 1)
    {
        TF2 = 0;
    }

    if(T2CON&0x40)
    {
        T2CON &= 0xBF;    //两个下降沿之间的时间，即 T2EX 的输入波形的周期可以推算
        Val=RCAP2H*0x100+RCAP2L;
        P0_1=~P0_1;       //P0.1 翻转的周期和 T2EX 的输入波形的周期一致
    }
}
```



4.2. 方式 1 例程

```
#include <SH79F161B.H>
#include <intrins.H>

voidPort_init(void)
{
    P1 = 0x00;
    P1CR = 0x01; //P1.0 as outflag
}

voidTimer2_init(void)
{
    CLKCON = 0x00; //sys clock 12.3MHz
    T2CON = 0x00;
    T2MOD = 0x00;
    TL2 = 0x60; // 5ms interrupt
    TH2 = 0xEA;
    RCAP2L = 0x60;
    RCAP2H = 0xEA;
}

voidmain(void)
{
    Port_init();
    Timer2_init();
    EA = 1;
    ET2 = 1; //enable timer2 interrupt
    TR2 = 1; //enable timer2 count
    while(1);
}

voidtimer2_int(void) interrupt 5
{
    TF2 = 0;
    P1_0 = ~P1_0; //P1.0 的翻转周期为 10ms，即频率为 100Hz
}
```



4.3. 方式 3 例程

```
#include <SH79F161B.H>
#include <intrins.H>

void Timer2_init(void)
{
    CLKCON = 0x00;    //sys clock 12.3MHz
    T2CON = 0x00;
    T2MOD = 0x02;    // Mode3 , Program output clock
    RCAP2L = 0x48;
    RCAP2H = 0xf4;    //output clock frequency is 1KHz
}

void main (void)
{
    Timer2_init();
    TR2 = 1;          //enable Timer2 count
    while(1);          //P3.1(T2) output clock frequency is 1KHz
}
```



五、SH79F161B SPI 用户指南

1. 概述

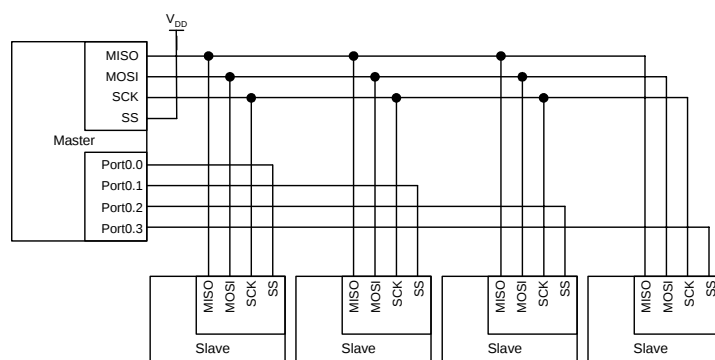
SH79F161B 是一种高速高效率 8051 可兼容单片机。在同样振荡频率下，较之传统的 8051 芯片它有着运行更快速的优越特性。

SH79F161B 其 SPI 模块特性为：

- ◆ 全双工，三线同步传输
- ◆ 主从机操作
- ◆ 6个可编程主时钟频率
- ◆ 极性相位可编程的串行时钟
- ◆ 带MCU中断的主模式故障出错标志
- ◆ 写入冲突标志保护
- ◆ 可选择LSB或MSB传输

串行外部设备接口（SPI）是一种高速串行通信接口，允许 MCU 与外围设备（包括其它 MCU）进行全双工，同步串行通讯。

下图所示即为典型的由一个主设备和若干从属外部设备组成的 SPI 总线网络，主设备通过 3 条线连接所有从设备，主设备控制连接从属设备 SS 引脚的 4 个并行端口来选中其中一个从属设备进行通讯。



控制寄存器

SPI 模块使用的所有控制寄存器如下表所示：

类别	缩写符号	功能说明
	SPCON	控制寄存器
	SPSTA	状态寄存器
	SPDAT	数据寄存器

针对每个寄存器的详细介绍，请参照“SH79F161B SPEC”

2. SPI 设置

2.1. 波特率设置

在主模式下，SPI 的波特率有六种可选择的频率，分别是内部时钟的 4，8，16，32，64 或 128



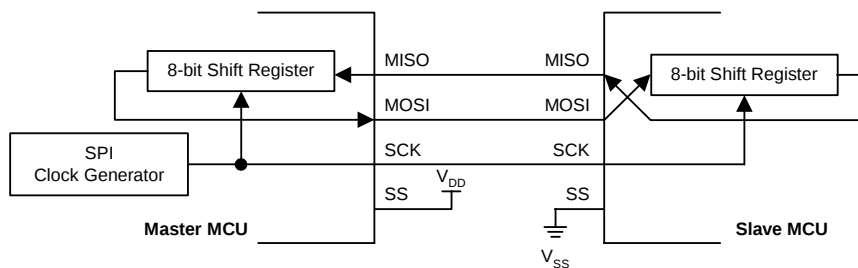
分频，可以通过设定 SPCON 寄存器的 SPR[2:0]位进行选择。

2.2. 工作模式设置

SPI 可配置为主模式或从模式中的一种。SPI 模块的配置和初始化通过设置 SPCON 寄存器（串行外围设备控制寄存器）和 SPSTA（串行外围设备状态寄存器）来完成。配置完成后，通过设置 SPCON，SPSTA，SPDAT（串行外围设备数据寄存器）来完成数据传送。

在 SPI 通讯期间，数据同步地被串行的移进移出。串行时钟线（SCK）使两条串行数据线（MOSI 和 MISO）上数据的移动和采样保持同步。从设备选择线（SS）可以独立地选择 SPI 从属设备；如果从设备没有被选中，则不能参与 SPI 总线上的活动。

当 SPI 主设备通过 MOSI 线传送数据到从设备时，从设备通过 MISO 线发送数据到主设备作为响应，这就实现了在同一时钟下数据发送和接收的同步全双工传输。发送移位寄存器和接收移位寄存器使用相同的特殊功能器地址，对 SPI 数据寄存器 SPDAT 进行写操作将写入发送移位寄存器，对 SPDAT 寄存器进行读操作将获得接收移位寄存器的数据。



全双工主从互联图

主模式

(1) 模式启动

SPI 主设备控制 SPI 总线上所有数据传送的启动。当 SPCON 寄存器中的 MSTR 位置 1 时，SPI 在主模式下运行，只有一个主设备可以启动传送。

(2) 发送

在 SPI 主模式下，写一个字节数据到 SPI 数据寄存器 SPDAT，数据将会写入发送移位缓冲器。如果发送移位寄存器已经存在一个数据，那么主 SPI 产生一个 WCOL 信号以表明写入太快。但是在发送移位寄存器中的数据不会受到影响，发送也不会中断。另外如果发送移位寄存器为空，那么主设备立即按照 SCK 上的 SPI 时钟频率串行地移出发送移位寄存器中的数据到 MOSI 线上。当传送完毕，SPSTA 寄存器中的 SPIF 位被置 1。如果 SPI 中断被允许，当 SPIF 位置 1 时，也会产生一个中断。

(3) 接收

当主设备通过 MOSI 线传送数据给从设备时，相对应的从设备同时也通过 MISO 线将其发送移位寄存器的内容传送给主设备的接收移位寄存器，实现全双工操作。因此，SPIF 标志位置 1 即表示传送完成也表示接收数据完毕。从设备接收的数据按照 MSB 或 LSB 优先的传送方向存入主设备的接收移位寄存器。当一个字节的数据完全被移入接收寄存器时，处理器可以通过读 SPDAT 寄存器获得该数据。如果发生超限（SPIF 标志未被清 0，就试图开始下一次传送），RXOV 位置 1，表示发生数据超限，此时接收移位寄存器保持原有数据并且 SPIF 位置 1，这样直到 SPIF 位被清 0，SPI 主设备将不会接收任何数据。



从模式

(1) 模式启动

当 SPCON 寄存器中的 MSTR 位清 0，SPI 在从模式下运行。在数据传送之前，从设备的 SS 引脚必须被置低，而且必须保持低电平直到一个字节数据传送完毕。

(2) 发送与接收

从属模式下，按照主设备控制的 SCK 信号，数据通过 MOSI 引脚移入，MISO 引脚移出。一个位计数器记录 SCK 的边沿数，当接收移位寄存器移入 8 位数据（一个字节）同时发送移位寄存器移出 8 位数据（一个字节），SPIF 标志位被置 1。数据可以通过读取 SPDAT 寄存器获得。如果 SPI 中断被允许，当 SPIF 置 1 时，也会产生一个中断。

为防止超限，SPI 从设备在向接收移位寄存器移入数据之前也必须软件清零 SPIF 标志位，否则 RXOV 位置 1，表示发生数据超限。此时接收移位寄存器保持原有数据并且 SPIF 位置 1，这样 SPI 从设备将不会接收任何数据直到 SPIF 清 0。

SPI 从设备不能启动数据传送，所以 SPI 从设备必须在主设备开始一次新的数据传送之前将要传送的数据写入发送移位寄存器。如果从设备在第一次开始发送之前未写入数据，从设备将传送“0x00”字节给主设备。如果写 SPDAT 操作发生在传送过程中，那么 SPI 从设备的 WCOL 标志位置 1，即如果传送移位寄存器已经含有数据，SPI 从设备的 WCOL 位置 1，表示写 SPDAT 冲突。但是移位寄存器的数据不受影响，传送也不会被中断。

2.3. 出错检测

SPSTA 寄存器中的标志位表示在 SPI 通讯中的出错情况：

(1) 模式故障 (MODF)

SPI 主模式下的模式故障出错表明 SS 引脚上的电平状态与实际的设备模式不一致。SPSTA 寄存器中 MODF 位置 1 后，表明系统控制存在多主设备冲突的问题。这种情况下，SPI 系统受到如下影响：

- 产生 SPI 接收/错误 CPU 中断请求；
- SPSTA 寄存器的 SPEN 位清 0，SPI 被禁止；
- SPCON 寄存器的 MSTR 位清 0。

当 SPCON 寄存器的 SS 引脚禁止位 (SSDIS) 清 0，SS 引脚信号为低时，MODF 标志位置 1。然而，对于只有一个主设备的系统来说，主设备的 SS 引脚被拉低，那决不是另外一个主设备试图驱动网络。这种情况下，为防止 MODF 置 1，可使 SPCON 寄存器中的 SSDIS 位置 1，SS 引脚作为普通 I/O 口或是其它功能引脚。

重新启动串行通信时，用户必须将 MODF 位软件清 0，将 SPCON 寄存器中的 MSTR 位和 SPSTA 寄存器的 SPEN 位置 1，重新启动主模式。

(2) 写冲突 (WCOL)

在发送数据序列期间写入 SPDAT 寄存器会引起的写冲突，SPSTA 寄存器中的 WCOL 位置 1。WCOL 位置 1 不会引起中断，发送也不会中止。WCOL 位需由软件清 0。

(3) 超限情况 (RXOV)

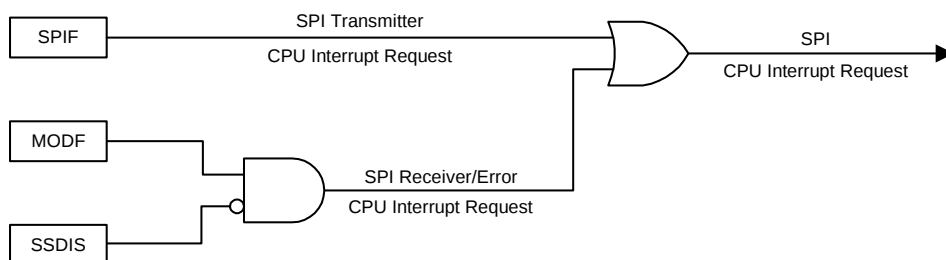
主设备或从设备尚未清除 SPIF 位，主或从设备又试图发送几个数据字节时，超限情况发生。在这种情况下，接收移位寄存器保持原有数据，SPIF 置 1，同样 SPI 设备直到 SPIF 被清除后才会再接收数据。在 SPIF 位被清除之前继续调用中断，发送也不会中止。RXOV 位置 1 不会引起中断，RXOV 位需由软件清 0。

两种 SPI 状态标志 SPIF & MODF 能产生一个 CPU 中断请求。



串行外围设备数据发送标志，SPIF：完成一个字节发送后由硬件置 1。

模式故障标志，MODF：该位被置 1 表示 SS 引脚上的电平与 SPI 模式不一致的。SSDIS 位为 0 并且 MODF 置 1 将产生 SPI 接收器/出错 CPU 中断请求。当 SSDIS 置 1 时，无 MODF 中断请求产生。



SPI 中断请求的产生

3. 应用实例

3.1. 要求

主模式，SCK = SYS/64，无效 SS 脚，不停循环发送 0x00~0xFF。

3.2. 例程

```
#include <SH79F161B.H>
#include <intrins.h>

unsigned char out_flag = 0;
unsigned char r_data = 0;

void spi_init(void)
{
    CLKCON = 0x00;    //SYS = 12.3Mz
    SPCON |= 0x40;    //Master Mode
    SPCON |= 0x08;    //disable ss pin
    SPCON |= 0x04;    //BaudRate = SYS/64
    SPSTA = 0x00;
}

void main(void)
{
    spi_init();
    EA = 1;
```



```
IEN1 |= 0x01; //enable spi interrupt
SPSTA |= 0x80;    //enable spi
SPDAT = 0x00;
while(1)
{
    if(out_flag == 1)
    {
        SPDAT ++;
        out_flag = 0;
    }
}

void spi_interrupt(void) interrupt 7
{
    out_flag = 1;
    SPSTA &= 0xBF;
}
```



六、SH79F161B EUART 用户指南

1. 概述

SH79F161B 是一种高速高效率 8051 可兼容单片机。在同样振荡频率下，较之传统的 8051 芯片它有着运行更快速的优越特性。

SH79F161B 其 EUART 模块特性为：

- ◆ SH79F161B带有1个EUART，兼容传统8051
- ◆ EUART波特率可选择为系统时钟分频或定时器1/2的溢出率
- ◆ 增强功能包括帧出错检测及自动地址识别
- ◆ EUART有四种工作方式

控制寄存器

EUART 模块使用的所有控制寄存器如下表所示：

类别	缩写符号	功能说明
	SCON	EUART 控制及状态寄存器
	SBUF	EUART 数据寄存器
	PCON	电源控制寄存器
	SADDR	EUART 从属地址寄存器
	SADEN	EUART 地址掩码寄存器
	RXCON	RXD 施密特电压控制寄存器

针对每个寄存器的详细介绍，请参照“SH79F161B SPEC”



2. EUART 设置

EUART 有 4 种工作方式。在通信之前用户必须先初始化 SCON，选择方式和波特率。如果使用方式 1 或方式 3 应先初始化定时器 1 或定时器 2。

在所有四种方式中，任何将 SBUF 作为目标寄存器的写操作都会启动发送。在方式 0 中由条件 RI = 0 和 REN = 1 初始化接收。这会在 TxD 引脚上产生一个时钟信号，然后在 RxD 引脚上移 8 位数据。在其它方式中由输入的起始位初始化接收（如果 REN = 1）。通过发送起始位，外部发送器开始通信。

EUART 方式列表

SM0	SM1	方式	类型	波特率	帧长度	起始位	停止位	第 9 位
0	0	0	同步	SYSCLK/(4 或 12)	8 位	无	无	无
0	1	1	异步	定时器 1 或 2 的溢出率/(16 或 32)	10 位	1	1	无
1	0	2	异步	SYSCLK/(32 或 64)	11 位	1	1	0, 1
1	1	3	异步	定时器 1 或 2 的溢出率/(16 或 32)	11 位	1	1	0, 1

2.1. 方式 0 设置

使用方式 0，需要先通过置 SM2 位 (SCON.5) 为 0 或 1，波特率固定为系统时钟的 1/12 或 1/4。当 SM2 位为 0 时，串行端口以系统时钟的 1/12 运行。当置 1 时，串行端口以系统时钟的 1/4 运行。

然后任何将 SBUF 作为目标寄存器的写操作都会启动发送。下一个系统时钟 Tx 控制块开始发送。数据转换发生在移位时钟的下降沿，移位寄存器的内容逐次从左往右移位，空位置 0。当移位寄存器中的所有 8 位都发送后，Tx 控制模块停止发送操作，然后在下一个系统时钟的上升沿将 TI 置 1 (SCON.1)。

若接收到数据，则 REN (SCON.4) 置 1 和 RI (SCON.0) 清 0 初始化接收。下一个系统时钟启动接收，在移位时钟的上升沿锁存数据，接收转换寄存器的内容逐次向左移位。当所有 8 位都接收到接收移位寄存器中后，Rx 控制块停止接收，然后在下一个系统时钟的上升沿上 RI 置 1，直到被软件清 0 才允许接收。

2.2. 方式 1 设置

方式 1 提供 10 位全双工异步通信，10 位由一个起始位（逻辑 0），8 个数据位（低位为第一位），和一个停止位（逻辑 1）组成。在接收时，这 8 个数据位存储在 SBUF 中而停止位储存在 RB8 (SCON.2) 中。方式 1 中的波特率是可变的，串行收发波特率可被设置为定时器 1 溢出率的 1/16 或 1/32，或是定时器 2 溢出率的 1/16（详见波特率章节）。

方式 1 的发送方式和方式 0 一致，请参照方式 0 设置章节。

只有 REN 位置 1 时才允许接收。当 RxD 引脚检测到下降沿时串行口开始接收串行数据。为此，CPU 对 RxD 不断采样，采样速率为波特率的 16 倍。当检测下降沿时，16 分频计数器立即复位，这有助于 16 分频计数器与 RxD 引脚上的串行数据位同步。16 分频计数器把每一位的时间分为 16 个状态，在第 7、8、9 状态时，位检测器对 RxD 端的电平进行采样。为抑制噪声，在这 3 个状态采样中至少有 2 次采样值一致数据才被接收。如果所接收的第一位不是 0，说明这位不是一帧数据的起始位，该位被忽略，接收电路被复位，等待 RxD 引脚上另一个下降沿的到来。若起始位有效，则移入移位寄存器，并接着移入其它位到移位寄存器。8 个数据位和 1 个停止位移入之后，移位寄存器的内容被分别装入 SBUF 和 RB8 中，RI 置 1，但必须满足下列条件：

1. RI = 0
2. SM2 = 0 或者接收的停止位 = 1



如果这些条件被满足，那么停止位装入 RB8，8 个数据位装入 SBUF，RI 被置 1。否则接收的帧会丢失。这时，接收器将重新去探测 RxD 端是否另一个下降沿。用户必须用软件清除 RI，然后才能再次接收。

2.3. 方式 2 设置

方式 2 是使用异步全双工通信中的 11 位。一帧由一个起始位（逻辑 0），8 个数据位（低位为第一位），一个可编程的第 9 数据位和一个停止位（逻辑 1）组成。方式 2 支持多机通信和硬件地址识别（详见多机通信章节）。在数据传送时，第 9 数据位（SCON 中的 TB8）可以写 0 或 1，例如，可写入 PSW 中的奇偶位 P，或用作多机通信中的数据/地址标志位。当接收到数据时，第 9 数据位进入 RB8 而停止位不保存。PCON 中的 SMOD 位选择波特率为系统工作频率的 1/32 或 1/64。

方式 2 的发送方式和方式 0 一致，请参照方式 0 设置章节。

只有 REN 位置 1 时才允许接收。当 RxD 引脚检测到下降沿时串行口开始接收串行数据。为此，CPU 对 RxD 不断采样，采样速率为波特率的 16 倍。当检测下降沿时，16 分频计数器立即复位。这有助于 16 分频计数器与 RxD 引脚上的串行数据位同步。16 分频计数器把每一位的时间分为 16 个状态，在第 7、8、9 状态时，位检测器对 RxD 端的电平进行采样。为抑制噪声，在这 3 个状态采样中至少有 2 次采样值一致数据才被接收。如果所接收的第一位不是 0，说明这位不是一帧数据的起始位，该位被忽略，接收电路被复位，等待 RxD 引脚上另一个下降沿的到来。若起始位有效，则移入移位寄存器，并接着移入其它位到移位寄存器。9 个数据位和 1 个停止位移入之后，移位寄存器的内容被分别装入 SBUF 和 RB8 中，RI 置 1，但必须满足下列条件：

1. RI = 0
2. SM2 = 0 或者接收的第 9 位 = 1，且接收的字节符合实际从机地址

如果这些条件被满足，那么第 9 位移入 RB8，8 位数据移入 SBUF，RI 被置 1。否则接收的数据帧会丢失。

在停止位的当中，接收器回到寻找 RxD 引脚上的另一个下降沿。用户必须用软件清除 RI，然后才能再次接收。

2.4. 方式 3 设置

方式 3 使用方式 2 的传输协议以及方式 1 的波特率产生方式。

2.5. EUART 波特率设置

在方式 0 中，波特率可编程为系统时钟的 1/12 或 1/4，由 SM2 位决定。当 SM2 为 0 时，串行端口在系统时钟的 1/12 下运行。当 SM2 为 1 时，串行端口在系统时钟的 1/4 下运行。

在方式 1 和方式 3 中，波特率可选择来至定时器 1 或定时器 2 的溢速率。

分别置 TCLK (T2CON.4) 和 RCLK (T2CON.5) 位为 1 来选择定时器 2 作为 TX 和 RX 的波特时钟源（详见定时器章节）。无论 TCLK 还是 RCLK 为逻辑 1，定时器 2 都为波特率发生器方式。如果 TCLK 和 RCLK 为逻辑 0，定时器 1 作为 Tx 和 Rx 的波特时钟源。

方式 1 和方式 3 波特率公式如下所示，其中 TH1 是定时器 1 的 8 位自动重载寄存器，SMOD 为 EUART 的波特率二倍频器 (PCON.7) [RCAP2H, RCAP2L] 是定时器 2 的 16 位重载寄存器。T1CLK 是定时器 1 的时钟源，T2CLK 是定时器 2 的时钟源。

$$\text{BaudRate} = \frac{2^{\text{SMOD}}}{32} \times \frac{f_{T1}}{256 - \text{TH1}}, \text{ 用定时器 1 作为波特率发生器, 定时器 1 工作在方式 2}$$

$$\text{BaudRate} = \frac{1}{2 \times 16} \times \frac{f_{\text{SYS}}}{65536 - [\text{RCAP2H}, \text{RCAP2L}]}, \text{ 用定时器 2 作为波特率发生器, 定时器 2 时钟源为}$$

$$\text{系统时钟 } \text{BaudRate} = \frac{1}{2 \times 16} \times \frac{f_{\text{SYS}} / 12}{65536 - [\text{RCAP2H}, \text{RCAP2L}]}, \text{ 用定时器 2 作为波特率发生器, 定时器 2}$$



时钟源为系统时钟/12

$$BaudRate = \frac{1}{16} \times \frac{f_{T2}}{65536 - [RCAP2H, RCAP2L]}, \text{ 用定时器 2 作为波特率发生器, 定时器 2 时钟源为 T2}$$

引脚输入时钟

在方式 2 中, 波特率固定为系统时钟的 1/32 或 1/64, 由 SMOD 位 (PCON.7) 决定。当 SMOD 位为 0 时, EUART 以系统时钟的 1/64 运行。当 SMOD 位为 1 时, EUART 以系统时钟的 1/32 运行。

$$BaudRate = 2^{SMOD} \times \left(\frac{f_{SYS}}{64}\right)$$

2.6. EUART 施密特触发设置

针对 5V 供电应用中, UART 与 3V 设备通讯电平不匹配问题, SH79F161B 的 EUART 增加了 RXD 脚的施密特触发功能, 由 RXDCON 寄存器控制。

当 RXDCON0=0 时, RXD 引脚的输入高电平阈值为 0.8VDD, 输入低电平阈值为 0.2VDD。此时, 若 SH79F161B 的 VDD=5V, 则 RXD 引脚的输入高电平阈值为 4V, 输入低电平阈值为 1V, 无法与 3V 设备的 UART 正常通讯; 而当 RXDCON0=1 时, RXD 引脚的输入高电平阈值为 0.55VDD, 输入低电平阈值为 0.2VDD。其他条件相同的情况下, 若 SH79F161B 的 VDD=5V, 则 RXD 引脚的输入高电平阈值为 2.75V, 输入低电平阈值为 1V, 能够正确接收 3V 设备所发送的数据。

注意: RXDCON 寄存器仅当 EUART 使能时有效。



3. 应用实例

3.1. EUART 例程要求

使用 EUART 进行串口通讯，通讯的波特率为 9600，1 位起始位，8 位数据位，1 位停止位，通讯内容为：收到一个数据后发送收到的数据出去。

3.2. EUART 例程

```
#include <SH79F161B.H>
unsigned char    r_data;
unsigned char    out_flag;

voiduart0_init(void)
{
    CLKCON = 0x00;          //SYS Clock is 12.3MHz
    SCON = 0x40;
    PCON = 0x00;            //MODE 1
    RXDCON = 0x00;
}
voidio_init(void)
{
    P2CR = 0x02;           //P2.1 OUTPUT/TXD    P2.0 INPUT PULL-HIGH/RXD
    P2PCR = 0x01;
    P2 = 0x02;
}
voidtimer2_init(void)
{
    T2CON = 0x30;           //Timer2 as Baud Rate Generator
    T2MOD = 0x00;
    RCAP2L = 0xD8;
    RCAP2H = 0xFF;
    TL2 = 0xD8;
    TH2 = 0xFF;            //Baud Rate is 9600
}

voidmain(void)
```



```
{
    uart0_init();
    io_init();
    timer2_init();
    r_data = 0x00;
    out_flag = 0;
    TI = 0;
    RI = 0;
    IEN0 = 0x90;
    REN = 1;
    TR2 = 1;
    while(1)
    {
        if(out_flag == 1)
        {
            SBUF = r_data;           //transmit received data
            out_flag = 0;
        }
    }
}

void uart0_int(void) interrupt 4
{
    if(TI == 1)
    {
        TI = 0;
    }
    else if(RI == 1)
    {
        r_data = SBUF;
        RI = 0;
        out_flag = 1;
    }
}
```



七、SH79F161B ADC 用户指南

1. 概述

SH79F161B 是一种高速高效率 8051 可兼容单片机。在同样振荡频率下，较之传统的 8051 芯片它有着运行更快速的优越特性。

SH79F161B 其 ADC 模块特性为：

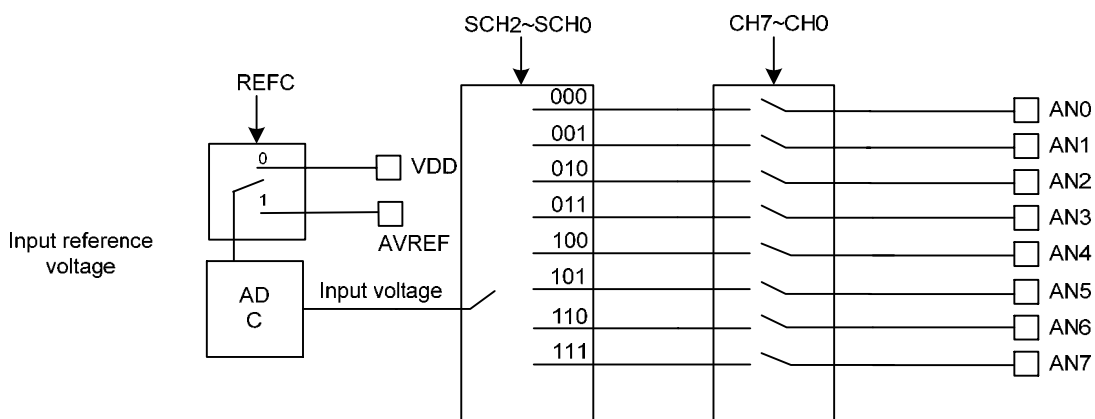
- ◆ 10位分辨率
- ◆ 可选外接或内建基准电压
- ◆ 8模拟通道输入

SH79F161B 包含一个单端型、10 位逐次逼近型模/数转换器 (ADC)，基准电压 V_{REF} 直接与 V_{DD} 相连，用户可以选择 VREF 端口输入基准电压。8 个 ADC 通道都可以输入独立的模拟信号，但是每次只能使用一个通道。GO/DONE 信号控制开始转换，提示转换结束。当转换完成时，更新 ADC 数据寄存器，设置 ADCON 寄存器中的 ADCIF 位，并产生一个中断（如果 ADC 中断被允许）。

ADC 模块整合数字比较功能可以比较 ADC 中的模拟输入的值与数字值。如果允许数字比较功能（在 ADCON 寄存器中的 EC 位置 1），并且 ADC 模块使能（在 ADCON 寄存器中的 ADON 位置 1），只有当相应的模拟输入的数字值大于或等于寄存器中的比较值（ADDH/L）时，才会产生 ADC 中断。当 GO/DONE 置 1 时，数字比较功能会持续工作，直到 GO/DONE 清 0。这一点与模数转换工作方式不同。

带数字比较功能的 ADC 模块能在 Idle 模式下工作，并且 ADC 中断能够唤醒 Idle 模式。但是，在掉电模式下，ADC 模块被禁止。

ADC 模块图如下：





控制寄存器

ADC 模块使用的所有控制寄存器如下表所示：

类别	缩写符号	功能说明
	ADCON	控制寄存器
	ADT	定时控制寄存器
	ADCH	通道设置寄存器
	ADDL	转换数据寄存器低 2 位
	ADDH	转换数据寄存器高 8 位
	RXDCON	基准电压选择寄存器

针对每个寄存器的详细介绍，请参照“SH79F161B SPEC”

2. ADC 设置

使用 ADC 模块，启动 ADC 转换请按以下步骤：

- (1) 选择模拟输入通道以及基准电压
- (2) 使能ADC模块
- (3) 延时10us
- (4) GO/DONE置1开始ADC转换
- (5) 等待GO/DONE = 0或者ADCIF = 1，如果ADC中断使能，则ADC中断将会产生，用户需要软件清0 ADCIF
- (6) 从ADDH/ADDL获得转换数据
- (7) 重复步骤4-6开始另一次转换

启动数字比较功能请按以下步骤：

- (1) 选择模拟输入通道以及基准电压
- (2) 写入ADDH/ADDL，设置比较值
- (3) EC置1使能数字比较功能
- (4) 使能ADC模块
- (5) 延时10us
- (6) GO/DONE置1开始数字比较功能
- (7) 如果模拟输入的值比设置的比较值大，ADIF会被置1。如果ADC中断使能，则ADC中断将会产生，用户需要软件清0 ADCIF
- (8) 数字比较功能会持续工作，直到 GO/DONE清 0

注：(1) 当选择外部AVREF端口输入为基准电压时 (REFC = 1)，P1.6作为 V_{REF} 输入。

(2) ADON置1后，请延时10us后，再设置GO/DONE = 1，以保障AD转换正常。

(3) 请确保ADC时钟周期 $t_{AD} = 1\mu s$ ；

(4) 即使TS[3:0] = 0000，最小采样时间为 $2t_{AD}$ ；即使TS[3:0] = 1111，最大采样时间为 $15t_{AD}$ ；

(5) 在设置TS[3:0]前，请估算连接到ADC输入引脚的串联电阻；

(6) 选择 $2 \cdot t_{AD}$ 为采样时间时，请确保连接到ADC输入引脚的串联电阻小于10k；

(7) 全部转换时间 = $12t_{AD} + \text{采样时间}$ 。



3. 应用实例

3.1. 要求

连续顺序采样 AN0, AN1, AN2 端口的输入电压, Vref = Vdd, 转换结果通过 Uart0 送出。
送出的格式为 AN0:xxxAN1:xxxAN2:xxx (xxx 为转换结果, 串口助手字符显示)

3.2. 例程

```
#include <SH79F161B.H>
#include <intrins.h>
unsigned char adc_ch = 0;
unsigned char adc_h = 0;
unsigned char adc_m = 0;
unsigned char adc_l = 0;
unsigned char outflag = 0;

voiduart_init(void)
{
    CLKCON = 0x00;           //SYS Clock is 12MHz
    SCON = 0x40;
    PCON = 0x00;             //MODE 1
}

voidtimer2_init(void)
{
    T2CON = 0x30;
    T2MOD = 0x00;
    TL2 = 0xD8;
    TH2 = 0xFF;
    RCAP2L = 0xD8;           //Timer2 as BaudRate Generrator
    RCAP2H = 0xFF;           //BaudRate is 9600
    TR2 = 1;                 //Timer2 GO
}

voidadc_init(void)
{
    ADCON = 0x80;            //enable adc module Vref = Vdd
    RXDCON = 0x00;           //P1.6 is IO, use build-in VREF
    ADT = 0x81;              //Tad = 12sys    sample time is 2Tad
    ADCH = 0x07;             //ANO,AN1,AN2 is selected otehrs is IO
```



```
    ADDL = 0x00;
    ADDH = 0x00;          //clear adc result register
}
```

```
void main(void)
{
    uart_init();
    timer2_init();
    adc_init();
    EA = 1;                //enable interrupt
    EADC = 1;              //enable adc interrupt
    ADCON &= 0xF1;          //select AN0
    ADCON |= 0x01;          //ADC GO
    RI = 0;
    TI = 0;
    while(1)
    {
        if(outflag == 1)
        {
            outflag = 0;

            adc_h = (ADDH>>6)&0x03;
            adc_h = adc_h+0x37;

            adc_m = (ADDH>>2)&0x0F;
            if(adc_m > 0x09)
            {
                adc_m = adc_m+0x37;
            }

            if(adc_m < 0x0A)
            {
                adc_m = adc_m+0x30;
            }

            adc_l = (ADDH<<2)&0x0C+ADDL;
```



```
if(adc_l > 0x09)
{
    adc_l = adc_l+0x37;
}

if(adc_l < 0x0A)
{
    adc_l = adc_l+0x30;
}

switch(adc_ch)
{
    case 1:
        SBUF = 'A';
        while(!TI);
        TI = 0;
        SBUF = 'N';
        while(!TI);
        TI = 0;
        SBUF = '0';
        while(!TI);
        TI = 0;
        SBUF = ':';
        while(!TI);
        TI = 0;
        SBUF = adc_h;
        while(!TI);
        TI = 0;
        SBUF = adc_m;
        while(!TI);
        TI = 0;
        SBUF = adc_l;
        while(!TI);
        TI = 0;
        ADCON &= 0xF1;
        ADCON |= 0x02;    //select AN1
```



```
ADCON |= 0x01;    //ADC GO  
break;
```

case 2:

```
SBUF = 'A';  
while(!TI);  
TI = 0;  
SBUF = 'N';  
while(!TI);  
TI = 0;  
SBUF = '1';  
while(!TI);  
TI = 0;  
SBUF = ':';  
while(!TI);  
TI = 0;  
SBUF = adc_h;  
while(!TI);  
TI = 0;  
SBUF = adc_m;  
while(!TI);  
TI = 0;  
SBUF = adc_l;  
while(!TI);  
TI = 0;  
ADCON &= 0xF1;  
ADCON |= 0x04;    //select AN2  
ADCON |= 0x01;    //ADC GO  
break;
```

case 3:

```
SBUF = 'A';  
while(!TI);  
TI = 0;  
SBUF = 'N';  
while(!TI);
```



```
        TI = 0;
        SBUF = '2';
        while(!TI);
        TI = 0;
        SBUF = ':';
        while(!TI);
        TI = 0;
        SBUF = adc_h;
        while(!TI);
        TI = 0;
        SBUF = adc_m;
        while(!TI);
        TI = 0;
        SBUF = adc_l;
        while(!TI);
        TI = 0;
        adc_ch = 0;
        ADCON &= 0xF1; //select AN0
        ADCON |= 0x01; //ADC GO
        break;

    default:
        break;
    }
}
}

void adc_int(void) interrupt 6
{
    ADCON &= 0xBF;
    adc_ch++;
    outflag = 1;
}
```



八、SH79F161B PWM0 用户指南

1. 概述

SH79F161B 是一种高速高效率 8051 可兼容单片机。在同样振荡频率下，较之传统的 8051 芯片它有着运行更快速的优越特性。

SH79F161B 其 PWM0 驱动器特性为：

- ◆ 6路带死区控制的互补输出
- ◆ 提供每个PWM周期溢出中断和占空比溢出中断
- ◆ 输出极性可选择
- ◆ 提供出错侦测功能可紧急关闭PWM输出
- ◆ 提供保护寄存器可使重要寄存器免受干扰出错

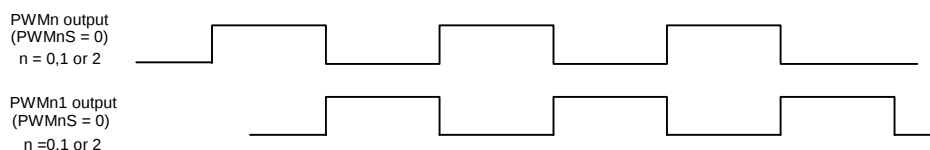
SH79F161B 集成了一个 12 位 PWM 模块和两个 8 位 PWM 模块。通过控制各自相应的寄存器，PWM 模块可以产生周期和占空比分别可调整的脉宽调制波形。此外，PWM 模块还提供了三路与 PWM0/1/2 有固定相位关系的 PWM 输出。

如果 EFLT 置位，PWM 输出能由 FLT 引脚的输入信号变化自动关闭。

PWM 定时器也为 PWM0/1/2 提供 3 个中断源，在每个 PWM 周期都会产生中断。它们有不同的控制位和标志位，共享一个中断向量地址。这样用户可以实现每个 PWM 周期中更改下一次循环的周期或占空比。

SH79F161B 包含两个 8 位 PWM 模块。PWM 模块可以产生周期和占空比分别可调整的脉宽调制波形。PWM1/2C 寄存器用于控制 PWM1/2 模块的时钟，PWMP1/2 寄存器用于控制 PWM1/2 模块输出波形的周期，PWMD1/2 寄存器用于控制 PWM1/2 模块输出波形的占空比。

SH79F161B 还包含 3 路互补输出 PWM：PWM01、PWM11 和 PWM21，且 PWM01/11/21 输出波形与 PWM0/1/2 输出波形互补。当 PWM 控制寄存器中 EPWM01/11/21 位置 1 时，PWM01/11/21 的输出波形硬件自动产生。



PWMn 和 PWMn1 引脚输出波形

注意：

- (1) 尽管 PWM0/1/2 被禁止，但是如果 PWM01/11/21 被允许，则它们仍然会有输出信号。
- (2) 如果 EFLT 置位，当 FLT 端口有效时，PWM01/11/21 和 PWM0/1/2 都输出低 (PWMnS = 0) 或者都输出高 (PWMnS = 1)。



控制寄存器

PWM0 模块使用的所有控制寄存器如下表所示：

类别	缩写符号	功能说明
12 位 PWM 模块寄存器	PWMEN	允许寄存器
	PWMLO	保护寄存器
	PWM0C	12 位 PWM0 控制寄存器
	PWM0PL	PWM0 周期控制寄存器高 4 位
	PWM0PH	PWM0 周期控制寄存器低 8 位
	PWM0DL	PWM0 占空比控制寄存器低 8 位
	PWM0DH	PWM0 占空比控制寄存器高 4 位
	PWM0DT	死区时间控制寄存器
8 位 PWM 模块寄存器	PWM1C	8 位 PWM1 控制寄存器
	PWM2C	8 位 PWM2 控制寄存器
	PWM1P	8 位 PWM1 周期寄存器
	PWM2P	8 位 PWM2 周期寄存器
	PWM1D	8 位 PWM1 占空比寄存器
	PWM2D	8 位 PWM2 占空比寄存器
	PWM1DT	PWM1 死区时间控制寄存器
	PWM2DT	PWM2 死区时间控制寄存器

针对每个寄存器的详细介绍，请参照“SH79F161B SPEC”

2. PWM0 设置

为了提高 SH79F161B 的抗干扰能力，对 PWM0 的相关寄存器进行操作之前，必须把 PWMLO 寄存器设置为 55H，这样才能有效更改 PWM0 相关寄存器的值。在 PWMLO 设置为 55H 后，再选择好 PWM 时钟的分频比，然后根据相关系统设计的需求，设置好 PWM0 的周期，初始占空比，死区时间。最后，将 PWMEN 中的 EPWM0 位置 1，PWM0 模块就会工作。

当 PWM0 模块使能之后，可以通过选择 PWM 输出允许位来选择需要输出的 PWM 的相应通道，其它没有输出的 PWM 管脚可以当作 IO 使用，PWM0 也可以当作一个 12 BIT 的 Timer 使用。同时，若使能 PWM0 中断，在每个 PWM 周期溢出和占空比溢出将产生 PWM0 中断，在中断程序中可以更改 PWM0 的周期，占空比，死区时间，这些更改会在下一个周期生效。在更改完毕后，请将 PWMLO 置为 00H。这样可以提高 SH79F161B 的抗干扰能力。



3. 应用实例

3.1. 要求

使用 PWM0，产生一个 2KHz，占空比为 50%的 PWM 波驱动蜂鸣器，其输出脚位是 PWM0。

3.2. 例程

```
#include <SH79F161B.H>
#include <intrins.h>
void PWM0_init(void)
{
    CLKCON = 0x00;    //SYS Clock is 12MHz
    PWMLO = 0x55;      //enable PWM0 register change
    PWM0C = 0x00;      //Disable PWM0 interrupt PWM0 clock is osc/2
    PWM0PL = 0xB8;
    PWM0PH = 0x0B;     //PWM0 frequency is 2KHz
    PWM0DL = 0xDC;
    PWM0DH = 0x05;     //PWM0 Duty is 50%
    PWM0DT = 0x00;     //NO deadtime
    PWMLO = 0x00;
}
void main(void)
{
    PWM0_init();
    PWMLO = 0x55;
    PWMEN |= 0x01;     //enable PWM0
    PWMLO = 0x00;      //PWM0(P2.4) output the wave
    while(1);
}
```



九、SH79F161B Buzzer 用户指南

1. 概述

SH79F161B 是一种高速高效率 8051 可兼容单片机。在同样振荡频率下，较之传统的 8051 芯片它有着运行更快速的优越特性。

SH79F161B 其 Buzzer 驱动器特性为：

- ♦ 为音频发生器输出方波信号
- ♦ 有10种频率可供选择输出或者禁止输出

2. 控制寄存器

Buzzer 模块使用的所有控制寄存器如下表所示：

类别	缩写符号	功能说明
模块控制	BUZCON	控制寄存器

针对每个寄存器的详细介绍，请参照“SH79F161B SPEC”

3. 应用实例

3.1. 要求

使用 Buzzer，产生一个 12KHz，占空比为 50%的方波驱动蜂鸣器，其输出脚位是 BZ（P2.0）。

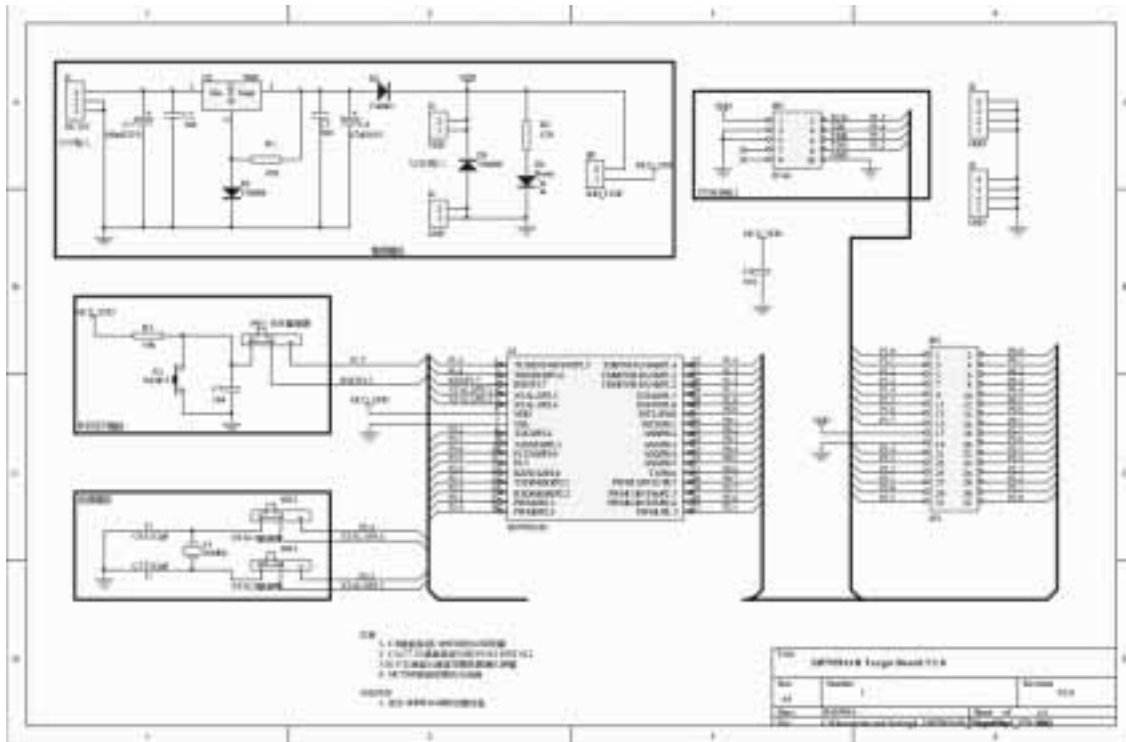
3.2. 例程

```
#include <SH79F161B.H>
#include <intrins.h>

void main()
{
    CLKCON = 0x00;           //设置系统时钟不分频
    BUZCON = 0x07;           //BZ 口输出方波频率为 Fsys/1024
    while(1);
}
```

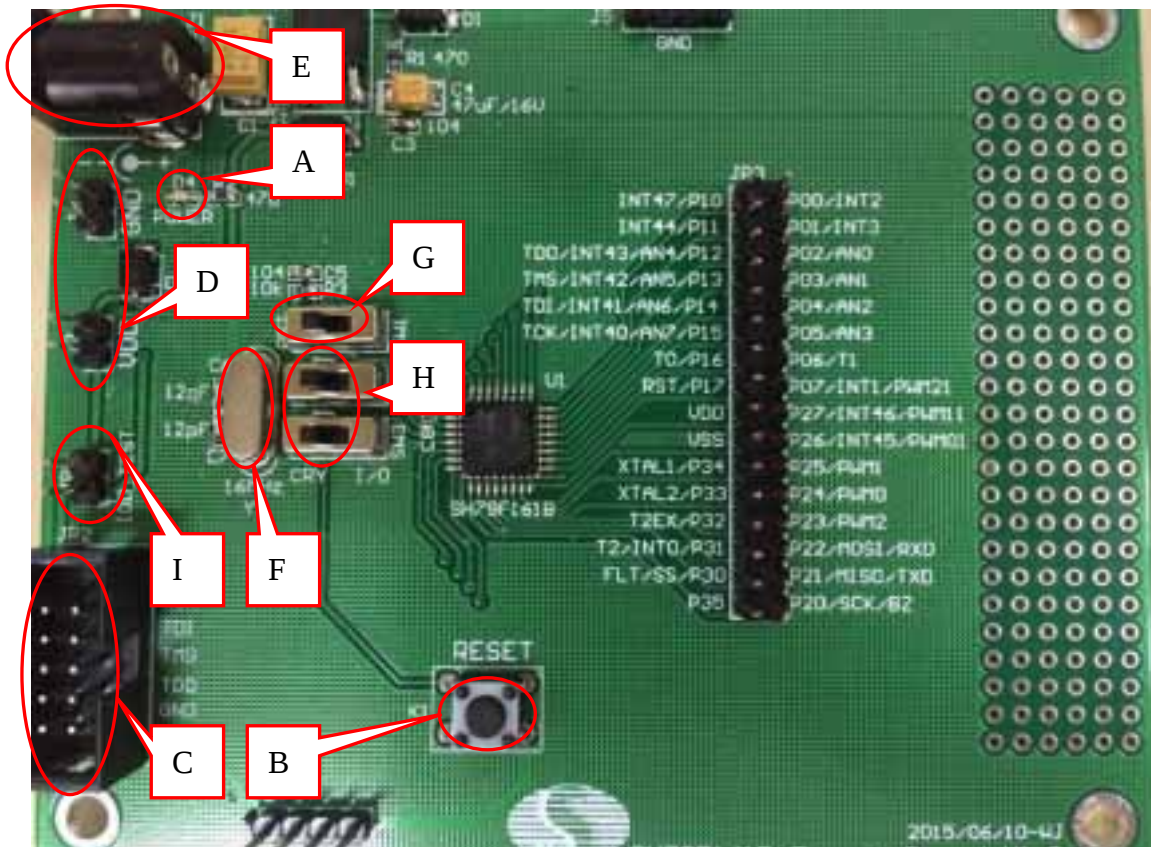


附件一：SH79F161B Target Board V2.0 原理图



附件二：SH79F161B 演示版 (Target Board)

SH79F161B 提供了一套演示板，供客户熟悉芯片功能。如下图：





板中各部分功能说明:

- A. 电源灯
- B. 复位按键
- C. 仿真接口

仿真接口和 JET51 仿真器 JTAG 接口一一对应,可用 10 芯扁平线直接连接。演示板的供电电源可通过仿真接口的 VDD 管脚获得。

D. 外部供电接口 1(VDD/GND)

外部供电接口 1 为 SH79F161B 供电电源接口,分别与 SH79F161B VDD 和 GND 管脚连接。

采用外部供电接口 1 供电,演示板供电电源可调。

E. 外部供电接口 2 (内正外负)

外部供电接口 2 可以外接 9~12V 电源,经过板上 U2 稳压电路产生 5V 电压作为 SH79F161A 的供电电源使用。

采用外部供电接口 2 方式供电,演示板供电电源为 5V,不可调。

注:

- 1). 演示板的供电可以由外部供电接口 1 或外部供电接口 2 或仿真接口提供,不管使用哪一种供电方式,外部供电接口 1 处的电压为演示板的供电电压。
- 2). 如果选择仿真接口供电,电源选项请选择“5V(JET51)”;如果选择外部供电接口 1 或 2 供电,电源选项请选择“外部供电(从用户板)”。

F. 外部振荡器插座

当系统下载程序时,此插座可以不接任何振荡器。

当系统联机仿真时,请选择与代码选项相对应的振荡器插在此插座上。

G. 复位选择开关

当代码选项选择“P1.7 used as RST pin”,复位选择开关拨至 RESET 端,否则拨至 I/O 端。

H. 振荡器选择开关

当代码选项选择“400K-16M crystal oscillator or ceramic oscillator”,2 个振荡器选择开关都拨至 CRY 端,否则拨至 I/O 端。

I. 电流测试接口

将电流表串接至此测试口,即可以测量 SH79F161B 的电流。不进行电流测试时,请用短路跳线将其短接。



附件三：SH79F161B 和 SH79F161A 差异对照表

SH79F161B 相对于 SH79F161A 主要改进特性如下：

1. 振荡器电路优化，改善了芯片的抗湿度及振荡器起振性能，无需外接反馈电阻；
2. RC时钟增加16MHz档并且提升了RC时钟精度，常温下由1%提高至0.5%，全温度范围内也有所增强；
3. SH79F161B相较于SH79F161A的EMC性能有改善；
4. 增加UART中RXD引脚施密特(3V/5V)，可直接与3V逻辑通讯；
5. P1.6增加ADC外灌参考电压VREF功能；

中颖 SH79F161B 单片机为 SH79F161A 引脚全兼容的增强型号，主要改进特性在于提升了晶振的抗湿度能力以及振荡器起振性能。详细特性对比如表 1 所示：

Table 1 SH79F161A 与 SH79F161B 主要差异表

型号	SH79F161A	SH79F161B
Package	LQFP32/QFP44/LQFP44	LQFP32/LQFP44 (与 SH79F161A 引脚分布全兼容)
OSC	12.3MHz $\pm$ 1.0% RC(@RT) 12.3MHz $\pm$ 2.0% RC(TA=-40~+85°C) 400kHz~16MHz Cry/Cer	12.3MHz/16MHz $\pm$ 0.5% RC(@RT) 12.3MHz/16MHz $\pm$ 2.0% RC(TA=-40~+85°C) 400kHz~16MHz Cry/Cer
通讯	UART	UART (带施密特)
ADC	8-ch 10-bit ADC	8-ch 10-bit ADC (支持外灌 VREF)



规格更改记录

版本	记录	日期
1.0	初始版本	2015 年 9 月

目录

一、SH79F161B I/O 用户指南	1
1. 概述	1
2. 控制寄存器	1
3. IO 设置.....	1
3.1. IO 输出设置	1
3.2. IO 输入设置	1
3.3. 开漏极 I/O 设置	1
4. 应用实例	2
4.1. 要求	2
4.2. 例程	2
二、SH79F161B Flash&EEPROM 用户指南	3
1. 概述	3
2. 控制寄存器	4
3. SSP 编程设置	4
3.1. Flash SSP 编程设置.....	4
3.2. EEPROM SSP 编程设置.....	5
3.3. Flash & EEPROM 控制流程图	5
4. 应用实例	6
4.1. 要求	6
4.2. 例程	6
5. 编程注意事项	9
6. FLASH/类 EEPROM 的烧写/擦除的超级抗干扰措施.....	10
三、SH79F161B Timer0/1 用户指南	12
1. 概述	12
2. 控制寄存器	12
3. Timer0/1 设置	12
3.1. 方式 0 设置.....	12
3.2. 方式 1 设置.....	13
3.3. 方式 2 设置.....	13
3.4. 方式 3 设置.....	14



4. 应用实例	15
4.1. 方式 0 例程.....	15
4.2. 方式 1 例程.....	16
4.3. 方式 2 例程.....	17
4.4. 方式 3 例程.....	17
四、SH79F161B Timer2 用户指南.....	19
1. 概述	19
2. 控制寄存器	19
3. Timer2 设置	19
3.1. 方式 0 设置.....	19
3.2. 方式 1 设置.....	20
3.3. 方式 2 设置.....	20
3.4. 方式 3 设置.....	20
4. 应用实例	21
4.1. 方式 0 例程.....	21
4.2. 方式 1 例程.....	22
4.3. 方式 3 例程.....	23
五、SH79F161B SPI 用户指南	24
1. 概述	24
2. SPI 设置.....	24
2.1. 波特率设置.....	24
2.2. 工作模式设置	25
2.3. 出错检测	26
3. 应用实例	27
3.1. 要求	27
3.2. 例程	27
六、SH79F161B EUART 用户指南.....	29
1. 概述	29
2. EUART 设置	30
2.1. 方式 0 设置.....	30
2.2. 方式 1 设置.....	30
2.3. 方式 2 设置.....	31
2.4. 方式 3 设置.....	31
2.5. EUART 波特率设置.....	31
2.6. EUART 施密特触发设置.....	32
3. 应用实例	33



3.1. EUART 例程要求	33
3.2. EUART 例程	33
七、SH79F161B ADC 用户指南	35
1. 概述	35
2. ADC 设置	36
3. 应用实例	37
3.1. 要求	37
3.2. 例程	37
八、SH79F161B PWM0 用户指南	42
1. 概述	42
2. PWM0 设置	43
3. 应用实例	44
3.1. 要求	44
3.2. 例程	44
九、SH79F161B Buzzer 用户指南	45
1. 概述	45
2. 控制寄存器	45
3. 应用实例	45
3.1. 要求	45
3.2. 例程	45
规格更改记录	49