从 MCS51 向 AVR 的快速转换

詹卫前

ATMEL 的 AVR 系列单片机是一个优秀的 RISC 结构单片机系列，与 MCS51 相比其有以下一些典型特点：

①AVR 的机器周期为 1 个时钟周期，绝大多数指令为单周期指令，因此每 MHZ 时钟有接近 1MIPS 的性能。

②程序存贮器与数据存贮器有分开的总线，程序可以高效地执行，8MHZ 频率下工作的 AVR 相当于 224MHZ 频率下工作的 MCS51。

③ 内置可重复编程的 FLASH 程序存贮器和 EEPROM 数据存贮器，支持对单片机的在系统编程（ISP）。在生产中可以“先装配后编程”，从而缩短工艺流程和节约购买万用编程器的费用，并且可以方便地升级或修改程序。

④内置上电复位电路和看门狗定时器（WatchDog）电路，在提高产品可靠性的同时降低了电路的成本。

⑤部分 AVR 单片机与 MCS51 系列单片机管脚兼容，如 AT90S1200/2313 对应 AT89C1051/2051，AT90S4414/8515 对应 AT89C51/52。因此可以做到一套 PCB 板对应两套电路，增加了用户备货的可选择性和灵活性。

⑥定时/计数器的功能大大增强，串口通信时波特率发生不占用定时器。

注：在本文中 AVR 的 C 语言是指 ICCAVR6.0 标准版，如需向其它版本的 AVR C 语言（如 IAR A90、CodeVision AVR）转换，可与双龙电子公司联系。

一、AVR 和 MCS51 存贮器配置的对比

1、存贮器布置

MCS51 的存贮器从使用角度看分三个地址空间，三个空间分别用 MOV、MOVX 和 MOVC 指令访问。

而 AVR 的存贮器在物理结构上可分为五个部分（AT90S8515 为例）：

1）、程序空间（000H~FFFH），访问时用 LPM 指令访问。

2）、片内数据存贮器（0060H~025FH），访问时用 STS、LDS 和 ST、LD 指令访问。

3）、片外数据存贮器（0260H~FFFFH），访问时用 STS、LDS 和 ST、LD 指令访问。

4）、32 个通用寄存器 R0~R31，它们之间数据传送可使用 MOV 指令。

5）、I/O 寄存器（00H~3FH），使用 IN、OUT 指令访问。

看了以上介绍，仔细的读者可能发现有一部分数据存贮器的地址（0000H~005FH）是空闲的。其实这部分地址空间并不空闲，其被映射为通用寄存器（R0~R31）和 I/O 寄存器的数据空间地址，具体为：32 个通用寄存器，直接映射到数据存贮器的 0000H~001FH；64 个 I/O 寄存器，直接映射到数据存贮器空间的 0020H~005FH。这种映射关系大大增强了 AVR 指令的灵活性，一方面对寄存器可以

象 SRAM 一样地访问；另一方面对寄存器的访问时，也可以使用 X、Y 和 Z 寄存器作为索引，从而大大提高了访问寄存器的灵活性。

2、堆栈工作方式

MCS51 的堆栈是一个由堆栈指针寄存器 SP（单字节）控制的向上生长型堆栈，即将数据压入堆栈时 SP 增大。

在 AVR 系列单片机的堆栈同样是受 SP 寄存器控制，而堆栈的生长方向与 MCS51 是不相同的，其向下生长，即将数据压入堆栈时 SP 减小。另外要注意以下几点：

1)、MCS51 的堆栈空间只能放置在片内的 SRAM 中，而 AVR 的堆栈空间既可以放置的片内 SRAM 中，也可以放置在片外 SRAM 中。

2)、AVR 的 SP 寄存器，对不支持外部 SRAM 的单片机为一个字节长度，对支持外部 SRAM 的单片机为两个字节长度（SPL、SPH)。

3)、为了提高速度，一般在初始化 SP 时，将其定位于内部 SRAM 的顶部（如对 8515，为 025FH)。

4)、AT90S1200 不支持软件堆栈（即由 SP 控制堆栈)，其包含了一个三级深度的硬件堆栈。

5)、在对 AVR 编程时一定要对 SP 进行初始化，否则很可能出现“在 AVR Studio 中模拟调试正常，而程序下载到芯片后程序却不工作”的现象。

3、外部 SRAM 的配置

在 MCS51 中外部 SRAM 是使用专用的“MOVX”指令访问的，而在 AVR 中访问片内或片外 SRAM 使用相同的指令，当访问数据空间的地址超过片内 SRAM 范围时，会自动选择片外的 SRAM 空间。但为了正常工作，还必须对寄存器 MCUCR 的 SRE（D7)、SRW（D6）位进行设置。

SRE	SRW	SE	SM	ISC11	ISC10	ISC01	ISC00

MCUCR 寄存器

当 SRE=1 时，使能外部 SRAM，如汇编指令 *SBI　MCUCR，SRE*；

SRE=0 时，禁止外部 SRAM，如汇编指令 *CBI　MCUCR，SRE*。

当 SRW=1 时，在访问外部 SRAM 中插入一个等待周期，如汇编指令 *SBI MCUCR，SRW*；

当 SRW=0 时，在访问外部 SRAM 中不插入一个等待周期，如汇编指令 *SBI MCUCR，SRW*。

在 C 语言中，可以直接用 *MCUCR|=0xC0* 或 *MCUCR&=0x3F* 来配置外部 SRAM。

4、程序空间的访问

MCS51 的程序存贮器是以字节为单位的，地址也是按字节进行寻址的，使用 MOVC 指令访问程序 ROM 和指令寄存器访问程序 ROM 没有什么区别。

在 AVR 中，程序存贮器的总线为 16 位，即指令寄存器访问程序 ROM 时是以字（双字节）为单位的，即一个程序地址对应两个字节；而 AVR 的数据存贮器的总线为 8 位，当用户使用 LPM 指令访问程序 ROM 时是以字节为单位进行读取的，此时 Z 寄存器中的一个地址只对应一个字节。因此要注意这两个地址的换算，否则很容易产

生错误，具体的换算是 LPM 指令使用的 Z 寄存器中的地址应该是程序地址的两倍。

如：
```
ldi   ZH，high(F_TABLE*2)
ldi ZL，low(F_TABLE*2)；初始化 Z 指针
.
lpm
st Y+，R0
.
F_TABLE：
.db  0,1        ;表的起点(20 bytes)
.
.db  18,19
```

二、AVR 输入/输出端口的使用

MCS51 单片机的 I/O 端口大部分是准双向口，在复位时全部输出高电平。对端口的输入和输出操作也是直接通过 I/O 端口的地址进行的。

而 AVR 的 I/O 端口为标准双向口，在复位时所有端口处于没有上拉电阻的输入状态（高阻态，管脚电平完全由外部电路决定），这在强调复位状态的场合是很有用的。AVR 的每一个端口对应三个地址，即 DDRX、PORTX 和 PINX（X 针对不同的单片机可从 A~F 中分别取不同的符号，注意只有 PINX 可取 F）。

DDRXn	PORTXn	I/O	上拉	备注
0	0	输入	关闭	高阻态
0	1	输入	打开	提供弱上拉，低电平必须由外电路位低，此时输出电流
1	0	输出	关闭	推挽输出 0
1	1	输出	关闭	推挽输出 1

表一、端口功能配置表（X=A~E、n=0~7）

DDRX 为端口方向寄存器，当 DDRX 的某一位置 1 时，相应端口的引脚作为输出使用；当 DDRX 的某一位清 0 时，相应端口的引脚作为输入使用。

PORTX 为端口数据寄存器，当引脚作为输出使用时，PORTX 的数据由相应引脚输出；当引脚作为输入使用时，POTRX 的数据决定相应端口的引脚是否打开弱上拉，具体可参考表一。

PINX 为相应端口的输入引脚地址。如果希望读取相应引脚的逻辑电平值，一定要读取 PINX，而不能读取 PORTX，这与 MCS51 是有区别的。

注意：在使用 AVR 单片机之前，一定要根据引脚功能定义，对相应的端口初始化，否则端口很可能在用作输出时不能正常工作。如设置端口 B 的高四位为输出，低四位为输入：

汇编语言：*ldi R16，$F0*

out DDRB，R16

在 C 语言中：*DDRB=0xF0；*

三、AVR 和 MCS51 定时器的对比

1、**功能比较**

在 MCS51 **中，定时/计数器有两种基本用法，即以晶振频率的十二分频信号为输入的定时器工作方式，和以外部引脚** INT0、INT1 **上的信号为输入的计数器工作方式。**

在 AVR **中有两个定时器** T0 **和** T1（AT90S1200 **只有一个** T0），T0 **的功能与** MCS51 **相似；而** T1 **的功能很强，除了普通的定时/计数功能外,还有一些增强的功能，如：比较匹配** A、**比较匹配** B、**由** ICP **引脚或模拟比较器触发的捕捉功能、**8~10 **位的** PWM **调制器。**

AVR **的定时/计数器用作定时器时，其输入信号为晶振频率的某一个分频信号，分频比为** 1、8、64、256、1024 **五种；作为计数器使用时，既可上升沿触发，也可下降沿触发。**

2、T0 **的使用**

在 AVR **中** T0 **为八位长度，其由** TCCR0 **寄存器控制。** TCCR0 **组成如下图所示：**

X	X	X	X	X	CS02	CS01	CS00

TCCR0 **的作用如下表所示：**

CSX2	CSX1	CSX0	说明	CSX2	CSX1	CSX0	说明
0	0	0	T0 **停止工作**	1	0	0	CK/256、**定时器**
0	0	1	CK、**定时器**	1	0	1	CK/1024、**定时器**
0	1	0	CK/8、**定时器**	1	1	0	**计数器(下降沿触发)**
0	1	1	CK/64、**定时器**	1	1	1	**计数器(上升沿触发)**

表二、 TCCR0 **功能表（**X=0、1）

例 1：**定时器** T0 **用作定时器，晶振频率** 4M，**定时时间** 10ms，**可以这样对** T0 **初始化：**

汇编语言：*LDI R16，$D9*

*OUT TCNT0，R16；**定时常数到定时寄存器** TCNT0*

LDI R16，$05

*OUT TCRR0，R16 ；1024 **分频比***

C **语言：***TCNT0=0xD9；*

TCRR0=0x05；

说明：TCNT0 **为定时计数寄存器。**

例 2：**定时器** T0 **用作计数器，下降沿触发，可以这样对** T0 **初始化：**

汇编语言：LDI R16，00

OUT TCNT0，R16；TCNT0 **清零**

LDI R16，$06

OUT TCRR0，R16；计数器方式工作下降沿触发

C 语言：TCNT0=0；

TCRR0=0x06；

3、T1 的使用

在 AVR 中，T1 是 16 位的，其控制寄存器有 TCCR1A 和 TCCR1B 两个，TCCR1A 组成如下图所示：

COM1A1	COM1A0	COM1B1	COM1B0	X	X	PWM 11	PWM 10

各位的功能如表三所示：

COM1X1	COM1X0	说明	PWM11	PWM10	说明
0	0	与输出 OC1X 不连接	0	0	禁止 PWM 操作
0	1	OC1X 电平翻转	0	1	8 位 PWM
1	0	OC1X 为低电平	1	0	9 位 PWM
1	1	OC1X 为高电平	1	1	10 位 PWM

表三、TCCR1A 功能表

TCCR1B 的组成如下图所示，其中 CS10~CS12 的用法同 T0，见表二所示：

ICNC1	ICES1	X	X	CTC1	CS12	CS11	CS10

ICNC1：置 1 时使能输入捕捉噪声消除，清 0 时禁止输入捕捉噪声消除。

ICES1：置 1 时在 ICP 的上升沿时发生定时器捕捉，清 0 时在 ICP 的下降沿发生定时器捕捉。

CTC1：置 1 时，当比较匹配 A 发生时将 TCNT1 清 0；清 0 时 TCNT1 继续计数，直至它被停止、清除、溢出为止。注意：只有比较匹配 A 有效，另外在 PWM 方式下该位无效。

例 1：定时器 1 用比较匹配 A 方式，在 OC1A 引脚输出一个 50HZ 的方波，晶振频率为 4MHZ，可以这样对 T1 初始化：

汇编语言：

```
LDI   R16，$20
OUT   DDRD，R16；引脚 OC1A 输出
LDI   R16，$00
OUT   TCNT1H，R16
OUT   TCNT1L，R16；TCNT1 清 0
LDI   R16，$02
OUT   OCR1AH，R16
LDI   R16，$71
OUT   OCR1AL，R16；送入对应 10ms 的比较常数
LDI   R16，$40
OUT   TCCR1A，R16；T1 和 OC1A 相连，比较匹配时 OC1A 翻转
```

```
LDI     R16，$0B
OUT     TCCR1B，R16；T1 以定时器方式工作，分频比为 64
                    ；比较匹配后自动清除 TCNT1
```

C 语言：

```
DDRD=0x20；//引脚 OC1A 输出
TCNT1=0x00；  //TCNT1 清 0
OCR1A=0x271；//送入对应 10ms 的比较常数
TCCR1A=0x40；//T1 和 OC1A 相连，比较匹配时 OC1A 翻转
TCCR1B=0x0B；//T1 以定时器方式工作，分频比为 64，比较匹配后
             //  自动清除 TCNT1
```

注意：

1)、由于 T1 的 TCNT1、OCR1A、OCR1B 和 ICR1 均为 16 位的定时器，为了正确地写入和读出，在写入数据时应先写入高位字节，后写入低位字节；在读取数据时，应先读取低位字节，后读取高位字节。

2)、T1 的捕捉方式，可用于 ICP 引脚上频率或周期的测量，在使用时只需使能捕捉中断即可，对 T1 的设置可参考定时的用法。

四、AVR 和 MCS51 中断系统的对比

MCS51 有六个中断源（5 个中断入口地址），分两个优先级，并且是通过 IE 寄存器控制中断的使能，通过 IP 控制中断的优先等级。

而在 AVR 中，根据不同的单片机有不同数量的中断源，典型的 AT90S8515 有 12 个中断源，这 12 个中断源各自有自己的中断向量入口地址，如表四所示。AVR 通过寄存器 GIMSK 和 TIMSK 及 SREG 来控制中断使能，其中 SREG 的 D7 位 I 是全局中断使能标志。在 AVR 中只有全局中断控制位和某一特定中断控制位同时使能，中断才会起作用。GIMSK 和 TIMSK 的功能见下图：

INT1	INT0	X	X	X	X	X	X

GIMSK

TOIE1	OCIE1A	OCIE1B	X	TICIE1	X	TOIE0	X

TIMSK

INT0、INT1：外部中断请求 0、1 使能，置 1 时开放中断，清 0 时禁止中断。外部中断的触发方式由 MCUCR 的 D0~D3 位控制，如表五所示。

TOIE0、TOIE1：定时器 0、1 的溢出中断使能。

OCIE1A、B：定时器 1 的比较匹配 A、B 中断使能。

TICIE1：定时器 1 的输入捕捉中断使能。

向量号	入口地址	中断源	说明
1	$000	RESET	硬件复位和看门狗复位
2	$001	INT0	外部中断请求 0
3	$002	INT1	外部中断请求 1
4	$003	TIMER1 CAPT	定时器 1 捕捉中断
5	$004	TIMER1 COMPA	定时器 1 比较匹配 A 中断

6	$005	TIMER1 COMPB	定时器 1 比较匹配 B 中断
7	$006	TIMER1 OVF	定时器 1 溢出中断
8	$007	TIMER0 OVF	定时器 0 溢出中断
9	$008	SPI、STC	SPI 传送完成
10	$009	UART，RX	串行通信接收完成
11	$00A	UART，UDRE	串行通信数据寄存器空
12	$00B	UART，TX	串行通信发送完成
13	$00C	ANA_COMP	模拟比较器中断

表四、中断向量名称与入口地址

ISCX1	ISCX0	说明	ISCX1	ISCX0	说明
0	0	低电平触发中断	1	0	下降沿触发中断
0	1	高电平触发中断	1	1	上升沿触发中断

表五、外部中断触发方式（X=0、1）

在 AVR 中没有专门的中断优先级控制寄存器来区分中断的优先等级，用户可在中断服务程序中通过使能全局中断 I 来使系统响应高优先级的中断。具体的做法：是当 AVR 单片机响应任何一个中断时，硬件会禁止全局中断 I，从而禁止系统响应其它中断，而当从中断服务程序中退出时硬件重新使能全局中断 I；而当我们在中断服务程序中用 SEI 指令打开全局中断使能时，系统在没有退出中断服务程序的情况下又恢复了对中断的响应能力，从而可以响应高优先级的中断。另外，在同一优级中入口地址较低的中断优先级较高。

例：系统使能定时器 1 溢出中断和外部 INT0 中断，其中 INT0 的优先级较高，此时可以这样对 MCU 初始化：

汇编语言：

```
LDI    R16，$40
OUT    GIMSK，R16；使能 INT0 中断
LDI    R16，$80
OUT    TIMSK，R16；使能 T1 溢出中断
SEI                ；使能全局中断
.
.
timer1_ovf：       ；T1 溢出中断服务程序
        SEI        ；在 T1 溢出中断服务程序中开放全局中断
                   ；保证 INT0 的优先级

        RETI
```

注意：在 AVR 的子程序中硬件不保护 SREG 状态寄存器，应根据实际情况由软件进行保护。

C 语言：*#pragma interrupt_handler timer1：7 //声明 timer1( )为中断处*

```
                                                    //理函数
 #pragma  interrupt_handler  int0：2   //声明_int0( )为中断处理
                                                    //函数
void  main  (void)
 {
   GIMSK=0x40；//使能 INT0 中断
   TIMSK=0x80；//使能 T1 溢出中断
  _SEI( )；//使能全局中断
   .
   }
 void  timer1(void)//T1 溢出中断服务程序
 {
  _SEI( )；  //在 T1 溢出中断服务程序中开放全局中断
                 //保证 INT0 的优先级
   .
   }
```

在 C 语言的中断服务程序（中断处理函数）中，会自动保护中断服务程序使用过的所有寄存器。

五、AVR 和 MCS51 位操作功能的对比

MCS51 和 AVR 都有较强的位操作功能，在汇编语言写的 AVR 源程序中对端口的某一位置 1 可用 SBI 指令，清 0 可用 CBI 指令。

在 C 语言程序中，可用位运算或在线汇编完成上述功能，如置 PORTB 的 D2 位为 1，清 PORTB 的 D6 位为 0：

```
PORTB|=(1<<2)；//D2 位置 1
PORTB&=~(1<<6)；//D6 位清 0
```

或

```
ASM(“SBI   0x18，2”  )；//D2 位置 1
ASM(“CBI   0x18，6”)；//D6 位清 0
```

六、AVR 单片机内置 EEPROM 的使用

AVR 是通过三个寄存器来访问 MCU 内置的 EEPROM 的，一个寄存器是 EEAR，存放访问 EEPROM 的地址，其根据片内 EEPROM 的多少可能有不同的长度；另一个是 8 位的 EEDR，用于存放访问 EEPROM 的数据；第三个是 EECR，用于控制对 EEPROM 的读写，EECR 的结构如下图所示：

X	X	X	X	X	EEMWE	EEWE	EERE

EEMWE：EEPROM 主写使能。只有在其置 1 后的 4 个时钟周期内将 EEWE 置 1，才能完成 EEPROM 写入，否则写操作无效。EEMWE 被置 1 后，在 4 个周期后由硬

件自动清除。

EEWE：EEPROM 写入使能

EERE：EEPROM 读取使能

例：写数据到片内 EEPROM 中子程序

汇编语言：

```
.def    EEdwr   =r16        ；写入 EEPROM 的数据
.def    EEawr   =r17        ；EEPROM 的地址低位
.def    EEawrh  =r18        ；EEPROM 的地址高位
EEWrite：sbic   EECR，EEWE
         rjmp EEWrite       ；等待 EEPROM 就绪
         out    EEARH，Eeawrh
         out    EEARL，EEawr      ；送入 EEPROM 地址
         out    EEDR，EEdwr ；送入写入 EEPROM 的地址
         sbi    EECR,EEMWE ；设置 EEPROM 主写使能
         sbi    EECR,EEWE   ；设置 EEPROM 写使能
         ret
```

C 语言：（注意应包含头文件 eeprom.h）

```
int EEPROMwrite( int location, unsigned char);
int location：片内 EEPROM 的地址
unsigned char：写入 EEPROM 的数据
```

七、AVR 单片机内置看门狗电路（WatchDog）的使用

AVR 系列单片机内置看门狗电路，其由寄存器 WDTCR 控制。WDTCR 的结构如下图所示：

X	X	X	WDTOE	WDE	WDP2	WDP1	WDP0

WDTOE：看门狗关闭使能

只有在该位被置 1 后的 4 个时钟周期内将 WDE 清 0 才能关闭看门狗电路，否则看门狗电路不会被关闭。WDTOE 在被置 1 后，在 4 个周期后由硬件自动清 0。

WDE：置 1 时，使能看狗电路，清 0 时关闭看门狗电路。注意关闭看门狗电路应在对 WDTOE 置 1 后 4 个时钟周期内进行。

WDP0~WDP2：看门狗电路的分频系数（产生复位所需要的振荡周期数），其影响看门狗电路复位的时间，如表六所示：

WDP 2	WDP 1	WDP 0	分频系数	DC3V 时 产生复位所需时间	DC5V（约 1MHZ） 产生复位所需时间
0	0	0	16K	47ms	15ms
0	0	1	32K	94ms	30ms
0	1	0	64K	0.19s	60ms

0	1	1	128K	0.38s	0.12s
1	0	0	256K	0.75s	0.24s
1	0	1	512K	1.5s	0.49s
1	1	0	1024K	3.0s	0.97s
1	1	1	2048K	6.0s	1.9s

注意：看门狗电路的振荡器为内部 RC 振荡器，其振荡频率受电压影响，在 DC5V 时，约为 1MHZ。

在 AVR 中有一条指令 WDR 来清除看门狗定时器，在 C 语言中对应为_WDR( ) 或 WDR( )函数。

八、AVR 和 MCS51 中串口通信 UART 功能的对比

在 MCS51 中串口通信的波特率发生需要使用一个定时器，而且支持的波特率也较低。AVR 单片机可以有较高的波特率，最高波特率可达 115200，而且有专用的波特率发生器。注意 AT90S1200 没有 UART，只能用软件模拟串口通信。

在 AVR 中用于 UART 的寄存器主要有以下几个：接收和发送数据寄存器 UDR、状态寄存器 USR、控制寄存器 UCR 和波特率寄存器 UBRR。

UDR 寄存器由两个物理上分开的寄存器共享同一个地址，写入数据时是写到发送寄存器，读出数据时是读取接收寄存器。

USR 如下图所示，其反映了 UART 的状态。

RXC	TXC	UDR	FE	OR	X	X	X

RXC：UART 接收完成

TXC：UART 发送完成

UDR：UART 数据寄存器空标志

FE：帧出错

OR：超越出错

UCR 控制 UART 的工作，其组成如下图所示：

RXCIE	TXCIE	UDRIE	RXEN	TXEN	CHR9	RXB8	TXB8

RXCIE：接收中断使能

TXCIE：发送中断使能

UDRIE：UART 数据寄存器空中断使能

RXEN：接收使能

TXEN：发送使能

CHR9：发送 9 位字符

RXB8：接收到的第 8 位字符

TXB8：发送的第 8 位字符

UBRR 控制 UART 的波特率，其与波特率的计算公式为：

BAUD=FCK/[16(UBRR+1)]

其中：BAUD 表示波特率、FCK 表示晶振频率、UBRR 表示 UBRR 寄存器中的常数。

注意：波特率的计算值与标准波特率相差不能超过 2%，否则会影响串行通信。

例 1：在 8MHZ 晶振下以 19200 波特率和 PC 机通信，可这样对 AVR 初始化：

汇编语言：
```
LDI   R16，25
OUT   UBRR，R16
LDI   R16，$18
OUT   UCR，R16
```
C 语言：
```
UBRR=25；
UCR=0x18；
```
例 2：在 8MHZ 晶振下以 19200 波特率与 PC 机通信，每接收到一个非 0 字节，发送一个“OK！”

```
#include <io8515.h>
#include <stdio.h>
void main(void)
    {
     unsigned char temp;
     UBRR = 25;
     UCR=0x18;
     puts("Hello World!");
     putchar(0x0d);
     putchar(0x0a);
     while (1)
          {
            temp=getchar();
            if (temp!=0)
               {
                puts("OK!");
               putchar(0x0d);
                putchar(0x0a);
               temp=0;
               }
          }
        }
```

九、C51 的源代码向 ICCAVR 的快速转换

熟悉 C51 的读者看了以上内容，完全可以很快写出 AVR 的 C 源程序来，下面再

将 C51 向 ICCAVR 的转换进行一次总结。

1、头文件

对 C51 中定义寄存器的头文件如 reg51.h、at89x51.h等替换成相应的 AVR 头文件，如 io8515.h、 io2313.h 等

2、中断处理函数

在 C51 中以 interrupt 关键字来说明某一个函数为中断处理函数，在 ICCVAR 中可采用#pragma interrupt_handler 预处理命令在程序开始处声明，具体用法如下：

*#pragma interrupt_handler <**中断处理函数名**>：<**中断向量号**>*

注意：对原 C51 源程序中的 interrupt 和 using 关键字应当删除。

3、对 C51 中的 bit 和 sbit 数据类型的处理

在 ICCAVR 中不支持 bit 和 sbit 数据类型，对这两种类型可用 unsigned char 来代替。

对有关位运算用标准 C 的位运算功能进行处理，也可采取在线汇编处理。

4、对中断系统、定时器初始化

需重新根据相应控制寄存器的功能给其赋值，方法与 C51 相同。具体如下：

对 MCS51 中 TMOD、TCON 的处理改为对 AVR 的 TCCR0、TCCR1A、TCCR1B、TIFR 的处理。

对 MCS51 中 IE、IP 的处理改为对 AVR 中 GIMSK、TIMSK、MCUCR、SREG 的处理。

5、将原 C51 中有关对看门狗电路、外部 EEPROM 的处理改为对 AVR 芯片内部看门狗电路、内部 EEPROM 的处理。

6、对 MCS51UART 的初始化改为对 UCR 和 UBRR 和被始化。

7、如果使用片外 SRAM，应当对 MCUCR 初始化；如果有引脚作为输出引脚使用，应当对其方向寄存器进行初始化。

8、对 C51 中符合 ANSI 标准的 C 语言，原则上不需要进行修改，除非为了程序结构的优化。