



MPC82x5x 用户指南

V1.01

目 录

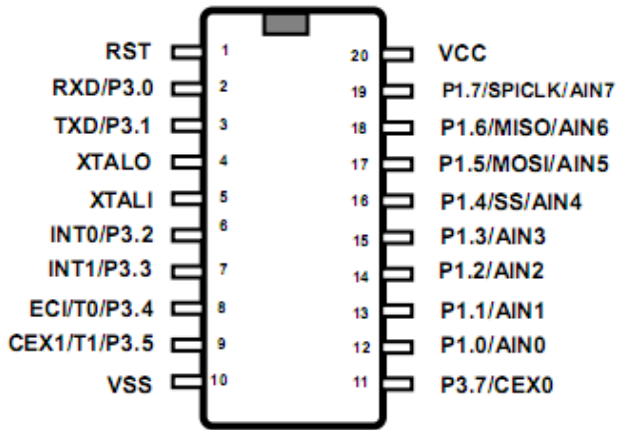
1	特性.....	3
2	引脚.....	4
3	特殊功能寄存器.....	6
4	存储器	7
4.1	RAM.....	7
	MPC82x52 RAM 空间.....	7
	MPC82x54 RAM 空间.....	7
4.2	FLASH	8
	MPC82x52 FLASH 空间	8
	MPC82x54 FLASH 空间	8
5	I/O 口	9
5.1	相关特殊寄存器	9
5.2	I/O 口模式配置	10
6	定时/计数器	12
6.1	相关特殊寄存器	12
6.2	定时/计数器的四种模式	13
7	中断.....	15
7.1	相关特殊寄存器	15
7.2	中断入口定义.....	17
	汇编语言	17
	C 语言	18
8	IAP/ISP 应用	20
8.1	相关特殊寄存器	20
8.2	IAP/ISP 基本操作(汇编)	21
8.3	IAP/ISP 操作实例(C 语言).....	22
9	串口(UART)的使用	31
9.1	相关特殊寄存器	31
9.2	波特率的设置.....	31
9.3	串口应用实例(C 语言).....	32
10	ADC 的使用.....	36
10.1	相关特殊寄存器.....	36
10.2	ADC 应用实例—按键输入(C 语言).....	37
11	PCA 的使用	42
11.1	相关特殊寄存器.....	42
11.2	PCA 的四种模式	44
	PCA 捕捉模式.....	44
	16 位软件定时器模式.....	48
	16 位高速输出模式	48
	8 位 PWM 输出模式	51
12	SPI 接口	55
12.1	相关特殊寄存器.....	55
12.2	SPI 示例(C 语言).....	56

SPI_MASTER	56
SPI_SLAYER.....	61
13 看门狗.....	64
13.1 相关特殊寄存器.....	64
13.2 看门狗时间计算.....	65
13.3 看门狗示例.....	65
汇编语言	65
C 语言	65
14 电源控制	66
14.1 相关特殊寄存器.....	66
14.2 IDLE 模式.....	66
14.3 睡眠模式	66
14.4 睡眠和 IDLE 模式示例(C 语言)	67
15 程序启动入口	71
从 ISP 程序切换到 AP 应用程序.....	71
从 AP 应用程序切换到 ISP 程序.....	71
16 程序烧录	72
16.1 使用 ISP PROGRAMMER 烧录程序	72
预备.....	72
从 PC 下载程序到 ISP Programmer	72
从 ISP Programmer 烧录到 MPC82x5x.....	74
16.2 使用 8051 WRITER U1 烧录程序.....	75
预备.....	75
从 PC 下载程序到 MPC82x5x	75
17 附录.....	80
17.1 指令集	80

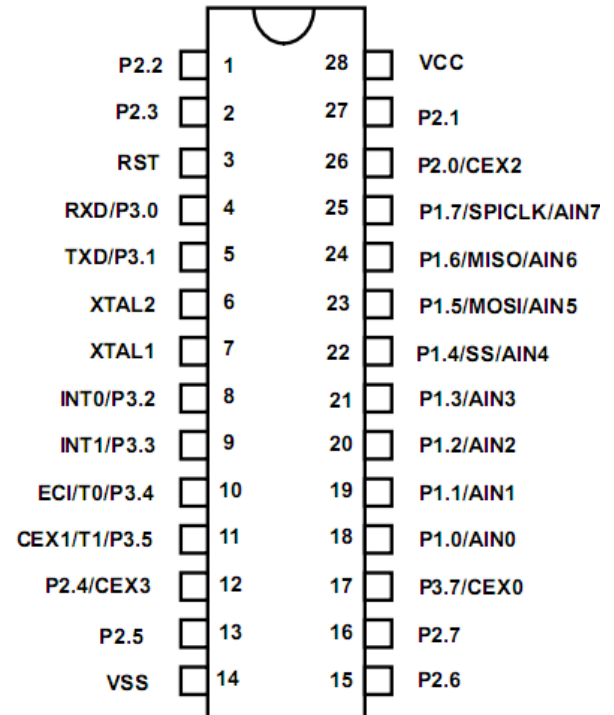
1 特性

- 增强型 80C51 内核
- **8KB (MPC82x52),15.5KB(MPC82x54)**FLASH 空间(AP/IAP/ISP 共享),超过 20,000 次的烧写寿命, 室温下数据可保存超过 100 年。
- 256 Bytes RAM, MPC82x54 还内嵌外部寻址 RAM (XDATA) 256Bytes.
- 两级代码加密保护
- 两个 16 位定时/计数器
- 7 个中断源, 4 级优先级
- 一组增强型 UART
- 15 位看门狗, 8 位预分频。使能后, 不能关闭
- 内置 SPI,可设主/从模式
- 一个 Programmable Counter Array (PCA)
- 8 通道 8 位 ADC, MPC82x54 是 10 位 ADC
- 对于 **PDIP-20(MPC82x5xAE),SOP-20(MPC82x5xAS),TSSOP-20(MPC82x5xAT)**, 有 15 个可编程 I/O 口。对于 **PDIP-28(MPC82x54AE2),SOP-28(MPC82x54AS2),TSSOP-28(MPC82x54AT2)** 则有 23 个可编程 I/O 口。对于 **PLCC-32(MPC82x54AP)**则有 27 个可编程 I/O 口
- 内置 6Mhz RC 振荡
- 1T 指令周期,远快于普通 8051 的 6T/12T。最大工作频率为 24MHz。工作在 4MHz 时相当于普通 8051 的 24MHz@6T, 48MHz@12T, 极大的降低了 EMI.
- 极好的抗干扰能力
- 超低功耗
- 工作电压:
MPC82E52/54: 4.5V~5.5V
MPC82L52/54: 2.4V~3.6V, 在写 FLASH 是最低为 2.7V
- 工作温度:
-40℃~ +85℃
- 封装:
PDIP-20: MPC82x52AE/MPC82x54AE
PDIP-28: MPC82x54AE2
PLCC-32: MPC82x54AP
SOP-20: MPC82x52AS/MPC82x54AS
SOP-28: MPC82x54AS2
TSSOP-20: MPC82x52AT/MPC82x54AT
TSSOP-28: MPC82x54AT2

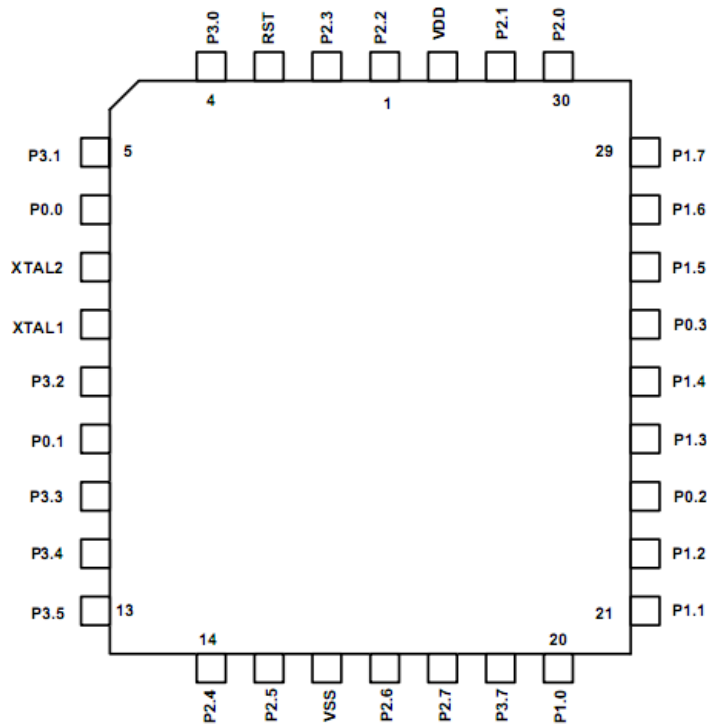
2 引脚



PDIP-20/ SOP-20/ TSSOP-20
 MPC82x5xAE/AS/AT



PDIP-28/ SOP-28/ TSSOP-28
 MPC82x54AE2/AS2/AT2



PLCC-32
 MPC82x54AP

引脚名	引脚号			类型	描述
	PLCC-32	PDIP-28	PDIP-20		
P0.0 P0.1 P0.2 P0.3	6 10 23 26			I/O	P0 是可编程 I/O 口
P1.0(AIN0) P1.1(AIN1) P1.2(AIN2) P1.3(AIN3) P1.4(SS/AIN4) P1.5(MOSI/AIN5) P1.6(MISO/AIN6) P1.7(SPICK/AIN7)	20 21 22 24 25 27 28 29	18 19 20 21 22 23 24 25	12 13 14 15 16 17 18 19	I/O	P1 是可编程 I/O 口 同时可作为 ADC 输入 P1 还具有以下特殊功能 SS(P1.4) SPI 接口串行模式选择 MOSI(P1.5) SPI 接口主数据输出或从数据输入 MISO(P1.6) SPI 接口主数据输入或从数据输出 SPICK(P1.7) SPI 接口时钟线
P2.2 P2.3 P2.4(CEX3) P2.5 P2.6 P2.7	1 2 14 15 17 18	1 2 12 13 15 16		I/O	P2 可编程 I/O 口 P2 还具有以下特殊功能 CEX3(P2.4)
P3.0(RXD) P3.1(TXD) P3.2(INT0) P3.3(INT1) P3.4(ECI/T0) P3.5(CEX1/T1) P3.7(CEX0)	4 5 9 11 12 14 19	4 5 18 9 10 12 17	2 3 6 7 8 12 11	I/O	P3 是可编程 I/O 口 P3 口还具有以下特殊功能 Rx(D)(P3.0) 串行输入口 Tx(D)(P3.1) 串行输出口 INT0(P3.2) 外部中断 0 INT1(P3.3) 外部中断 1 T0(P3.4) 定时器 0 外部输入 ECI(P3.4) PCA 外部时钟输入口 T1(P3.5) 定时器 1 外部输入 CEX1(P3.5) CEX0(P3.7)
RST	3	3	1	I	当该脚输入高电平超过 2 个机器周期时, 芯片就会产生复位
VSS	16	14	10	G	电源地
VCC	32	28	20	P	电源

3 特殊功能寄存器

	8	9	A	B	C	D	E	F
F8		CH	CCAP0H	CCAP1H	CCAP2H**	CCAP3H**		
F0	B		PCAPWM0*	PCAPWM1*	PCAPWM2**	PCAPWM3**		
E8		CL	CCAP0L	CCAP1L	CCAP2L	CCAP3L		
E0	ACC	WDTCR	IFD	IFADRH	IFADRL	IFMT	SCMD	ISPCR
D8	CCON	CMOD	CCAPM0	CCAPM1				
D0	PSW							
C8								
C0						ADCTL	ADCV	PCON2*
B8	IP	SADEN					ADCVL**	
B0	P3	P3M0	P3M1					IPH
A8	IE	SADDR						
A0	P2**							TSTWD**
98	SCON	SBUF						
90	P1	P1M0	P1M1	P0M0**	P0M1**	P2M0**	P2M1**	
88	TCON	TMOD	TL0	TL1	TH0	TH1	AUXR	
80	P0**	SP	DPL	DPH	SPISTAT	SPICTL	SPIDAT	PCON
	0	1	2	3	4	5	6	7

*: 只写

** : 只在 MPC82x54 中有效

4 存储器

4.1 RAM

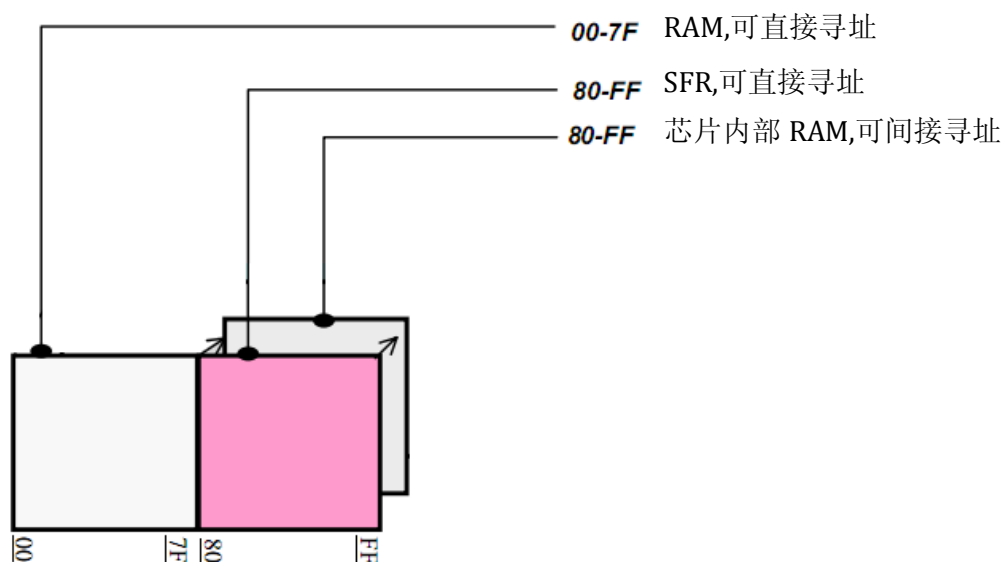
对于 MPC82x52, 内建 256 字节 RAM. 对于 MPC82x54, 内建 512 字节 RAM.

用户可直接寻址开始的 128 字节 RAM, 我们叫它直接 RAM, 它的地址空间是 00h~7Fh.

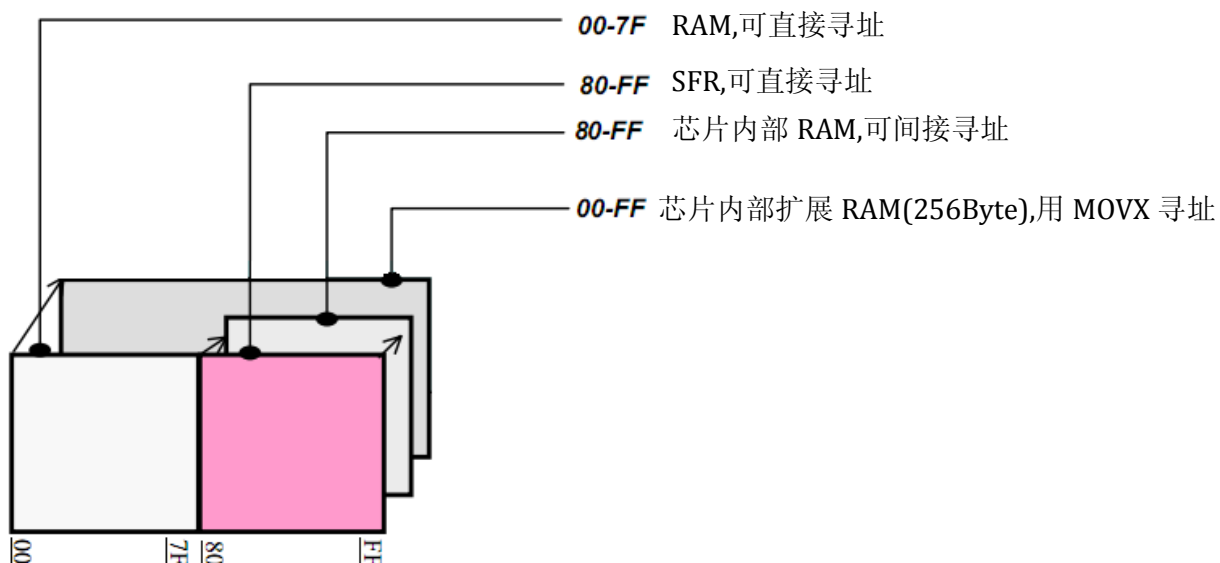
接下来的 128 字节 RAM, 用户可以间接寻址到它。我们叫它间接 RAM, 它的空间地址是 80h~FFh

其它的 RAM(仅 MPC82x54 有 256Bytes)被叫做扩展 RAM, 它占用的空间地址 00h~FFh 用户可以通过寄存器 Ri 或数据指针 DPTR, 使用 MOVX 指令来访问它, 如: MOVX A, @R1 or MOVX A, @DPTR。

MPC82x52 RAM 空间



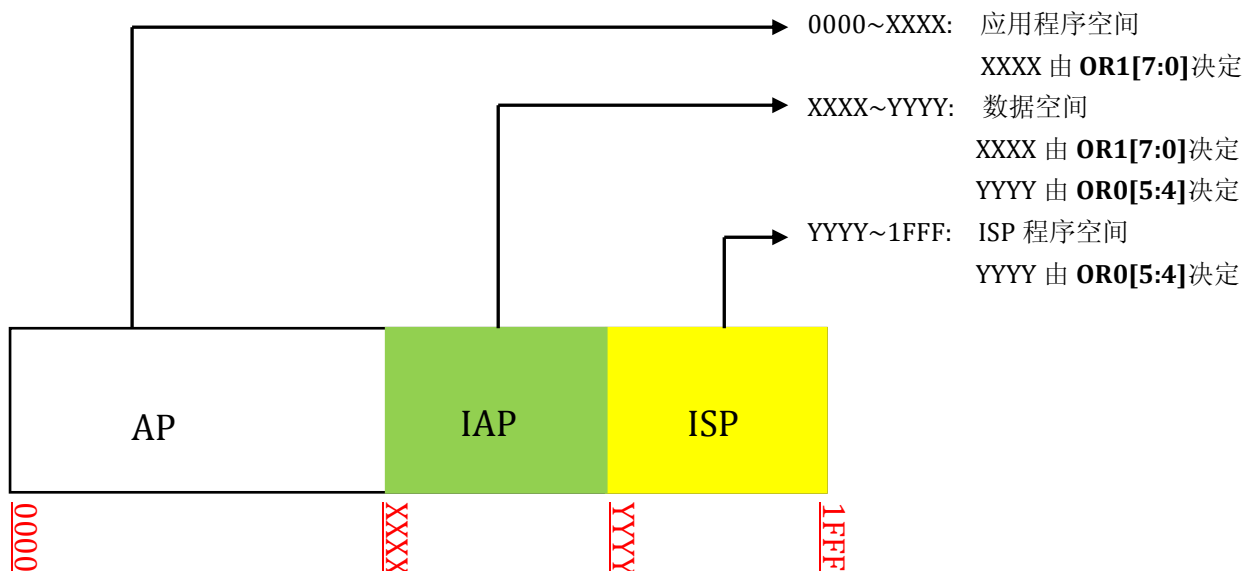
MPC82x54 RAM 空间



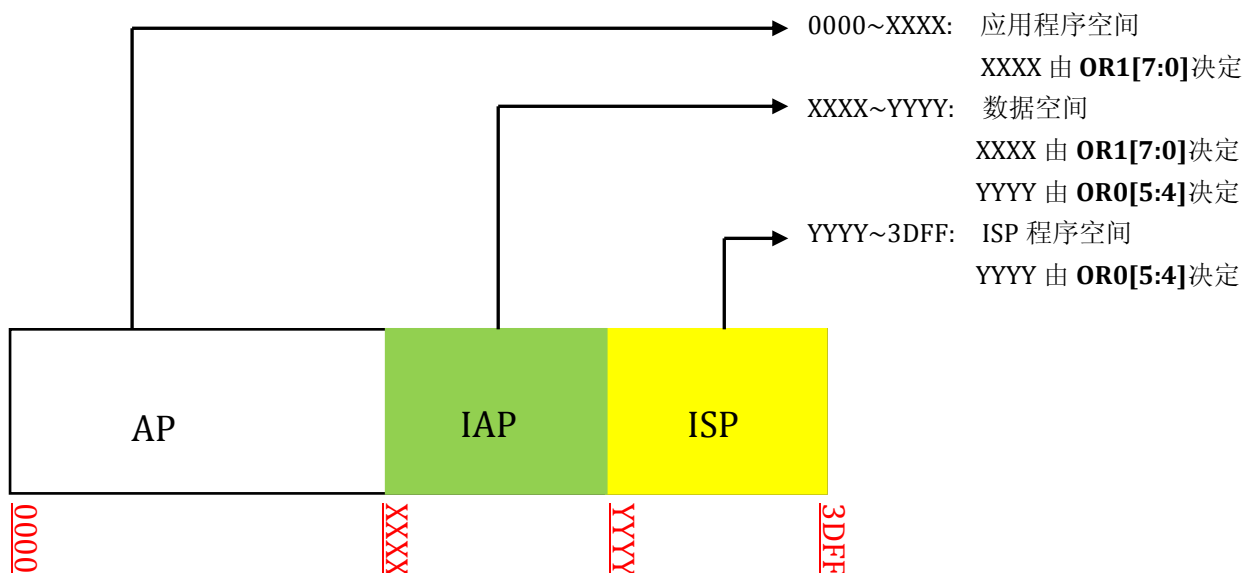
4.2 FLASH

对于 MPC82x52, 总共有 8K 字节 FLASH, 对于 MPC82x54 总共有 15.5K 字节 FLASH.

MPC82x52 FLASH 空间



MPC82x54 FLASH 空间



***注: OR0,OR1 只能通过 8051 Writer U1 才能写入。详见“[程序烧录](#)”部分**

5 I/O 口

MPC82x5x 的 I/O 口可分别设成 4 种不同的模式:

1: 标准 51 输出模式; 2: CMOS 输出模式; 3: 仅输入模式; 4: 开集输出模式;

5.1 相关特殊寄存器

■ P1M0(0x91): P1 口模式配置寄存器 0

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit0
P1M07	P1M06	P1M05	P1M04	P1M03	P1M02	P1M01	P1M00

■ P1M1(0x92): P1 口模式配置寄存器 1

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit0
P1M17	P1M16	P1M15	P1M14	P1M13	P1M12	P1M11	P1M10

■ P3M0(0xB1): P3 口模式配置寄存器 0

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit0
P3M07		P3M05	P3M04	P3M03	P3M02	P3M01	P3M00

■ P3M1(0xB2): P3 口模式配置寄存器 1

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit0
P3M17		P3M15	P3M14	P3M13	P3M12	P3M11	P3M10

■ P1(0x90), P3(0xB0): P1, P3 口的值

■ P0M0(0x93): P0 口模式配置寄存器 0 (只对 MPC82x54 有效)

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit0
				P0M03	P0M02	P0M01	P0M00

■ P0M1(0x94): P0 口模式配置寄存器 1 (只对 MPC82x54 有效)

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit0
				P0M13	P0M12	P0M11	P0M10

■ P2M0(0x95): P2 口模式配置寄存器 0 (只对 MPC82x54 有效)

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit0
P2M07		P2M05	P2M04	P2M03	P2M02	P2M01	P2M00

■ P2M1(0x96): P2 口模式配置寄存器 1 (只对 MPC82x54 有效)

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit0
P2M17		P2M15	P2M14	P2M13	P2M12	P2M11	P2M10

■ P0(0x80), P2(0xA0): P0, P2 口的值 (只对 MPC82x54 有效)

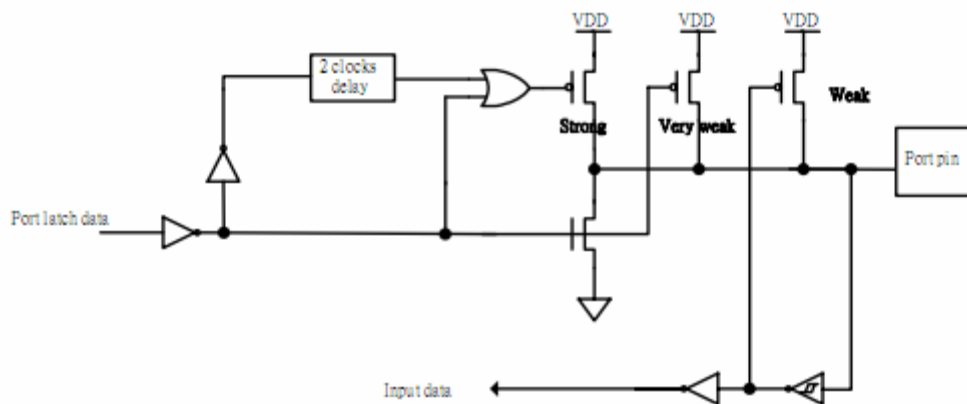
5.2I/O 口模式配置

PxM0n	PxM1n	I/O 口模式
0	0	标准 51 输出模式(默认)
0	1	CMOS 输出模式
1	0	仅输入模式(高阻)
1	1	开集输出模式

X=1 或 3, n=0,1,2,3,4,5,6,7

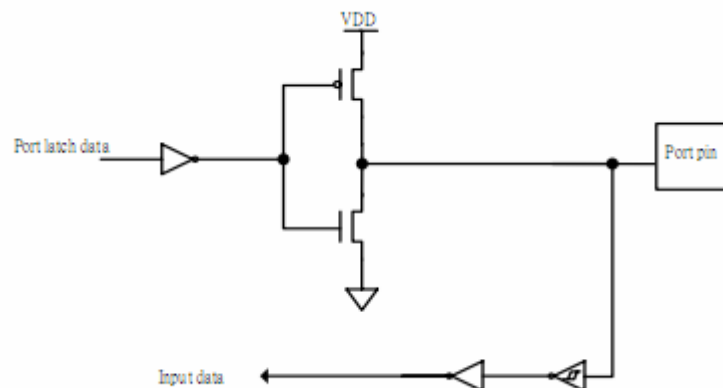
■ 标准 51 输出模式(默认)

PxM0n = 0, PxM1n = 0

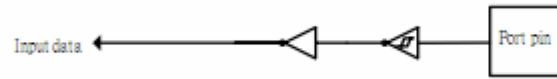


■ CMOS 输出模式

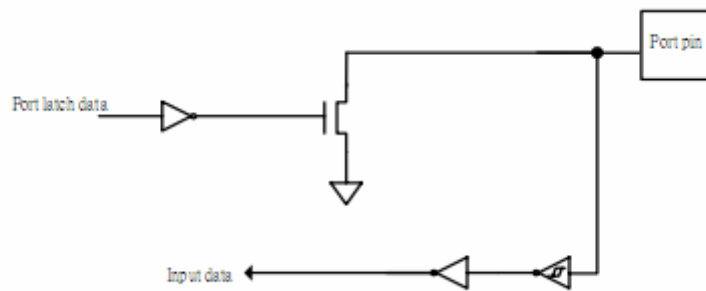
PxM0n = 0; PxM1n = 1;



- 仅输入模式(高阻)
 $PxM0n = 1; PxM1n = 0;$



- 开集输出模式
 $PxM0n = 1; PxM1n = 1;$



6 定时/计数器

MPC82x5x 提供了两个 16 位定时/计数器 T0,T1。

6.1 相关特殊寄存器

■ TMOD(0x89): TIMER 模式控制寄存器

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit0
GATE	C/T	M1	M0	GATE	C/T	M1	M0

用于 T1

用于 T0

- GATE:** 0: 只要 TRx 置 1, Timer x 即使能
 1: 必须 TRx 置 1, 且/INTx 为高, Timer x 才使能
- C/T:** 0: 作为定时器
 1: 作为计数器
- M1,M0** 模式选择
 0,0: 作为 13 位定时/计数器
 0,1: 作为 16 位定时/计数器
 1,0: 作为 8 位自动重载定时/计数器, 重载值存于 THx
 1,1: 对于 T0, TL0 是一个 8 位定时/计数器, TH0 是一个 8 位定时器
 T1 被停止

■ TCON(0x88)

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

- TF1:** T1 溢出标志位
 当 T1 溢出时, 该位会自动置 1. 当执行 T1 溢出中断时, 该位自动清零.
- TR1:** 0: 停止 T1
 1: 开始 T1
- TF0:** T0 溢出标志位
 当 T0 溢出时, 该位会自动置 1. 当执行 T0 溢出中断时, 该位自动清零.
- TR0:** 0: 停止 T0
 1: 开始 T0
- IE1:** 外部中断 1 标志
 当外部中断 1 产生时, 该位会自动置 1. 当执行外部中断 1 时, 该位自动清零.
- IT1:** 0: 引脚 EX1 低电平, 产生外部中断 0
 1: 引脚 EX1 下降沿, 产生外部中断 0
- IE0:** 外部中断 0 标志
 当外部中断 0 产生时, 该位会自动置 1. 当执行外部中断 0 时, 该位自动清零.
- IT0:** 0: 引脚 EX0 低电平, 产生外部中断 0
 1: 引脚 EX0 下降沿, 产生外部中断 0

■ AUXR(0x8E): 辅助寄存器

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit0
T0X12	T1X12	URM0X6	EADCI	ESPI	ENLVFI	---	---

T0X12: T0 时钟选择

0(默认): Fosc/12

1: Fosc

T1X12: T1 时钟选择

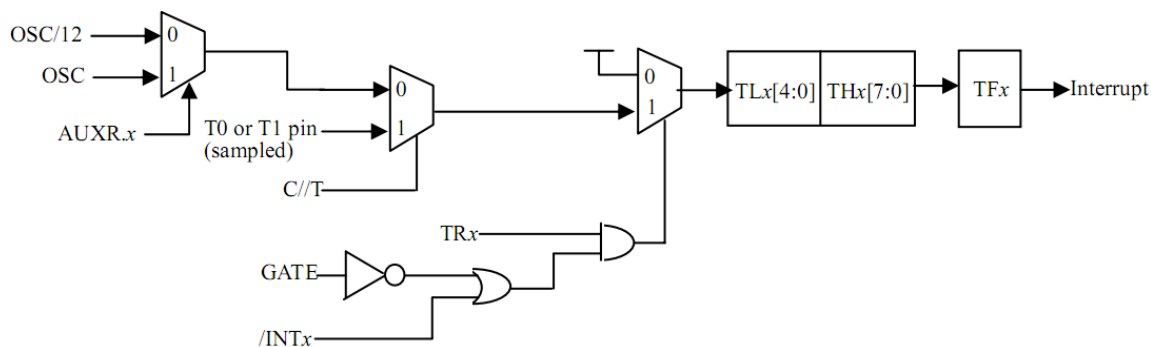
0(默认): Fosc/12

1: Fosc

6.2 定时/计数器的四种模式

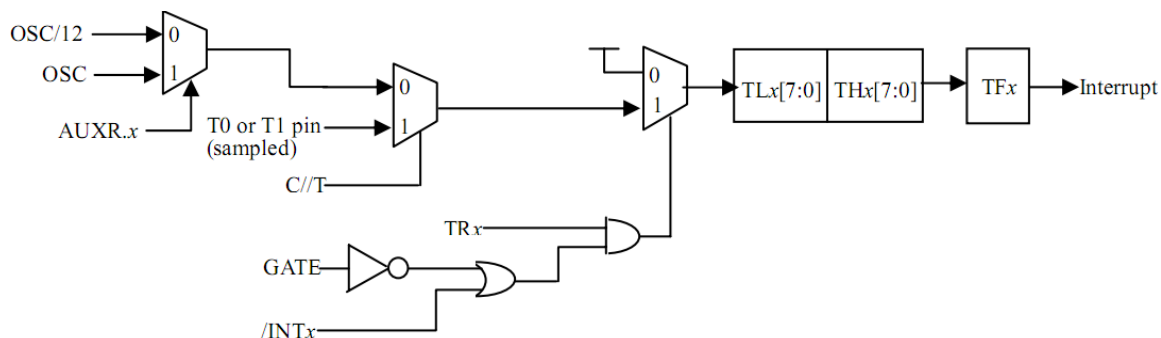
■ M1,M0 = 0,0: 模式 0

13 位定时/计数器

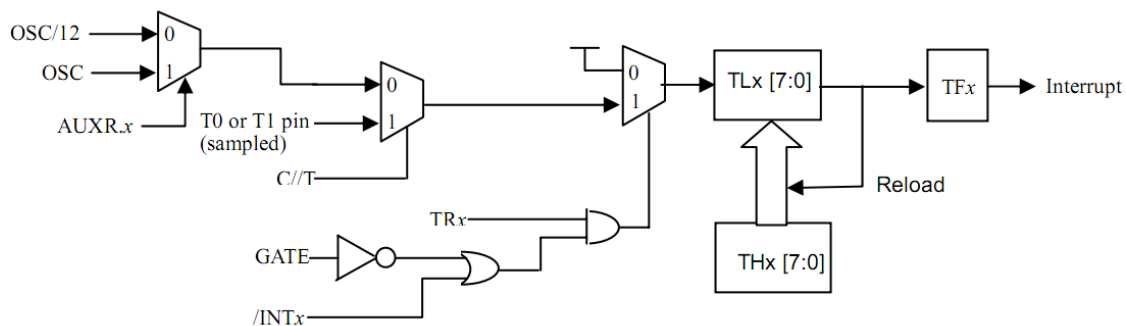


■ M1,M0 = 0,1: 模式 1

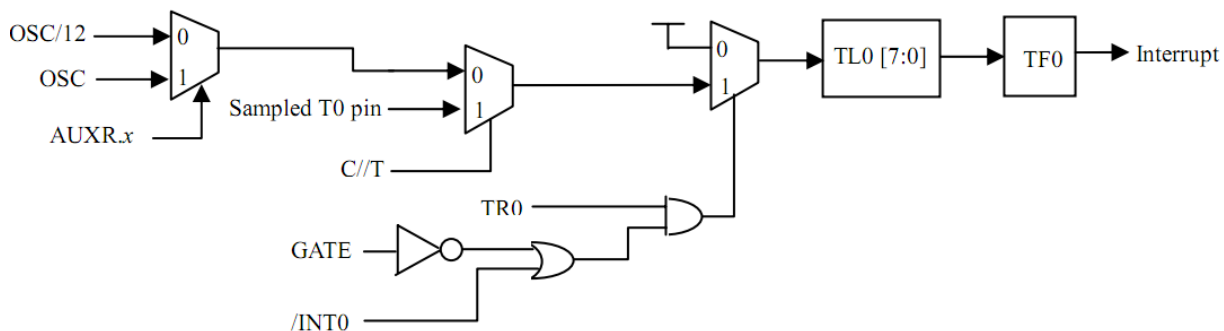
16 位定时/计数器



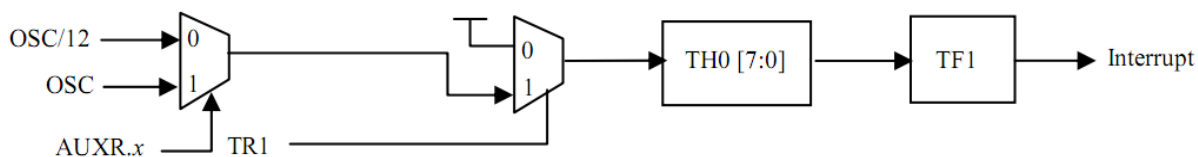
■ **M1,M0 = 1,0: 模式 2**
 8 位自动重载定时/计数器



■ **M1,M0 = 1,1: 模式 3**
 TL0 是一个 8 位定时/计数器



TH0 是一 8 位定时器, 使用 TR1 使能, 溢出时置位 TF1



7 中断

MPC82x5x 提供 7 个中断源, 4 级优先级。每个中断源都有两个对应的位来定义它的优先级。一个在 **IPH**, 另一个在 **IP**。处理高优先级中断时, 不会响应低优先级的中断请求。如果两个不同优先级的中断同时发出请求, 高优先级的中断请求将会被响应。如果相同优先级的中断同时发出请求, 则由内部优先级来决定哪个中断会被响应。下表说明了内部优先级和中断向量地址

中断源	中断向量地址	中断内部优先级
外部中断 0	03H	1(最高)
定时器 0	0BH	2
外部中断 1	13H	3
定时器 1	1BH	4
串口	23H	5
SPI/ADC	2BH	6
PCA/LVD	33H	7

7.1 相关特殊寄存器

■ IE(0xA8)

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit0
EA	EPCA_LVD	ESPI_ADC	ES	ET1	EX1	ET0	EX0

EA: 0: 禁止所有中断

1: 使能中断

EPCA_LVD: PCA 和低压中断使能位

0: 禁止

1: 使能

ESPI_ADC: SPI 和 ADC 中断使能位

0: 禁止

1: 使能

ES: 串口中断使能位

0: 禁止

1: 使能

ET1: 定时器 1 中断使能位

0: 禁止

1: 使能

EX1: 外部中断使能位

0: 禁止

1: 使能

ET0: 定时器 0 中断使能位

0: 禁止

1: 使能
 EX0: 外部中断 0 使能位
 0: 禁止
 1: 使能

■ AUXR(0x8E): 辅助寄存器

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit0
T0X12	T1X12	URM0X6	EADCI	ESPI	ENLVFI	---	---

EADCI: 0: 禁止 ADC 中断

1: 使能 ADC 中断

ESPI: 0: 禁止 SPI 中断

1: 使能 SPI 中断

ENLVFI 0: 禁止低压检测中断

1: 使能低压检测中断

■ IP(0xB8) 中断优先级低位

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit0
---	PPCA_LVD	PSPI_ADC	PS	PT1	PX1	PT0	PX0

■ IPH(0xB7) 中断优先级高位

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit0
---	PPCAH_LVD	PSPIH_ADC	PSH	PT1H	PX1H	PT0H	PX0H

IPH.x, IP.x	优先级
1,1	1(最高)
1,0	2
0,1	3
0,0	4

7.2 中断入口定义

汇编语言

```
CSEG    AT 0000h    ;程序起始地址
JMP     Start
CSEG    AT 0003h    ;/INT0 中断向量地址
JMP     INT0_isr
CSEG    AT 000Bh    ;T0 中断向量地址
JMP     T0_isr
CSEG    AT 0013h    ;/INT1 中断向量地址
JMP     INT1_isr
CSEG    AT 001Bh    ;T1 中断向量地址
JMP     T1_isr
CSEG    AT 0023h    ;串口 中断向量地址
JMP     UART_isr
CSEG    AT 002Bh    SPI 和 ADC 中断向量地址
JMP     SPI_ADC_isr
CSEG    AT 0033h    ;PCA 和 LVD 中断向量地址
JMP     PCA_LVD_isr
Start:   ;...          ;程序开始
MainLoop: ;...          ;程序主循环
        JMP     MainLoop
INT0_isr: ;/INT0 中断处理程序
        ;....
        RETI
T0_isr:   ; T0 中断处理程序
        ;....
        RETI
INT1_isr: ;/INT1 中断处理程序
        ;....
        RETI
T1_isr:   ; T1 中断处理程序
        ;....
        RETI
UART_isr: ;串口中断处理程序
        ;....
        RETI
SPI_ADC_isr: ;SPI 和 ADC 中断处理程序
        ;....
        RETI
```

PCA_LVD_isr: ;PCA 和 LVD 中断处理程序

;....
RETI

C 语言

```
#include "REG_MPC82L52.H"

/*INT0 中断处理程序*/
void INT0_isr (void) interrupt 0
{
    //在此加入用户处理程序
}
/*T0 中断处理程序*/
void T0_isr (void) interrupt 1
{
    //在此加入用户处理程序
}
/*INT1 中断处理程序*/
void INT1_isr (void) interrupt 2
{
    //在此加入用户处理程序
}
/*T1 中断处理程序*/
void T1_isr (void) interrupt 3
{
    //在此加入用户处理程序
}
/*串口 中断处理程序*/
void UART_isr (void) interrupt 4
{
    //在此加入用户处理程序
}
/*SPI 和 ADC 中断处理程序*/
void SPI_ADC_isr (void) interrupt 5
{
    //在此加入用户处理程序
}
/*PCA 和 LVD 中断处理程序*/
void PCA_LVD_isr (void) interrupt 6
{
    //在此加入用户处理程序
}
/*主程序*/
void main (void)
{
```

```
//在此加入用户处理程序  
}
```

8 IAP/ISP 应用

笙泉 8051 芯片的存储空间被分为如下 3 类

- **AP 空间:** 用于存放客户应用程序和数据, 这部分数据可由编程器和 ISP 程序进行擦除和读写
- **IAP 空间:** 是一段非易失性的数据存储空间, 可当做 **EEPROM** 使用。这部分数据可由编程器、ISP 程序以及 AP 应用程序进行擦除和读写
- **ISP 空间:** 是一段特殊的存储空间, **可独立运行程序代码**, 一般是用于对 **AP 和 IAP 空间进行在线编程**, 而 ISP 本身的存储空间只能用编程器来进行编程

8.1 相关特殊寄存器

- **IFD(0xEA):** ISP/IAP 操作的数据
- **IFADRH(0xEB), IFDADRL(0xEC):** ISP/IAP 操作的地址
- **IFMT(0xED):** ISP/IAP 操作模式表
 - =xxxxxx00: 静态
 - =xxxxxx01: 读数据
 - =xxxxxx10: 写数据
 - =xxxxxx11: 页面(512Bytes)擦除
- **SCMD(0xEE):** ISP/IAP 操作时的命令触发寄存器, 当顺序写入 0x46, 0xB9 后, 如果 ISP 使能 (ISPCR.7 = 1), 将会启动 ISP 操作。
- **ISPCR(0xEF):** ISP/IAP 控制命令寄存器

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit0
ISPEN	SWBS	SWRST	CFAIL	---	WAIT2	WAIT1	WAIT0

ISPEN: 置 1 时, ISP 使能。

SWBS: 0: 芯片从 **AP** 空间启动;
1: 芯片从 **ISP** 空间启动;

SWRST: 置 1 时, 芯片将会复位; 硬件自动清零

CFAIL: 上一次 ISP 操作结果标志
0: 成功; 1: 失败

WAIT2,1,0: ISP 忙等待时间表

ISPCR.2:0	CPU 等待时间(时钟周期)			
	页擦除	写	读	对应系统时钟
000	672384	1760	2	30M~24M
001	504288	1320	2	24M~20M
010	420240	1100	2	20M~12M
011	252144	660	2	12M~6M
100	126072	330	2	6M~3M
101	63036	165	2	3M~2M
110	42024	110	2	2M~1M
111	21012	55	2	<1M

8.2 IAP/ISP 基本操作(汇编)

```
$include(REG_MPC82L52.INC) ;包含 MPC82x52 特殊寄存器的定义文件
ISP_READ EQU 1 ;读
ISP_WRITE EQU 2 ;写, 字节为空(=0xFF)才能写进去
ISP_ERASE EQU 3 ;页面(512Bytes)擦除, 要某字节为空, 只能擦除该字
;节所在的整个页面。
ISP_WAIT_TIME EQU 3 ;设置等待时间。 具体数值参照“ISP 忙等待时间表”
;此处系统时钟为 11.0592Mhz
```

;字节读

```
MOV IFADRH, #BYTE_ADDR_H ;送地址高字节
MOV IFADRL, #BYTE_ADDR_L ;送地址低字节
CLR EA ;关中断
;在这请加入软件陷阱判断, 如出错, 则不进行以下的操作了, 可让芯片软复位
MOV ISPCR, #ISP_WAIT_TIME ;设置等待时间
ORL ISPCR, #10000000B ;允许 ISP/IAP 操作
MOV IFMT, #ISP_READ ;送读命令
;在这请加入软件陷阱判断, 如出错, 则不进行以下的操作了, 可让芯片软复位
MOV SCMD, #46h ;先送 46h
;在这请加入软件陷阱判断, 如出错, 则不进行以下的操作了, 可让芯片软复位
MOV SCMD, #B9h ;再送 B9h, ISP/IAP 命令被触发启动
;CPU 等待 ISP/IAP 动作完成, 此时程序被挂起。
```

```
NOP ;
MOV ISPCR, #00000000B ;清 ISP/IAP 特殊寄存器, 防止误操作
SETB EA ;开中断
MOV A, IFD ;将读出的数据送到 ACC
```

;页面擦除, 擦除指定地址所在的页面。没有字节擦除, 只能页面擦除, 512 字节/页面,
;如果要对某个页面擦除, 而其中有些字节又需要保留, 则需在擦除前将其读到 RAM 中
;保存, 然后再擦除页面, 最后再将保存的数据写回该页面。

```
MOV IFADRH, #BYTE_ADDR_H ;送地址高字节
MOV IFADRL, #BYTE_ADDR_L ;送地址低字节
CLR EA ;关中断
;在这请加入软件陷阱判断, 如出错, 则不进行以下的操作了, 可让芯片软复位
MOV ISPCR, #ISP_WAIT_TIME ;设置等待时间
ORL ISPCR, #10000000B ;允许 ISP/IAP 操作
MOV IFMT, #ISP_ERASE ;送页面擦除命令
;在这请加入软件陷阱判断, 如出错, 则不进行以下的操作了, 可让芯片软复位
MOV SCMD, #46h ;先送 46h
;在这请加入软件陷阱判断, 如出错, 则不进行以下的操作了, 可让芯片软复位
MOV SCMD, #B9h ;再送 B9h, ISP/IAP 命令被触发启动
;CPU 等待 ISP/IAP 动作完成, 此时程序被挂起。
```

```
NOP ;
MOV ISPCR, #00000000B ;清 ISP/IAP 特殊寄存器, 防止误操作
SETB EA ;开中断
```

;写字节到指定地址, 该地址必须为空(0xFF),否则要先执行页面擦除

```
MOV    IFD,          #TEST_BYTE
MOV    IFADRH,        #BYTE_ADDR_H    ;送地址高字节
MOV    IFADRL,        #BYTE_ADDR_L    ;送地址低字节
CLR     EA            ;关中断
```

;在这请加入软件陷阱判断, 如出错, 则不进行以下的操作了, 可让芯片软复位

```
MOV    ISPCR,         #ISP_WAIT_TIME   ;设置等待时间
ORL     ISPCR,         #10000000B       ;允许 ISP/IAP 操作
MOV    IFMT,          #ISP_WRITE       ;送页面擦除命令
```

;在这请加入软件陷阱判断, 如出错, 则不进行以下的操作了, 可让芯片软复位

```
MOV     SCMD,         #46h              ;先送 46h
```

;在这请加入软件陷阱判断, 如出错, 则不进行以下的操作了, 可让芯片软复位

```
MOV     SCMD,         #B9h              ;再送 B9h,  ISP/IAP 命令被触发启动
;CPU 等待 ISP/IAP 动作完成, 此时程序被挂起。
```

```
NOP                                     ;
MOV     ISPCR,         #00000000B       ;清 ISP/IAP 特殊寄存器, 防止误操作
SETB    EA            ;开中断
```

8.3 IAP/ISP 操作实例(C 语言)

通过串口发送命令, 读写 MPC82x5x 里 IAP(相当于 EEPROM)的内容。

代码如下:

```
#include <Intrins.h>
#include "REG_MPC82L52.H"

typedef unsigned char    uchar;
typedef unsigned short   ushort;
typedef unsigned long    ulong;
union WTYPE
{
    uchar B[2];
    ushort W;
};

union DWTYPE
{
    uchar B[4];
    ushort W[2];
    ulong DW;
};

#define OscFreq          11059200L //系统时钟
#define TClock           12
```

```
#define T0OVER          1000

#define BuadRate        9600L      //串口波特率

//Timer0 用作定时器
#define TIMER0_TH0      (uchar)(((65536-(OscFreq/TClock)/T0OVER)/256)
#define TIMER0_TL0      (uchar)(((65536-(OscFreq/TClock)/T0OVER)%256)

//Timer1 用作波特率,
#define TIMER1_TH1      (uchar)(256-(OscFreq/(BuadRate*32*12)))

//For ISP/IAP
#define  ISP_READ        1          //读
#define  ISP_WRITE       2          //写, 字节为空(=0xFF)才能写进去
#define  ISP_ERASE       3          //页面(512Bytes)擦除, 要某字节为空, 只能擦除该字
                                     //节所在的整个页面。
#define  ISP_WAIT_TIME  3          //设置等待时间。 具体数值参照"ISP 忙等待时间表"
                                     //此处系统时钟为 11.0592Mhz

//变量定义
uchar IAPBuf[16];
uchar CheckRule[3];
ushort CurrentMillSceond;
uchar RxBuf[16];
uchar RxBufIn;
uchar RxBufOut;
uchar CurrentStatus;
union DWTYPE IAPCmd;

//函数声明
void WriteByte(unsigned short ByteAddr,unsigned char ByteData);
void EasePage(unsigned short ByteAddr);
unsigned char ReadByte(unsigned short ByteAddr);
void SoftTrap( void );
void SendByte(uchar ToSend);
void SendInf(uchar *pInf);
void InitSystem();

const uchar MAIANSTART[]="Now Start Program!! ";
```

//读指定地址的字节

```
unsigned char ReadByte(unsigned short ByteAddr)
{
    IFADRH = ByteAddr>>8;      //送地址高字节
    IFADRL= ByteAddr;          //送地址低字节;
    EA = 0;                    //关中断
    SoftTrap();                //在这请加入软件陷阱
    ISPCR = ISP_WAIT_TIME+0x80; //设置等待时间, 允许 ISP/IAP 操作
    IFMT = ISP_READ;
    SoftTrap();                //在这请加入软件陷阱
    SCMD = 0x46;
    SoftTrap();                //在这请加入软件陷阱
    SCMD = 0xB9;
    _nop_();
    ISPCR = 0;                 //清 ISP/IAP 特殊寄存器, 防止误操作
    EA = 1;                   //开中断
    return IFD;
}
```

//页面擦除, 擦除指定地址所在的页面。

```
void EasePage(unsigned short ByteAddr)
{
    IFADRH = ByteAddr>>8;      //送地址高字节
    IFADRL= ByteAddr;          //送地址低字节;
    EA = 0;                    //关中断
    SoftTrap();                //在这请加入软件陷阱
    ISPCR = ISP_WAIT_TIME+0x80; //设置等待时间, 允许 ISP/IAP 操作
    IFMT = ISP_ERASE;
    SoftTrap();                //在这请加入软件陷阱
    SCMD = 0x46;
    SoftTrap();                //在这请加入软件陷阱
    SCMD = 0xB9;

    _nop_();
    ISPCR = 0;                 //清 ISP/IAP 特殊寄存器, 防止误操作
    EA = 1;                   //开中断
}
```

//写字节到指定地址, 该地址必须为空(0xFF),否则要先执行页面擦除

```
void WriteByte(unsigned short ByteAddr,unsigned char ByteData)
{
    IFD = ByteData;           //送所要写的数据
    IFADRH = ByteAddr>>8;     //送地址高字节
    IFADRL= ByteAddr;         //送地址低字节;
    EA = 0;                   //关中断
    SoftTrap();               //在这请加入软件陷阱
    ISPCR = ISP_WAIT_TIME+0x80; //设置等待时间, 允许 ISP/IAP 操作
    IFMT = ISP_WRITE;
    SoftTrap();               //在这请加入软件陷阱
    SCMD = 0x46;
    SoftTrap();               //在这请加入软件陷阱
    SCMD = 0xB9;
    _nop_();
    ISPCR = 0;                //清 ISP/IAP 特殊寄存器, 防止误操作
    EA = 1;                   //开中断
}
```

//软件陷阱, 如果 CheckRule != "Win",则表示是非法进入,单片机软复位

```
void SoftTrap( void )
{
    if ((CheckRule[0]!='W')||((CheckRule[1]!='i')||(CheckRule[2]!='n'))
    {
        //软件复位
        ISPCR = ISPCR&0xBF;           //SWBS=0, 选择从 AP 空间启动
        ISPCR = ISPCR|0x20;          //SWRST=1, 触发软件复位
    }
}
```

```
void InitSystem()
{
    CheckRule[1]='i';

    //T0 16 位定时器模式
    //T1 8 位, 自动重载 模式
    TMOD = 0x21;

    TH0 =TIMER0_TH0;           //1ms overflow
    TL0 =TIMER0_TL0;
```

```
TH1 = TIMER1_TH1;

/*以下为串口配置*/
SCON = 0x50;           //MODE1 1,8,1, 运行接收

RI = 0;
TI = 0;

/*以下为中断配置打开 T0,串口中断,其它都关闭*/
IE = 0x12;

TR0 = 1;//启动 Timer0
TR1 = 1;//启动 Timer1

EA = 1;  //打开中断

/*初始化变量*/

P1 = 0xFF;
P3 = 0xFF;
}

/*Timer0 中断处理程序*/
void time0ISR(void) interrupt 1
{
    TF0=0;
    TR0=0;
    TH0=TIMER0_TH0;
    TL0=TIMER0_TL0;
    TR0=1;
    CurrentMillSceond++;
}

/*串口中断处理程序*/
void SerISR(void) interrupt 4 using 2
{
    if (TI)
    {
        TI=0;
    }
    else
    {
        /*接收到一个 Byte,把它存到 RxBuf 中去
        RxBuf[RxBufln] = SBUF;
        RxBufIn++;
    }
}
```

```
        if(RxBufln>=16)
        {
            RxBufln = 0;
        }
        RI = 0;
    }
}

/*串口发送一个 Byte*/
void SendByte(uchar ToSend)
{
    EA = 0;
    SBUF = ToSend;
    while(TI == 0)
    {
    }
    TI = 0;
    EA = 1;
}

/*串口发送一个字符串*/
void SendInf(uchar *pInf)
{
    while(*pInf != 0)
    {
        SendByte(*pInf);
        pInf++;
    }
}

void main()
{
    uchar RxData,i;
    CheckRule[0]='W';
    InitSystem();
    SendInf(MAIANSTART); //发送字符串
    while(1)
    {
        if(CurrentMillSceond >= 200)
        {
            CurrentMillSceond = 0;
            P1=~P1;
        }
    }
}
```

```
if(RxBufIn != RxBufOut)
{
    RxData = RxBuf[RxBufOut];
    RxBufOut++;
    if(RxBufOut >= 16)
    {
        RxBufOut = 0;
    }
    switch(CurrentStatus)
    {
        case0:
            //获取命令头
            if(RxData == 0xA5)
            {
                CurrentStatus ++;
            }
            break;
        case1:
            if(RxData == 0x5A)
            {
                CurrentStatus ++;
                i = 0;
            }
            else
            {
                CurrentStatus = 0;
            }
            break;
        case2:
            //获取命令串
            IAPCmd.B[i] = RxData;
            i++;
            if(i>=4)
            {
                if(IAPCmd.B[0] == 1)
                { //读命令
                    for(i=0;i<IAPCmd.B[1];i++)
                    {
                        CheckRule[2]='n';
                        RxData = ReadByte(IAPCmd.W[1]);
                        CheckRule[2]=0;
                        SendByte(RxData);
                        IAPCmd.W[1] ++;
                    }
                }
            }
        }
    }
}
```

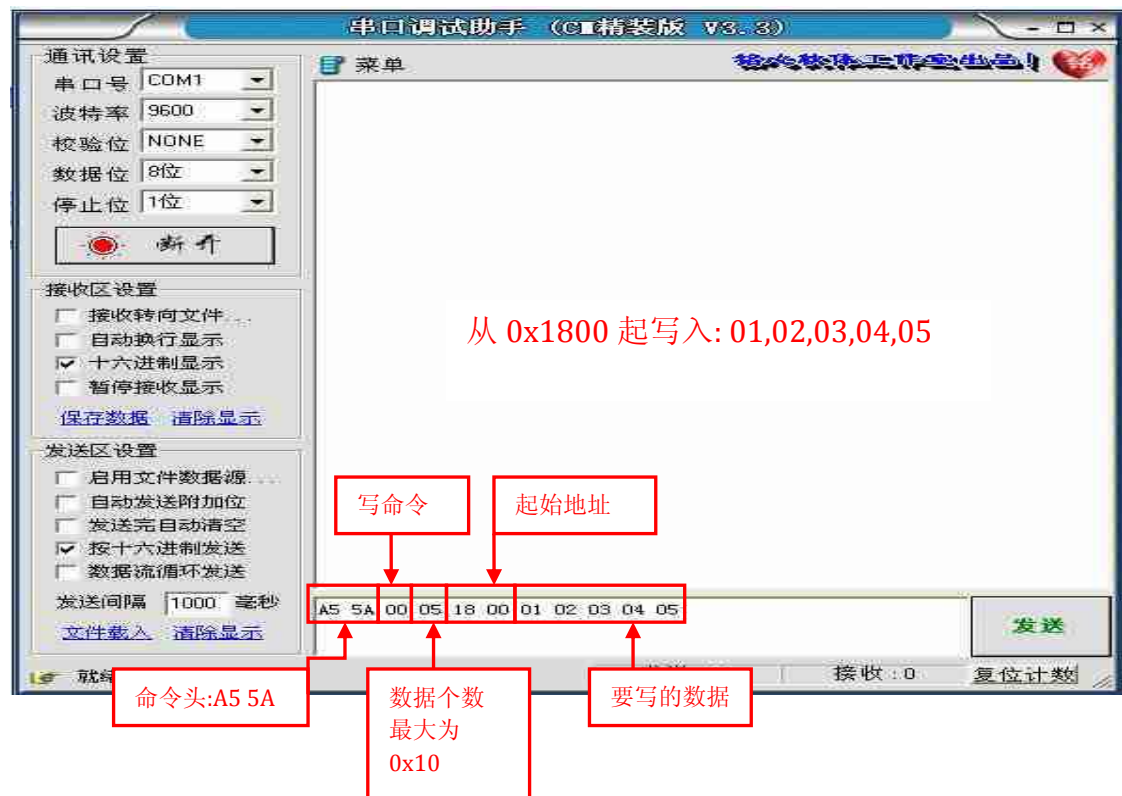
```
        CurrentStatus = 0;
    }
    else if(IAPCmd.B[0] == 0)
    { //写命令
        CurrentStatus ++;
        i = 0;
    }
    else
    {
        CurrentStatus = 0;
    }
}
break;
case 3:
    //获取数据
    IAPBuf[i] = RxData;
    i++;
    if((i>=16)||i>=IAPCmd.B[1])
    { //获取数据结束，将这些数据写到 IAP 中去
        CheckRule[2]='n';
        EasePage(IAPCmd.W[1]);
        for(i=0;i<IAPCmd.B[1];i++)
        {
            CheckRule[2]='n';
            WriteByte(IAPCmd.W[1],IAPBuf[i]);
            CheckRule[2]=0;
            IAPCmd.W[1]++;
        }
        CurrentStatus = 0;
    }
    break;
}
}
}
```

以上代码经 uVision3 V3.30a 编译通过

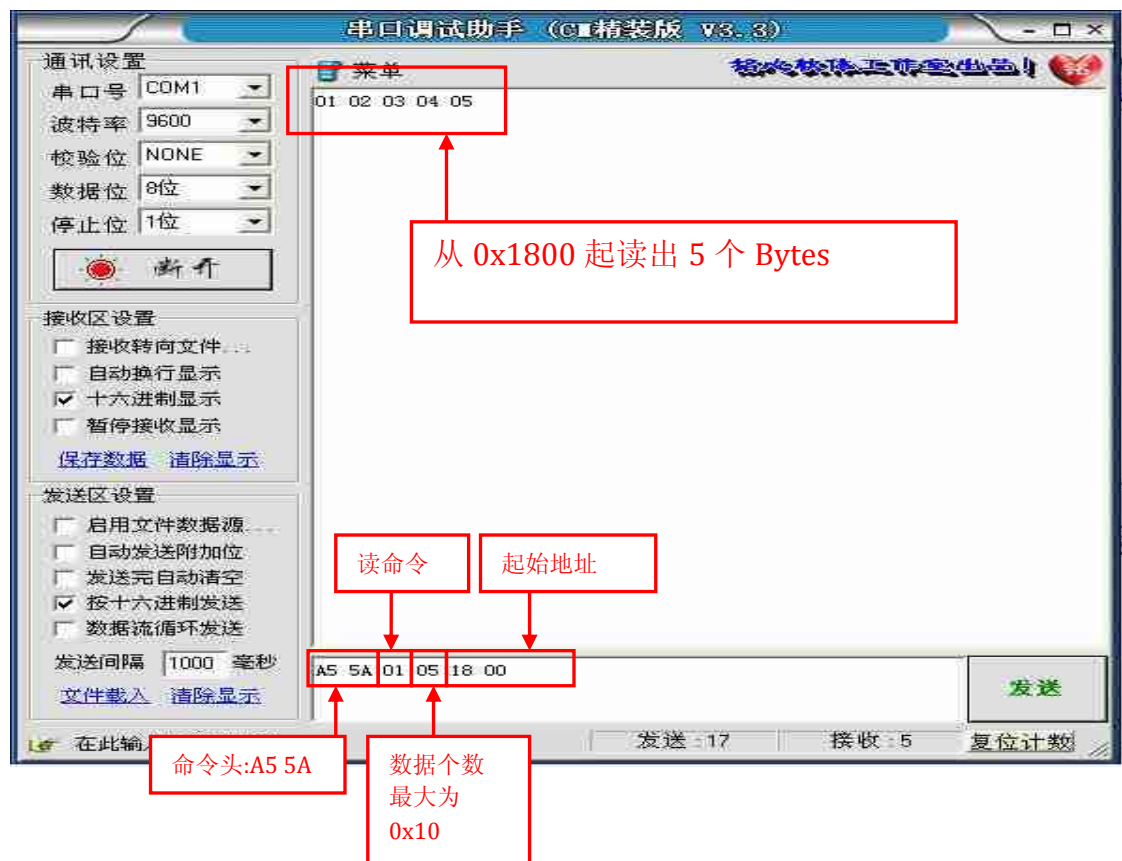
使用出厂设置（AP 空间 6K,IAP 空间 1K,ISP 空间 1K），未做更改,所以可读写的地址范围为 IAP 空间(0x1800~0x1BFF)。如果想读取更大地址范围的数据，可使用 Megawin 8051 Writer U1 烧录程序时，将 IAP SPACE 调大。例如：将 IAP SPCE 设为 3K, 则可读写的范围就变为 0x1000~0x1BFF 了。

使用串口调试程序测试结果如图:

写测试: 发送命令 A5 5A 00 05 18 00 01 02 03 04 05



读测试: 发送命令 A5 5A 01 05 18 00



9 串口(UART)的使用

9.1 相关特殊寄存器

■ **SBUF(0x99):** 串口发送, 接收数据寄存器

■ **SCON(0x98):** 串口控制寄存器

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit0
FE/SM0	SM1	SM2	REN	TB8	RB8	TI	RI

FE 帧错误位 当检测到一个无效停止位时 通过 UART 接收器设置该位 但它必须由软件清零 要使该位有效 PCON 寄存器中的 SMOD0 位必须置 1

SM0 和 SM1 定义串口操作模式 要使该位有效 PCON 寄存器中的 SMOD0 必须置 0

SM1 和 SM0 定义串行口操作模式 见下表

SM0	SM1	UART 模式	波特率
0	0	模式 0, 同步移位寄存器	AUXR.5(URM0X6) 0:fosc/12, 1:fosc/2
0	1	模式 1, 8 位 UART	可变, 取决于 Timer1 溢出
1	0	模式 2, 9 位 UART	fosc /64 或 fosc /32
1	1	模式 3, 9 位 UART	可变, 取决于 Timer1 溢出

SM2 在模式 2 和 3 中多处理机通信使能位 在模式 2 或 3 中 若 SM2=1 且接收到的第 9 位数据 RB8 是 0 则 RI 接收中断标志 不会被激活 在模式 1 中 若 SM2=1 且没有接收到有效的停止位 则 RI 不会被激活 在模式 0 中 SM2 必须是 0

REN 允许接收位 由软件置位或清除 REN=1 时允许接收 REN=0 时 禁止接收

TB8 模式 2 和 3 中发送的第 9 位数据 可以按需要由软件置位或清除

RB8 模式 2 和 3 中已接收的第 9 位数据 在模式 1 中 或 sm2=0 RB8 是已接收的停止位 在模式 0 中 RB8 未用

TI 发送中断标志 模式 0 中 在发送完第 8 位数据时 由硬件置位 其它模式中 在发送停止位之初 由硬件置位 在任何模式中 都必须由软件来清除 TI

RI 接收中断标志 模式 0 中 接收第 8 位结束时由硬件置位 其它模式中在接收停止位的中间时刻 由硬件置位.在任何模式(SM2 所述情况除外)必须由软件清除

■ **AUXR(0x8E):** 辅助寄存器

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit0
T0X12	T1X12	URM0X6	EADCI	ESPI	ENLVFI	---	---

URM0X6: 串口在模式 0 下的波特率选择

0(默认): Fosc/12

1: Fosc/2

9.2 波特率的设置

- 将 UART 设置成 Mode1(8 位 UART)或 Mode3(9 位 UART), 即 SCON=01010000,或 11010000
- 将 Timer1 设置成 Mode2(8 位自动重载模式), 即 TMOD=0010xxxx,
- 通过以下公式来计算 TH1:
 - ◆ 当 T1X12(AUXR.6)=0, 即 Timer1 的时钟源为 Fosc/12
$$\text{BaudRate} = 2^{\text{SMOD}} \times \text{Fosc} / (32 \times 12 \times (256 - \text{TH1}))$$

所以:

$$TH1 = 256 - 2^{SMOD} \times Fosc / (BaudRate \times 32 \times 12)$$

- ◆ 当 T1X12(AUXR.6)=1, 即 Timer1 的时钟源为 Fosc

$$BaudRate = 2^{SMOD} \times Fosc / (32 \times (256 - TH1))$$

所以:

$$TH1 = 256 - 2^{SMOD} \times Fosc / (BaudRate \times 32)$$

■ 部分晶振, 波特率, TH1 关系对应表

波特率	11.0592MHz				18.432MHz				22.1184MHz			
	T1X12=0		T1X12=1		T1X12=0		T1X12=1		T1X12=0		T1X12=1	
	SMOD=0	SMOD=1	SMOD=0	SMOD=1	SMOD=0	SMOD=1	SMOD=0	SMOD=1	SMOD=0	SMOD=1	SMOD=0	SMOD=1
300	160	64	---	---	96	---	---	---	64	---	---	---
600	208	160	---	---	176	96	---	---	160	64	---	---
1200	232	208	---	---	216	176	---	---	208	160	---	---
1800	240	224	64	---	---	---	---	---	224	192	---	---
2400	244	232	112	---	236	216	16	---	232	208	---	---
4800	250	244	184	112	246	236	136	16	244	232	112	---
7200	252	248	208	160	---	---	176	96	248	240	160	64
9600	253	250	220	184	251	246	196	136	250	244	184	112
14400	254	252	232	208	---	---	216	176	252	248	208	160
19200	---	253	238	220	---	251	226	196	253	250	220	184
38400	---	---	247	238	---	---	241	226	---	253	238	220
57600	---	255	250	244	---	---	246	236	255	254	244	232
115200	---	---	253	250	---	---	251	246	---	255	250	244

“---”: 表示不支持

9.3 串口应用实例(C 语言)

接收串口发过来数据, 同时发送出去。

源代码如下:

```
#include <Intrins.h>
#include "REG_MPC82L52.H"

typedef unsigned char      uchar;
typedef unsigned short     ushort;
typedef unsigned long      ulong;

#define OscFreq             11059200L //系统时钟 ,注意"L"不能少
#define T0Clock             12        //1: FOSC, 12:FOSC/12
#define T1Clock             1        //1: FOSC, 12:FOSC/12
#define T1SMOD              2        //1: 1X 波特率   2:2X 波特率
#define T0OVER              1000
#define BuadRate            115200L   //串口波特率,注意"L"不能少

//Timer0 用作定时器
```

```
#define TIMER0_TH0      (uchar)((65536-(OscFreq/T0Clock)/T0OVER)/256)
#define TIMER0_TL0      (uchar)((65536-(OscFreq/T0Clock)/T0OVER)%256)
//Timer1 用作波特率,
#define TIMER1_TH1      (uchar)(256-(T1SMOD*OscFreq/(BuadRate*32*T1Clock)))
//变量定义
ushort CurrentMillSceond;

//函数声明
void SendByte(uchar ToSend);
void SendInf(uchar *pInf);
void InitSystem();
const uchar MAIANSTART[]="Now Start Program!! ";
void InitSystem()
{
    //T0 16 位定时器模式
    //T1 8 位, 自动重载 模式
    TMOD = 0x21;

    TH0 =TIMER0_TH0;           //1ms overflow
    TL0 =TIMER0_TL0;
    TH1 = TIMER1_TH1;
    /*以下为串口配置*/
    #if (T1SMOD == 2)
        PCON = 0x80;
    #else
        PCON = 0x00;
    #endif
    #if (T0Clock == 12)
        AUXR = AUXR & 0xEF;
    #else
        AUXR = AUXR | 0x80;
    #endif
    #if (T1Clock == 12)
        AUXR = AUXR & 0xBF;
    #else
        AUXR = AUXR | 0x40;
    #endif
    SCON = 0x50;               //MODE1 1,8,1, 运行接收
    RI = 0;
    TI = 0;
    /*以下为中断配置打开 T0,串口中断,其它都关闭*/
    IE = 0x12;
    TR0 = 1;//启动 Timer0
    TR1 = 1;//启动 Timer1
```

```
EA = 1; //打开中断
/*初始化变量*/
P1 = 0xFF;
P3 = 0xFF;
}

/*Timer0 中断处理程序*/
void time0ISR(void) interrupt 1
{
    TF0=0;
    TR0=0;
    TH0=TIMER0_TH0;
    TL0=TIMER0_TL0;
    TR0=1;
    CurrentMillSceond++;
}
/*串口中断处理程序*/
void SerISR(void) interrupt 4 using 2
{
    if (TI)
    {
        TI=0;
    }
    else
    {
        //接收到一个 Byte,即刻发出去
        SBUF = SBUF;
        RI = 0;
    }
}
/*串口发送一个 Byte*/
void SendByte(uchar ToSend)
{
    EA = 0;
    SBUF = ToSend;
    while(TI == 0)
    {
    }
    TI = 0;
    EA = 1;
}
/*串口发送一个字符串*/
void SendInf(uchar *pInf)
{

```

```

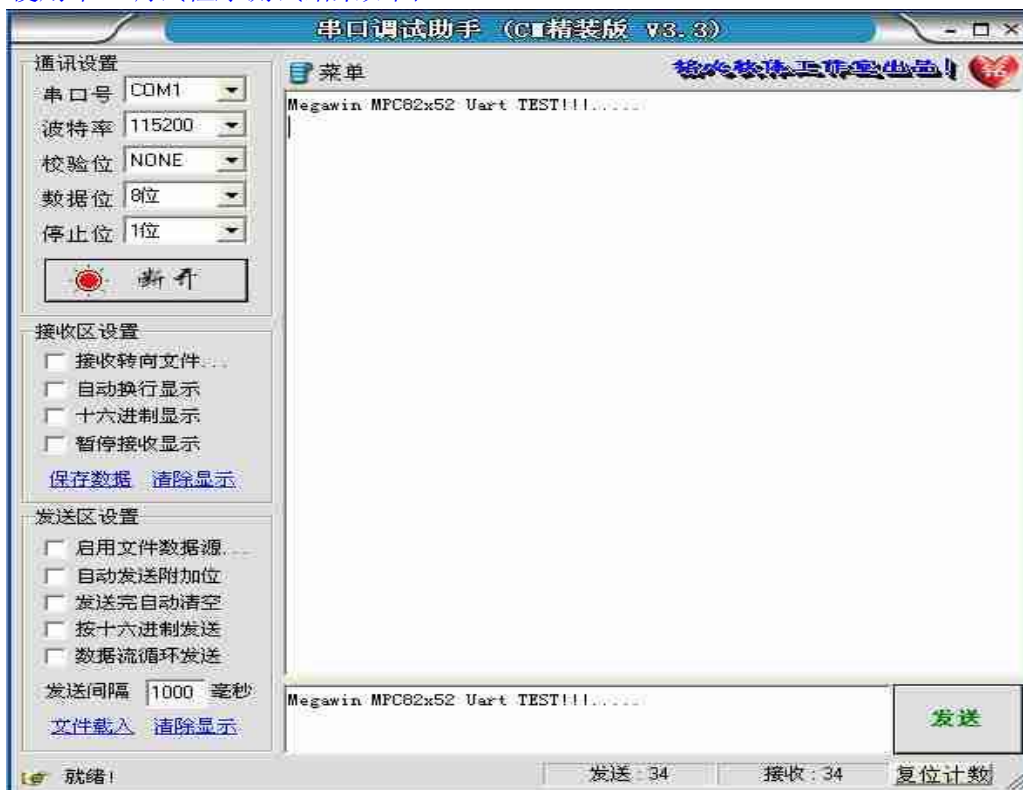
while(*pInf != 0)
{
    SendByte(*pInf);
    pInf++;
}

void main()
{
    InitSystem();
    SendInf(MAIANSTART); //发送字符串
    while(1)
    {

        if(CurrentMillSceond >= 200)
        {
            CurrentMillSceond = 0;
            P1=~P1;
        }
    }
}

```

以上代码经 uVision3 V3.30a 编译通过
使用串口调试程序测试结果如图:



10 ADC 的使用

MPC82x5x 提供了 8 通道的 ADC，与 P1 口共用。当作为 ADC 通道使用时，该 I/O 口必须设为仅输入模式(高阻)。MPC82x52 的精度是 8 位，MPC82x54 的精度是 10 位

10.1 相关特殊寄存器

- **ADCV(0xC6):** ADC 转换结果数据寄存器
- **ADCVL(0xBE):** ADC 转换结果数据低两位寄存器 (仅 MPC82x54 有效)

对于 MPC82x52:

$$ADCV = 256 * ((V_{in} - V_{ss}) / (V_{cc} - V_{ss}))$$

对于 MPC82x54

$$ADCV: ADCVL[1:0] = 1024 * ((V_{in} - V_{ss}) / (V_{cc} - V_{ss}))$$

- **ADCTL(0xC5):** ADC 转换控制寄存器

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit0
ADCON	SPEED1	SPEED0	ADCI	ADCS	CHS2	CHS1	CHS0

ADCON 0: 关闭 ADC 模块的电源; 1: 打开 ADC 模块的电源

SPEED1,SPEED0 ADC 转换速度选择寄存器

MPC82x52

0,0 (default),840 个机器周期

0,1 630 个机器周期

1,0 420 个机器周期

1,1 210 个机器周期

MPC82x54

(default)1080 个机器周期

810 个机器周期

540 个机器周期

270 个机器周期

AADCI ADC 完成中断标志。当 ADC 完成后，该位置 1。必须由软件清零。

ADCS 置 1 后，ADC 开始转换，转换完成后自动清零。

CHS2,CHS1,CHS0 通道选择位

0,0,0 设置 P1.0 为 ADC 通道(default)

0,0,1 设置 P1.1 为 ADC 通道

0,1,0 设置 P1.2 为 ADC 通道

0,1,1 设置 P1.3 为 ADC 通道

1,0,0 设置 P1.4 为 ADC 通道

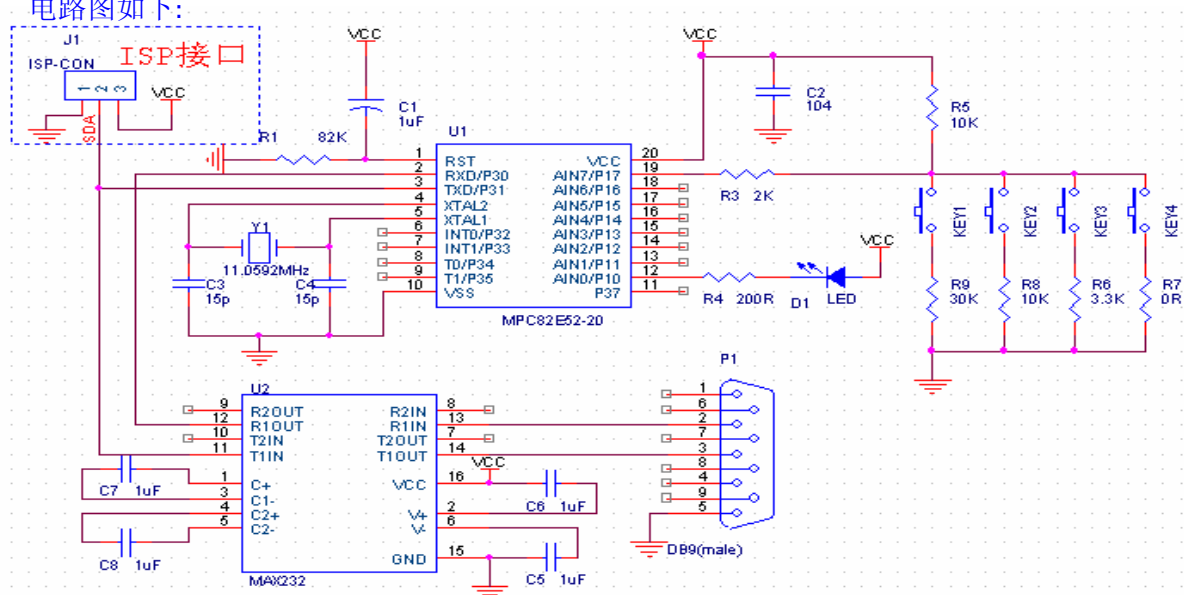
1,0,1 设置 P1.5 为 ADC 通道

1,1,0 设置 P1.6 为 ADC 通道

1,1,1 设置 P1.7 为 ADC 通道

10.2 ADC 应用实例—按键输入(C 语言)

电路图如下:



功能说明:按键按下时发送按键值到串口, 并且灯亮, 松开时发送 0x00 到串口, 并且灯灭。

源代码如下:

```
#include <Intrins.h>
#include "REG_MPC82L52.H"

typedef unsigned char    uchar;
typedef unsigned short   ushort;
typedef unsigned long     ulong;

#define OscFreq          11059200L //系统时钟
#define T0Clock          12        //1: FOSC, 12:FOSC/12
#define T1Clock          1         //1: FOSC, 12:FOSC/12
#define T1SMOD           2         //1: 1X 波特率   2:2X 波特率
#define T0OVER           1000
#define BuadRate          115200L   //串口波特率
//Timer0 用作定时器
#define TIMER0_TH0        (uchar)(((65536-(OscFreq/T0Clock)/T0OVER)/256)
#define TIMER0_TL0        (uchar)(((65536-(OscFreq/T0Clock)/T0OVER)%256)
//Timer1 用作波特率,
#define TIMER1_TH1        (uchar)(256-(T1SMOD*OscFreq/(BuadRate*32*T1Clock)))
//变量定义
ushort CurrentMillSecond;
ushort KeyMillSecond;
uchar   KeyStatus;
uchar   KeyValue;
uchar   KeyOldValue;
uchar   KeyOld;
//函数声明
uchar Get_ADC_Channel(char channel);
```

```
void SendByte(uchar ToSend);
void InitSystem();
uchar KeyScan();
void KeyBoard();
/*获取指定通道的 ADC 值*/
uchar Get_ADC_Channel(char channel)
{
    channel &= 0x07;                //确定通道为 0-7
    ADCTL = 0x88|channel;           //开始转换
    while(!(ADCTL & 0x10)){         //检查转换是否完成
        return ADC;
    }
}
/*串口发送一个 Byte*/
void SendByte(uchar ToSend)
{
    EA = 0;
    SBUF = ToSend;
    while(TI == 0){}
    TI = 0;
    EA = 1;
}
/*Timer0 中断处理程序*/
void time0ISR(void) interrupt 1
{
    TF0=0;
    TR0=0;
    TH0=TIMER0_TH0;
    TL0=TIMER0_TL0;
    TR0=1;
    CurrentMillSecond++;
    if(CurrentMillSecond >= 1000)
    {
        CurrentMillSecond = 0;
    }
}
void InitSystem()
{
    /*IO 口配置*/
    P1M0 = 0x80;                    //作为 ADC 输入的 IO 口，一定要设成输入(Input Only)模式
    P1M1 = 0x00;                    //此 DEMO 程序只用到 P17 作为 ADC 输入口
    //T0 16 位定时器模式 T1 8 位，自动重载 模式
    TMOD = 0x21;
    TH0=TIMER0_TH0;                 //1ms overflow
    TL0=TIMER0_TL0;
```

```
    TH1 = TIMER1_TH1;
    /*以下为串口配置*/
    #if (T1SMOD == 2)
        PCON = 0x80;
    #else
        PCON = 0x00;
    #endif
    #if (T0Clock == 12)
        AUXR = AUXR & 0xEF;
    #else
        AUXR = AUXR | 0x80;
    #endif
    #if (T1Clock == 12)
        AUXR = AUXR & 0xBF;
    #else
        AUXR = AUXR | 0x40;
    #endif
    SCON = 0x50;                //MODE1 1,8,1, 运行接收
    /*以下为中断配置打开 T0,串口中断,其它都关闭*/
    IE = 0x12;
    TR0 = 1; //启动 Timer0
    TR1 = 1; //启动 Timer1
    EA = 1;  //打开中断
}

ushort IntValMillSecond(ushort MillSecond)
{
    if(MillSecond > CurrentMillSecond)
        return ((1000-MillSecond)+CurrentMillSecond);
    else
        return (CurrentMillSecond-MillSecond);
}

void main()
{
    InitSystem();
    while(1)
    {
        KeyBoard();
        if(KeyOldValue != KeyValue)
        {
            KeyOldValue = KeyValue;
            if(KeyValue != 0)
                P10 = 0;    //按键按下, 灯亮
            else

```

```
P10 = 1;    //按键松开，灯灭
SendByte(KeyValue); //将按键值发送到串口
    }
}
}
uchar KeyScan()
{
    uchar KeyAdcValue;
    KeyAdcValue = Get_ADC_Channel(7);
    if(KeyAdcValue> 0xE0) return 0x00; //NO KEY   >0.875VDD
    if(KeyAdcValue> 0xA0) return 0x01; //KEY1(0.75VDD)   >0.625VDD
    if(KeyAdcValue> 0x60) return 0x02; //KEY2(0.50VDD)   >0.375VDD
    if(KeyAdcValue> 0x20) return 0x03; //KEY3(0.25VDD)   >0.125VDD
    return 0x04;                      //KEY4(0.00VDD)   <0.125VDD
}
enum{
    KEY_IDLE,
    KEY_PRESS_WAIT_TIME,
    KEY_HOLD,
    KEY_UP_WAIT_TIME
};
void KeyBoard()
{
    uchar TKeyValue1;
    TKeyValue1 = KeyScan();
    switch(KeyStatus)
    {
        case KEY_IDLE:
            if(TKeyValue1 != 0)
            {
                KeyStatus = KEY_PRESS_WAIT_TIME;
                KeyMillSecond = CurrentMillSecond;
                KeyOld = TKeyValue1;
            }
            break;
        case KEY_PRESS_WAIT_TIME:
            if(KeyOld == TKeyValue1)
            {
                if(IntValMillSecond(KeyMillSecond)>20)
                {
                    KeyValue = KeyOld;
                    KeyStatus = KEY_HOLD;
                }
            }
    }
}
```

```
        else
        {
            KeyStatus = KEY_IDLE;
        }
        break;
    case KEY_HOLD:
        if(KeyOld != TKeyValue1)
        {
            KeyStatus = KEY_UP_WAIT_TIME;
            KeyMillSecond = CurrentMillSecond;
        }
        break;
    case KEY_UP_WAIT_TIME:
        if(KeyOld != TKeyValue1)
        {
            if(IntValMillSecond(KeyMillSecond)>20)
            {
                KeyValue =0;
                KeyStatus = KEY_IDLE;
            }
        }
        else
        {
            KeyStatus = KEY_HOLD;
        }
        break;
    default:
        KeyStatus = KEY_IDLE;
        break;
    }
}
```

11 PCA 的使用

MPC82x5x 提供的 PCA 包含**两组(MPC82x52)/四组(MPC82x54)**16 位捕捉/比较模块，它们共用一个 16 位定时器，各对应一个 I/O 口。每组模块都能编程设置成 4 种不同的模式。

- 上/下沿捕捉(高/低电平脉冲宽度计算)
- 软件定时器
- 高速输出
- 脉宽调制

11.1 相关特殊寄存器

■ CMOD(0xD9): PCA 模式控制寄存器

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit0
CIDL					CPS1	CPS0	ECF

CIDL 当 MCU 在 IDLE 模式时，0(默认):关闭 PCA； 1: 使能 PCA

CPS1,CPS0 PCA 计数器时钟来源选择:

CPS0 0,0: Fosc/12

0,1: Fosc/2

1,0: Timer0 Overflow

1,1: ECI(P3.4) input

ECF PCA 定时器溢出中断标志使能位

0(默认): PCA 定时器溢出中断标志不会传给 MCU

1: PCA 定时器溢出中断标志会传给 MCU

■ CCON(0xD8): PCA 控制寄存器

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit0
CF	CR			CCF3*	CCF2*	CCF1	CCF0

CF PCA 定时器溢出标志位。必须由软件清零。

CR PCA 工作控制位。0(默认): 关闭, 1: 打开。

CCF3* 模块 3 中断标志位 (仅对 MPC82x54)

CCF2* 模块 2 中断标志位 (仅对 MPC82x54)

CCF1 模块 1 中断标志位

CCF0 模块 0 中断标志位

■ CH(0xF9),CL(0xE9): PCA 16 位计数寄存器

■ CCAP0H(0xFA),CCAP0L(0xEA): PCA 模块 0 计数寄存器

■ CCAP1H(0xFB),CCAP1L(0xEB): PCA 模块 1 计数寄存器

■ CCAP2H(0xFC),CCAP2L(0xEC): PCA 模块 2 计数寄存器 (仅对 MPC82x54)

■ CCAP3H(0xFD),CCAP3L(0xED): PCA 模块 3 计数寄存器 (仅对 MPC82x54)

■ PCAPWM0(0xF2): PWM 模式, 模块 0 辅助寄存器

■ PCAPWM1(0xF3): PWM 模式, 模块 1 辅助寄存器

■ PCAPWM0(0xF4): PWM 模式, 模块 2 辅助寄存器 (仅对 MPC82x54)

■ PCAPWM1(0xF5): PWM 模式, 模块 3 辅助寄存器 (仅对 MPC82x54)

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit0
						EPCnH	EPCnL

EPCnH 当 CL 溢出时,此位将传送给 EPCnL

EPCnL 与 CCAPnL 一起组成个 9 位数, 用来与 CL 作比较来控制 PWM 输出的电平占空比
 当 $CL[7:0] < (EPCnL, CCAPnL[7:0])$, PWM 输出低电平
 否则输出高电平。

*n=0,1

- CCAPM0(0xDA): PCA 模块 0 控制寄存器
- CCAPM1(0xDB): PCA 模块 1 控制寄存器
- CCAPM2(0xDC): PCA 模块 2 控制寄存器 (仅对 MPC82x54)
- CCAPM3(0xDD): PCA 模块 3 控制寄存器 (仅对 MPC82x54)

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit0
	ECOM	CAPP	CAPN	MAT	TOG	PWM	ECCF

ECOMn 0:关闭比较器
 1:打开比较器

CAPPn 当在 CEXn 上升沿时, PCA 计数器的数据是否要传送到模块 n 计数寄存器
 0: 不用; 1: 要

CAPNn 当在 CEXn 下降沿时, PCA 计数器的数据是否要传送到模块 n 计数寄存器
 0: 不用; 1: 要

MATn 当 PCA 计数器与模块 n 计数器比较相同时是否需要置位 CCON 中的 CCFn
 0: 不用; 1: 需要

TOGn 当 PCA 计数器与模块 n 计数器比较相同时是否需要设定 CEXn 引脚
 0: 不用; 1: 需要

PWMn 使能 PWM 模块 n
 0: 禁止; 1:使能

ECCFn 设置 CCON 中的 CCFn 位是否能产生中断
 0:禁止; 1: 使能

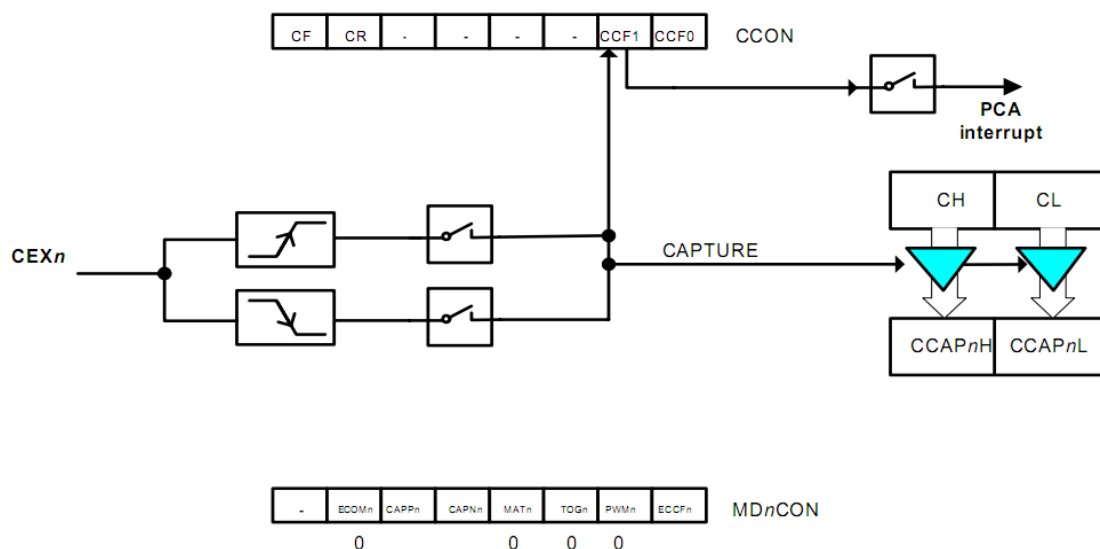
N= 0,1

11.2 PCA 的四种模式

ECOMn	CAPPn	CAPPn	MATn	TOGn	PWMn	ECCFn	功能模式
0	0	0	0	0	0	0	PCA 禁止
x	1	0	0	0	0	X	16 位 CEXn 引脚上升沿捕捉模式
x	0	1	0	0	0	X	16 位 CEXn 引脚下降沿捕捉模式
x	1	1	0	0	0	X	16 位 CEXn 引脚变化捕捉模式
1	0	0	1	0	0	X	16 位软件定时器模式
1	0	0	1	1	0	X	16 位高速输出模式
1	0	0	0	0	1	0	8 位 PWM 模式

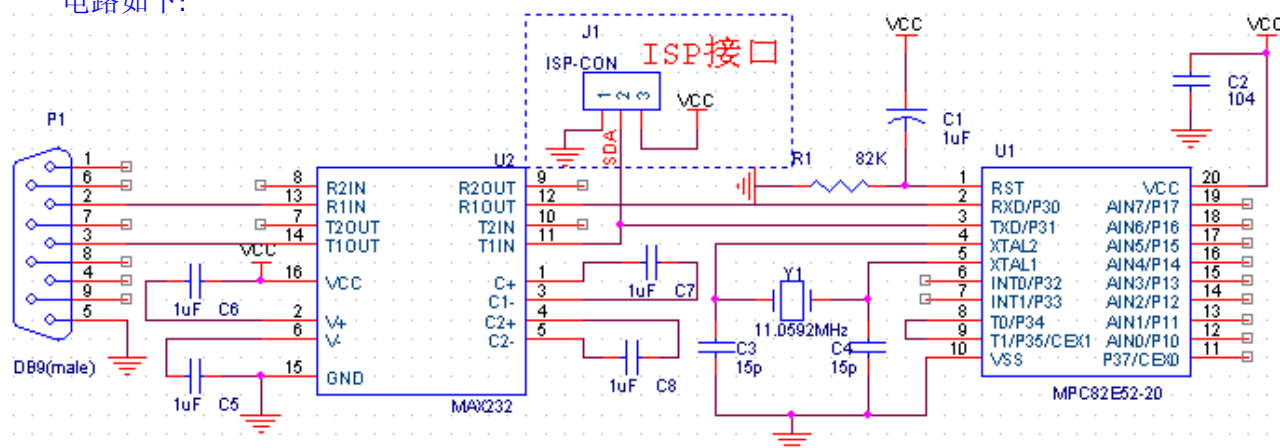
PCA 捕捉模式

要使用捕捉模式，必须将 CAPPn 和 CAPNn 任意一个或全部置 1，而 MATn, TOGn, PWMn 必须设为 0。当 CEXn 引脚发生变化时(CAPPn:CAPNn = 1:0 时仅上升沿，=0:1 时仅下降沿，=1:1 时上升/下降)，PCA 将会把 CH:CL 的数据传送到 CCAPnH:CCAPnL，并且 CCON 中的 CCFn 将会被置位，如果 ECCFn 为 1，则此时会产生 PCA 中断。



■ 示例之捕捉模式(C 语言):

说明: 每 5 秒由 P3.4 口输出一串脉冲，P3.5(CEX1)进行捕捉，然后将捕捉的数据发到串口电路如下:



源代码如下:

```
#include <Intrins.h>
#include "REG_MPC82L52.H"
typedef unsigned char      uchar;
typedef unsigned short     ushort;
typedef unsigned long      ulong;
#define OscFreq            11059200L //系统时钟
#define T0Clock            12        //1: FOSC, 12:FOSC/12
#define T1Clock            1        //1: FOSC, 12:FOSC/12
#define T1SMOD             2        //1: 1X 波特率  2:2X 波特率
#define T0OVER             1000
#define BaudRate           115200L   //串口波特率
//Timer0 用作定时器
#define TIMER0_TH0         (uchar)((65536-(OscFreq/T0Clock)/T0OVER)/256)
#define TIMER0_TL0         (uchar)((65536-(OscFreq/T0Clock)/T0OVER)%256)
//Timer1 用作波特率,
#define TIMER1_TH1         (uchar)(256-(T1SMOD*OscFreq/(BaudRate*32*T1Clock)))
//变量定义
ushort CurrentMillSecond;
uchar  CurrentSecond;
uchar  capture_high[9];
uchar  capture_low[9];
uchar  save_count;
//函数声明
void SendByte(uchar ToSend);
void InitSystem();
/*串口发送一个 Byte*/
void SendByte(uchar ToSend)
{
    EA = 0;
    SBUF = ToSend;
    while(TI == 0){}
    TI = 0;
    EA = 1;
}
/*Timer0 中断处理程序*/
void time0ISR(void) interrupt 1
{
    TF0=0;
    TR0=0;
    TH0=TIMER0_TH0;
    TL0=TIMER0_TL0;
    TR0=1;
    CurrentMillSecond++;
}
```

```
    if(CurrentMillSecond >= 1000)
    {
        CurrentMillSecond = 0;
        CurrentSecond ++;
    }
}

void InitSystem()
{
    //T0 16 位定时器模式 T1 8 位, 自动重载 模式
    TMOD = 0x21;
    TH0=TIMER0_TH0;          //1ms overflow
    TL0=TIMER0_TL0;
    TH1 = TIMER1_TH1;
    /*以下为串口配置*/
    #if (T1SMOD == 2)
        PCON = 0x80;
    #else
        PCON = 0x00;
    #endif
    #if (T0Clock == 12)
        AUXR = AUXR & 0xEF;
    #else
        AUXR = AUXR | 0x80;
    #endif
    #if (T1Clock == 12)
        AUXR = AUXR & 0xBF;
    #else
        AUXR = AUXR | 0x40;
    #endif
    SCON = 0x50;              //MODE1 1,8,1, 运行接收
    /*以下为中断配置打开 T0,串口中断,其它都关闭*/
    IE = 0x12;

    CMOD   =   0x00;           //设置 PCA 时钟来源为 Fosc/12
    CCAPM1 =   0x31;         //配置 PCA 模块 1 为 16 位 CEX1 引脚变化捕捉模式
    EPCALVD =   1;           //使能 PCA 和 LVD 中断
    CCON   |=  0x40;         //启动 PCA

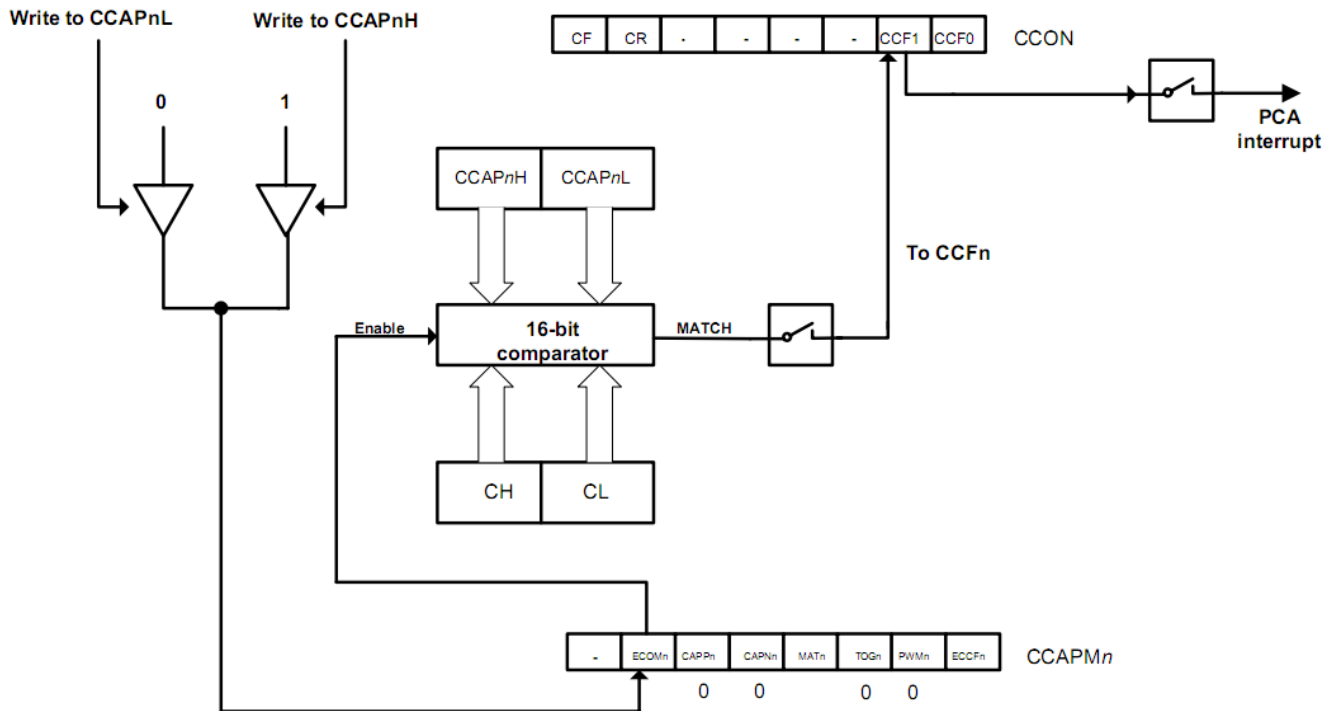
    TR0 = 1; //启动 Timer0
    TR1 = 1; //启动 Timer1
    EA = 1;  //打开中断
}
```

```
void main()
{
    ushort i,invert_count;
    InitSystem();
    while(1)
    {
        if(CurrentSecond >= 5)
        {
            CurrentSecond = 0;
            i = 0;
            invert_count = 0x10;
            P34 = !P34;
            do{
                if(i++ > invert_count)
                {
                    P34 = !P34;
                    invert_count += 0x10;
                    i = 0;
                }
            }while(save_count <9);
            for(i=1;i<save_count;i++)
            {
                SendByte(capture_high[i]);
                SendByte(capture_low[i]);
            }
            save_count = 0;
        }
    }
}

//PCA 中断处理程序
void PCA_Interrupt() interrupt 6 using 1
{
    EA      = 0;
    CCON    = 0x00;                //Stop PCA Counter and clear PCA interrupt flag
    CH = CL = 0;
    capture_high[save_count] = CCAP1H;
    capture_low[save_count] = CCAP1L;
    save_count++;
    EA      = 1;
    CCON    |= 0x40;                //Start PCA Counter
}
```

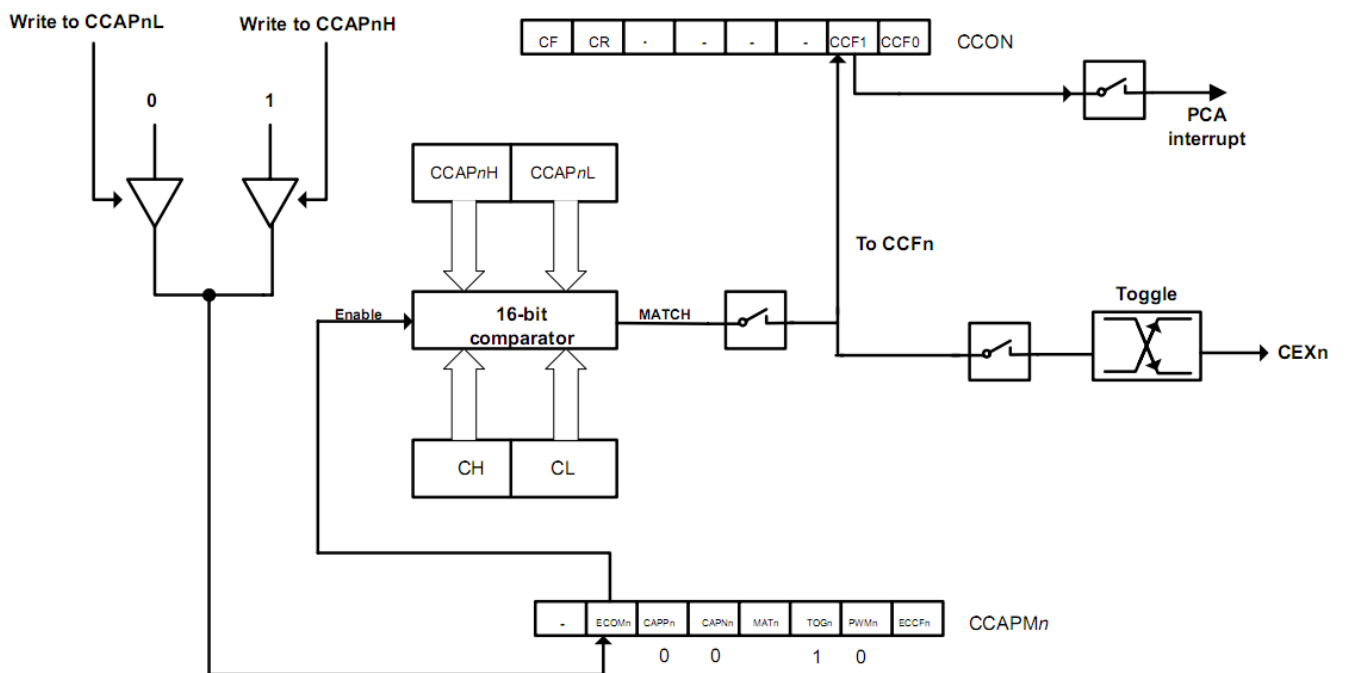
16 位软件定时器模式

将 ECOM_n 和 MAT_n 设为 1, PCA 可用作 16 位软件定时器。如果 CH:CL 等于 CCAP_nH:CCAP_nL, 则 CCF_n 将会被置位, 如果 ECCF_n 为 1, 则此时会产生 PCA 中断。



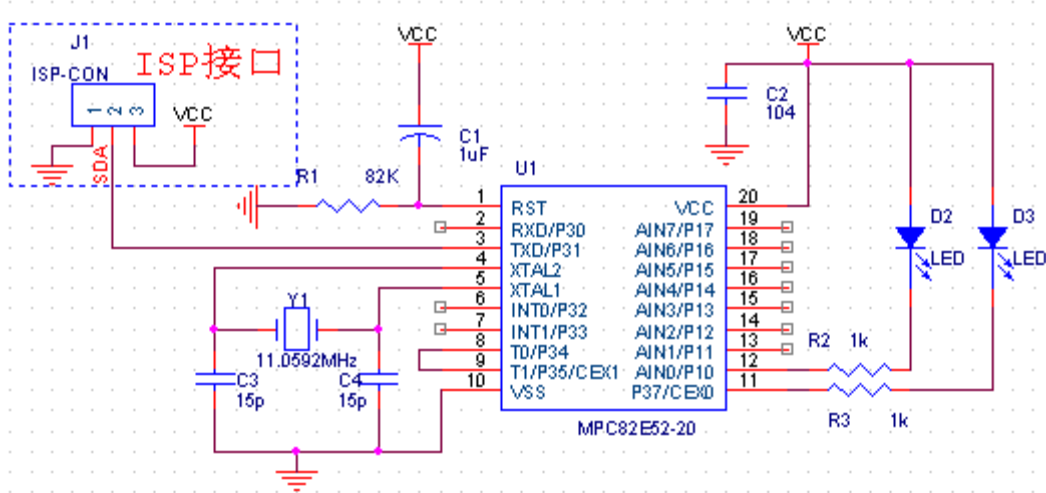
16 位高速输出模式

将 ECOM_n, MAT_n 和 TOG_n 设为 1, PCA 可用作 16 位高速输出。如果 CH:CL 等于 CCAP_nH:CCAP_nL, 则 CEX_n 引脚会反向, 且 CCF_n 将会被置位, 如果 ECCF_n 为 1, 则此时会产生 PCA 中断。



■ 示例之软件定时器和高速输出 (C 语言):

说明: P1.0 输出 200ms 的周期方波,P3.7 输出 2ms 的方波
 电路如下:



源代码如下:

```
#include <Intrins.h>
#include "REG_MPC82L52.H"
typedef unsigned char    uchar;
typedef unsigned short   ushort;
typedef unsigned long    ulong;
#define OscFreq          11059200L //系统时钟

//PCA 用作软件定时器
#define CCAP_OVER         1000L
#define CCAP0H_VALUE     (uchar)(((OscFreq/12)/CCAP_OVER)/256)
#define CCAP0L_VALUE     (uchar)(((OscFreq/12)/CCAP_OVER)%256)

//变量定义
ushort CurrentMillSecond;
ushort P100OutTime;
//函数声明
ushort IntValMillSecond(ushort MillSecond);
void InitSystem();

ushort IntValMillSecond(ushort MillSecond)
{
    if(MillSecond > CurrentMillSecond)
        return ((1000-MillSecond)+CurrentMillSecond);
    else
        return (CurrentMillSecond-MillSecond);
}

void InitSystem()
```

```
{
    CMOD    =    0x00;                //设置 PCA 时钟来源为 Fosc/12
    CCAPM0 =    0x4D;                //配置 PCA 模块 0 为 16 位 16 位高速输出模式

    CCAP0L =    CCAP0L_VALUE;
    CCAP0H =    CCAP0H_VALUE;
    CL = 0;
    CH = 0;

    EPCALVD =    1;                //使能 PCA 和 LVD 中断
    CCON    |=    0x40;            //启动 PCA

    EA = 1; //打开中断
}
void main()
{
    InitSystem();
    while(1)
    {
        if(IntValMillSecond(P10OutTime)>=100)
        {
            P10OutTime = CurrentMillSecond;
            P10 = !P10;

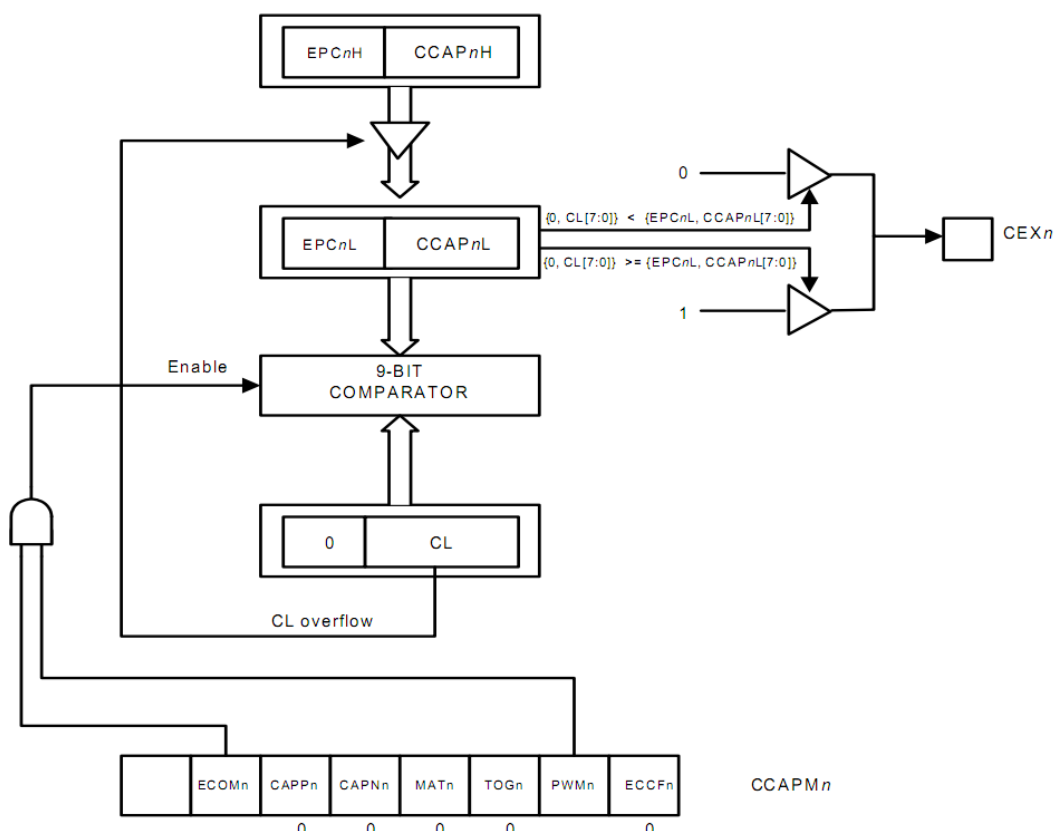
        }

    }
}

void PCA_Interrupt() interrupt 6 using 1
{
    EA      =    0;
    CCON    =    0x00;                //Stop PCA Counter and clear PCA interrupt flag
    CH = CL =    0;
    CurrentMillSecond++;
    if(CurrentMillSecond >= 1000)
    {
        CurrentMillSecond = 0;
    }
    EA      =    1;
    CCON    |=    0x40;                //Start PCA Counter
}
```

8 位 PWM 输出模式

PCA 可设置成 8 位 PWM 输出模式。要设成 PWM 模式, CCAPMn 中的 ECOMn 和 PWMn 必须设成 1,其余的设成 0。它的频率取决于 PCA 计数器的时钟来源, $F_{PWM}=F_{PCA}/256$ 。而它的占空比取决于 EPCnL:CCAPnL。当 $CL[7:0] < \{EPCnL, CCAPnL[7:0]\}$ 时, PWM 输出低电平, 否则输出高电平。当 $CL[7:0]$ 由 0xFF 溢出, 变为 0x00 时, EPCnH:CCAPnH[7:0] 中的数据将会设给 EPCnL:CCAPnL

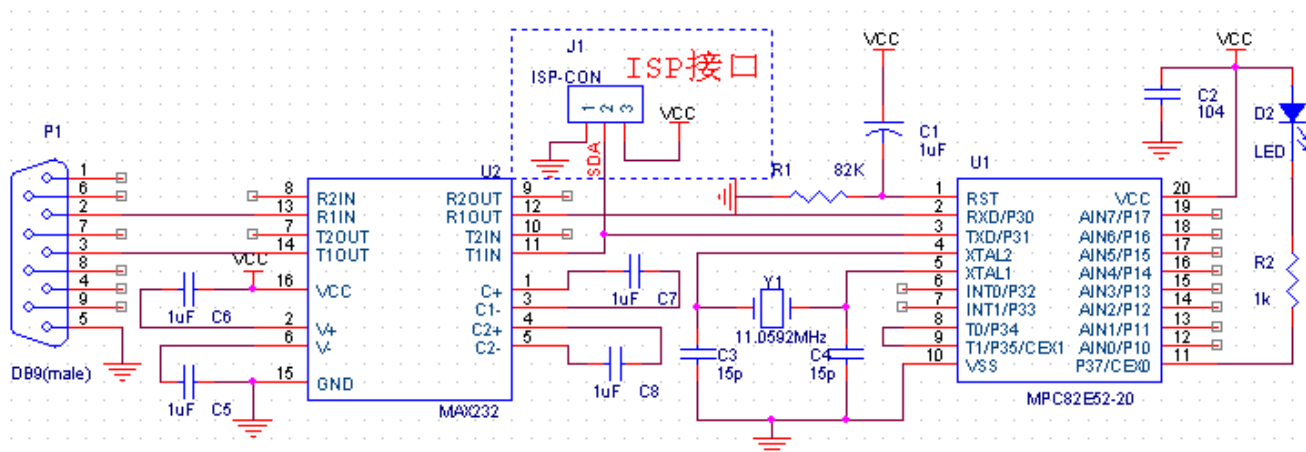


■ PCA 示例之 PWM 输出(C 语言):

说明: PWM 由 P3.7 输出, 频率和占空比通过串口发送命令进行控制。

命令格式为: A5 5A XX YY (XX: 用来控制频率, YY: 用来控制占空比)

电路如下:



源代码如下:

```
#include <Intrins.h>
#include " REG_MPC82L52.H "
typedef unsigned char      uchar;
typedef unsigned short     ushort;
typedef unsigned long      ulong;
#define OscFreq            11059200L //系统时钟
#define BuadRate           9600L     //串口波特率
//Timer1 用作波特率,
#define TIMER1_TH1         (uchar)(256-(OscFreq/(BuadRate*32*12)))
//变量定义
uchar RxBuf[16];
uchar RxBufIn;
uchar RxBufOut;
uchar RxData;
uchar CurrentStatus;
//函数声明
void InitSystem();
void InitSystem()
{
    TMOD    =    0x22;           //设置 Timer0,Timer1 是 8 位重载模式
    TL0     =    0x80;           //Timer0 用确定做 PWM 的频率
    TH0     =    0x80;           //Fpwm = (OscFreq/12)/((256-TH0)*256)
                                   //此处频率约位 28Hz

    TH1 = TIMER1_TH1;

    CMOD    =    0x04;           //设置 PCA 时钟来源为 Timer0 溢出
    CCAPM0 =    0x42;           //配置 PCA 模块 0 为 8 位 PWM 输出模式
    CCAP0L =    0x80;           //设置 CCAP0, 来设定 PWM 的占空比
    CCAP0H =    0x80;           //此处占空比约为 1:1
    CL = 0;
    CH = 0;
    EPCALVD =    0;             //关闭 PCA 和 LVD 中断
    CCON    |=    0x40;         //启动 PCA
    SCON = 0x50;                //MODE1 1,8,1, 运行接收
    ES = 1;                     //启动串口中断
    TR0 = 1; //启动 Timer0
    TR1 = 1; //启动 Timer1
    EA = 1; //打开中断
}

void main()
{
    InitSystem();
    while(1)
```

```
{
    if(RxBufIn != RxBufOut)
    {
        RxData = RxBuf[RxBufOut];
        RxBufOut++;
        if(RxBufOut >= 16)
        {
            RxBufOut = 0;
        }
        switch(CurrentStatus)
        {
            case 0:
                //获取命令头
                if(RxData == 0xA5)
                {
                    CurrentStatus++;
                }
                break;
            case 1:
                if(RxData == 0x5A)
                {
                    CurrentStatus++;
                }
                else
                {
                    CurrentStatus = 0;
                }
                break;
            case 2:
                //设置 TH0, 来改变 PWM 的频率
                TH0 = RxData;
                CurrentStatus++;
                break;
            case 3:
                //设置 CCAP0H,来改变 PWM 的占空比
                CCAP0H = RxData;
                CurrentStatus = 0;
                break;
            default:
                CurrentStatus = 0;
                break;
        }
    }
}
```

```
    }  
}  
  
/*串口中断处理程序*/  
void SerISR(void) interrupt 4 using 2  
{  
    if (TI)  
    {  
        TI=0;  
    }  
    else  
    {  
        //接收到一个 Byte,把它存到 RxBuf 中去  
        RxBuf[RxBufIn] = SBUF;  
        RxBufIn++;  
        if(RxBufIn>=16)  
        {  
            RxBufIn = 0;  
        }  
        RI = 0;  
    }  
}
```

12 SPI 接口

MPC82x5x 提供了一组 SPI 接口。可分为主/从模式。在 $F_{osc}=12\text{MHz}$ 的情况下，最高速度可以达到 3Mbit/s。SPI 接口包括三个引脚：

SPICLK(P1.7) : SPI 时钟线

MISO(P1.6): 数据线(主模式输入，从模式输出)

MOSI(P1.5): 数据线(主模式输出，从模式输入)

另有一引脚 SS(P1.4)来控制 SPI 是主模式还是从模式，但可以由 SSIG(SPICTL.7)来忽略该引脚。

12.1 相关特殊寄存器

■ SPICTL(0x85): SPI 控制寄存器

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit0
SSIG	SPEN	DORD	MSTR	CPOL	CPHA	SPR1	SPR0

SSIG	0: SS(P1.4)引脚用来设定主模式还是从模式 1: 忽略 SS(P1.4)
SPEN	0: 禁止 SPI, 所有 SPI 引脚作为普通 I/O 口 1: 使能 SPI
DORD	0: 数据是高位在前 1: 数据是低位在前
MSTR	0: 设定为从模式 1: 设定为主模式
CPOL	0: 静态下 SPICLK 是低电平,首变化沿是上升沿, 次变化沿是下降沿 1: 静态下 SPICLK 是高电平,首变化沿是下降沿, 次变化沿是上升沿
CPHA	0: 数据在 SS(PIN1.4)为低并且是次变化沿发出, 在首变化沿接收 (仅在 SSIG=0 有效) 1: 数据在首变化沿发出, 在次变化沿接收
SPR1,SPR0	SPI 时钟选择 0,0 : $F_{osc}/4$ 0,1: $F_{osc}/16$ 1,0: $F_{osc}/64$ 1,1: $F_{osc}/128$

■ SPIDAT(0x86): SPI 数据寄存器

■ SPISTAT(0x84): SPI 状态寄存器

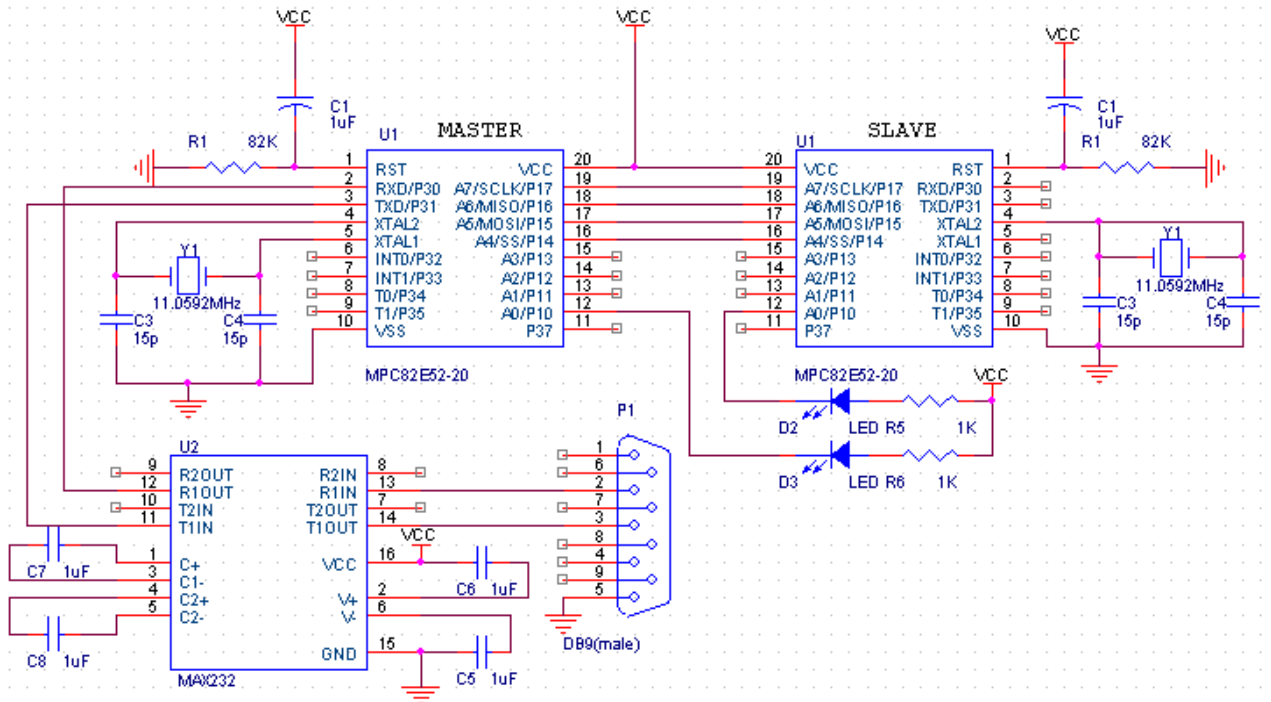
Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit0
SPIF	WCOL						

SPIF	SPI 传输完成标志 当传输完成, 这位将会被置 1, 此时如果 ESPI(IE.5)和 EA(IE.7)都为 1 则将会产生中断。要清除这个标志, 只能在软件上“写 1 到这位”
WCOL	SPI 写冲突标志 当数据在传送的时候写 SPIDAT , 该位将会被置位。要清除这个标志, 只能在软件上“写 1 到这位”

12.2 SPI 示例(C 语言)

说明: 由串口发送 16 个数据到 Master, 然后 Master 通过 SPI 发给 Slaver, Slaver 收到后取反再通过 SPI 发给 Master, 最后 Master 发到串口

电路如下:



源代码如下:

SPI_MASTER

```
/*
 *   SPI_MASTER.C
 */
#include <Intrins.h>
#include "REG_MPC82L52.H"
```

```
typedef unsigned char    uchar;
typedef unsigned short   ushort;
typedef unsigned long    ulong;
```

```
#define OscFreq      11059200L //系统时钟
#define TClock       12
#define T0OVER       1000
#define BuadRate     9600L     //串口波特率
```

```
//Timer0 用作定时器
#define TIMER0_TH0    (uchar)((65536-(OscFreq/TClock)/T0OVER)/256)
#define TIMER0_TL0    (uchar)(((65536-(OscFreq/TClock)/T0OVER)%256)
```

```
//Timer1 用作波特率,
#define TIMER1_TH1      (uchar)(256-(OscFreq/(BuadRate*32*12)))

ushort CurrentMillSceond;

uchar RxBuf[16];
uchar RxBufIn;
uchar SpiBuf[16];
uchar SpiBufIn;

bit   bSerRec;
bit   bSpiRec;
//函数定义
void SendByte(uchar ToSend);
void InitSystem();

void InitSystem()
{
    //T0 16 位定时器模式
    //T1 8 位, 自动重载 模式
    TMOD = 0x21;

    TH0    =TIMER0_TH0;           //1ms overflow
    TL0=TIMER0_TL0;

    TH1 = TIMER1_TH1;

    /*以下为串口配置*/
    SCON = 0x50;                 //MODE1 1,8,1, 运行接收

    RI = 0;
    TI = 0;

    //打开 T0,SPI,串口中断
    IE = 0x32;

    TR0 = 1;//启动 Timer0
    TR1 = 1;//启动 Timer1

    EA = 1; //打开中断
}
```

```
/*SPI 中断处理程序*/
void SpiISR(void) interrupt 5
{
    SpiBuf[SpiBufIn] = SPDAT;
    SPSTAT = 0x80;
    SpiBufIn++;
    if(SpiBufIn>=16)
    {
        SpiBufIn = 0;
        bSpiRec = 1;
        AUXR = 0;
    }
}

/*Timer0 中断处理程序*/
void time0ISR(void) interrupt 1
{
    TF0=0;
    TR0=0;
    TH0=TIMER0_TH0;
    TL0=TIMER0_TL0;
    TR0=1;
    CurrentMillSceond++;
}

/*串口中断处理程序*/
void SerISR(void) interrupt 4
{
    if (TI)
    {
        TI=0;
    }
    else
    {
        //接收到一个 Byte,把它存到 RxBuf 中去
        RxBuf[RxBufIn] = SBUF;
        RxBufIn++;
        if(RxBufIn>=16)
        {
            RxBufIn = 0;
            bSerRec = 1;
            REN = 0;
        }
        RI = 0;
    }
}
```

```
/*串口发送一个 Byte*/
void SendByte(uchar ToSend)
{
    EA = 0;
    SBUF = ToSend;
    while(TI == 0)
    {
    }
    TI = 0;
    EA = 1;
}

void main()
{
    uchar i,j;
    InitSystem();
    while(1)
    {

        if(CurrentMillSceond >= 200)
        {
            CurrentMillSceond = 0;
            P10=!P10;
        }
        if(bSerRec)
        {
            //从串口收到 16 个数,将这 16 个数发给 Slave
            SPCTL = 0xd1;           //设置成 MASTER 模式
            AUXR = 0x00;           //禁止 SPI 中断
            for(i=0;i<16;i++)
            {
                P14=0;             //控制 Slaver 为从模式
                SPDAT = RxBuf[i];
                while(SPSTAT != 0x80){}
                SPSTAT = 0x80;
                P14=1;
                for (j=0;j<5;j++){ //delay
                }
                //发送完毕
                AUXR = 0x08;       //使能 SPI 中断
                SPCTL = 0x41;      //转换到 SLAVE 模式
                bSerRec = 0;
                RxBufIn = 0;
            }
        }
    }
}
```

```
if(bSpiRec)
{
    AUXR = 0x00;           //禁止 SPI 中断
    for(i=0;i<16;i++)
    {
        SendByte(SpiBuf[i]);
    }
    //发送完毕
    bSpiRec = 0;
    SpiBufIn = 0;
    //继续从串口接收数据
    REN = 1;
}
}
```

SPI_SLAVER

```
/*
 *   SPI_SLAVER.C
 */
#include <Intrins.h>
#include "REG_MPC82L52.H "

typedef unsigned char      uchar;
typedef unsigned short    ushort;
typedef unsigned long      ulong;

#define OscFreq            11059200L //系统时钟
#define TClock             12
#define T0OVER            1000

//Timer0 用作定时器
#define TIMER0_TH0         (uchar)(((65536-(OscFreq/TClock)/T0OVER)/256)
#define TIMER0_TL0         (uchar)(((65536-(OscFreq/TClock)/T0OVER)%256)

ushort CurrentMillSceond;
uchar SpiBuf[16];
uchar SpiBufIn;
bit   bSpiRec;
void InitSystem()
{
    //T0 16 位定时器模式
    //T1 8 位, 自动重载 模式
    TMOD = 0x21;

    TH0   =TIMER0_TH0;           //1ms overflow
    TL0=TIMER0_TL0;

    /*以下为串口配置*/
    SCON = 0x50;                //MODE1 1,8,1, 运行接收

    RI = 0;
    TI = 0;

    //打开 T0,SPI 中断
    IE = 0x22;

    TR0 = 1;//启动 Timer0
    EA = 1; //打开中断
```

```
}
/*SPI 中断处理程序*/
void SpiISR(void) interrupt 5
{
    SpiBuf[SpiBufIn] = SPDAT;
    SPSTAT = 0x80;
    SpiBufIn++;
    if(SpiBufIn>=16)
    {
        SpiBufIn = 0;
        bSpiRec = 1;
        AUXR = 0;
    }
}
/*Timer0 中断处理程序*/
void time0ISR(void) interrupt 1
{
    TF0=0;
    TR0=0;
    TH0=TIMER0_TH0;
    TL0=TIMER0_TL0;
    TR0=1;
    CurrentMillSceond++;
}

void main()
{
    uchar   i,k;
    InitSystem();
    SPCTL = 0x41;           //设置成 SLAVE 模式
    AUXR = 0x08;           //使能 SPI 中断
    while(1)
    {

        if(CurrentMillSceond >= 200)
        {
            CurrentMillSceond = 0;
            P10=!P10;
        }
        if(bSpiRec)
        {
            AUXR = 0x00;           //禁止 SPI 中断
            SPCTL = 0xd1;         //转换成 MASTER 模式
            for(i=0;i<16;i++)
```

```

{
    P14=0;
    SPDAT = ~SpiBuf[i];
    while(SPSTAT != 0x80){}
    SPSTAT = 0x80;
    P14=1;
    for (k=0;k<5;k++){        //delay
    }
    //发送完毕
    bSpiRec = 0;
    SpiBufIn = 0;
    SPCTL = 0x41;              //设置成 SLAVE 模式
    AUXR = 0x08;              //使能 SPI 中断
}
}
}

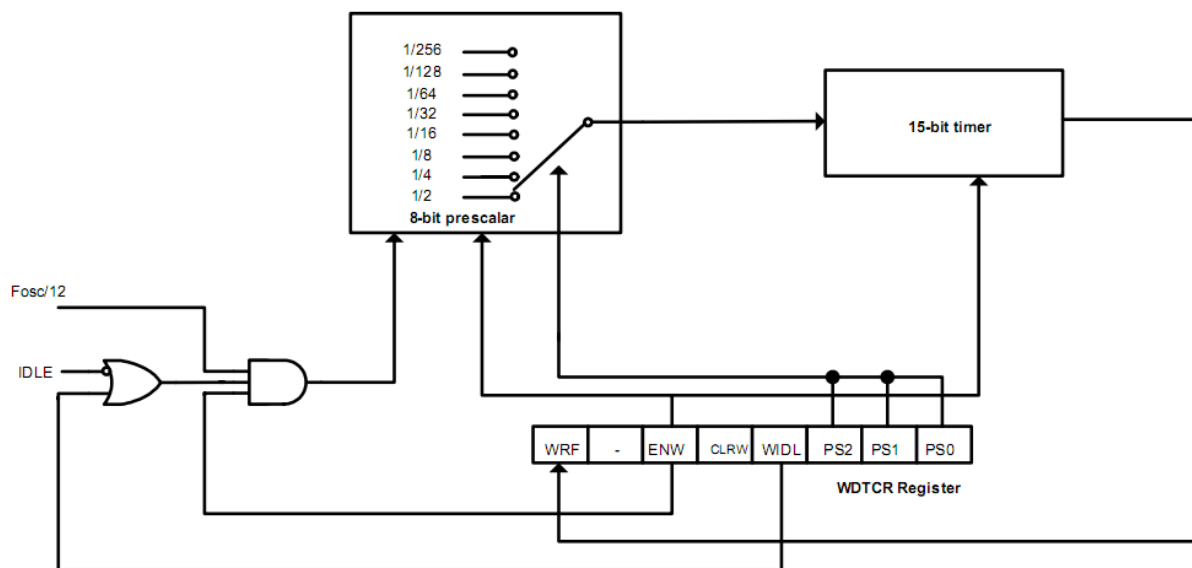
```

使用串口调试程序测试结果如图:



13 看门狗

MPC82x5x 提供了一个 15 位看门狗，8 位预分频。使能后，不能关闭。



13.1 相关特殊寄存器

■ WDTCR(0xE1): 电源控制寄存器 2

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit0
WRF		ENW	CLRW	WIDL	PS2	PS1	PS0

上电复位

HWENW

HWIDL

HWPS2

HWPS1

HWPS0

WRF: 当 WDT 溢出时，该位被置 1。由软件清零

ENW: 看门狗使能位。上电后，该位由 **HWENW** 设定

1: 使能, (注意: 使能后软件不能关闭)

0: 禁止

CLRW: 对该位置 1, 将清零 WDT 计数器

WIDL: 上电后，该位由 **HWIDL** 设定

0: 在 IDLE 模式下停止 WDT 计数

1: 在 IDLE 模式下继续 WDT 计数

PS2,PS1,PS0 设置看门狗计数器预分频。上电后，该位由 **HWPS2:0** 设定

0,0,0	2
0,0,1	4
0,1,0	8
0,1,1	16
1,0,0	32
1,0,1	64
1,1,0	128
1,1,1	256

13.2 看门狗时间计算

公式如下:

$$2^{15} \times (12 \times \text{预分频} / F_{\text{osc}})$$

下表位 6MHz 和 12MHz 在不同预分频情况下的 WDT 溢出时间

PS2,PS1,PS0	预分频值	Fosc=6MHz	Fosc=12MHz
0,0,0	2	131.072ms	
0,0,1	4	262.144ms	131.072ms
0,1,0	8	524.288ms	262.144ms
0,1,1	16	1.048s	524.288ms
1,0,0	32	2.097s	1.048s
1,0,1	64	4.194s	2.097s
1,1,0	128	8.389s	4.194s
1,1,1	256	16.778s	8.389s

13.3 看门狗示例

汇编语言

```
Start:    ;...
          ;...
          MOV    WDTCCR,#033h    ;使能 WDT,IDLE 模式下禁止, 预分频为 16
                                   ;在 6MHz 下,  WDT 溢出时间为 1.048s

MainLoop:
          MOV    WDTCCR,#13h     ;清 WDT
          ;...
          ;用户程序, 注意整个 MainLoop 最长时间不能超过 1.048s
          ;...
          JMP     MainLoop
```

C 语言

```
Void    main()
{
    WDTCCR = 0x33;    //使能 WDT,IDLE 模式下禁止, 预分频为 16
                      //在 6MHz 下,  WDT 溢出时间为 1.048s

    While(1)
    {
        WDTCCR = 0x13; //清 WDT
        //用户程序, 注意整个循环最长时间不能超过 1.048s
        //...
        //...
    }
}
```

14 电源控制

14.1 相关特殊寄存器

■ PCON2(0xC7): 电源控制寄存器 2

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit0
					CKS2	CKS1	CKS0

CKS2,CKS1,CKS0 控制在 IDLE 模式下的 Fosc

0,0,0	Fosc = OSC
0,0,1	Fosc = OSC/2
0,1,0	Fosc = OSC/4
0,1,1	Fosc = OSC/8
1,0,0	Fosc = OSC/16
1,0,1	Fosc = OSC/32
1,1,0	Fosc = OSC/64
1,1,1	Fosc = OSC/128

注: 对于 MPC82x54, 即使不在 IDLE 模式下, PCON2 同样有效。

■ PCON(0x87): 电源控制寄存器 2

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit0
SMOD		LVF	POF			PD	IDL

SMOD 0: 串口波特率不变
1: 串口波特率翻倍

LVF 低压标志位, 上电时会被置位, 应在程序初始化时清零。
当电压在(3.7V@5V, 2.3V@3V)时, 会被置位

POF 上电标志位
上电时会被置位, 由程序清零。

PD 置位后, 系统将会进入睡眠模式。

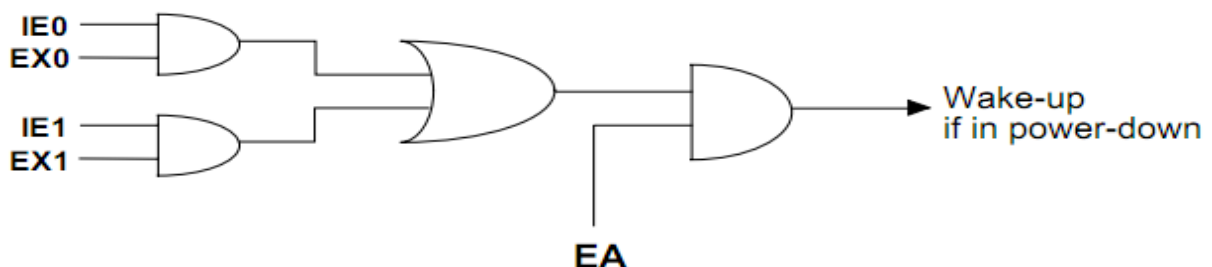
IDL 置位后, 系统将会进入 IDLE 模式

14.2 IDLE 模式

置位 IDL(PCON.0)将会进入 IDLE 模式, 在此模式下程序将停止运行, 但中断, 定时器, 串口, PCA, SPI, ADC, WDT, 等将会继续运行, 所用到的时钟 Fosc 将由 PCON2.2,1,0 来决定。任何中断和 RST 引脚复位都可以使芯片退出 IDLE 模式, 继续以正常模式运行。

14.3 睡眠模式

置位 PD(PCON.1)将会进入睡眠模式, 在此模式下 OSC 将停止, 最大限度的降低功耗。可通过外部中断(INT0, INT1)或 RST 引脚来唤醒。



■ 通过 INT0 唤醒的汇编示例

```
$include(REG_MPC82L52.INC)
```

```

CSEG    AT 0000h
JMP     Start
CSEG    AT 0003h      ;/INT0 中断向量地址
JMP     IE0_isr

IE0_isr:
  CLR    EX0
  ;... 用户程序
  RETI

Start:
  : ... 用户程序
  ;开始准备进入睡眠
  SETB   INT0          ;把 EX0/P3.2 置高
  CLR    IE0           ;清 INT0 的中断标志位
  SETB   IT0           ;选择下降沿触发
  SETB   EX0           ;打开 INT0 中断
  SETB   EA            ;打开全局中断
  ORL    PCON,#02h     ;进入睡眠

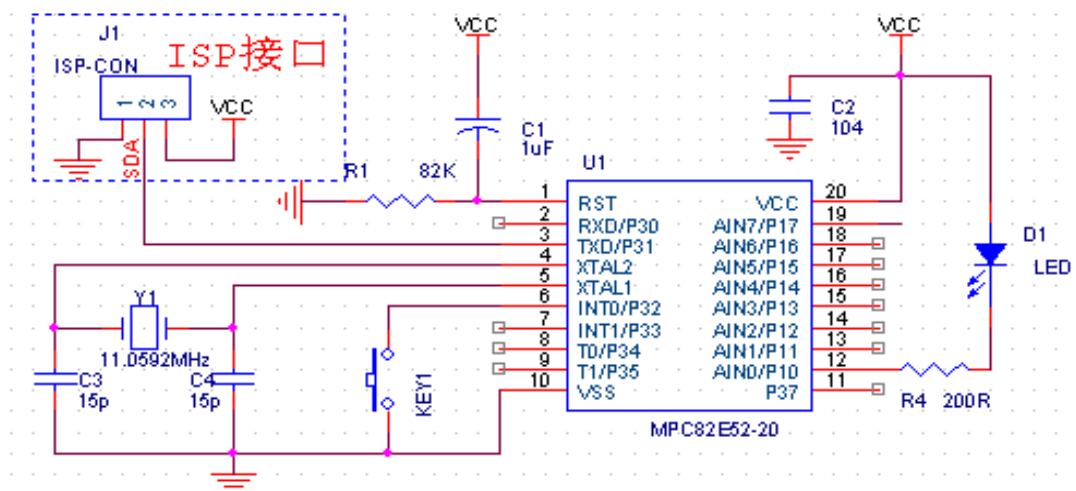
Resume_operation:
  ;如/INT0 有一下降沿到来，则芯片将会被唤醒，并进入“IE0_isr”处理
  ;处理完后将会返回到这里
  NOP                      ;注意，这里必须加一 NOP 指令
  ;...

```

14.4 睡眠和 IDLE 模式示例(C 语言)

说明: P1.0 输出脉冲。正常模式下 2 秒后进入睡眠模式，DLE(OSC/2)模式下 4 秒，IDLE(OSC/4)模式下 8 秒。按键(INT0)唤醒。每次唤醒，系统循环进入常规模式，IDLE(OSC/2)模式，IDLE(OSC/4)模式。

电路如下:



源代码如下:

```
#include <Intrins.h>
#include " REG_MPC82L52.H "
typedef unsigned char      uchar;
typedef unsigned short     ushort;
typedef unsigned long      ulong;
#define nop()              _nop_()
#define OscFreq            11059200L //系统时钟
#define T0Clock            12         //1: FOSC, 12:FOSC/12
#define T0OVER             1000L
//Timer0 用作定时器
#define TIMER0_TH0         (uchar)(((65536-(OscFreq/T0Clock)/T0OVER)/256)
#define TIMER0_TL0         (uchar)(((65536-(OscFreq/T0Clock)/T0OVER)%256)
//变量定义
ushort  CurrentMillSecond;
uchar   CurrentSecond;
ushort  LedTime;
uchar   PowerDownTime;
uchar   SysMode;
//函数声明
void InitSystem();
void PowerDown();
/*Int0 中断处理程序*/
void Int0ISR(void) interrupt 0
{
    IE0 = 0;
}
/*Timer0 中断处理程序*/
void time0ISR(void) interrupt 1
{
    TF0=0;
    TR0=0;
    TH0=TIMER0_TH0;
    TL0=TIMER0_TL0;
    TR0=1;
    CurrentMillSecond++;
    if(CurrentMillSecond >= 1000)
    {
        CurrentMillSecond = 0;
        CurrentSecond ++;
        if(CurrentSecond >= 60)
        {
            CurrentSecond = 0;
        }
    }
}
```

```
    }  
}  
void InitSystem()  
{  
    //T0 16 位定时器模式 T1 8 位, 自动重载 模式  
    TMOD = 0x21;  
    TH0=TIMER0_TH0;           //1ms overflow  
    TL0=TIMER0_TL0;  
    /*以下为中断配置打开 T0,其它都关闭*/  
    IE = 0x02;  
    TR0 = 1;//启动 Timer0  
    EA = 1; //打开中断  
}  
uchar IntValSecond(ushort Second)  
{  
    if(Second > CurrentSecond)  
        return ((60-Second)+CurrentMillSecond);  
    else  
        return (CurrentSecond-Second);  
}  
void main()  
{  
    InitSystem();  
    LedTime = 0;  
    PowerDownTime = CurrentSecond;  
    SysMode = 0;  
    while(1)  
    {  
        if(SysMode != 0)  
        {  
            //因为每次中断都会退出 IDLE 模式, 所以要重置 IDLE 模式  
            PCON2 = SysMode;  
            PCON = 0x01;  
            //进入 IDLE 模式, 直到 Timer0 产生中断后, 程序才继续运行  
            //PCON2 = 1 时,  Timer0  2ms 产生一次中断  
            //PCON2 = 2 时,  Timer0  4ms 产生一次中断  
        }  
        LedTime ++;  
        if(LedTime >= 20)  
        {  
            LedTime = 0;  
            P10 = !P10;  
        }  
        if(IntValSecond(PowerDownTime)>=2)  
        {
```

```
        PowerDownTime = CurrentSecond;
        P10 = 1;
        PowerDown();
        SysMode ++;
        if(SysMode > 2)
        {
            SysMode = 0;
        }
    }
}

void PowerDown()
{
    EA = 0;           //关闭全局中断
    INT0 = 1;         //把 EX0/P3.2 置高
    IE0 = 0;          //清 INT0 的中断标志位
    IT0 = 1;          //选择下降沿触发
    EX0 = 1;          //打开 INT0 中断
    EA = 1;           //打开全局中断
    PCON = 0x02;      //进入睡眠
    //如/INT0 有一下降沿到来，则芯片将会被唤醒，并进入"Int0ISR"处理
    //处理完后将会返回到这里
    nop();
    nop();
    EX0 = 0;
}
```

15 程序启动入口

MPC82x5x 启动依照下列规则:

If **HWBS** 在烧录时被勾选, 并且 **ISP 空间**不是 **NONE**

系统从 **ISP 空间**启动 **ISP 程序**

Else

系统从 **AP 空间**启动一般应用程序

以上规则仅在上电复位时有效, 其它复位无效。

从 **ISP 程序**切换到 **AP 应用程序**

一旦 **ISP 程序**完成 **FLASH** 的数据更新, 芯片就允许用户运行他的 **AP 应用程序**, 只有在 **ISP 程序**的末尾加一条指令, 如下:

ISPCR ← 001xxxxx

禁止写 **FLASH**, 设置 **SWBS=0**, 并且触发软件复位。然后系统就会复位(不是上电复位), 并且系统会检测 **SWBS=0**, 从而在 **AP 应用程序**入口启动。对于上电过程, **HWBS** 将会决定程序的入口, 但是软件复位的入口, 是由 **SWBS** 决定的。

从 **AP 应用程序**切换到 **ISP 程序**

芯片允许用户应用程序切换到 **ISP 程序**, 只有在应用程序中加一条指令, 如下:

ISPCR ← x11xxxxx

设置 **SWBS=1**, 从而选择软件复位的入口是 **ISP 程序**, 并且触发软件复位。之后, 系统就会复位(不是上电复位), 并且系统会检测 **SWBS=1**, 从而在 **ISP 程序**入口启动。

16 程序烧录

MPC82x5x 在出厂时已经固化有 ISP code。

OR 选项出厂默认为

	MPC82x52	MPC82x54
ISP 空间:	1K: 1C00~1FFF	1.5K: 3800~3DFF
IAP 空间:	1K: 1800~1BFF	1K: 3400~37FF
AP 空间:	6K: 0000~17FF	13K: 0000~33FF

LOCK,SB,HWBS 使能。

如果不用更改 OR 选项, 则推荐用 Megawin 8051 ISP Programmer 来烧录程序, 否则请用 Megawin 8051 Writer U1 来烧录程序。

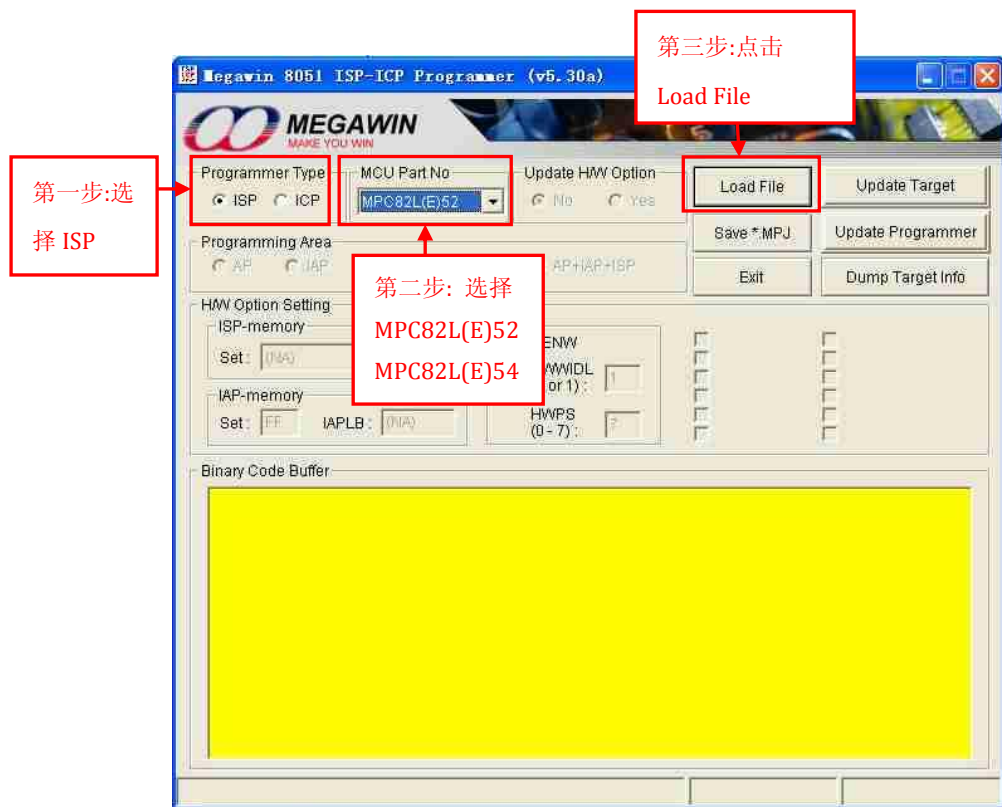
16.1 使用 ISP PROGRAMMER 烧录程序

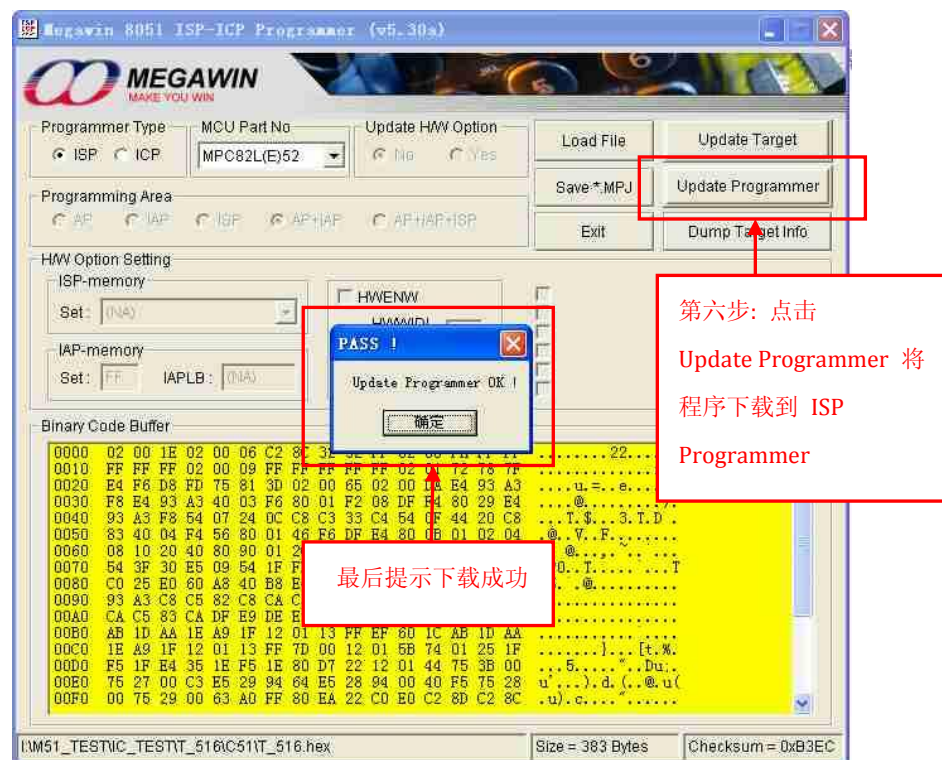
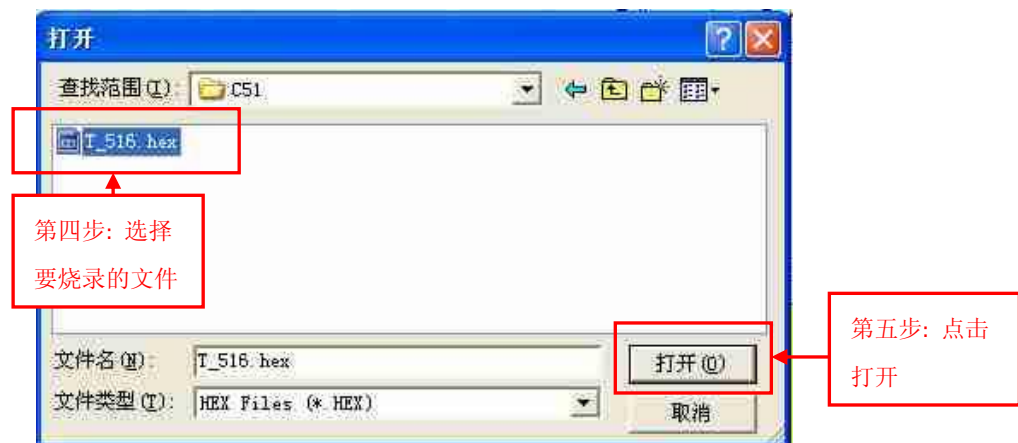
注: 在使用 ISP Programmer 烧录程序前, 要确定 MPC82x5x 里已经有 ISP code. 芯片在出厂前都会固化有 ISP code. 如果没有 ISP code 则请先用 8051 Writer U1 烧录 ISP code 到 MPC82x5x。

预备

1. 装有 Megawin ISP-ICP Programmer 软件的 PC
2. Megawin ISP-ICP Programmer 设备
3. 带 MPC82x5x 的用户目标板系统

从 PC 下载程序到 ISP PROGRAMMER

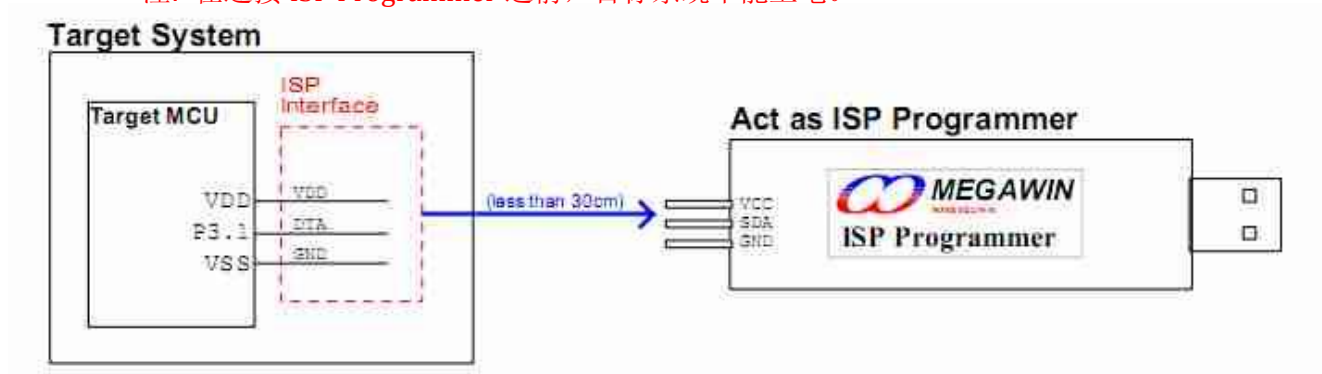




从 ISP PROGRAMMER 烧录到 MPC82x5x

- 将目标系统如下图连上 ISP Programmer.

注: 在连接 ISP Programmer 之前, 目标系统不能上电。



- 连接好后, 对目标板上电。此时如果 ISP Programmer 绿灯亮, 则表示连接成功, 可以进行下一步了。否则表示连接不成功。
- 按 ISP Programmer 的 “Program”按钮, 开始烧录。红灯闪烁表示正在烧录。闪烁停止后, 如果绿灯亮, 则表示烧录成功, 否则烧录失败。
- 烧录成功后, 断掉目标板的电源, 断开与 ISP Programmer 的连接。然后对目标板重新上电, 此时 MPC82x5x 运行的就是刚刚烧录的程序了。

16.2 使用 8051 WRITER U1 烧录程序

预备

1. 装有 Megawin 8051 Writer U1 软件的 PC
2. Megawin 8051 Writer U1 设备
3. 对于不同封装，应有不同转接座，如下：

PDIP-20: MegaWin DIP20 to DIP40

PDIP-28: MegaWin DIP20 to DIP40

PLCC-32: MegaWin PLCC-32 to DIP40

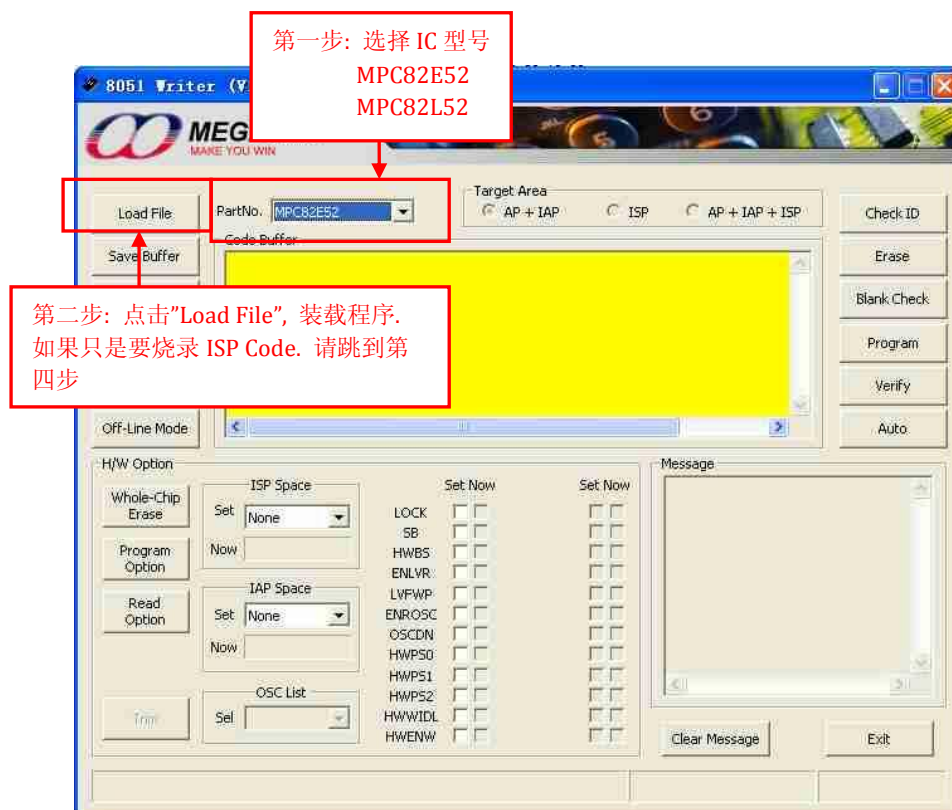
SOP-20: 标准 SOP-20 to DIP20, MegaWin DIP20 to DIP40

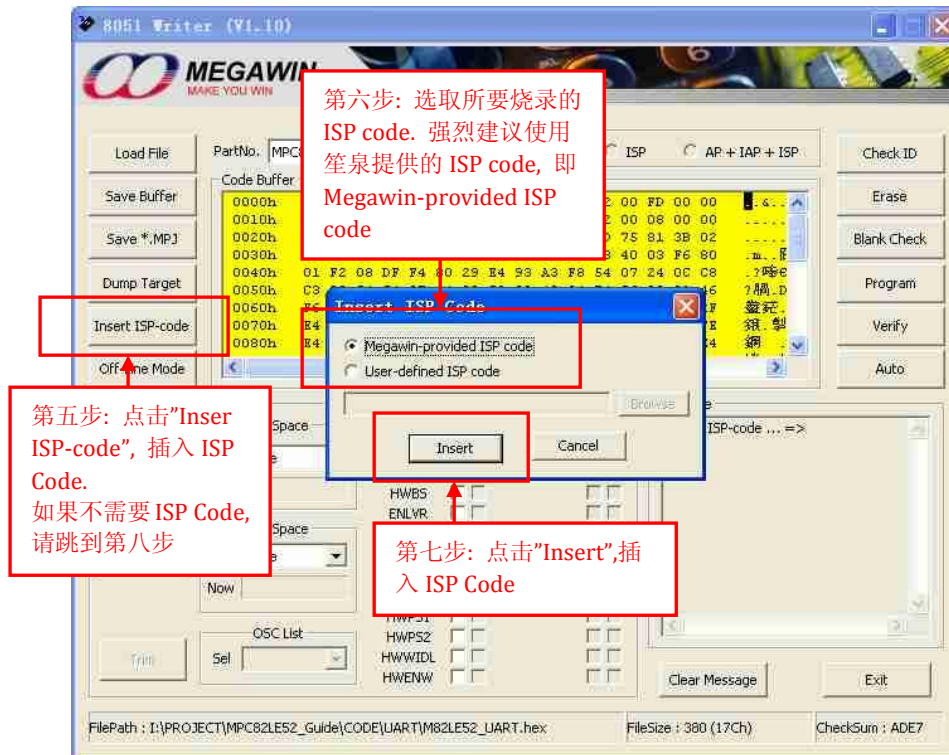
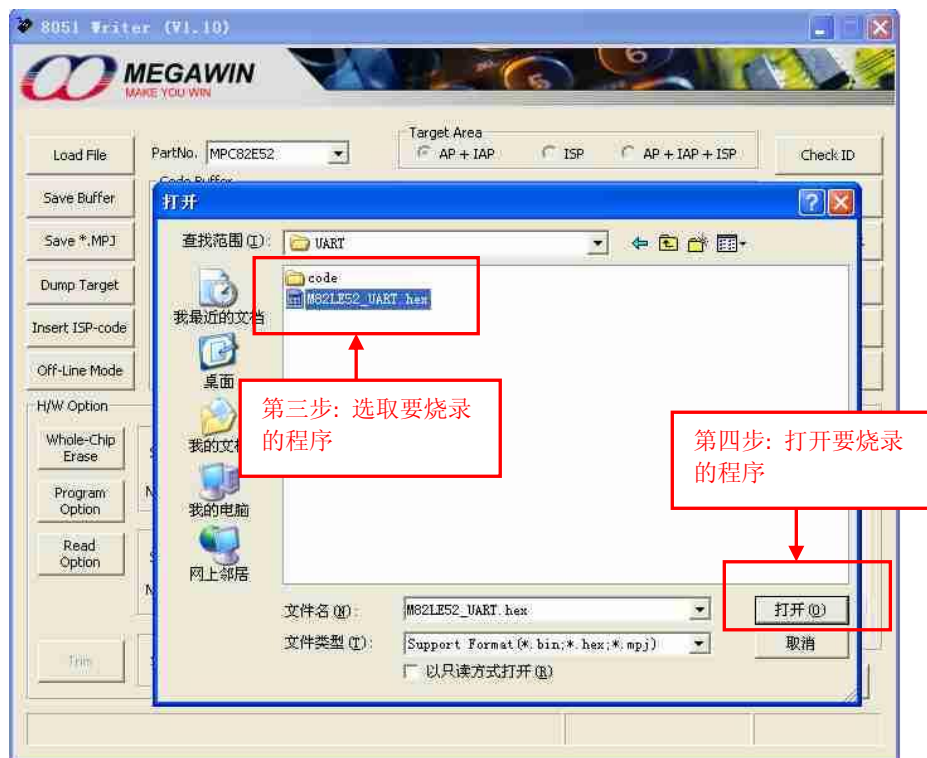
SOP-28: 标准 SOP-28 to DIP28, MegaWin DIP28 to DIP40

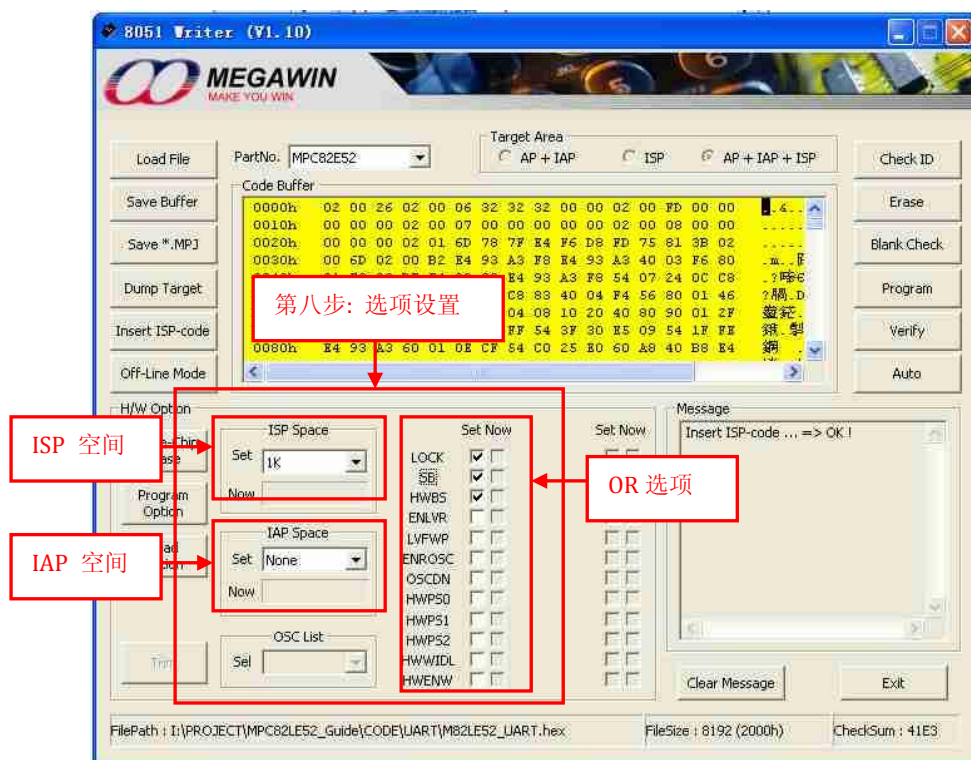
TSSOP-20: 标准 TSSOP -20 to DIP20, MegaWin DIP20 to DIP40

TSSOP-28: 标准 TSSOP -28 to DIP28, MegaWin DIP28 to DIP40

从 PC 下载程序到 MPC82x5x







ISP 空间:

如果使用的是笙泉提供的 ISP code, 则会自动设置为 1K。如果是用户自己的 ISP code, 则根据 ISP code 的大小来设置不同的值, 可设为 1K,2K,3K(MPC82x52)或 1.5K,2.5K,3.5K(MPC82x54)。如果没有使用 ISP,则请设为 None。

IAP 空间:

根据用户的需要来设置。如没有使用请设为 None。

OR 选项:

如果使用的是笙泉提供的 ISP code, 则 HWBS 会自动勾选。

LOCK,SB: 为了程序的保密, 建议勾选。

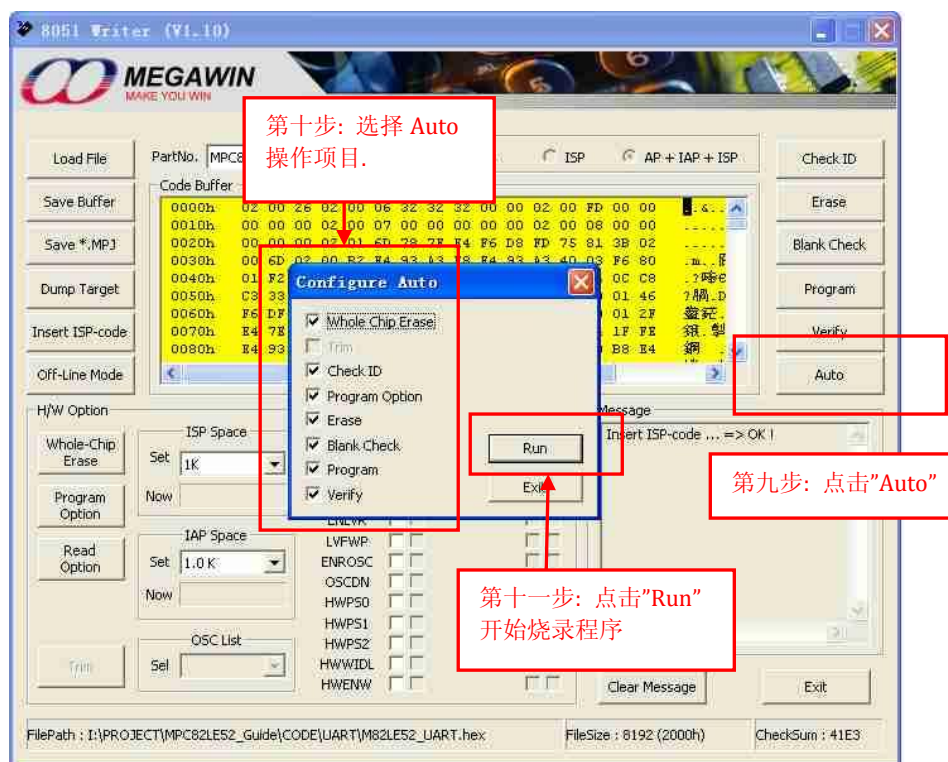
ENLVR: 如需低压时复位, 请勾选。

ENROSC: 使用内部 RC 振荡时, 请勾选。

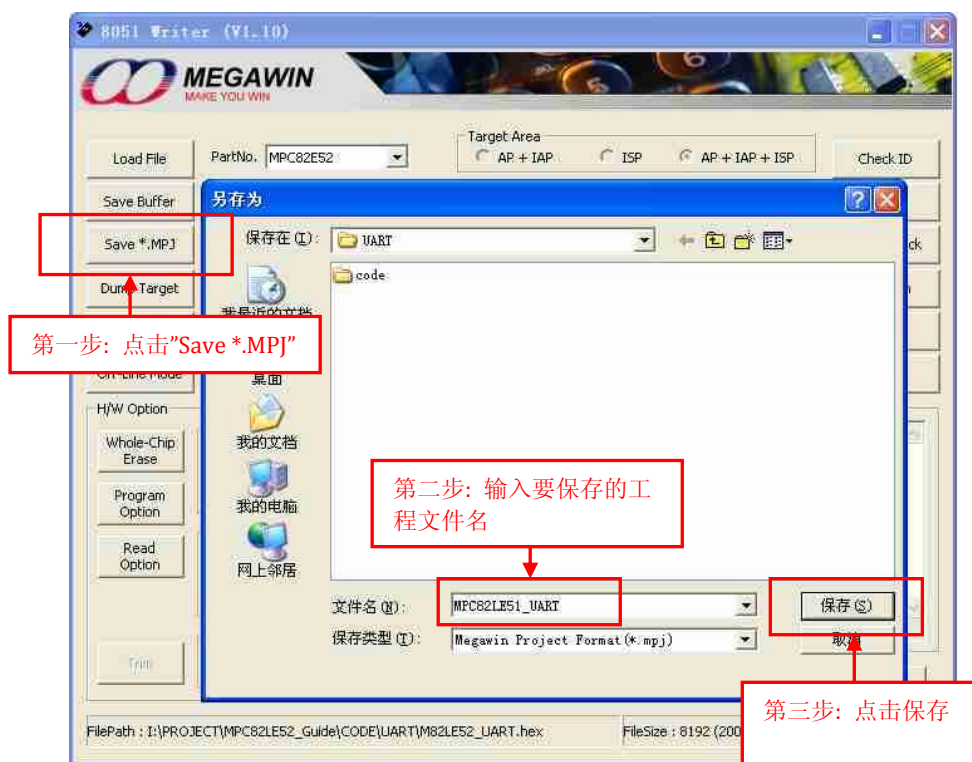
OSCDN: 勾选时, OSC 为 1/2 gain. 有助于 EMI。

HWPS0,HWPS1,HWPS2,HWWIDL,HWENW:

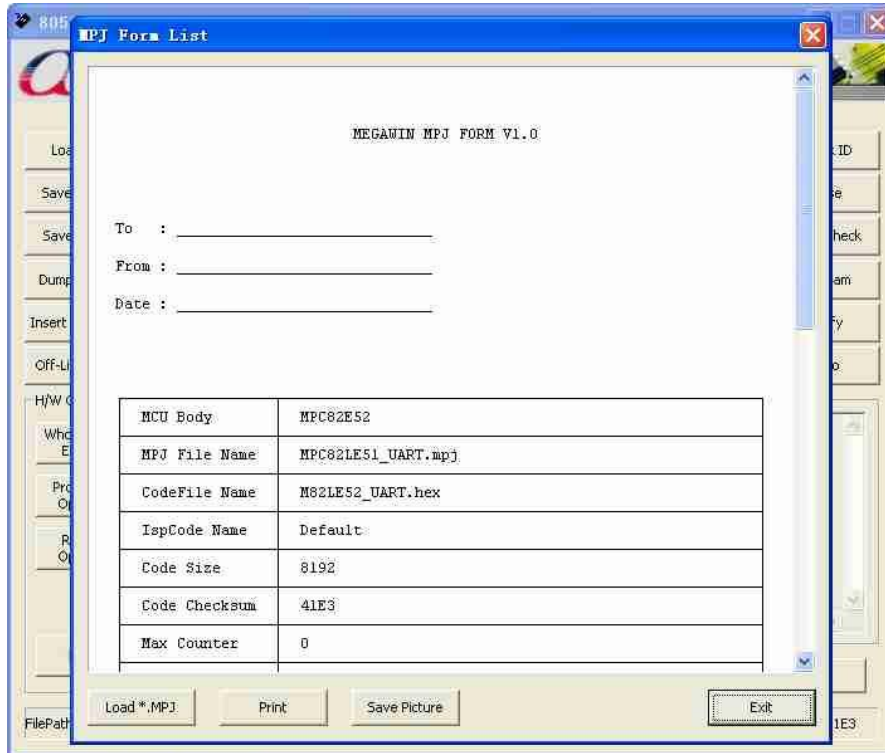
看门狗选项。HWENW 被勾选后, 看门狗上电即打开, 无需软件打开。



烧录完后，可以保存为工程文件，方便下次烧录或者量产时烧录。



点击保存后，会弹出工程文件的相关信息，在这里可以打印出来，或保存成图片。以后调出此工程文件来烧录前，就可以按打印出来的文件或保存的图片来检查此工程文件是否正确。



The image shows a software window titled "MPJ Form List" with a blue title bar. Inside the window, the text "MEGAWIN MPJ FORM V1.0" is displayed at the top. Below this, there are three input fields labeled "To :", "From :", and "Date :". At the bottom of the window, there are four buttons: "Load *.MPJ", "Print", "Save Picture", and "Exit".

MCU Body	MPC82E52
MPJ File Name	MPC82LE51_UART.mpj
CodeFile Name	M82LE52_UART.hex
IspCode Name	Default
Code Size	8192
Code Checksum	41E3
Max Counter	0

17 附录

17.1 指令集

Rn	暂存器 R0~R7
direct	8 位内部存储器，包括 - 内部存储器(00~7F)的地址 - 特殊功能寄存器(80~FF)的地址，如 P0,PSW,TMOD,...等
@Ri	由寄存器 R0 或 R1 所索引的内部 RAM 数据
#data	8 位常数
#data16	16 位常数
Addr16	16 位的目的地地址，可使跳转指令跳转 64K
Addr11	11 位的目的地地址，可使跳转指令跳转 2K
rel	有正负号的 8 位地址偏移量，用于相对地址的跳转
bit	1 个 bit: 指所有可以位寻址的的位元
A	累加器 Acc
C 或 CY	进位标志
AC	辅助进位标志
Bb	指定位元 B0~B7
D	半位元组(4bit)
F0	旗号 0
I	中断
PC	程序计数器
SP	堆栈
B	寄存器 B
DPTR	程序数据地址寄存器
@	间接寻址符号
\$	程序计数器当前的值
reg	寄存器

数据传送			
助记符	描述	字节数	指令周期
MOV A, Rn	$Acc \leftarrow Rn$	1	1
MOV A, direct	$Acc \leftarrow direct$	2	2
MOV A, @Ri	$Acc \leftarrow Ri$	1	2
MOV A, #data	$Acc \leftarrow data$	2	2
MOV Rn,A	$Rn \leftarrow Acc$	1	2
MOV Rn,direct	$Rn \leftarrow direct$	2	4
MOV Rn,#data	$Rn \leftarrow data$	2	2
MOV direct,A	$direct \leftarrow Acc$	2	3
MOV direct,Rn	$direct \leftarrow Rn$	2	3
MOV direct,direct	$direct \leftarrow direct$	3	4
MOV direct,@Ri	$direct \leftarrow Ri$	2	4
MOV direct,#data	$direct \leftarrow data$	3	3

MOV @Ri,A	$Ri \leftarrow Acc$	1	3
MOV @Ri,direct	$Ri \leftarrow direct$	2	3
MOV @Ri,#data	$Ri \leftarrow data$	2	3
MOV DPTR,#data16	$DPTR \leftarrow 16bit\ data$	3	3
MOVC A,@A+DPTR	$Acc \leftarrow (A+DPTR) \text{ 地址所指的数据}$	1	4
MOVC A,@A+PC	$Acc \leftarrow (A+PC) \text{ 地址所指的数据}$	1	4
PUSH direct	堆栈 $\leftarrow direct$	2	4
POP direct	$direct \leftarrow \text{堆栈}$	2	3
XCH A,Rn	A 和 Rn 互换	1	3
XCH A,direct	A 和 direct 互换	2	4
XCH A,@Ri	A 和 Ri 互换	1	4
XCHD A,@Ri	A 和 Ri 的低四互换	1	4
算术运算			
ADD A,Rn	$Acc \leftarrow Acc+Rn$	1	2
ADD A,direct	$Acc \leftarrow Acc+direct$	2	3
ADD A,@Ri	$Acc \leftarrow Acc+Ri$	1	3
ADD A,#data	$Acc \leftarrow Acc+data$	2	2
ADDC A,Rn	$Acc \leftarrow Acc+Rn+C$	1	2
ADDC A,direct	$Acc \leftarrow Acc+direct+C$	2	3
ADDC A,@Ri	$Acc \leftarrow Acc+Ri+C$	1	3
ADDC A,#data	$Acc \leftarrow Acc+data+C$	2	2
SUBB A,Rn	$Acc \leftarrow Acc-Rn-C$	1	2
SUBB A,direct	$Acc \leftarrow Acc-direct-C$	2	3
SUBB A,@Ri	$Acc \leftarrow Acc-Ri-C$	1	3
SUBB A,#data	$Acc \leftarrow Acc-data-C$	2	2
INC A	$Acc \leftarrow Acc+1$	1	2
INC Rn	$Rn \leftarrow Rn+1$	1	3
INC direct	$direct \leftarrow direct+1$	2	4
INC @Ri	$Ri \leftarrow Ri+1$	1	4
INC DPTR	$DPTR \leftarrow DPTR+1$	1	1
DEC A	$Acc \leftarrow Acc-1$	1	2
DEC Rn	$Rn \leftarrow Rn-1$	1	3
DEC direct	$direct \leftarrow direct-1$	2	4
DEC @Ri	$Ri \leftarrow Ri-1$	1	4
MUL AB	两数相乘, 结果高八位存入 B, 低八位存入 A	1	4
DIV AB	Acc 除以 B, 商存入 Acc, 余数存入 B	1	5
DA A	Acc 作十进制调整	1	4
逻辑运算			
ANL A,Rn	$Acc \leftarrow Acc\ and\ Rn$	1	2
ANL A,direct	$Acc \leftarrow Acc\ and\ direct$	2	3
ANL A,@Ri	$Acc \leftarrow Acc\ and\ Ri$	1	3
ANL A,#data	$Acc \leftarrow Acc\ and\ data$	2	2

ANL direct,A	Direct $\leftarrow$ direct and Acc	2	4
ANL direct,#data	Direct $\leftarrow$ direct and data	3	4
ORL A,Rn	Acc $\leftarrow$ Acc or Rn	1	2
ORL A,direct	Acc $\leftarrow$ Acc or direct	2	3
ORL A,@Ri	Acc $\leftarrow$ Acc or Ri	1	3
ORL A,#data	Acc $\leftarrow$ Acc or data	2	2
ORL direct,A	Direct $\leftarrow$ direct or Acc	2	4
ORL direct,#data	Direct $\leftarrow$ direct or data	3	4
XRL A,Rn	Acc $\leftarrow$ Acc xor Rn	1	2
XRL A,direct	Acc $\leftarrow$ Acc xor direct	2	3
XRL A,@Ri	Acc $\leftarrow$ Acc xor Ri	1	3
XRL A,#data	Acc $\leftarrow$ Acc xor data	2	2
XRL direct,A	Direct $\leftarrow$ direct xor Acc	2	4
XRL direct,#data	Direct $\leftarrow$ direct xor data	3	4
CLR A	清除累加器 Acc	1	1
CPL A	累加器反相	1	2
RL A	累加器向左旋转	1	1
RLC A	累加器和 C 左旋	1	1
RR A	累加器向右旋转	1	1
RRC A	累加器和 C 右旋	1	1
SWAP A	累加器的高低四位互换	1	1
位逻辑运算			
CLR C	清除进位标记	1	1
CLR bit	清除直接位元	2	4
SETB C	设定进位标记	1	1
SETB bit	设定直接位元	2	4
CPL C	进位标记取反	1	1
CPL bit	直接位元取反	2	4
ANL C,bit	C $\leftarrow$ C and bit	2	3
ANL C,/bit	C $\leftarrow$ C and bit(反相)	2	3
ORL C,bit	C $\leftarrow$ C or bit	2	3
ORL C,/bit	C $\leftarrow$ C or bit(反相)	2	3
MOV C,bit	C $\leftarrow$ bit	2	3
MOV bit,C	bit $\leftarrow$ bit	2	4
位逻辑跳转			
JC rel	如果 C=1 跳转到 rel	2	3
JNC rel	如果 C=0 跳转到 rel	2	3
JB bit,rel	如果 bit=1 跳转到 rel	3	4
JNB bit,rel	如果 bit=0 跳转到 rel	3	4
JBC bit,rel	如果 bit=1 跳转到 rel,并且清除 bit	3	5
程序跳转			
ACALL addr11	绝对式子程序调用	2	6

LCALL addr16	远程子程序调用	3	6
RET	从子程序返回	1	4
RETI	从中断返回	1	4
AJMP addr11	绝对式跳转	2	3
LJMP addr16	远程跳转	3	4
SJMP rel	短程跳转	2	3
JMP @A+DPTR	间接跳转	1	3
JZ rel	如果 Acc=0 则跳到 rel	2	3
JNZ rel	如果 Acc≠0 则跳到 rel	2	3
CJNE A,direct,rel	如果 Acc≠direct 则跳到 rel	3	5
CJNE A,#data,rel	如果 Acc≠data 则跳到 rel	3	4
CJNE Rn,#data,rel	如果 Rn≠data 则跳到 rel	3	4
CJNE @Ri,#data,rel	如果 Ri≠data 则跳到 rel	3	5
DJNZ Rn,rel	如果(Rn-1)≠0 则跳到 rel	2	4
DJNZ direct,rel	如果(direct-1)≠0 则跳到 rel	3	5
NOP	无动作	1	1