

### 3 AVR 大学 零点篇 为什么选择 AVR 单片机?

本页关键词: avr 单片机学习 什么是单片机? 什么是 AVR?

为什么选用 AVR 单片机? 选自 AVR 研讨会演讲提纲!

(1) 为什么选用 AVR 单片机? Flash 程序存储器可擦写 1000 次以上,不再有报废品产生。PIC 有的是 OTP,只能烧录一次。AVR 程序存储器数据为 16 位组织,也可按 8 位理解。PIC 是 12/14 位程序 存储器,作寄存器转移和算术、逻辑运算带来不便。

(2) 为什么选用 AVR 单片机? 高速度(50ns)、低功耗!硬件应用 Harward 结构,具有预取指令功能,使得指令可以在一个时钟周期内执行。PIC 要 4 个时钟周期执行一条指令。MSC-51 要 12 个时钟周期 执行一条指令。

(3) 为什么选用 AVR 单片机? 超功能精简指令!具有 32 个通用工作寄存器(相当于 8051 中的 32 个累加器,克服了单一累加器数据处理造成的瓶颈现象),有 128B~4KB 个 SRAM,可灵活使用指令运算。

(4) 为什么选用 AVR 单片机? 工业级产品!具有大电流(灌电流)10~20mA 或 40mA(单一输出),可直接驱动 SSR 或继电器;有看门狗定时器(WDT),安全保护,防止程序走飞,提高产品的抗干扰能力。

(5) 为什么选用 AVR 单片机? 程序写入可以并行写入(用万用编程序器),也可用串行在线 ISP 擦 写。也就是说不必将 IC 拆下拿到万用编程上烧录,而可直接在电路板上进行程序修改、烧录等操作,方便产品现场升级。有 ISP、JTAG 及自编功能,这是今后单片机编程的发展方向。

(6) 为什么选用 AVR 单片机? AVR 的 I/O 口是真正的 I/O 口,能正确反映 I/O 口的真实情况。I/O 口 有输入/输出, 三态高阻输入,也可设定内部拉高电阻作输入端的功能,便于作各种应用特性所需 (多功能 I/O 口)

(7) 为什么选用 AVR 单片机? 高度保密(LOCK)!

—不可破解的 Lock bit 技术

—不象 Mask ROM 那样可通过电子显微镜破解—Flash 单元深藏于芯片内部 —可多次烧写的 Flash 且具有多重密码保护锁死(LOCK)功能,因此可快速完成产品商品化,并可多次更改程序(产品升级)而不必浪费 IC 或电路板,大大提高产品质量及竞争力。

(8) 为什么选用 AVR 单片机? AVR 内带模拟比较器,I/O 口可作 A/D 转换用,可组成廉价的 A/D 转换器。

(9) 为什么选用 AVR 单片机?可重设启动复位。AVR 系列有内部电源开关启动计数器,可将低电平复位(/RESET)直接接到 Vcc 端。当电源开时,由于利用内部 RC 的看门狗定时器,可延迟 MCU 启动执行程序。这种延时使 I/O 口稳定后执行程序,以提高单片机工作可靠性。有的还有内部复位电压检测电路 BOD,而检测电压可调。

(10)为什么选用 AVR 单片机?具有休眠省电功能(POWER DOWN)及闲置(IDLE)低功耗功能。一般耗电在 1~2.5mA,典型功耗情况,WDT 关闭时为 100nA

(11) 为什么选用 AVR 单片机? 像 8051 一样,有多个固定中断向量入口地址,可快速响应中断。而 PIC 只有一个中断入口,要查询后才能响应中断,失去了最佳响应中断时间。

(12) 为什么选用 AVR 单片机? AT90S1200/2343/ATtiny15 等部分 AVR 器件具有内部 RC 振荡器-1MHz 的工作频率,使该类单片机成为无外加元器件即可工作,就是一片芯片,可谓简单方便,作加密 器件使用更妙。

(13) 为什么选用 AVR 单片机? 计数器/定时器,C/T 有 8 位和 16 位,可作比较器;计数器外部中断和 PWM(也可当 D/A)用于控制输出,有的有 3-4 个 PWM,作电机无级调速是理想器件。

(14) 为什么选用 AVR 单片机?有串行异步通讯 UART 接口,不占用定时器和 SPI 传输功能,因其高速故 可以工作在一般标准整数频率,而波特率可达 576K。

(15) 为什么选用 AVR 单片机? AT90S4414/AT90S8515 具有可扩展外部数据存储器达 64KB。它们的引脚排列及功能与 8051 相似,即可替代 8051 系列单片机(8751 或 8752)的应用系统。仅差复位 电平,只需对调复位电阻、电容位置。还增加很多新功能,WDT,A/D,PWM 等

(16) 为什么选用 AVR 单片机? 工作电压范围宽(2.7~6.0V),电源抗干扰性强。AT90LXX 为低电压器件(2.7~6.0V), AT90SXX 电压为 (4.0~6.0V),最低器件 ATtiny12 已到 1.8V ~ 5.5V。

(17) 为什么选用 AVR 单片机? AT90S4434/8535 具有 8 路 10 位 A/D;AT90S2333/4433 具有 6 路 10 位 A/D;功能更强的 ATmega103/128 有 Flash 128KB,EEPROM 4KB,RAM 4KB,I/O 端口 48 个,中断源 16 个 ,外中断 8 个,SPI,UART,8 路 10 位 A/D,ISP。

(18) 为什么选用 AVR 单片机? 并具有较大容量、可擦写 10 万次的 EEPROM,对掉电后数据保存带来 方便,来电后能记住掉电时的工体状态,EEPROM(64B~4KB) 。

(19) 为什么选用 AVR 单片机? 新的高档 AVR ATmega16/32/64/128 还具有 JTAG 边界扫描、仿真、编程功能,不会造成以往仿真通过,脱机不行的现象。

(20) 为什么选用 AVR 单片机? AVR 微处理器---可以对自己编程..ATmega161 为使用便利性而设计

-写入新代码时无需外部器件

-小扇区：128 字节

-Boot 区可变

-Read-While-Write 技术

-减少编程时间

-受控于硬件的编程方法

-打开通向新世界的大门

-通过任何接口进行编程(并行编程器、ISP、JTAG、 UART 、自编程)-重复编程无需外部器件

-100%安全的远程加密更新方式

(21) 为什么选用 AVR 单片机? 从高级语言 C 代码,看各种单片机性能比较: 从一个小 C 函数为例:

```
/* Return the maximum value of a table of 16 integers */
```

;返回最大值的表格的 16 位整数

```
int max(int *array);数组*array
```

```
{
```

```
char a;
```

```
int maximum=-32768;最大的=-32768
```

```
for (a=0;a<16;a++)  
  
if (array[a]>maximum)  
  
maximum=array[a];  
  
return (maximum);返回  
  
}
```

性能比较:

AT90S8515 8 MHz

80C51 24 MHz

68HC11A8 12 MHz

PIC16C74 20 MHz

编译结果结论:

8 MHz AVR ——224 MHz 80C51

HC11: 代码效率高, 但是处理能力只有 AVR 的 1/10, 功耗却高 2.5 倍

PIC 速度快, 但是在相同功耗下 AVR 性能比其高 3.5 倍

(22) 为什么选用 AVR 单片机?AVR 使用众多功能强大的高级语言

- IAR AVR C 编译器 编译器与 AVR 同步设计,支持 C 和 EC++, Demo 版只生成调试文件,
- ICC AVR C 编译器 支持无 SRAM 器件;增加组软件模块;Icc Demo 版 30 天是完全版, 30 天后转 限 2KB 版
- Code Vision AVR C 编译器 有组软件模块,Demo 版为限 2KB 版
- GNU C 编译器 网友联盟自由免费版,升级慢
- BASCOM-AVR Demo 版为限 2KB 版

(23) 为什么选用 AVR 单片机?AVR 有各种档次的开发工具

评估工具

—AVR Studio

—STK500

—GNU GCC 编译器高性能开发工具

—AVR Studio

—STK500

–ICE10 / ICE30/ICE200

–IAR C

低成本开发工具

–AVR Studio

–ICE200 / JTAGICE

–Imagecraft C

- 本站开发 AVR mega16/32 学习开发板

(23)为什么选用 AVR 单片机?有了 AVR 基础,我们学习 FPSLIC(=AVR+FPGA+SRAM),使我国单片机开发 进入芯片级开发。

## 4 AVR 大学 零点篇 怎样成为单片机高手?

本页关键词: avr 单片机学习 怎样成为单片机高手

怎样成为单片机高手

不管出于什么原因学习单片机,成为单片机高手几乎可以说是每一个认真学单片机高手的愿望。

单片机高手的秘诀

1.不要看到别人的回复第一句话就说:给个代码吧!你应该想想为什么。当你自己想出来再参考别人的提示,你就知道自己和别人思路的差异。

2.别小家子气,买本书几十块都舍不得,你还学个 P。为了省钱看电子书,浪费的时间绝对 超过书的价值。当然如果查资料,只能看 PDF。

3.学习新的开发软件时,一定要看帮助手册。买的书不够全面。刚接触一个软件,什么都不 懂,就盲目的问东问西,让人看起来很幼稚。

4.不要蜻蜓点水,得过且过,细微之处往往体现实力。

5.把时髦的技术挂在嘴边，还不如把过时的技术记在心里。

6.看得懂的书，请仔细看；看不懂的书，请硬着头皮看。别指望看第一遍书就能记住和掌握什么——请看第二遍、第三遍。

7.多实践，去焊板子、调试，去写去调，只用软件模拟，是永远成不了高手的。

8.保存好你做过的所有的源程序、PCB、原理图等----那是你最好的积累之一。

9.对于网络，还是希望大家能多利用一下，很多问题不是非要到论坛来问的，首先你要学会自己找答案，比如 google、百度都是很好的搜索引擎，你只要输入关键字就能找到很多相关资料，别老是等待别人给你希望，看的出你平时一定也很懒！

10 到一个论坛，你学会去看以前的帖子，不要什么都不看就发帖子问，也许你的问题早就有人问过了，你再问，别人已经不想再重复了，做为初学者，谁也不希望自己的帖子没人回的。

11，虽然不是打击初学者，但是这句话还是要说：论坛论坛，就是大家讨论的地方，如果你总期望有高手总无偿指点你，除非他是你亲戚！！讨论者，起码是水平相当的才有讨论的说法，如果水平真差距太远了，连基本操作都需要别人给解答，谁还跟你讨论呢。

什么样的人 是浮躁的人？

浮躁的人容易问：我到底该学什么；——踏踏实实的学点基本的吧？单片机不知道是什么就想去学 ARM？ c 语言不会想搞 LINUX?别老是好高骛远。

浮躁的人容易问：谁有 xxx 源码？——（你给人家多少钱啊？自己的劳动白送你？）

浮躁的人容易说：跪求 xxx——（就算网络也要点尊严吧？）

浮躁的人容易说：紧急求救——（其实只是个简单的课程设计）

浮躁的人容易说：有没有 xxx 中文资料？——一个字：懒。别说别的。E 文不行？谁不是从 ABC 学起的啊？

浮躁的人容易说：求xxx,我的email是 [xxx@xxx.com](mailto:xxx@xxx.com)，然后消失 ---- 你以为你是大爷啊，人家请你吃饭，还要喂到你口里不成？

浮躁的人容易问：做单片机有钱途吗----只是为了钱，搞不好技术的，你去抢银行好了。

浮躁的人容易说：哪里有 xxx 芯片资料？（其实大部分资料网络上都有，但是偏偏来找人问，懒！）

浮躁的人分两种：只观望而不学的人；只学而不坚持的人；浮躁的人永远不是一个高手。

学好单片机之后

基于单片机在各行各业的广泛应用，社会对单片机越来越重视，高校也纷纷开设单片机课程。

学好单片机可以让你在电子行业内找到一个比较好的工作。

学好单片机可以让你增加对电子产品的了解，扩展产品开发的思路，提高社会竞争力。

5AVR 大学 零点篇 怎样开始学习单片机？

学习 AVR，需要软件，硬件，开发工具的支持。我们介绍一下开发环境的比较与选择。

AVR的编译软件常用的有，IAR AVR，imagecraft（ICC AVR），Code Vision，GNU GCC AVR等。

你可以在本站下载到这些软件：[http://www.avrvi.com/start/avr\\_software.html](http://www.avrvi.com/start/avr_software.html)

特点	IAR	imagecraft	Code Vision	GNU GCC
代码效率	+++	++	++	++
价格	¥¥	¥	¥	Free
易用性	++	+++	+++	+
与 AVR STUDIO 集成度	++	+++	+++	++
技术支持	+	+++	+++	—

综合易用性，价格，集成度，技术支持多方面因素，本站推荐 imagecraft (ICC AVR)，本站的教程，代码例子也是基于本平台下编写的。如果是个人使用，可以在 google 上搜索 ICC 的注册机，用于专业版的测试。请你支持正版，支持软件业的发展。

AVR 的下载软件常用的有：AVR STUDIO，ICC AVR，SL AVR

软件	AVR STUDIO	ICC AVR	SL AVR
简介	Atmel 官方开发工具软件安，支持所有的芯片系列，还可以用于软件仿真。通过*.cof 文件与外部编译软件集成。	ICC AVR内部支持并口ISP下载，STK500 下载，详细配置及使用方法，请看 <a href="#">使用ICC和并口ISP下载线下载程序</a>	双龙公司开发
支持	串口，USB，JTAG 在线仿真，ISP 下载	支持串口，并口，ISP 下载。不支持仿真	支持并口 ISP 下载

本站推荐AVR STUDIO，易用性好，可靠性高，支持所有芯片，官方免费提供，关于详细的使用方法,新手入门中有介绍。同样这些软件你可以在本站下载得到：[http://www.avrvi.com/start/avr\\_software.html](http://www.avrvi.com/start/avr_software.html)

AVR的开发工具有，原装的MKii，原装MKisp，普通版JTAG，并口ISP，[豪华版 JTAG 与 ISP 二合一 编程仿真器](#)。

原装的开发工具功能和稳定性有保障，但是价格比较昂贵（原装 JTAG mkII 仿真下载器价格为 2380 元），如果不是特别需要，没有必要做那么大的投入。

普通版 JTAG 仿真器价格便宜，一般在 100~200 元之间，本站提供的普通版 JTAG 仿真器仅售 120 元。但是普通版 JTAG 仿真器存在抗干扰能力差的问题，在一些特殊的电路下（比如带有继电器的电路），回灌电流可能将仿真器主控芯片内的程序冲掉。

并口 ISP 是最经济的开发方案，本站提供的并口 ISP 下载线只需 30 元，但是并口 ISP 下载速度慢，如果你有一定电路基础，你可以自制并口 ISP 下载线，制作方法本站后面的教程将会提供。

本站推荐折衷的解决方案，豪华版 JTAG 与 ISP 二合一 编程仿真器，稳定性好，功能强，价格低，是目前性价比非常高的开发工具方案。 [使用说明书](#)，详细的介绍了JTAG仿真的方法，ISP下载的使用



本编程仿真器特有功能：

- 1、JTAG 与 ISP 完美二合一
- 2、自动识别目标板是 JTAG 还是 ISP
- 3、使用时无需手动重启，目标板插拔时自动重启
- 4、各种保护电路，保护编程器和目标板
- 5、支持 USB 和串口双接口
- 6、超低价格：每套仅需 350 元

ISP 功能：

- 1、ISP 下载功能
- 2、使用 AVR Studio 下载时选择 STK500/ISP 即可
- 3、支持几乎所有的带 ISP 功能的芯片

JTAG 功能：

- 1、完全支持 AVR Studio 4.XX。
- 2、支持所有具有 JTAG 接口的 AVR 系列单片机。
- 3、完全真实的实现 AVR 单片机的所有的电性能。
- 4、可以完全实现片内的数字或模拟功能。
- 5、可以在程序执行过程中实现单步（step）、连续、断点、变量具有数据或程序空间断点。
- 6、支持汇编和高级语言(C,C++)开发。
- 7、可以对 Flash、EEPROM、熔丝位、加密位进行编程。
- 8、通讯速率可达 115200bps。
- 9、支持仿真电压 2.3~6V， JTAG 仿真器可由目标供电或外接电源供电。
- 10、使用原厂 AVR Studio 的升级文件。当有新版本时，能自动检测并自动升级。
- 11、支持芯片列表：  
AT90CAN128, ATmega128, ATmega128L, ATmega16, ATmega162, ATmega162V, ATmega165,  
ATmega165V, ATmega169, ATmega169V, ATmega16L, ATmega32,

ATmega323, ATmega323L, ATmega32L, ATmega64, ATmega64L。

产品清单：

- 1、AVR JTAG ISP 二合一编程仿真器 1 台
- 2、串口连接线 1 条
- 3、电源适配器 1 个
- 4、优质 USB 线 1 条
- 5、资料光盘 1 张

资料光盘包括（AVR JTAG ISP 二合一编程仿真器用户使用手册，AVR Studio 4.12 RC1，ICCAVR 编译器，AVR 系列芯片中英文手册，ISP 软件，Bootloader 源程序，AVR 系列新手入门教程）

小贴示:本站的仿真器输出接口具有 HC244 保护,保护 JTAG 的输出口及监控程序不受损坏.(其它简易型的 JTAG 是无保护功能,经常会出现监控程序丢失.)

## 6 AVR 大学 零点篇 怎样看懂数据手册？

首先声明一点，真正的新手是看不懂数据手册的，如果你能，说明你已经入门了。所以，当你看到数据手册头疼的时候，不要灰心，其实别人和你一样。

但是数据手册是 AVR 最好的书，你又必须得看懂数据手册，看数据手册不能急于求成，要一遍一遍的多看，应该说每次看都有不同的收获。

新手看第一遍，迷迷糊糊。

看第二遍，大体了解 AVR 有哪些模块。

第三次看，知道 AVR 的部分寄存器，但是仍然不知道如何使用，如何编程。

在这里就要发挥 ICC AVR 的优势了，使用 ICC avr 的程序生成向导（使用方法会在新手入门里面介绍），你使用哪一个模块，就生成哪一个模块的程序，然后再去看程序中使用到了哪写寄存器，再到数据手册里面去搜索相应词语，查看寄存器各个位的介绍和意义，以及设置方法。多这样几次，你就能够熟悉起来了，也就能看明白数据手册了。

一个精通单片机的人，无论拿来一个什么芯片，看看数据手册，很快就可以使用。

这里以定时器的数据手册为例：

```
//ICC-AVR application builder : 2007-3-22 10:17:15
// Target : M16
// Crystal: 7.3728Mhz

#include
#include

void port_init(void)
{
    PORTA = 0x00;
    DDRA  = 0x00;
    PORTB = 0x00;
    DDRB  = 0x00;
    PORTC = 0x00; //m103 output only
    DDRC  = 0x00;
    PORTD = 0x00;
    DDRD  = 0x00;
}

//TIMER0 initialize - prescale:1024
// WGM: Normal
// desired value: 10mSec
// actual value: 9.861mSec (1.4%)
void timer0_init(void)
{
    TCCR0 = 0x00; //stop
```

```

    TCNT0 = 0xB9; //set count
    OCR0  = 0x47; //set compare
    TCCR0 = 0x05; //start timer
}

#pragma interrupt_handler timer0_ovf_isr:10
void timer0_ovf_isr(void)
{
    TCNT0 = 0xB9; //reload counter value
}

//call this routine to initialize all peripherals
void init_devices(void)
{
    //stop errant interrupts until set up
    CLI(); //disable all interrupts
    port_init();
    timer0_init();

    MCUCR = 0x00;
    GICR  = 0x00;
    TIMSK = 0x01; //timer interrupt sources
    SEI(); //re-enable interrupts
    //all peripherals are now initialized
}

```

上面的程序是由 **ICC** 的程序生成向导自动生成的，红色部分是和定时器 **0** 相关的，那么现在你可以去查数据手册，**TCCR0**，**TCNT0**，**OCR0**，**TCCR0**，**TIMSK** 你就可以充分了解定时器的工作原理及控制方法了。

比如：查看 TIMSK，描述为

中断屏蔽寄存器— TIMSK

Bit 1 – OCIE0: T/C0 输出比较匹配中断使能

当 OCIE0 和状态寄存器的全局中断使能位 I 都为“1”时，T/C0 的输出比较匹配中断使能。当 T/C0 的比较匹配发生，即 TIFR 中的 OCF0 置位时，中断服务程序得以执行。

Bit 0 – TOIE0: T/C0 溢出中断使能

当 TOIE0 和状态寄存器的全局中断使能位 I 都为“1”时，T/C0 的溢出中断使能。当 T/C0 发生溢出，即 TIFR 中的 TOV0 位置位时，中断服务程序得以执行。

可以看出 **TIMSK = 0x01**；说明 T/C0 溢出中断使能了，允许寄存器定时溢出中断，这就使得下面这段程序可以运行。

```
#pragma interrupt_handler timer0_ovf_isr:10
void timer0_ovf_isr(void)
{
    TCNT0 = 0xB9; //reload counter value
}
```

下面一段是一些朋友的理解：

古欣(286629322) 10:20:15

我正在写 新手入门中的 怎样看懂数据手册，大家有什么意见或好的建议

学习(603761546) 10:15:50

我想比较麻烦. 我遇到的最大困难就是英语

古欣(286629322) 10:21:30

看懂中文的呢

学习(603761546) 10:16:49

数据手册. 只要是中文. 应该很好看的. 除非电子基础不好, 不知道

里面的数据代表什么

学习 (603761546) 10:17:03

这是我的看法啊

BG1 (530377725) 10:17:25

如果能看懂某功能寄存器的设置方法，该功能就不是问题了

学习 (603761546) 10:17:36

也是我以前入门时的看法

BG1 (530377725) 10:18:19

单片机，我觉得还是应该以掌握寄存器为主

古欣(286629322) 10:24:07

我觉得 真正的新手是看不懂数据手册的

学习 (603761546) 10:19:14

其实. 古老师. 不如帮助把英语资料给翻译一下. 倒是比较好的方法

独步(76759976) 10:19:23

就是就是，我就看不懂

BG1 (530377725) 10:19:35

其实使用单片机，最后不过是控制寄存器而已

独步(76759976) 10:19:57

没这么简单吧？

我可不大懂

BG1 (530377725) 10:20:26

我觉得就这么简单

BG1 (530377725) 10:22:38

当然，c 的语法或汇编代码及他们伪指令的掌握是必需的

独步(76759976) 10:26:19

呵呵，这一说东西就又多了

BG1 (530377725) 10:27:42

语言只是怎么去用寄存器，看 datasheet 是为了怎么用寄存器

BG1 (530377725) 10:28:28

难道没有人认同吗

古欣(286629322) 10:34:27

认同, 过来之后认同, 但是刚开始学的时候不是这样的

BG1(530377725) 10:30:12

那是怎样的呢

古欣(286629322) 10:35:50

一团遭

学习(603761546) 10:33:33

怎样看懂数据手册? 其实没有什么写的!我这样认为

:第一... 怎样找什么器件完成你的功能... 这你必须有电子基础

;二:得到器件的名字. 你得使用器件的功能... 看资料

第三:怎么看懂... 你得认识语言. 转换为能接受的信号

所以. 都是自己设计产品时. 一系列基本联想到的步骤, 基本别人不说也因该这样走下去的....

个人理解

BG1(530377725) 10:33:39

呵呵, 我刚开始学 51 的时候, 还不会使用 keil 的调试功能呢, 那时候也不知道程序会跑出什么结果来, 老是下载程序, 得不到正确的结果,

所以, 新手掌握调试方法可以少走很多弯路

学习(603761546) 10:34:43

所以... 感觉写这个. 不如帮助翻译.. 这样比较好

古欣(286629322) 10:40:40

一会儿我就写完了, 呵呵

BG1(530377725) 10:35:32

不是的, 给新手指明学习方法很重要的

BG1(530377725) 10:36:26

很多新手就是因为第快板不顺利, 没信心而放弃的

BG1(530377725) 10:36:56

哦，写错了，是“第一块板”

古欣(286629322) 10:42:52

所以，我说成功的单片机高手后面有一块优秀的板

学习(603761546) 10:37:31

忘记说了..兴趣是学习的关键

学习(603761546) 10:38:31

如果第一块板都坚持不下去,还要继续吗..因该放弃

BG1(530377725) 10:39:19

假如第一块板成功了，会加强兴趣的

古欣(286629322) 10:44:51

嗯

学习(603761546) 10:40:41

假如第一块板成功了，会加强兴趣的.....这是逼你学

不是你要学

学习(603761546) 10:41:18

.如果你要学.第一块算什么

BG1(530377725) 10:41:43

第一块算催化剂

学习(603761546) 10:41:45

这就说的太元了

学习(603761546) 10:44:58

其实.我最想做的事情.就是出来自己开个店

BG1(530377725) 10:45:52

为 Money 而奋斗，但是技术也是重要的，我写程序去了

学习(603761546) 10:49:32

这几年做下来..感觉最重要的是模拟电路和英语.....

这次这个任务结束.好好放下心来学习模拟电路



## 7 AVR 大学 零点篇 用 C 还是汇编？

这是一个非常有争议的问题，以前有人推崇汇编，是因为他不会 C，有人也说有的人说不能用汇编写大的程序是因为他没有学好汇编。我[希望你看过这个页面之后不要再去寻找是学汇编还是学 C 的答案](#)。

一个合格的硬件工程师，应该都学，我推荐你[先学习 C 语言](#)，因为汇编入手太慢，写程序要以 C 为主，需要高速的或者底层的操作用内嵌汇编的方式。

汇编和 c 同样重要，相互配合，缺一不可！

汇编的重要性：

- 帮助你从根本上彻底和完全了解芯片的结构和性能，以及工作原理，如何使用。
- 在小的芯片上实现小的系统。
- 系统的调试。尽管你使用了高级语言，在调试中可以帮助你了解 C 代码的性能和特点，甚至找到使用开发平台本身的 BUG。
- 编写时序要求严格的代码，实现一些高级语言不易实现的功能。

从目前的技术和应用发展来看，对硬件工程师的要求越来越高。以我的观点，作为单片机和嵌入式系统开发真正的高手，应具备以下几个方面的综合能力：

- 硬件。模拟、数字电路的雄厚基础，了解跟踪现在市场上的各种元器件的应用和发展，能够进行可靠、完善的电路设计以及 PCB 的设计。
- 软件。不仅需要精通汇编语言，也要精通 C 语言，要有好的单片机系统程序设计理念和能力，学校中学的那些分支结构、循环结构等基本原理远远不够！要有基本的数据结构的知识。否则你如何设计实现 USB HOST 读 U 盘的接口？如何实现嵌入式 WEB 系统？以及如何使用真正了解和使用 RTOS？
- 具备计算机网络和数字通信的基础知识，从根本上熟悉和了解各种协议的构造和实现，如：UART、RS232、SPI、I2C、USB、IEEE802、TCP/IP 等。
- 计算机应用的高手。

- 熟练阅读英文资料。
- 热爱和喜欢电子技术，具备刻苦精神、踏踏实实，不弄虚作假，不浮躁。多动手，勤实践。有强烈的专业和钻研精神。**最后一条最重要！**

## 8 AVR 大学 零点篇 avr C 语言入门知识

本页关键词：avr单片机学习 avr C语言入门知识 c语言学习

### 1、基本语法介绍

一个简单的 AVR 程序

```
#include <iom16v.h>
#include <macros.h>

void main()
{
    PORTA = 0x0F; //给 PA 口赋值，让 PA 口低四位为 1，高四位为 0
    while(1)
    ;
}
```

本程序的作用是把 PA 口的值设为 0x0F。

1、程序中以井号开头的语句 `#include <iom16v.h>` 是包含特定的头文件，叫预处理指令，`iom16v` 表示使用的是 `mega16`，`macros.h` 包含了必须的 `avr` 操作命令。

2、C 语言的程序是由函数构成的，如上面的那个 `void main()`，前面的 `void` 表明函数没有返回值。每一个 `c` 程序里面里有且只有一个 `main()` 函数，系统启动后就从 `main()` 开始运行。

3、函数内部的内容以大括号“{”和“}”扩起来，每句语句用分号“；”结束，若分号前面没有内容，编译之后也无任何操作语句。

4、C程序中可以加入一些说明文字，单行以双斜杠“//”开始，如果是多行，就用“/\*”开始，以“\*/”结束，如 `/*注释 */` 。

5、函数可以有参数，一律放在小括号内。

6、利用 C 语言可以轻松的对 AVR 的设备组件进行操作，如程序中的 `PORTA = 0x0F;`

7、任何一个 AVR C 程序都必须是一个无限循环，否则程序会沿着程序存储区一直运行，直至溢出程序存储区，程序从头运行。

## 2、AVR C 语言的基本字符、标识符和关键字

avr c语言和普通c语言一样，**基本字符** 有阿拉伯数字 0~9；大小写拉丁字母a~z和A~Z；一些选定的可打印字符，如 " ~! @#¥%^&\* ( ) \_-+= { } [ ] , . ; <> / ? | \ " ；空格符、换行符和制表符这三种空白符起到分割成分和编排格式的作用。

对系统对象命名，称为 **标识符**。标识符由数字、字母、下划线组合的字符串序列构成，**字母区分大小写**。如下都是合法的表示符：

`AVR_IO PORTA CSR IT0 temp1`

注意：**不能以数字开头的字符串做标识符**。比如 `1abc` 是不合法的。

C语言的合法标识符有一部分被编译器保留作为特殊用途，这样的标识符称为 **关键字**。C语言的关键字有：(以字母先后为序)

`auto break case char const continue default do double else enum extern float for goto if int long register return short signed`

`sizeof static struct switch typedef union unsigned void volatile while`

注意，原则上除关键字外，可以使用任何有效的标识符。但实际上，根据系统不同，有些特殊标识符具有特殊含义，不应被使用。在 AVR 里，一些端口的名称、寄存器的名称已经被系统定义，最好不要改变其定义，如 `PORTA DDRA TIMASK` 等。

### 3、数据类型

C 语言严格规定数据类型，AVR 资源有限，如果数据类型选用不好，资源很快就会耗尽。如，尽量不要使用浮点类型的运算，`1.2*1.2` 的浮点数运算至少要占用 `mega16` 的百分之十以上的空间。

选择数据类型时需要注意不要操作数据能表示的范围，比如要表示 `60000`，就不能用 `char`，必须用 `int`。

整数类型的类型名前可加修饰符 `unsigned` 和 `signed`，表示无符号数和有符号数，其中 `unsigned` 可以省略，默认表示无符号数，一般来说，尽量使用无符号数可以节约资源。

以下三种为整数类型：

**1、整数类型：**一般类型的整型 `int`，16 位二进制编码，表示的数  $0 \sim 65536$ ，及  $2^{16}$ 。有符号类型，`signed int` 表示范围  $-32768 \sim 32767$ 。

**2、长整型类型：**长整型（`long int`）类型的二进制编码是 32 位。有符号的长整型类型（`long`）表示范围  $-2^{31} \sim 2^{31}-1$ ；无符号的类型（`unsigned long`）的表示范围为  $0 \sim 2^{32}-1$ 。

**3、超长整数类型：**超长整数类型（`long long`）的二进制编码是 64 位。有符号的长整型类型（`long`）表示范围  $-2^{63} \sim 2^{63}-1$ ；无符号的类型（`unsigned long`）的表示范围为  $0 \sim 2^{64}-1$ 。

以下为实数类型（浮点数类型）：

**1、单精度浮点数类型：**`float`，用 32 位二进制数表示。

**2、双精度浮点数类型：**`long double`，用 64 位二进制数表示。

字符类型和字符串

字符类型的类型名为 `char`，目前最常用的是 `ASCII` 字符集，其中字符包扩所有的大小写字母，数字，常用字符等共计 128 割字符。扩展的 `ASCII` 字符集包括 256 个字符，字符类型占用一个字节。

一些特殊的字符串的表示方法：以反斜杠加特定字符。如 `'\n'` 回车符；`'\"'` 双引号；`'\''` 单引号字符；`'\\'` 反斜杠。

**无符号的字符类型：** `char` 表示范围 0~255。

**有符号的字符类型：** `signed char`表示范围 -128~127。

## 9 AVR 大学 零点篇 怎样高效快速学习 AVR 单片机？

怎样成为单片机高手，下面是本站的建议[学习的流程](#)。

一、购买一两本书，笔者推荐两本 《单片机 C 语言开发入门指导》，《高档 8 位单片机 ATmega128 原理与开发应用指南》。买书的目的：看书大体了解单片机的结构和工作原理，了解基本概念和基础知识，其实新手是不可能完全看懂一本书的，如果你能，你已经是高手了，所以不要期望一字一句去搞懂书上说的到底是什么东西。看完书对相关内容有个概念性的了解就可以了。

二、开始动手配置开发环境，动手去做，实践出真知。笔者推荐使用 `ICC AVR` + `AVR studio` + `AVR mega16` + `JTAG&ISP` 下载仿真器的组合。抄几个程序，增强一下自己的信心，看到自己的程序在单片机上跑起来，那种愉悦的心情是和用软件仿真仅仅看到 `IO` 口的变化是截然不同的，不在一个层次。（关于软件硬件开发环境的选择和环境如何配置后面都有讲述）

三、抄过一段时间的程序后，你需要自己动手写程序，如果你是新手，你需要一点一点做起，从 `IO` 口操作，到定时器，到数码管，`AD` 转换，键盘，中断，`LCD` 等等，通过实际写程序去了解芯片的工作情况，那样你将进步得更快。

四、想一想自己要做个什么东西，围绕一个主题去展开学习，去查相关的资料。事件驱动的学习效率是最高的，你可以更深层次的理解应该学习什么，怎样的学习才有效，学到的内容在实际操作中有用。

五、关于开发板的选择

1. 一个成功的男人后面有一个能干的女人；
2. 一个失败的男人后面有一群妩媚的女人；
3. 一个成功的单片机高手后面有一块优秀的开发板；
4. 一个失败的开发板后面有一群劣质的开发板。

选择一个好的开发板是非常重要的，要根据实际情况，如果你是单片机新手，建议你购买有个功能全面，外围多的开发板系统的学习，如果你是从其他单片机转型过来，建议你购买一个最小系统板就可以，还有一个方法就是自己用万用板焊接学习板，不过比较费时间，而且出了问题，不知道到底是程序不行还是自己的硬件不行，就会出现问题。

## 10 AVR 大学 零点篇 怎样看本站的新手入门教程

本站的教程实例都分为 **学习目标 | 硬件 | 软件 | 说明 | 总结** 几个部分。

1、你先了解学习目标，然后大体知道要做什么，看看硬件，看看程序代码。

2、如果觉得自己大体上能做出来，那么就可以下手搭建硬件，如果你购买本站的开发板，只需要使用连接线把各部分连接起来即可。如果没有，你可以用万用板搭建电路。实在不愿意搭电路，就可以使用 studio 仿真，但是不推荐，因为一直仿真你永远无法成为高手。

3、搭好硬件，抄上软件，下载到芯片中，查看运行效果。

4、查看硬件连接，熟悉硬件连接情况，单片机编程硬件当先。

5、查看程序的说明，深入了解程序流程，和编程思路方法等。

6、自己动手写一遍程序（仅限于比较简单的，如果是模块化的驱动，你就不必要花时间去写了），举一反三，写写类似的程序。

7、查看总结说明，并记住其中的描述，一些经验，编程规则等,这样有助于你的成长。

说明:以上描述虽然加了编号，但是你不需要严格按照这个步骤来做，可以根据自己的实际情况调整。

实践出真知，时间证明一切，肯动手，肯花时间，你就能成功。愿你在 AVR 单片机学习的路上走好。

## AVR 芯片入门知识

ATmel 挪威设计中心的 A 先生与 V 先生，于 97 年设计出一款使用 RISC 指令集的 8 位单片机，起名为 AVR。

AVR 芯片的主要特性，及与其它单片机比较的优点，相信我不必多说了，大家随便找一本参考书就可以看到洋洋洒洒的十几页的介绍。如果你想看到只有一页的介绍，可以参考我们网站上的资料：[AVR 单片机性能简介](#)。我就 AVR 单片机分 3 个档次，四种封装做一个介绍。

AVR 单片机系列齐全,可适用于各种不同场合的要求。AVR 单片机有 3 个档次：

低档 Tiny 系列 AVR 单片机: 主要有 Tiny11/12/13/15/26/28 等；

中档 AT90S 系列 AVR 单片机: 主要有 AT90S1200/2313/8515/8535 等；（正在淘汰或转型到 Mega 中）[查看详细情况](#)

高档 ATmega 系列 AVR 单片机: 主要有 ATmega8/16/32/64/128（存储容量为 8/16/32/64/128 KB）以及 ATmega8515/8535。新的型号还有 ATmega48/88/168（存储容量为 4/8/16K）等。

如果你想获得最新的 AVR 芯片资料，可以下载：[2006-11 AVR 芯片选型指南](#)，包含所有 AVR 芯片的参数信息。

AVR 器件引脚从 8 脚到 64 脚（新的芯片高达 100 脚），还有各种不同封装供选择。FLASH, RAM 及配置的不同，形成比较宽的产品线系列。详细的选型信息可以参考本网站的[AVR 单片机全系列性能参数表](#)。

AVR前几年已经显示了进军中国市场的决心。几乎所有的AVR主流芯片，都已经有了官方正规翻译的中文 DataSheet(数据手册)。我们网站整理了国内最完整的中文datasheet供大家下载学习：[点击打开AVR数据手册下载界面](#)。

虽然我们网站也收录了双龙翻译的一些旧芯片资料，但建议大家不要使用，错误较多，并且严重的偷工减料。官方翻译的中文手册比较严谨，但仍可能存在一些小缺陷。有需要时，请参考英文版本：[点击打开AVR数据手册下载界面](#)。

AVR 芯片型号的解释， 以 ATmega48V-10AI 为例：

ATmega48 代表产品。

V 代表低/宽电压版本。新出的 AVR(M48/88/168,Tiny13/2313.....)产品系列来说, -V 是 1.8-5.5V 工作范围。 不带"V"是 2.7-5.5V 工作范围 。 老的系列以 L 表示, 2.7-5.5V 工作范围 ， 不带"L"是 4.5-5.5V 工作范围 。

-10, 表示最高工作频率, 10MHz

A, 表示封装 。AVR芯片有四种封装：（如果你不熟悉封装，请参考我们为你准备的资料：[AVR封装图例](#)）

1. A: TQFP
2. P: PDIP
3. S: SOIC
4. M: MLF

I, 表示温度范围，将来还表示 ROHS

1. I, 工业级
2. C, 商业级



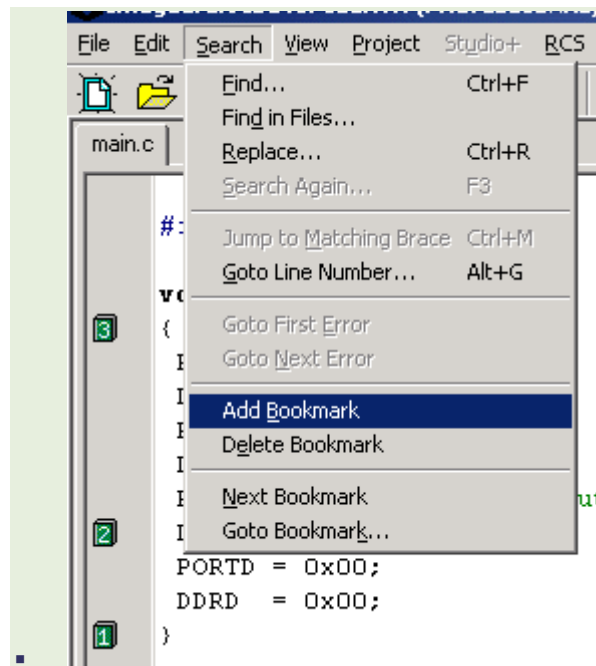
3. A, 汽车级
4. E, 扩展级 (-40--105C)
5. U, 符合 ROHS,工业级, 将来会以这个后缀为主, 商业级不作新的标示

**AVR ICC 使用快速入门 菜单解释 下一页: [新手的第一个程序, 实现红绿灯不断闪烁。](#)**

ICC AVR是一款非常好用的AVR编译软件, 官方网站: [www.imagecraft.com](http://www.imagecraft.com)目前最新版为 7.0, 本站的所有例程都以ICC AVR为开发平台。如果你还没有配置好开发环境, 请参看: [AVR 开发环境](#)  
[ICC+AVRstudio配置](#)。如果你有时间看很多理论的没有实际操作价值的说明, 请下载 [ICCavr中文使用说明](#)。如果你想快速进入使用, 请参看本文档。

1、关于窗口设置: [如图]



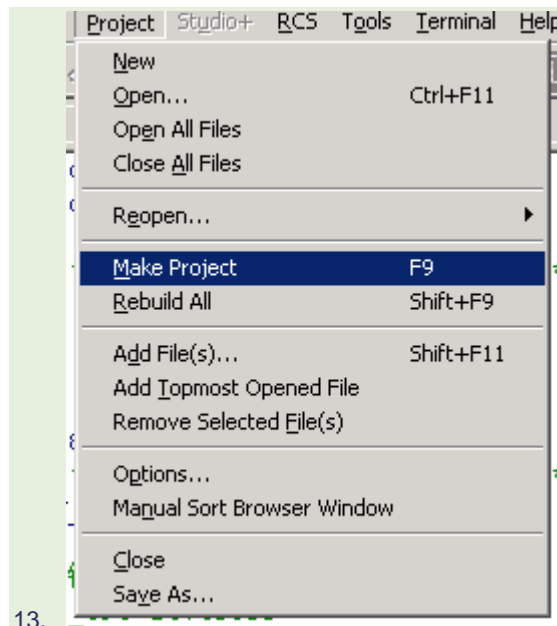


#### 4. 视图

- 对新手几乎没有用处, 你可以用它显示或者隐藏工程文件窗口, 编译状态窗口。  
查看输出文件, makefile 文件, Map 文件。

#### 5. Project Menu 工程菜单

1. New... – 创建一个新的工程文件
2. Open – 打开一个已经存在的工程文件
3. Open All Files... – 打开工程的全部源文件
4. Close All Files – 关闭全部打开的文件
5. Reopen... – 重新打开一个最近打开过的工程文件
6. Make Project – 解释和编译已经修改的文件为输出文件 \*注意与 7Rebuild All的区别
7. Rebuild All – 重新构筑全部文件注意在版本升级后对原有工程最好全部重新构筑 \*
8. Add File(s) – 添加一个文件到工程中这个文件可以是非源文件
9. Remove Selected Files – 从工程中删除选择的文件
10. Option... – 打开工程编译选项对话框
11. Close – 关闭工程
12. Save As... – 将工程换一个名称存盘



13.

#### Tools Menu 工具菜单

0. Environment Options – 打开环境和终端仿真器选项对话框
1. Editor and Print Options – 打开编辑和打印选项对话框
2. AVR Calc – 打开 AVR 计算器可以计算
3. UART 的波特率定时器的定时常数
4. Application Builder – 打开应用向导程序生成硬件的初始化代码 **cool**, 对新手非常有用
5. Configure Tools – 允许你添加自己的内容到工具菜单
6. Run – 以命令行方式运行一个程序

#### Terminal 电脑终端

- 使用很少，串口调试 终端

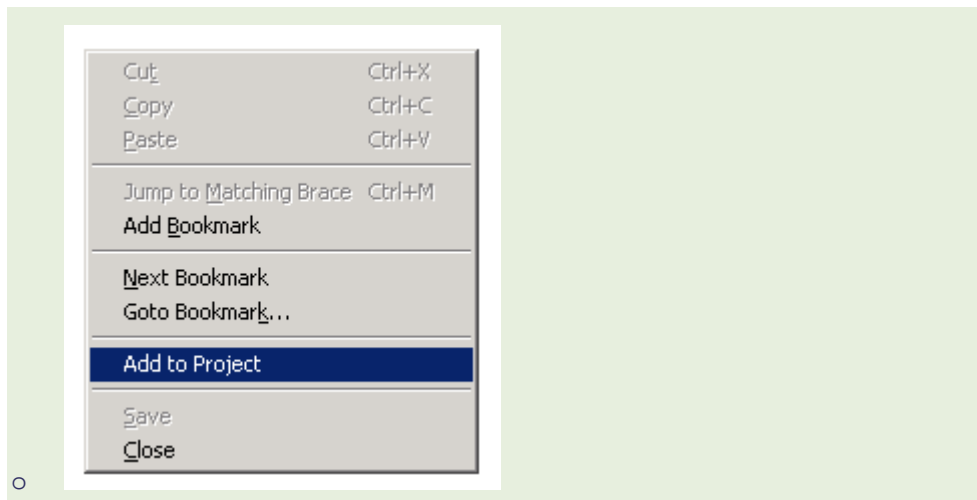
Help 帮助 不用我说了。

#### 快捷菜单图标

- 从左到右依次为：新建，打开，保存，打印，剪切，复制，粘贴，查找，编译，停止，工程属性，到第一个错误，下一个错误，应用向导程序 Application Builder, ISP 对话框。

#### 程序编辑区

- 所有的程序编辑工作在这里完成。
- 点右键有快捷菜单，最常用的事新建一个文件编辑之后，加入到工程。



编译状态显示栏 [图中为报错的情况]

- 显示编译状态，查看报错情况进行相应修改。

文件列表框

- 文件分类管理查看，这里的分类只是为了方便管理，并没有实际的分开。
- 文件分为.c 程序文件，.h 头文件，说明文档。

上一页: [ICCAvr 的菜单与界面解释](#) 新手的第一个程序 下一页: [ICCAvr 生成程序文件解释](#)

论坛链接: [ICC+avrstudio下的第一个程序](#)。本页和论坛内容相同，欢迎你到论坛参与讨论。

本例实现红绿灯不断闪烁的效果。

如果你还没有配置好开发环境，请首先看: [avr 开发环境配置](#) 【icc + AVRStudio】

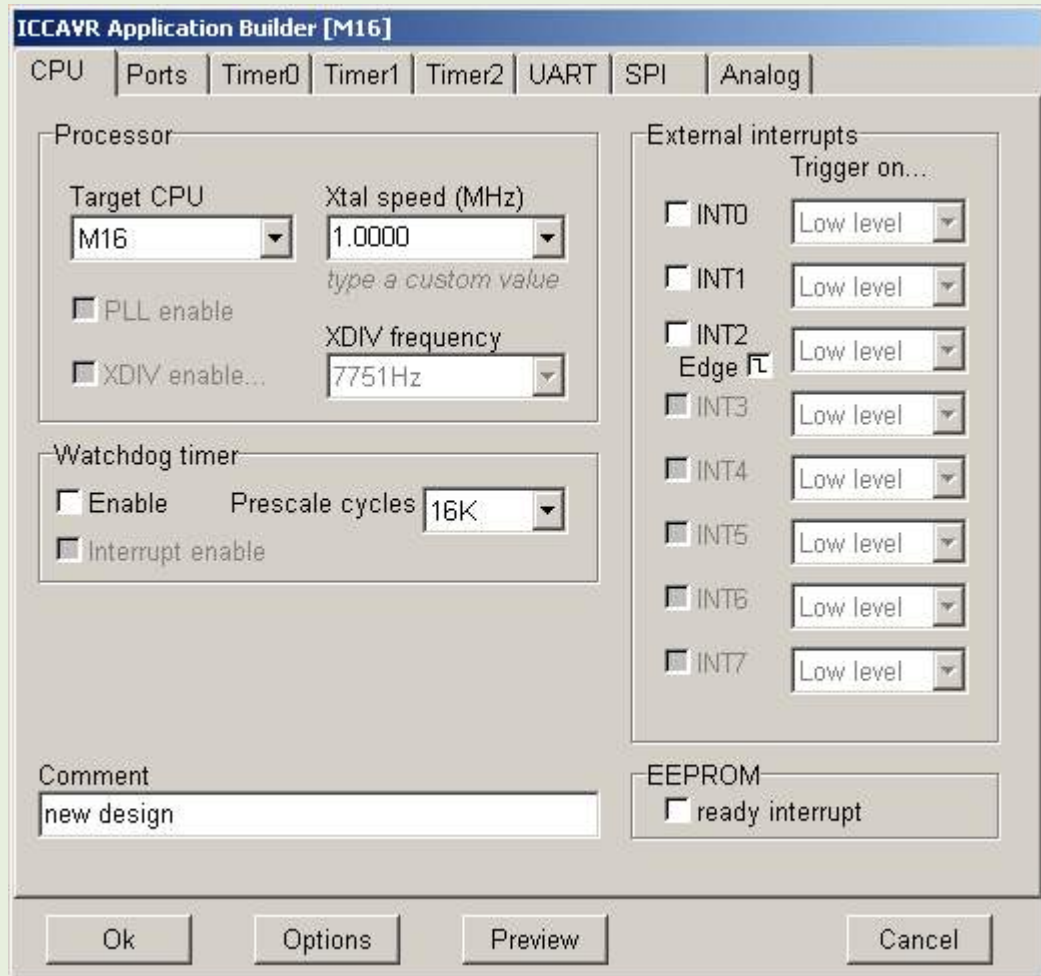
<http://bbs.avrvi.com/read-htm-tid-1.html>

第一步: 运行 iccavr, 运行 **project>>new**, 新建一个工程, 保存为 main.prj, 保存在新文件夹 avr\_first 下。

第二步: 运行 **Tools>>Application Builder** 建立工程的工具。

看到如下图的设置界面。

## icc 建立 avr 工程属性设置



将 Target CPU 改为 M16，因为我们使用的芯片是 mega16

将 Xtal speed 改为 1.0000，我们使用内部晶振，内部晶振频率为 1Mhz。

切换到 Ports 选项，作如下图所示的更改。

### Ports 选项

ICCAVR Application Builder [M16]

CPUPortsTimer0Timer1Timer2UARTSPIDigitalAnalog

Port A

	7	6	5	4	3	2	1	0
Direction	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Value	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Change	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Port B

	7	6	5	4	3	2	1	0
Direction	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Value	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Change	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Port C

	7	6	5	4	3	2	1	0
Direction	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Value	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Change	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Port D

	7	6	5	4	3	2	1	0
Direction	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Value	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Change	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

right click on a "value" bit to define signal name

OkOptionsPreviewCancel

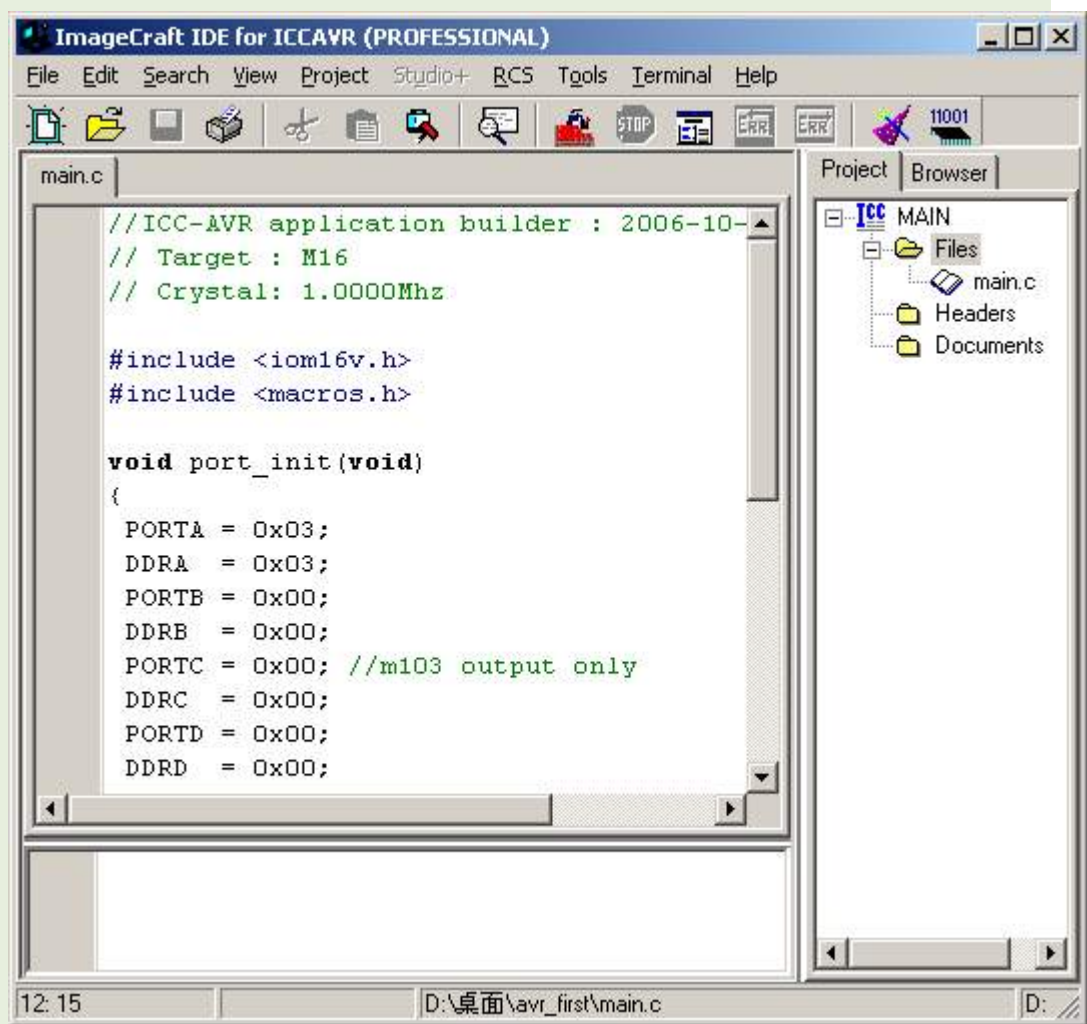
意思是使用单片机的端口 A，使用 0 和 1 两个脚输出高电平，因为我们所用二极管是低电平驱动的。

其他项不做更改，点 OK 完成。

第三步：File>>Save as 保存到你刚才第一步建立的工程的文件夹中。**注意：保存的文件名必须输入后缀名，形式如main.c。**

第四步：右键>>add to project

你将看到如下效果： 工程建立完成



第五步：在编辑区添加如下程序代码，

CODE:

```
void Delay(void)  
{  
    unsigned char i,j;  
    for(i=200;i>0;i--)  
    {  
        for(j=200;j>0;j--)  
        ;  
    }  
}
```



```

}

void main(void)
{

init_devices();      //初始化

while(1)
{

PORTA = 0x02;      //1 脚为高，0 脚为低，0 脚灯亮

Delay();      //延时

PORTA = 0x01;      //0 脚为高，1 脚为低，1 脚灯亮

Delay();      //延时

}

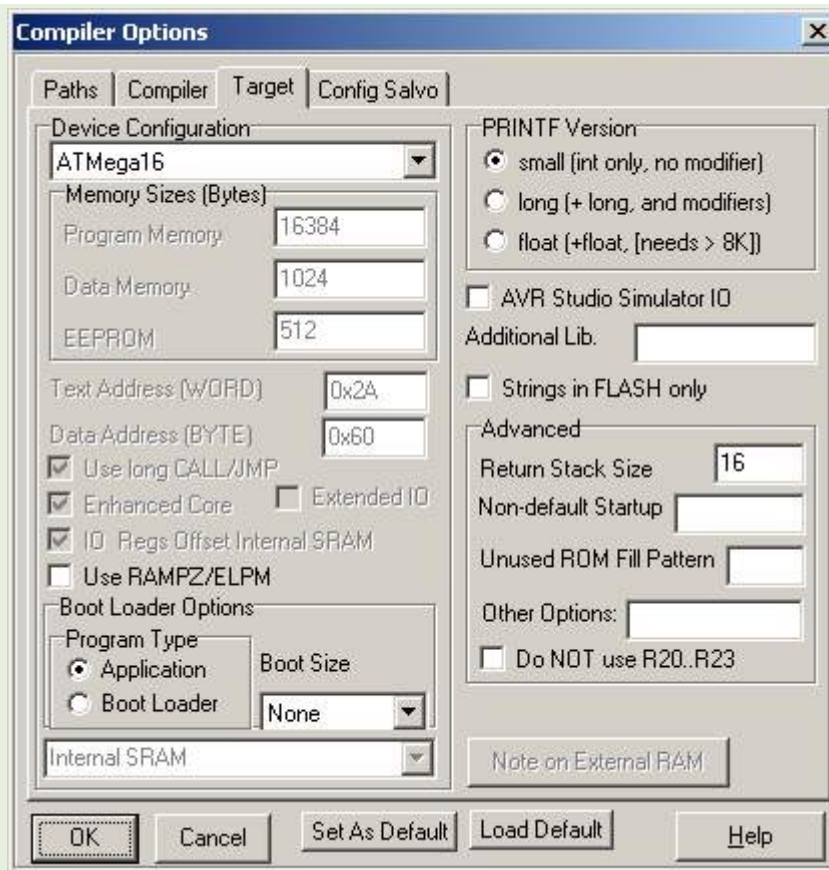
}

```

其中 Delay 为延时程序，main 为主程序，保存程序。

**第六步：Project>>options**，将Device configuration改为ATmega16，其他不变，点击ok，如下图

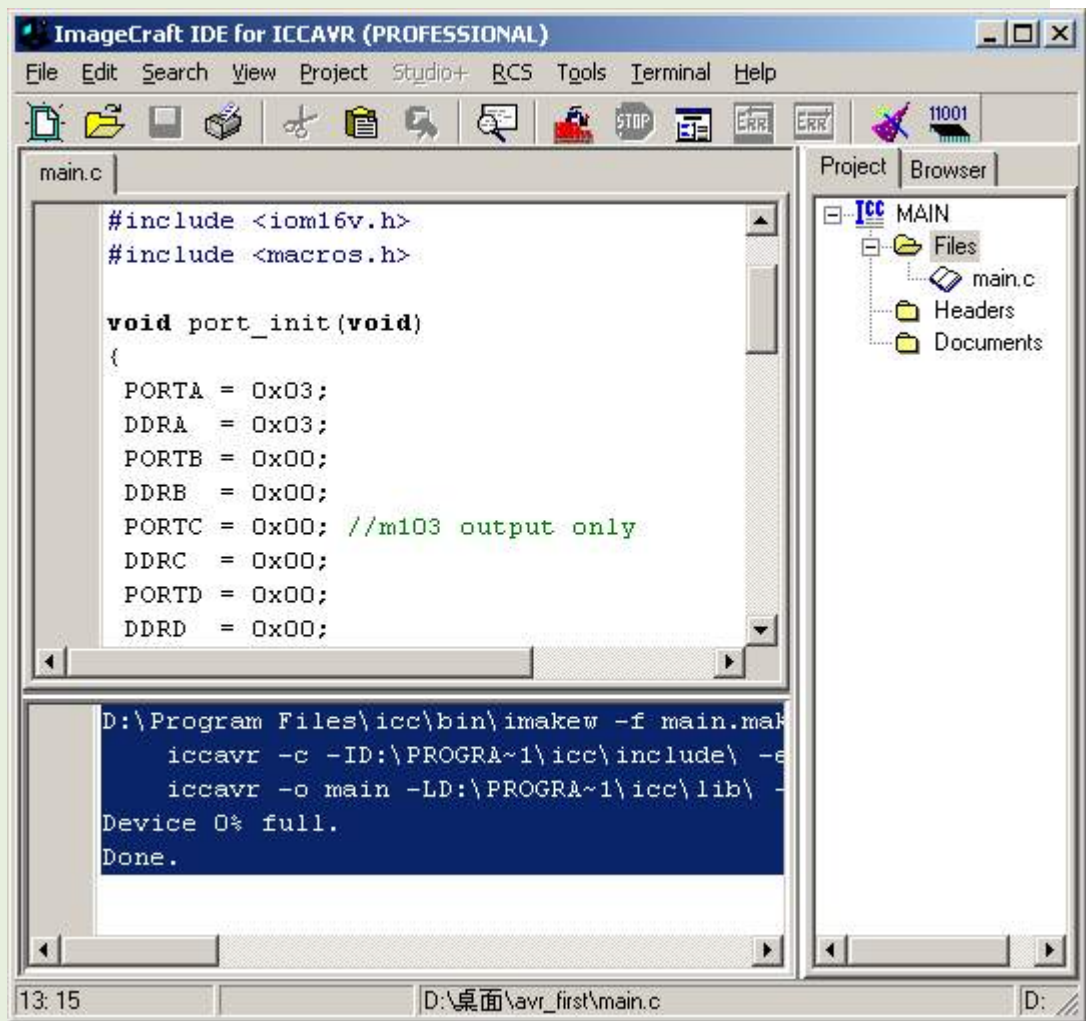
编译器环境配置



第七步：**Project>>rebuild all** 或者直接使用快捷键 **shift + f9** ，编译工程。

成功即可看到下图所示的样子。

编译完成



在状态区看到这样的代码：

```
D:\Program Files\icc\bin\imakev -f main.mak
```

```
iccavr -c -ID:\PROGRA~1\icc\include\ -e -DATMEGA -DATMega16 -l -g -Mavr_enhanced D:\桌面
\avr_first\main.c
```

```
iccavr -o main -LD:\PROGRA~1\icc\lib\ -g -ucrtatmega.o -bfunc_lit:0x54.0x4000 -dram_end:0x45f
-bdata:0x60.0x45f -dhwstk_size:16 -beeprom:1.512 -fihx_coff -S2 @main.lk -lcatmega
```

```
Device 0% full.
```

```
Done.
```

第八步：此时打开第一步建立工程的文件夹，可以看到如下文件。

文件列表

名称 ▲	大小	类型	修改时间
 main._c	1 KB	_C 文件	2006-11-11 14:43
 main.c	1 KB	C 文件	2006-11-11 14:44
 main.cof	2 KB	COF 文件	2006-11-11 10:54
 main.dbg	1 KB	DBG 文件	2006-11-11 10:54
 main.dp2	1 KB	DP2 文件	2006-11-11 10:54
 main.hex	1 KB	HEX 文件	2006-11-11 10:54
 main.lis	7 KB	LIS 文件	2006-11-11 10:54
 main.lk	1 KB	LK 文件	2006-11-11 10:54
 main.lst	4 KB	LST 文件	2006-11-11 10:54
 main.mak	1 KB	MAK 文件	2006-11-11 10:54
 main.mp	2 KB	MP 文件	2006-11-11 10:54
 main.o	2 KB	O 文件	2006-11-11 10:54
 main.prj	1 KB	PRJ 文件	2006-11-11 10:54
 main.s	3 KB	S 文件	2006-11-11 10:54
 MAIN.SRC	1 KB	SRC 文件	2006-10-16 9:27
 main_cof.aps	3 KB	aps File	2006-11-11 11:01

至此，在ICC AVR下的工作完成，请继续下一步，[ICCavr生成程序文件解释](#)。

## 上一页：[使用 ICC+AVRstudio 编写第一个程序](#) ICCavr 生成程序文件解释

ICC avr 编译成功后会生成很多文件，对我们普通用户以及新手，有用的只有一个文件，\*.cof 调试用，\*.hex 机器码。

首先看图：

名称 ▲	大小	类型	修改时间
 main._c	1 KB	_C 文件	2006-11-11 14:43
 main.c	1 KB	C 文件	2006-11-11 14:44
 main.cof	2 KB	COF 文件	2006-11-11 10:54
 main.dbg	1 KB	DBG 文件	2006-11-11 10:54
 main.dp2	1 KB	DP2 文件	2006-11-11 10:54
 main.hex	1 KB	HEX 文件	2006-11-11 10:54
 main.lis	7 KB	LIS 文件	2006-11-11 10:54
 main.lk	1 KB	LK 文件	2006-11-11 10:54
 main.lst	4 KB	LST 文件	2006-11-11 10:54
 main.mak	1 KB	MAK 文件	2006-11-11 10:54
 main.mp	2 KB	MP 文件	2006-11-11 10:54
 main.o	2 KB	O 文件	2006-11-11 10:54
 main.prj	1 KB	PRJ 文件	2006-11-11 10:54
 main.s	3 KB	S 文件	2006-11-11 10:54
 MAIN.SRC	1 KB	SRC 文件	2006-10-16 9:27
 main_cof.aps	3 KB	aps File	2006-11-11 11:01

1. main.\_c main.c 修改并保存时，程序自动备份的文件，如果确认 main.c 没有问题了，可以删除。
2. main.c 主程序文件
3. main.cof COFF 格式输出文件用于在ATMEL 的AvrStudio 环境下进行程序调试 <记住这个文件>
4. main.dbg ImageCraft 调试命令文件
5. main.dp2
6. main.hex INTEL HEX 格式文件其中包含了程序的机器代码
7. main.lis
8. main.lk
9. main.lst 列表文件在这个文件中列举出了目标代码对应的最终地址
10. main.mak
11. main.mp 内存映像文件它包含了您程序中有关符号及其所占内存大小的信息
12. main.o 由汇编文件汇编产生的目标文件多个目标文件可以链接成一个可执行文件
13. main.prj 工程文件
14. main.s 表示是汇编语言源文件
15. MAIN.SRC 工程配置记录
16. main\_cof.aps 使用 AvrStudio 调试后保存的调试环境相关信息。

下一步，要把程序写到芯片里面去了：

1. 如果你用的是本站的 [豪华版JTAG&ISP二合一 \(AVR JTAG & ISP v3.0\)](#) 或者是 [企业版JTAG&ISP二合一 \(AVR JTAG & ISP v3.0\)](#)，请查看使用说明  
书：[http://www.avrvi.com/start/AVR\\_JTAG\\_ICE\\_ISP\\_STK500\\_USER\\_GUIDE.pdf](http://www.avrvi.com/start/AVR_JTAG_ICE_ISP_STK500_USER_GUIDE.pdf)，按照说明书进行操作。
2. 只是下载程序，你可以：[STK500/ISP JTAG 烧录快速入门](#)
3. 要使用JTAG仿真：[AVR JTAG在线仿真调试快速入门](#)
4. 或者 [使用ICC和并口ISP下载线下载程序](#)

## STK500 JTAG 下载烧录快速入门

导读：本文介绍 AVR 官方唯一推荐的下载方法：STK500。也介绍了 AVR Studio 同时支持的 JTAG 下载，如果要仿真的话，还是少不了 JTAG，所以 JTAG 下载也很常用。并口下载由于速度很慢，AVR Studio 也不支持并口下载，(仅能使用第三方的软件下载) 故我们不推荐使用。我们的感觉：用过 STK500 下载后，就不会再使用并口下载了。感觉是两种完全不同档次的方式，不过并口下载成本要低很多。

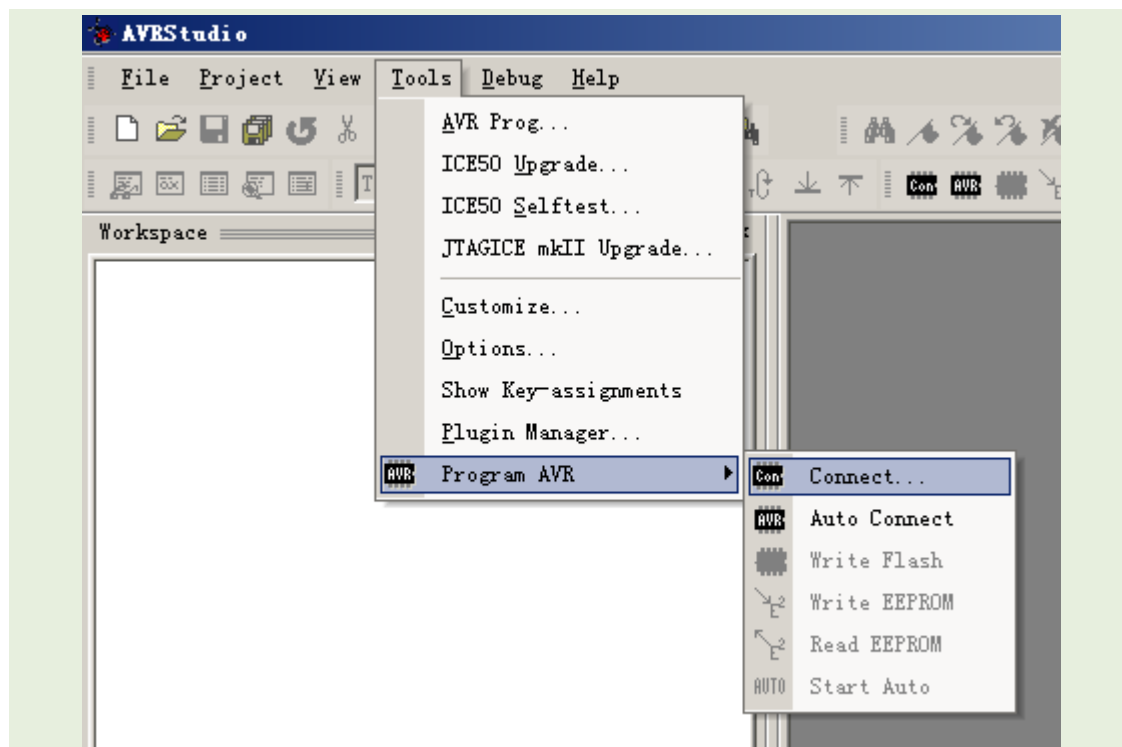
如果你没有配置好开发环境，请看 [ICC avr + AVRstudio 开发环境的配置](#)。

请确保你已经了解AVR Studio：[AVR Studio 快速入门](#)。

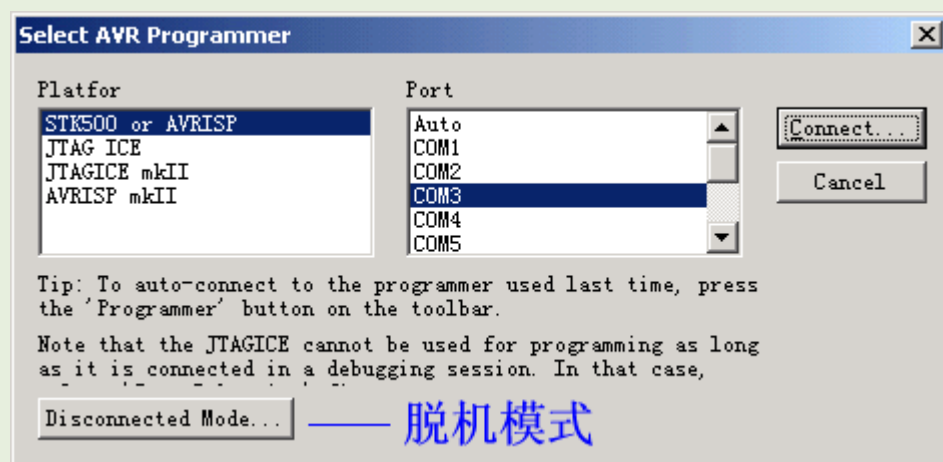
### 一：STK500 下载

支持的芯片：支持全系列的 AVR 芯片。并且，支持未来的 AVR 新芯片。实际生产过程中，很多情况下是不留 JTAG 电路的，因为 JTAG 电路要占用 IO 口，并且并不是所有的芯片都支持 JTAG 下载，所以使用 STK500 下载很重要。

操作方法：打开 AVR Studio 软件，按下图操作。



Connect 与 Auto Connect 的区别，是每次都会提示选择的设备名称与连接端口。Auto Connect 会自动使用上一次的设置，提高操作效率。使用 Connect 会弹出如下的界面：



Port 口为硬件连接端口，如果使用本站的下载器，usb 设备会默认模拟到 com3，如果你不清楚，选择 Auto 即可。如果你没有连接 STK500、JTAG、mkII 等设备，可以使用 Disconnected Mode (脱机模式)进入查看操作界面。

如果你已经按下图连接好，就能按 Connect 进行连接了：（注意：JTAG 下载时接 JTAG，ISP 下载时接 ISP，普通的编程器如果接错了，有可能会烧坏芯片或者丢失程序，本战的编程器有极强的保护功能，不会有这个危险。）





## 编程界面



# 高级选项

AVRISP

Program Fuses LockBits Advanced Board Auto

Signature Bytes **器件识别码**

0x1E 0x94 0x03 Read

Signature matches selected device

Oscillator Calibration byte

Calibrate for

1.0 MHz **时钟校正**

Value Write

0x9B  ☒ Flash ☐ Eeprom Read Cal. Byte Write to Memory

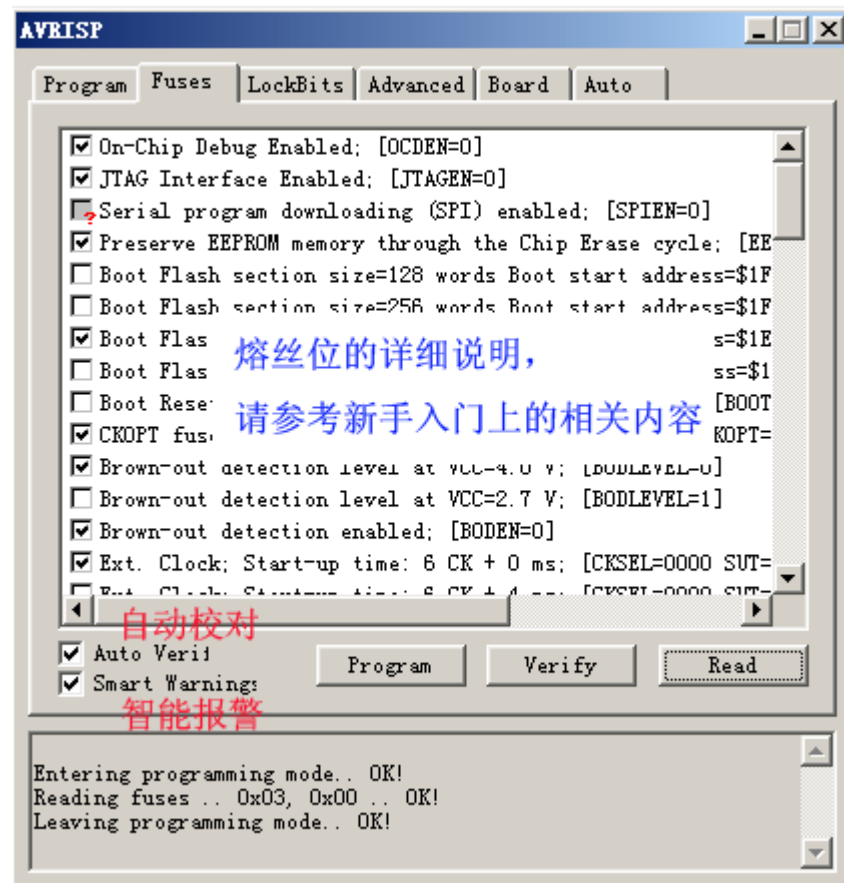
Communication Settings

Baud 115200 Baud rate changes are active immediately.

**通信设置**

Entering programming mode.. OK!  
Reading osc. cal. byte .. 0x9B .. OK!  
Leaving programming mode.. OK!

# 熔丝位



# 存储器锁定位



存储器锁定位(从上到下，一一对应)			保护类型
lb 模式	lb2	lb1	
1	1	1	没有使能存储器保护特性
2	1	0	在并行和 spi/jtag 串行编程模式中 flash 和 eeprom 的进一步编程被禁止，熔丝位被锁定。(1)
3	0	0	在并行和 spi/jtag 串行编程模式中 flash 和 eeprom 的进一步编程及验证被禁止，锁定位和熔丝位被锁定(1)
blb0 模式	blb02	blb01	
1	1	1	spm 和 lpm 对应用区的访问没有限制

2	1	0	不允许 <b>spm</b> 对应用区进行写操作
3	0	0	不允许 <b>spm</b> 指令对应用区进行写操作，也不允许运行于 <b>boot loader</b> 区的 <b>lpm</b> 指令从应用区读取数据。若中断向量 位于 <b>boot loader</b> 区，那么执行应用区代码时中断是禁止的。
4	0	1	不允许运行于 <b>boot loader</b> 区的 <b>lpm</b> 指令从应用区读取数据。若中断向量 位于 <b>boot loader</b> 区，那么执行应用区代码时中断是禁止的。
<b>blb1 模式</b>	<b>blb12</b>	<b>blb11</b>	
1	1	1	允许 <b>spm/lpm</b> 指令访问 <b>boot loader</b> 区
2	1	0	不允许 <b>spm</b> 指令对 <b>boot loader</b> 区进行写操作
3	0	0	不允许 <b>spm</b> 指令对 <b>boot loader</b> 区进行写操作，也不允许运行于应用区的 <b>lpm</b> 指令从 <b>boot loader</b> 区读取数据。若中断向量位于应用区，那么执行 <b>boot loader</b> 区代码时中断是禁止的。
4	0	1	不允许运行于应用区的 <b>lpm</b> 指令从 <b>boot loader</b> 区读取数据。若中断向量 位于应用区，那么执行 <b>boot loader</b> 区代码时中断是禁止的。

**notes:** 1. 在编程锁定位前先编程熔丝位。2. “1” 表示未被编程, “0” 表示被编程

# STK500下载器电路板

AVRISP

Program Fuses LockBits Advanced Board Auto

Voltages

VTarget - 6.0 ARef: - 6.0  
5.0 5.0

电压

Read Voltages  
Write Voltages

Oscillator and ISP Clock 下载器晶振与ISP频率

STK500 Osc: 3.686 MHz Attainabl3.686 MHz Read  
ISP 230.4 kHz Attainabl230.4 kHz Write

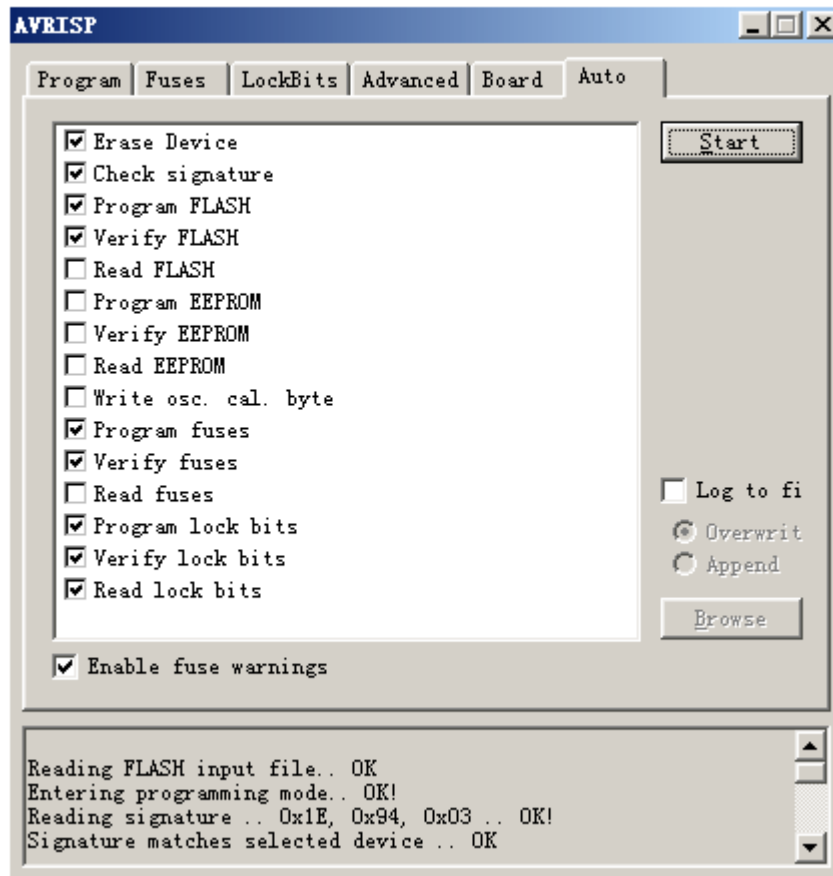
Note: The ISP frequency must be less than 1/4 of the

Revision 版本号

Hw: 0x0f, Sw. major: 0x02, Sw. minor: 0x0 Upgrade

Getting revisions.. HW: 0x0f, SW Major: 0x02, SW Minor: 0x01 .. OK  
Getting oscillator parameters.. P=0x01, N=0x00, SD=0x01 .. OK

# 自动批处理



## 二：使用 JTAG 仿真器下载

下载方法与使用 STK500 的几乎完全一样。在上面的介绍中，设备不选 STK500，改成选 JTAG ICE 就能进入。支持的芯片：仅支持带 JTAG 接口的芯片。按官方文档的描述，支持的芯片清单为：ATmega128, ATmega64, ATmega32, ATmega16, ATmega162, ATmega165, ATmega169, ATmega323 可见 JTAG 下载很有局限。另外由于通信协议的不同，感觉 JTAG 下载的速度，尤其是瞬时反应速度，没有 STK500 快。（但如果大量的数据传输，速度感觉差不多）。

## AVR jtag 在线仿真调试快速入门

本文详细介绍AVR jtag的连接连接，下载和调试方法与过程。[论坛相关连接](#)

软件环境：ICC+AVRstudio，硬件环境atmega16 开发板，JTAG&ISP下载器。 进行之前请确认你已经装好开发软件环境：[软件环境选择](#) **【ICC+AVRstudio环境配置】**，也拥有硬件环境，[AVR 开发与入门芯片选择](#)。

## AVR jtag 在线仿真调试

第一部：硬件连接。

第二步：使用 AVRstudio 打开\*.cof 文件

如果你没有自己写AVR程序，你可以使用本站的新手入门第一个程序。[AVR\\_first](#)，实现红绿灯不断闪烁的例子。

程序启动时候的样子如下图：

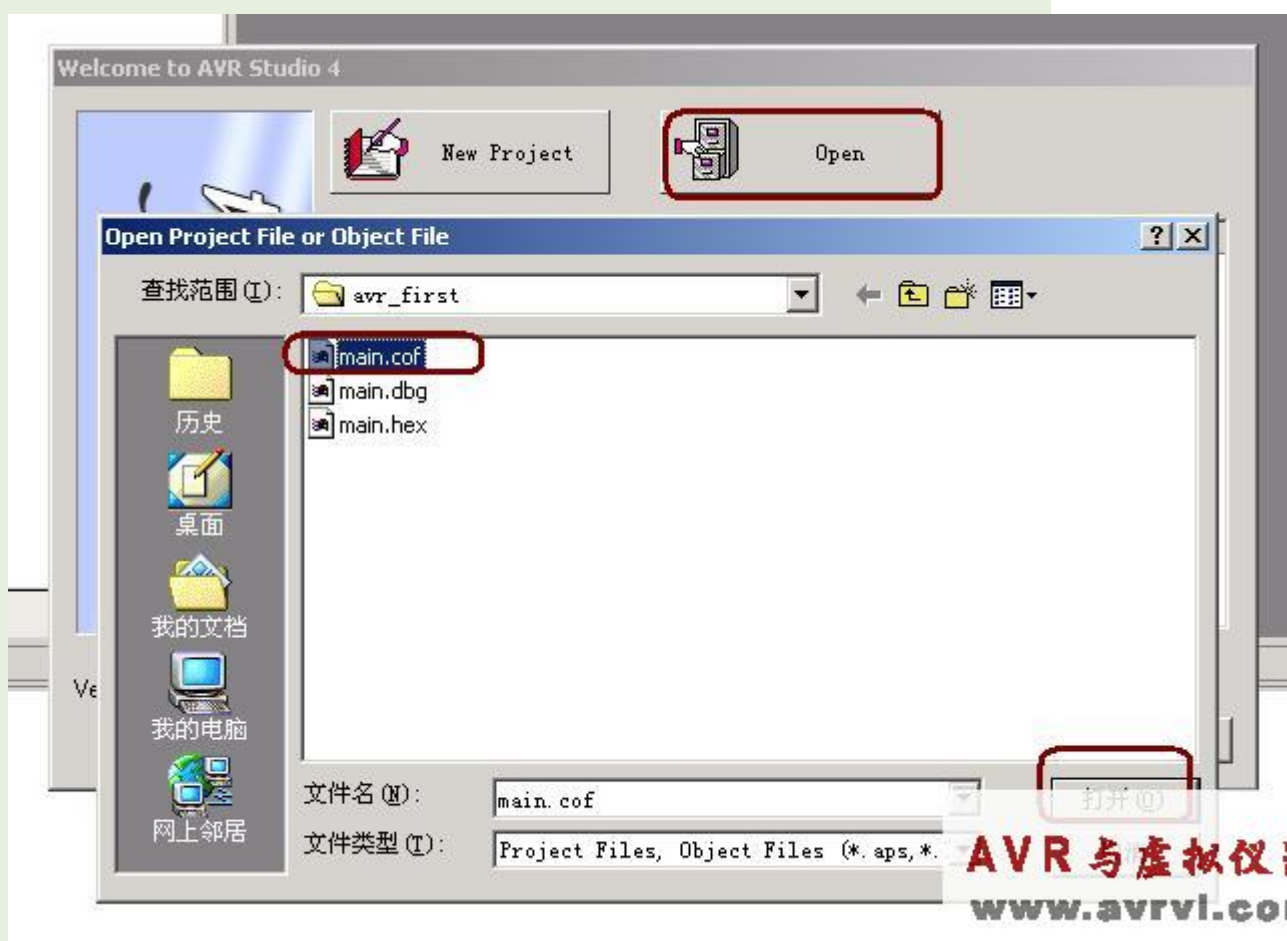


程序启动界面：





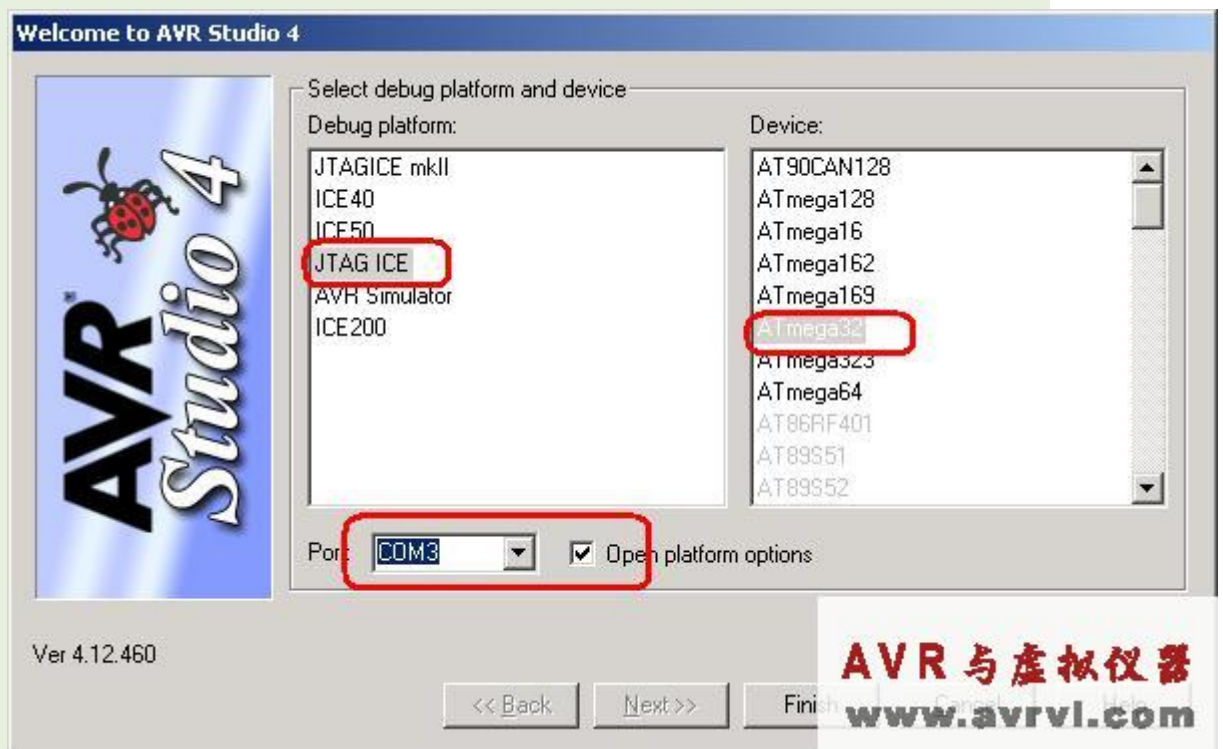
打开 main.cof 文件



工程文件存为 main\_cof.aps 方便下次打开

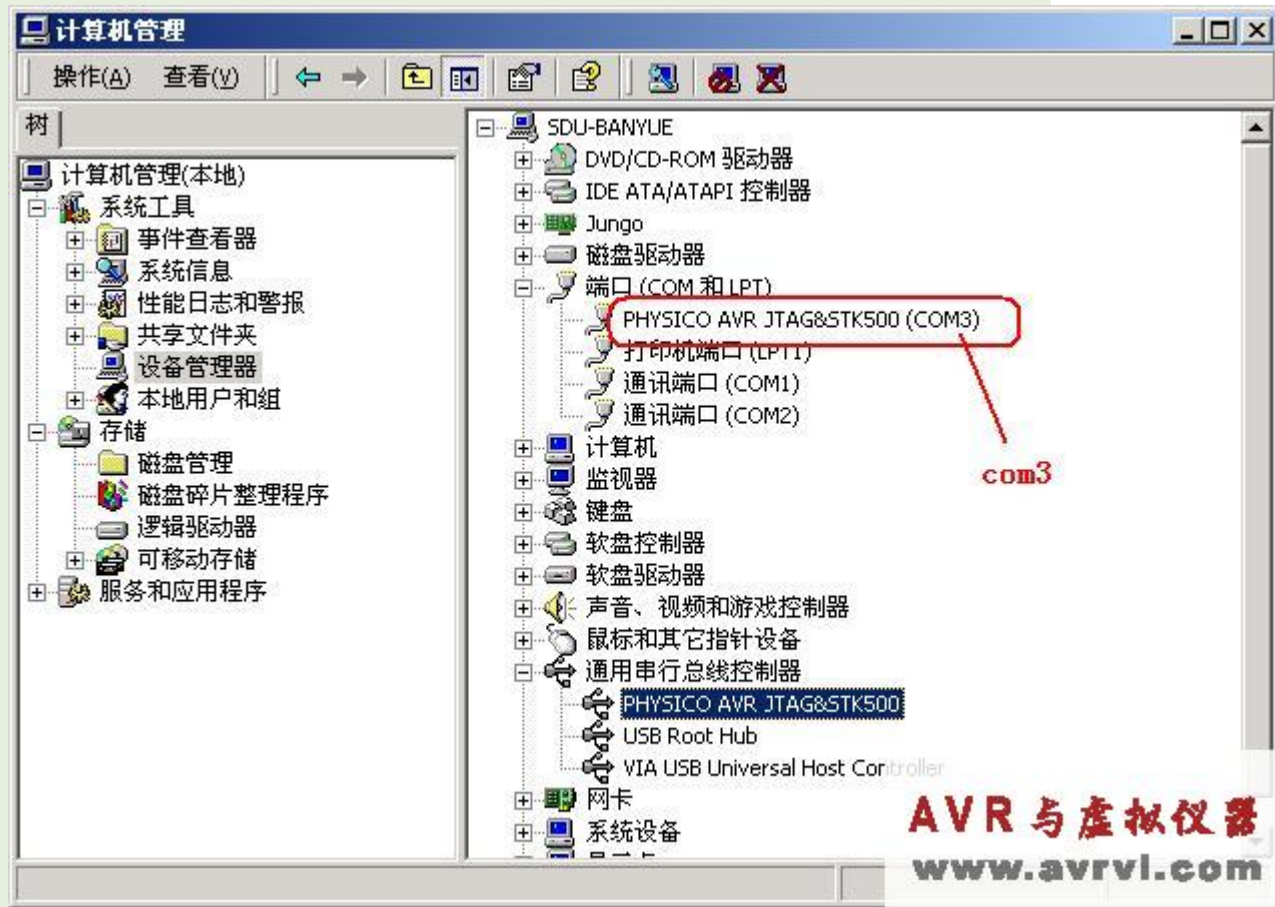


选择相关硬件配置



(端口的选择参见下图) 使用 JTAG&SIP 默认会是 COM3。

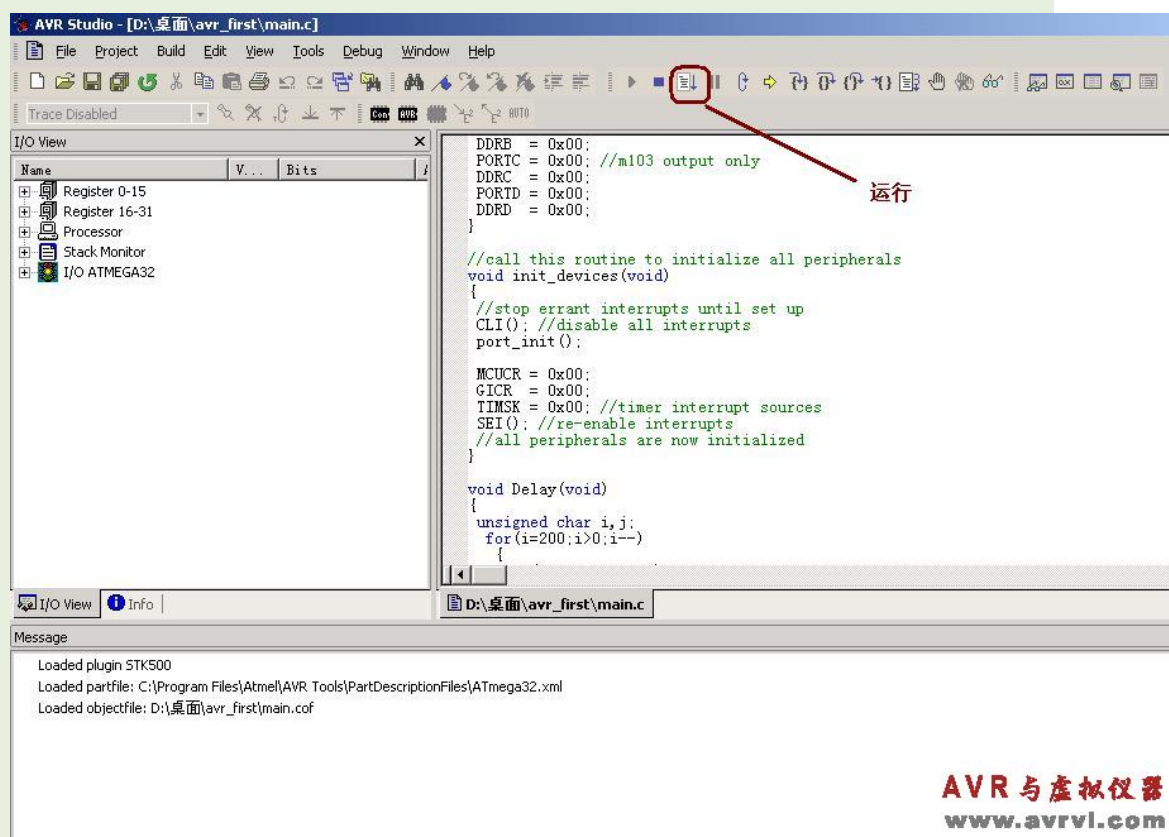
端口的选择（说明：这个画面可以在:右击我的电脑>--管理>--设备管理器里面找到。）



端口频率设置



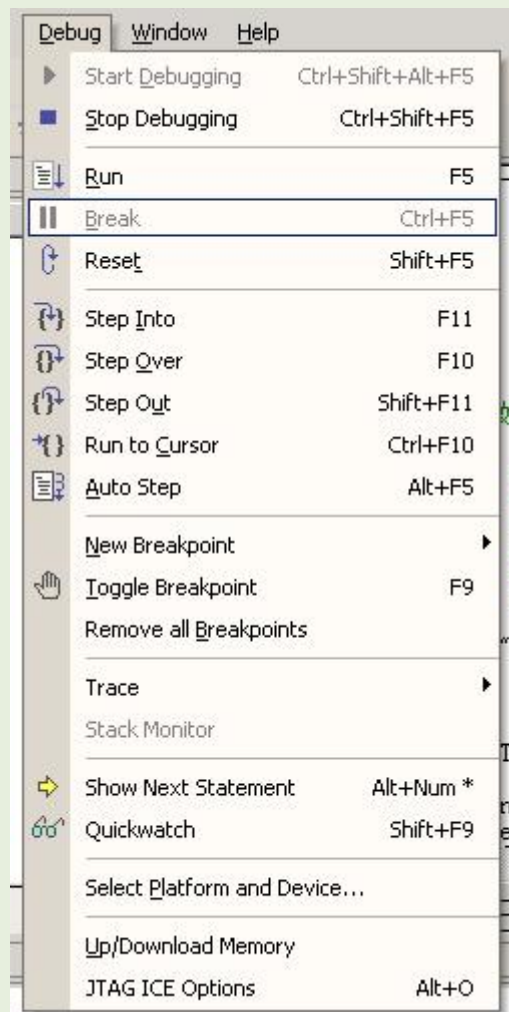
运行程序 直接运行你可以看到红绿灯闪动的效果。{点击看大图}



你还可以通过Debug里面的命令进行在线调试。好了，现在你已经进入AVR的精彩世界了。如果你在使用中遇到任何问题，欢迎在 [论坛上发帖参与讨论](#)。

### 三、调试命令的使用

调试快捷键，具体可以通过 **DEBUG** 菜单项查看，按照相应的指令就可以进行调试操作了。



观察窗口的说明：使用以下窗口可以即时查看寄存器，变量，以及数据地址的值，更多内容大家自己摸索吧。{点击看大图}





并口 ISP 下载线连接 PC 的并口和 AVR 的 ISP 接口。

一: Project>>Options>>Target

设置编译目标及选项。



二: Tools>>Environment Options.....>>ISP

设置 stk500.exe 的绝对路径, stk500.exe 在 avr studio 目录下面。

并口 ISP 下载线连接 PC 的并口和 AVR 的 ISP 接口。

一: Project>>Options>>Target

设置编译目标及选项。



二: Tools>>Environment Options.....>>ISP

设置 stk500.exe 的绝对路径, stk500.exe 在 avr studio 目录下面。

并口 ISP 下载线连接 PC 的并口和 AVR 的 ISP 接口。

一: Project>>Options>>Target

设置编译目标及选项。



二: Tools>>Environment Options.....>>ISP

设置 stk500.exe 的绝对路径, stk500.exe 在 avr studio 目录下面。



并口 ISP 下载线连接 PC 的并口和 AVR 的 ISP 接口。

一: Project>>Options>>Target

设置编译目标及选项。



二: Tools>>Environment Options.....>>ISP

设置 stk500.exe 的绝对路径, stk500.exe 在 avr studio 目录下面。

并口 ISP 下载线连接 PC 的并口和 AVR 的 ISP 接口。

一: Project>>Options>>Target

设置编译目标及选项。



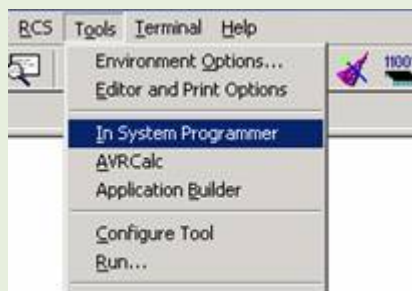
二: Tools>>Environment Options.....>>ISP

设置 stk500.exe 的绝对路径, stk500.exe 在 avr studio 目录下面。

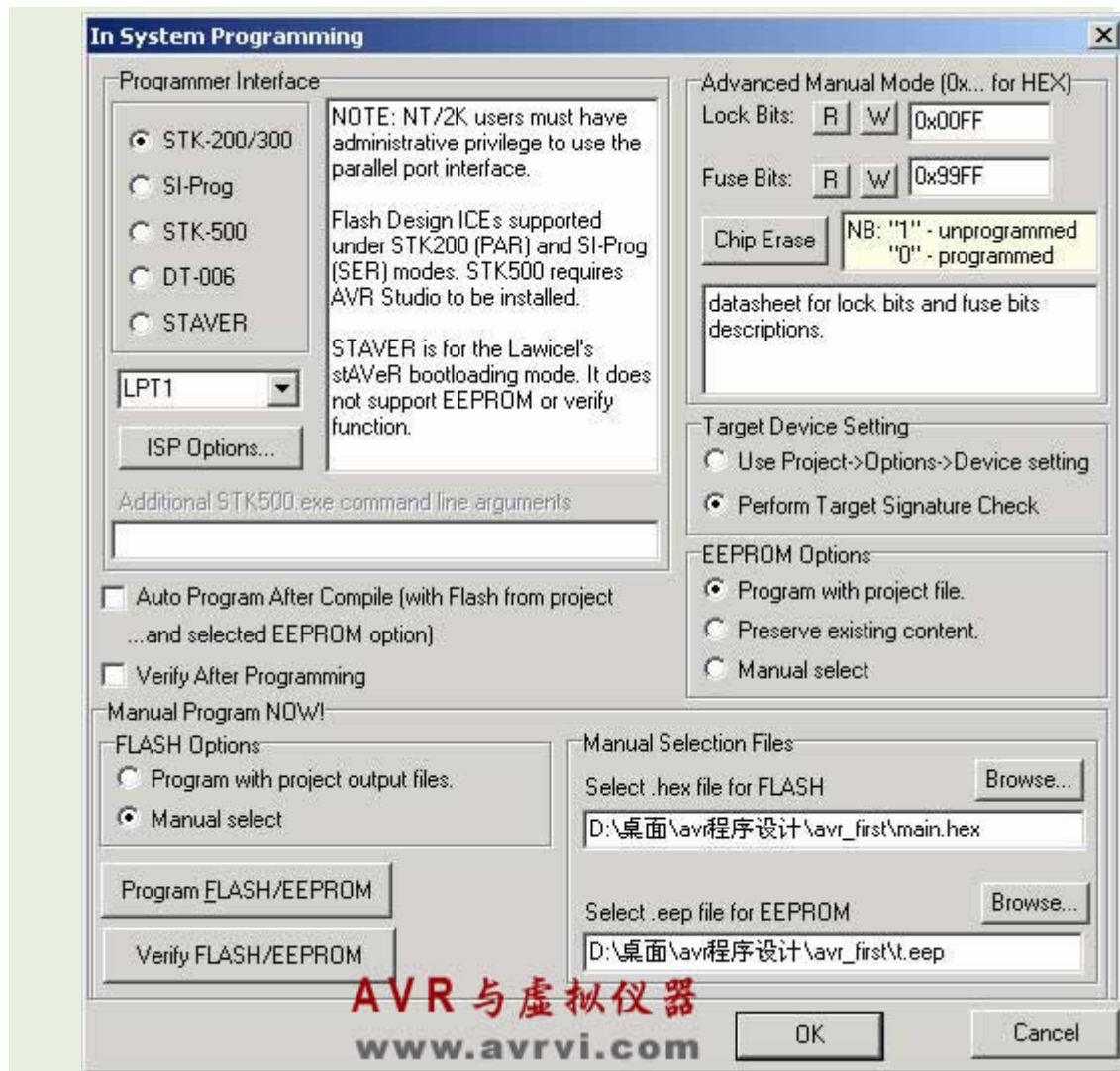


三: Tools>>In system programmer

最后使用软件进行下载, 可以 flash, eepro, 熔丝位, 锁定位进行操作。







## AVR 开发前准备—熔丝位(Fuse)快速入门

本页关键词：AVR 熔丝位(Fuse)快速入门 熔丝位 熔丝的作用

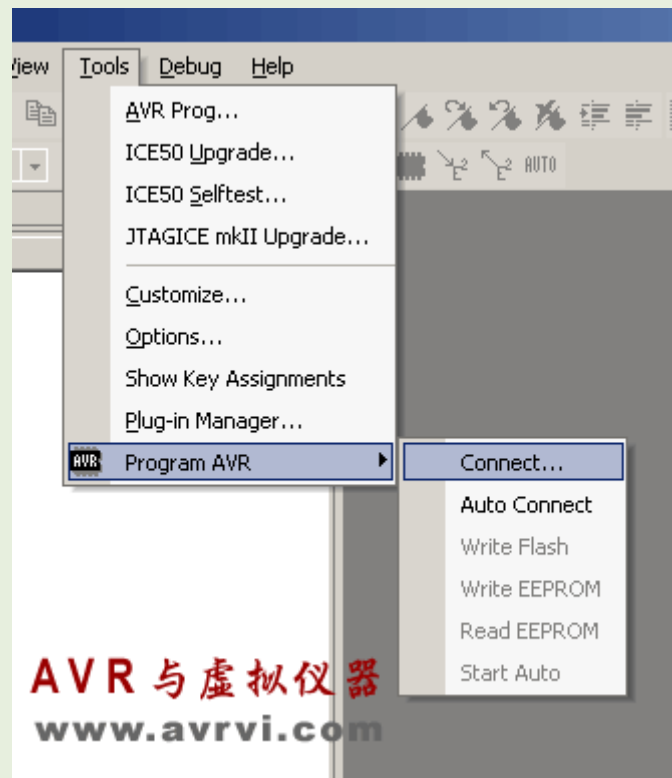
AVR 通过熔丝来控制芯片内部的一些功能，比如 JTAG，时钟的使用，掉电检测电压，是否允许调试等。

AVR Studio 中 STK500 处理熔丝位有巨大的优势：它是以功能组合让用户配置。这种方式与小马 (PnoyProg2000,SL-ISP)相比，具有以下的优势(优势是如此明显，可以用“巨大优势”来形容)：

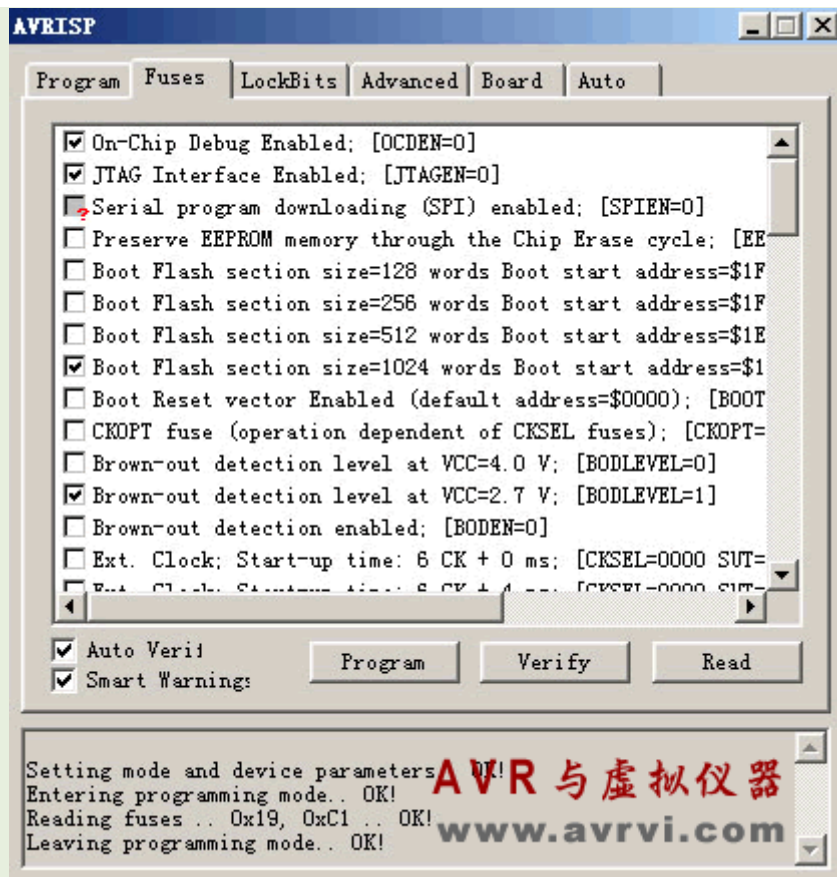
1. 有效避免因不熟悉熔丝位让芯片锁死 (这是初学者的噩梦)，笔者曾经锁死过三片 Atmega16。

2. 不需要靠记忆与查文档，就能配置熔丝位(这也是初学者的恶梦)
3. **动手之前**：请你一定弄清楚了，你这样改会有什么后果，除非你有很多钱不在乎多锁死几个芯片。备份你的熔丝位状态，在点击**Program**之前再次检查熔丝位设置正确与否，不要误点了某项而没有注意到。

通过下图的方法打开连接：



使用操作界面如下：（注意：下图中，打勾的表示选中，代表 0。没有打勾的表示 1）。



上图的资料有很多相关项，你需要认识以下的代码，以理解意思。英文翻译说明如下：

英文	中文
On-Chip Debug Enabled	片内 调试 使能
JTAG Interface Enabled	JTAG 接口 使能
Serial program downloading (SPI) enabled	串行编程下载(SPI) 使能 (ISP 下载时该位不能修改)
Preserve EEPROM memory through the Chip Erase cycle;	芯片擦除时 EEPROM 的内容保留
Boot Flash section size=xxxx words	引导(Boot)区大小为 xxx 个词
Boot start address=\$yyyy;	引导(Boot)区开始地址为 \$yyyy
Boot Reset vector Enabled	引导(Boot)、复位 向量 使能
Brown-out detection level at VCC=xxxx V;	掉电检测的电平为 VCC=xxxx 伏
Brown-out detection enabled;	掉电检测使能

Start-up time: xxx CK + yy ms	启动时间 xxx 个时钟周期 + yy 毫秒
Ext. Clock;	外部时钟
Int. RC Osc.	内部 RC(阻容) 振荡器
Ext. RC Osc.	外部 RC(阻容) 振荡器
Ext. Low-Freq. Crystal;	外部 低频 晶体
Ext. Crystal/Resonator Low Freq	外部晶体/陶瓷振荡器 低频
Ext. Crystal/Resonator Medium Freq	外部晶体/陶瓷振荡器 中频
Ext. Crystal/Resonator High Freq	外部晶体/陶瓷振荡器 高频

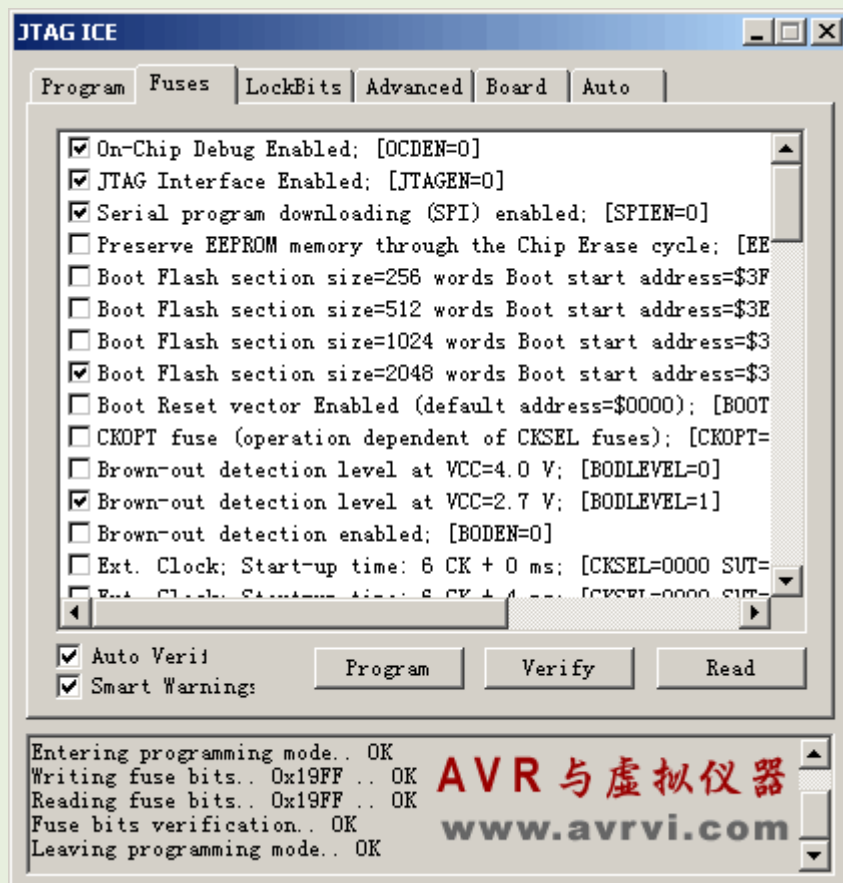
注：以上中文是对照 ATmega16 的中、英文版本数据手册而翻译。尽量按照了官方的中文术语。

应用举例：

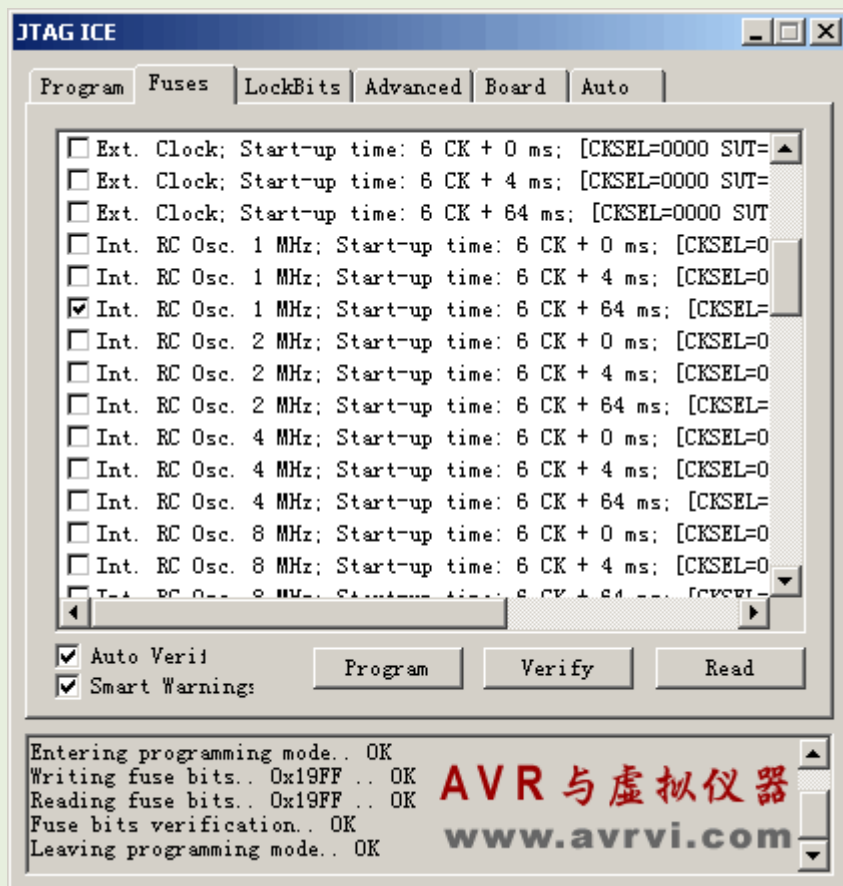
比如我们想使用片内的 RC 振荡(即不需要接晶振)，可以选择选择下面三者之一：

- Int. RC Osc. 8 MHz; Start-up time: 6 CK + 0 ms;
- [CKSEL=0100 SUT=00] Int. RC Osc. 8 MHz; Start-up time: 6 CK + 4 ms;
- [CKSEL=0100 SUT=01] Int. RC Osc. 8 MHz; Start-up time: 6 CK + 64 ms; [CKSEL=0100 SUT=10]

如图：内部 1M 晶振，默认情况典型设置。（两个图分别为上下两部分，没有显示的部分均为不选中状态。）



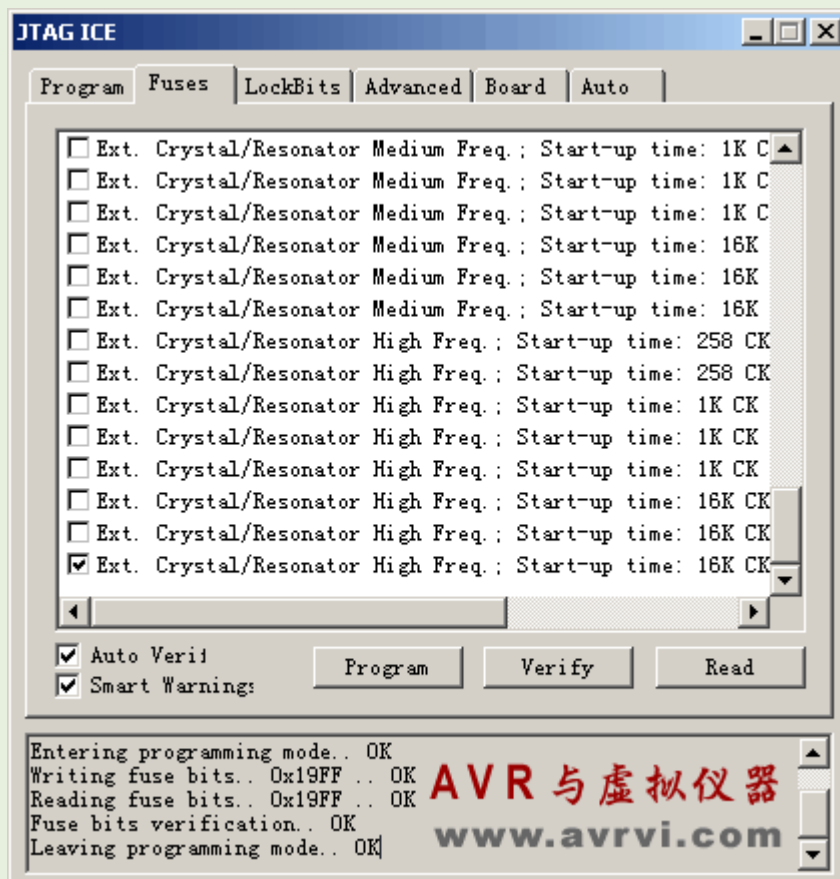
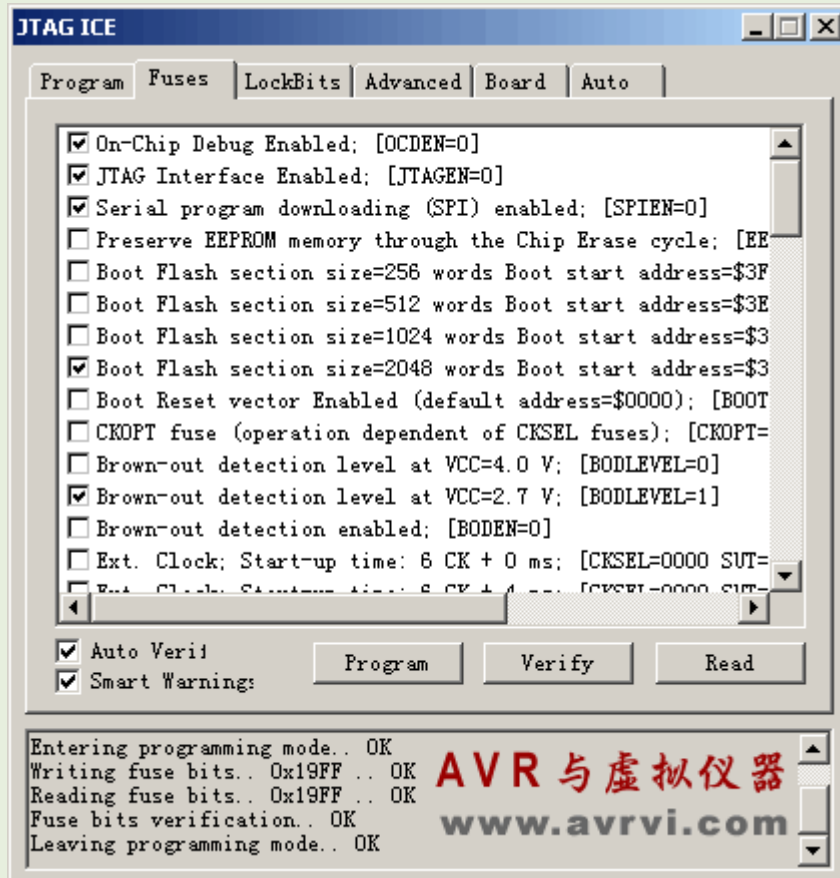
下图显示的是选择内部晶振，1 Mhz RC



比如我们想使用外部 7.3728M 晶振，可以选择下面三者之一：

- Ext. Crystal/Resonator High Freq.;
- Start-up time: 258 CK + 4 ms;
- [CKSEL=1110 SUT=00] 或后面与Ext. Crystal/Resonator High Freq.;.... 有关的选择。

如下两图：7.3728M晶振典型融丝位（及本站的开发板使用时候的典型设置）



如果你在使用过程中遇到什么问题，欢迎讨论，<http://bbs.avrvi.com>。

---

后记：说说 Mega128 的熔丝位

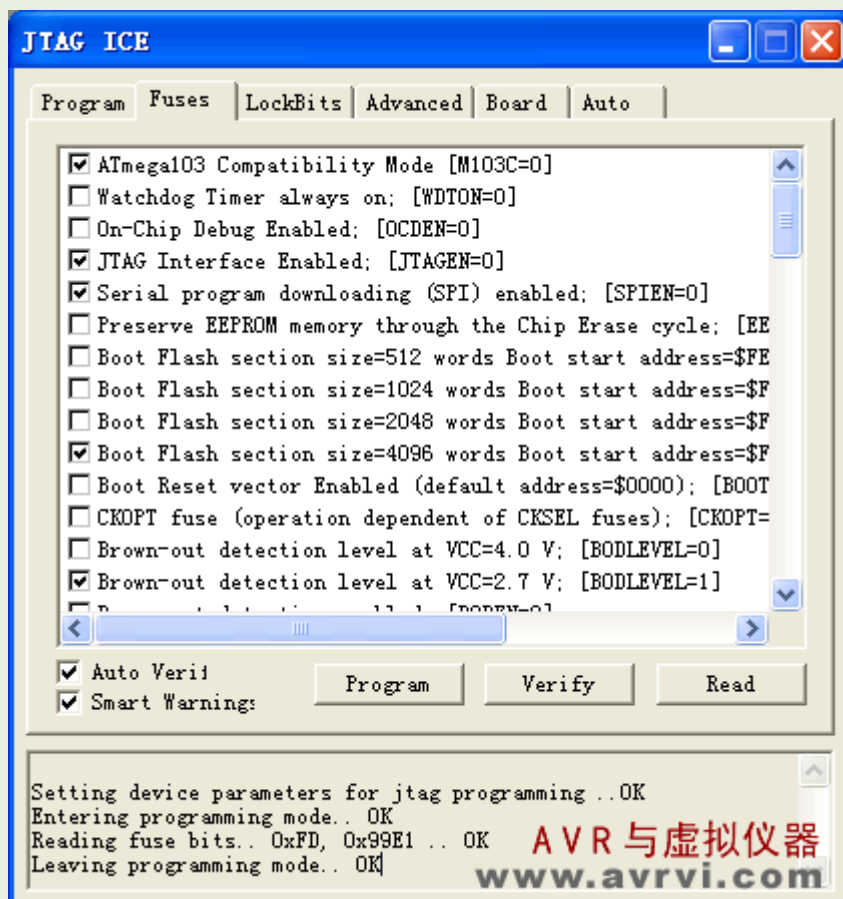
ATmega128 是 avr 系列中一款高性能的芯片，设计的时候兼容 M103 模式，但是这个 M103 模式经常害人。基于此，说说 ATmega128 的熔丝位，顺便说说其他的功能。

默认情况下 M103 模式是选中的，应该将其去掉；晶振是内部 1M 晶振，如果你使用外部晶振，应该进行修改。M128 可以开启硬件的看门狗，选中此项，看门狗不需要程序初始化，只需要程序里面喂狗就可以了。

#### 默认熔丝第一部分

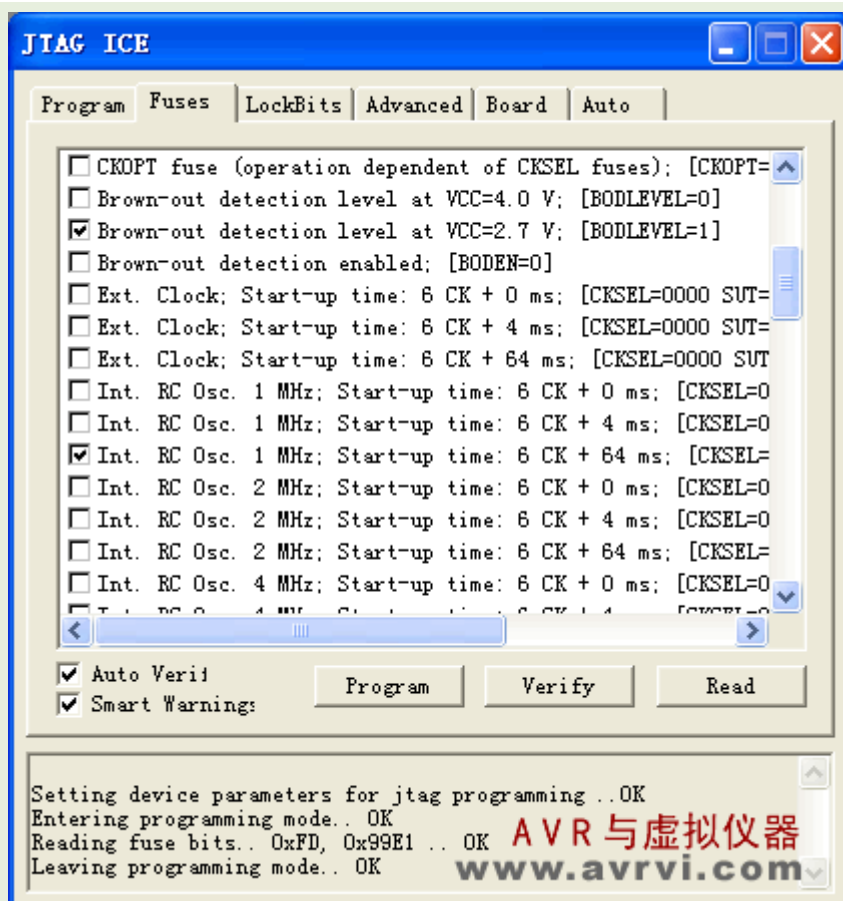
M103 兼容模式，使能 JTAG，使能 SPI，Bootloader 区大小 4096，未使能 BOOT。





默认熔丝第二部分

DOD 为 2.7V，内部 1M 晶振。



下面是本站使用 M128 开发板的典型设置，M103 模式取消，使用 M128 模式，使用外部 7.3728M 晶振。

#### 典型熔丝第一部分（只说修改部分）

去掉了 M103，从而使用 M128 模式。



### 典型熔丝第二部分

选择最后一项，即使用外部高频晶振。



## AVR 基本硬件线路与分析

单片机最小系统 单片机最小系统设计

AVR基本硬件线路设计与分析 (ATmega16 功能小板) AVR DB-CORE Ver2.3 Atmega16 开发板

本站商城提供本最小系统销售: [99 元 AVR学习套件](#) [AVR学习板](#) [AVR开发板](#) [easyavrm16](#) , [ATmega16 开发板](#) [AVR学习板](#) [Mega16 核心板](#) (特价)。

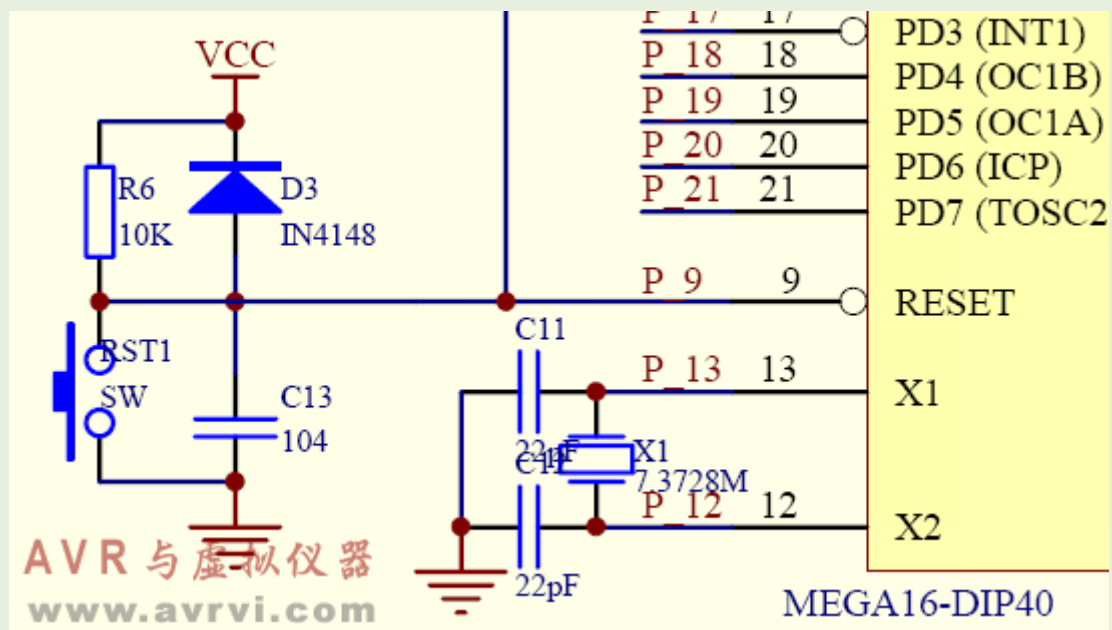
基本的 AVR 硬件线路, 包括以下几部分:

- 1. 复位线路
- 2. 晶振线路
- 3. AD 转换滤波线路

- 4. ISP 下载接口
- 5. JTAG 仿真接口
- 6. 电源
- 7. 串口电路

下面以本网站推荐的AVR入门芯片 ATmega16L-8AI 分析上述基本线路。(-8AI表示 8M频率的TQFP贴片封装，工业级，更详细的型号含义资料，请参考：[AVR芯片入门知识](#))

复位线路的设计（下图上面一部分）



Mega16 已经内置了上电复位设计。并且在熔丝位里，可以控制复位时的额外时间，故 AVR 外部的复位线路在上电时，可以设计得很简单：直接拉一只 10K 的电阻到 VCC 即可(R6)。

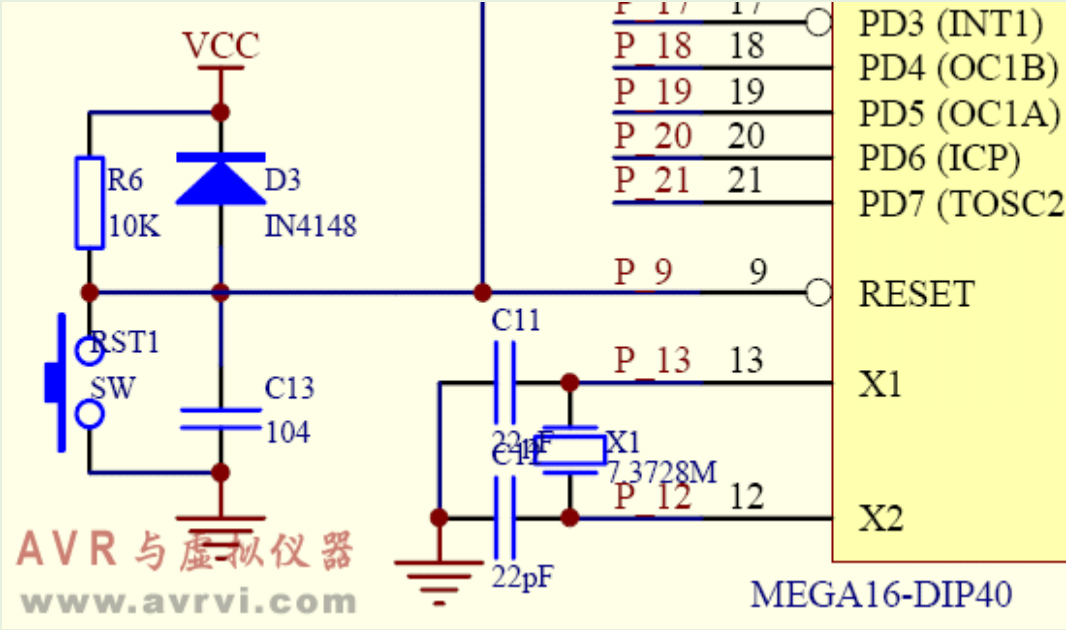
为了可靠，再加上一只 0.1uF 的电容(C13)以消除干扰、杂波。

D3(1N4148)的作用有两个：作用一是将复位输入的最高电压钳在  $V_{cc}+0.5V$  左右，另一作用是系统断电时，将 R0(10K)电阻短路，让 C0 快速放电，让下一次来电时，能产生有效的复位。

当 AVR 在工作时，按下 S0 开关时，复位脚变成低电平，触发 AVR 芯片复位。

重要说明：实际应用时，如果你不需要复位按钮，复位脚可以不接任何的零件，AVR 芯片也能稳定工作。即这部分不需要任何的外围零件。

晶振电路的设计（下图下面一部分）

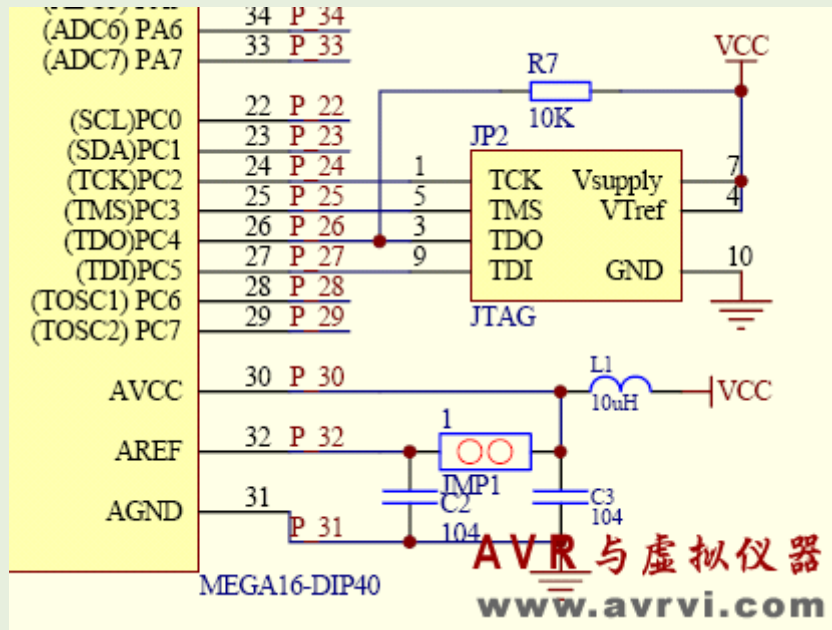


Mega16 已经内置 RC 振荡线路，可以产生 1M、2M、4M、8M 的振荡频率。不过，内置的毕竟是 RC 振荡，在一些要求较高的场合，比如要与 RS232 通信需要比较精确的波特率时，建议使用外部的晶振线路。

早期的 90S 系列，晶振两端均需要接 22pF 左右的电容。Mega 系列实际使用时，这两只小电容不接也能正常工作。不过为了线路的规范化，我们仍建议接上。

重要说明：实际应用时，如果你不需要太高精度的频率，可以使用内部 RC 振荡。即这部分不需要任何的外围零件。

AD 转换滤波线路的设计（下图下面部分）



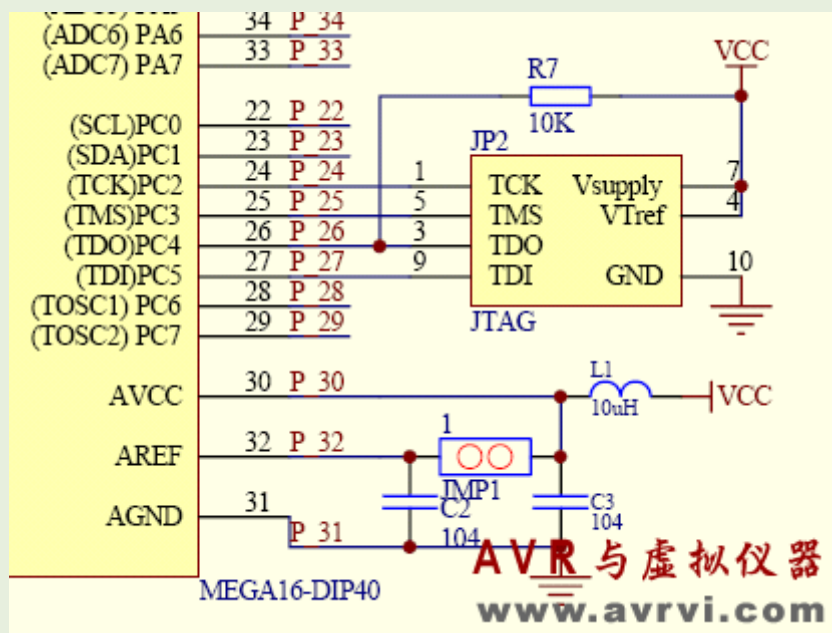
为减小 AD 转换的电源干扰，Mega16 芯片有独立的 AD 电源供电。官方文档推荐在 VCC 串上一只 10uH 的电感（L1），然后接一只 0.1uF 的电容到地（C3）。

Mega16 内带 2.56V 标准参考电压。也可以从外面输入参考电压，比如在外面使用 TL431 基准电压源。不过一般的应用使用内部自带的参考电压已经足够。习惯上在 AREF 脚接一只 0.1uF 的电容到地（C2）。

此处跳线 JMP1 为 AD 转换跳线，当你使用 AD 转换时，请连接，否则断开。

重要说明：实际应用时，如果你想简化线路，可以将 AVCC 直接接到 VCC，AREF 悬空。即这部分不需要任何的外围零件。

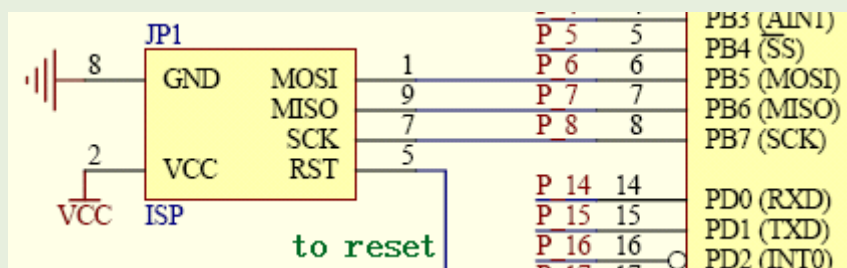
JTAG 仿真接口设计(下图上面部分)



仿真接口也是使用双排 2\*5 插座。需要一只 10K 的上拉电阻（R7）。

重要说明：实际应用时，如果你不想使用 JTAG 仿真，并且不想受四只 10K 的上拉电阻的影响，可以将 JP1—JP4 断开。

## ISP 下载接口设计



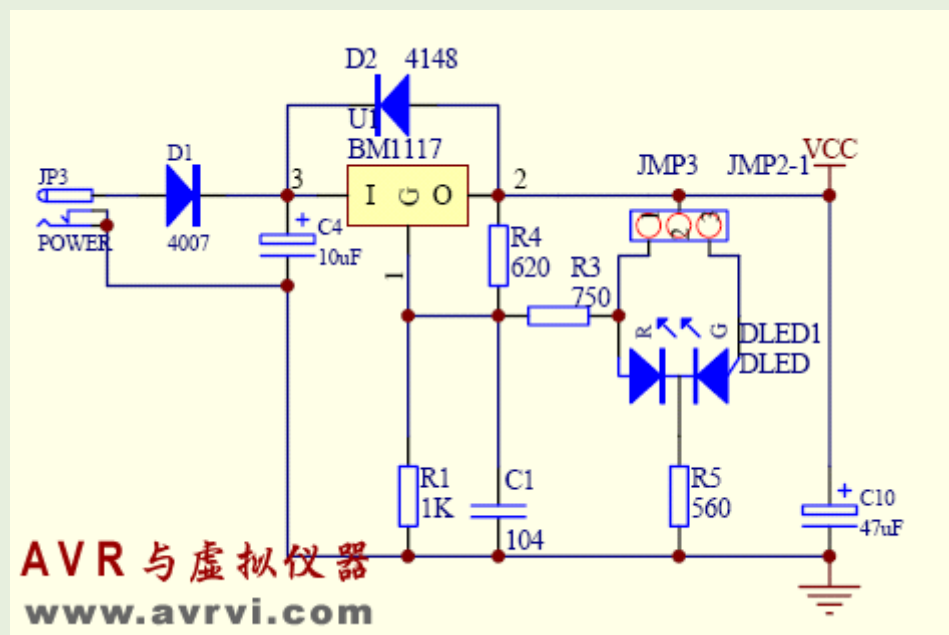
ISP 下载接口，不需要任何的外围零件。使用双排 2\*5 插座。由于没有外围零件，故 PB5（MOSI）、PB6（MISO）、PB7（SCK）、复位脚仍可以正常使用，不受 ISP 的干扰。



RST 连接倒 RESET （9），为了 减小图片大小这里没有画出，你可以从本页顶上的那个图片看出来。

重要说明：实际应用时，如果你想简化零件，可以不焊接 2×5 座。但在 PCB 设计时最好保留这个空位，以便以后升级 AVR 内的软件。

## 电源设计



AVR单片机最常用的是 5V 与 3.3V 两种电压。本线路以开关切换两种电压，并且以双色二极管指示（5V 时为绿灯，3.3V 时为红灯）。**JP3 输入电压为 7.5v—9v。**

二极管 D1 防止用户插错电源极性。D2 可以允许用户将电压倒灌入此电路内，不会损坏 BM1117。

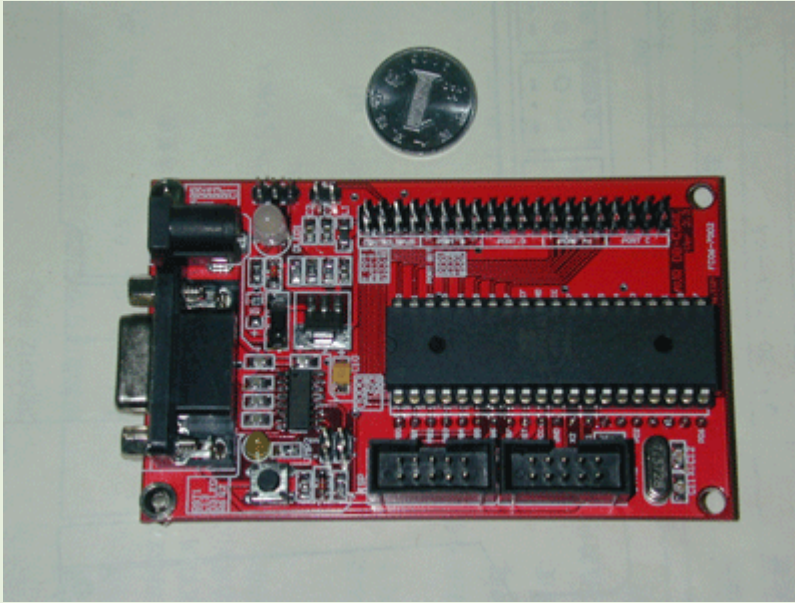
BM1117 的特性为 1 脚会有 50uA 的电流输出，1—2 脚会有 1.25V 电压。利用这个特点，可以计算出输出电压：

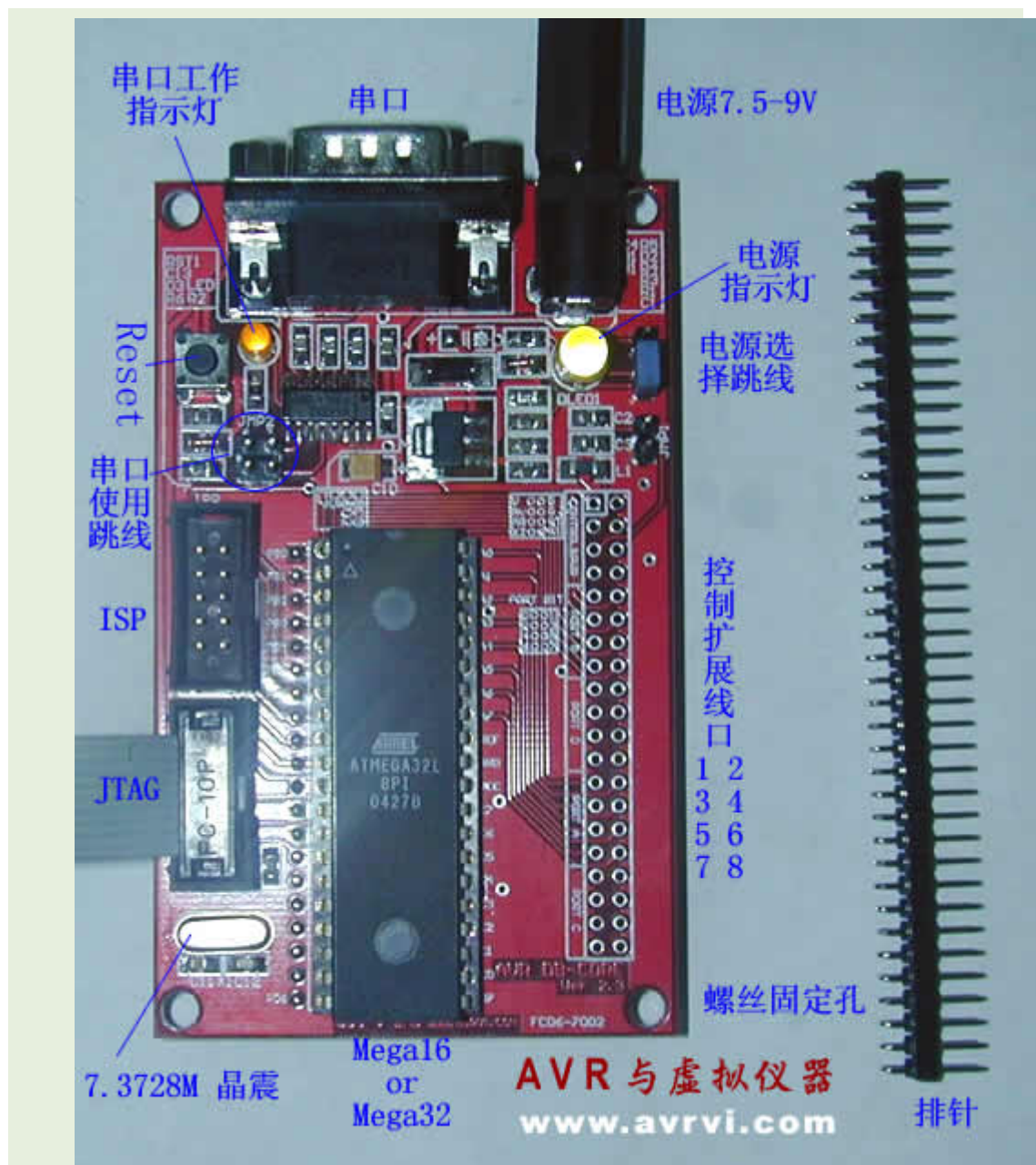
当 JMP3 开关打向左边时，R4 上的电流为  $1.25/0.33 = 3.78\text{ma}$ 。R1 上的电流为 BM1117 脚电流加上 R3 上的电流，即  $0.05+3.78=3.83\text{ma}$ 。可以计算得 R4 上的电压为 3.84V。于是得出  $VCC=1.25+3.83=5.08\text{V}$ 。误差在 2% 以内。

VCC 与 R2out 之间接串上一个电阻 R2 和一个发光二极管 LED1，特别说明，只有当此二极管闪的时候才说明串口在工作，直接接上的时候，此发光二极管也可能会亮。

## 总设计图

本站提供PDF和SCH文件原理图下载:【[PDF文件格式](#)】【[SCH文件格式](#)】,制作完成的AVR DB-CORE Ver2.3 Atmega16 如下图。





希望你设计出优秀的 AVR 电子产品，预祝你成功！

Avr 单片机和其他单片机一样，有自己的指令和寄存器，特定的 IO 口操作方法，有定时器，AD 转换，串行口，PWM 输出，EEPROM 等。为了缩短大家学习的时间，快速的了解 avr 编程思路，拟定 avr 程序设计系列教程。我使用 avr 也不是很长时间，技术有限，错误在所难免，还请指出，以期改正。

【说明】：本页是针对从来没有碰过单片机或者用过单片机而没有用 c 语言的朋友，如果你已经用过其他单片机，此页说明可能对你没有很大帮助，且显得有些幼稚，请你跳过。

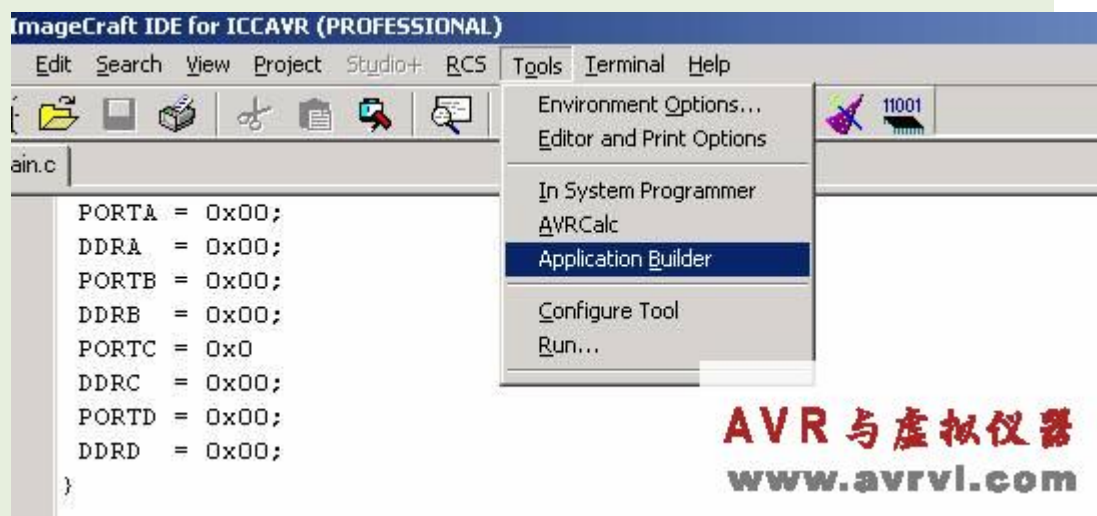
本站所有程序都是在icc+avrstudio下进行的，如果你没有配置好环境，请参看：[avr 开发环境配置](#)

### 【icc + AVRStudio】

#### icc 生成的第一个程序解释

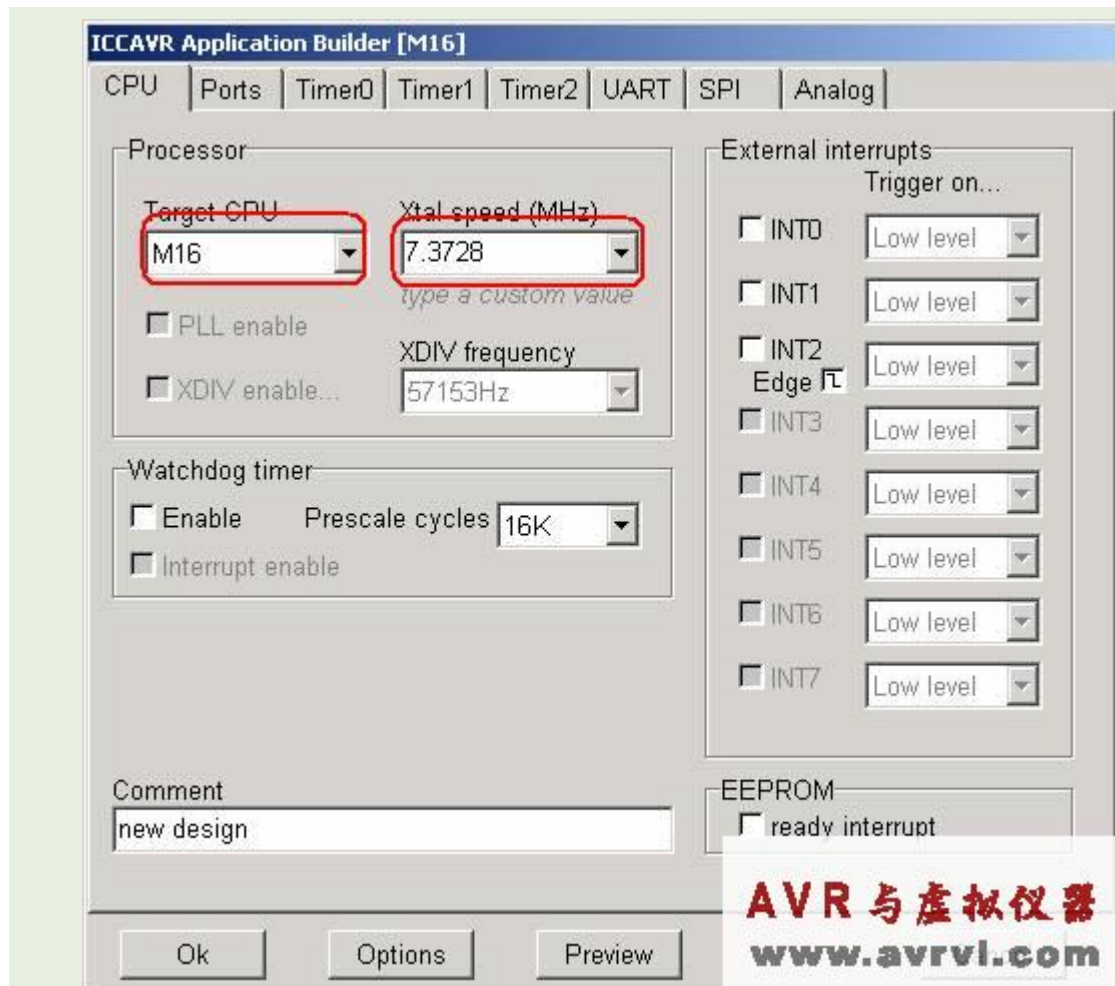
采用 icc 的原因之一是它自带一个非常好用的 application builder，生成的代码结构清晰，层次清楚，特别适合新手，如下图。

#### 非常好用的 icc application builder



下面通过一个简单的程序，分析 avr 程序要素，使用 icc 的程序生成器，Application Builder 界面如下：

#### icc 的程序生成器



进入该界面后，我更改了两项设置，Target CPU，及目标芯片设置位mega16 这个非常常用的芯片，Xtal speed 及晶振频率，我选了我用的 7.3728MHz。

下面我们来看看我们得到的代码。【说明】：“#”只是为了方便解释代码，并非为程序内容。

```
#1 //ICC-AVR application builder : 2006-11-3 14:42:54
```

```
#2 // Target : M16
```

```
#3 // Crystal: 7.3728Mhz
```

```
#4 #include <iom16v.h>
```

```
#5 #include <macros.h>
```



```

#6 void port_init(void)

#7 {

#8 PORTA = 0x00;

#9 DDRA = 0x00;

    PORTB = 0x00;

    DDRB = 0x00;

    PORTC = 0x00; //m103 output only

    DDRC = 0x00;

    PORTD = 0x00;

    DDRD = 0x00;

#10 }


#11 //call this routine to initialize all peripherals

#12 void init_devices(void)

#13 {

#14     //stop errant interrupts until set up

#15 CLI(); //disable all interrupts

#16 port_init();


#17 MCUCR = 0x00;

#18 GICR = 0x00;

#19 TIMSK = 0x00; //timer interrupt sources

#20 SEI(); //re-enable interrupts

#21 //all peripherals are now initialized

#13 }

```

#1——#3，注释行，分别说明了程序生成者是ICC-AVR application builder，生成时间，CPU类型，晶振大小，在本例中晶振大小没有用途，但在实际操作中，晶振大小将影响定时器的设置，PWM波形的输出，

串口波特率的设定，总线时序等多方面的问题。

**#4——#5**，包含特定头文件，学过c语言的人都知道，c语言里几乎每个程序都要包含特定的头文件，在这里，`iom16v.h`和`macros.h`都在`icc`程序安装文件夹下一个`include`的目录中，`iom16v.h`和你选择的芯片相对应，如果你选择的是`atmega32`，那个这个文件就是`iom32v.h`，等等，在这样的文件中定义了对应的芯片的各个硬件地址。`macros.h`文件中定义了一些宏命令和一些老的语言写法。通常每一个程序都要包含这个头文件。

**#6** `void port_init(void)`，定义函数`port_init`，函数的用途是初始化串口，前面一个`void`表示函数没有输出，括号里的`void`表示这个函数没有输入变量。

**#7，#10** C语言中所有函数体都要写在`{}`内，一个开始，一个结束。

**#8**，定义`PA`口为 `0x00`，及`PA`口上的所有管脚输出为低电平，同样道理，如果些微`PORTA=0xFF`，则都输入高电平，`0xFF`及二进制的 `11111111`，如果想让某一位输出高，其他为低，如 `00000001`，则是`PA0` 输出为高电平，`PA1——PA7` 输出为低电平。说明：每个口线与单片机实际引脚的对应关系请参看技术手册。

**#9**，定义引脚的输出输入模式，`1` 为输出，`0` 为输入。当使用输出模式及设置为 `1` 时，`PORTA`的值将影响内部上拉电阻的使用，如果`PORTA`对应管脚的值为 `1`，则使用内部上拉电阻，否则不采用。 除了这里说的“=”之外，管脚的控制还有其他算符，`|=`，`&=`，`^=`，`==`，请参考学习笔记：[avr](http://bbs.avrvi.com/read-htm-tid-45.html)端口操作的例子理解 <http://bbs.avrvi.com/read-htm-tid-45.html>，将在本页后面部分说明。

同理，下面部分为`BCD`口的操作。

**#11**，注释，说明你可以使用下面的程序来实现初始化。

**#13**，程序的开始和结束。

**#15**，内部定义的宏，关闭所有中断。

**#16**，调用端口初始化的程序。



#17, MCUCR = 0x00;电源管理及睡眠模式寄存器设置, 这里设为 0x00, 为空闲模式, 更多内容, 请参看 atmega16 技术手册。 [http://www.avrvi.com/down.php?file=datasheet/ATmega16\\_cn.pdf](http://www.avrvi.com/down.php?file=datasheet/ATmega16_cn.pdf)

#18, GICR = 0x00;通用中断控制寄存器设置, 0x00, 代表禁用任何中断, 更多内容请参考技术手册。

#19, TIMSK = 0x00;定时器设置, 0x00 代表不使用定时器。

#20, 与#15 对应, 内部定义的宏, 开中断。

#21, 注释, 所有功能初始化完成。

到此, 一个简单的程序就看完了, 本程序并不能真正运行, 因为没有main主函数。进一步的学习请看

【icc + AVRStudio】下的第一个程序【一】 <http://bbs.avrvi.com/read-htm-tid-2.html>

#### AVR 端口操作说明

一些端口操作的运算符总结, 我初学时总结出来的东西, 端口操作是单片机操作的基本要素。 以下给出不是完整的程序, 只是对端口操作的一些理解。

```
#include <iom32v.h>
```

```
void main(void)
```

```
{
```

```
PORTA=0xff; //在定义 DDRA 之前定义 PORTA 将影响上拉电阻的使用。1 为使用上拉电阻,0 为不使用。
```

```
DDRA=0xff; //输出 模式 , IO 口上, 1 为输出, 0 为输入。
```

```
PORTA=0xf0; //等
```

```
PORTA&=~0xf0; //清零
```

```
PORTA|=0x77; //置一
```

```
PORTA^=0x70; //翻转
```

```
(P & 0x80)==0x80; //按位与 判断 p 的第七位是否是一,是则成立
```

```
}
```

ADIF 就是 4 跟手册的为定义是一样的

```
(1<<ADIF)=(1<<4)=0b00010000
```

```
ADCSR=(1<<ADIF);    //只是 ADIF 位 =1,其他=0
```

```
ADCSR|=(1<<ADIF);    //只是 ADIF 位 =1,其他不变
```

```
ADCSR&=~(1<<ADIF);    //只是 ADIF 位 =0,其他不变
```

```
while(ADCSR&(1<<ADIF)); //等待 ADIF 位为 0，才退出循环，执行下一步
```

```
while(1)
```

```
{
```

```
while(ADCSR&(1<<ADIF)); //等待 ADIF 位为 0，才退出循环，执行下一步
```

```
{
```

```
程序.....
```

```
}
```

```
}
```

## AVR 移位算法详细解释（ $1 \ll X$ ）

很多初学者都会被移位算法迷惑，移位算法形如（ $1 \ll X$ ）这样的形式，高手写程序时，习惯用移位算法来写出各个寄存器的使用。比如下面一段是 AVR 的 USART 的初始化代码。

```
UCSRC = (1<<URSEL)|(1<<UCSZ1)|(1<<UCSZ0);
```

```
UCSR0B = (1<<RXCIE)|(1<<TXCIE)|(1<<RXEN)|(1<<TXEN); //
```

```
RXCIE=1;TXCIE=1;UDREIE=0;RXEN=1;TXEN=1
```

这样的写法对高手是福，这些代码里面说明了操作了寄存器的哪些位，能够看出它的操作的意义；对新手确是祸害，因为新手看不懂这样的程序。

---

回到开始的地方，解释一下，什么是移位算法：

如：A = (1<<2)，1 写成二进制就是 0000 0001，这个一左移 2 位就是 0000 0100，所以得到的数 A 为 0000 0100，即 0x04。

再如：B = (2<<4)，2 写成二进制就是 0000 0010，这个一左移 4 位就是 0010 0000，所以得到的数 B 为 0010 0000，即 0x20。

上面两个移位算法都是正确的，第一种写法，表示第三位为 1 其余都是 0 的数，数的时候是从 0 数起的，再比如 (1<<0) 表示的是 0000 0001，(1<<7) 表示的是 1000 0000，但是第二种写法没有没有这种意义，移位也用于乘除法，左移一位乘以 2，右移移位除以 2，上面的第二种写法 2 左移四位得到的数是  $2 \times 2 \times 2 \times 2 = 32$ ，也就是上面的 0x20。

---

我们再来看上面的这句话：UCSRC = (1<<URSEL)|(1<<UCSZ1)|(1<<UCSZ0);

UCSRC 是一个和串口通讯有关的一个八位寄存器，他的每一位都有特殊的定义，我们通过查数据手册可以看到，如下的内容。

## USART 控制和状态寄存器 C - UCSRC

AVR与虚拟仪器

[www.avrvi.com](http://www.avrvi.com)

Bit	7	6	5	4	3	2	1	0	
	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	UCSRC
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	1	0	0	0	0	1	1	0	

UCSRC寄存器与UBRRH寄存器共用相同的I/O地址。对该寄存器的访问,请参见P151“访问 UBRRH/ UCSRC 寄存器”。

### • Bit 7 – URSEL: 寄存器选择

通过该位选择访问 UCSRC 寄存器或 UBRRH 寄存器。当读 UCSRC 时,该位为 1 ;当写 UCSRC 时,URSEL 为 1。

### • Bit 6 – UMSEL: USART 模式选择

通过这一位来选择同步或异步工作模式。

我们在程序中包含的头文件 iom16v.h 类似的文件会有#define URSEL 7 这样的定义, 1<<URSEL 即是 wei7, 1<<UCSZ1 选择位 2, 1<<UCSZ0 选择位 1, 整句话 UCSRC = (1<<URSEL)|(1<<UCSZ1)|(1<<UCSZ0);的效果就是让 UCSRC 的位七, 位二, 位一为高, 其他都为低, 然后在数据手册里面你可以看到各个位的作用。UCSRC = (1<<7)|(1<<2)|(1<<1) 即 UCSRC = 1000 0110

朋友, 讲到这里, 你明白了吗? 如果还有疑问, 到论坛相关帖子看看吧, 问答式的内容可以帮你更深入的了解。<http://bbs.avrvi.com/read-htm-tid-897.html>

## AVR c 语言优秀编程风格

作为一个初学者如何具有良好的程序设计风格呢? 我想引用一个关于初学者请教编程大师的故事让读者自己去领悟。

有一位编程大师, 他写非结构化的程序, 一位初学者刻意模仿他, 也写非结构化的程序。当他让大师看他的进步时, 大师批评了他的非结构化程序: “对一位编程大师合适的东西未必对一个初学者同样合适, 在超越结构化之前, 你必须理解编程之道。” 我个人认为作为一个初学者应该踏踏实实的打好程序设计的基础, 不要急功近利, 舍本逐末。我走过不少弯路, 希望大家能和我一样能牢记编程大师的忠告: “对编程大师合适的东西未必对一个初学者同样合适”。

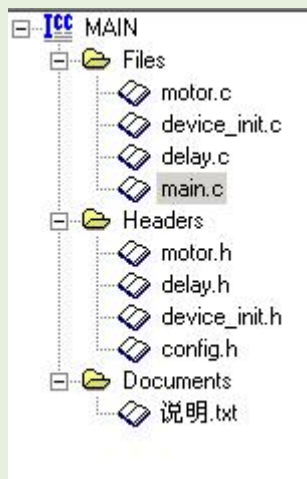
本文所描述的优秀编程风格适合于大部分语言，文章中可能提到你不是很了解的概念，没有关系，你放心的读下去，当你使用 AVR 一个月之后，你什么都明白了。

## AVR c 语言优秀编程风格

### 文件结构

模块化的程序应该是有一个很好的程序结构的。AVR C 语言程序有两种用户文件，.c 程序文件，.h 头文件，程序中编写过程中需要在.c 文件中包含.h 头文件。初学者往往出现重复包含或者头文件包含错误的问题，我当时也时常为这种错误而发愁。下面我以我写的电机驱动例程来给大家说明一下，优秀的编程文件结构。

这个工程中有 8 个文件，一个说明文件，如下图：下载程序例子 [电机控制案例](#)。



我写的成型的程序的文件个数基本上都是偶数，因为每一个结构化的函数定义.c 文件都会对应一个.h 文件。main.c 对应 config.h。我们来看看各文件的包含关系。下面我们看看这些文件的包含关系与内容：[推荐的文件包含顺序与关系]

- 所有.c 文件都包含了 config.h 文件。如： `#include "config.h"`
- 在 config.h 中有如下代码：
- `#include "delay.h"`
- `#include "device_init.h"`
- `#include "motor.h"`

- 这样做就不容易出现错误的包含关系，为了预防万一，我们还引入了宏定义与预编译。如下：
- `#ifndef _UNIT_H__`
- `#define _UNIT_H__ 1`
- `//100us`
- `extern void Delay100us(uint8 n);`
- `//1s`
- `extern void Delay1s(uint16 n); // n <= 6 ,when n==7, it is 1.`
- `//1ms`
- `extern void Delay1ms(uint16 n);`
- 
- `#endif`
- 
- 第一次包含本文件的时候正确编译，并且`#define _UNIT_H__ 1`，第二次包含本文件`#ifndef _UNIT_H__`就不再成立，跳过文件。
- 预编译还有更多的用途，比如可以根据不同的值编译不同的语句，如下：
- `//#pragma REGPARMS`
- `#if CPU_TYPE == M128`
- `#include <iom128v.h>`
- `#endif`
- `#if CPU_TYPE == M64`
- `#include <iom64v.h>`
- `#endif`
- `#if CPU_TYPE == M32`
- `#include <iom32v.h>`
- `#endif`
- `#if CPU_TYPE == M16`
- `#include <iom16v.h>`
- `#endif`
- `#if CPU_TYPE == M8`

- `#include <iom8v.h>`
- `#endif`
- `#include<filename>` 与 `#include "filename"` 的区别：前者是包含系统目录 `include` 下的文件，后者是包含程序目录下的文件。

### 变量名与函数名

变量以及函数命名应该按照 **尽量短**，**按需长**，**具有实际意义**。可以通过下划线或者大小写结合的方法组合动词和名词组成变量函数名。下面对比好的命名方法与不好的命名方法：

1. 好的：`Delay100us();`  
不好的：`Yanshi();`
2. 好的：`init_devices();`  
不好的：`Chengxuchushihua();`
3. 好的：`int temp;`  
不好的：`int dd;`

### 外部调用

1. 首先在模块化程序的.h文件中定义 **extern**
2. `//端口初始化`
3. `extern void port_init(void);`
- 4.
5. `//T2 初始化`
6. `void timer2_init(void);`
- 7.
8. `//各种参数初始化`  
`extern void init_devices(void);`

9. 模块化程序的.c文件中定义函数，**不要在模块化的程序中调用程序**，及不要出现向`timer2_init();`这样函数的使用，因为你以后不知道你到底什么地方调用了函数，导致程序调试难度增加。可以在定义函数的过程中调用其他函数作为函数体。

```

10. /*****采用 timer2 产生波形
    *****/
11. // PWM 频率 = 系统时钟频率/（分频系数*2*计数器上限值）
12. void timer2_init(void)
13. {
14.     TCCR2 = 0x00; //stop
15.     TCNT2 = 0x01; //set count
16.     OCR2 = 0x66; //set compare
17.     TCCR2 = (1<<WGM20) | (1<<WGM21) | (1<<COM21) | 0x06; // start timer
    快速 pwm 模式，匹配清零，溢出置位 256 分频
18. //占空比=高比低为：(OCR2-0x01)/(0xFF-OCR2)
    0x01++++++(OCR2)_____0xFF (+表示输出高，_表示输出低)
19. //即 OCR2 越大，输出越大
    }

```

20. 在少数几个文件中调用函数，在 **main.c** 中调用大部分函数，在 **interrupts.c** 中根据不同的中断调用服务函数。

```

21. void main(void)
22. {
23.
24. /*****
    *****/
25. //初始工作
26. /*****
    *****/
27.     init_devices();
28.
29.     while(1)
30.     {
31.         for_ward(0); //默认速度运转 正

```



```

32. Delay1s(5);           //延时 5s
33. motor_stop();        //停止
34. Delay1s(5);           //延时 5s
35. back_ward(0);         //默认速度运转 反
36. Delay1s(5);           //延时 5s
37. speed_add(20);        //加速
38. Delay1s(5);           //延时 5s
39. speed_subtract(20);   //减速
40. Delay1s(5);           //延时 5s
41. }
42.
    }

```

### 宏定义

宏定义主要用于两个地方：

1. 一是用得非常多的命令或语句，利用宏将其简化。
2. #ifndef TRUE
3. #define TRUE 1
4. #endif
5. #ifndef FALSE
6. #define FALSE 0
7. #endif
8. #ifndef NULL
9. #define NULL 0
10. #endif
11. #define MIN(a, b) ((a < b) ? (a) : (b))
12. #define MAX(a, b) ((a > b) ? (a) : (b))
13. #define ABS(x) ((x > 0) ? (x) : (-x))

```

14. typedef unsigned char  uint8;
    /* 定义可移植的无符号 8 位整数关键字          */
15. typedef signed   char  int8;
    /* 定义可移植的有符号 8 位整数关键字          */
16. typedef unsigned int   uint16;
    /* 定义可移植的无符号 16 位整数关键字         */
17. typedef signed   int   int16;
    /* 定义可移植的有符号 16 位整数关键字         */
18. typedef unsigned long  uint32;
    /* 定义可移植的无符号 32 位整数关键字         */
19. typedef signed   long  int32;
    /* 定义可移植的有符号 32 位整数关键字         */

20. 二是利用宏定义方便的进行硬件接口操作，再程序需要修改时，只需要修改宏定义即可，而不需要满篇去找命令行，进行修改。

21. //PD4, PD5 电机方向控制 如果更改管脚控制电机方向，更改PORTD |=
    0x10 即可。

22. #define moto_en1 PORTD |= 0x10
23. #define moto_en2 PORTD |= 0x20
24. #define moto_uen1 PORTD &=~ 0x10
25. #define moto_uen2 PORTD &=~ 0x20
26. //启动 TC2 定时比较和溢出
27. #define TC2_EN TIMSK |= (<<10CIE2) | (1<<TOIE2)
28. //禁止 TC2 再定时比较和溢出
    #define TC2_DIS TIMSK &=~ (1<<OCIE2) | (1<<TOIE2)

```

#### 关于注释

为了增加程序的可读性，方便合作者读动程序，或者程序作者在一段时间之后还能看懂程序，我们需要在程序中写 注释。

1. 在比较特殊的函数使用或者命令调用的地方加单行注释。使用方法为：

```

2. Tbuf_putchar(c, RTbuf);          // 将数据加入到发送缓冲区并开
   中断
   extern void Delay1s(uint16 n); // n <= 6 ,when n==7, it is 1.

```

```

3. 在模块化的函数中使用详细段落注释:
4. /*****
5. ** 函数名称: Com_putchar
6. ** 功能描述: 从串行口输出一个字符 c
7. ** 输 入: c:输出字符
8. ** 输出   : 0:失败 1:成功
9. ** 全局变量: 无
10.** 调用模块:
11.** 说明:
12.** 注意:
   *****/

```

```

13. 在文件头上加文件名, 文件用途, 作者, 日期等信息。
14. /*****
   *****/
15.**          serial   driver
16.**          (c) Copyright 2005-2006, limaokui
17.**          All Rights Reserved
18.**
19.**          V1.1.0
20.**
21.**
22.**-----文件信息
   -----
   -----
23.**文    件    名:sio.c

```

24.\*\*创建人：李茂奎

25.\*\*最后修改日期：2005 年 7 月 13 日

26.\*\*描述：serial driver

27.\*\*

28.\*\*-----历史版本信息

-----  
-----

29.\*\* 创建人：李茂奎

30.\*\* 版本：V1.00

31.\*\* 日期：2005 年 7 月 13 日

32.\*\* 描述：原始版本

33.\*\*

\*\*\*\*\*  
\*\*\*\*\*/

要清楚，注释是为了方便阅读，增强程序的可读性，不要本末倒置，不要给很简单大家都能看明白的程序加注释，不要让注释淹没了你的程序结构。对于函数，变量等尽量使用文件名自注释的方法，及通过文件名就可以知道意思。

本文结束了，新手教程也结束了，希望我们教程能让你轻松进入 AVR 的世界。

[AVR与虚拟仪器网站](#)全体工作人员谢谢你对本站的支持，谢谢你光临本站。

## AVR 使用范例--AVR 软件延时精确计算指导

和软件延时时间长短有关的因素有，单片机，晶振，延时语句，此处以 for 循环语句为例。

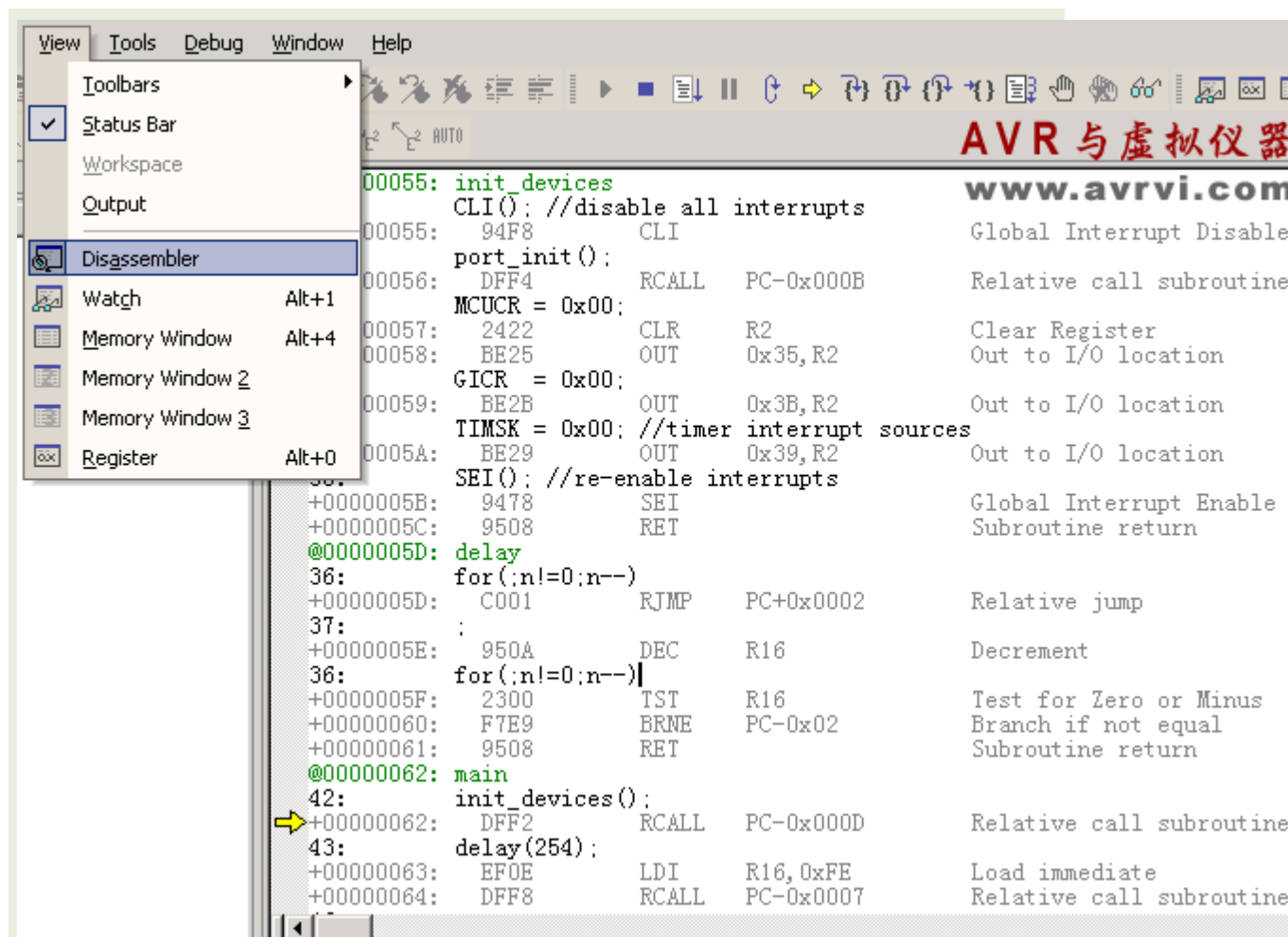
首先，我们编写一个for循环的延时语句，如下：非关键代码省略，[点击查看全部代码](#)。

```
void delay(unsigned char n)
```

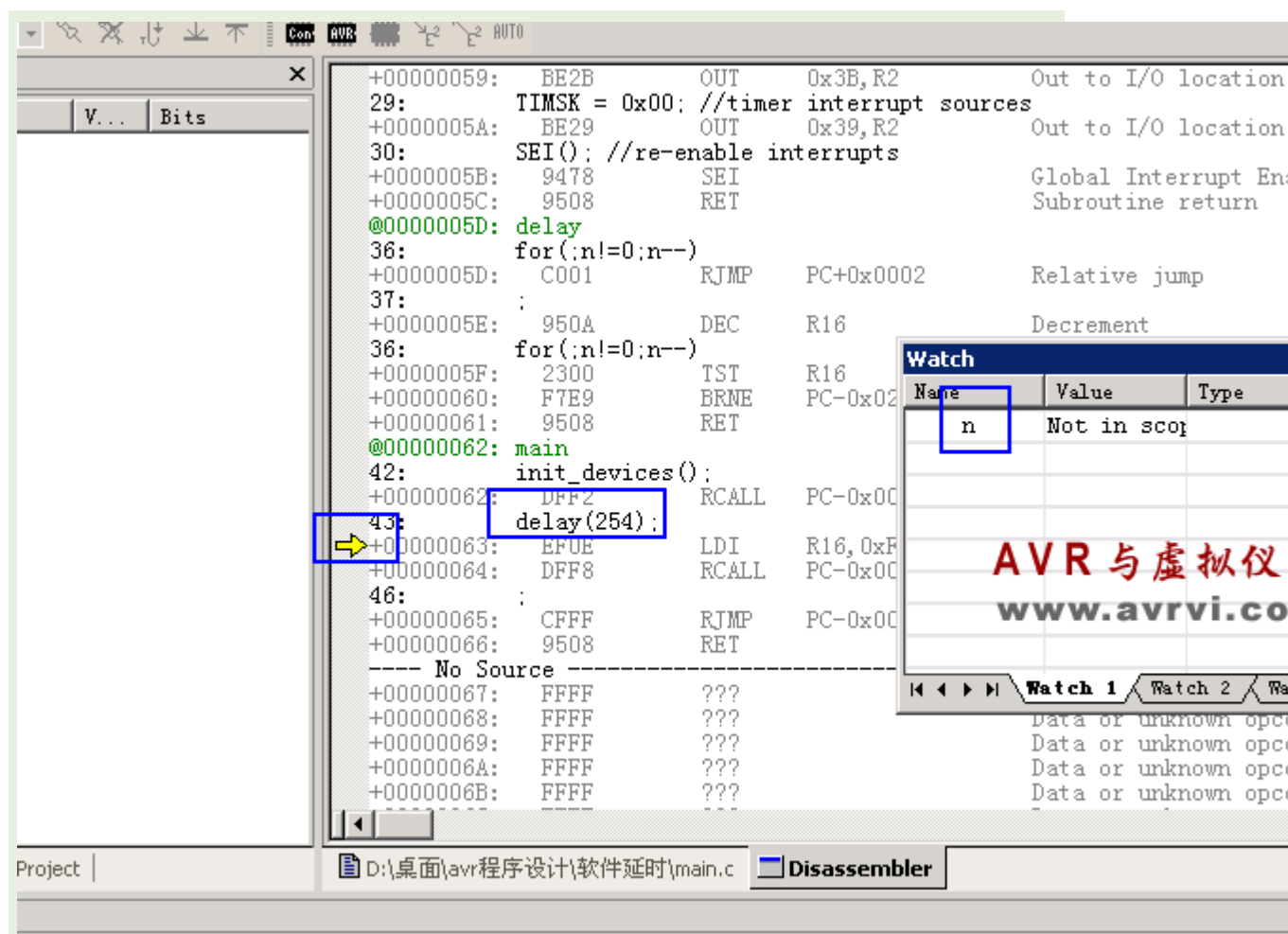
```
{  
    for(;n!=0;n--)  
    ;  
}  
  
void main(void)  
{  
    init_devices();  
    delay(254);/*计算结果,本条语句延时约 138 微秒,avr studio仿真结果延时  
141 微妙,以仿真的为准。*/  
  
    while(1)  
    ;  
}
```

正常编译,按照常规方法打开 **JTAG** 下载并进入调试。我们要想办法获取程序的运行指令个数。

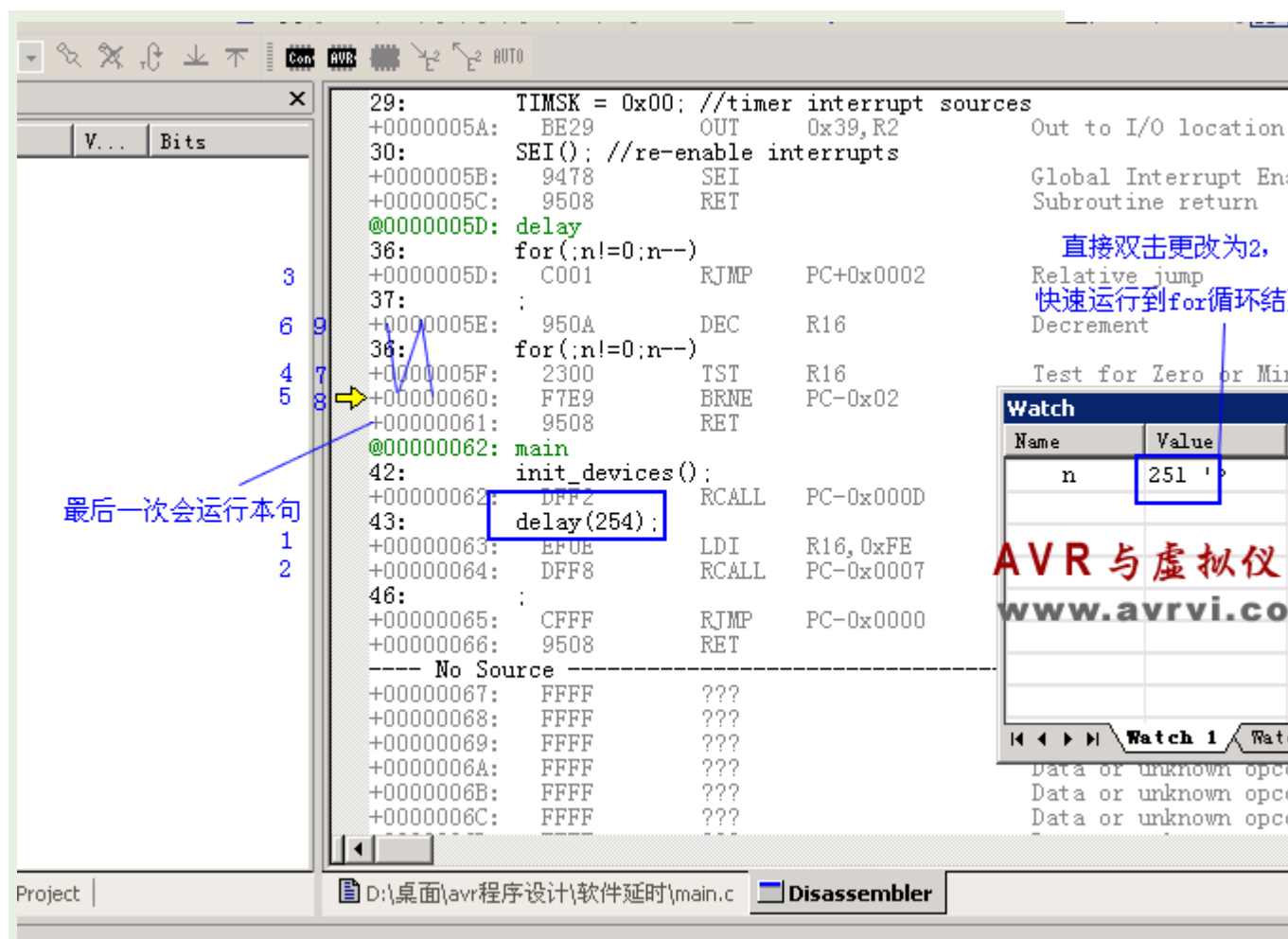
按下图操作调出汇编程序框:



打开 watch 窗口，找到 `delay(254);`，使用常规调试方法 F10, F11，使当前光标指向 `delay(254);` 的下一行，如下图：



使用 F11 逐条语句执行，你会看到如下图的运行规律，按照 1234567 的方向运动，最后循环，这就是我们想要找的执行语句条数，同时注意观察 watch 窗口的数值变化。通过更改 watch 窗口的数值，使循环结构快速结束。



我们由此得到语句的条数是  $3 + 3 \times (n + 1)$ ，这里是  $3 + 254 \times (3 + 1) = 1020$  条。在普通的计算中，我们可以这样认为，for 循环的语句数量是  $n \times 4 + 4$ 。

AVR 多数指令的执行时间是晶振频率分之一，也就是一个时钟周期，部分指令的时钟周期是 2-4 个时钟周期，详细内容请查看数据手册。那么 `delay(254);` 的总运行时间 1020 个时钟周期，即为  $1020 / (7.3728 \times 1000000)$  秒，约和  $1020 / 7.3728 = 138$  微秒。在要求不高的延时中，就可以使用 for 循环来多次调用这个 `delay` 作为 100 微秒使用，而不用考虑外层 for 循环造成的时钟周期延时。

结语：这里只是给出了一个软件延时的简单例子，并不具有很强的使用性，实际操作中可以定义 `delay100us`，`delay1ms`，`delay1s` 等函数直接使用。

```

/*****
*****

```

延时 M32 7.3728M 粗略计算



```

*/
void Delay100us(uint8 x)
{
    uint8 i;          //4clock
    for(i=147;x!=0;x--)
        while(--i);    //5 * i clock
}

void Delay1ms(uint16 n)
{
    for (;n!=0;n--) {
        Delay100us(10);
    }
}

void Delay1s(uint16 m)          // m <= 6 ,when m==7, it is 1.
{
    m=m*40;
    for (;m!=0;m--) {
        Delay100us(250);
    }
}

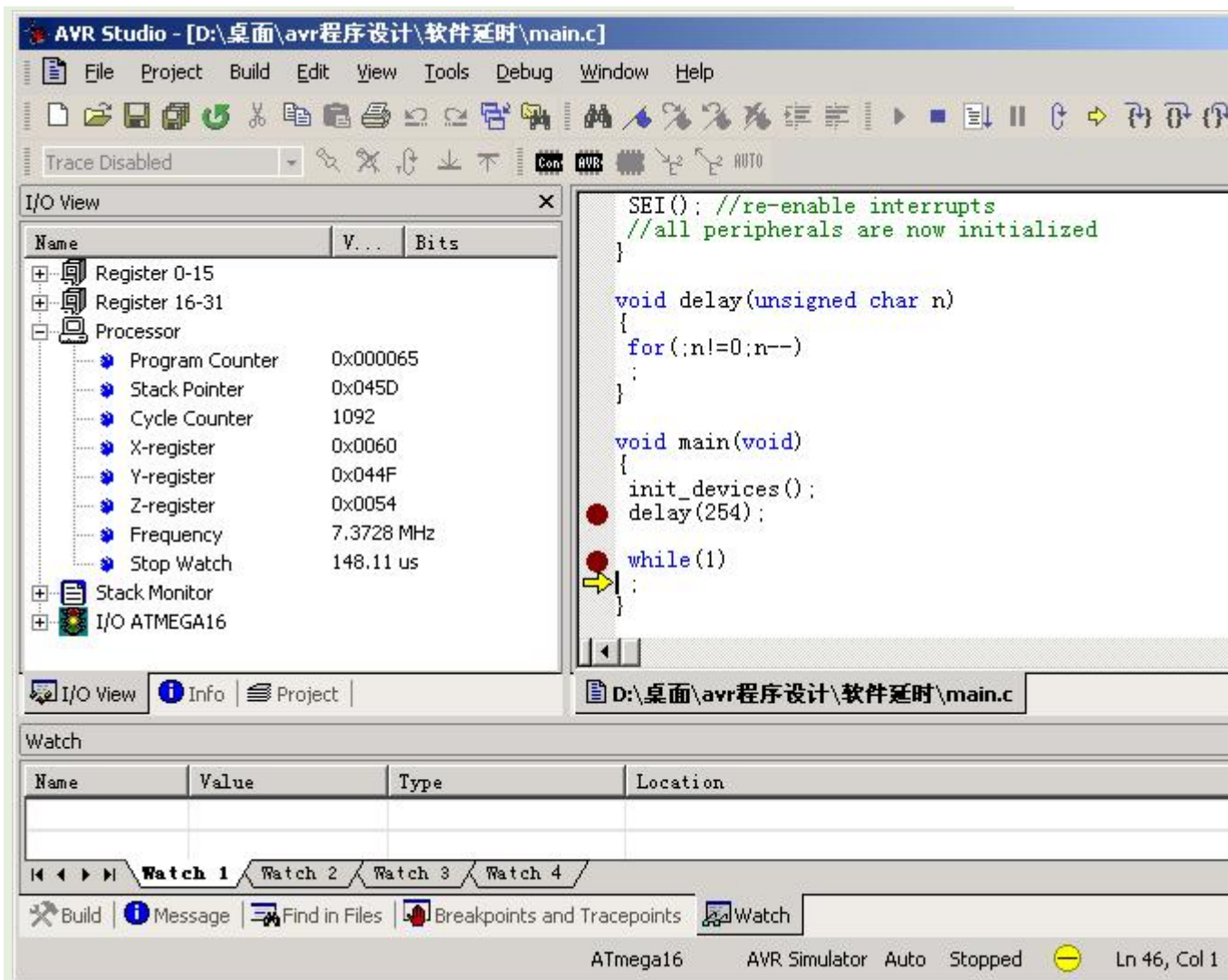
/*****
*****/

```

### 软件准确仿真延时时间

使用 **AVRstudio** 软件仿真可以看到准确的程序运行的时间，设置中断的方式就可以了解到。

调入 **AVR Studio**，为观察延时时间，点击左侧 **Workspace** 中的 **Processer**，注意看其中的几个参数：**Cycle Counter** 和 **Stop Watch**，前一个是执行周期数，即从复位开始到目前为止共执行了多少个周期，而 **Stop Watch** 则表示从复位开始到目前为止共用去的时间数，如果 **Frenance** 中的频率值正确，那么这个时间就是正确的。这样，我们可以通过观察这个时间来调循环次数，将时间基本精确地调整到延时 **1ms**。



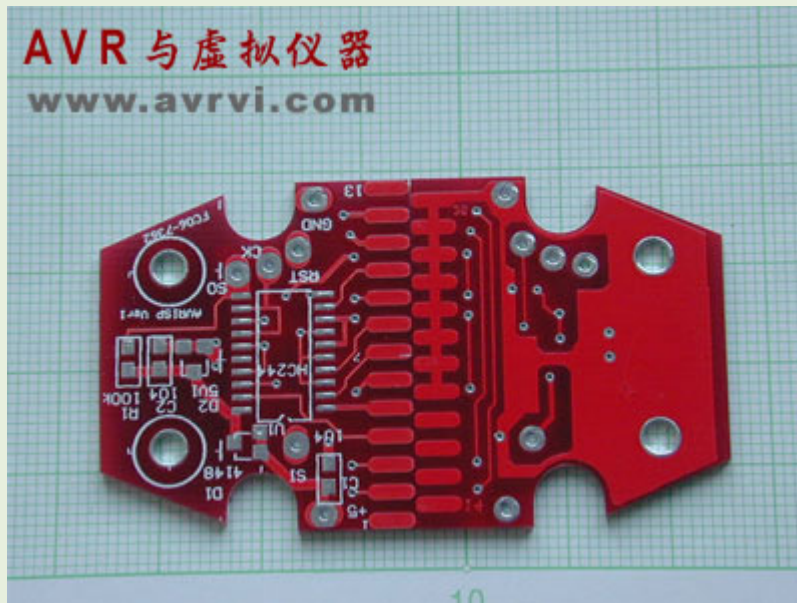
在运行到第一个中断的时候 stop watch 的值是 6.68，当运行到第二个中断的时候，stop watch 的值为 148.11，可以得到 delay(254)这条语句的执行时间约为  $148.11 - 6.86 = 141.25\mu s$ 。我们看到软件仿真的时钟周期是 1028 个，与上面计算的 1020 个有一定差距，因为上面的计算我们忽略了调用程序所花的时间。

由于笔者技术有限，错漏之处在所难免，还望高手指点，以期我们更正。

## AVR 使用范例--自制简易 ISP 下载线

本页关键词：ISP 下载线制作资料 ISP 下载线制作 自制 ISP

本站的完成的 ISP 下载线：

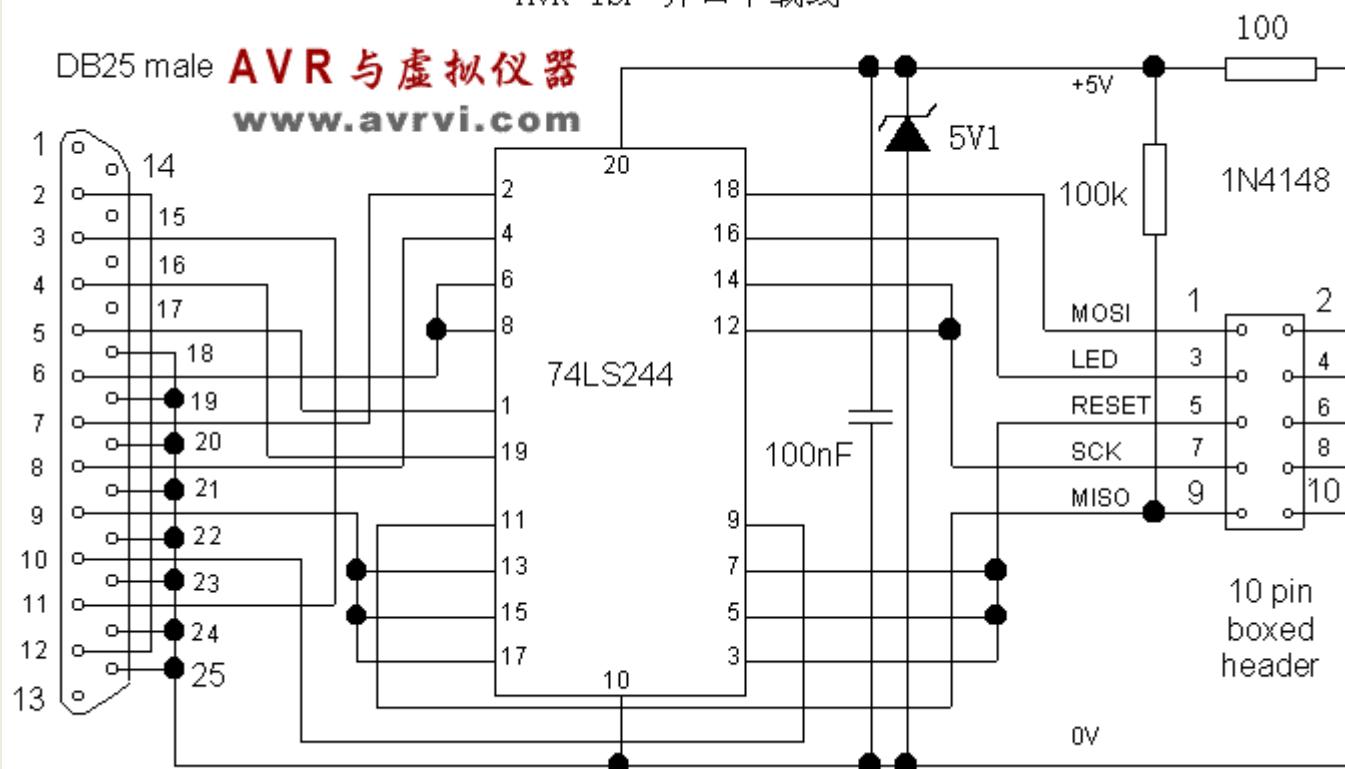


#### 并口 ISP 下载线

重要声明： 本网站提供的下面两个 ISP 线路图，已经过通过 实验验证，绝对没有问题。如果你碰到制作后不能使用的情况，请认真检查有没有焊错或漏焊。

访问 <http://www.mcselec.com>，下载 BASCOM-AVR 的 DEMO 版软件，在它的 HELP 中可以找到 STK200/STK300 Programmer 的电路图。使用一片 244 加几个电阻和电容。在 BASCOM-AVR、ICCAVR、CVAVR 中以及那个免费小马头的下载软件中都支持该 AVR 下载线，安全可靠。

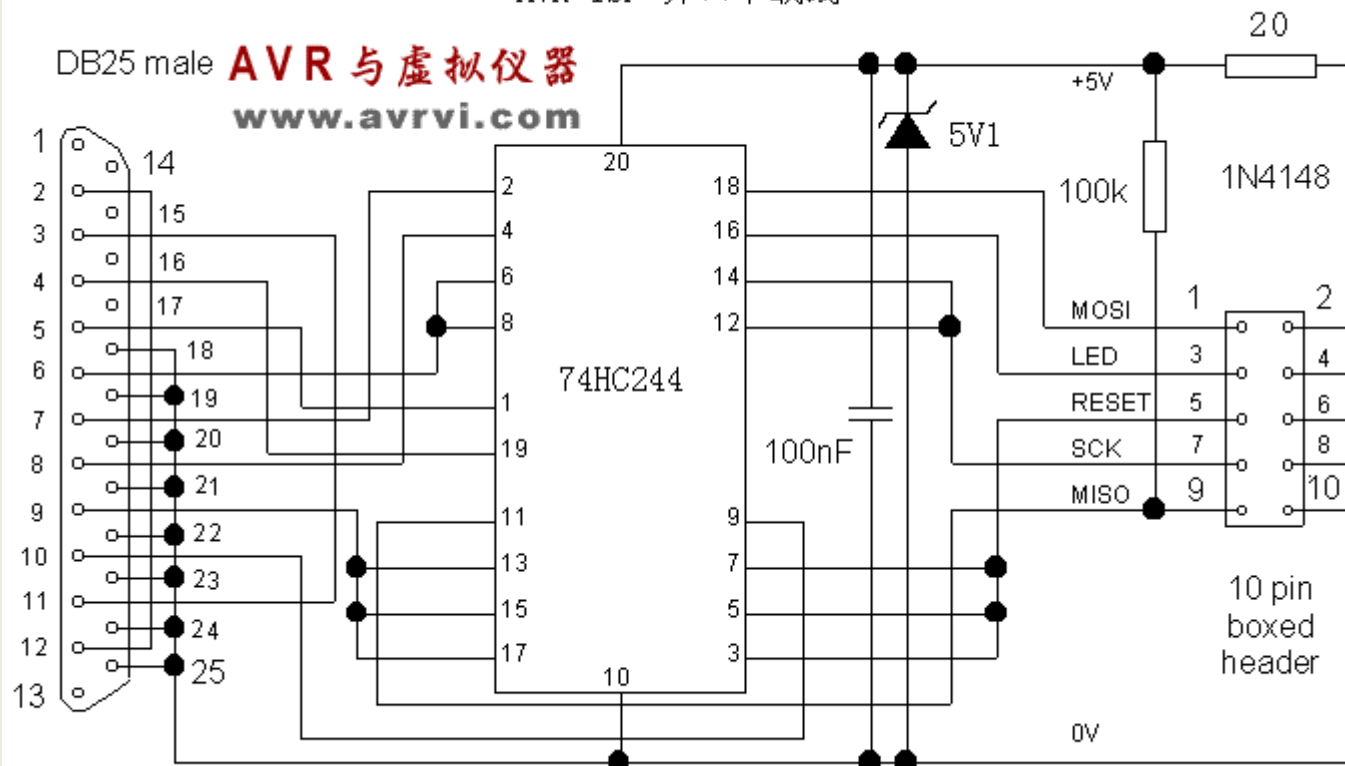
## AVR ISP 并口下载线



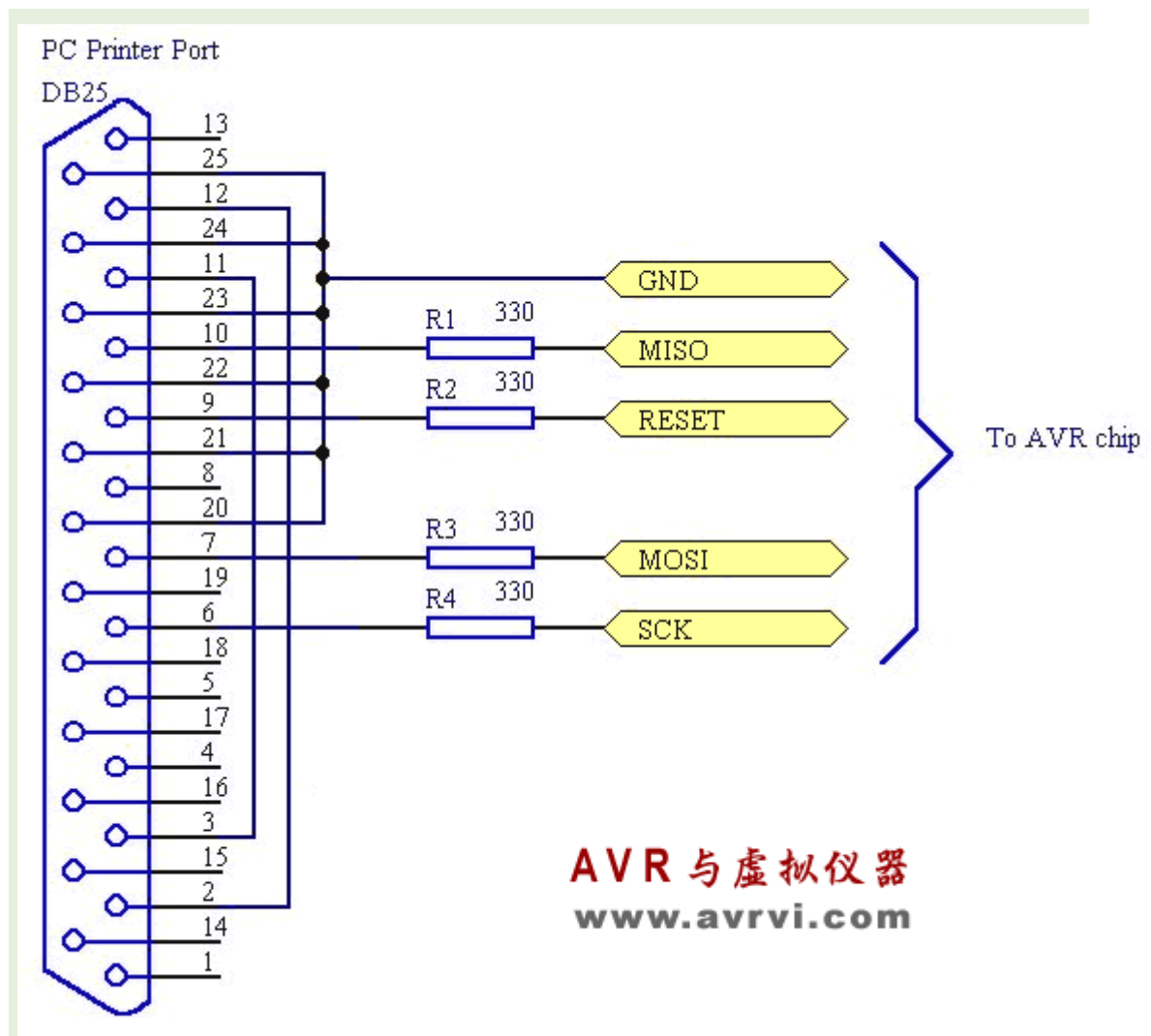
### 电路图更新说明

根据网友 JAMESKING 的描述，上面的电路图存在一些毛病。图上的那个 100 欧电阻应该改为 20 欧或者干脆去掉，不去掉的结果就是很容易锁死芯片，而采用 74LS244 在 3V 系统使用中会出现无法下载的现象，所以也应该改为 74HC244，将这两处改动后，这种下载线还是很好用的。

## AVR ISP 并口下载线



网上收集的最简化的设计图。该线路图已经在 SLisp1.32 和 PonyProg2000 下测试过，下载顺利。



并口 ISP 下载线成本低，制作容易，对串口资源紧张的用户不失一个好的选择，但是速度比较慢。

ISP 相对于 JTAG 来说，不如 JTAG 方便，不可以在线调试。

## AVR 使用范例--自制简易 JTAG

本页关键词：自制 JTAG，JTAG 使用，JTAG 制作资料

推荐：豪华版 AVR JTAG ICE & ISP stk500 二合一 avr 下载编程 avr jtag 仿真器

详细使用说明书：[http://www.avrvi.com/start/AVR\\_JTAG\\_ICE\\_ISP\\_STK500\\_USER\\_GUIDE.pdf](http://www.avrvi.com/start/AVR_JTAG_ICE_ISP_STK500_USER_GUIDE.pdf)



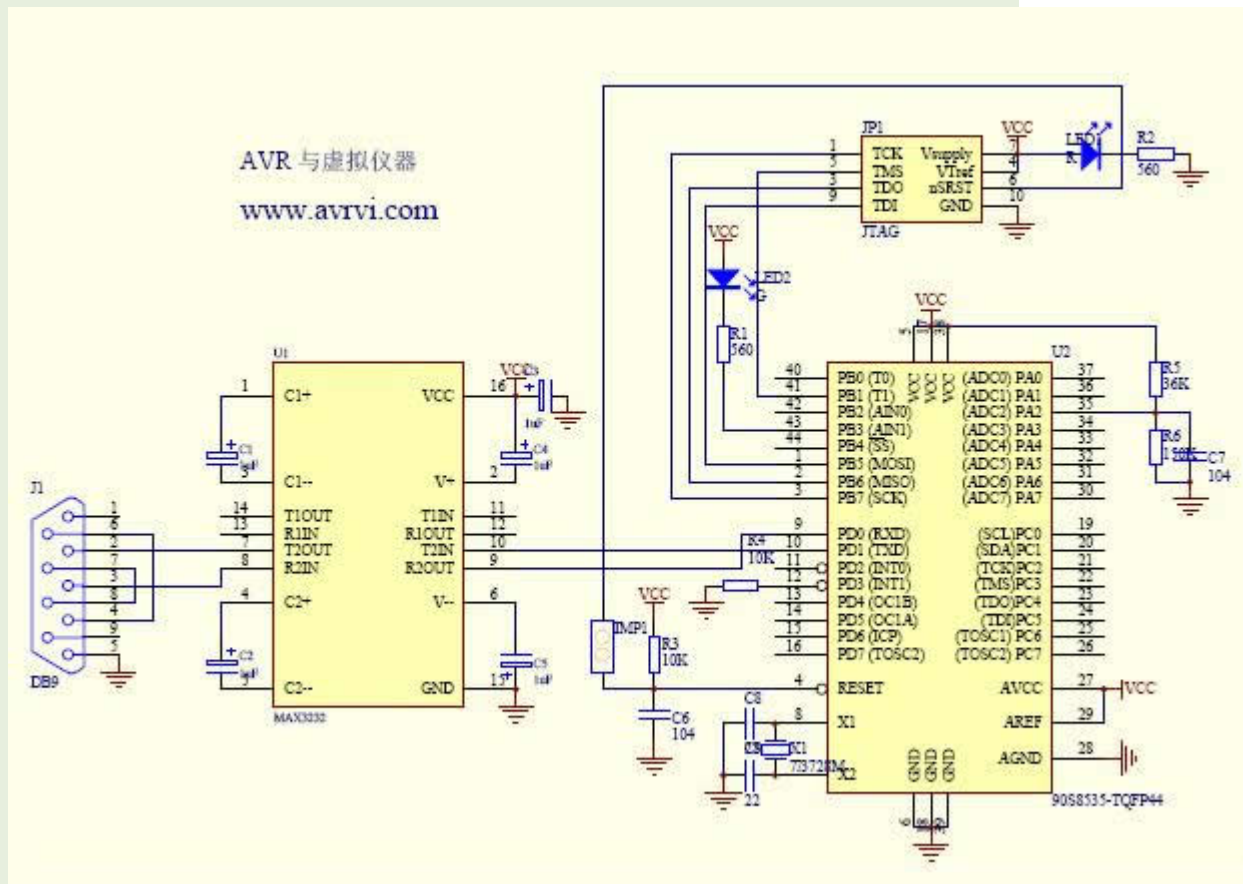
点击图片进入论坛查看讨论。 <http://shop.avrvi.com/goods-1.html>

本站提供的 JTAG 制作资料：

1. 按电路图接好电路，本例给出的是贴片封装，如果使用直插封装采用对应的接口即可。



## 2. [使用max232 的电路图【pdf】](#)



使用分立元件的电路图(稍后奉上)

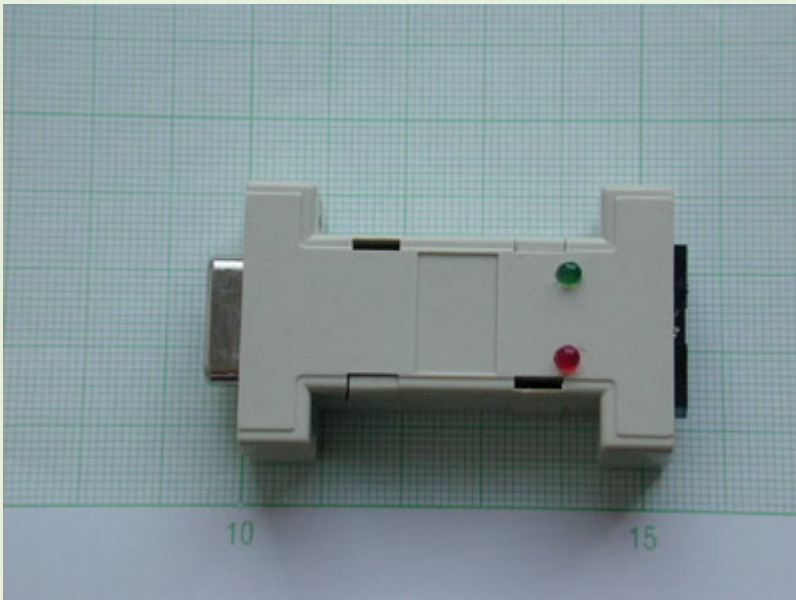
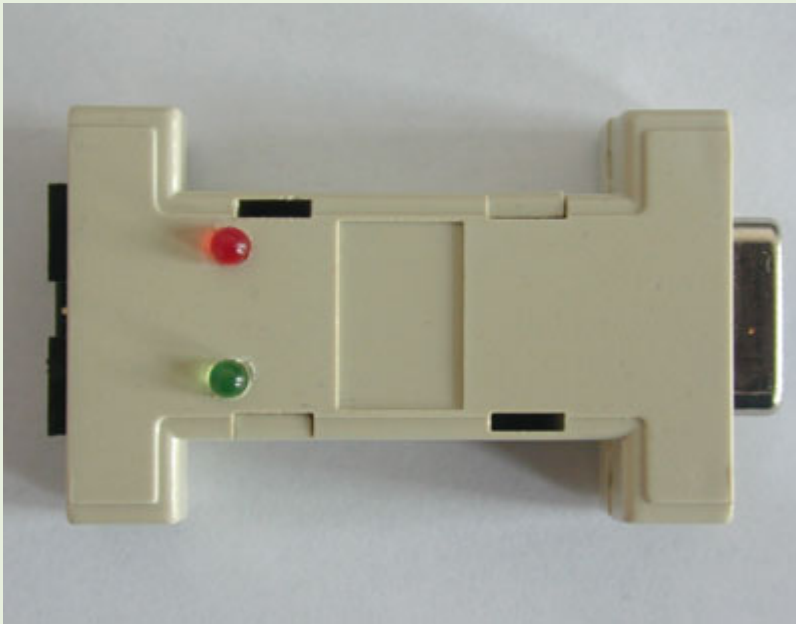
## 3. 将下面的 [HEX文件](#)通过其他方式写入Atemega16, eeprom.hex写入eeprom, flash.hex写入flash。

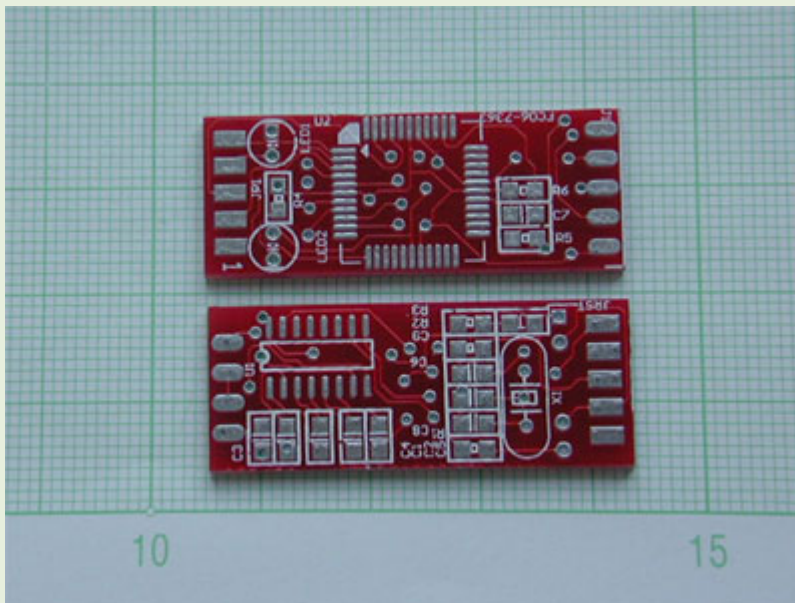
官方提供的 JTAG 制作资料：

官方提供的JTAG制作资料，[点击下载](#)。

本站成品：







为了节约时间成本，你无需自己制作，你可以和本站联系。

## AVR 使用范例--定时器应用范例

本页关键词：**avr 定时器计算** **avr 定时器应用** **avr 定时器中断范例** **avr 定时器** **avr 定时器0编程** **avr 定时器的应用程序** **avr 定时器使用例程** **avr 定时器输出脉冲** **avr 定时器程序** **avr 定时器初值**

本文详细介绍 **avr 定时器 timer** 的使用方法, **PWM** 的设定与注意事项, 通过 **ICC** 快速完成参数设定, 并通过一个实际例子定时 **LED** 的显示解说详细过程。定时器在工程中应用非常广泛, **avr** 有八位和十六位两种定时器, **AVR** 定时器能够非常精确的定时, 下面介绍使用 **ICC application builder** 快速使用定时器的方法。

### 使用 ICC application builder 快速配置定时器

第一步：新建工程保存到特定目录下。

第二步：ICC>>Tools>>application builder

第三步：设置单片机型号和晶振频率，如下图，非常重要，因为这会关系到定时的准确性甚至正确性。

**ICCAVR Application Builder [M16]**

CPU | Ports | Timer0 | Timer1 | Timer2 | UART | SPI | Analog

**Processor**

Target CPU: M16  
Xtal speed (MHz): 7.3728  
*type a custom value*  
☐ PLL enable  
☐ XDIV enable...  
XDIV frequency: 57153Hz

**Watchdog timer**

☐ Enable Prescale cycles: 16K  
☐ Interrupt enable

**External interrupts**

Trigger on...

Interrupt	Trigger on...
<input type="checkbox"/> INT0	Low level
<input type="checkbox"/> INT1	Low level
<input type="checkbox"/> INT2	Low level
<input type="checkbox"/> INT3	Low level
<input type="checkbox"/> INT4	Low level
<input type="checkbox"/> INT5	Low level
<input type="checkbox"/> INT6	Low level
<input type="checkbox"/> INT7	Low level

Edge ☒

**EEPROM**

☐ ready interrupt

Comment: new design

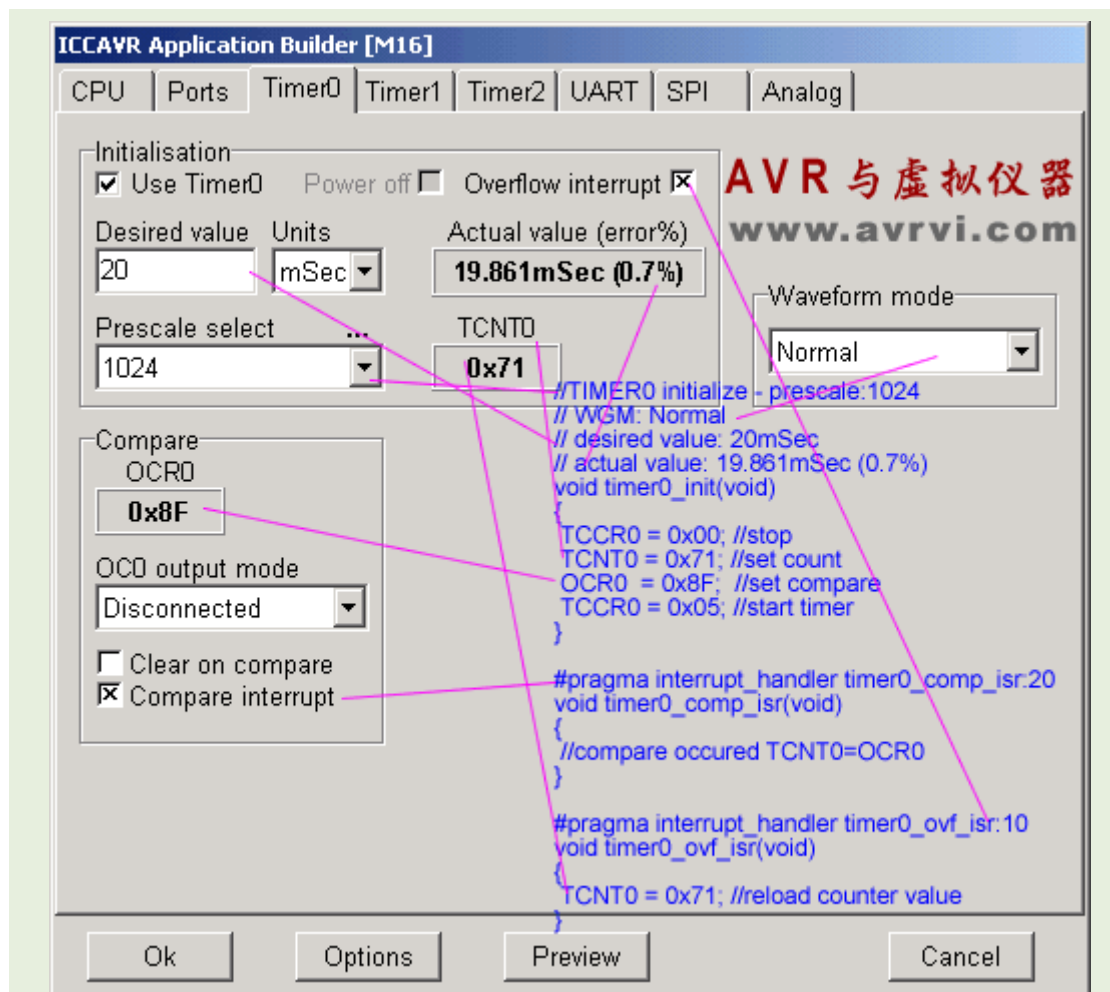
Ok Options Preview Cancel

**AVR 与虚拟仪器**  
**www.avrvi.com**

第四步：设定定时器 Timer0，操作如下图，请仔细核对每一项。这里我们跳过了端口 IO 的设置，因为对我们不是很重要。



点击 OK，检查生成的程序，下图展示了生成的程序的详细含义。



你将得到的程序如下，灰色部分为添加的注释。只介绍定时器部分，其他请参考新手入门相关内容，

如果你并不急于理解这些内容，请跳过。

/\*定时器的工作原理是：定时器在预分频这么多个时钟周期后使 **TCNTn** 的值加一，当 **TCNTn** 到达最大值时发生溢出中断。

用户在溢出中断中填写定时器重载语句，开始下一次定时工作，通过设定 **TCNTn** 的值和 **OCRn** 的值可以设置定时器的定时长短。\*/

```

//TIMER0 initialize - prescale:1024 /*定时器预分频，预分频由TCCRn的
CS02, CS01, CS00 确定，详情查看数据手册*/
  
```

```

// WGM: Normal/*定时器*/
  
```

```

// desired value: 20mSec/*定时器期望设定时间*/
  
```

```

// actual value: 19.861mSec (0.7%)/*定时器实际定时时间，误差比例*/
  
```

```

void timer0_init(void)
{
    TCCR0 = 0x00; //stop/*定时器停止，TCCR0 寄存器完全控制timer0 的运行情况，详细可参考数据手册。*/

    TCNT0 = 0x71; //set count/*定时器寄存器开始值*/
    OCR0 = 0x8F; //set compare/*定时器比较值*/
    TCCR0 = 0x05; //start timer/*定时器开始*/
}

#pragma interrupt_handler timer0_comp_isr:20
void timer0_comp_isr(void)
{
    //compare occured TCNT0=OCR0 /*定时器比较匹配中断，这里没有添加任何语句，实际操作中可以用此实现自制PWM*/
}

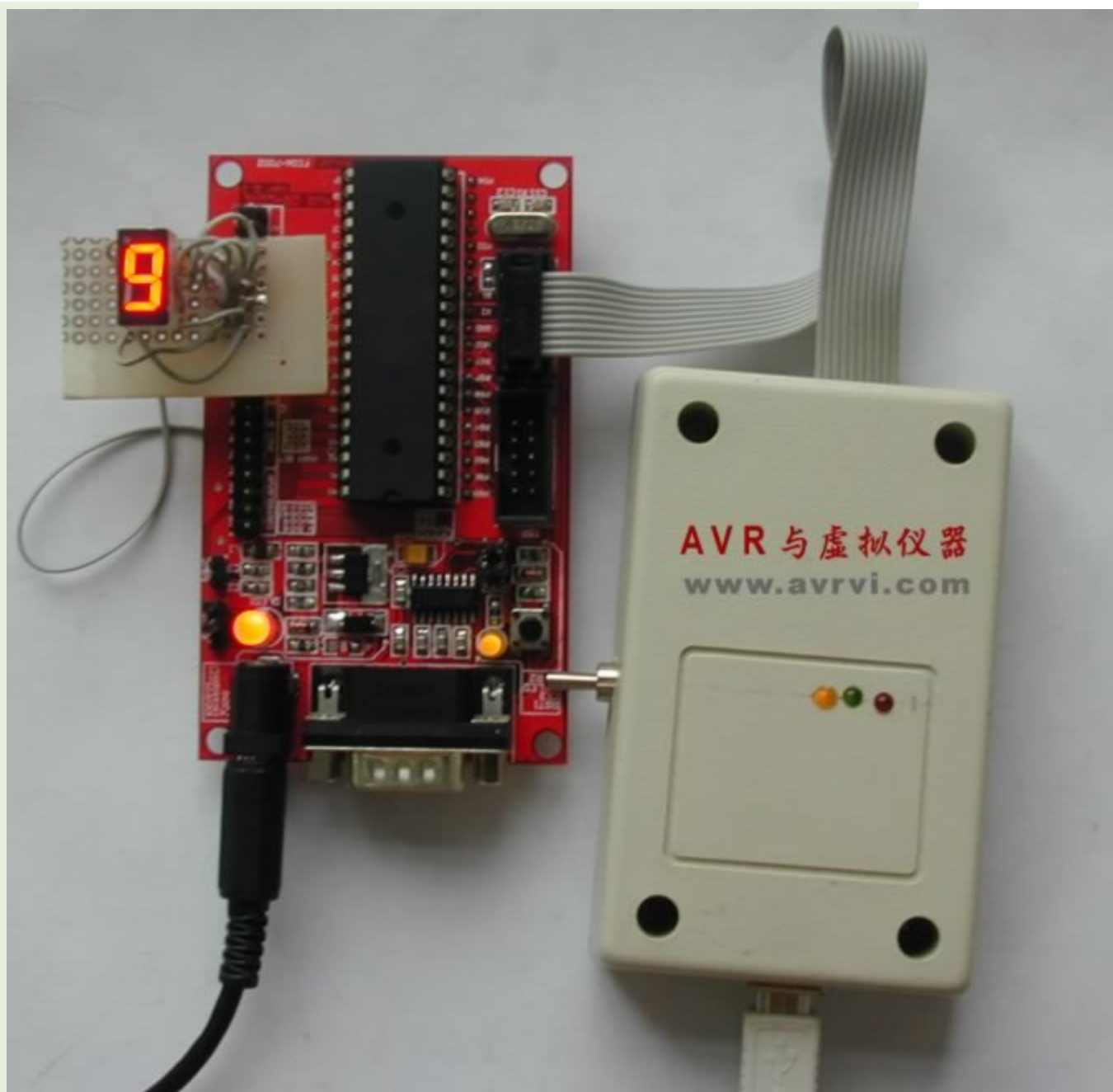
#pragma interrupt_handler timer0_ovf_isr:10
void timer0_ovf_isr(void)
{
    TCNT0 = 0x71; //reload counter value /*定时器溢出后需要重载TCNTn，然后在之后添加用户程序，
    记住不要在定时器中断里添加特别耗时的程序。*/
}

//call this routine to initialize all peripherals
void init_devices(void)
{
    //stop errant interrupts until set up
    CLI(); //disable all interrupts
}

```

```
port_init();  
timer0_init();  
  
MCUCR = 0x00;  
GICR  = 0x00;  
TIMSK = 0x03; //timer interrupt sources/*这里设定允许Timer0 比较中断  
和溢出中断。*/  
SEI(); //re-enable interrupts  
//all peripherals are now initialized  
}
```

下面通过用同样的方法操作做一个“小秒表”， LED 每秒更新一次，效果如下图：



本程序我们用了定时器二，因为timer0 是八位定时器，无法完成一秒的定时任务。[下载相关文件](#)



**ICCAVR Application Builder [M16]**

CPU | Ports | Timer0 | **Timer1** | Timer2 | UART | SPI | Analog

**Initialisation**

☒ Use Timer1    Power off ☐    Overflow interrupt ☒

Desired value: 1    Units: Sec    Actual value (error%): 1.000Sec (0.0%)

Prescale select: 1024    TCNT1H: 0xE3    TCNT1L: 0xE1

Waveform mode: SELECT...    TOP: 0xFFFF

**Compare A**

OCR1AH: 0x1C    OCR1AL: 0x1F

OC1A output mode: Disconnected

☐ Clear on compare    ☒ Compare A interrupt

**Compare B**

OCR1BH: 0x1C    OCR1BL: 0x1F

OC1B output mode: Disconnected

☒ Compare B interrupt

**Input capture**

☐ Noise cancel    ☐ Pin enable    ICR1H: 0x1C    ICR1L: 0x1F

☐ Rising edge    ☐ Capture interrupt

Ok    Options    Preview    Cancel

程序如下：

```
//ICC-AVR application builder : 2006-11-24 11:46:03
// Target : M16
// Crystal: 7.3728Mhz

#include <iom16v.h>
#include <macros.h>

const
led_table[16]={0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07, 0x7f, 0x6f, 0x88
, 0x86, 0xc7, 0xc4, 0x83, 0x8b};// 0~~f
//定义LED的数据表，注意，只有0到9是正确的，A-F我没有认真写。
```

```
typedef unsigned char uint8;

uint8 i;

void port_init(void)
{
    PORTA = 0x00;
    DDRA  = 0xFF;
    PORTB = 0x00;
    DDRB  = 0x00;
    PORTC = 0x00; //m103 output only
    DDRC  = 0x00;
    PORTD = 0x00;
    DDRD  = 0x00;
}

//TIMER1 initialize - prescale:1024
// WGM: 0) Normal, TOP=0xFFFF
// desired value: 1Sec
// actual value: 1.000Sec (0.0%)
void timer1_init(void)
{
    TCCR1B = 0x00; //stop
    TCNT1H = 0xE3; //setup
    TCNT1L = 0xE1;
    OCR1AH = 0x1C;
    OCR1AL = 0x1F;
    OCR1BH = 0x1C;
    OCR1BL = 0x1F;
    ICR1H  = 0x1C;
```

```

ICR1L  = 0x1F;
TCCR1A = 0x00;
TCCR1B = 0x05; //start Timer
}

#pragma interrupt_handler timer1_compa_isr:7
void timer1_compa_isr(void)
{
    //compare occurred TCNT1=OCR1A
}

#pragma interrupt_handler timer1_compb_isr:8
void timer1_compb_isr(void)
{
    //compare occurred TCNT1=OCR1B
}

#pragma interrupt_handler timer1_ovf_isr:9
void timer1_ovf_isr(void) //将每秒执行一次
{
    //TIMER1 has overflowed
    TCNT1H = 0xE3; //reload counter high value
    TCNT1L = 0xE1; //reload counter low value
    i++;
    if(i==10) i=0; //等于十时，恢复零
    PORTA=led_table[i]; //这里使LED变化
}

//call this routine to initialize all peripherals

```

```

void init_devices(void)
{
    //stop errant interrupts until set up
    CLI(); //disable all interrupts
    port_init();
    timer1_init();

    MCUCR = 0x00;
    GICR  = 0x00;
    TIMSK = 0x1C; //timer interrupt sources /*定时器使用中断规则设定*/
    SEI(); //re-enable interrupts
    //all peripherals are now initialized
}

void main(void)
{
    init_devices();
    while(1)
    ;
}

```

LED 的连接方法，如下：

如果你看完本文，对定时器仍然没有很清晰的认识，请和我们联系，向我们建议。

## AVR 使用范例--定时器实现 PWM 功能

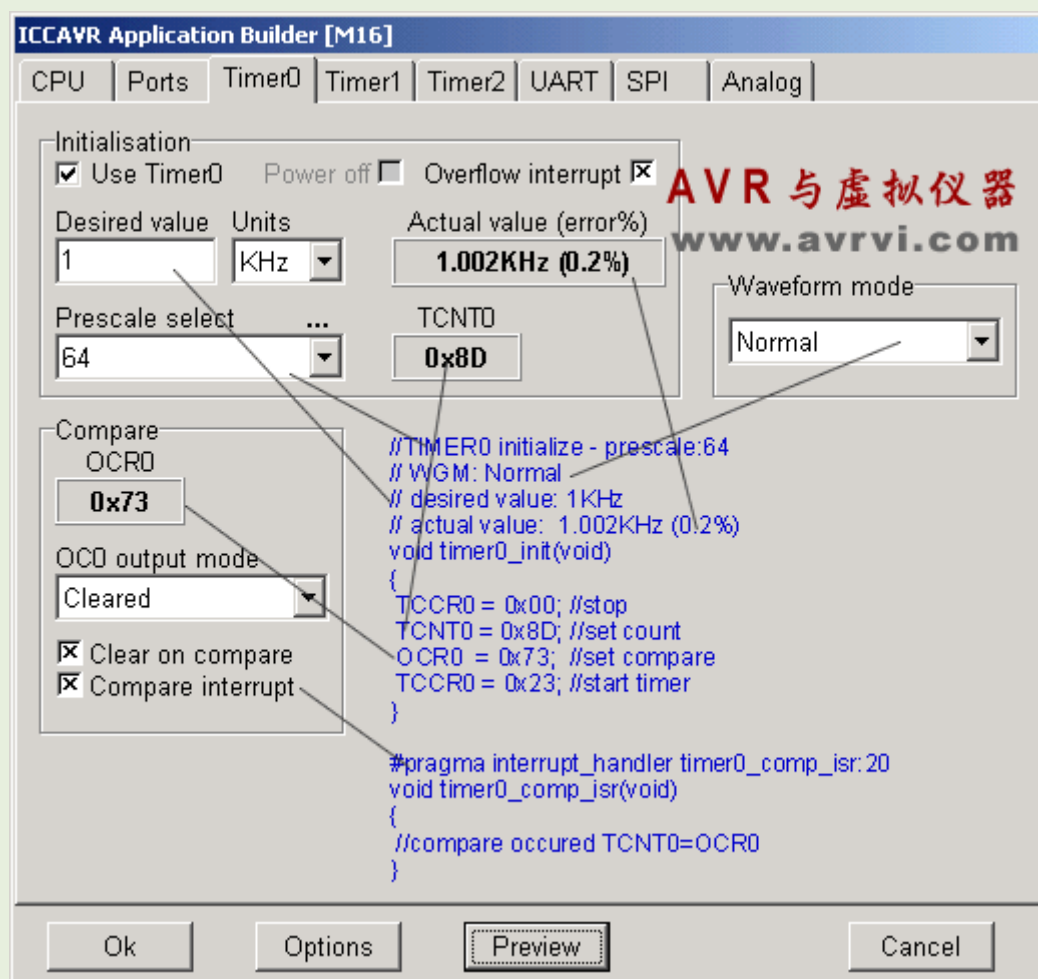
本页关键词： 什么是 pwm pwm 原理 pwm 控制 pwm 芯片 pwm 单片机 单片机 pwm 控制应用  
pwm 电路 pwm 输出 pwm 调制

脉宽调制 PWM 是开关型稳压电源中的术语。这是按稳压的控制方式分类的，除了 PWM 型，还有 PFM 型和 PWM、PFM 混合型。脉宽宽度调制式（PWM）开关型稳压电路是在控制电路输出频率不变的情况下，通过电压反馈调整其占空比，从而达到稳定输出电压的目的。

通俗的说 PWM 就是波形，波形的波峰以波谷的比例关系成为占空比，我们可以通过 PWM 控制电机，音量控制，模拟控制等。

AVR 单片机的定时器可以轻松实现 PWM 功能。mega16 和 mega32 的 timer0 是和 timer2 都具有 PWM 功能，timer0 和 timer2 都为 8 位定时器。timer2 为异步操作定时器，在操作过程中要等待寄存器状态更改完成。详情参看数据手册：8 位有 pwm 操作的异步操作定时器 timer2。

下图设定使用 timer0 来实现 PWM 功能。PWM 电机控制：访问 [AVR 与 L298 进行直流电机控制](#)。



OC0 output mode 设定了 pwm 输出控制选择：正常的端口操作，不与 OC0 相连接，比较匹配发生时 OC0 取反，比较匹配发生时 OC0 清零，比较匹配发生时 OC0 置位。

**Waveform mode**设定了波形产生模式：比较匹配输出模式，快速PWM 模式，相位修正PWM 模式。

更详细的内容请参看数据手册。

看看程序代码：[下载相关文件](#)

```
//ICC-AVR application builder : 2006-11-25 0:15:12
// Target : M16
// Crystal: 7.3728Mhz

#include
#include

void port_init(void)
{
    PORTA = 0x00;
    DDRA  = 0x00;
    DDRB  = 0x08; //PB3 为PWM输出，非常重要，否则无法输出波形
    DDRB  = 0x00;
    PORTC = 0x00; //m103 output only
    DDRC  = 0x00;
    PORTD = 0x00;
    DDRD  = 0x00;
}

//TIMER0 initialize - prescale:64
// WGM: Normal
// desired value: 1KHz
// actual value: 1.002KHz (0.2%)
void timer0_init(void)
{

```

```

TCCR0 = 0x00; //stop
TCNT0 = 0x8D; //set count /*TCNT0*/
OCR0  = 0x73; //set compare /*OCR0*/
TCCR0 = 0x23; //start timer /*TCCR0*/
}

#pragma interrupt_handler timer0_comp_isr:20
void timer0_comp_isr(void)
{
    //compare occurred TCNT0=OCR0
}

#pragma interrupt_handler timer0_ovf_isr:10
void timer0_ovf_isr(void)
{
    TCNT0 = 0x8D; //reload counter value
}

//call this routine to initialize all peripherals
void init_devices(void)
{
    //stop errant interrupts until set up
    CLI(); //disable all interrupts
    port_init();
    timer0_init();

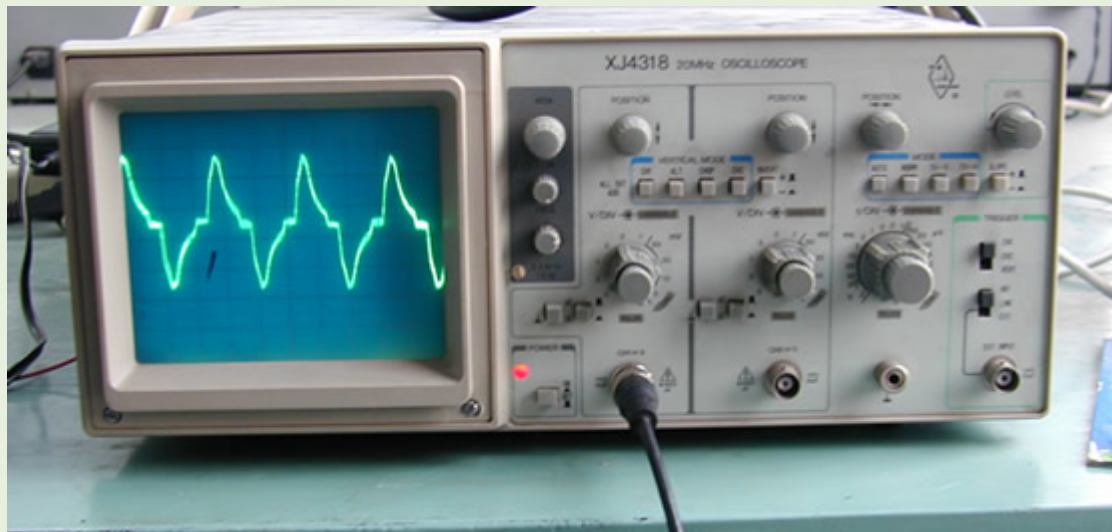
    MCUCR = 0x00;
    GICR  = 0x00;
    TIMSK = 0x03; //timer interrupt sources /*TIMSK*/

```

```
SEI(); //re-enable interrupts
//all peripherals are now initialized
}

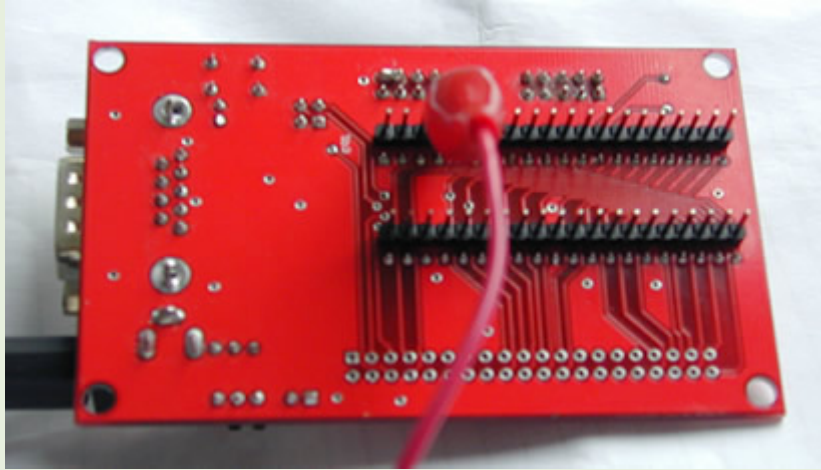
void main(void) /*加上这些，程序就可以运行了。*/
{
    init_devices();
    while(1)
    ;
}
```

程序运行效果：



波形从 PB3 输出：





看看几个关键的寄存器的意义：

1. **TCNT0**: 定时器计数值，定时过程中不断增大，溢出后重新置数，开始下一轮。
2. **OCR0**: 定时器比较的值，当 $TCNT0 = OCR0$  时，会产生timer0\_comp\_isr中断。
3. **TCCR0**: 控制timer0 的寄存器，这里 0x23 代表的是：

7	6	5	4	3	2	1	0	
FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
0	0	0	0	0	0	0	0	
		1	www.avrvi.com			1	1	

波形产生模式为普通模式，比较匹配发生时 OC0 清零，clkI/O/64 ( 来自预分频器)，详细内容请查看数据手册。

4. **TIMSK**: 定时器中断选项，这里允许timer0 比较中断，溢出中断。
5. 预分频器：预分频器是独立运行的。也就是说，其操作独立于 T/C 的时钟选择逻辑，且它由 T/C1 与 T/C0 共享。由于预分频器不受 T/C 时钟选择的影响，预分频器的状态需要包含预分频时钟 被用到何处这样的信息。一个典型的例子发生在定时器使能并由预分频器驱动( $6 > CSn2:0 > 1$ )的时候：从计时器使能到第一次开始计数可能花费 1 到  $N+1$  个系统时钟周期， 其中  $N$  等于预分频因子(8、64、256 或 1024)。

PWM 的工作流程：

1. 初始化，定时器开始工作，TCNT0 逐渐增大，在预分频这么多个时钟周期里变化一次。
2. 输出比较寄存器包含一个 8 位的数据，不间断地与计数器数值 TCNT0 进行比较。匹配事件可以用来产生输出比较中断，或者用来在 OC0 引脚上产生波形。
3. TCNT0 溢出，溢出中断用于在 OC0 上产生波形，置位或者清零。
4. TCNT0 复位，进行下一次定时操作。

#### PWM 的占空比：

调节PWM的占空比，只需要用程序更改 OCR0的值即可，根据不同的情况，可能是增加也可能是减小。注意：因为Timer2 是异步控制器，使用Timer2 时，调节OCR2 需要等待寄存器更新完成才能进行其他操作。

### AVR 使用范例--usart 串口使用范例

本页关键词：usart 什么是 usart usb usart usart 接口 usb to usart usb usart 驱动 usb usart tx line usart 串口 usart 芯片

下载 [超好用的串口调试助手【绿色免安装版】](#)，下载 [结构化的uart驱动程序与范例](#)

#### usart 简介：

- 通用同步和异步串行接收器和转发器(USART) 是一个高度灵活的串行通讯设备。主要特点为：
  - 全双工操作( 独立的串行接收和发送寄存器)
  - 异步或同步操作
  - 主机或从机提供时钟的同步操作
  - 高精度的波特率发生器
  - 支持 5, 6, 7, 8, 或 9 个数据位和 1 个或 2 个停止位
  - 硬件支持的奇偶校验操作
  - 数据过速检测
  - 帧错误检测

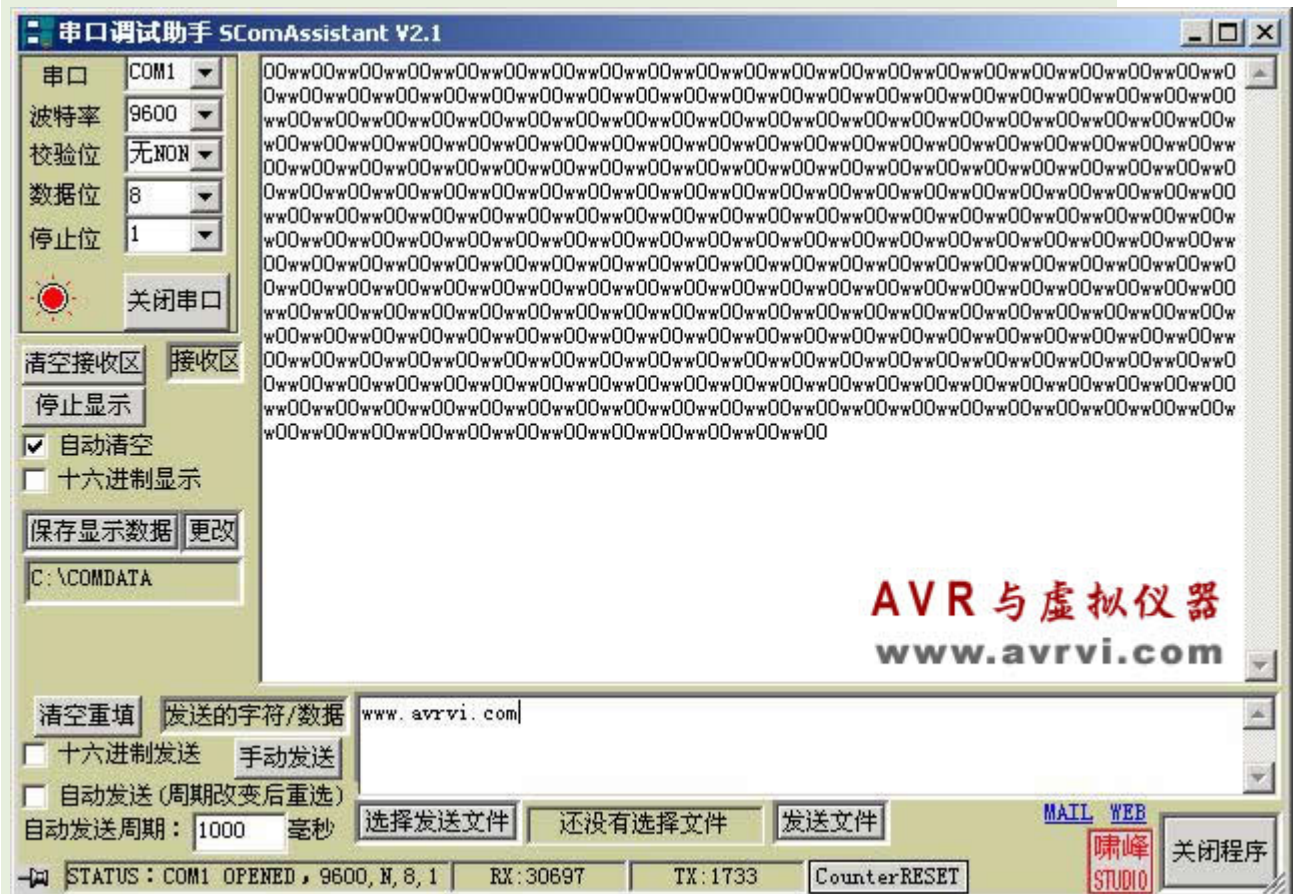
噪声滤波，包括错误的起始位检测，以及数字低通滤波器

三个独立的中断：发送结束中断，发送数据寄存器空中断，以及接收结束中断

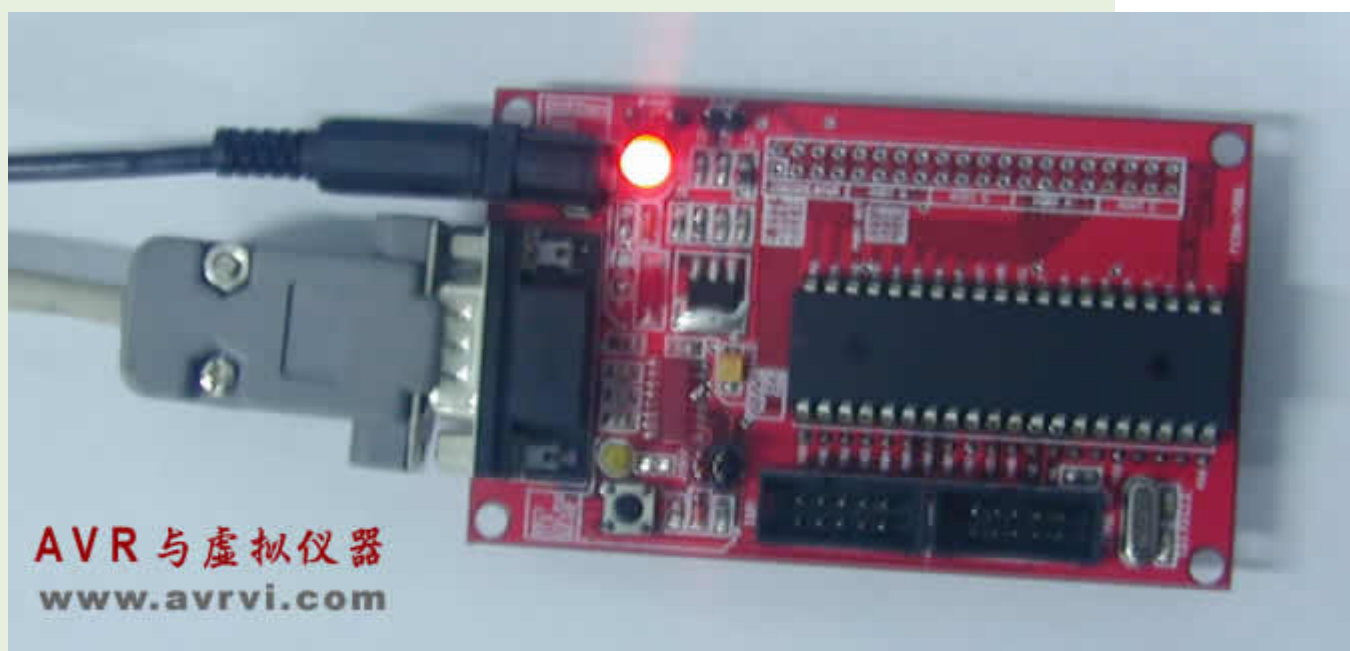
多处理器通讯模式

倍速异步通讯模式

调试结果与实物图

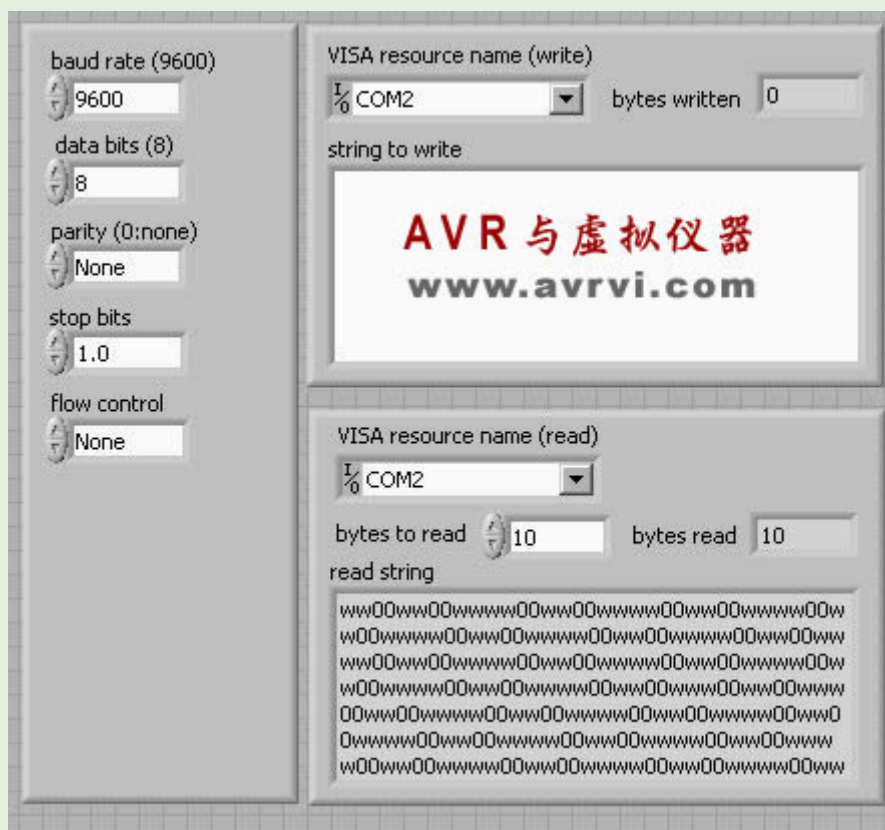


串口硬件电路采用的是MAX232cse，硬件连接电路图，[详见开发板电路的详细介绍](#)。

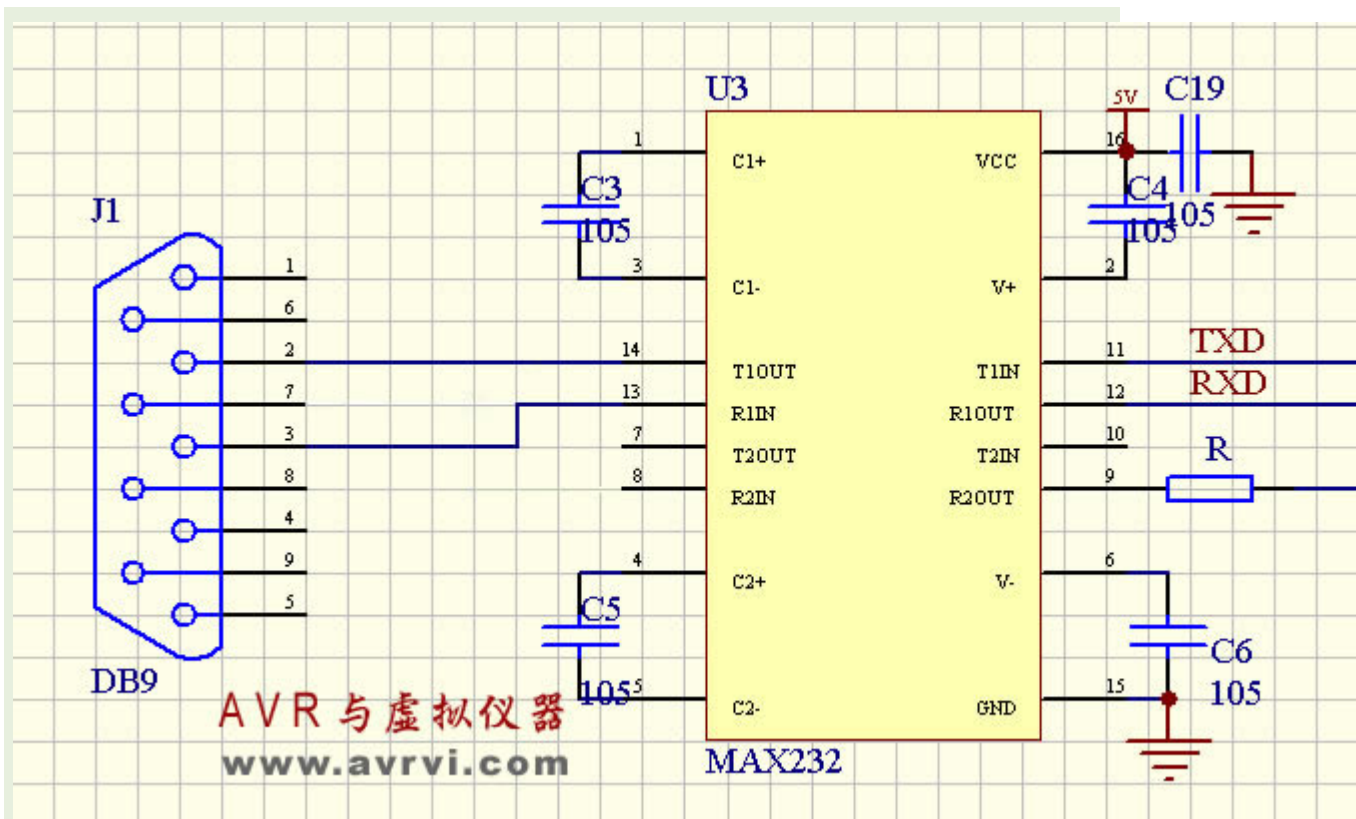


在labview中用程序调试串口的结果，labview串口使用范例

见: <http://bbs.avrvi.com/read.php?fid=8&tid=497&toread=1>



电路图:



串口使用一个 max232 芯片。

数据读写 官方解释

- USART 的初始化

```
void USART_Init( unsigned int baud )
{
    /* 设置波特率*/
    UBRRH = (unsigned char)(baud>>8);
    UBRL = (unsigned char)baud;

    /* 接收器与发送器使能*/
    UCSRB = (1<<RXEN)|(1<<TXEN);

    /* 设置帧格式: 8 个数据位, 2 个停止位*/
    UCSRC = (1<<URSEL)|(1<<USBS)|(3<<UCSZ0);
}
```

- 数据发送【发送 5 到 8 位数据位的帧】

```
void USART_Transmit( unsigned char data )
{
    /* 等待发送缓冲器为空 */

    while ( !( UCSRA & (1<<UDRE)) )

        ;

    /* 将数据放入缓冲器，发送数据 */

    UDR = data;
}
```

- 数据发送【发送 9 位数据位的帧】

```
void USART_Transmit( unsigned int data )
{
    /* 等待发送缓冲器为空 */

    while ( !( UCSRA & (1<<UDRE))) )

        ;

    /* 将第 9 位复制到 TXB8 */

    UCSRB &= ~(1<<TXB8);

    if ( data & 0x0100 )

        UCSRB |= (1<<TXB8);

    /* 将数据放入缓冲器，发送数据 */

    UDR = data;
}
```

- 数据接收【以 5 到 8 个数据位的方式接收数据帧】

```
unsigned char USART_Receive( void )
{
    /* 等待接收数据*/

    while ( !(UCSRA & (1<<RXC)) )

        ;

    /* 从缓冲器中获取并返回数据*/
```

```
return UDR;
```

```
}
```

- 数据接收【以 9 个数据位的方式接收帧】

```
unsigned int USART_Receive( void )
```

```
{
```

```
    unsigned char status, resh, resl;
```

```
    /* 等待接收数据*/
```

```
    while ( !(UCSRA & (1<<RXC)) )
```

```
    ;
```

```
    /* 从缓冲器中获得状态、第 9 位及数据*/
```

```
    /* from buffer */
```

```
    status = UCSRA;
```

```
    resh = UCSRB;
```

```
    resl = UDR;
```

```
    /* 如果出错，返回-1 */
```

```
    if ( status & (1<<FE)| (1<<DOR)| (1<<PE) )
```

```
    return -1;
```

```
    /* 过滤第 9 位数据，然后返回*/
```

```
    resh = (resh >> 1) & 0x01;
```

```
    return ((resh << 8) | resl);
```

```
}
```

结构化的驱动程序与范例 [下载所有相关文件](#)

- 首先包含 sio.h 和相关头文件。
- 调用 **Com\_init()**; //串口初始化
- 主程序

```
void main(void)
```

```
{
```

```
    uint8 SIO_buff[4]; //定义串口数据缓冲区
```



```

init_devices();

//指示单片机正常开始工作，一亮，二亮，都亮，都灭

PORTA = 0x02; //1 脚为高，0 脚为低，0 脚灯亮

Delay(); //延时

PORTA = 0x01; //0 脚为高，1 脚为低，1 脚灯亮

Delay(); //延时

PORTA = 0x00; //同时亮

Delay(); //延时

PORTA = 0x03; //同时灭


Com_putstring ("ww",2,&RTbuf_UART0); //输出一个字符串到串行口

while(1)

{

if(Com_R_count(&RTbuf_UART0)!=0)

{

Com_getstring (SIO_buff,1,&RTbuf_UART0);

Com_Rbuf_Clear(&RTbuf_UART0); //清空 buffer

if(SIO_buff[0]==1) //接收到一个一

{

Com_putstring ("ok!!",4,&RTbuf_UART0); //输出一个字符串到串行口

PORTA = 0x00; //同时亮

Delay(); //延时

PORTA = 0x03; //同时灭

}

} //end of if(Com_R_count(&RTbuf_UART0)!=0)

Delay(); //延时

Com_putstring ("ww00",4,&RTbuf_UART0); //输出一个字符串到串行口

}

}

```



串口调试经验谈：

1、确保你的串口连线正常，有的串口的接法，9针串口线的7针和第八针要交叉，本站最小系统板的V2.3需要交叉，V2.5不需要交叉。

2、确保你的MAX232工作正常，方法：在本开发小板上把跳线JP2的1和3短接,2和4不接,通过串口调试助手发什么数据就可以收到什么数据。

3、最容易出现的一个问题是，晶振的选择，本程序要求晶振是7.3728M外部晶振，为什么使用7.3728的特殊晶振，因为它可以产生标准的波特率，如果你收到的是乱码或者根本无法收到，请检查你的晶振的相关熔丝位设置是否正确，这里看看[熔丝位快速入门](#)。

## AVR 使用范例--AVR 模拟比较器使用范例

AVR 的模拟比较器模块可以用来比较接在 AIN0(mega16 PB2 的第二功能)和 AIN1(PB3)两个引脚的电压大小。

程序操作流程：初始化 >> 开中断 >> 中断服务程序判断，比较结果将会同步到

模拟比较器控制和状态寄存器—ACSR 的第五位 ACO，检测 ACO 的值就可以得出比较结果。

AIN0<AIN1(ACO=0); AIN0>AIN1(ACO=1)

论坛相关链接：<http://www.avrvi.com/bbs/read.php?fid=30&tid=1569&toread=1>

// ICC-AVR application builder : 2007-3-15 8:22:58

// Target : M16

// Crystal: 7.3728Mhz

// AVR 模拟比较器使用范例

#include <iom16v.h>

#include <macros.h>

#include "delay.h"

```

//管脚定义

#define LED0 0 //PB0

#define AIN_P 2 //PB2(AIN0)

#define AIN_N 3 //PB3(AIN1)


//宏定义

#define LED0_ON() PORTB|= (1<<LED0) //输出高电平，灯亮

#define LED0_OFF() PORTB&=~(1<<LED0) //输出低电平，灯灭


//常量定义

/*

模拟比较器的正输入端由 ACBG 位决定，=0 选择 AIN0 引脚，=1 选择 1.23V 内部能隙基准源

模拟比较器多工输入 （不常用，因为 ADC 将无法使用）

可以选择 ADC7..0 之中的任意一个来代替模拟比较器的负极输入端。

ADC 复用器可用来完成这个功能。

当然，为了使用这个功能首先必须关掉 ADC。

如果模拟比较器复用器使能位(SFIOR 中的 ACME) 被置位，且 ADC 也已经关掉(ADCSRA 寄存器的
ADEN 为 0)，

则可以通过 ADMUX 寄存器的 MUX2..0 来选择替代模拟比较器负极输入的管脚，

如果 ACME 清零或 ADEN 置位，则模拟比较器的负极输入为 AIN1。

*/

#define AC_ADC0 0x00 //ADC0

#define AC_ADC1 0x01 //ADC1

#define AC_ADC2 0x02 //ADC2

#define AC_ADC3 0x03 //ADC3

#define AC_ADC4 0x04 //ADC4

#define AC_ADC5 0x05 //ADC5

#define AC_ADC6 0x06 //ADC6

#define AC_ADC7 0x07 //ADC7

```

```

void port_init(void)
{
PORTA = 0x00;

DDRA = 0x00;

PORTB = ~(1<<AIN_P)|(1<<AIN_N); //作模拟比较器输入时，不可使能内部上拉电阻。

DDRB = (1<<LED0); //PB0 作输出

PORTC = 0x00; //m103 output only

DDRC = 0x00;

PORTD = 0x00;

DDRD = 0x00;
}

//初始化的步骤，关中断，更改 ACSR 的值，配置模拟比较器，开中断。

//Comparator initialize

// trigger on: Output toggle

void comparator_init(void)
{

ACSR = ACSR & 0xF7; //ensure interrupt is off before changing

//上面一句会使 ACIE 为零，不允许中断

ACSR=(1<<ACIE);

// 使能模拟比较器中断，比较器输出变化即可触发中断，AIN0 为正输入端，AIN1 为负输入端。

}

#pragma interrupt_handler ana_comp_isr:17

void ana_comp_isr(void)
{

//analog comparator compare event

//硬件自动清除 ACI 标志位

delay_us(10);

if ((ACSR&(1<<ACO))==0) //检测 ACO

```

```

//Bit 5 ACO: 模拟比较器输出 模拟比较器的输出经过同步后直接连到 ACO。

LED0_ON(); //如果 AIN0<AIN1(ACO=0),LED 亮

else

LED0_OFF(); //否则 LED 灭

delay_ms(200); //当电压差接近 0V 时，模拟比较器会产生临界抖动，故延时 200mS 令肉眼能看到
}

//call this routine to initialize all peripherals

void init_devices(void)
{
//stop errant interrupts until set up

CLI(); //disable all interrupts

port_init();

comparator_init();

MCUCR = 0x00;

GICR = 0x00;

TIMSK = 0x00; //timer interrupt sources

SEI(); //re-enable interrupts

//all peripherals are now initialized
}

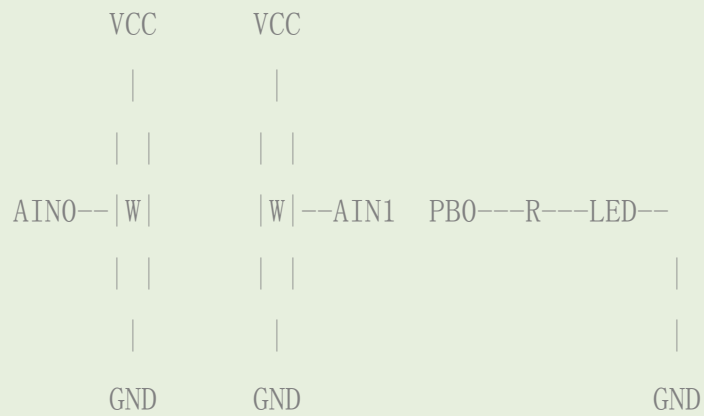
void main(void)
{
init_devices();

while(1)
;
}

```

/\*

程序测试：



两个电位器，一端接VCC,一端接地，构成电位器分压电路

AIN0 和AIN1 都分别接到电位器的中心抽头。

PB0输出串电阻驱动LED,高电平有效。

然后分别旋转电位器，增减抽头的电压，将会发现PB0 的输出(LED0)会根据 AIN0/AIN1 的电压关系变动

由于电源纹波，IO电流及外界干扰的影响，当电压差接近 0V时，模拟比较器会产生临界抖动，AIN0/AIN1 对地并上小电容可以改善这种情况。

只有一个电位器时，可以变通

1 可以使能ACBG，利用 1.23V内部能隙基准源代替AIN0 作模拟比较器的正输入端。

```
ACSR=(1<<ACIE)|(1<<ACBG);
```

2 可以使能ADC的内部 2.56V电压基准，然后把AIN0 或AIN1 连接到pin32 AREF脚。

```
ADCSRA=(1<<ADEN); //需要打开ADC
```

```
ADMUX=(1<<REFS1)|(1<<REFS0);
```

模拟比较器控制和状态寄存器—ACSR

Bit 7 – ACD: 模拟比较器禁用

模拟比较器上电默认是已经工作中的，跟其他的模块有所不同

**ACD**置位时，模拟比较器的电源被切断。可以在任何时候设置此位来关掉模拟比较器。

这可以减少器件工作模式及空闲模式下的功耗。

改变**ACD**位时，必须清零**ACSR** 寄存器的**ACIE**位来禁止模拟比较器中断。否则**ACD**改变时可能会产生中断

**Bit 6 – ACBG:** 选择模拟比较器的能隙基准源

**ACBG** 置位后，模拟比较器的正极输入由 1.23V能隙基准源所取代。否则， **AIN0** 连接到模拟比较器的正极输入。

**Bit 5 – ACO:** 模拟比较器输出

模拟比较器的输出经过同步后直接连到**ACO**。同步机制引入了 1-2 个时钟周期的延时。

**Bit 4 – ACI:** 模拟比较器中断标志

当比较器的输出事件触发了由**ACIS1** 及**ACIS0** 定义的中断模式时，**ACI** 置位。

如果**ACIE** 和**SREG** 寄存器的全局中断标志**I** 也置位，那么模拟比较器中断服务程序即得以执行，同时**ACI** 被硬件清零。

**ACI** 也可以通过写"1" 来清零。

**Bit 3 – ACIE:** 模拟比较器中断使能

当**ACIE** 位被置"1" 且状态寄存器中的全局中断标志**I** 也被置位时，模拟比较器中断被激活。

否则中断被禁止。

**Bit2 – ACIC:** 模拟比较器输入捕捉使能

这个功能用于检测一些微弱的触发信号源，节省一个外部运放。

**ACIC**置位后允许通过模拟比较器来触发**T/C1** 的输入捕捉功能。

此时比较器的输出被直接连接到输入捕捉的前端逻辑，从而使得比较器可以利用**T/C1** 输入捕捉中断逻辑的噪声抑制器及触发沿选择功能。

为了使比较器可以触发**T/C1** 的输入捕捉中断，定时器中断屏蔽寄存器**TIMSK** 的**TICIE1** 必须置位。

**ACIC** 为"0" 时模拟比较器及输入捕捉功能之间没有任何联系。

Bits 1, 0 – ACIS1, ACIS0: 模拟比较器中断模式选择

这两位确定触发模拟比较器中断的事件。

ACIS1 ACIS0 中断模式

0 0 比较器输出变化即可触发中断

0 1 保留

1 0 比较器输出的下降沿产生中断

1 1 比较器输出的上升沿产生中断

需要改变ACIS1/ACIS0 时，必须清零ACSR 寄存器的中断使能位来禁止模拟比较器中断。否则有可能在改变这两位时产生中断。

\*/

ICC avr下的工程文件下载: [avr\\_comp.rar](#)

## AVR 使用范例--AVR 外部中断范例

本页关键词： 单片机外部中断 外部中断的应用 AVR 外部中断 INTERUPTS

关于 AVR 中断：

系统在正常运行主程序时，如果突然有一个重要的任务要马上处理，那么系统就要保存现在的工作，然后再去处理这个任务，执行这个重要任务完毕以后再返回原来的主程序继续运行，这就是中断。

主程序一旦进入中断服务程序，那么AVR芯片将自动的关闭全局中断，在这个期间不再执行其它的中断请求，直到中断程序结束以后芯片才自动的重新开放全局中断。(注意，在这个期间某些中断请求可能会被丢弃，某些请求会留下中断请求标致，一旦当前的中断执行完毕，这个有中断标致的请求就有可能马上得到响应，如INT0的下降沿触发就会留下中断请求标致，而低电平触发就不会流下中断请求标致)。如果你想在执行中断服务程序时响应另外一个更重要的中断，那么就要在中断服务程序中加入一条打开全局中断的语句。

使用ICC快速建立中断服务程序 [程序文件](#)[txt]

使用 ICCAVR Application Builder 进行如下图所示的设置。可以设置是否使用中断，上升延，下降延，低电平，任意的逻辑电平变化。



将生成的程序进行修改，`DDRA = 0x01;PORTD = 0x0C;`，添加 MAIN 函数，如下

```

• //ICC-AVR application builder : 2006-12-8 17:04:44
• // Target : M16
• // Crystal: 7.3728Mhz
•
• #include <iom16v.h>
• #include <macros.h>
•
• unsigned int i=0;
•

```



```

• void port_init(void)
• {
•     PORTA = 0x00;
•     DDRA  = 0x01;
•     PORTB = 0x00;
•     DDRB  = 0x00;
•     PORTC = 0x00; //m103 output only
•     DDRC  = 0x00;
•     PORTD = 0x0C; //使 INT0, INT1 对应口上拉电阻有效
•     DDRD  = 0x00; //必须设置INT0, INT1 对应口为 输入
• }
•
• #pragma interrupt_handler int0_isr:2
• void int0_isr(void)
• {
•     //external interupt on INT0
•     i++; //在中断里进行操作
• }
•
• #pragma interrupt_handler int1_isr:3
• void int1_isr(void)
• {
•     //external interupt on INT1
•     PORTA = 0x01; //在中断里进行操作
• }
•
• //call this routine to initialize all peripherals
• void init_devices(void)
• {

```

```

• //stop errant interrupts until set up
• CLI(); //disable all interrupts
• port_init();
•
• MCUCR = 0x08; //INT1 的下降沿产生异步中断请求，INT0 上升延
• GICR = 0xC0; //INT0 和 INT1 使能
• TIMSK = 0x00; //timer interrupt sources
• SEI(); //re-enable interrupts
• //all peripherals are now initialized
• }
•
• void main(void)
• {
•   init_devices();
•   while(1) //死循环等待中断
•   ;
• }

```

#### 相关解释

1. `#pragma interrupt_handler int0_isr:2`
2. 中断的约定表示方法: `int0_isr` 有一个与之对应的应用程序 `void int0_isr(void)`, `2` 为中断向量, 值越小, 优先级越高, `INT0` 的优先级仅次于复位。
3. MCU 控制寄存器 — `MCUCR` MCU 控制寄存器包含中断触发控制位与通用 MCU 功能
4. 通用中断控制寄存器 — `GICR` 使能或禁用外部中断请求

使用过程中: 为了降低干扰, 中断引脚请加上一个 `4.7K` 的上拉电阻, 还可以根据实际需要, 加上电容滤波, 防抖动。

更详细的内容, AVR 外部中断, 时钟中断, 串行通信, IIC 综合实例 [一个中断示例程序\[txt\]](#)

//中断示例程序——此程序为 AVR 外部中断, 时钟中断, 串行通信, IIC 综合实例

```

#include <iom16v.h>
#include <macros.h>
#define uchar unsigned char
#define uint unsigned int
#pragma interrupt_handler int2:19//外部中断 2
#pragma interrupt_handler timer0_ovf_isr:10//时钟中断
#pragma interrupt_handler rec:12//串口接收中断
void delay(uint ticks);//延时
unsigned char const Tab[]={0x14, 0x9F, 0x38, 0x1A, 0x93, 0x52, 0x50, 0x1F,
                           0x10, 0x12, 0x11, 0xD0, 0x74, 0x98, 0x70, 0x71};
//数码管显示代码
unsigned int count=0;//软件记数
void main()
{

    uchar Address, date;
    int x=0, j=0;
    init_devices();
    //中断设置
    CLI();//关总中断
    //INT2 设置
    MCUCSR&=~0x40;//下降沿触发
    GIFR|=0x20;//清 INT2 标志
    GICR|=0x20;//开 INT2 中断
    //定时设置
    TCCR0 = 0x00; //stop
    TCNT0 = 0x83; //set count
    OCR0 = 0x7D; //set compare
    //TCCR0 = 0x03; //start timer
    TIMSK = 0x01; //timer interrupt sources
    //串口设置, 波特率:9600
    UCSRB = 0x00; //disable while setting baud rate
    UCSRA = 0x00;
    UCSRC = BIT(URSEL) | 0x06;
    UBRRL = 0x33; //set baud rate lo
    UBRRH = 0x00; //set baud rate hi
    //UCSRB = 0x08;
    UCSRB = (1<<RXCIE) | (1<<RXEN) | (1<<TXEN);
    SEI();//开总中断

    PORTA=0x80;
    PORTC=Tab[8];
    //DS3231 每秒产生一个中断:
    WriteDs3231_OneByte(0x0e, 0x07);

```

```

WriteDs3231_OneByte(7, 0x80); WriteDs3231_OneByte(8, 0x80);
WriteDs3231_OneByte(9, 0x80); WriteDs3231_OneByte(10, 0x81);
/*
while(1)
{
    //读出并显示:
    date=ReadDs3231_OneByte(j++); //读取数据
    if(j==7) j=0; //循环读出 00-06 地址的数据:秒、分、时、星期、日期、月和
    年信息
    for(x=0; x<200; x++)
    {
        PORTA=0x80;
        PORTC=Tab[date&0x0f]; //显示个位
        delay(50);
        PORTA=0x40;
        PORTC=Tab[date>>4]; //显示十位
        delay(50);
    }
    PORTA=0; //关闭显示
    delay(5000);
}
}*/
void delay(uint ticks)
{
    uchar i;
    while(ticks-->0) for(i=100; i!=0; i--); //约 0.1mS
}
void int2() //外部中断服务程序
{
    CLI();
    PORTC=Tab[ReadDs3231_OneByte(0)&0x0f];
    SEI();
}
void timer0_ovf_isr(void) //定时中断服务程序
{
    unsigned char data;
    TCNT0 = 0x83; //reload counter value
    if(++count==3000) //1000 个 1mS 等于 1 秒
    {
        count=0;
        data=ReadDs3231_OneByte(0)&0x0f;
        PORTC=Tab[data];
        printf("%dn", data);
    }
}
}

```

```
void rec()//接收中断服务程序
{
    unsigned char data;
    data=UDR;
    PORTC=data;
}
```

## AVR 使用范例--AVR 复位检测 AVR 复位电路 AVR 新手入门

AVR 复位时所有的 I/O 寄存器都被设置为初始值，程序从复位向量处开始执行。复位向量处的指令必须是绝对跳转 **JMP** 指令，以使程序跳转到复位处理例程。

AVR 的复位信号源有五个：

- 上电复位。电源电压低于上电复位门限 **VPOT** 时，MCU 复位。
- 外部复位。引脚 **RESET** 上的低电平持续时间大于最小脉冲宽度时 MCU 复位。
- 看门狗复位。看门狗使能并且看门狗定时器溢出时复位发生。
- 掉电检测复位。掉电检测复位功能使能，且电源电压低于掉电检测复位门限 **VBOT** 时 MCU 即复位。
- JTAG AVR 复位。复位寄存器为 1 时 MCU 复位。

### 1. 上电复位

上电复位(POR) 脉冲由片内检测电路产生，POR 电路保证器件在上电时复位。VCC 达到上电门限电压后触发延迟计数器。在计数器溢出之前器件一直保持为复位状态。当 VCC 下降时，只要低于检测门限，RESET 信号立即生效。

### 2. 外部复位

外部复位由外加于 **RESET** 引脚的低电平产生。当复位低电平持续时间大于最小脉冲宽度时即触发复位过程，即使此时并没有时钟信号在运行。当外加信号达到复位门限电压 **VRST**( 上升沿) 时，**tTOUT** 延时周期开始。延时结束后 MCU 即启动。

### 3. 看门狗复位

看门狗定时器溢出时将产生持续时间为 1 个 CK 周期的复位脉冲。在脉冲的下降沿，延时 定时器开始对 tTOUT 记数

### 4. 掉电检测复位

BOD 电路的开关由熔丝位 BODEN 控制。当 BOD 使能后(BODEN 被编程)，一旦 VCC 下降到 触发电平以下(VBOT-, Figure 19)， BOD 复位立即被激发。当 VCC 上升到触发电平以上 时 (VBOT+, Figure 19)，延时计数器开始计数，一旦超过溢出时间 tTOUT，MCU 即恢复工作。

### 5. JTAG AVR 复位

JTAG 通过复位寄存器 mcur，复位寄存器为 1 时 MCU 复位。通过 JTAG 指令 AVR\_RESET 可以使 JTAG 复位寄存器置位，并引发 MCU 复位，并使 JTRF 置位。上电复位将使其清零，也可以通过写"0" 来清除。

### 6. MCU 控制和状态寄存器提供了有关引起 MCU 复位的复位源的信息。

MCU 控制和状态寄存器提供了有关引起 MCU 复位的复位源的信息。

#### 1. Bit 4 – JTRF: JTAG 复位标志

通过 JTAG 指令 AVR\_RESET 可以使 JTAG 复位寄存器置位，并引发 MCU 复位，并使 JTRF 置位。上电复位将使其清零，也可以通过写"0" 来清除。

#### 2. Bit 3 – WDRF: 看门狗复位标志

看门狗复位发生时置位。上电复位将使其清零，也可以通过写"0" 来清除。

#### 3. Bit 2 – BORF: 掉电检测复位标志

掉电检测复位发生时置位。上电复位将使其清零，也可以通过写"0" 来清除。

#### 4. Bit 1 – EXTRF: 外部复位标志

外部复位发生时置位。上电复位将使其清零，也可以通过写"0" 来清除。

#### 5. Bit 0 – PORF: 上电复位标志

上电复位发生时置位。只能通过写"0" 来清除。

为了使用这些复位标志来识别复位条件，用户应该尽早读取此寄存器的数据，然后将其复位。如果在其他复位发生之前将此寄存器复位，则后续复位源可以通过检查复位标志来了解。

## AVR 使用范例--AVR 看门狗使用范例 avr wdt 看门狗详解

论坛连接，细说avr看门狗：<http://bbs.avrvi.com/read.php?fid=5&tid=1237&toread=1>

论坛连接，AVR看门狗使用范例：<http://bbs.avrvi.com/read.php?fid=5&tid=1234&toread=1>

AVR 看门狗 一个硬件单元，当程序由于某种原因跑“飞”了，它就 Reset 程序。就像小狗看门一样。

```
//Watchdog initialize
// prescale: 2048K
void watchdog_init(void)
{
WDR(); //this prevents a timeout on enabling
WDTCR = 0x0F; //WATCHDOG ENABLED - dont forget to issue WDRs
}
```

上面是用ICC的App Builder生成的看门狗初始化程序，**这些语句达不到初始化看门狗的目的，需要在中间加一句WDTCR = 0x1F;**。最后一行代码提醒狗主人，别忘了及时清零看门狗定时器（喂狗），否则，小狗就咬人了。

一个相对独立的计数自动重启单片机的硬件部件，如果启用它后，不在一定的时间内清除它的计数值，就会达到计数的最高值而溢出，然后它就指挥单片机重启。所以要在你的程序里适当的加入清看门狗的指令，一旦你的单片机程序出了问题，当然就不能按照你的程序原先设定那样自动清看门狗了，也就是常说的程序跑飞了，这个时候看门狗就会重启单片机试图解决问题。一般只对瞬间干扰造成的问题有效，要是长时间的干扰或是软硬件问题，看门狗的意义不是很大。

我的理解 就象是监视程序执行的保安一样，程序正常执行时会在他的益处时间之内给他一个复位信号,当程序跑飞的时候他在溢出时间之内是收不到复位信号的,这时看门狗就会在设定的时间内产生系统复位的信号!

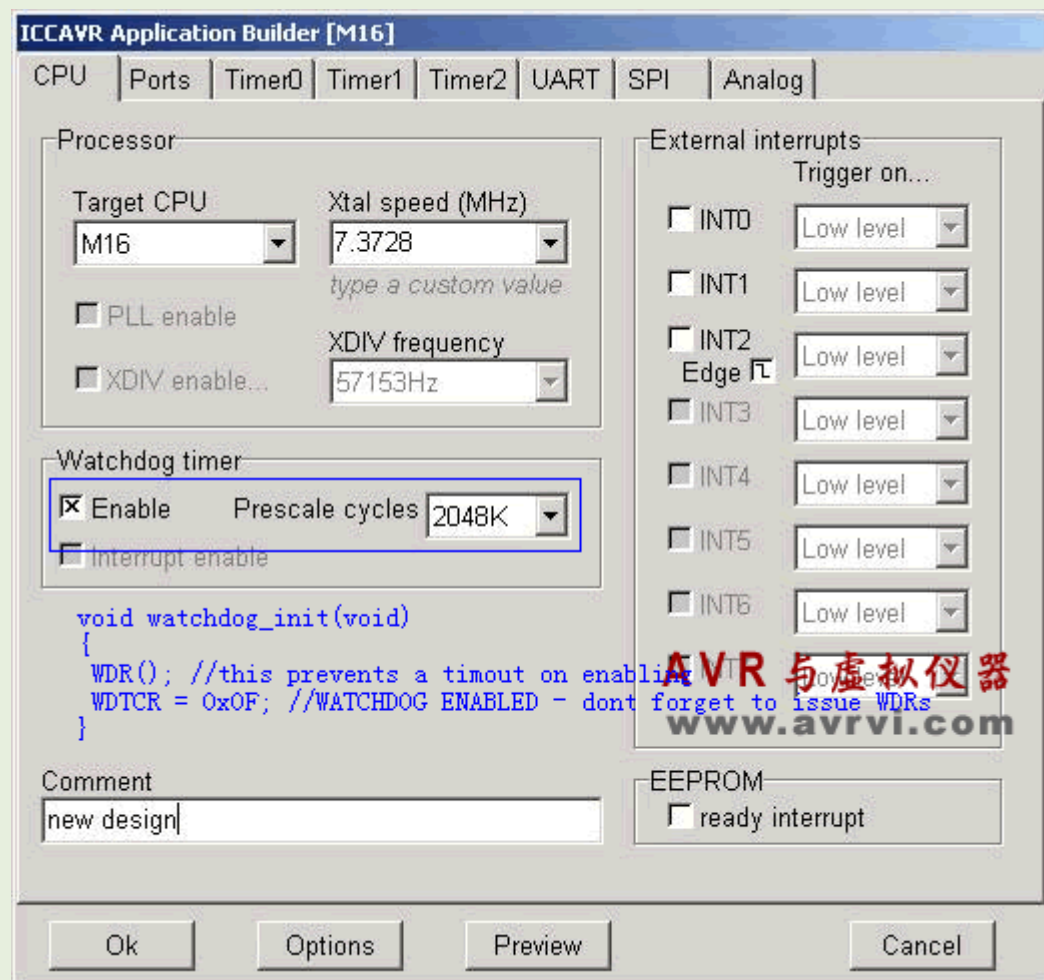
AVR 的看门狗是软狗，也是硬狗！ 如果熔丝位不设定，就是软狗，因为程序可以关闭，也可以打开 如果熔丝位设定了，就是硬狗，因为程序只可以清除，而无法打开或关闭！

是不是在程序中加入 `WDR();` 就算“喂狗”了？喂狗好象要计算好时间吧？我每执行一个函数就喂狗一次如何？

如果你的循环体内每循环一次的时间不超过看门狗的复位时间，主要喂狗一次就可以了。

**AVR看门狗程序范例**，程序演示了看门狗的复位过程，使用了本站新手入门的第一个范例，普通情况下，程序最后陷入死循环，但是这个程序里，看门狗让单片机复位，你会看见LED一直闪动，效果和第一个范例程序相同。

就两点，初始化，然后喂狗，喂狗要在看门狗咬人之前，（复位之前喂狗）。



//ICC-AVR application builder : 2007-1-31 17:27:04

// Target : M16

// Crystal: 7.3728Mhz

// Author: 古欣



```
// 看门狗复位演示程序。

#include <iom16v.h>
#include <macros.h>

void port_init(void)
{
    PORTA = 0x03; //设置为输出
    DDRA  = 0x03; //高电平，两个LED都灭
    PORTB = 0x00;
    DDRB  = 0x00;
    PORTC = 0x00; //m103 output only
    DDRC  = 0x00;
    PORTD = 0x00;
    DDRD  = 0x00;
}

//Watchdog initialize
// prescale: 2048K //预分频越大，定时时间越长，也就是可以更长时间不喂
// 约为 2.1s 复位 （根据数据手册，2048K，5V 典型值）
void watchdog_init(void)
{
    WDR(); //this prevents a timeout on enabling
    WDTCR = 0x1F; //特别注意这一条不是ICC生成的，是后来加上的。
    WDTCR = 0x0F; //WATCHDOG ENABLED - dont forget to issue WDRs
}

//加入了喂狗的延时程序
void Delay(void)
{

```

```

unsigned char i, j;
for(i=200; i>0; i--)
{
    for(j=200; j>0; j--)
    ;
}
WDR();    //这里喂狗
}

//call this routine to initialize all peripherals
void init_devices(void)
{
    //stop errant interrupts until set up
    CLI(); //disable all interrupts
    port_init();
    watchdog_init();

    MCUCR = 0x00;
    GICR  = 0x00;
    TIMSK = 0x00; //timer interrupt sources
    SEI(); //re-enable interrupts
    //all peripherals are now initialized
}

void main(void)
{
    unsigned int i;
    init_devices();    //初始化

```

```

for(i=10;i>0;i--)    //看到程序的闪动
{
    PORTA = 0x02;      //1 脚为高，0 脚为低，0 脚灯亮
    Delay();           //延时
    PORTA = 0x01;      //0 脚为高，1 脚为低，1 脚灯亮
    Delay();           //延时
}

while(1) //普通情况下，程序会陷入这里一直循环。
;        //看门狗能够让单片机复位，程序重新运行，我们看到 LED 闪烁。
        //如果在这里加入 WDR(); 喂狗，单片机就不会复位了。
}

```

补充，特别说明

使能看门狗不能用 |=，必须要直接赋值=。

还有数据手册上说：

改变定时器溢出时间及禁止(已经使能的)看门狗定时器需要执行一个特定的时间序列：

1. 在同一个指令内对 WDCE 和 WDE 写"1"，即使 WDE 已经为"1"。
2. 在紧接的 4 个时钟周期之内将 WDE 和 WDP 设置为合适的值，而 WDCE 写"0"。

所以在 WDTCSR=0x0E;之前加上一句 WDTCSR=0x1F;

```
void watchdog(void)
```

```
{
```

```
WDR(); //看门狗计数清零
```

```
WDTCSR=0x1F; //使能 watchdog,并且，采用 2048K 分频，典型溢出时间 5V 时 2.1S
```

```
WDTCSR=0x0F; //使能 watchdog,并且，采用 2048K 分频，典型溢出时间 5V 时 2.1S
```

```

}

initial_WDT:

    WDR                      ;2006-09-22 增加看门狗

    LDI    R16, $1F

    OUT    WDTCR, R16        ;使能 watchdog, 并且, 采用 2048K 分频, 典型
溢出时间 5V 时 2.1S

    LDI    R16, $0F

    OUT    WDTCR, R16

```

### 细说AVR看门狗

AVR看门狗使用教程：[http://www.avrvi.com/avr\\_examples/avr\\_wdt.html](http://www.avrvi.com/avr_examples/avr_wdt.html)

一个硬件单元，当程序由于某种原因跑“飞”了，它就Reset程序。就像小狗看门一样。

```

//Watchdog initialize
// prescale: 2048K
void watchdog_init(void)
{
    WDR(); //this prevents a timeout on enabling
    WDTCR = 0x0F; //WATCHDOG ENABLED - dont forget to issue WDRs
}

```

上面是用ICC的App Builder生成的看门狗初始化程序。最后一行代码提醒狗主人，别忘了及时清零看门狗定时器（喂狗），否则，小狗就咬人了。

---

一个相对独立的计数自动重启单片机的硬件部件，如果启用它后，不在一定的时间内清除它的计数值，就会达到计数的最高值而溢出，然后它就指挥单片机重启。

所以要在你的程序里适当的加入清看门狗的指令，一旦你的单片机程序出了问题，当然就不能按照你的程序原先设定那样自动清看门狗了，也就是常说的程序跑飞了，这个时候看门狗就会重启单片机试图解决问题。一般只对瞬间干扰造成的问题有效，要是长时间的干扰或是软硬件问题，看门狗的意义不是很大。

---

---

我的理解 就象是监视程序执行的保安一样,  
程序正常执行时会在他的益处时间之内给他一  
个复位信号,当程序跑飞的时候他在溢出时间之内是收不到复  
位信号的,这时看门狗就会在设定的时间内产生系统复位的信号!

---

AVR的看门狗是软狗，也是硬狗！

如果熔丝位不设定，就是软狗，因为程序可以关闭，也可以打开  
如果熔丝位设定了，就是硬狗，因为程序只可以清除，而无法打开或关闭！

---

是不是在程序中加入

WDR();

就算“喂狗”了？喂狗好象要计算好时间吧？我每执行一个函数就喂狗一次如何？

如果你的循环体内每循环一次的时间不超过看门狗的复位时间，主要喂狗一次就可以了。

#### AVR看门狗程序范例 制作完成

AVR看门狗程序范例，程序演示了看门狗的复位过程，使用了本站新手入门的第一个范例，普通情况下，程序最后陷入死循环，但是这个程序里，看门狗让单片机复位，你会看见LED一直闪动，效果和第一个范例程序相同。

就两点，初始化，然后喂狗，喂狗要在看门狗咬人之前，（复位之前喂狗）。

CODE:

```
//ICC-AVR application builder : 2007-1-31 17:27:04
// Target : M16
// Crystal: 7.3728Mhz
// Auther: 古欣 AVR 与虚拟仪器 [url]www.avrvi.com[/url]
// 看门狗复位演示程序。
#include <iom16v.h>
#include <macros.h>
```

```
void port_init(void)
```

```

{
PORTA = 0x03; //设置为输出
DDRA = 0x03; //高电平，两个 LED 都灭
PORTB = 0x00;
DDRB = 0x00;
PORTC = 0x00; //m103 output only
DDRC = 0x00;
PORTD = 0x00;
DDRD = 0x00;
}

//Watchdog initialize
// prescale: 2048K //预分频越大，定时时间越长，也就是可以更长时间不喂 约为 2.1s 复位
void watchdog_init(void)
{
WDR(); //this prevents a timeout on enabling
WDTCSR = 0x1F; //特别注意这一条不是 ICC 生成的，是后来加上的。
WDTCSR = 0x0F; //WATCHDOG ENABLED - dont forget to issue WDRs
}

//加入了喂狗的延时程序
void Delay(void)
{
{
unsigned char i,j;
for(i=200;i>0;i--)
{
for(j=200;j>0;j--)
;
}
CLI(); //这里喂狗
}

//call this routine to initialize all peripherals
void init_devices(void)
{
//stop errant interrupts until set up
CLI(); //disable all interrupts
port_init();
watchdog_init();

MCUCR = 0x00;
GICR = 0x00;
TIMSK = 0x00; //timer interrupt sources

```

```

SEI(); //re-enable interrupts
//all peripherals are now initialized
}

void main(void)
{
unsigned int i;
init_devices();    //初始化

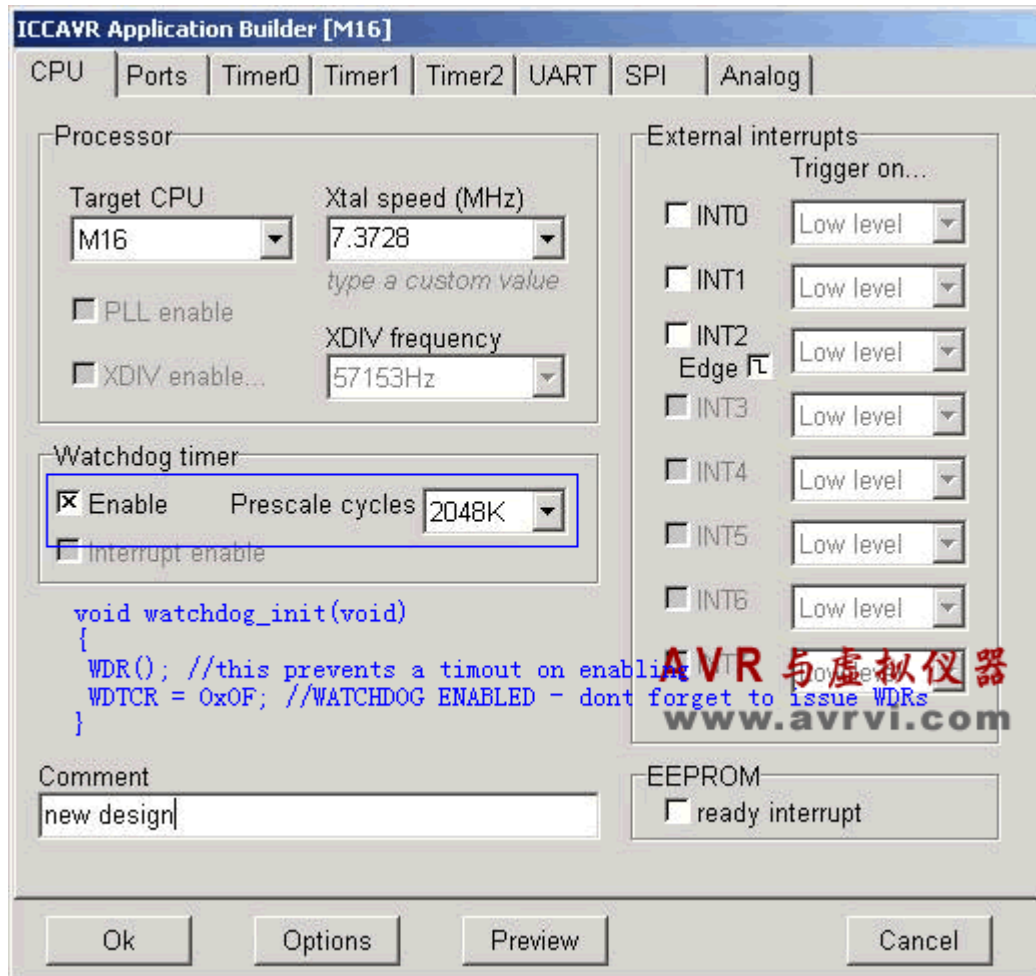
for(i=10;i>0;i--)  //看到程序的闪动
{
PORTA = 0x02;      //1 脚为高，0 脚为低，0 脚灯亮
Delay();           //延时
PORTA = 0x01;      //0 脚为高，1 脚为低，1 脚灯亮
Delay();           //延时
}

while(1) //普通情况下，程序会陷入这里一直循环。
;        //看门狗能够让单片机复位，程序重新运行，我们看到 LED 闪烁。
        //如果在这里加入 WDR(); 喂狗，单片机就不会复位了。
}

```

[\[Copy to clipboard\]](#)

**看门狗程序ICC生成界面**



补充，特别说明

使能看门狗不能用 |=，必须要直接赋值=。

还有数据手册上说：

改变定时器溢出时间及禁止(已经使能的)看门狗定时器需要执行一个特定的时间序列：

1. 在同一个指令内对WDCE 和WDE 写"1"，即使WDE 已经为"1"。
2. 在紧接的 4 个时钟周期之内将WDE 和WDP 设置为合适的值，而WDCE 写"0"。

所以在WDTCSR=0x0E;之前加上一句WDTCSR=0x1F;

CODE:

```
void watchdog(void)
{
    WDR();    //看门狗计数清零
    WDTCSR=0x1F; //使能 watchdog,并且，采用 2048K 分频，典型溢出时间 5V 时 2.1S
    WDTCSR=0x0F; //使能 watchdog,并且，采用 2048K 分频，典型溢出时间 5V 时 2.1S
}
```

[\[Copy to clipboard\]](#)



这一段汇编效果很好，供参考

initial\_WDT:

WDR ; 2006-09-22 增加看门狗

LDI R16, \$1F

OUT WDTCR,R16 ;使能watchdog,并且,采用 2048K分频,典型溢出时间 5V  
时 2.1S

```
LDI    R16, $0F
```

```
OUT    WDTCR,R16
```

## AVR 使用范例--EEPROM 使用详解

本页关键词: 什么是 eeprom spi eeprom eeprom 程序 eeprom 资料 eeprom 结构 eeprom 读写

本页详细介绍ICC自带EEPROM操作函数的操作方法，包括 **单字符** 读写，**数组** 读写，**结构体** 读写。

程序代码: [下载相关文件](#)

- `void main(void)`
- `{`
- `unsigned char temp1,temp2; /*`  
定义变量\*/
- `unsigned char buffer[10]; /*`  
定义数组\*/
- `unsigned char buf[]="AVR与虚拟仪器"; /*`  
定义字符串\*/
- 
- `EEPROMwrite(0x10, 'a'); /*`  
单字符写入到 0x10, 注意是单引号\*/

```

•   temp1 = EEPROMread(0x10);                                /*
    读一个字符到temp1*/

•

•

•   EEPROM_WRITE(0x20, "abcdefg");                            /*
    写字符串到 0x20*/

•   EEPROM_READ(0x20, temp2);                                  /*
    读 字符 到temp2, temp2=a*/

•   EEPROM_READ(0x20, buffer);                                /*
    读 字符串 到数组中 buffer[10]=abcdefg */

•

•   EEPROM_WRITE(0x30, buf);                                    /*
    数组中的值写到EEPROM中: 0x30 开始为"AVR与虚拟仪器"*/

•

•   while(1)

•   ;

•   }

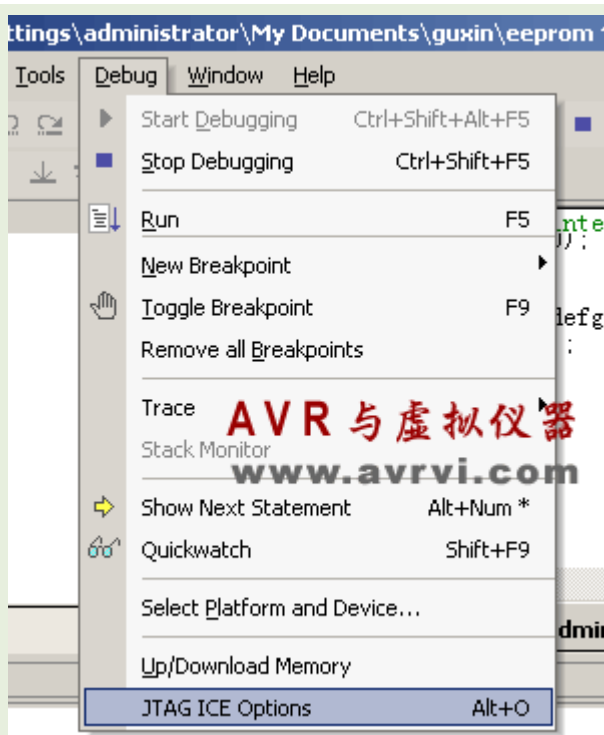
•

```

调试后的效果:

调试eeprom的时候, 记住设置Avr studio保护eeprom数据, 否则每次都会将eeprom中的数据改为0xFF。如下图:

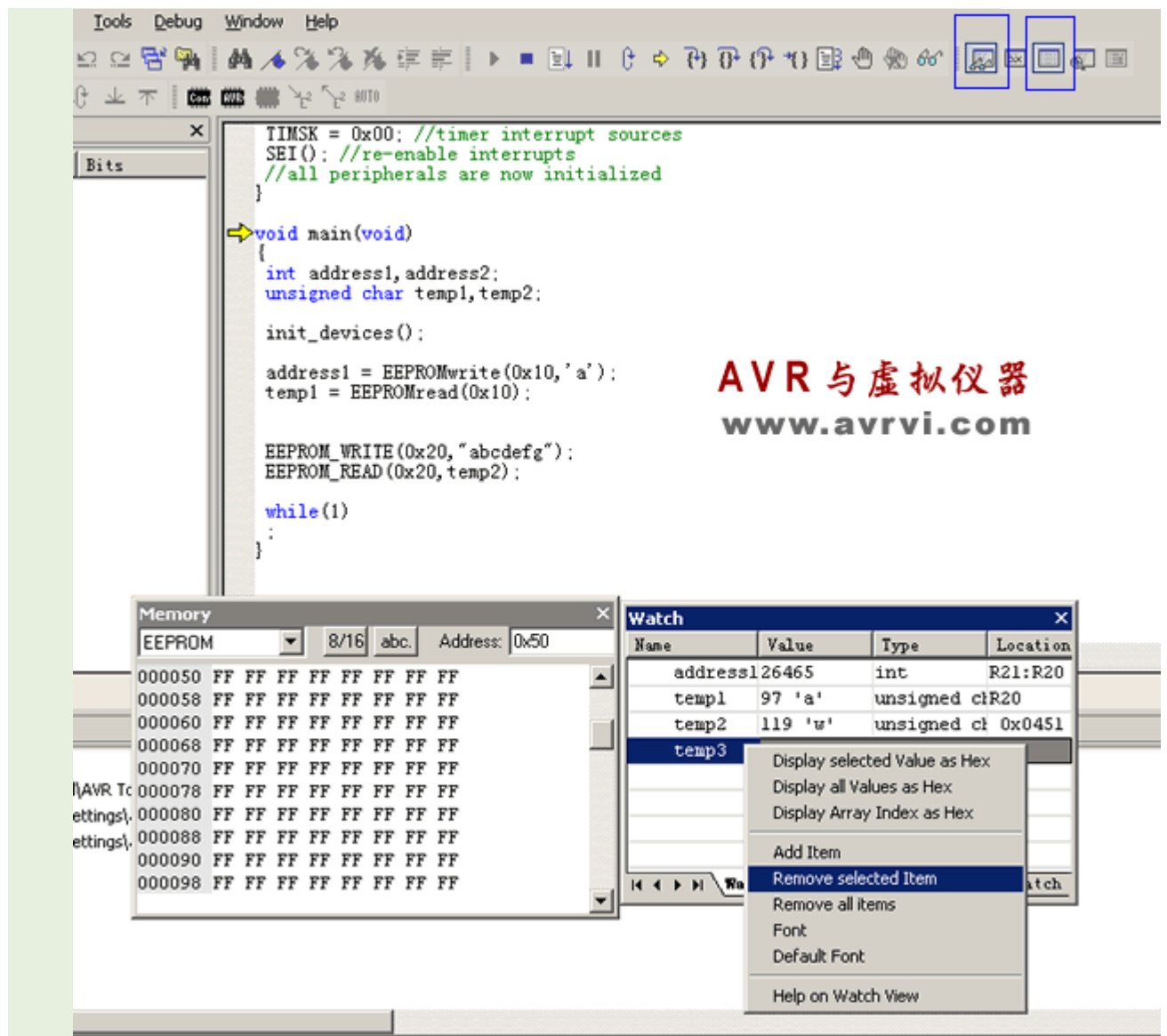
打开调试选项:



勾选保护 eeprom 数据选项：

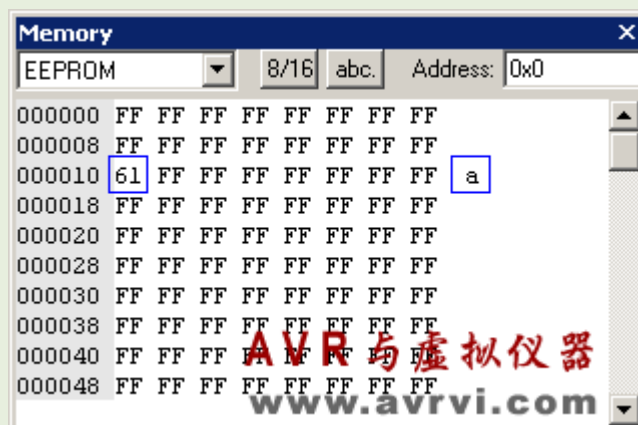


打开相关观测窗口：



按 F10 逐条语句运行，查看运行结果。

1. 地址 10 上的值被改写为“a”



2. temp1 的值变为 a，及读取了地址 0x10 的值 a:

Watch			
Name	Value	Type	Location
address126465		int	R21:R20
temp1	97 'a'	unsigned char	R20
temp2	103 'g'	unsigned char	0x044E

AVR 与虚拟仪器  
www.avrvi.com

Watch 1 Watch 2 Watch 3 Watch

3. abcdefg 写入 20 开始的地址:

Memory	
EEPROM	8/16 abc Address: 0x0
000000	FF FF FF FF FF FF FF FF
000008	FF FF FF FF FF FF FF FF
000010	61 FF FF FF FF FF FF FF a
000018	FF FF FF FF FF FF FF FF
000020	61 62 63 64 65 66 67 00 abcdefg.
000028	FF FF FF FF FF FF FF FF
000030	FF FF FF FF FF FF FF FF
000038	FF FF FF FF FF FF FF FF
000040	FF FF FF FF FF FF FF FF
000048	FF FF FF FF FF FF FF FF

AVR 与虚拟仪器  
www.avrvi.com

4. temp2 的变为地址 20 的值 a:

Watch			
Name	Value	Type	Location
address126465		int	R21:R20
temp1	97 'a'	unsigned char	R20
temp2	97 'a'	unsigned char	0x044E

Watch 1 Watch 2 Watch 3 Watch

5. 以 0x20 开始的值都读入 buffer 数组中:

Watch			
Name	Value	Type	Location
address1	26465	int	R21:R20
templ	97 'a'	unsigned char	R20
temp2	97 'a'	unsigned char	0x0444
buffer	[...]	unsigned char	0x0445
[0]	97 'a'	unsigned char	0x0445
[1]	98 'b'	unsigned char	0x0446
[2]	99 'c'	unsigned char	0x0447
[3]	100 'd'	unsigned char	0x0448
[4]	101 'e'	unsigned char	0x0449
[5]	102 'f'	unsigned char	0x044A
[6]	103 'g'	unsigned char	0x044B
[7]	0 ''	unsigned char	0x044C
[8]	255 ''	unsigned char	0x044D
[9]	255 ''	unsigned char	0x044E

6. 预定义的数组中的值写到 EEPROM 中:

Memory	
EEPROM	8/16 abc. Address: 0x0 Cols: Auto
000000	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000010	61 FF FF FF FF FF FF FF FF FF FF FF FF FF FF a
000020	61 62 63 64 65 66 67 00 FF FF FF FF FF FF FF FF abcdefg.
000030	41 56 52 D3 EB D0 E9 C4 E2 D2 C7 C6 F7 00 FF FF AVR与虚拟仪器.
000040	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF

7. 设置不保护 EEPROM 的值之后, 停止调试再开始调试, EEPROM 中的值改为 0xFF:

Memory	
EEPROM	8/16 abc. Address: 0x0 Cols: Auto
000000	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000010	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000020	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000030	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000040	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF

通过以上的观察, 我想你已经对 EEPROM 的操作有了一定的认识, 同时, 你可以打开 `eeeprom.h` 查看内部内容, 是如何定义函数的。

结构体的操作方法与数组类似: (一般简单应用中, 你用不到下面的知识, 看不懂没有关系, 不要被吓倒了)

- EEPROMWriteBytes(0x10,&red\_flag,sizeof(red\_flag)); //将结构写入 EEPROM
- EEPROMReadBytes(0x10,&red\_flag,sizeof(red\_flag)); //从 EEPROM 中将结构全部读出来
- 旗帜的结构为
- /\*
- typedef struct
- {
- uint8 position; //0 底 1~180: 中间 180: 顶
- uint8 fangxiang; //0 静止中 'u' : 向上 'd': 向下
- uint8 T; //0 总时间
- uint8 t; //0 已经耗去的时间
- uint8 S; //0 目标路程
- uint8 s; //0 已经走过的路程
- uint8 move\_flag; //0 是否运动中
- uint8 half\_mode; //0 非半旗 1: 半旗
- uint8 purse; //0 没有暂停 1: 暂停状态
- uint8 half\_short\_or\_long; //0 1: 半旗长路程中 2: 半旗短路程中
- uint16 maichong; //0 脉冲数目
- }QIZHI;
- QIZHI red\_flag; //旗帜结构
- \*/

## AVR 使用范例--使用 Bootloader 升级芯片内程序详解

感谢网站热心会员 likeeavr 提供本页原型。

Bootloader 是 flash 中高地址的一个程序区域，通过它可以实现程序的运行控制，程序更新等。本页给出的是通过串口实现在线更新的方法。程序编译环境:icc avr 6.31。

如果你使用Atmega16 芯片，7.3728M晶振，1024 的bootloader区，19200 的波特率，那么你直接使用下面的hex文件就可以了，[下载hex文件](#)。

本站的开发板内，预制这个hex文件，可以按照本页的方法用串口线直接更新程序。

否则，你需要对程序进行调整，然后重新编译Bootloader的hex文件，[下载软件包](#)，**软件中需要更改的部分**：

1. 芯片的类型

```
#include <iom16v.h> //选择对应的芯片
```

2. Bootloader 区的大小，和 ICC 的 option 相对应。

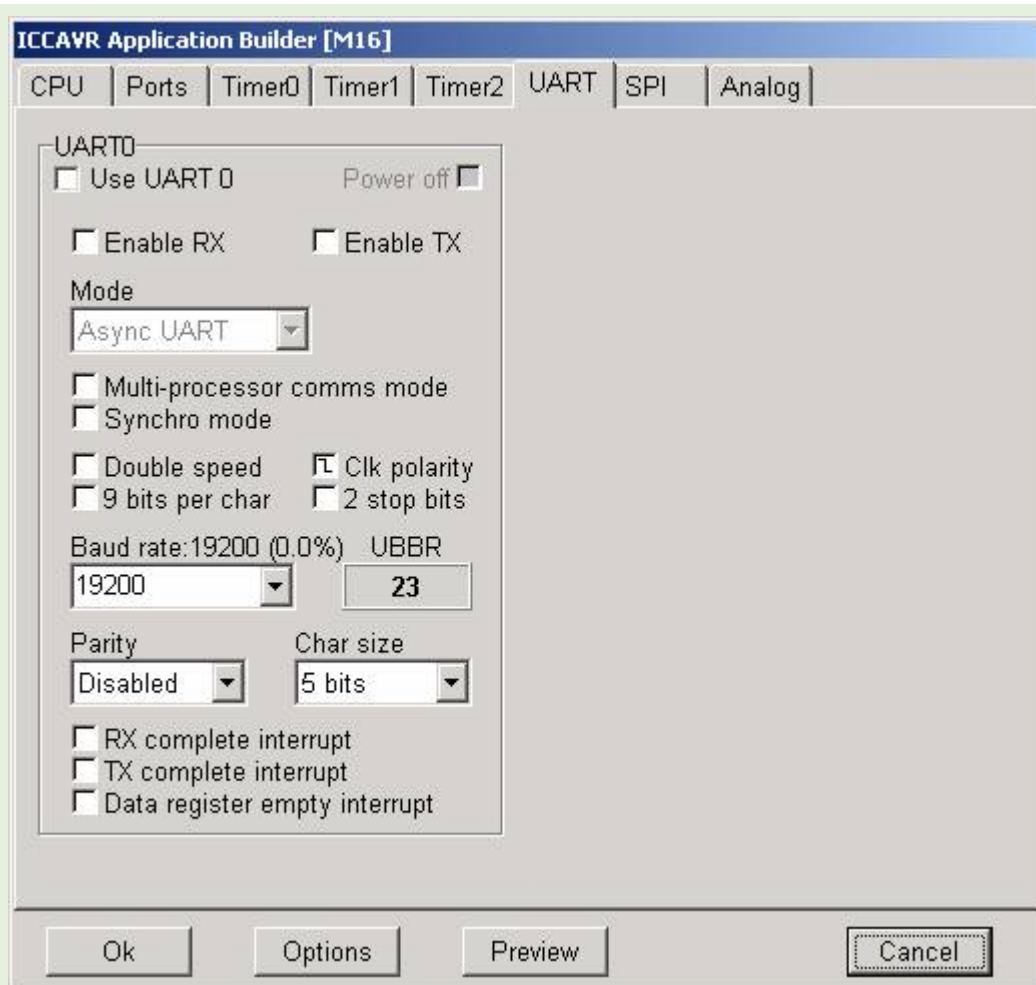
```
#define BootSize 'd' //1024
```

3. 串口的通讯频率设置，根据晶振大小对应的进行修改。

```
#define BAU 23 // 7.3728M 19200（计算方法如下图）
```

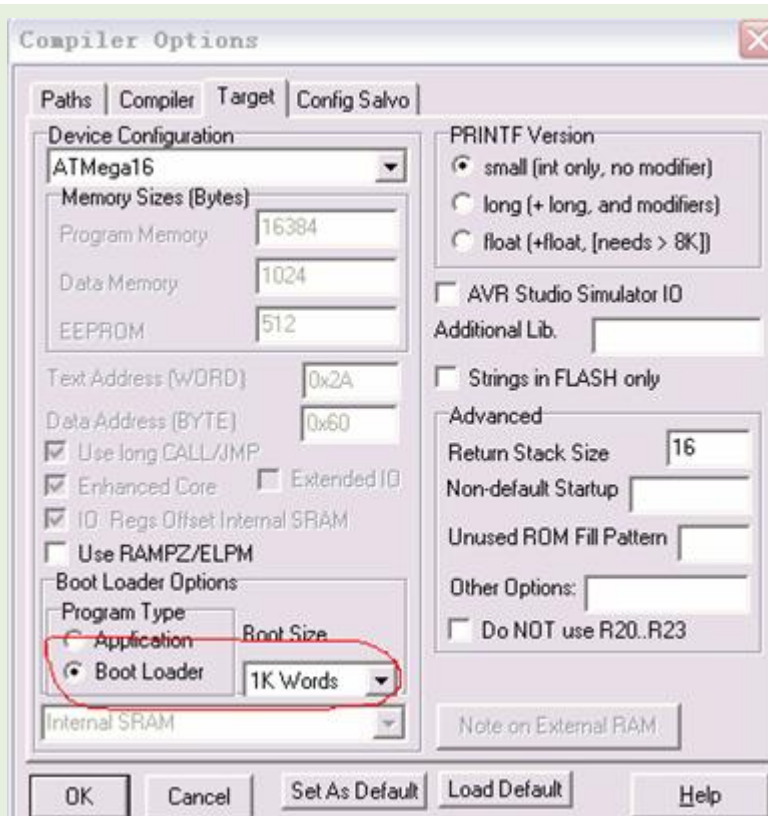
下面这个界面计算出 UBBR 的值正确的前提是 CPU 选项中的芯片类型和晶振都选择正确。





程序修改之后，进行如下操作，操作详细步骤说明：

- 1、在 ICC 中设置设置 **bootloader** 选项。

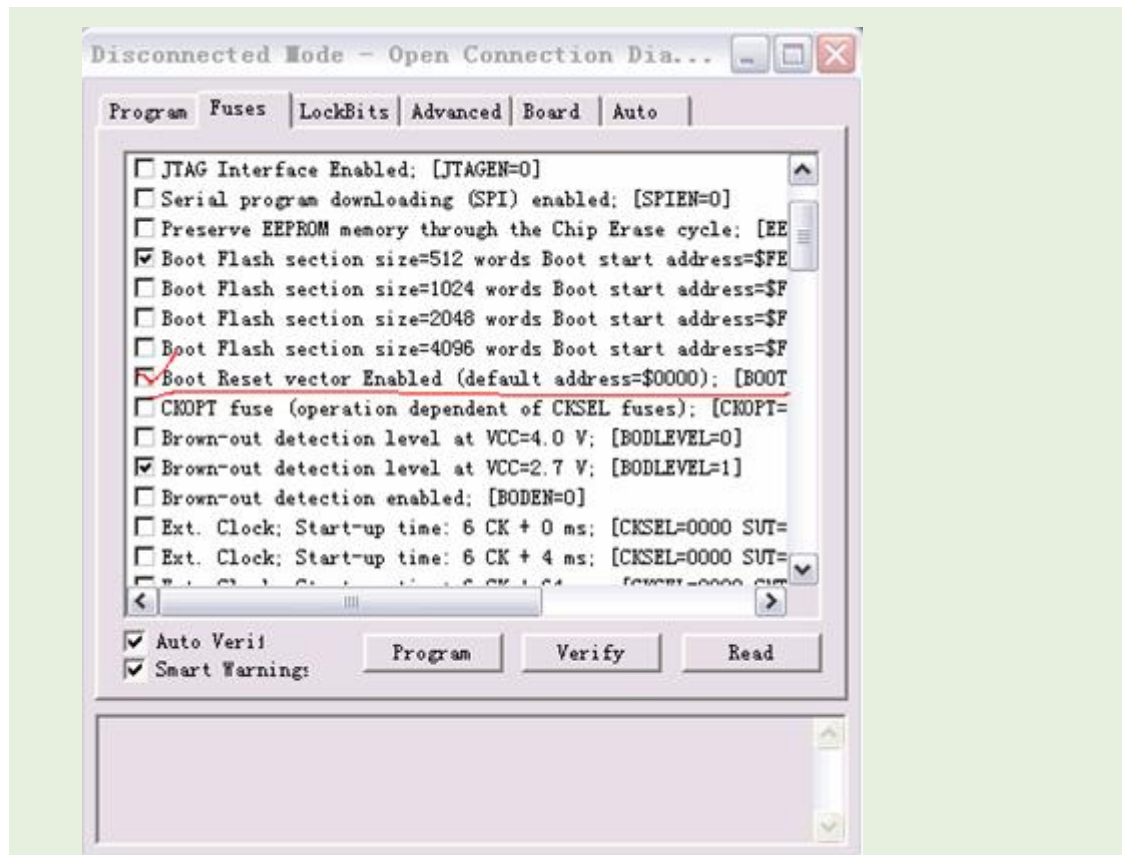


2、编译，这里很容易出问题，因为你的 ICC 的安装路径，以及下载后存放文件的路径可能会不同，所以刚开始的时候编译可能会出错，解决的方法，Projects—>Options —>Paths 把路径修改为正确的路径，然后再编译。

3、用 ISP 下载线或 JTAG 调试器把生成的 hex 文件烧写到单片机里。

校验芯片ID码:1E 94 03 完成!擦除.....完成!  
写入FLASH.....校验FLASH.....完成!

4、配置熔丝，需要用 ISP 或者 JTAG 来完成。



其他熔丝位请参考其他文献，在此不作详诉。Boot loader 在 flash 中对应的是高位字节，选中此熔丝的目的是让程序在复位时从 boot loader 开始运行，而不是 0x0000。（此处再配置熔丝选择 BOOTSZ=1024）

然后打开 AVR.EXE

上位机操作说明：

- 1、确保所要升级程序的单片机与 PC 机的 RS232 接口连接正常；
- 2、打开本软件，选定所使用的通讯端口，选定波特率（本软件默认为 19200bps，对应单片机使用 7.3728MHz 晶振）
- 3、点击“打开串口”按钮；

4、请您复位单片机或断电后重新启动单片机系统，此时在芯片信息栏里会出现您所操作的单片机的相关信息；

其中版本可能显示与实际版本不符，但不影响使用；

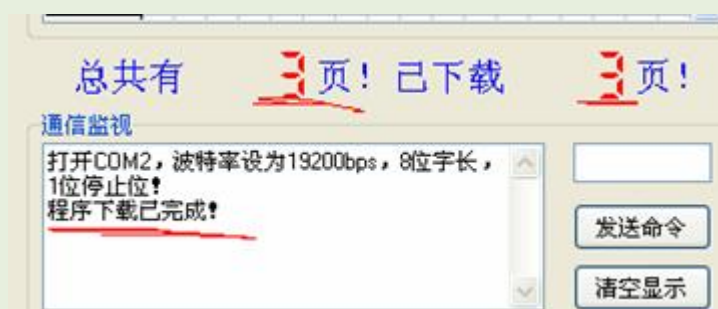


5、点击“载入文件”按钮，将您要更新的程序载入，完成后会在数据区正下方出现数据占用的页数； 每次都需重新点击“载入文件”选择相应的 hex 文件，不然下一步无法继续。



6、点击“下载程序”按钮开始下载，测试在总页数旁边会实时显示已经完成的页数，请耐心等待；

7、当所有程序都更新完后，实时显示的页数与总页数相等，并在信息框中会显示“程序下载已完成”；



8、点击“退出 BOOT”按钮，此时单片机跳转到 0x0000 开始运行从串口写入的程序，更新完毕。

想写入下一个程序时，再次点载入文件，然后复位一下单片机，就可以再出现第 5 步之后的界面了。

说明：上位机借用了别人的软件，此举是为了大家共同学习所用。在单片机复位，如接受不到上位机发出的命令，则自动转到应用程序区运行。

Ps: 此 bootloader 是由上位机通过串口发出命令控制，在实际的操作中完全可以模拟此程序写出满足自己要求的逻辑炸弹或相应的更新软件。自己做的测试程序 小灯 可以正常显示。



### 单片机入门系列--MEGA端口操作

说明：本节重点介绍真正双向端口操作的方法，及与伪双向端口操作的不同。跑马灯例子。建议先看跑马灯，再绕回来看前面的介绍。

AVR端口是真正双向端口，不像 51 伪双向。这也是AVR的一项优势，只是操作时大家注意DDRn就可以了。真正双向端口在模拟时序方面不如伪双向的方便。

DDRn PORTn PINn 解释：n为端口号：ABCDE

DDRn：控制端口是输入还是输出，0 为输入，1 为输出。个人记忆方法：一比零大所以往外挤，即 1 为输出，0 为输入。

PORTn：从引脚输出信号，当DDRn为 1 时，可以通过PORTn=x等端口操作语句给引脚输出赋值。

PINn：从引脚读输入信号，无论DDRn为何值，都可以通过x=PINn获得端口n的外部电平。

当引脚配置为输入时，若PORTxn 为"1"，上拉电阻将使能。内部上拉电阻的使用在键盘扫描的时候还要说到。

端口更详细功能及介绍以及端口第二功能请参考数据手册。

端口引脚配置

DDxn	PORTxn	PUD (in SFIOR)	I/O	上拉电阻说明
------	--------	----------------	-----	--------

0	0	X	输入	No 高阻态 (Hi-Z)
---	---	---	----	---------------



0	1	0	输入	Yes被外部电路拉低时将输出电流
0	1	1	输入	No高阻态(Hi-Z)
1	0	X	输出	No输出低电平 (漏电流)
1	1	X	输出	No输出高电平 (源电流)

如果有引脚未被使用，建议给这些引脚赋予一个确定电平。最简单的保证未用引脚具有确定电平的方法是使能内部上拉电阻。但要注意的是复位时上拉电阻将被禁用。如果复位时的功耗也有严格要求则建议使用外部上拉或下拉电阻。不推荐直接将未用引脚与VCC 或GND 连接，因为这样可能会在引脚偶然作为输出时出现冲击电流。

下面我们来看例子：

```
void port_init(void)
{
    PORTA = 0x03;
    DDRA = 0x03;
    PORTB = 0x00;
    DDRB = 0x01;
    PORTC = 0x00;
    DDRC = 0x00;
    PORTD = 0x00;
    DDRD = 0x00; // 建议赋值为零
}
```

PORTA = 0x03; DDRA = 0x03; 这两句使PA口的PA1 和PA0 处于输出状态，PA7—PA2 处于输入状态。这里的 0x03 即二进制的 00000011，从左到右对应于Pn7--Pn0 八个IO口。

通过跑马灯程序来深入理解IO口的操作：

**CODE:**

```
//ICC-AVR application builder : 2006-11-21 9:20:57
// Target : M32
// Crystal: 7.3728Mhz
```

```
#include <iom32v.h>
#include <macros.h>
```

```
void _delay(unsigned char n) //延时函数定义
{
    unsigned char i,j;
    for(; n!=0; n--) //n*10ms
    {
        for(j=100; j!=0; j--) //100us*100=10ms
        {
            for(i=147; i!=0; i--) //delay 100us
            ;
        }
    }
}
```

```

}
}
}

int main(void)
{
    unsigned char i,j,k; //
    PORTA=0xFF;          //PA 口设为输出高电平，灯灭
    DDRA=0xFF;           //PA 口设置为输出
    while(1)
    {
        i=1;
        for (j=0;j<8;j++) //循环 8 次，即 PA0~~PA7 轮流闪亮
        {
            PORTA=~i;      //反相输出,低电平有效,对应的灯亮
            for (k=0;k<10;k++) _delay(100);    // 延时 100*10=1 秒，可自行调
节            i=i<<1;      //左移一位,i 的值将向下面的列表那样变化
            // 0b00000001 PA0
            // 0b00000010 PA1
            // 0b00000100 PA2
            // 0b00001000 PA3
            // 0b00010000 PA4
            // 0b00100000 PA5
            // 0b01000000 PA6
            // 0b10000000 PA7
        }
    }
}

```

[\[Copy to clipboard\]](#)

其他IO口操作指令：

```

void main(void)
{
    PORTA=0xff;
    DDRA=0xff; //输出 模式，IO口上拉电阻有效，1 为输出，0 为输入。
    PORTA=0xf0; //等
    以下三条指令只对操作符号右边的数字位是一的位操作。
    PORTA&=~0x70; //清零 0x70 为 01110000，即把 654 三位清零，其余数位不变。
    PORTA|=0x77; //置一 0x77 为 01110111，即把 654210 六位清零，其余数位不变。
    PORTA^=0x70; //翻转 0x70 为 01110000，即 654 三位，如果是零变成 1，是一变成 0。
    (P & 0x80)==0x80; //按位与 判断p的第七位是否是一,是则成立
}

```



关于  $1 < x$  的说明，网上的程序中经常会看到  $1 < ADIF$  类似的语句，新手很难看明白是什么意思，我这里简单说明一下：

ADIF 是一个寄存器变量，可以堪称数字 4，跟手册中的定义，包含芯片头文件的定义是一样的。

$(1 < ADIF) = (1 < 4) = 0b00010000$

$ADCSR = (1 < ADIF);$  //只是 ADIF 位 = 1, 其他 = 0

$ADCSR |= (1 < ADIF);$  //只是 ADIF 位 = 1, 其他不变

$ADCSR \&= \sim(1 < ADIF);$  //只是 ADIF 位 = 0, 其他不变

$\text{while}(ADCSR \& (1 < ADIF));$  //等待 ADIF 位为 0，才退出循环，执行下一步

```
while(1)
```

```
{
```

```
while(ADCSR & (1 < ADIF)); //等待 ADIF 位为 0，才退出循环，执行下一步
```

```
{
```

```
程序.....
```

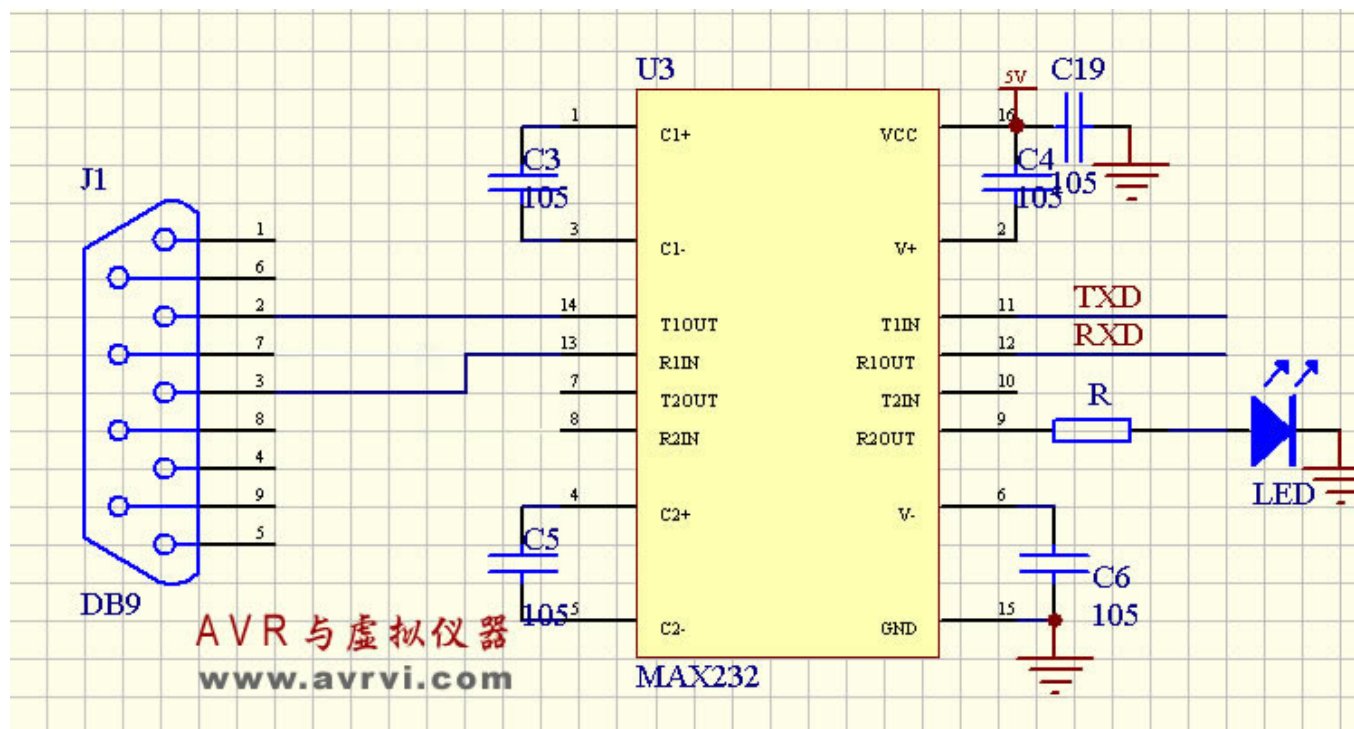
```
}
```

```
}
```

#### 个简单易懂的串口通讯例子程序

只有基本的功能，用于串口操作的演示，如果需要结构化的程序，到 [http://www.avrvi.com/avr\\_examples/usart.html](http://www.avrvi.com/avr_examples/usart.html)

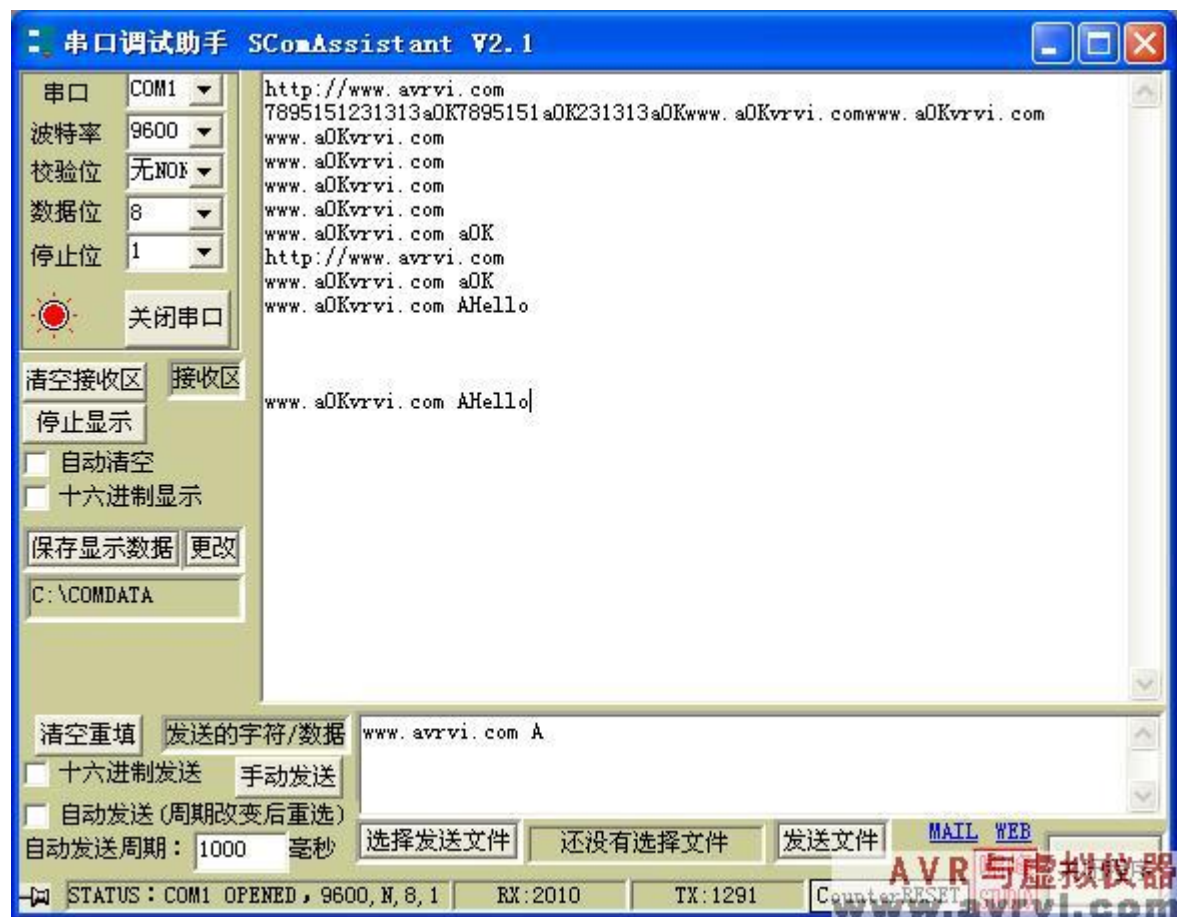
#### RS232 电路图



本程序在本站的开发板上通过，绝对没有问题，欢迎讨论。

测试效果，启动时发送 <http://www.avrvi.com>加回车换行，然后，收到什么，返回什么，如果给它发送一个小a，会多发回一个OK，如果发的是大A，则发回Hello字符串。

### 串口程序测试



```
// ICC-AVR application builder : 2007-5-20 17:21:25
```

```
// Target : M16
```

```
// Crystal: 7.3728Mhz
```

```
// AVR mega16 串口测试
```

```
// AVR与虚拟仪器 http://www.avrvi.com 古欣
```

```
#include <iom16v.h>
```

```
#include <macros.h>
```

```
#define F_CPU 7372800
```

```
const unsigned char buffer[] = "http://www.avrvi.com";
```

```

void USART_Init( unsigned int baud )
{
    unsigned int tmp;
    /* 设置波特率*/
    tmp= F_CPU/ baud/16-1;
    UBRRH = (unsigned char)(tmp>>8);
    UBRRL = (unsigned char)tmp;
    /* 接收器与发送器使能*/
    UCSRB = (1<<RXEN)|(1<<TXEN);
    /* 设置帧格式: 8 个数据位, 2 个停止位*/
    UCSRC = (1<<URSEL)|(1<<USBS)|(1<<UCSZ0)|(1<<UCSZ1);
}
/* UMSEL 模式选择
    0 异步操作
    1 同步操作

```

USBS 停止位位数

```

0 1
1 2

```

UCSZ2 UCSZ1 UCSZ0 字符长度

```

0 0 0 5 位
0 0 1 6 位
0 1 0 7 位
0 1 1 8 位
1 0 0 保留
1 0 1 保留
1 1 0 保留
1 1 1 9 位
*/

```

// ICC 生成的初始化

//UART0 initialize

// desired baud rate: 9600

// actual: baud rate: 9600 (0.0%)

// char size: 8 bit

// parity: Disabled

void uart0\_init(void)

```

{
    UCSRB = 0x00; //disable while setting baud rate
    UCSRA = 0x00;
    UCSRC = BIT(URSEL) | 0x06;
    UBRRL = 0x2F; //set baud rate lo
    UBRRH = 0x00; //set baud rate hi

```

```
UCSRB = 0x18;  
}
```

//下面两个函数直接从数据手册上拷贝过来的

```
// 数据发送【发送 5 到 8 位数据位的帧】  
void USART_Transmit( unsigned char data )  
{  
    /* 等待发送缓冲器为空 */  
    while ( !( UCSRA & (1<<UDRE)) )  
        ;  
    /* 将数据放入缓冲器，发送数据 */  
    UDR = data;  
}
```

```
// 数据接收【以 5 到 8 个数据位的方式接收数据帧】  
unsigned char USART_Receive( void )  
{  
    /* 等待接收数据*/  
    while ( !(UCSRA & (1<<RXC)) )  
        ;  
    /* 从缓冲器中获取并返回数据*/  
    return UDR;  
}
```

```
//连续发送字符  
void USART_Transmit_2( void )  
{  
    while ( !( UCSRA & (1<<UDRE)) )  
        ;  
    UDR = 'H';  
    while ( !( UCSRA & (1<<UDRE)) )  
        ;  
    UDR = 'e';  
    while ( !( UCSRA & (1<<UDRE)) )  
        ;  
    UDR = 'l';  
    while ( !( UCSRA & (1<<UDRE)) )  
        ;  
    UDR = 'l';  
    while ( !( UCSRA & (1<<UDRE)) )  
        ;  
    UDR = 'o';
```

```

}

void main(void)
{
    unsigned char n=0,tmp=0;

    USART_Init(9600); //波特率 9600 初始化串口
    // uart0_init();

    for(i=0;n<20;n++) //发送数组里面的字符串, http://www.avrvi.com
    {
        USART_Transmit(buffer[n]);
    }

    USART_Transmit(0x0d); //发送一个回车
    USART_Transmit(0x0a); //发送一个换行

    while(1)
    {
        if(UCSRA&(1<<RXC)) //如果接收缓存区有数据
        {
            tmp=USART_Receive(); //接收数据
            USART_Transmit(tmp); //发送数据

            if(tmp=='a') //对接收到的数据进行, 如果是a, 再发一个OK回来
            {
                USART_Transmit('O');
                USART_Transmit('K');
            }
            if(tmp=='A') //对接收到的数据进行, 如果是A, 再发一个Hello回来
            {
                USART_Transmit_2();
            }
        }
    }
}

```

USB 转串口, 转换过程是由硬件完成的,

你拿到板子了吗? 接好对应的跳线 (JP4, 2 和 3 短接), 插上 USB 线, 计算机会识别出芯片所在的串口号, 运行程序。

在串口调试助手里面选择对应的串口号就可以看到这些了。

为了防止网页复制出错，我把程序打包，如果不方便用 ICC，我把 HEX 文件也传上来。

### 调试通过的 AVR mega16 SPI 双机通讯例子（为新手设计，简单易懂）

本程序实现的功能：主机发送 1~255，从机接收并在 LED 上显示出来。

连接方式：两个 mega16 最小系统板 PB4 到 PB7 全部对连。

本程序在本站的最小系统板上测试通过，我向你担保本程序的正确性。

主机程序：

CODE:

```
//ICC-AVR application builder : 2007-7-18 13:01:11
// Target : M16
// Crystal: 7.3728Mhz
// 作者：古欣
// AVR 与虚拟仪器 [url]http://www.avrvi.com[url]
// 功能：SPI 主机模式，循环发送从 1~255

#include <iom16v.h>
#include <macros.h>

void port_init(void)
{
PORTA = 0x00;
DDRA = 0x00;
PORTB = 0x00;
DDRB = 0x00;
PORTC = 0x00; //m103 output only
DDRC = 0x00;
PORTD = 0x00;
DDRD = 0x00;
}

//SPI initialize
// clock rate: 57599hz
void spi_init(void)
{
PORTB |= (1<<PB4) | (1<<PB5) | (1<<PB6) | (1<<PB7);
DDRB |= (1<<DDB5) | (1<<DDB7) | (1<<DDB4); //Set MOSI, SCK AND SS as
outputs
SPCR = 0x73; //setup SPI
```

```

SPSR = 0x00; //setup SPI
}

//call this routine to initialize all peripherals
void init_devices(void)
{
//stop errant interrupts until set up
CLI(); //disable all interrupts
port_init();
spi_init();

MCUCR = 0x00;
GICR = 0x00;
TIMSK = 0x00; //timer interrupt sources
SEI(); //re-enable interrupts
//all peripherals are now initialized
}

void SPI_MasterTransmit(char cData)
{
PORTB &= ~ (1<<PB4); //强制接收方进入从模式
SPCR |= (1<<MSTR); // MSTR 有时会被清零，这里强制进入主机模式
/* 启动数据传输 */
SPDR = cData;
/* 等待传输结束 */
while(!(SPSR & (1<<SPIF)))
;
PORTB |= (1<<PB4);
}

void Delay(void) //延时，没有详细计算
{
unsigned int i,j;
for(i=1000;i>0;i--)
{
for(j=200;j>0;j--)
;
}
}

void main(void)
{
unsigned char i=0;
init_devices();

```

```

while(1)
{
for(i=255;i>0;i--)
{
SPI_MasterTransmit(i);
Delay();
}
}
}

```

从机程序

**CODE:**

```

//ICC-AVR application builder : 2007-7-18 12:56:10
// Target : M16
// Crystal: 7.3728Mhz
// 作者: 古欣
// AVR 与虚拟仪器 [url]http://www.avrvi.com[url]
// 功能: 从机模式, 中断方式接收, 并在 LED 上显示

```

```

#include <iom16v.h>
#include <macros.h>

```

```

void port_init(void)
{
PORTA = 0x00;
DDRA = 0xFF;
PORTB = 0x00;
DDRB = 0x00;
PORTC = 0x00; //m103 output only
DDRC = 0x00;
PORTD = 0x00;
DDRD = 0x00;
}

```

```

//SPI initialize
// clock rate: 57599hz
void spi_init(void)
{
SPCR = 0xE3; //setup SPI
SPSR = 0x00; //setup SPI
}

```



```

#pragma interrupt_handler spi_stc_isr: 11
void spi_stc_isr(void)
{
    //byte in SPDR has been sent/received
    PORTA = SPDR;
}

//call this routine to initialize all peripherals
void init_devices(void)
{
    //stop errant interrupts until set up
    CLI(); //disable all interrupts
    port_init();
    spi_init();

    MCUCR = 0x00;
    GICR = 0x00;
    TIMSK = 0x00; //timer interrupt sources
    SEI(); //re-enable interrupts
    //all peripherals are now initialized
}

void main(void)
{
    init_devices();
    DDRB|=(1<<PB6); //MOSI 设置为输出
    while(1)
    ;//等待中断
}

```

今天搞了AVR的TWI主从机通讯，专门写了一个简单的易于新手看懂的例子程序

我们有工程师级的程序，相信很多人都看不懂，曾经给人看过。

这次写了个新手级的AVR TWI程序，方便学习者了解TWI工作流程和原理。

显得稍微有点乱，整理下再发上来。

程序实现功能：

主机从 0 到 255 循环发送字节到从机，从机收到后返回给主机，当主机收到的数为 0x10 时，

主机发送再次从零开始。

硬件连接，两个最小mega16 系统板，PC0 和PC1 互相连接，使用芯片的内部上拉电阻，电路上没有接上拉。

PS：为了保险，建议接 1K~10K的上拉电阻到总线上。

主机主程序

CODE:

```
//ICC-AVR application builder : 2007-7-19 9:50:03
// Target : M16
// Crystal: 7.3728Mhz
// 作者: 古欣
// AVR 与虚拟仪器 [url]http://www.avrvi.com[url]

//主机从 1 到 255 发一个数，收一个数。从机接收到任何数都返回。
//当主机收到的数为 0x10 时，从零开始发送，主机采用查询方式，从机采用中断方式

#include "config.h"

void main(void)
{
    uint8 i=0,tmp=0;
    //re-enable interrupts
    DDRC=0X00;
    PORTC=0x03; //使能内部上拉电阻
    twi_master_init();

    while(1)
    {
        i2c_maste_transt(0x50,i);
        i++;
        delay_1s();
        tmp=i2c_maste_read(0x50); //读一个数
        if(tmp==0x10) i=0;
    }
}
```

[\[Copy to clipboard\]](#)

从机主程序:

CODE:

```
// 作者: 古欣
// AVR 与虚拟仪器 [url]http://www.avrvi.com[/url]
// TWI 通讯从机, 接受到数据回传给主机
// 关键内容在 中断函数 void twi_isr(void)中
```

```
#include "config.h"
```

```
void main(void)
{
    DDRC=0X00;
    PORTC=0x03; //使能内部上拉电阻
    DDRA=0xff;
    twi_slave_init(0x50); //初始化为从机, 地址 0x01。
    SEI();
    while(1)
    ;
}
```

描述: TWI通讯程序范例 ICC

附件:  [simple\\_twitest.rar](#) (60 K) 下载次数:1613

## 7 段数码管（使用范例）

本程序是增强版MEGA16/32 开发板的示例程序之一, 具体原理和说明会在配套教程中给出

CODE:

```
/*
*****
** file_name led.c          **
** describe 七段数码管函数  **
** author 古欣 [url]www.avrvi.com[/url] **
** Time 200-2-25          **
** temp=(data*1000)%1; 这个式子中的%为取余数, temp 为 data 的第三位小数,
0.023(data) --> 3(temp)
*****
#include "config.h"

//LED 数据 不显点, 亮的段为 1, 连接为 P0~P7 对应 a~g,dp
const
```

```
led_data[16]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x73,0x39,0x63,0x79};
```

```
/******
```

```
** 初始化对应端口为输出
```

```
** ABCD 和 abcdefg dpi 共 12 个 IO 口
```

```
** 默认为 PB(0123) 和 PA
```

```
*****/
```

```
void led_init(void)
```

```
{
```

```
led_ddr = 0xFF; //abcdefg dpi 各位设为输出
```

```
led_contrl_ddr |= (1<<led_a)|(1<<led_b)|(1<<led_c)|(1<<led_d); //ABCD 四个控制设为输出
```

```
// 以下两句将会使 LED 全亮,如果立即更改状态,将看不出来效果
```

```
// led_port = 0xFF;
```

```
// led_contrl_port |= (1<<led_a)|(1<<led_b)|(1<<led_c)|(1<<led_d);
```

```
}
```

```
/******
```

```
** 显示一位
```

```
** 输入: count 显示在第几位(3210), data 要显示的数(0~f)
```

```
*****/
```

```
void display_one(uint8 count, uint8 data)
```

```
{
```

```
led_port = led_data[data]; //显示的数
```

```
led_contrl_port |= (1<<count); //选中对应要显示的位,从右至左,0123
```

```
//如果对位进行了调整,就不在是 0123,而是对应的值
```

```
}
```

```
/******
```

```
** 显示四位整数
```

```
** 输入: 要显示的四位数 data,显示模式 mode, 1 为补零显示模式,默认不显示零
```

```
** 说明: 可以小于四位数, mode=1 时 自动补零
```

```
*****/
```

```
void display(uint16 data,uint8 mode)
```

```
{
```

```
uint8 temp;
```

```
//千位
```

```
if(data>=1000)
```

```
{
```

```
temp=data/1000;
```

```
display_one(3,temp);
```

```
delay_ms(6); //6ms 是个经验值,刚好无闪烁,并且亮度较高
```

```

led_contrl_port &= ~ (1<<3);
}
else
{//补零
if(mode==1)
{
display_one(3,0);
delay_ms(6);
led_contrl_port &= ~ (1<<3);
}
}
//百位
if(data>=100)
{
temp=(data%1000)/100;
display_one(2,temp);
delay_ms(6);
led_contrl_port &= ~ (1<<2);
}
else
{
if(mode==1)
{
display_one(2,0);
delay_ms(6);
led_contrl_port &= ~ (1<<2);
}
}
//十位
if(data>=10)
{
temp=(data%100)/10;
display_one(1,temp);
delay_ms(6);
led_contrl_port &= ~ (1<<1);
}
else
{
if(mode==1)
{
display_one(1,0);
delay_ms(6);
led_contrl_port &= ~ (1<<1);
}
}

```

```

}
//个位
temp=data%10;
display_one(0,temp);
delay_ms(6);
led_contrl_port &= ~ (1<<0);
}

/*****
** 显示四位浮点数 本函数占用 mega16 的空间 19%
** 输入：要显示的四位数
** 说明：可以小于四位数，自动补零
*****/

void display_float(float data)
{
uint8 temp;
uint16 temp2; //用于把浮点数变为整形
// 人为保证 data 的值，可以不要错误处理。
if (data>=1000)
{
led_error();
}
if (data<=0)
{
led_error();
}
//
if(data<1)    //比如 0.123
{
display_one(3,0);    //显示 0.
led_port |= 0x80; //点亮对应小数点
delay_ms(6);
led_contrl_port &= ~ (1<<3);

temp=(data*10);    // 0.123*10 = 1
display_one(2,temp);
delay_ms(6);
led_contrl_port &= ~ (1<<2);

temp2=(data*100);
temp=temp2%10; // 0.123*100%10 = 2
display_one(1,temp);
delay_ms(6);

```

```

led_contrl_port &= ~ (1<<1);

temp2=(data*1000);
temp=temp2%10; // 0.123*100%10 = 3
display_one(0,temp);
delay_ms(6);
led_contrl_port &= ~ (1<<0);
}
else if(data<10) //比如 1.234
{
temp=(data/1); // 1.234/1 = 1
display_one(3,temp);
led_port |= 0x80; //点亮对应小数点
delay_ms(6);
led_contrl_port &= ~ (1<<3);
//第一位小数
temp2=(data*10);
temp=temp2%10; // 1.234*10%10 = 2
display_one(2,temp);
delay_ms(6);
led_contrl_port &= ~ (1<<2);
//第二位小数
temp2=(data*100);
temp=temp2%10; // 1.234*100%10 = 3
display_one(1,temp);
delay_ms(6);
led_contrl_port &= ~ (1<<1);
//第三位小数
temp2=(data*1000);
temp=temp2%10; // 1.234*1000%10 = 4
display_one(0,temp);
delay_ms(6);
led_contrl_port &= ~ (1<<0);
}
else if(data<100) // 例如 12.34
{
temp2=data;
temp=(temp2/10); // 12.34/10=1
display_one(3,temp);
delay_ms(6);
led_contrl_port &= ~ (1<<3);

temp2=data;
temp=(temp2%10); // 12.34%10=2

```

```

display_one(2,temp);
led_port |= 0x80; //点亮对应小数点
delay_ms(6);
led_contrl_port &= ~ (1<<2);

temp2=(data*10);
temp=temp2%10; // 12.34*10%10=3
display_one(1,temp);
delay_ms(6);
led_contrl_port &= ~ (1<<1);

temp2=(data*100);
temp=temp2%10; // 12.34*100%10=4
display_one(0,temp);
delay_ms(6);
led_contrl_port &= ~ (1<<0);
}
else if(data<1000) //比如 123.4
{
temp2=data;
temp=(temp2/100); // 123.4/100=1
display_one(3,temp);
delay_ms(6);
led_contrl_port &= ~ (1<<3);

temp2=data/10;
temp=temp2%10; // 123.4/10%10=2
display_one(2,temp);
delay_ms(6);
led_contrl_port &= ~ (1<<2);

temp2=data;
temp=(temp2%10); // 123.4%10=3
display_one(1,temp);
led_port |= 0x80; //点亮对应小数点
delay_ms(6);
led_contrl_port &= ~ (1<<1);

temp2=(data*10);
temp=temp2%10; // 123.4*10%10=4
display_one(0,temp);
delay_ms(6);
led_contrl_port &= ~ (1<<0);
}

```



```
else led_error();  
}
```

```
/******
```

```
** 显示浮点数 本函数占用 mega16 的空间 9%，有一点闪烁感，可将 display 中的延时调小
```

```
** 思路二：直接显示整数，再显示点
```

```
** 注意：输入的 data 必须为小于 1000 的四位小数(0.234,1.23,12.45,224.8) 等
```

```
*****/
```

```
void display_float2(float data)
```

```
{  
  uint16 temp;  
  if(data>1000)  
  {  
    temp=data;  
    display(temp,1);  
    return;  
  }  
  if(data>100)  //比如 123.4  
  {  
    temp=data*10;  
    display(temp,0); //普通模式显示 1234  
    led_contrl_port |= (1<<1); //选中第三位  
    led_port = 0x80; //点亮对应小数点  
    delay_ms(6);  
    led_contrl_port &=~ (1<<1);  
    return;  
  }  
  if(data>10)  //比如 12.34  
  {  
    temp=data*100;  
    display(temp,0); //普通模式显示 1234  
    led_contrl_port |= (1<<2); //选中第二位  
    led_port = 0x80; //点亮对应小数点  
    delay_ms(6);  
    led_contrl_port &=~ (1<<2);  
    return;  
  }  
  if(data>1)  //比如 1.234  
  {  
    temp=data*1000;  
    display(temp,0); //普通模式显示 1234  
    led_contrl_port |= (1<<3); //选中第一位
```

```

led_port = 0x80; //点亮对应小数点,这里用等于, 否则 display 函数的显影有残留。
delay_ms(6);
led_contrl_port &= ~ (1<<3);
return;
}
}

```

```

/*****
** 异常处理, led 全亮 8。8。8。8。
*****/

void led_error(void)
{
// 以下两句将会使 LED 全亮,如果立即更改状态, 将看不出来效果
led_port = 0xFF;
led_contrl_port |= (1<<led_a)|(1<<led_b)|(1<<led_c)|(1<<led_d);
}

```

[\[Copy to clipboard\]](#)

#### CODE:

```

/*****
** file_name led.h          **
** describe 七段数码管头文件 **
** LED 型号 ARK SR420361K   **
** auther 古欣 [url]www.avrvi.com[url] **
** Time 2007-2-25          **
*****/

#ifndef _LED_H_
#define _7LED_H_ 1

#define led_port PORTA
//定义输出端口
#define led_ddr DDRA
//定义输出控制寄存器
//可以自行修改, 必须保持 led_port 和 led_ddr 一致, PORTA 对应 DDRA

//定义位显示控制, 及 ABCD, 这里用 PB0-A; PB1-B; PB2-C; PB3-D
#define led_contrl_port PORTB
#define led_contrl_ddr DDRB
//无论使用哪个口, 请保持四位为 0123, 否则你需要调整函数 display_one()和 display();
#define led_a 0
#define led_b 1
#define led_c 2
#define led_d 3

```

```

/*****
** LED 显示初始化
** 说明：将对应端口设置为输出
** 默认为 PB(0123) 和 PA
*****/
extern void led_init(void);

/*****
** 显示一位
** 输入：count 显示在第几位(3210)，data 要显示的数(0~f)
**
*****/
extern void display_one(uint8 count, uint8 data);

/*****
** 显示四位整数
** 输入：要显示的四位数 data,显示模式 mode，1 为补零显示模式,默认不显示零
** 说明：可以小于四位数，mode=1 时 自动补零
*****/
extern void display(uint16 data,uint8 mode);

/*****
** 显示四位浮点数
** 输入：要显示的四位数
** 说明：可以小于四位数，自动补零
*****/
extern void display_float(float data);

/*****
** 显示浮点数
** 思路二：直接显示整数，再显示点
** 注意：输入的 data 必须为小于 1000 的四位小数(0.234,1.23,12.45,224.8) 等
*****/
extern void display_float2(float data);

/*****
** 异常处理，led 全亮 8。8。8。8。
*****/
void led_error(void);

#endif
\[Copy to clipboard\]

```

CODE:

```
// ICC-AVR application builder : 2007-2-14 21:34:00
// Target : M16
// Crystal: 7.3728Mhz
// 工程 DEMO 七段数码管显示演示

#include "config.h"

extern const led_data[16]; //声明外部 led 显示数据, led_data 在 led.c 中
uint16 countdata;

void port_init(void)
{
    PORTA = 0x00;
    DDRA = 0x00;
    PORTB = 0x00;
    DDRB = 0x00;
    PORTC = 0x00; //m103 output only
    DDRC = 0x00;
    PORTD = 0x00;
    DDRD = 0x00;
}
/*****/
//定时器 1 用于一秒刷新 countdata 的值, 达到从 1—9999 计秒的功能
//TIMER1 initialize - prescale:1024
// WGM: 0) Normal, TOP=0xFFFF
// desired value: 1Sec
// actual value: 1.000Sec (0.0%)
void timer1_init(void)
{
    TCCR1B = 0x00; //stop
    TCNT1H = 0xE3; //setup
    TCNT1L = 0xE1;
    OCR1AH = 0x1C;
    OCR1AL = 0x1F;
    OCR1BH = 0x1C;
    OCR1BL = 0x1F;
    ICR1H = 0x1C;
    ICR1L = 0x1F;
    TCCR1A = 0x00;
    TCCR1B = 0x05; //start Timer
}

#pragma interrupt_handler timer1_ovf_isr:9
```

```

void timer1_ovf_isr(void)
{
//TIMER1 has overflowed
TCNT1H = 0xE3; //reload counter high value
TCNT1L = 0xE1; //reload counter low value
countdata++;
if(countdata==9999) countdata=0;
}

//call this routine to initialize all peripherals
void init_devices(void)
{
//stop errant interrupts until set up
CLI(); //disable all interrupts
port_init();
led_init(); //7 段数码管显示

MCUCR = 0x00;
GICR = 0x00;
TIMSK = 0x00; //timer interrupt sources
SEI(); //re-enable interrupts
//all peripherals are now initialized
}

/*****/
//只有综合应用才用到本函数
//call this routine to initialize all peripherals
void init_devices2(void)
{
//stop errant interrupts until set up
CLI(); //disable all interrupts
port_init();
led_init(); //7 段数码管显示
timer1_init(); //定时器 1 用于更新 countdata 的值，从 1 到 9999 变化

MCUCR = 0x00;
GICR = 0x00;
TIMSK = 0x04; //注意这里允许定时器一溢出
SEI(); //re-enable interrupts
//all peripherals are now initialized
}
/*****
*****/

```

//以下显示模式只保留一个，查看效果

```
/*  
*****  
******/
```

//综合应用例子

```
void main(void)  
{  
init_devices2(); //比 init_devices()多了 timer1_init(),TIMSK = 0x04;  
while(1)  
{  
display(countdata,0); //普通模式，显示 countdata 的值，countdata 将在 timer1 定时一  
秒溢出时改变  
}  
}
```

```
/*  
//用第二种思路显示一个浮点数，有点闪烁感
```

```
void main(void)  
{  
init_devices();  
while(1)  
{  
display_float2(1.234);  
}  
}  
*/
```

```
/*  
//用第一种思路显示一个浮点数，耗费大量（mega16 19%）空间和 cpu 时间
```

```
void main(void)  
{  
init_devices();  
while(1)  
{  
display_float(1.234);  
}  
}  
*/
```

```
/*  
//最常用的显示，调用 display 显示 1—9999 中的数
```

```
void main(void)  
{
```

```

init_devices();
while(1)
{
// display(567,0); //普通模式
display(567,1); //补零模式
}
}

```

```

/*
//从 0 到 F 依次显示，一秒变换一次
void main(void)
{
unsigned char i=0;
init_devices();
led_contrl_port |= (1<<led_a)|(1<<led_b)|(1<<led_c)|(1<<led_d); //选中对应位
while(1)
{
led_port=led_data[i];
delay_1s();
i++;
if(i==15) i=0;
}
}
*/

```

```

/*
//一位一位的显示
void main(void)
{
init_devices();
while(1)
{
display_one(0,1);
delay_ms(6); //6ms 为经验值
led_contrl_port &= ~ (1<<0);
display_one(1,2);
delay_ms(6);
led_contrl_port &= ~ (1<<1);
display_one(2,3);
delay_ms(6);
led_contrl_port &= ~ (1<<2);
display_one(3,4);
}
}

```

```

delay_ms(6);
    led_contrl_port &= ~ (1<<3);
}
}
*/

```

整个工程文件下载：

描述：avr 7 断数码管（使用范例）

附件： [7 断数码管.rar](#) (70 K) 下载次数:552

#### 4×4 矩阵键盘（使用范例） 线翻转法矩阵扫描程序

本程序是增强版MEGA16/32 开发板的示例程序之一，详细原理及说明会在配套教程中给出。

CODE:

```

/*****
** Filename: keyboard.c
** Describe: 矩阵键盘程序
** Author : 古欣 [url]www.avrvi.com[url]
** Time : 2007-2-15
*****/

#include "config.h"

/*****
** 说明: 线翻转法进行键盘扫描
** 输出: 获得高低位的扫描值
** 有键时需要耗时 14ms
*****/

unsigned char key_scan(void)
{
    unsigned char temp=0,key=0;
    KEY_DDR = 0xF0; //高四位输出 0，键按下，则对应的值为 0
    KEY_PORT = 0x0F; //低四位输入，内部电阻上拉，没有键按下时为高
    temp = KEY_PIN&0x0F; //与掉高四位
    if(temp==0x0F)
    {
        return 0; // 无按键返回
    }
    else
    {
        delay_ms(10);
        temp = KEY_PIN&0x0F; //延时去抖后再检测
    }
}

```



```

if(temp==0x0F)
    return 0;
else
    key=temp;
}
//翻转
KEY_DDR = 0x0F; //低四位输出 0，键按下，则对应的值为 0
KEY_PORT = 0xF0; //高四位输入，内部电阻上拉，没有键按下时为高
delay_ms(3); //延时等待稳定

temp = KEY_PIN&0xF0; //与掉低四位
if(temp==0xF0)
{
return 0; // 无按键返回
}
else //这里不再延时再扫描，因为已经确定了不是抖动才会进入本步操作。
{
    key |= temp; //高低位的键值进入 KEY
}
KEY_DDR = 0x00; /*输出复位*/
KEY_PORT = 0xFF;

return key;
}

/*****
** 说明：获得键盘的值
** 内部调用函数 key_scan
** 输出：实际键值
*****/
unsigned char get_key(void)
{
    unsigned char i=0;
    i=key_scan();
    switch (i) { /*将按键码转换成键值*/
        case 0x00: return 0x00;
        case 0xEE: return '1';
        case 0xED: return '2';
        case 0xEB: return '3';
        case 0xE7: return 'A';
        case 0xDE: return '4';
        case 0xDD: return '5';
        case 0xDB: return '6';
    }
}

```

```

case 0xD7: return 'B';
case 0xBE: return '7';
case 0xBD: return '8';
case 0xBB: return '9';
case 0xB7: return 'C';
case 0x7E: return '*';
case 0x7D: return 'O';
case 0x7B: return '#';
case 0x77: return 'D';
default : return 0x00;
}
}

```

[\[Copy to clipboard\]](#)

keyboard.h

**CODE:**

```

/*****
** Filename: keyboard.h (线翻转法)
** Describe: 矩阵键盘程序 头文件
** Author : 古欣 [url]www.avrvi.com[url]
** Time : 2007-2-15
** -----
** | PA0 PA1 PA2 PA3
** |   |   |   |
** |PA4  1  2  3  A
** |PA5  4  5  6  B
** |PA6  7  8  9  C
** |PA7  *  0  #  D
** -----
*****/

#ifndef _KEYBOARD_
#define _KEYBOARD_ 1

#define KEY_DDR DDRA
#define KEY_PORT PORTA
#define KEY_PIN PINA

/*****
** 说明: 线翻转法进行键盘扫描
** 输出: 获得高低位的扫描值
** 有键时需要耗时 14ms
*****/

```

```
extern unsigned char key_scan(void);

/*****
** 说明：获得键盘的值
** 内部调用函数 key_scan
** 输出：实际键值
*****/
extern unsigned char get_key(void);

#endif
```

整个工程文件下载：

描述：AVR 矩阵键盘程序 键盘扫描

附件：  [keyboard.rar](#) (31 K) 下载次数: 1374

## 18b20 程序范例

CODE:

```
/*****
** Filename: 18b20.c
** Describe: 18b20 温度传感器
** Author : 古欣 [url]www.avrvi.com[url]
** Time : 2007-2-15
*****/

#include "config.h"

unsigned char wml,flag,count; //flag 温度为负标志，count 为实际温度
void init_1820(void)
{
    SET_DIR_1WIRE;    //设置 PC2 为输出
    SET_OP_1WIRE;
    CLR_OP_1WIRE;
    delay_us(480);    //480us 以上
    SET_OP_1WIRE;
    CLR_DIR_1WIRE;
    delay_us(20);    //15~60us
    while(CHECK_IP_1WIRE);
    SET_DIR_1WIRE;
    SET_OP_1WIRE;
    delay_us(140);    //60~240us
}
```

```

void write_1820(unsigned char x)
{
    unsigned char m;
    for(m=0;m<8;m++)
    {
        CLR_OP_1WIRE;
        if(x&(1<<m))    //写数据了，先写低位的！
            SET_OP_1WIRE;
        else
            { CLR_OP_1WIRE; }
        delay_us(40);    //15~60us
        SET_OP_1WIRE;
    }
    SET_OP_1WIRE;
}

unsigned char read_1820(void)
{
    unsigned char temp,k,n;
    temp=0;
    for(n=0;n<8;n++)
    {
        CLR_OP_1WIRE;
        SET_OP_1WIRE;
        CLR_DIR_1WIRE;
        k=(CHECK_IP_1WIRE);    //读数据,从低位开始
        if(k)
            temp|=(1<<n);
        else
            temp&=~(1<<n);
        delay_us(50);    //60~120us
        SET_DIR_1WIRE;
    }
    return (temp);
}

void gettemp(void)    //读取温度值
{
    unsigned char temh,templ,wm0,wm1,wm2,wm3;
    init_1820();    //复位 18b20
    write_1820(0xcc);    // 发出转换命令
    write_1820(0x44);
    // delay_nms(800);    //不延时也好使，不知道怎么回事！
    init_1820();
    write_1820(0xcc);    //发出读命令

```

```

write_1820(0xbe);
teml=read_1820(); //读数据
temh=read_1820();
wm0=teml>>4;      //只要高 8 位的低四位和低 8 位的高四位，温度范围 0~99 啦！
wm1=temh<<4;

//count=(temh*256+teml)*6.25; //计算具体温度
if((temh&0xF8) == 0xF8)
{
    flag=1;
    count=((0xFF-temh)*256+(0xFF-teml))*6.25;
    //count=((0xFF-temh)*256+(0xFF-teml))*625;
}
else
{
    flag=0;
    count=(temh*256+teml)*6.25;
    //count=(temh*256+teml)*625; //计算具体温度
}
wm2=wm1+wm0;      //16 进制转 10 进制
wm3=wm2/100;
wmh=(wm2%100)/10;  //出口参数了！wmh 是显示的高位，wml 是显示的低位
wml=(wm2%100)%10;

}

```

整个工程文件下载：

描述：avr 18b20 程序

附件：  [18B20.rar](#) (56 K) 下载次数: 1076

## Mage128 总线方式的 128×64 液晶源程序和仿真

包括数字和字符显示，汉字显示，以及图形显示三个部分！！

附件：  [LCD 12864.rar](#) (129 K) 下载次数: 2318

## AVR与L298 进行直流电机控制

关键词：直流电机控制，L298，avr电机控制。

本程序由AVR与虚拟仪器网站提供，免费共享，不记版权，欢迎转载，请注明出处。

网址: <http://www.avrvi.com> 论坛: <http://www.avrvi.com/bbs/>

本程序通过测试，实现预期效果，如果你在使用中遇到问题，请在论坛交流。

程序编写整理：古欣

软件环境：icc+avrstudio

下载工具：PHYSICO AVR JTAG&stk500 仿真编程器

介绍: <http://www.avrvi.com/bbs/read.php?tid-61-page-e.html>

硬件环境：mega32+N298，你可以选择我们提供的运动控制开发板。

硬件连结：请参考 硬件连接电路图.jpg

使用方法：直接将main.hex文件下载到单片机中就可以使用。你可以打开工程文件查看

， motor.c已经是一个结构化的驱动程序，可以方便的调用。

更改连接：请打开motor.h更改以下内容即可。

```
//PD4,PD5 电机方向控制
#define moto_en1 PORTD |= 0x10
#define moto_en2 PORTD |= 0x20
#define moto_uen1 PORTD &= ~ 0x10
#define moto_uen2 PORTD &= ~ 0x20
```

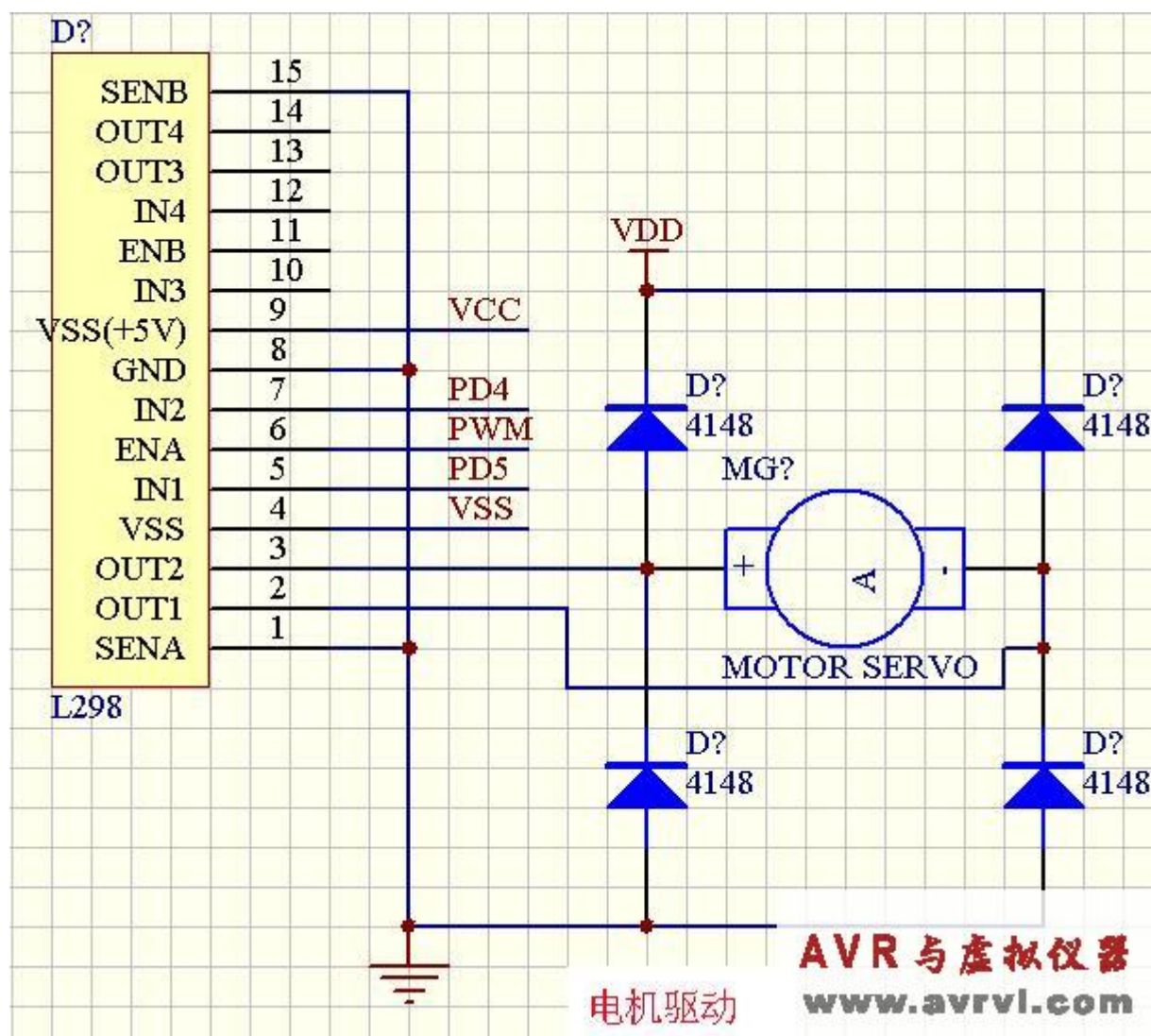
软件核心介绍：

通过控制电机的IN1 和IN2 改变方向， pwm控制速度。

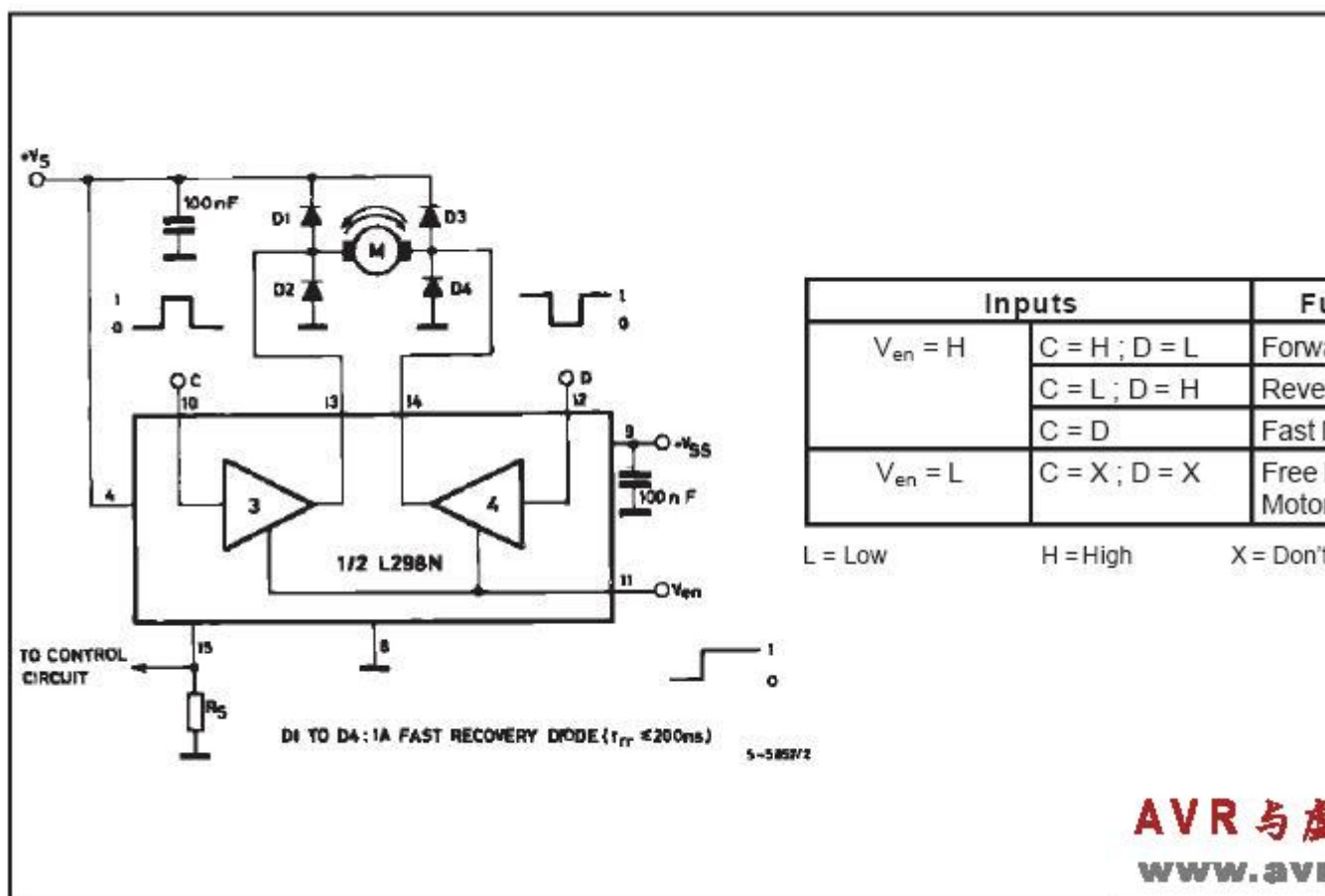
IN1 和IN2 不同时，电机转动；

IN1 和IN2 相同时，电机急停。

**硬件连接电路图**

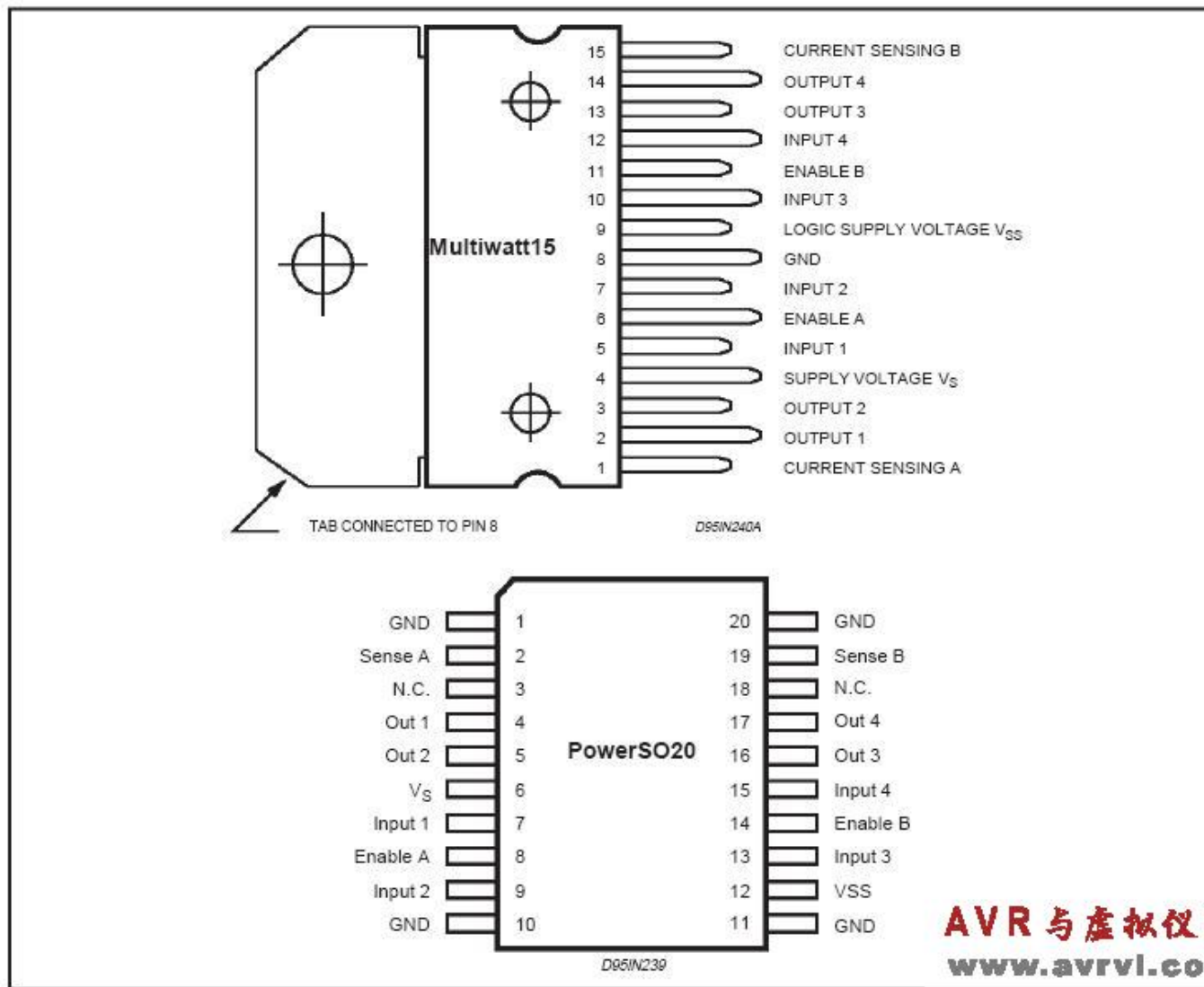


avr+N298 控制电机输入信号与电机工作方式



N298 电机控制芯片引脚图





电机控制资料包括 所有电机控制例子源程序，L298 数据手册，连接电路图。

今天有人反映，程序有点小问题

程序中这里的IO初始化

```
void port_init(void)
{
PORTA = 0x00;
DDRA = 0x00;
PORTB = 0x00;
DDRB = 0x00;
PORTC = 0x00;
DDRC = 0x00;
PORTD = 0x00;
```

```
// DDRD = 0x80; // D7 PWM //参考芯片手册
//这里修改
DDRD =0xB0; //PD4 PD5 PD7 OUT
}
```

另外注意如果，启动时的PWM占空比太低 电机是转不起来的，

```
void main(void)
{


/*****
*****/
//初始工作
/*****
*****/
init_devices();

while(1)
{
    for_ward(0);          //默认速度运转 正
    Delay1s(5);           //延时 5s
    motor_stop();         //停止
    Delay1s(5);           //延时 5s
    back_ward(0);         //默认速度运转 反
    Delay1s(5);           //延时 5s
    speed_add(20);         //加速
    Delay1s(5);           //延时 5s
    speed_subtract(20);    //减速
    Delay1s(5);           //延时 5s
}

}
```

及这里的for\_ward(0); //默认速度运转 正  
如果改为for\_ward(5);电机转不起来

描述：电机控制资料

附件：  [电机控制.rar](#) (257 K) 下载次数:2771

基于TC1 的 16 位PWM输出程序

```

/*****
*          TC1 产生 16 位双路PWM          *
* 实验内容：由TC1 产生两路独立的 16 位PWM输出  *
* 实验环境：本站M16 学习板          *
* 日 期：2007 年 08 月 16 日          *
* 作 者：tonghe          *
* 版 本：V1.0          *
* 修改日期：2007 年 08 月 16 日          *
* 芯 片：M16          *
* 工作频率：内部 8M          *
* 编 译 器：ICCAVR 6.31A          *
* 输 出：PD4 输出PWMB，PD5 输出PWMA          *
*          产生的PWM频率为 8M/65536 约 122HZ          *
*****/

#include <iom16v.h>
#include <macros.h>

//延时函数：入口time 需延时的MS数
void delay_ms(unsigned int time)
{
    unsigned char c;
    for(; time; time--)
    {
        for(c=220; c; c--)
        {
            ;
        }
    }
}

//端口初始化
void port_init(void)
{
    PORTA = 0xFF;
    DDRA = 0x00;
    PORTB = 0xFF;
    DDRB = 0xFF;
    PORTC = 0x00;
    DDRC = 0x00;
    PORTD = 0x00;
    DDRD = 0x30;
}

//TC1 初始化

```

```

void timer1_init(void)
{
    TCCR1A = 0xA2;          //两路PWM，匹配清零
    TCCR1B = 0x19;          //快速PWM模式，位数可调，预分频 1
    ICR1   = 0xFFFF;        //计数上限值，此数为 16 位PWM
}

//器件初始化
void init_devices(void)
{
    port_init();            //端口初始化
    timer1_init();          //TC1 初始化

    MCUCR = 0x00;
    GICR = 0x00;
}

//主函数
void main(void)
{
    unsigned int a=32768,b=32768;
    init_devices();          //器件初始化

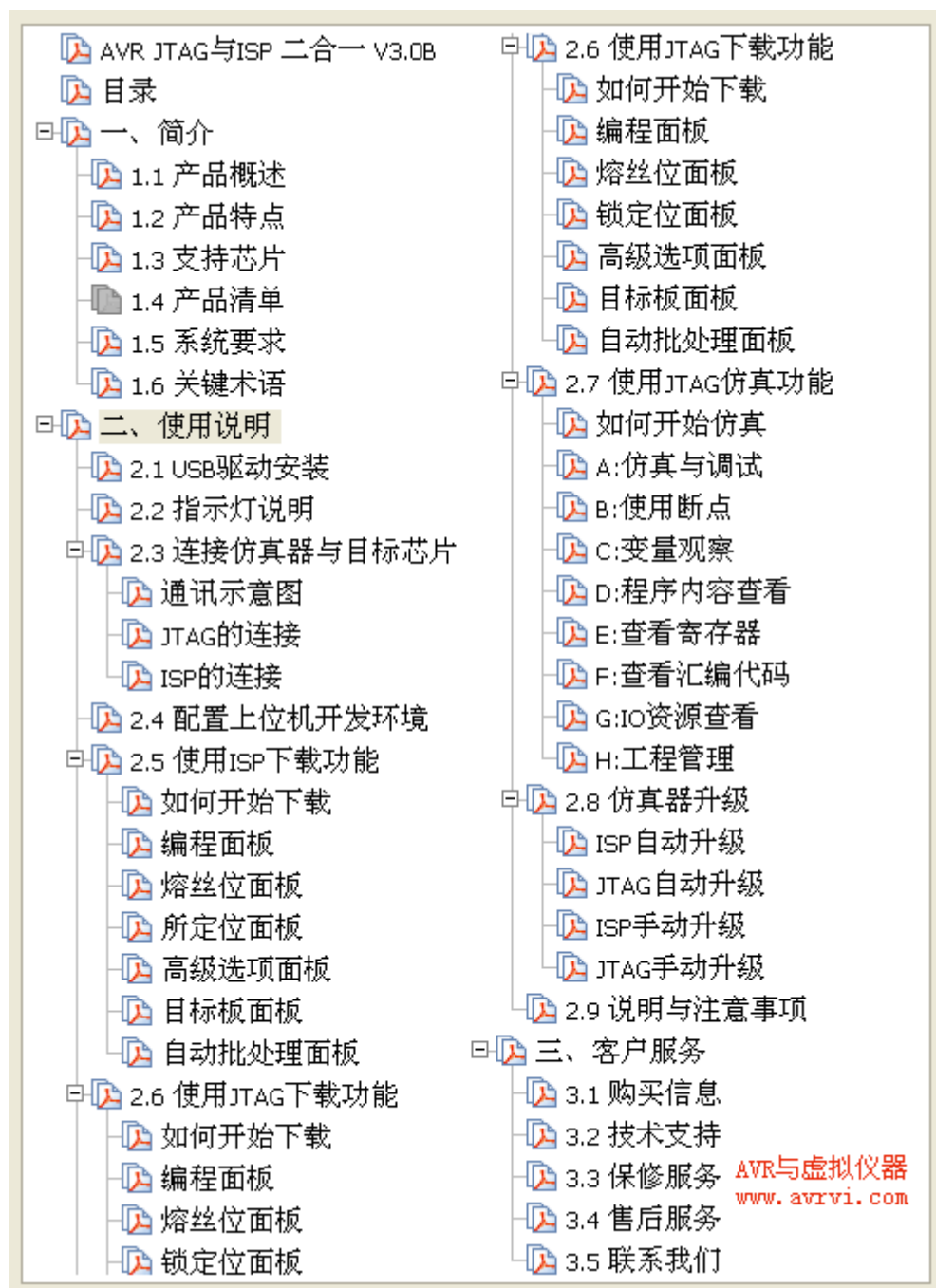
    OCR1A=a;                 //匹配初值
    OCR1B=b;

    while(1)
    {
        a-=10;               //两个值向不同方向变化
        b+=10;               //PWMA宽度减小，PWMB宽度增加
        if(a<=20)            //上下限
        {
            a=65530;
            b=5;
        }
        OCR1A=a;
        OCR1B=b;
        delay_ms(20);
    }
}

```

## AVR JTAG与ISP 二合一 V3.0B 使用说明书 及 AVR 下载和仿真教程

这是一本说明书，也是一本教程，值得你收藏的技术手册。 打开地址：[http://www.avrvi.com/start/AVRvi\\_JTAG\\_ISP\\_V3.0B.pdf](http://www.avrvi.com/start/AVRvi_JTAG_ISP_V3.0B.pdf)



### 1、 产品概述

AVR JTAG & ISP V3.0B是一款集成了 AVR JTAG和 AVR ISP的双功能多用途的仿真、编程一体机，与 AVR Studio(AVR Studio 4.09 或更高版本才能使用 AVR JTAG & ISP)相结合，通过 COM 或 USB 可以对所有带 JTAG 接口的 AVR 单片机进行在片调试（On-Chip Debugging）和编程，同时支持 AVR 全系列单片机的 ISP 程序下载。仿真器固件内核适应

AVRstudio4.13, 建议使用AVRstudio4.13 进行操作, 本仿真器可以自动升级。

## 2、产品特点

- ① JTAG 和 ISP 功能的完美单芯片解决方案, 大幅降低 AVR 入门成本。
- ② JTAG 和 ISP 功能分别与 ATMEL AVR JTAGICE和ATMEL AVR ISP 完全兼容。
- ③ 彻底防错插功能, 即插即用, 超强保护和隔离。
- ④ 目标自动识别, 无需跳线或开关转换。
- ⑤ 自动实现系统重启, 无需电源开关和复位键。
- ⑥ 实现与 AVR Studio 无缝连接, 仿真、下载、编程同步实现。
- ⑦ 在片仿真芯片的数字和模拟功能, 完全实现 AVR 单片机的所有的电性能。
- ⑧ 采用 USB 或 RS232 接口与PC 连接, 由PC 进行编程和控制。
- ⑨ 多种供电方式选择, 宽电压自适应 3.3~20V, 目标板供电、USB 供电、电源供电。
- ⑩ 支持全系列 AVR 单片机的 ISP 下载和具有 JTAG 接口的在片仿真。

### V3.0 新增功能特性:

- ① 精致外观, 与ATmel原装开发工具相同。
- ② 适应最新AVRstudio4.13 软件, 使用方便。
- ③ 更快的识别速度, 插拔的瞬间完成。

### V3.0B新增功能:

- ① JTAG功能和ISP功能可以独立自动升级, 随AVR studio更新自己的内核。

## 3、支持芯片

### ISP下载:

#### AT Tiny系列:

ATtiny12、ATtiny13、ATtiny15、ATtiny22、ATtiny24、ATtiny26、ATtiny2313

#### AT90 系列:

AT90S1200、AT90S2313、AT90S/LS2323、AT90S/LS2343、AT90S/LS2333、  
AT90S4414、AT90S/LS4433、AT90S/LS4434、AT90S8515、AT90S/LS8535

#### AT Mega系列:

ATmega8、ATmega16、ATmega32、ATmega48、ATmega64、ATmega88、ATmega103、  
ATmega128、ATmega161、ATmega162、ATmega163、ATmega165、ATmega169、  
ATmega323、ATmega325、ATmega329、ATmega644、ATmega645、ATmega649、  
ATmega2560、ATmega2561、ATmega3250、ATmega3290、ATmega6450、  
ATmega6490、ATmega8515、ATmega8535、AT90CAN128、AT90PWM2、AT90PWM3

#### 其他:

AT86RF401、AT89S51、AT89S52

#### JTAG仿真 (具有JTAG 接口):

AT90CAN128、ATmega128、ATmega128L、ATmega16、ATmega162、ATmega162V、  
ATmega165、ATmega165V、ATmega169、ATmega169V、ATmega16L、ATmega32、  
ATmega323、ATmega323L、ATmega32L、ATmega64、ATmega64L

## 4、产品清单

- ① AVR JTAG与ISP 二合一 V3.0B 1 台
- ② 本站产品说明书 1 本
- ③ 优质USB连接线 1 条
- ④ 标准串口延长线 9-pin RS232 1 条
- ⑤ 直流DC电源电缆 1 条
- ⑥ 技术资料及驱动光盘 1 张

## 5、 系统要求

PC 软硬件至少满足：

- Pentium (Pentium II 或以上)
- 64 MB RAM
- 100 MB 空余硬盘空间(用来安装AVR Studio 4.XX)
- Windows 操作系统如Windows 2000 或Windows XP
- 115200 Baud RS-232 Port (COM Port)或USB 接口

## 6、 关键术语

什么是JTAG：

JTAG 接口是一个符合IEEE 1149.1 标准的 4 线的测试存取端口控制器 (Test Access Port (TAP) controller)。这个IEEE 标准制定了一套标准的方法，采用了边界扫描技术(Boundary Scan)，用于有效的对芯片进行测试。Atmel AVR 芯片扩展了这项功能，使其能完全支持编程下载和片上调试功能。JTAGICE使用标准的JTAG 接口，使用户可以对目标系统上运行的单片机进行实时的仿真。( AVR On-Chip Debug (AVROCD)) 协议能够让用户对AVR 单片机的内部资源进行全部的控制。与传统仿真器相比，JTAGICE的花费很小，但却能实现更准确的仿真。

什么是ISP：

ISP是In System Program的缩写，意思是在系统编程。目前几乎所有的AVR芯片都具备ISP 接口，可通过ISP接口进行编程。它一共使用了两条电源线：VCC、GND，三条信号线：SCK、MOSI、MISO，以及复位线：RESET。由于仅仅使用了几个数据线，所以我们亦常将其称为串行编程。

需要说明的是：大部分AVR的ISP端口为MCU的 SCK，MOSI，MISO，RESET引脚，但少部分AVR的ISP端口则不是使用这些接口，例如：ATmega64、ATmega128，它们使用的ISP端口是：SCK，PDI，PDO，RESET。

JTAG与ISP的区别和联系：

JTAG和ISP都是用于PC和AVR芯片通讯的方法，ISP只能用于下载，而JTAG既可以用于下载也可以用于在线调试。绝大多数AVR芯片都有SPI接口，可以用于ISP下载，但并不是所有的器件都支持JTAG，所以拥有一套JTAG与ISP二合一的工具非常有必要，具体的支持芯片，前面已经列出。