

AVR 单片机快速入门

AVR 单片机——是 ATMEL 公司推出 RISC 精简指令集的高速 8 位单片机。
AVR 单片机强大的功能——详细请阅读 AVR 单片机数据手册。

草稿

本教程主要以 ATmega8 单片机与 CodevisionAVR 开发环境作实例讲解。

采用 CodevisionAVR 开发 AVR 单片机让你倍感轻松，非常适合单片机初学者、电子工程师及一些工程管理人员。

之所以让你学习倍感轻松那是因为 CodevisionAVR 拥有一个代码自动生成器。有了这个代码生成器，你可以：

- 1、不需要看单片机数据表就能随意配置输出输出引脚。
- 2、不需要知道串口通信是什么原理，不用查看控制串口寄存器是如何操作的，只需要处理收到的数据与发出去的数据就行了。
- 3、不需要知道 ADC 是什么工作的，也不需要去核对数据表上的 ADC 操作方法，如果你需要读取 ADC0 引脚上的模拟信号量到变量“a”可以这样操作“a=read_adc(0);”函数“read_adc(unsigned char channels)” CodevisionAVR 已为你做好了。

- 4、不用了解标准 LCD 液晶屏是如何驱动的，更不用关心 MCU 是什么工作的，只需将 LCD 按标准连接到你的项目中就行了，想显示什么就显示什么。

还有更多功能如 PWM、IIC 通信、单总线通信、SPI 通信等，都可以让你轻松应用，当然还有操作单片机中断及定时器及片内的模拟比较器等都把易如反掌。

本教程特色：

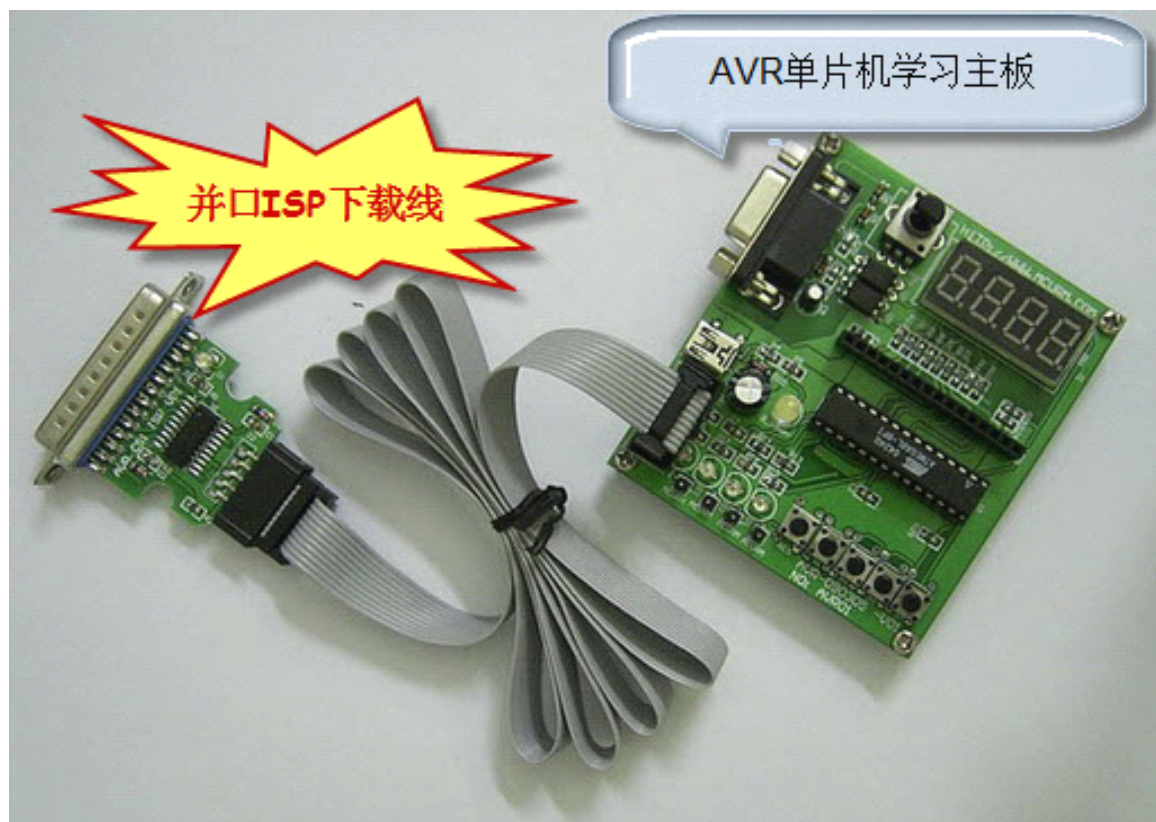
本教程直接从简单实例开始到最后完成复杂的功能，不管您是否了解过单片机与学习过 C 语言，都会让您感觉学习单片很轻松，同时感受学会单片机编程的乐趣。

没学过 C 语言——本教程实例中对 C 语言有详细说明，由浅入深，让您感觉 C 语言编程原来如此简单。

实例目录:

- 1、[静态发光 LED 显示实验](#)
- 1、[流水灯实验](#)
- 2、[按键操作实验](#)
- 3、[LED 数码管显示实验](#)
- 4、[ADC 实验](#)
- 5、[PWM 实验](#)
- 6、[PWM to DAC 实验](#)
- 7、[三角波输出实验](#)
- 8、[正弦波输出实验](#)
- 9、[串口通信实验](#)
- 10、[LCD1602 显示实验](#)

配合本实教程所用的学习板:



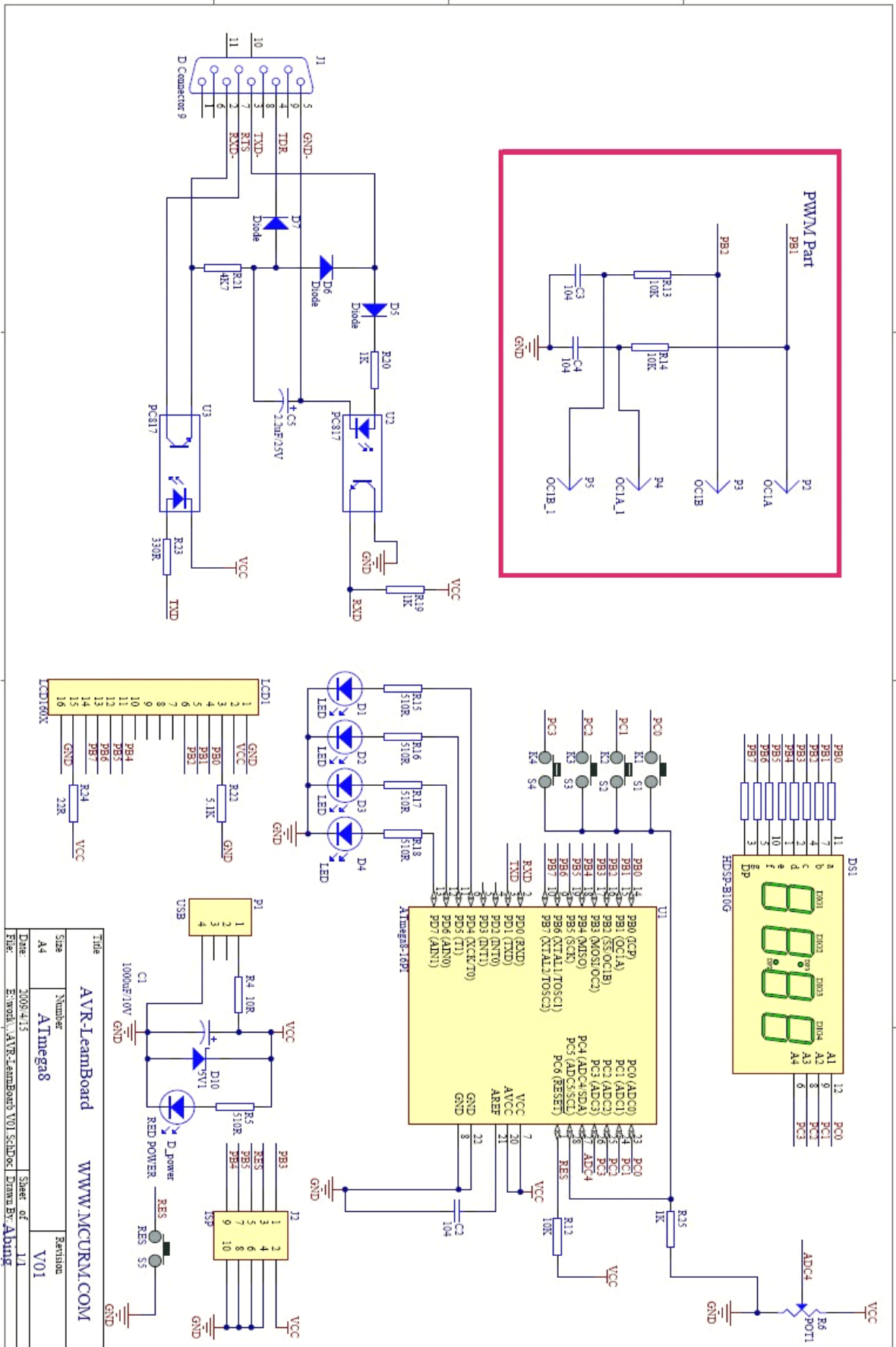
本学习实验板特色:
内置振荡器

学习中所需的软件都有在配套的光盘里, 请先安装好所编译软件 CodevisionAVR 及程序下载软件 SLISP。

草稿

草稿

实验板原理图:

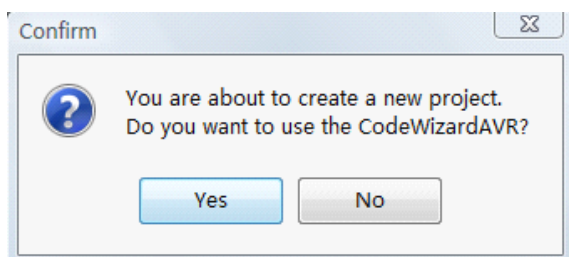


实例一：静态 LED 发光二极管显示

实例功能：点亮发光二极管 D1。

从原理图上可以看出发光二极管 D1 接于 PORTD.4 上，如点亮 D1 实例代码“PORTD.4=1;”即可。请按以下步骤开始：

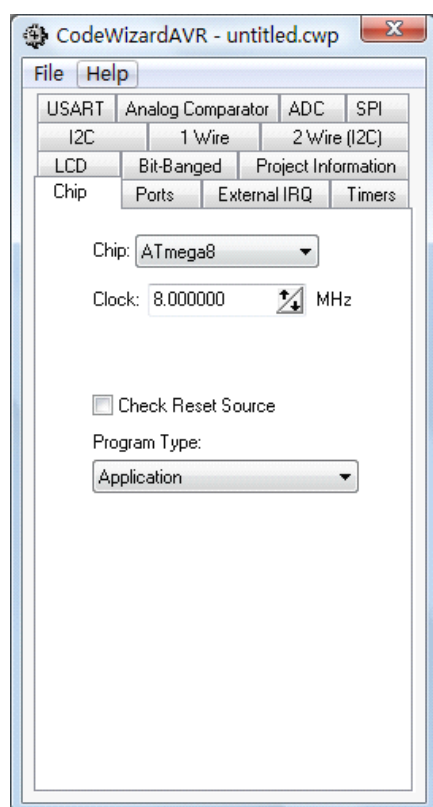
打开 CodevisionAVR，选择 File 点击 New，在弹出的对话框中选择“project”然后点击 OK，接下来将弹出如下提示框：



草稿

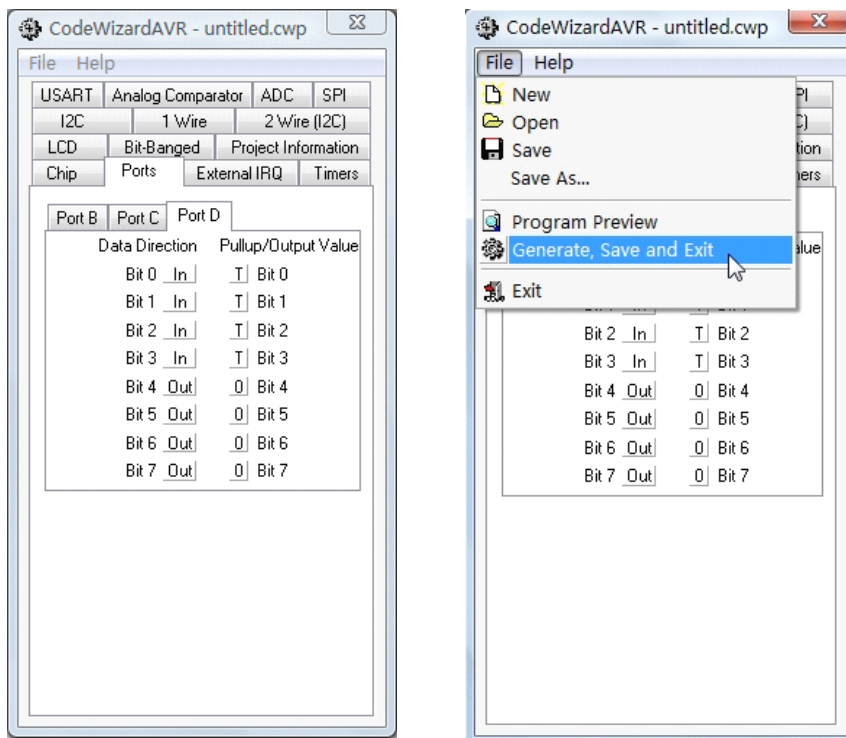
注意了，这个提示框就是提示是否采用代码向导，也就是说是否采用代码自动生成器，在这当然先选是啦，否则不能体验到上面所说的那么多好处。

当选择“Yes”会弹出如下对话框：

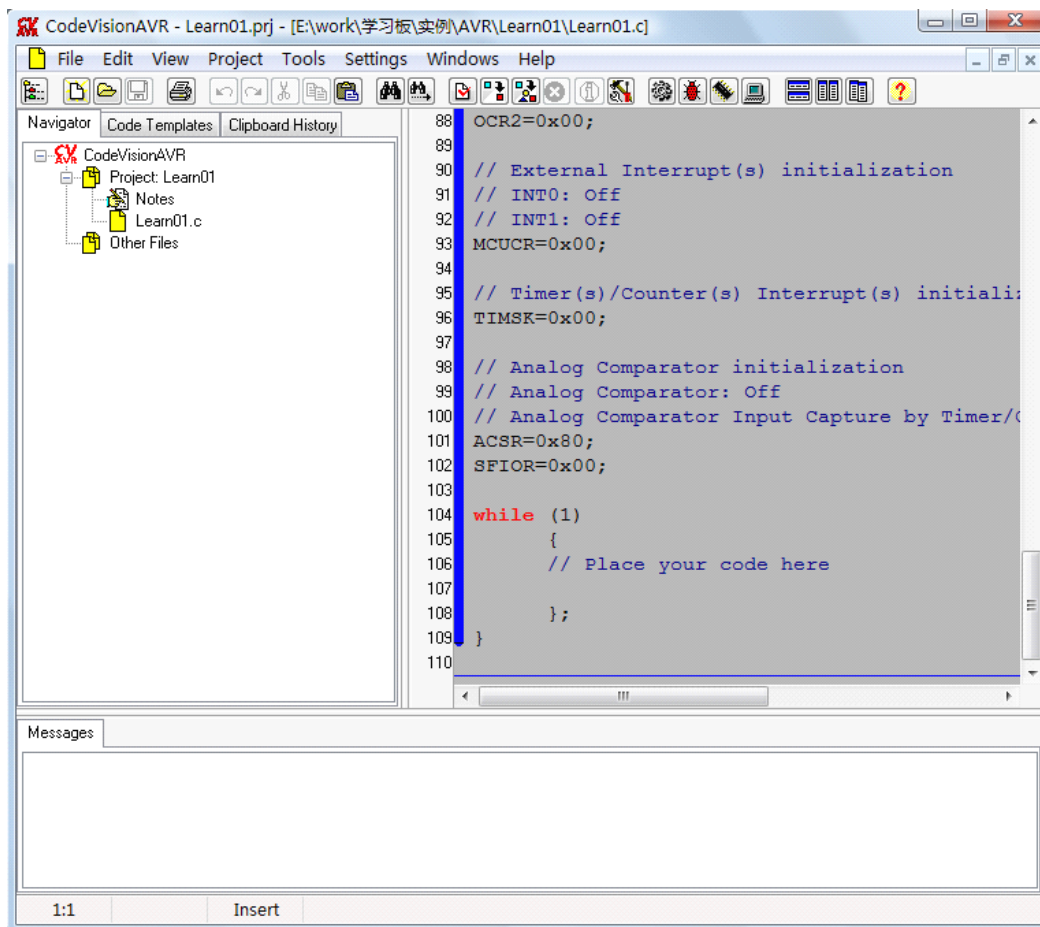


1. 在 chip 选项框中选择芯片 ATmega8。
2. 注意在 Clock 选项框中所选择的时钟频率应与实际频率一致，单片机实际运行频率是外部晶振的频率或是与编程时所选的内部振荡器的频率。正确填写这个选项框可以让代码生成器为你产生精确的定时器定时及 PWM 操作等。

接下来选择“Ports”选项卡，然后再选择“Port D”选项卡，如要让 LED 亮，就需要让单片机接 LED 的引脚设为输出，如原理图中的 D1—D4 对应接于 PORTD.4—PORTD.7，所下图所示已将 PORTD.4—PORTD.7 设为输出端口。



最后选择“File”点击“Generate, Save and Exit”然后提示保存文件名及文件位置，按操作提示完成就可以了。到这一步 CodevisionAVR 已按所配置要求及按你输入的文件名生成了一个初始化代码，如下图的 Learn01.c




到这我们就要写入需要完任务的代码，我们这一实例的任务是点亮发光二极管 D1；其实这个任务很简单，D1 接于 PORTD.4 上，要点亮 D1 只要 PORTD.4 输出高电平 D1 就亮了，怎样让 PORTD.4 输出高电平呢？在代码中加入“PORTD.4=1;”就这么简单可这代码加在哪呢？

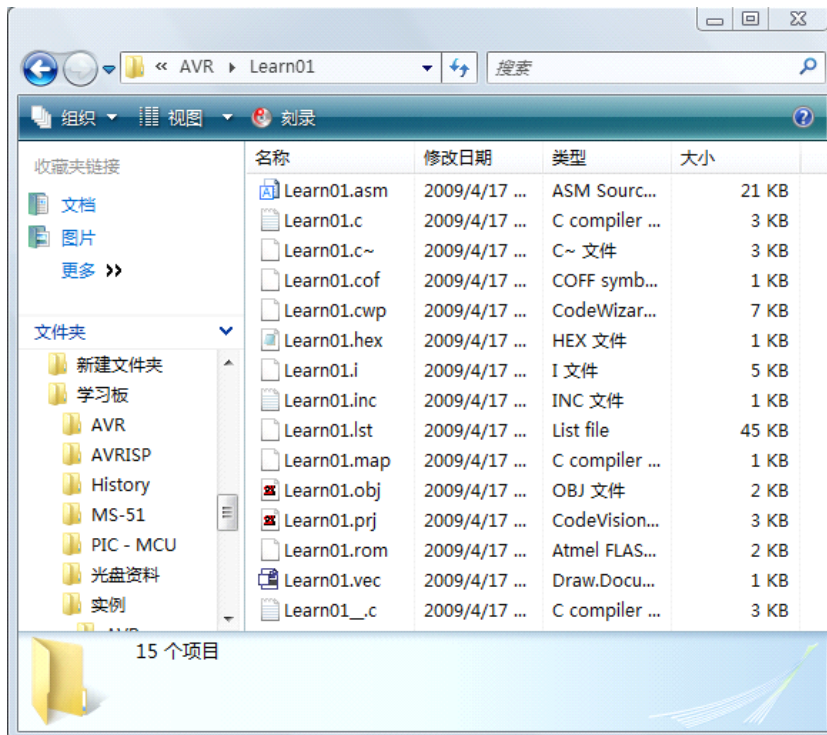
```
104 while (1)
105 {
106     // Place your code here
107     PORTD.4=1;
108 }
```

当然是加在这啦

草稿

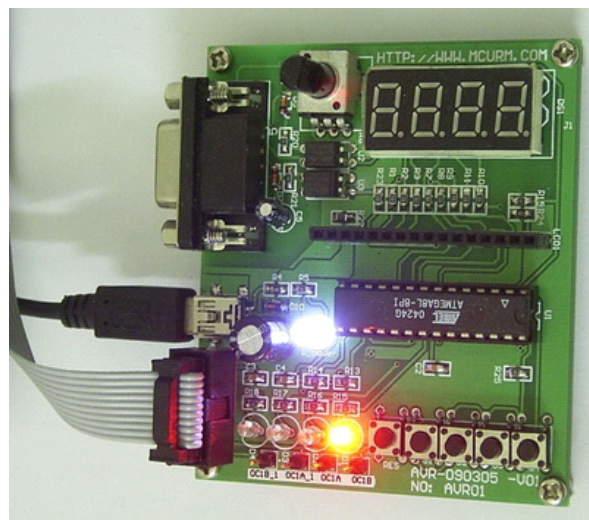
呵呵！“Place your code here”就是指“将你的代码放在这”不懂英文的朋友也不要紧，在单片机编程中不过也就那么几个英文，看多了就都记住了。

接下来请点击 CodevisionAVR 工具栏上的按钮，这时会弹出一个信息提示框，点击 OK 就可以了，同时会在文件所保存在的文件夹中生成了另外几个文件，其中有一个就是用于烧写到单片机的 HEX 文件。如下图所示：



上图中“Learn01.hex”就是用来写入单片内的文件，将并口下载线连接到电脑并口上，然后启动 SLISP，按提示将 Learn01.hex 写入到单片机中。

如右图所示程序运行的结果，白色的灯是电源指示灯，靠按键的那个红灯就是 D1。



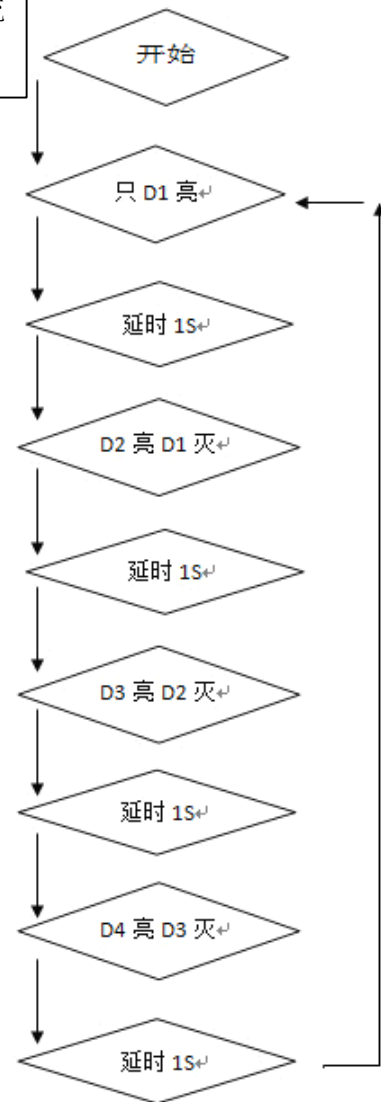
实例二：流水灯实验

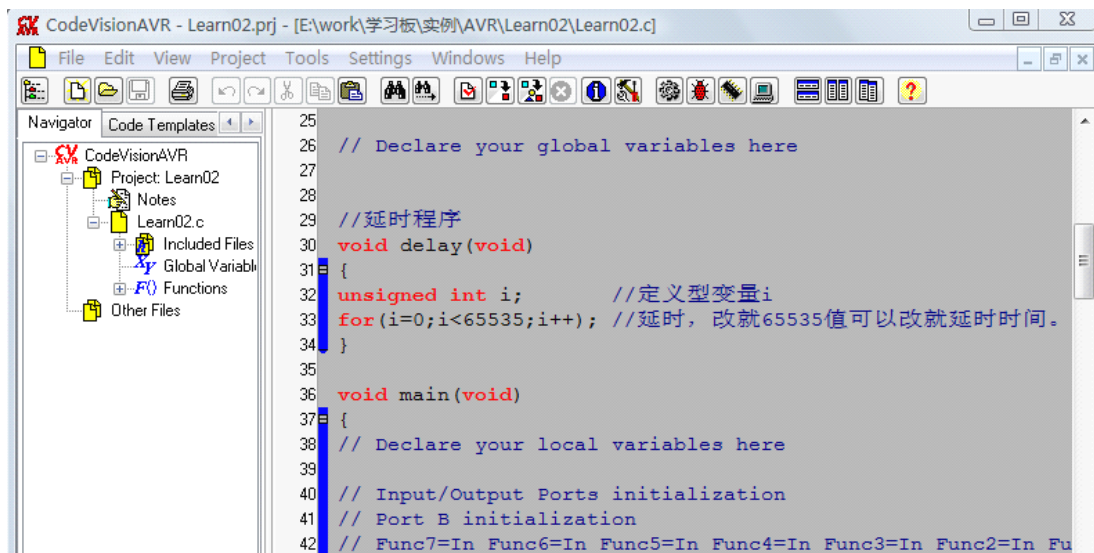
实验任务：如右图流程图所示。

```
PORTD=0;    //所有 LED 灭
PORTD.4=1;  //点亮 D1
delay();    //延时
PORTD.4=0;  //关掉 D1
PORTD.5=1;  //打开 D2
delay();    //延时
PORTD.5=0;  //关掉 D2
PORTD.6=1;  //打开 D3
delay();    //延时
PORTD.6=0;  //关掉 D3
PORTD.7=1;  //打开 D4
delay();    //延时

//延时程序
void delay(void)
{
    unsigned int i;        //定义型变量 i
    for(i=0;i<65535;i++); //延时，改就 65535 值可以改就延时时间。
}
```

```
112 while (1)
113     {
114         // Place your code here
115         PORTD=0;    //所有LED灭
116         PORTD.4=1;  //点亮D1
117         delay();    //延时
118         PORTD.4=0;  //关掉D1
119         PORTD.5=1;  //打开D2
120         delay();    //延时
121         PORTD.5=0;  //关掉D2
122         PORTD.6=1;  //打开D3
123         delay();    //延时
124         PORTD.6=0;  //关掉D3
125         PORTD.7=1;  //打开D4
126         delay();    //延时
127     };
```





```
25 // Declare your global variables here
26
27
28
29 //延时程序
30 void delay(void)
31 {
32     unsigned int i; //定义型变量i
33     for(i=0;i<65535;i++); //延时, 改就65535值可以改就延时时间。
34 }
35
36 void main(void)
37 {
38     // Declare your local variables here
39
40     // Input/Output Ports initialization
41     // Port B initialization
42     // Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Fu
```

如果上面实例看得不是很明白的朋友，请认真阅读以下内容：

数据类型：么什么是数据类型呢？

关于数据类型大家可以这么理解，就是定义一个变量的最大值，常用的数据类型有字符型（Char），整型（Int），Char 的范围是(-127,+127)，就是说如果定义一个字符型变量“i” i 的取值就在(-127,+127)之间。整型数 int 的范围是（-32768， +32768），在实际应用中通常加入（unsigned）无符号数据类型，如下例子：

Unsigned char i; //定义一个无符字符型变量 i。

这时变量“i”的范围是 0-255，也就是说“i”的值最大为 255，如果 i=255 在执行 i=i+1;或是 i++;后 i 值将变为零。

Unsigned int i; //定义一个无符号整型变量。

无符号整型变量取值范围是 0-65535。可能有的朋友问，为什么搞这么复杂？直接用最大的数据类型不就可以了吗？在单片机中处理小数及负数都很慢而且很占用资源，当一个变量被定义为 Char 时占用一个字节空间，而定义为 int 时占用两个字节空间，所以程序开发应根据实际情况来定义变量数据类型。

C 语言应用很灵活，常用的运算都能用上如（+，-，x，/）例子中用到（i++）同等于（i=i+1）即自增 1 的意思，（i--）同理就是自减 1。

例子中的 for 语句详解，for(i=0;i<65535;i++); i=0 是将 0 赋值给予 i，i<65535 指如果 i 小于 65535 那么执行后面的 i++。这语句的意思就是 i 从零自加 1 到 65535 所用的时间就是我们需要的延时的时间。

小记事：有位初学者为加大延时时间，把 for(i=0;i<65535;i++);改为 for(i=0;i<100000;i++);上面说过对于 int 类型最大值 65535，而 100000 已大于 65535，所以不管 i 什么加都达不到数，所以程序就在这个地方“死机”了。

本实例重点：

- 1， 掌握变量的定义，及变量数据类型。
- 2， for 函数的使用。

实例三：按键操作实验

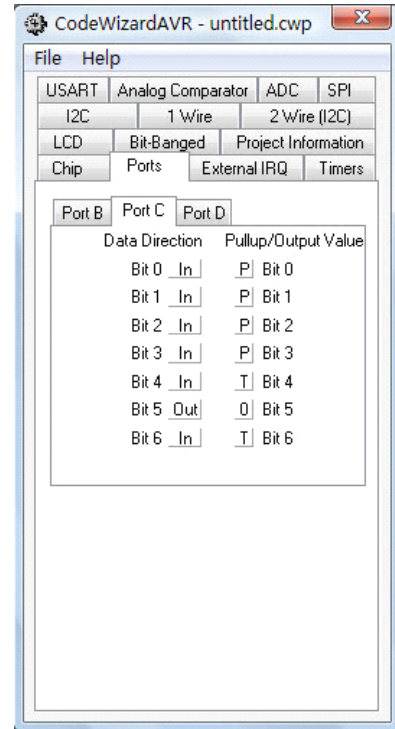
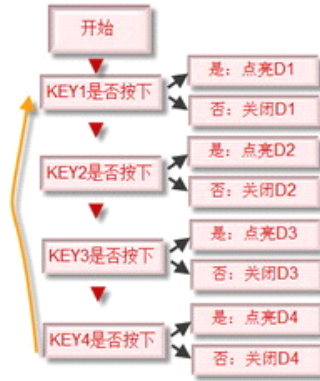
实验功能：当 KEY1 按下时 D1 亮，KEY2 按下时 D2 亮，KEY3 按下时 D3 亮，KEY4 按下时 D4 亮。

程序清单：

```
#define S1 PINC.0
#define S2 PINC.1
#define S3 PINC.2
#define S4 PINC.3

#define D1 PORTD.4
#define D2 PORTD.5
#define D3 PORTD.6
#define D4 PORTD.7
```

```
void key(void) //按键处理程序
{
if(S1==0)D1=1; //如果 S1 按下,D1 亮
    else D1=0; //否则 D1 灭
if(S2==0)D2=1; //如果 S2 按下,D2 亮
    else D2=0; //否则 D2 灭
if(S3==0)D3=1; //如果 S3 按下,D3 亮
    else D3=0; //否则 D3 灭
if(S4==0)D4=1; //如果 S4 按下,D4 亮
    else D4=0; //否则 D4 灭
}
```



在原理图中可知 S1—S4 共用一条接于 PC5 引脚上，其它四个按键引脚接于 PORTC.0—PORTC.3 上。如要让单片机检测到按键，可将 PC5 设为输出并且输出低电平，然后 PORTC.0—PORTC.3 设计为输入并且使能上拉电阻，如上图所示采用 CodeWizardAVR 来配置。

配置好后 PORTC.0—PORTC.4 为输入端口，并由单片机内部上拉电阻拉为高电平，当按键按下时被拉到低电平。也就是说当检测到 PORTC.0—PORTC.4 有低电平时表示有按键按下。

“=”在 C 语言中叫“赋值号”而“等号”是“==”。如“PORTD.4=1;”是指将 1 赋值给 PORTD.4，“if(S1==0)”是指如果 S1 等于 0，这是很多初学者容易搞错的。

#define 语句顾名思义就是“定义”的意思，例中：#define D1 PORTD.4 是指将 D1 定义为 PORTD.4 有了这个定义后程序中执行 D1=1; 与 PORTD.4=1; 是相同的。同样可以这样定义：#define S1 PINC.0，当按键 S1 按下时 D1 点亮可以这样写：if(S1==1)D1=1;else D1=0; 也可以这样写：if(PINC.0==0)PORTD.4=1; else PORTD.4=0; 可以看出前式要直观多了也利于理解。

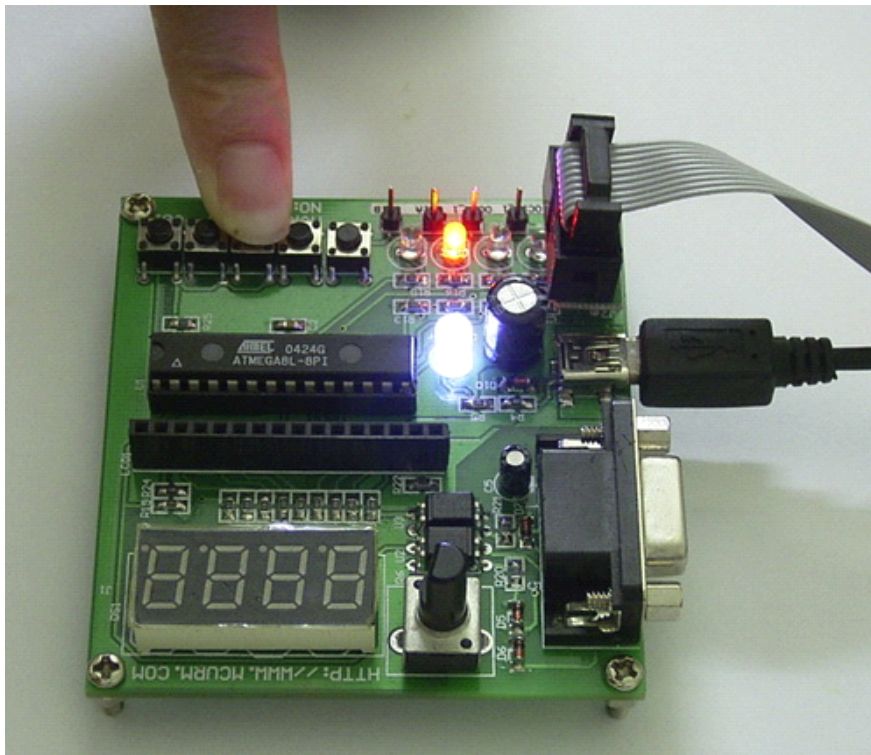
if 函数，“if”就是“如果”的意思，else 就是“否则、那么”的意思，C 语言是一个很接近人思维的语言，如：if(你没有迟到)这个月你有奖金；else 扣除你的奖金；呵呵！知道 C 语言就这么简单了吧？

C 语言灵活性很强，以上实例可以通过多种方法实现，就像人们常讲的千万大道条条通北京。

```
CodeVisionAVR - Learn03.prj - [E:\work\学习板\实例\AVR\Learn03\Learn03.c]
File Edit View Project Tools Settings Windows Help
Navigator Code Ter
CodeVisionAVR
  Project: LearnC
  Notes
  Learn03.c
  Include
  Global
  Function
  Other Files
24 #include <mega8.h>
25
26 // Declare your global variables here
27 #define S1 PINC.0
28 #define S2 PINC.1
29 #define S3 PINC.2
30 #define S4 PINC.3
31
32 #define D1 PORTD.4
33 #define D2 PORTD.5
34 #define D3 PORTD.6
35 #define D4 PORTD.7
36
37 void key(void) //按键处理程序
38 {
39     if (s1==0) D1=1; //如果s1按下, D1亮
40         else D1=0; //否则D1灭
41     if (s2==0) D2=1; //如果s2按下, D2亮
42         else D2=0; //否则D2灭
43     if (s3==0) D3=1; //如果s3按下, D3亮
44         else D3=0; //否则D3灭
45     if (s4==0) D4=1; //如果s4按下, D4亮
46         else D4=0; //否则D4灭
47 }
48
49 void main(void)
50 {
51     // Declare your local variables here
52     ...

```

```
124
125 while (1)
126 {
127     // Place your code here
128     key();
129 }
```

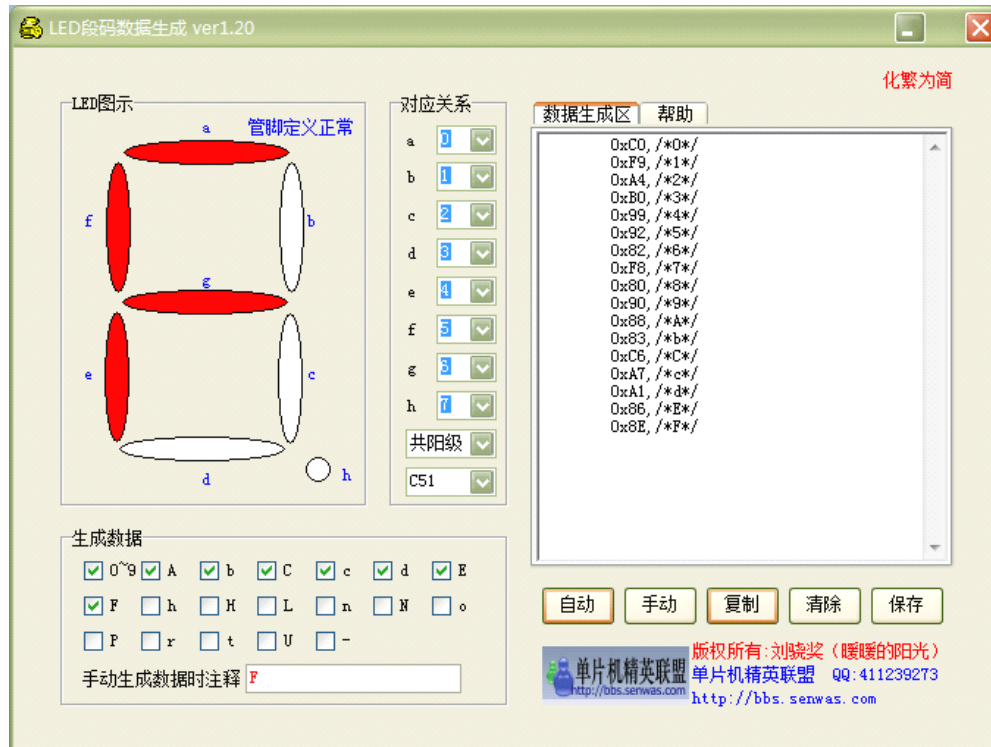


实例四：LED 数码管显示

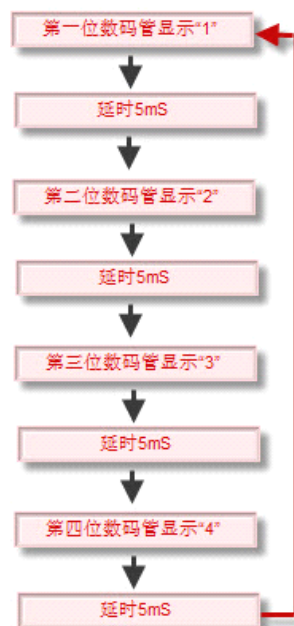
实例功能：让 LED 数码管显示 1234

请看原理图，LED 的 7 段接于 PB 口的 PB0—PB6，PB7 接于数码管的小数点，LED 数码管公共极 A1—A4 分别接于 PC0—PC3，实验板上的数码管采用共阳数码管。如果要让第一位显示“8”那么 A1 应输出高电平，而 PB0—PB6 应拉到低电平。

如要显示“5”呢？或是显示“b”应向 PB 口写什么数据多少呢？请看下面一个实用工具：

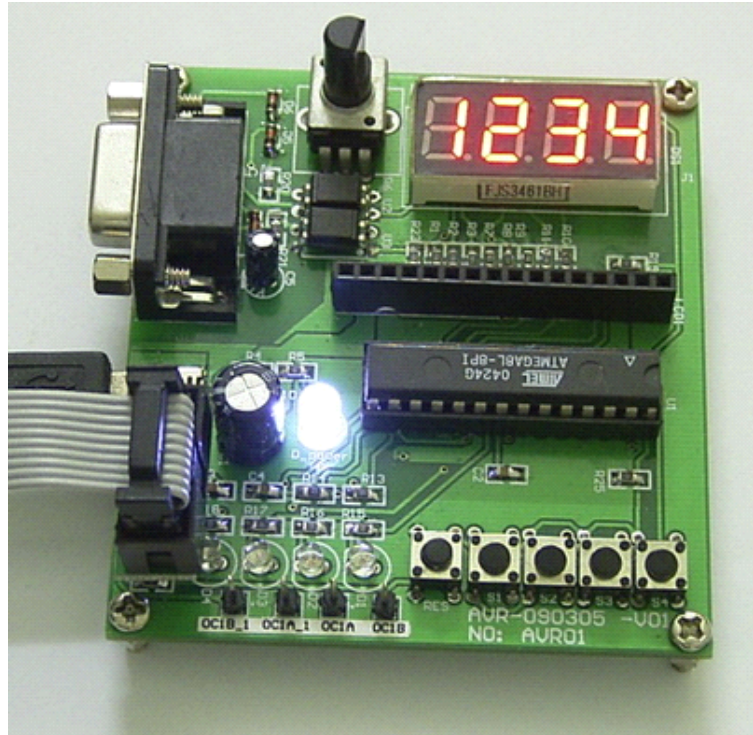


看上图：如果要显示数字“1”向 PB 口写入 0xf9，如果要显示“5”就要向 PB 口写入 0x92，有了这工具就不需要手工一个个去算了，多方便啊。（此程序在光盘里有）



首先定义两个数组:

```
unsigned char display_code[]={ //显示代码数组
    0xC0,/*0*/
    0xF9,/*1*/
    0xA4,/*2*/
    0xB0,/*3*/
    0x99,/*4*/
    0x92,/*5*/
    0x82,/*6*/
    0xF8,/*7*/
    0x80,/*8*/
    0x90,/*9*/
    0x88,/*A 10*/
    0x83,/*B 11*/
    0xC6,/*C 12*/
    0xA7,/*C 13*/
    0xA1,/*D 14*/
    0x86,/*E 15*/
    0x8E /*F 16*/
};
```



unsigned char display_buf[]={1,2,3,4}; //显示缓冲数组。如果我们相让显示“1234”改为“8665”只需要将缓冲数组时的内容改为“8,6,6,5”就可以了。在下个实例我们将讲 ADC（模拟到数字转换）读取引脚电位信号处理后如何更新到这个缓冲数组，然后让 LED 显示。

```
void delay5ms(void) //延时程序
{
    unsigned int i;
    for(i=0;i<800;i++);
}
```

```
void led_display(void)
{
    PORTC=0; //显示第一位
    PORTC.0=1;
    PORTB=display_code[display_buf[0]];
    delay5ms();
    PORTC=0; //显示第二位
    PORTC.1=1;
    PORTB=display_code[display_buf[1]];
    delay5ms();
    PORTC=0; //显示第三位
    PORTC.2=1;
    PORTB=display_code[display_buf[2]];
    delay5ms();
    PORTC=0; //显示第四位
    PORTC.3=1;
    PORTB=display_code[display_buf[3]];
    delay5ms();
}
```



C 语言灵活多样，唯有常练习学习，才能写出精简高效的程序，左边的非常直观，利于学习者掌握，程序还可以这样写：

```
Void led_display(void)
{
    Unsigned char i;
    For(i=0;i<4;i++)
    {
        PORTC=1<<i;
        PORTB=display_code[display_buf[i]]
    }
    Delay5ms();
};
```

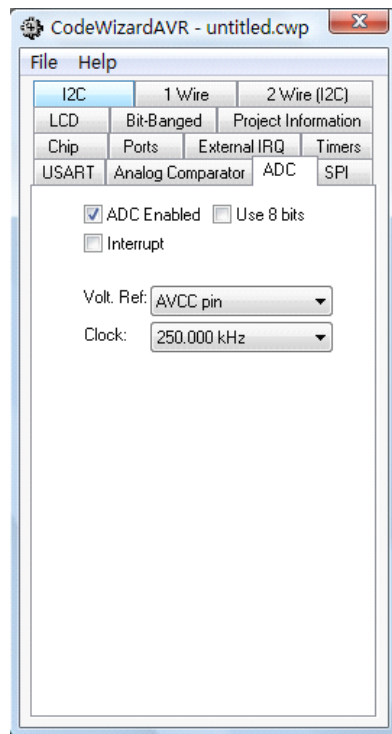
看完全一样功能的程序是不是简洁多了。

上面的“<<”是左移号，也就是就当 i=3 时，PORTC.3=1;

实例五：ADC 采样实验

实验功能：读取 ADC4 引脚上的模拟信号并通过 LED 显示出来。

如原理图所示，电位器中心抽头接于单片机 ADC4 引脚上，ATmega8 有 10 位精度的 ADC，在前面我们讲过采用 CodevisionAVR，让我们编程变得更加简单，不需要去看单片机数据表上的 ADC 是怎操作及怎么工作的，只要几个简单的选择就可以了。



```
30 // Read the AD conversion result
31 unsigned int read_adc(unsigned char adc_input)
32 {
33     ADMUX=adc_input | (ADC_VREF_TYPE & 0xff);
34     // Delay needed for the stabilization of the ADC input voltage
35     delay_us(10);
36     // Start the AD conversion
37     ADCSRA|=0x40;
38     // Wait for the AD conversion to complete
39     while ((ADCSRA & 0x10)==0);
40     ADCSRA|=0x10;
41     return ADCW;
42 }
```

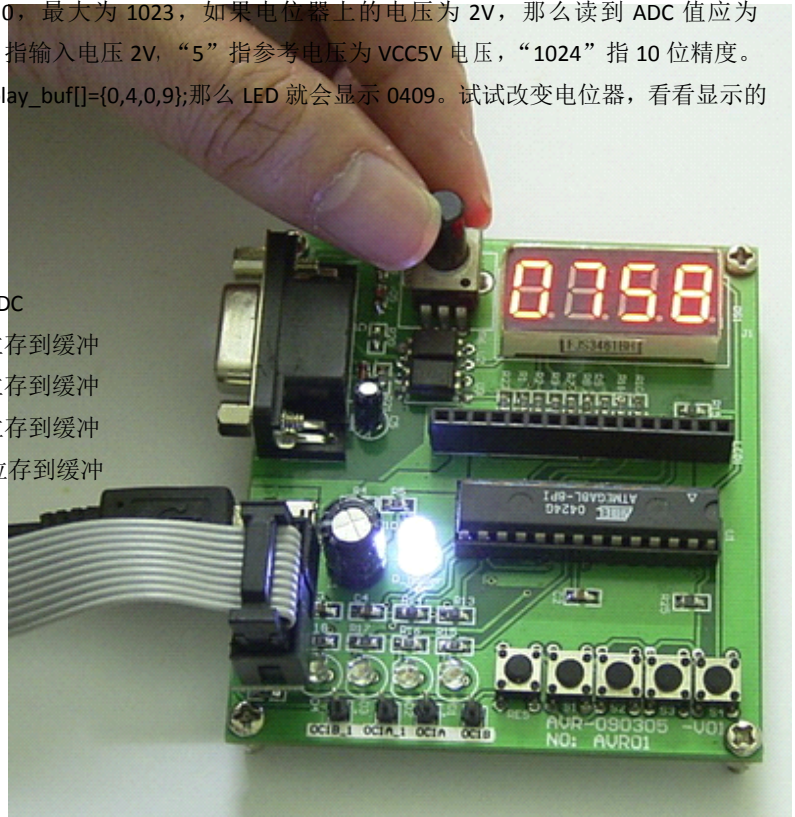
左图所示，ADC 选项操作很简单，选择 ADC 使能“ADC Enabled”在 Volt Ref 中选择 ADC 的参考电压，这个例子中我们先择“AVCC pin”时钟频率选择“250KHz”退出保存后可以看到已产生如上图所示 ADC 读取函数，如果我们要将 ADC4 引脚上的电压信号读入到一个变量 a 中，代码可这样写：`a=read_adc(4);`

接下来的问题就是将读到的 ADC 值处理到 LED 数码管显示上。

10 位 ADC 采样到的 ADC 值最小为 0，最大为 1023，如果电位器上的电压为 2V，那么读到 ADC 值应为 $2/5*1023=409$ (注意，没有小数)式中“2”指输入电压 2V，“5”指参考电压为 VCC5V 电压，“1024”指 10 位精度。

我位只需要把读到的 409 转换成 `display_buf[]={0,4,0,9}`；那么 LED 就会显示 0409。试试改变电位器，看看显示的变化吧？

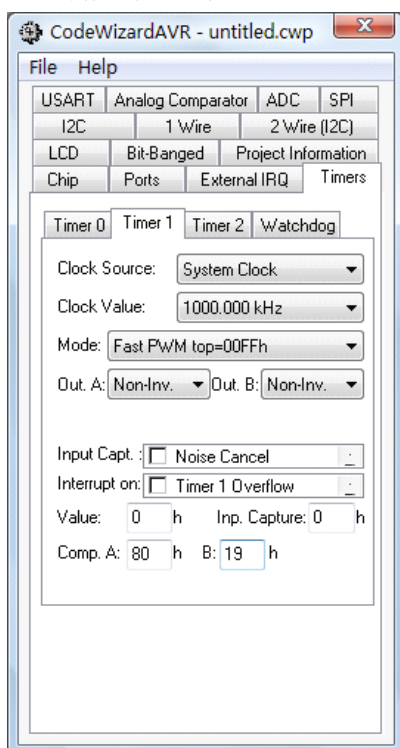
```
void adc_pro(void) //ADC 处理程序
{
    unsigned int a;
    a=read_adc(4); //读取 ADC
    display_buf[3]=a%10; //将个位存到缓冲
    display_buf[2]=(a/10)%10; //将十位存到缓冲
    display_buf[1]=(a/100)%10; //将百位存到缓冲
    display_buf[0]=a/1000; //将千位存到缓冲
}
```



实例六：PWM 实验

实例功能：在 PWM 引脚 OC1A 上输出 50% 占空比的方法，在 OC1B 上输出 10% 占空比的方法。

带 PWM 模块的单片机，输出 PWM 信号是由硬件完成的，如果要改变 PWM 占空比，CPU 只需将新占空比值写入占空比寄存器就可以了。



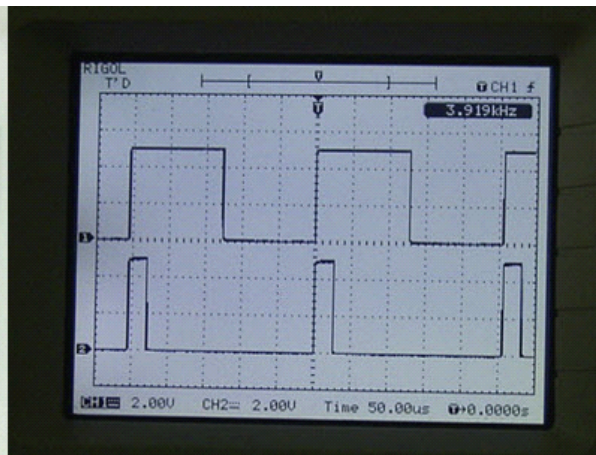
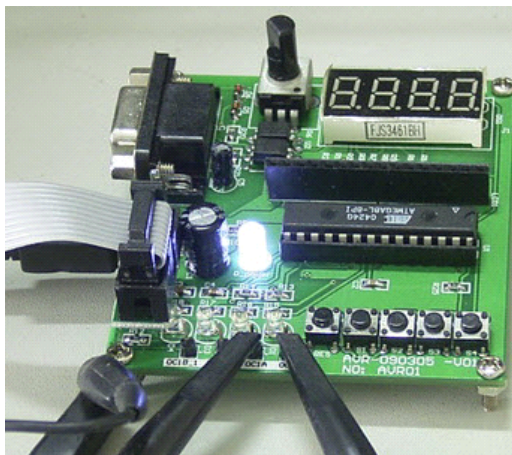
PWM 占用 Timer1 定时器，在左图中可以看到配置 PWM 方法很简单。

- 1, “Clock Source ” 选择时钟
- 2, “Clock Value” 选择时钟频率
- 3, “Mode ” 模式选择。
- 4, Out A 指 OC1A, Out B 指 OC1B
- 5, 最下面一行里写入占空比值。

在 PWM 模式里我们选择快速 8 位 PWM，所以 50% 占空比值为 80H，10% 占空比值为 19H。

以上几个选项详细说明请看 ATmega8 数据表，但特别提示初学习，看数据表时用哪部分才看哪部分，要不然一份几百页的数据表看到什么时候，而且很多都看不懂，等大部份功能都会用或了解时再整体系统的看，这样看得就更容易理解。

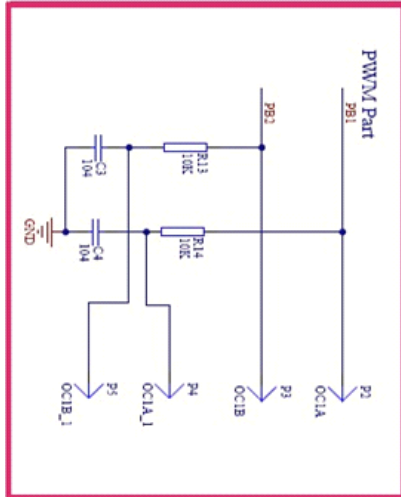
另外需要注意 OC1A 与 OC1B 接于 PB1 与 PB2 上，如要输出 PWM 需将 PB1 与 PB2 设为输出。



如果只是输出一个固定占空比的 PWM 信号，代码自动生成器生成的程序就能工作了，所以我们不再需要写相关程序，除非工作过程中需要改占空比，如 PWM 产生三角波与正弦波实验等。

实例七: PWM to DAC

实例功能: 利用 PWM 实现 DAC 输出



利用单片机的 PWM 功能实现 DAC 输出是一种很常用而且价格低廉的方法。

在上例子中, OC1A 与 OC1B 输出 PWM 信号, 如经一电阻与电容滤波后就变为一个与占空比对应的直流电压, 在上例 OC1A 输出 50% 占空比, 由于 VCC 等于 5V 所以对应 OC1A_1 输出电压为: $5 \times 0.5 = 2.5V$ 。同样 OC1B_1 输出为: $5 \times 0.1 = 0.5V$ 。

如要改变输出电压, 只要改变占空比就行了。

实例八: 三角波输出实验

实例功能: 在 OC1A_1 上输出一个三角波信号

输出三角波是通过 PWM to DAC 实现的, 在上面例子中, 是输出一个固定电压, 如果输出一个从零开始到最大然后再返回零, 这样就生成了一个三角波了。

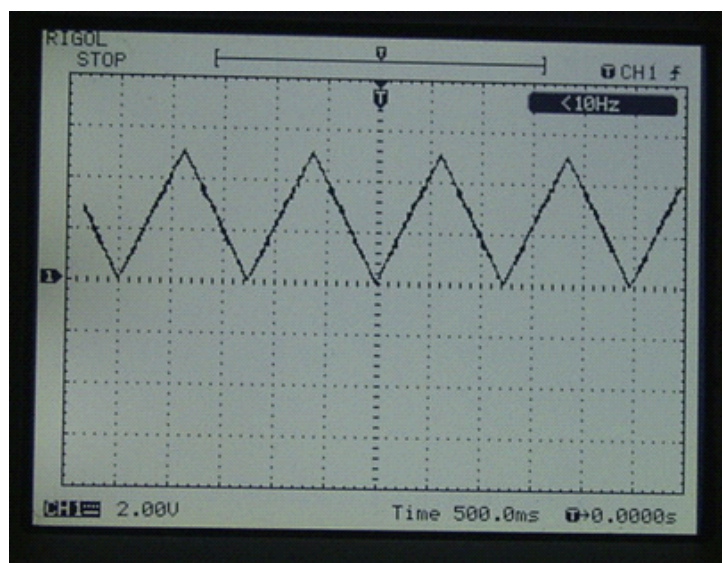
程序实例:

```
void delay1ms(void) //延时程序
{
    unsigned int i;
    for(i=0;i<300;i++);
}
```

```
void waveform(void)
{
    unsigned char i;
    for(i=0;i<255;i++) //占空比由零往上加
    {
        OCR1A=i;
        delay1ms();//延时
    };
    for(i=255;i>0;i--) //占空比从大向下减
    {
        OCR1A=i;
        delay1ms();//延时
    };
}
```

程序中, OCR1A 为占空比寄存器, 只要改变 OCR1A 的值就能改变 OC1A 引脚上的占空比, 同样要改变 OC1B 引脚上的占空比只要改变 OCR1B 的值就可以了。

在这个三角波形产生程序中, 改变延时时间可以改变三角波的频率, 三角波的幅度可以控制 OCR1A 来实现。

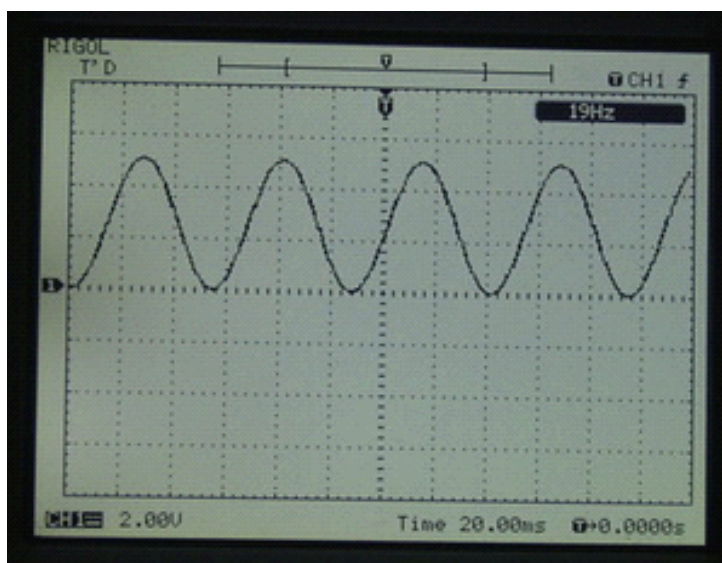


实例九：正弦波输出实验

实例功能：在 OC1A_1 上输出正弦波信号

```
24 #include <mega8.h>
25 #include<math.h>
26
27 // Declare your global variables here
28 unsigned char sin_table[100]; //定义一个数组用于存放正弦表
29
30 void inital_sin_table(void) //初始化正弦表
31 {
32     unsigned char i;
33     for(i=0;i<100;i++)
34     { //式中127指中点值, 0.0628是指一个周期分100份即: 2Pi/100=2X3.14/100
35         sin_table[i]=127+sin(0.0628*i)*128;
36     };
37 }
38
39 void delay(void) //延时程序
40 {
41     unsigned int i;
42     for(i=0;i<60;i++);
43 }
44
45 void waveform(void) //波形产生程序
46 {
47     unsigned char i;
48     for(i=0;i<100;i++)
49     {
50         OCR1A=sin_table[i];//加载占空比值
51         delay();
52     }; //延时
53 }
```

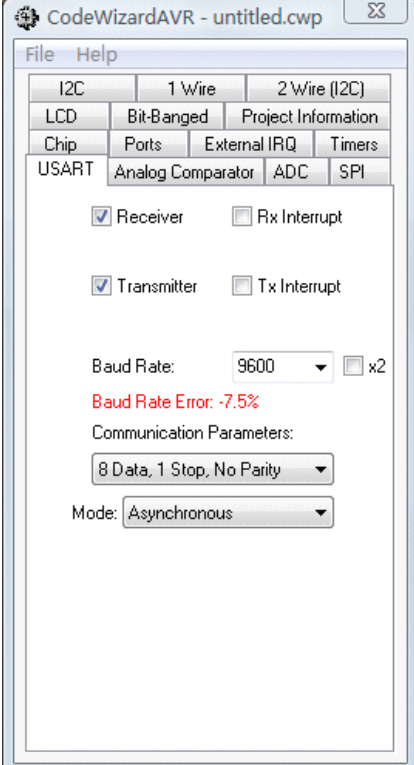
上面两行指包括了头文件, #include <mega8.h>是 CodeWizrdAVR 自动生成的, 而#include<math.h>需要手动加上, 在 math.h 里有所需的 sin 函数。对于还不理解头文件的朋友可以先不用管它, 待有时间时再系统看下 C 语言程序设计。




实例十：串口通信实验

实例功能：接收来自计算机串口的一个数据并采用数码管显示出来

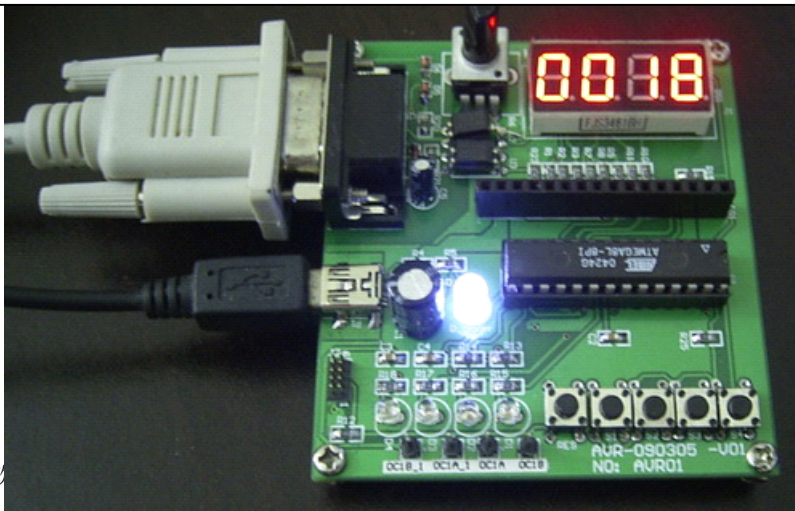
请看下图：串口设置



看左图，串口配置很简单，只要选择使能发送与接收，然后再选波特率就可以了。



```
if(UCSRA.7==1) //接收到新的数据时
    dat_pro(); //按数据处理
void dat_pro(void) //数据处理程序
{unsigned char a;
a=UDR; //读取数据
display_buf[3]=a%10; //将个位存到缓冲
display_buf[2]=(a/10)%10; //将十位存到缓冲
display_buf[1]=(a/100)%10; //将百位存到缓冲
display_buf[0]=a/1000; //将千位存到缓冲
```

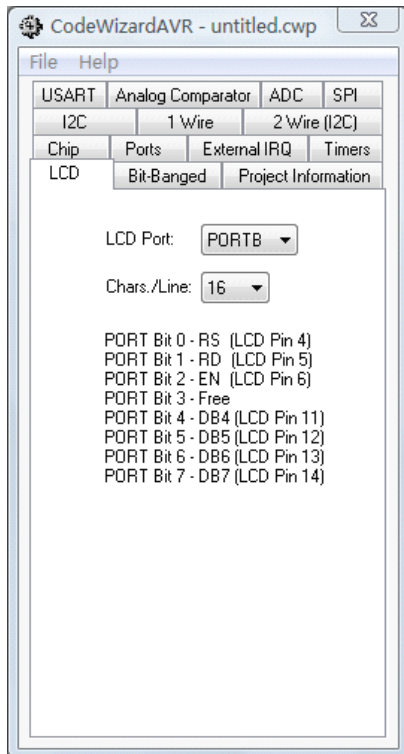


注意：上面发送的是采用 ASCII 码发送，在 ASCII 里数字“2”是 18，数字“1”是 17，数字“0”是 16，所以发送 2 时数码管显示 18。

实例十一：LCD1602 显示实验

实例功能：驱动 LCD1602 显示

很多时候需要用到 LCD1602 作为系统显示，如设计电压表、电流表及电子负载等仪器时，大都需要用到 LCD 显示屏。市场上用的大部份 LCD1602 都采用统一标准驱动方式。以前采用 C51 单片机给 LCD1602 编程时，花了好几天时间才搞好，因为那时不理解 LCD 数据表中的几个控制要求，总是有一些小问题而出错。如采用 CodeWizardAVR，我们可以不用知道 LCD 是如何驱动的，直接调用相关函数就可以了，看了这个实例相信你也会认为原来驱动 LCD1602 如此简单。



左图中“LCD Port”是选择单片机与 LCD 的端口，在原理图上看到 LCD 接于 PB 口上，“Chars/Line”是选择 LCD 的类型，实验用的是 1602，所以选择 16。接下来是单片机与 LCD 连接对应的引脚位。

程序清单：

```
//LCD 显示驱动
void lcd_display(void)
{
//转到 LCD 起始位 第一行第四个字开始
lcd_gotoxy(3,0);
//输入要显示的内容
lcd_putsf("Wellcome");
//转到 LCD 起始位 第二行第二个字开始
lcd_gotoxy(1,1);
//输入要显示的内容
lcd_putsf("WWW.MCURM.COM");
}
```

例子中的两个函数已由 CodeWizardAVR 生成，只需调用就可了，驱动 LCD 就是这么简单。



结束语: