

MC68HC908 应用手记

(2006 年初整理稿)

作者：程序匠人

出处：《匠人的百宝箱》

目 录

1.	前言	2
2.	C 语言中嵌入汇编的 7 种方式	2
3.	C 语言中数的表示方式	3
4.	循环体的 3 种写法	3
5.	关于复位及中断的入口地址	4
6.	对被调函数的说明(声明)	4
7.	对中文的支持	4
8.	中断定义有两种方法	5
9.	相关网址	5
10.	数据类型	5
11.	位的定义与使用	6
12.	数据结构	8
13.	工程文件系统介绍	9
14.	I/O 口使用注意事项	9
15.	关于强制类型转换	9
16.	中断的使用方法	10
17.	定时器中断频率的计算	10
18.	如何产生 LST 文件	11
19.	工程文件的组织方法	12
20.	关于 MON08	13
21.	mon08 的仿真模式的断点	14
22.	关于 MON08 调试的频率	14
23.	关于运算中需要注意的问题	14
24.	PLL 功能的启动方式	15
25.	参考文章	15

1. 前言

匠人最近开始和 MC68HC908 的亲密接触。(忙着和它约会,连上网次数也少了许多,嘿嘿....)

作为一个长期用汇编的工程师“转行”用 C 写程序,感觉挺累人,不过苦中也有乐.愿将原汁原味的手记与大家分享.

这篇手记将随着匠人的经验值不断升级而不断更新,其中会有心得,也会有困惑,权当是一个过程的见证吧.

首先得声明,匠人是第一次用摩托罗拉的芯片,也是第一次用 ANSI_C 语言,所以手记中如有错误和问题,望大伙指点.还有一点需要声明的是,本文中有些段落是从其它文档中摘抄或整理而来,匠人无抄袭之意.

本文的正式发布版本为 PDF 格式,欢迎转载.匠人唯一的要求是,转载者不可对文件中的任何内容(包括作者和出处信息)进行修改.转载者有义务保证此文档的完整性.

2. C 语言中嵌入汇编的 7 种方式

//嵌入汇编方式 1(宏):

```
EnableInterrupts;           // 开中断
```

//嵌入汇编方式 2(可嵌入多条指令):

```
asm {  
    lda _PTB ;  
}
```

//嵌入汇编方式 3(单条指令):

```
asm eor #0b00000100 ;  
asm nop ;
```

//嵌入汇编方式 4(单条指令):

```
_asm nop;
```

//嵌入汇编方式 5(单条指令):

```
asm "nop";
```

//嵌入汇编方式 6(单/多条指令):

```
asm (" eor #4 ");  
asm ("nop; nop");  
asm("nop\n nop");
```

//嵌入汇编方式 7(可嵌入多条指令):

```
#asm
```

```
nop  
nop  
#endasm
```

3. C 语言中数的表示方式

****举例如下：

```
二进制： 0b00000100  
十进制： 4  
十六进制： 0x4
```

****注意：汇编指令的表示方式与 C 的表示方式不一样，但如果是在 C 中嵌入汇编，则也要按 C 的方式来写

****问题：八进制的引导符号还不知道

4. 循环体的 3 种写法

写法 1：

```
while(1);  
{  
.....  
}
```

写法 2(推荐)：

```
for(;;);  
{  
.....  
}
```

写法 3：

```
loop:  
.....  
goto loop
```

****网友 gtw 答：CodeWarrior 对恒为“真”的表达式编译时经常会有提示，很罗嗦，如 while(1)；
不如用 for(;;)

5. 关于复位及中断的入口地址

所有的入口地址都存储在\$FFD0~\$FFFF 区域中

比如:复位地址存储在\$FFFE~\$FFFF 中(缺省值=DC8C, 即复位后从 DC8C 处开始执行)

****问题:如何修改入口地址, 还不清楚

****网友 gtw 答:定义入口地址, 有多种方法 interrupt void (VECTOR_NUMBER) FUNC_NAME(void)即为其一, 可以自动设置。

6. 对被调函数的说明(声明)

1. 如果被调用函数出现在主调用函数之后, 则在调用之前应该先对被调用函数的返回值类型作出说明

一般形式, 如: void delay(void);

2. 如果被调用函数出现在主调用函数之前, 则不需要作出说明

3. 如果被调用函数和主调用函数不在一个文件中, 则说明方式如下:

```
extern void delay(void); //注:如果不作说明, 系统会警告, 但也能进入 DEBUG 状态
```

7. 对中文的支持

缺省的系统环境对中文的支持不好, 表现为按删除键时回将中文删除一半从而显示乱码.

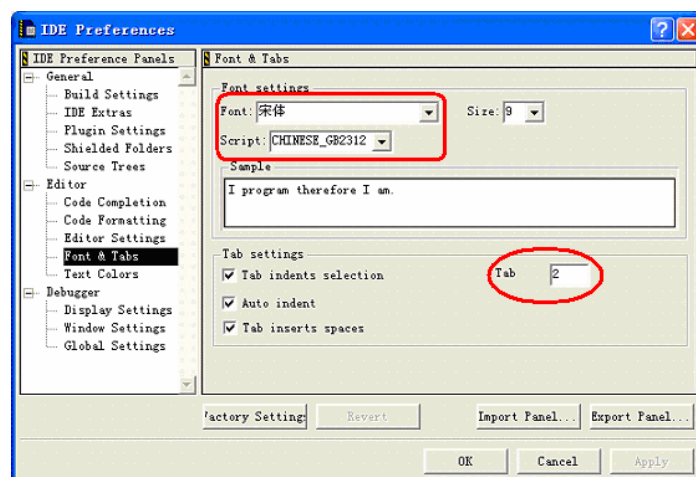
解决方法是选择[Edit]菜单下的[Preferences...], 并按如下设置:

将 EDITOR 中的 font&tabs 中改动:

font 选择为宋体

script 选择为 chinese_gb2312

****注:也可以在这里设置 TAB 所代表的空格数(缺省是 2)



8. 中断定义有两种方法

方法一：

```
#pragma TRAP_PROC
void IntFunc1(void)
{
/* your code */
}
```

In your prm file:

```
VECTOR ADDRESS 0xFFFF4 IntFunc1 /* 0xFFFF4 contains the address of IntFunc1 */
```

方法二：

```
interrupt 3 IntFunc1()
{
.../*code*/
}
```

Means that the third entry in the vector table is initialized with the address of IntFunc1().

9. 相关网址

ehua 发表于 2005-7-3 09:27 侃单片机 ←返回版面

到这里看看吧：<http://www.edatech.com/bbs/index.asp>

下面是关于这个软件的一些常见问题：

<http://www.edatech.com/bbs/dispbbs.asp?BoardID=3&replyID=284&id=284&star=1&skin=0>

10. 数据类型

类型		缺省格式	缺省范围	可选格式
char(unsigned)	8bit		0~255	8bit,16bit,32bit
signed char	8bit		-128~127	8bit,16bit,32bit
unsigned char	8bit		0~255	8bit,16bit,32bit
signed short	16bit		-32768~32767	8bit,16bit,32bit
unsigned short	16bit		0~65535	8bit,16bit,32bit
enum(signed)	16bit		-32768~32767	8bit,16bit,32bit
signed int		16bit	-32768~32767	8bit,16bit,32bit
unsigned int	16bit		0~65535	8bit,16bit,32bit
signed long	32bit		-2147483648~2147483647	8bit,16bit,32bit



更多精彩文章，尽在《匠人的百宝箱》，网址：<http://cxjr.21ic.org>

unsigned long	32bit		0~4294967295	8bit,16bit,32bit
signed long	32bit	-2147483648~2147483647	8bit,16bit,32bit	
unsigned long	32bit	0~4294967295		8bit,16bit,32bit

定义寄存器

C51 : volatile unsigned char Bank1R0 _at_ 0x0008;

CW08 : volatile unsigned char PTA @0x0000;

11. 位的定义与使用

位必须通过结构体来定义

定义方法：（以 PTB 口的操作为例子）

```
/** PTB - Port B Data Register; 0x00000001 */
typedef union {
    byte Byte;
    struct {
        byte PTB0      :1;          /* Port B Data Bit
0 */
        byte PTB1      :1;          /* Port B Data Bit
1 */
        byte PTB2      :1;          /* Port B Data Bit
2 */
        byte PTB3      :1;          /* Port B Data Bit
3 */
        byte PTB4      :1;          /* Port B Data Bit
4 */
        byte PTB5      :1;          /* Port B Data Bit
5 */
        byte PTB6      :1;          /* Port B Data Bit
6 */
        byte PTB7      :1;          /* Port B Data Bit
7 */
    } Bits;
    struct {
        byte grpPTB   :8;
    } MergedBits;
} PTBSTR;
extern volatile PTBSTR _PTB @0x00000001;
```

```
#define PTB _PTB.Byte
#define PTB_PTBO _PTB.Bits.PTB0
#define PTB_PTBI _PTB.Bits.PTB1
#define PTB_PTBI2 _PTB.Bits.PTB2
#define PTB_PTBI3 _PTB.Bits.PTB3
#define PTB_PTBI4 _PTB.Bits.PTB4
#define PTB_PTBI5 _PTB.Bits.PTB5
#define PTB_PTBI6 _PTB.Bits.PTB6
#define PTB_PTBI7 _PTB.Bits.PTB7
#define PTB_PTBI _PTB.MergedBits.grpPTB
```

```
#define PTB_PTBI0_MASK 1
#define PTB_PTBI0_BITNUM 0
#define PTB_PTBI1_MASK 2
#define PTB_PTBI1_BITNUM 1
#define PTB_PTBI2_MASK 4
#define PTB_PTBI2_BITNUM 2
#define PTB_PTBI3_MASK 8
#define PTB_PTBI3_BITNUM 3
#define PTB_PTBI4_MASK 16
#define PTB_PTBI4_BITNUM 4
#define PTB_PTBI5_MASK 32
#define PTB_PTBI5_BITNUM 5
#define PTB_PTBI6_MASK 64
#define PTB_PTBI6_BITNUM 6
#define PTB_PTBI7_MASK 128
#define PTB_PTBI7_BITNUM 7
#define PTB_PTBI_MASK 255
#define PTB_PTBI_BITNUM 0
```

使用方法:(以 PTB2 口的操作为例子)

```
PTB_PTBI2 = 0;
PTB_PTBI2 = 1;
PTB_PTBI2 = ~ PTB_PTBI2 ; //位取反
```

位定义

C51:例子

```
sbit RD = P3^7;
sbit WR = P3^6;
sbit T1 = P3^5;
sbit T0 = P3^4;
sbit INT1 = P3^3;
sbit INTO = P3^2;
sbit TXD = P3^1;
```

```
sbit RXD = P3^0;
```

CW08:例子

```
typedef union {
    struct {
        unsigned char D0:1;
        unsigned char D1:1;
        unsigned char D2:1;
        unsigned char D3:1;
        unsigned char D4:1;
        unsigned char D5:1;
        unsigned char D6:1;
        unsigned char D7:1;
    }Bit;
    unsigned char _BYTE;
}BitType;//数据结构定义
#pragma DATA_SEG SHORT _DATA_ZEROPAGE
BitType PORTC;
#define PTC PORTC._BYTE //8位
#define RD PORTC.Bit.D0
#define WR PORTC.Bit.D1
#define T1 PORTC.Bit.D2
#define T0 PORTC.Bit.D3
#define INT1 PORTC.Bit.D4
#define INT0 PORTC.Bit.D5
#define TXD PORTC.Bit.D6
#define RXD PORTC.Bit.D7
```

12. 数据结构

零页(0x00-0xff)内均可位寻址和直接读写操作,直接寻址.

但零页数据定义要有:

#pragma DATA_SEG SHORT 加上你的 prn 文件当中零页的数据位置

如_DATA_ZEROPAGE 之类(SHORT 表示零页)

零页的数据可以实现 MOVE data1, data2 之类的短指令

ROM 的数据(例如字符表,码表)要有

#pragma DATA_SEG FAR DEFAULT_ROM //(FAR 表示放在 ROM 当中)

DEFAULT_RAM 的空间是放堆栈和其他普通全局变量,堆栈是由 ram 的底部开始(由大变小);变量是由顶部开始(由小变大);所以如果有太多的堆栈数据会覆盖普通全局变量.

13. 工程文件系统介绍

首先看 codewarrior 自动生成的工程文件系统：

1、sources 文件夹里包含一个 main.c 的主文件。

用户的主程序将要放在这个文件中。另外用户自己编写的一些文件（存放用户自己定义的函数）也要放在这个文件夹中

2、startup Code 文件里包括一个 Start08.c 的文件。

该文件中包含一个 _Startup() 的函数，MCU 复位后，将会首先执行 _Startup()。该函数初始化堆栈，拷贝初始数据到 RAM 中，并调用 main() 主函数。一般用户不必修改该文件

3、prm 文件夹 后缀为 prm 的文件中可以根据硬件决定 ROM, RAM 的分配。用户可以自己修改；后缀为 map 的文件中可以看到在文件，函数，变量在存储区中的分配

4、Libs 文件夹

头文件中定义了 I/O 控制、状态、数据寄存器的地址。在用户自己的程序中可以直接使用这些宏定义

14. I/O 口使用注意事项

1. I/O 口的方向寄存器中内容与方向的关系

0 代表输入

1 代表输出

需要注意的是，这一点与其它一些芯片（如 PIC, EMC, HT）正好相反

2. I/O 口在使用之前必须先初始化，否则会报错

15. 关于强制类型转换

符号：()

使用方法：(类型)(表达式)

使用强制类型转换运算符可以将一个表达式转换成所需的类型；

在强制转换时，得到一个所需类型的中间变量，原来变量的类型不变；

在给指针变量赋值时特别有用，比如当指针变量被定义为 char 型，而需要将一个 int 型的变量地址赋值给该指针时，如果不作转换，则系统会报警告。这时做个类型转化则可避免该问题。

如：

```
wrpage(0x0000, (char *)&ccc, 2) ;
```

//说明：由于 ccc 的数据类型为 int 型，所以要将其转化为 char 型

又如：

```
tx((char)(ADDR_24%256));
```

//送数据地址并检测应答信号 //说明:ADDR_24 是 int 型数据,要转化为 char 型

16. 中断的使用方法

关于中断，有很多种方法，在 AN2616 中提到：

把 #pragma TRAP_PROC 放在中断程序前面，并把中断向量表放到 linker.prm。例如：

```
#pragma TRAP_PROC  
void intSW1(void) {  
}
```

或者使用关键词 interrupt，并把向量表加入 linker.prm。例如：

```
interrupt void intSW1(void) {  
}
```

把向量表的首地址放入 linker.prm。例如：

```
VECTOR ADDRESS 0xFFD2 intSW1
```

在定义中断程序的时候使用关键词“interrupt”并把特定的中断向量号。这种方法不需要在 linker.prm 中更改任何东西。例如：

```
interrupt 22 void intSW1(void) {  
}
```

这种方法有很明显的优点，它把所有的东西都写在一个文件中，而不需要依赖于另一个文件。它的缺点是不符合标准的 C 的写法，当需要移植到其它平台时，有可能需要改写代码。

17. 定时器中断频率的计算

公式： 中断频率=总线频率/(分频因子*计数器预置值)

注： 总线频率=晶振频率/4

分频因子可设置为 1, 2, 4, 8, 16, 32, 64

例如：晶振频率=8MHZ，总线频率=2MHZ，分频因子=32，预置值=625

中断频率=2000000/(32*625)=100HZ

又如：晶振频率=9.8304MHZ，总线频率=2.4567MHZ，分频因子=1，预置值=65536 (0X10000)

中断频率=2457600/(1*65536)=37.5HZ

再如：晶振频率=9.8304MHZ，总线频率=2.4567MHZ，分频因子=64，预置值=19200 (0X4B00)

中断频率=2457600/(64*19200)=2HZ

初始化程序如下：

```
//定时器 1 初始化
```

```
//晶振频率=9.8304MHZ，总线频率=2.4567MHZ
```

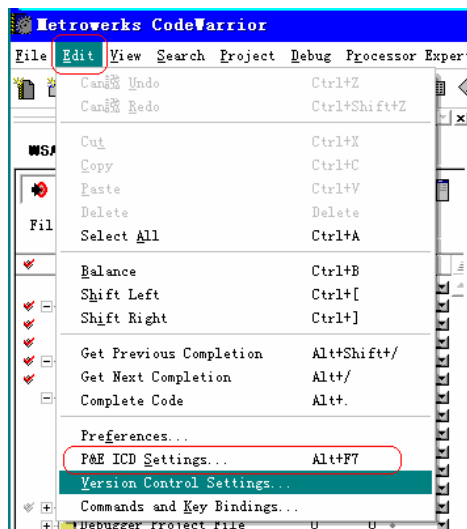
```
//中断频率=总线频率/(分频因子*预置值)=2457600/(64*19200)=2HZ
```

```
T1SC=0b01000110 ; //开中断,分频因子=64
T1MODH=0x4B; //预置值=0x4B00
T1MODL=0x00;
```

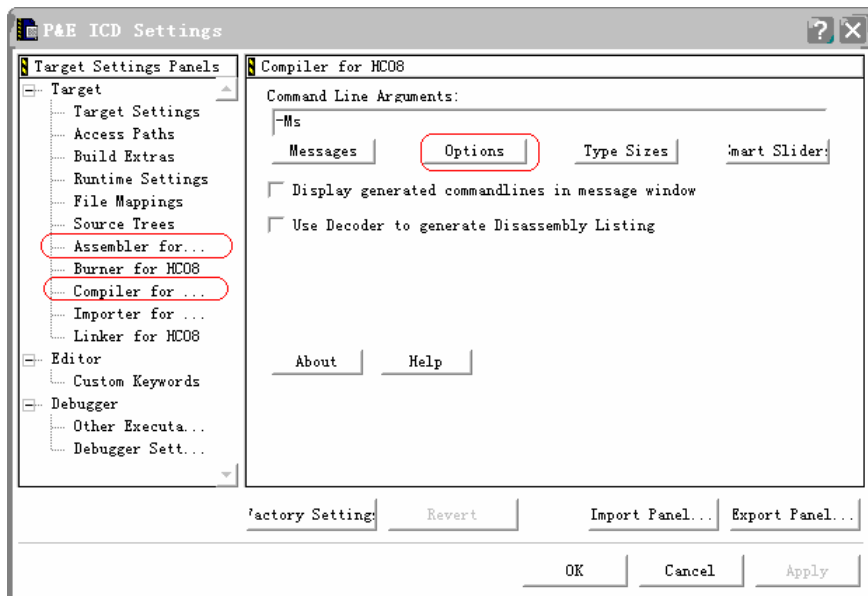
18. 如何产生 LST 文件

系统缺省时不会产生 LST 文件. 如果需要查看 LST 文件, 可以:

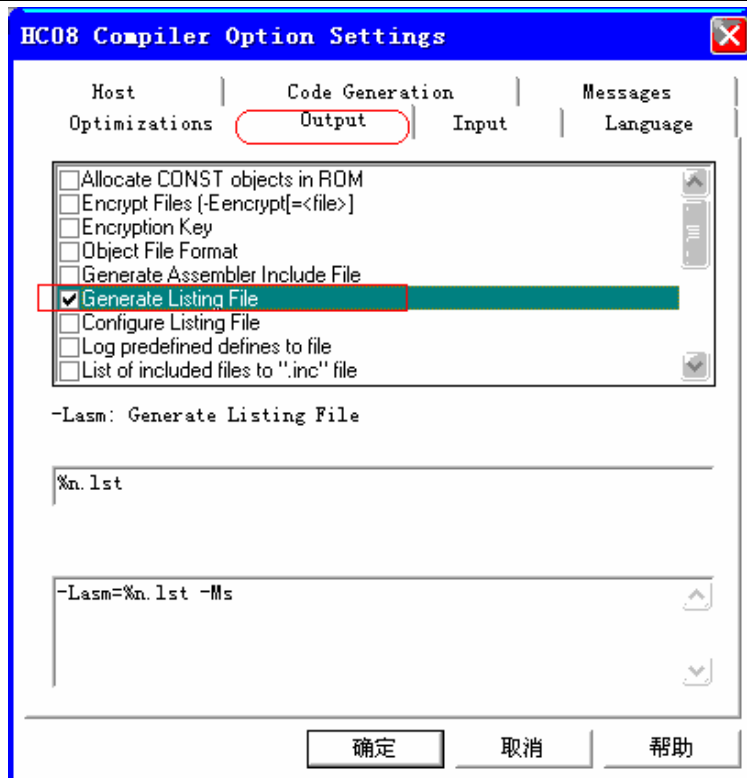
1. 选择菜单[edit]->[P&E ICD SETTINGS]



2. 在弹出窗口中选择 [TARGET]->[ASSEMBLER FOR] (汇编); 或 [COMPILER FOR] (C)
3. 在窗口右边点击[options]



4. 在新弹出窗口中切换到[OUTPUT]页
5. 在[generate listing file]前打勾即可



19. 工程文件的组织方法

huangxd 发表于 2005-7-11 10:28 侃单片机

一个大的单片机程序往往包含很多模块，我是这样组织的

1. 每一个 C 源文件都要建立一个与之名字一样的 H 文件，里面仅仅包括该 C 文件的函数的声明，其他的什么也不会有，比如变量的定义啊等等不应该有。

2. 建立一个所有的文件都要共同使用的头文件，里面当然就是单片机的管脚使用的定义，还有里面放那些需要的 KEIL 系统的头文件，比如 `include<reg52.h>`, `#include<absacc.h>` 等等，把这个文件命名为 `common.h`，或者干脆就叫 `main.h`

3. 每个 C 源文件应该包含自己的头文件以及那个共同使用的头文件，里面还放自己本文件内部使用的全局变量或者以 `extern` 定义的全局变量

4. 主文件 `main.c` 里面包含所有的头文件包括那个共同使用的文件，`main.c` 里面的函数可以再做一个头文件，也可以直接放在文件的开头部分声明就可以了，里面一般还有中断服务程序也放在 `main.c` 里面

5. 对于那些贯穿整个工程的变量，可以放在那个共同使用的头文件里面，也可以用 `extern` 关键字在某个 C 源文件里面定义，哪个文件要使用就重复定义一下

6. 建立工程的时候，只要把 C 源文件加到工程中，把 H 文件直接放到相应的目录下面就可以了，不需要加到工程里面。

20. 关于 MON08

在 MON08 中疑惑的问题？

我看可很多 freescale 的单片机的 datasheet,在 MONitor ROM 一章中会提到有两种进入 monitor rom 的方式,一种为 normal 方式,一种为 forced 方式.forced 方式较为简单,不需要占用太多的 I/O 口,不需要一个高电压,但是它需要一个特殊的条件,就是

If \$FFFE and \$FFFF contain \$FF

其中\$FFFE and \$FFFF 是复位时的地址,也就是说只有在复位向量中包含 FFFF,FFFF 时在能进入这种方式.我对这一点不是很理解.它的意思是不是在芯片中没有程序的时候.才能进入 forced mode.

因为在上电复位的时候,\$FFFE and \$FFFF 中的数据会装入程序计数器中,我想问下,\$FFFE and \$FFFF 中在芯片为空的时候是什么数据?在芯片非空的时候是什么数据?

发贴时间： Jul 21 2005 9:15AM ||

strongchen

头衔： 版主

对，\$FFFE 和\$FFFF 就是复位中断的矢量地址。当芯片为空时，复位矢量也为空，其中的数据为\$FFFF。当芯片复位后，监控程序会检测复位矢量的值，如果发现数据为空（即为\$FFFF），监控程序会强制启动，使芯片进入监控模式，即所谓的强制方式（forced）。

发贴时间： Jul 21 2005 9:56AM ||

天涯倦客

那这种烧写方式意义好像不大啊,只有是空芯片才能进入监控方式.如果单只有这种方式,那好像就是一次性烧写了.

发贴时间： Jul 21 2005 10:16AM ||

strongchen

头衔： 版主

所以还有一种正常方式（normal）。当芯片不空时，通过特殊电平的设置，仍可以进入监控模式。

发贴时间： Jul 21 2005 11:34AM ||

天涯倦客

谢谢版主,这个我知道,但是这种方式需要占用一些 I/O 口,不过现在也在凑合着用.i/o 口尽量复



更多精彩文章，尽在《匠人的百宝箱》，网址：<http://cxjr.21ic.org>

用吧

发贴时间： Jul 21 2005 11:45AM ||

strongchen

头衔： 版主

只是复位时需要几个口线的电平设置。进入监控状态后，只需一个口进行通讯，其余的口线都被释放，可用回正常功能。

21. mon08 的仿真模式的断点

黄果树 发表于 2005-8-1 00:24 侃单片机 ←返回版面

code warror 功能强大,但是感觉界面没有 PIC 的 MPALB 友好,特别是仿真时另外弹出一个窗口,不能在此窗口修改源代码,编译.烦!

听说 code warror 软仿真功能强大,一直没有琢磨过,期望那天有高手出马,写本教程!

mon08 的仿真模式,断点只能设一个,占用 IO 口,没有经验的很难连接上,连接上也会很容易死掉,必须复位..还不如一些小日本的 mcu(也用片上仿真).目前我用过的最好的片上仿真的 8bit mcu 是 zilog 的新款 flash mcu,专用一个 DBG 口仿真烧录,一连就上.百发百中:)

22. 关于 MON08 调试的频率

由于强制通讯波特率为 FBUS/256,因此总线频率受到主机软件允许的标准波特率的限制
当晶振=9.8304MHZ 时,内部总线=9.8304/4=2.4576MHZ.

这里会有一个问题,即调试时的工作频率和产品的实际工作频率不一致

23. 关于运算中需要注意的问题

当不同长度的变量进行运算时,要特别当心,避免某些变量被“截肢”

//比如:

```
unsigned long aaa;
```

```
unsigned int bbb;
```

```
unsigned int ccc;
```

```
bbb=1000 ;
```

```
ccc=2000;
```

//下面的指令计算后,aaa 并没有如预期的变成 20000000,而是变成了 33920,原因是高位被裁减掉了

```
aaa=bbb*ccc;
```

//改成下面的指令后,结果正确

```
aaa=ccc;
aaa=aaa*bbb;
```

24. PLL 功能的启动方式

```
//-----
//启动 PLL 功能(8MHZ)
//
//总线频率与参数值表(外部晶振=32.768KHZ)
//-----
void PLL_ON(void)
{
    PCTL = 0;           //设置 PCTL，关闭中断

    PCTL_PRE = 0;      //P
    PCTL_VPR = 2;      //E
    PMS = 0X03D1;      //N
    PMRS = 0XD0;       //L
    PMDS = 1;          //R

    PCTL_PLLON = 1;    //启动 VCO 时钟
    PBWC_AUTO = 1;      //设置工作模式自动
    while(!PBWC_LOCK); //等待 PLL 稳定
    PCTL_BCS = 1;       //选择 PLL 信号为系统时钟源

    asm nop;
    asm nop;
}
```

25. 参考文章

您可以到《匠人的百宝箱》(<http://cxjr.21ic.org>) 搜索更多相关内容。

