填空：

1.

命名规则

下面以 MC9S12XS128MAA 芯片进行说明。

MC	9	S12X	S	128	M	AA
①	②	③	④	⑤	⑥	⑦

① 产品状态。MC(合格)PC工程测试品)。

② 存储器类型标志。9 表示片内带闪存 Flash。

③ 内核类型。S12X 使用内核 CPU12X，16 位单片机。

④ 子系列型号标志。S12XS 是 S12X 的一个子系列，故子系列名即为 S，一般把该子系列简称为 XS。

⑤ MCU 内 Flash 存储器大小（单位 KB）。128 表示内含 128KB 的 Flash 存储器。

⑥ 工作温度范围标志。C 表示-40~85℃；V 表示-40~105℃；M 表示-40~125℃。

⑦ 引脚个数与封装标志。AE=64LQFP（64 引脚 LQFP 封装）；AA=80QFP（80 引脚 QFP 封装）；AL=112LQFP（112 引脚 LQFP 封装）。

字长、主频范围

S12X 系列单片机的中央处理器 CPU12X 是 16 位 MCU，它的指令系统与 S12 兼容，CPU 主频 40MHZ。

2.

编程模型：

CPU12XS CPU 内部寄存器结构：8 位累加器 A、B，16 位累加器 D、变址寄存器 X、变址寄存器 Y、堆栈指针 SP、程序计数器 PC、条件码寄存器 CCR。

CCR 有哪些位？每一位啥意思？

程序状态寄存器 CCR 的内容分 2 部分。第 1 部分是 5 个算术特征位：H、N、Z、V、C，它们反映了上一条指令执行结果的特征；第 2 部分是 3 个 MCU 控制位：非屏蔽中断位 X、I 和 STOP 指令控制位 S，这 3 位通常由软件设定，控

制 S12CPU 的操作。

复位默认值：1101 0000B

读写	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
R / W	S	X	H	I	N	Z	V	C

CCR 是真正的专用寄存器，除 C、H 位之外，其他各位都不参与任何运算。CCR 各位的作用简要说明如下：

S STOP 指令禁止位。该位置 1 将禁止 CPU 执行 STOP 指令。

X 非屏蔽中断位 XIRQ 。该位置 1 将屏蔽来自 XIRQ 引脚的中断请求，复位默认值为 1。

I 中断屏蔽位。该位置 1 将屏蔽所有的可屏蔽中断源，复位默认值为 1。

H 辅助进位。该位 BCD 操作时累加器 A 的 Bit3 向 Bit4 进位。

N 符号位。当运算结果为负时，该位置 1。N 位实际是运算结果最高位的拷贝。

Z 0 标志。当运算结果为 0 时，该位置 1。

V 补码溢出标志。当运算结果出现补码溢出时，该位置 1。

C 进 / 借位标志。当加法运算产生进位或者减法运算产生借位时，该位置 1。移位操作或者直接针对 C 的指令也可改变 C 的值。

3

存储器：128KB 程序 Flash;8KB RAM;8KB 数据 Flash

4

关于中断：

中断是一种异步行为，通过硬件提供给 CPU 一个信号（高低电平）。

XS128 的中断过程详细描述如下：

中断请求、中断响应、中断服务程序

CPU 每执行完一条指令，若程序有开放某些中断及总中断（使用 CLI 指令），则 CPU 按照优先级次序查询所有中断标志位，若某个中断已发生，则响应该中断请求。当 CPU 收到一个许可的中断请求，它在响应的中断之前要完成当前正在执行的指令。中断顺序遵守与 SWI 指令相同的循环顺序，包括：

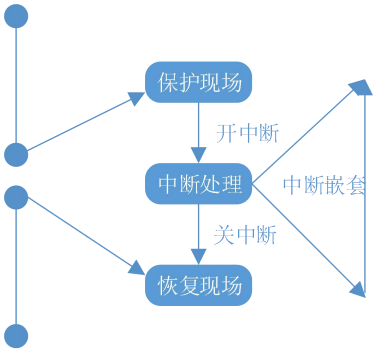
1、堆栈中保存 CPU 寄存器----CPU 内部的寄存器 PC、Y、X、D、CCR 依次进栈。

2、CCR 中的 I 位置 1，即自动关总中断（即相当于自动执行 SEI 指令），防

止其他中断进入。

3、在目前等待的中断中取出最高优先级中断的中断向量，从相应的中断向量地址取出中断向量（即中断服务例程的入口地址）送给 PC 寄存器。

4、执行中断服务例程，直到中断返回指令 RTI。RTI 指令从堆栈中依次弹出 CCR、D、X、Y、PC，CPU 返回原来中断处继续执行。这个过程包括：①保护现场②开中断③中断处理（先清中断请求）④关中断，恢复现场⑤中断退出执行中断返回指令 RTI。



5

伪指令：只能被汇编程序识别并指导汇编如何进行。

ORG 汇编起始指令

SECTION、OFFSET

EQU 等值指令

SET 设置符号值指令

FCB 定义字节常量指令 FDB 定义双字节常量指令

FCC 定义字符常量指令

DC 定义常量指令 DC.B 1 字节 DC.W 2 字节 DC.L 4 字节

DCB 定义常量块指令

RMB 保留内存字节指令

END、INCLUDE

XDEF 、XREF 汇编导入导出指令

ABSETNTRY

6 复位类型：6 种：上电复位、低电压复位、外部复位、看门狗复位、时钟监控复位、非法地址复位。

单选：

1 指令：S12X 汇编源程序格式为：标号 操作码 操作数 注释

1 St：将寄存器中的数据保存到内存中。7 种寻址方式
2 ld：将内存中的数据加载到寄存器中。部分不支持 IMM（立即寻址）
3 mov：将一个数据从源地址传送到目标地址中，不破坏源地址单元的内容。

例：

```
MOV    AX，2000H；将 16 位数据 2000H 传送到 AX 寄存器
MOV    AL，20H；将 8 位数据 20H 传送到 AL 寄存器
MOV    AX，BX；将 BX 寄存器的 16 位数据传送到 AX 寄存器
MOV    AL，[2000H]；将 2000H 单元的内容传送到 AL 寄存器
```

4 JMP：无条件转移指令（跳转，子程序地址→PC）支持 6 种寻址方式，除了 REL（相对寻址）
5 子程序调用与返回指令
BSR:　相对寻址　　-128~+127　　返回指令（RTS）
JSR:　　7 种寻址方式　64KB 地址空间　　返回指令（RTS）
CALL:　　6 种寻址方式　可以超过 64KB 地址空间。返回指令（RTC）

助记符	说明	寻址方式							
		INH 隐含/固有寻址	REL 相对寻址	DIR 直接寻址	EXT 扩展寻址	IDX 变址寻址	IDX1 9 位常数偏移变址寻址	[D,IDX] 累加器 D 间接变址寻址	[IDX2] 16 位常数偏移间接变址寻址
BSR	返回地址入栈 目标地址→PC		√						
JSR	返回地址入栈 目标地址→PC			√	√	√	√	√	√
RTS	返回地址出栈 →PC	√							
CALL	返回地址入栈 当前页号入栈 目标页号→PPAGE				√	√	√	√	√

	目标地址 →PC								
RTC	页号出栈 → PPAGE 返回地址 出栈→PC	√							

6、SWI：中断指令，通过中断响应的方式进入服务程序，返回指令为 RTI。

助记符	操　作	操作说明	寻址方式
SWI	(SP)-2→sp,(PC)→M(sp):M(sp+1); (sp)-2→SP,(YH:YL)→M(sp):M(sp+1); ……	软件中断指令，返回地址、寄存器 Y、X、D、CCR 依次入栈，可屏蔽中断标志 I 置 1，中断向量→PC	INH （隐含/固有寻址）
RTI	(M(sp))→CCR,(SP)+1→SP; (M(sp):M(sp+1))→B:A,(SP)+2→SP; ……	CCR、寄存器 D、X、Y 依次出栈，返回地址→PC	INH （隐含/固有寻址）

2 堆栈：实顶堆栈先转为指针然后再压栈。出栈：压栈，+1；入栈：压栈，-1；

虚顶堆栈先压栈然后再转为指针。出栈：+1，压栈；入栈：-1，压栈；

3Xs128 有 4 个地址寄存器：GPAGE、PPADE、EPAGE、RPAGE

表 5-1 XS128 的分页存储器映像

存储器类型	大小	GPAGE	全局首地址	页面寄存器	每页大小
RAM （0x0F_E000~0x0F_FFFF）	8KB	0x0F	0x0F_E000	RPAGE=0xFE	4KB
			0x0F_F000	RPAGE=0xFF	
D-Flash (0x10_0000~0x10_1FFF)	8KB	0x10	0x10_0000	EPAGE=0x00	1KB
			0x10_0400	EPAGE=0x01	
			0x10_0800	EPAGE=0x02	
			0x10_0C00	EPAGE=0x03	
			0x10_1000	EPAGE=0x04	
			0x10_1400	EPAGE=0x05	
			0x10_1800	EPAGE=0x06	
			0x10_1C00	EPAGE=0x07	
P-Flash (0x7E_0000~0x7F_FFFF)	128KB	0x7E	0x7E_0000	PPAGE=0xF8	16KB
			0x7E_4000	PPAGE=0xF9	
			0x7E_8000	PPAGE=0xFA	
			0x7E_C000	PPAGE=0xFB	
		0x7F	0x7F_0000	PPAGE=0xFC	
			0x7F_4000	PPAGE=0xFD	
			0x7F_8000	PPAGE=0xFE	
			0x7F_C000	PPAGE=0xFF	

4

监控程序：

用 D 命令显示一段内存的内容。按下 D 键后要填入需要显示的内存起始地址和终止地址，然后回车就会显示一段地址的内容。

M 命令来改变 RAM 内存中的内容。按 M 键后填入要改变的 RAM 内存地址。

之后会显示此 RAM 内存地址的内容，接着用户可以填入要改成的内容。如果修改成功，用户可以修改下一个地址的内容。如需要停止修改，按回车键即可。

R 命令显示所有 CPU 寄存器的内容。

Ctrl+Q 命令用来改变 Flash 页面寄存器 PPAGE 的值。

Ctrl+Q 命令显示 PPAGE 寄存器的值。然后用户可以填入希望修改的值。

Ctrl 键与 A、B、X、Y、C、P、G、R、E 键同时按下：分别改变 A 寄存器、B 寄存器、X 寄存器、Y 寄存器、CCR 寄存器、PC 寄存器、GPAGE 寄存器、RPAGE 寄存器和 EPAGE 寄存器的值。

①实验中先用 E 命令来擦除 Flash 的$4000—$7FFF 和$C000—$EFFF 两处。Flash 的$F000—$FFFF 处有监控程序受保护。然后用 L 命令下载程序到 Flash 中。

②在实验中从某一地址开始执行程序的操作分为两步：

1、用修改 PC 寄存器命令“Ctrl+P”修改映像寄存器 PC 的值；

2、用程序运行命令 G 把映像寄存器的内容复制到 CPU 个内部寄存器中并立即运行。

判断：

1

记住指令的寻址方式；大部分都是 INH，也有 REL

2

I/O 功能:通用 I/O 口通过配置相应寄存器位，可以设置输入/输出端口、驱动能力、内置上拉/下

拉电阻使用、中断输入方式等多种功能。

I/O 口复用功能：GPIO，串口、定时器、模拟 AD 转换 4 种

3

1）小端模式:字的低字节存储在低地址中,字的高字节存储在高地址中。（Inter）利于计算机处理

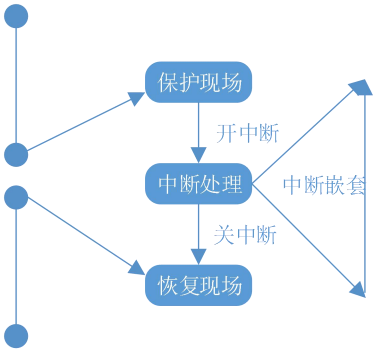
2)大端模式:字的低字节存储在高地址中,字的高字节存储在低地址中。(飞思卡尔)符合人类思维

问答

1 中断

XS128 的中断过程详细描述如下：中断请求、中断响应、中断服务程序

CPU 每执行完一条指令，若程序有开放某些中断及总中断（使用 CLI 指令），则 CPU 按照优先级次序查询所有中断标志位，若某个中断已发生，则响应该中断请求。当 CPU 收到一个许可的中断请求，它在响应的中断之前要完成当前正在执行的指令。



中断顺序遵守与 SWI 指令相同的循环顺序，包括：

1、堆栈中保存 CPU 寄存器----CPU 内部的寄存器 PC、Y、X、D、CCR 依次进栈。

2、CCR 中的 I 位置 1，即自动关总中断（即相当于自动执行 SEI 指令），防止其他中断进入。

3、在目前等待的中断中取出最高优先级中断的中断向量，从相应的中断向量地址取出中断向量（即中断服务例程的入口地址）送给 PC 寄存器。

4、执行中断服务例程，直到中断返回指令 RTI。RTI 指令从堆栈中依次弹出 CCR、

D、X、Y、PC，CPU 返回原来中断处继续执行。这个过程包括：①保护现场②开中断③中断处理（先清中断请求）④关中断，恢复现场⑤中断退出执行中断返回指令 RTI。

2

1、C 语言程序建造过程：首先利用交叉编译器对 C 程序进行编译，再将 C 代码翻译成汇编代码，然后再用汇编器、连接器生成目标文件。

2、启动代码：①初始化堆栈；②初始化 RAM 为变量分配空间，然后将 ROM 中存储的数据拷贝到 RAM 中；③调用 main 函数。

3 数据和变量的实现

局部变量

4 C 函数的实现

首先在声明定义，参数存入堆栈中，调用函数时将参数左到右压栈

5 监控程序的功能与使用

单片机的监控程序有以下功能：

1、实现人机对话，这是人与单片机通信的基本手段；

2、显示和修改 CPU 内部各寄存器的值；

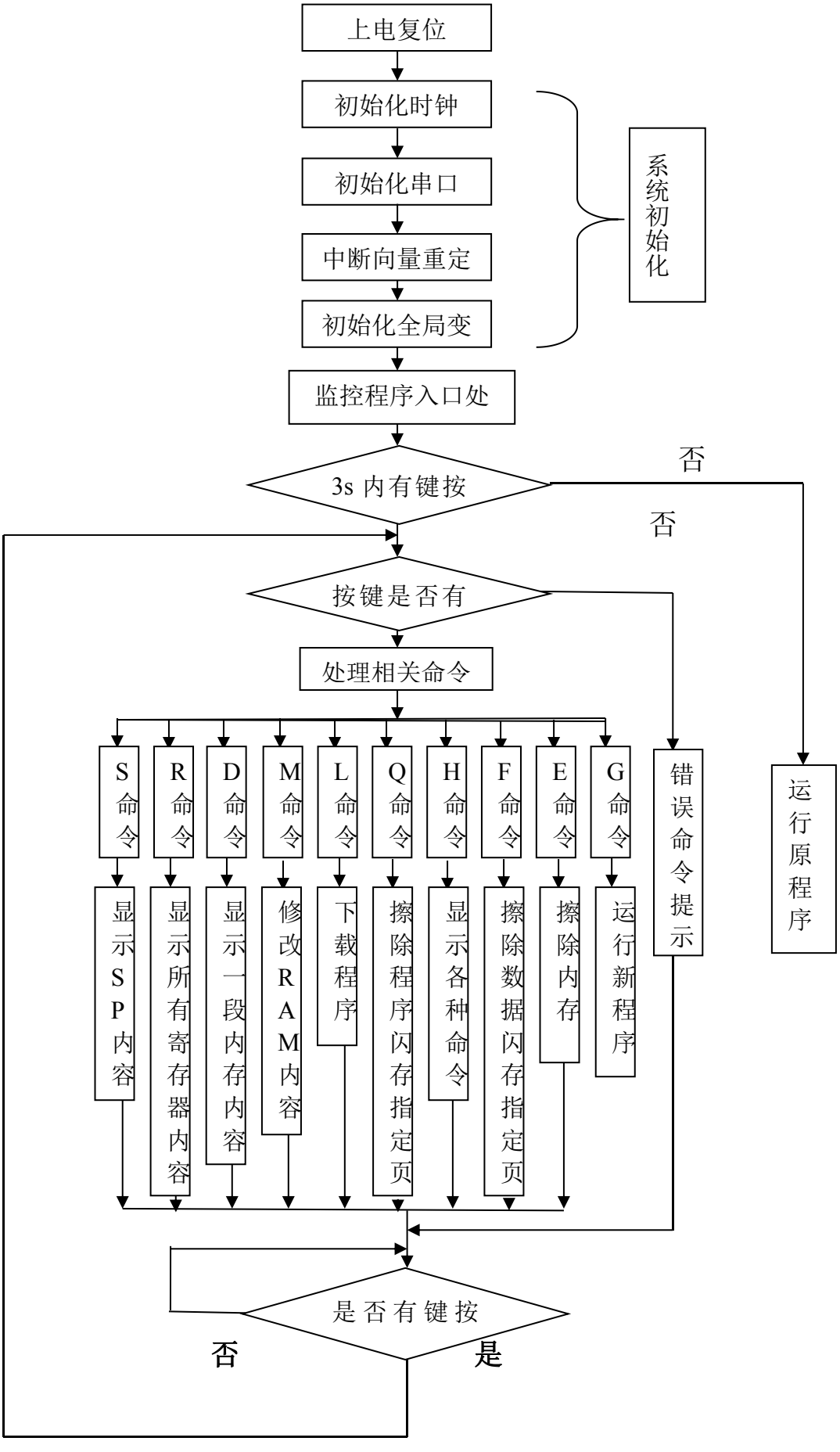
3、显示和修改某一段内存的值；

4、通过串行口向单片机系统内存中装载程序；

5、能在被调试程序中设置断点，以观看到程序执行的中间过程；

6、从内存任意地址执行程序。

监控程序的工作流程


上电复位
初始化时钟
初始化串口
中断向量重定
初始化全局变
系统初始化
监控程序入口处
3s 内有键按
否
否
按键是否有
处理相关命令
S命令
R命令
D命令
M命令
L命令
Q命令
H命令
F命令
E命令
G命令
错误命令提示
运行原程序
显示SP内容
显示所有寄存器内容
显示一段内存内容
修改RAM内容
下载程序
擦除程序闪存指定页
显示各种命令
擦除数据闪存指定页
擦除内存
运行新程序
是否有键按
否
是

编程：

全亮、流水灯

```
#include <hidef.h>
#include "derivative.h"
#define uchar unsigned char
#define On 1 #define Off 0
uchar change = 0;
void PLL_Init(void){
  CLKSEL_PLLSEL = 0;
  PLLCTL_PLLON = 1;
  SYNR = 0xC0|0x04;
  REFDV = 0x80|0x01; POSTDIV = 0x00;
  asm NOP
  asm NOP
  while(!(CRGFLG_LOCK == 1)){ }
  CLKSEL_PLLSEL = 1;
}
void delay(uchar m){
  unsigned int i,j;
  for(i=0;i<m;i++)
    for(j=0;j<1000;j++){
    }}
void LED_Light(void){
  delay(220);
  PORTB<<=1;
  PORTB |= 0x01;
  if(PORTB == 0xFF)
    PORTB = 0xFE;}
void main(void) {
  PORTB = 0xFE;
  DDRB = 0xFF;     //将 B 口初始化为输出口
  IRQCR = 0xC0;
     EnableInterrupts;
  for(;;) {
    if(change%2==On)
      LED_Light();
    else
      PORTB = 0x00;
  } }
#pragma  CODE_SEG __NEAR_SEG NON_BANKED
interrupt VectorNumber_Virq
void IRQ_ISR(void)
{  change++;}
```

行列键盘

```
#include <hidef.h>
#include <MC9S12XS128.h>
#pragma LINK_INFO DERIVATIVE "mc9s12xs128"
#define uchar unsigned char
#define KB_P PORTA
#define KB_D DDRA  //键盘宏定义
void Keyboard_Init(void);
uchar Keyboard_Scan(void);
void main(void) {
        uchar data=0;
        DDRB = 0xFF;
        PORTB = 0xFF;
        Keyboard_Init();
        for(;;)
        {
                data = Keyboard_Scan();
                PORTB = data;
        }  }
void Keyboard_Init(void){
        KB_P = 0x00;
        KB_D = 0x0F;
        PUCR |= 0x01;
}
uchar Keyboard_Scan(void){
        uchar line,i,tmp;
        line = 0b11111110;
        for(i=0;i<=3;i++) {
                tmp = KB_P;
                tmp |= 0b00001111;
                KB_P = tmp & line;
                asm("NOP");
                asm("NOP");
                tmp = KB_P;
                tmp &= 0b11110000;
                if(tmp != 0b11110000) {
                        tmp = KB_P;
                        break;}
                else
                line = (line<<1)|0x01;
        }
        if(i == 4) tmp = 0xFF;
        return tmp;
}
```

SCI：

```
#include <hidef.h>
#include <MC9S12XS128.h>
#pragma LINK_INFO DERIVATIVE "mc9s12xs128"
unsigned int temp;
unsigned char TPP, i;
void Delay_us(unsigned int n_us);
void Delay_ms(unsigned int n_ms);
void SCI_Init(void);
void main(void)
{
        DisableInterrupts;
        SCI_Init( );
        PORTB =0X00;
        DDRB  =0XFF;
        while(SCI0SR1_TC==0);
                SCI0DRL ='R';
        for(;;)
        {
            while(SCI0SR1_RDRF==0);
                        SCI0DRL = SCI0DRL;
                while(!SCI0SR1_TC);
                temp = SCI0DRL;
            if(temp<0x39&&temp>0x30)
                {
 switch(temp)
   {
    case 0x31:PORTB=0xfe; break;
    case 0x32:PORTB=0xfc; break;
    case 0x33:PORTB=0xf8; break;
    case 0x34:PORTB=0xf0; break;
    case 0x35:PORTB=0xe0; break;
    case 0x36:PORTB=0xc0; break;
    case 0x37:PORTB=0x80; break;
    case 0x38:PORTB=0x00; break;
    default: break;        }
              TPP = PORTB;
               for(i = 0; i < 100; i ++)
              {Delay_ms(1000);   //延时 1s
                 PORTB<<=1;
                  PORTB |=0X01;
          if(PORTB==0xff)
                PORTB =TPP;  }  }}}
```

```
void SCI_Init(void)
{
        SCI0BD = 52;
        SCI0CR1 =0X00;
        SCI0CR2 =0X0C;
}
void Delay_us(unsigned int n_us)
{
        unsigned int loop_i;
        for(loop_i=0;loop_i<n_us;loop_i++);
}
void Delay_ms(unsigned int n_ms)
{
   unsigned int loop_i;
   for(loop_i=0;loop_i<n_ms;loop_i++)
   {
     Delay_us(667);              //总线时钟
8MHZ 时，667*12 个机器周期约 1ms
   }
}
```

数码管静态显示

```
#include <hidef.h>
#include <MC9S12XS128.h>
#pragma LINK_INFO DERIVATIVE "mc9s12xs128"
void main(void)
{
        DDRB = 0xFF;    //将段选口设置为输出
        DDRK = 0xFF;    //将位选口设置为输出
        PORTK = 0x7E;  //01111110 最后一位为
0，第一个数码管被选中  K 口为位选口，B 口为段
选口     PORTB = 0x80;  //10000000，显示数字
“8”，最高位是小数点，为 1，表示不显示小数点
        for(;;) {;}
}
// 位 选 表 （ 1~6 个 数 码 管 ） ：
0x7E,0x7D,0x7B,0x77,0x6F,0x5F
// 段 选 表 （ 数 字 0~9 ， 不 显 示 小 数 点 ）
0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x8
0,0x90
```

AD:

```
#include <hidef.h>
#include "derivative.h"
void SCI_init(void);
void SCISendChar(char c1);
void SCISendString(char *pc1);
void SCISendHex(char c1);
void ATD_init(void);
unsigned char dtoa(char c1);
void Delay_ms(unsigned int n_ms)
{
        unsigned int i,j;
        for(i=0;i<n_ms;i++)
        {
                for(j=0;j<667;j++)
                      { ; }
        }
}
void main(void)
{
        SCI_init();
        ATD_init( );
        DDRB = 0XFF;
        PORTB = 0XFF;
        ATD0CTL5=0X45;
        while(!(ATD0STAT0&0X80));
        SCISendString("VRL=");
        SCISendHex(ATD0DR0H);
        SCISendString("\r\n") ;
        ATD0CTL5=0X44;
        while(!(ATD0STAT0&0X80));
        SCISendString("VRH=");
        SCISendHex(ATD0DR0H);
        SCISendString("\r\n");
        ATD0CTL5=0X46;
        while(!(ATD0STAT0&0X80));
        SCISendString("(VRL+VRH)/2=");
        SCISendHex(ATD0DR0H);
        SCISendString("\r\n");
        ATD0CTL5=0X47;
        while(!(ATD0STAT0&0X80));
        SCISendString("AN7=");
        SCISendHex(ATD0DR0H);
        SCISendString("\r\n");
        for(;;)
        {
                ATD0CTL5=0X00;
                //MULT=1 多通道采样转换
                while(!(ATD0STAT0&0X80));
                SCISendString("AN1=");
                SCISendHex(ATD0DR0H);
                PORTB= ATD0DR0H;
                SCISendString("\r\n");
                Delay_ms(1000);
        }
}
void SCI_init(void){
     SCI0BD = 52;
     SCI0CR1 =0X00;
     SCI0CR2 =0X0C;
}
void SCISendChar(char c1){
        while(SCI0SR1_TC==0);
        SCI0DRL =c1;
}
void SCISendString(char *pc1){
        while((*pc1) !=0)
        {
                SCISendChar(*pc1);
                pc1++;
        }
}
void SCISendHex(char c1){
     SCISendChar(dtoa((c1&0XF0)>>4));
     SCISendChar(dtoa((c1&0x0f)));
}
unsigned char dtoa(char c1){
     return (c1>=10)?(c1+0x37):(c1+0x30);
}
void ATD_init(void) {
        ATD0CTL1 =0X0F;  //8 位精度
        ATD0CTL2 =0X40;  //打开 CCF 快速清零位，关闭外部触发输入，关闭中断
        ATD0CTL3 =0X08;  //只转换一个通道，数据左对齐 non_fifo，转换序列长度为 1
        ATD0CTL4 =0XE3;  //采样时间为 24 个 ATD 时钟周期，ATDCLK=8MBHZ/8=1MHZ
}
```

锁相环 PIT、CRG 数码管显示数字 0、1、2、3、4、5

```
#include <hidef.h>
#include <MC9S12XS128.h>
#pragma LINK_INFO DERIVATIVE "mc9s12xs128"
unsigned int count = 0;
unsigned char m = 0;
unsigned char n;
unsignedcharLED_DATA[]={0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90};
// 数字 0 到 9 的段选码
unsignedcharLED_CS[]={0x3E,0x3D,0x3B,0x37,0x2F,0x1F};  // 6 为数码管的位选码
void PLL_init(void);        //初始化时钟
void PORTB_init(void);      //初始化 B 口
void PIT_init(void);        //初始化 PIT
void PORTK_init(void);      //初始化 K 口
void main(void) {
 PLL_init();
PORTB_init();
PORTK_init();
PIT_init();
EnableInterrupts;
  for(;;)
  {
  _FEED_COP();
  }
}
void PLL_init(void) {
CLKSEL = 0x00;      //不加载 IPLL 到系统
SYNR = 0x00|0x04;
 //SYNR 为 0b00000100 即值 SYNDIV 值为 4
REFDV = 0x40|0x03;
 //REFDV 为 0b01000011 即 REFDIV 值为 3
POSTDIV =0x00;
 PLLCTL_PLLON = 1;     //启动 IPLL
_asm(nop);
_asm(nop);
while(!(CRGFLG_LOCK == 1));// 延时以等待 IPLL 稳定
CLKSEL_PLLSEL = 1; /加载 IPLL 到系统
}
void PORTB_init(void) {
 DDRB = 0xFF;            // B 口为输出
 PORTB = 0x00;
}
void PORTK_init(void){
DDRK = 0xFF;                    // K 口为输出
PORTK = 0x00;
}
void PIT_init(void)  {
  PITCFLMT_PITE = 0;          //禁止 PIT 模块
  PITCE_PCE0 = 1;           //使能定时器通道 0
  PITMUX_PMUX0 = 1;
  PITMTLD1 = 20 - 1;
  PITLD0 = 100 - 1;
  PITINTE_PINTE0 = 1;
PITCFLMT_PITE = 1;           //使能 PIT 模块
}
#pragma CODE_SEG __NEAR_SEG NON_BANKED
void interrupt 66 PIT0(void)
{
  count++;
  n = m%6;           //使用六位数码管
  if(count == 1)  // 处理数码管位选与段选
  {
      PORTB = 0xFF;
      PORTB = LED_DATA[n];
      PORTK = LED_CS[n];
      count = 0;
      m++;                        //下一位
  }
  PITTF_PTF0 = 1;               //清中断标志
}
#pragma CODE_SEG DEFAULT
```