NV32F100x FLASH 模块编程

第一章 库函数简介

库函数列表

uint16_t Flash_Init(void)；
Flash 模块初始化
uint16_t Flash_Program1LongWord(uint32_t wNVMTargetAddress, uint32_t dwData)；
加载一个字
uint16_t Flash_Program2LongWords(uint32_t wNVMTargetAddress, uint32_t dwData0, uint32_t dwData1)；
加载两个字
uint16_t Flash_Program(uint32_t wNVMTargetAddress, uint8_t *pData, uint16_t sizeBytes)；
Flash 加载编程
uint16_t Flash_EraseSector(uint32_t wNVMTargetAddress)；
擦除一个扇区
uint16_t NVM_EraseAll(void)；
整片擦除
void EFM_LaunchCMD(uint32_t EFM_CMD)；
命令加载
uint16_t Flash_VerifyBackdoorKey()；
加密区密码认证
uint16_t EEPROM_Erase(uint32_t adr)；
模拟 EEPROM 擦除命令(所在地址的 512Bytes 的 EEPROM)；
uint32_t EEPROM_Read(uint32_t adr)；
读取对应地址的 EEPROM
uint16_t EEPROM_Write(uint32_t adr, uint32_t Data)；
写对应地址的 EEPROM
uint16_t EEPROM_WriteByte(uint32_t adr, uint8_t Data)；
字节写入
uint8_t EEPROM_ReadByte(uint32_t adr)；
字节读取
uint16_t EERPOM_Writeup4byte(uint32_t adr,uint8_t *pData,uint32_t length)；
定义长度写入

***NV32 的 Flash 存储器包含以下特性：**

带有校验功能的自动化编程，擦除算法
快速扇区擦除和长字程序操作
灵活的保护方案以防止意外的编程和擦除操作
无需外部高电压电源即可进行 Flash 存储器的编程和擦除操作
编程完成时可产生中断
利用加密机制防止未经授权访问 Flash 存储器
可利用 Flash 模拟 EEPROM 编程

FLASH 的具体参数见 NV32F100X 系列用户手册和参考手册。

***NV32 的 Flash 存储器地址映射：**

其中非易失性存储器（NVR）地址为：
0x00400000-0x00401400（5K) + 0x00401400-0x00401600(512Bytes)

全局地址	大小	说明
0x0000_0000-0x0000_8000	32KB	Flash 数据块包含 Flash 配置字段。
0x0000_0000-0x0000_FFFF	64KB	Flash 数据块包含 Flash 配置字段。
0x0000_0000-0x0001_FFFF	128KB	Flash 数据块包含 Flash 配置字段。

特别提醒：

NV32 对 FLASH 的写操作是以字为单位的。

NV32 擦除 FLASH 的命令须在 SRAM 中进行。

NV32 擦除 FLASH 须以扇区（512BYTES）为单位。

Flash 模块的寄存器能够以 32 位、16 位（对齐到[31:16]数据或[15:0]数据）或 8 位存取。如果是可写的寄存器，在 Flash 命令执行期间将会禁止写入访问。写保留地址位置没有影响并读取返回 0。

在编程之前，Flash 字节或长字必须处于已擦除状态。不允许对 Flash 中已经编程（写 0）的区域重复编程（写 0）。

1.1 Flash 模块初始化

函数名	Flash_Init
函数原形	Flash_Init(void)
功能描述	初始化 FLASH
输入参数	无
输出参数	无
返回值	无
先决条件	无
函数使用实例	Flash_Init()

```
/*******************************************************************************//*!
*
* @初始化 Flash 模块
*
*******************************************************************************/
uint16_t Flash_Init(void)
{
    uint16_t err     = FLASH_ERR_SUCCESS;
    uint32_t clkDIV = BUS_CLK_HZ/1000000L - 1;
    uint32_t Tpgs    =(285 *(BUS_CLK_HZ/100))/1000000L;
    uint32_t Tprog =(675*(BUS_CLK_HZ/100))/1000000L;
    EFMCR=(clkDIV<<24) + 0x00000001; //divide to 1M hz
    EFMETM0=(Tpgs<<16)   + 0x00001194;   //0x00281194
    EFMETM1=(Tprog<<16) + 0x000088B8;
  return(err);
}
```

1.2 加载一个字

函数名	Flash_Program1LongWord
函数原形	Flash_Program1LongWord(uint32_t wNVMTargetAddress, uint32_t dwData)
功能描述	加载一个字的大小，编程到 FLASH 中
输入参数	目标写入地址 wNVMTargetAddress 数据 dwData
输出参数	无
返回值	状态
先决条件	无
函数使用实例	Flash_Program1LongWord(0x800, 0x100)

```
/*******************************************************************************//*!
*
* @加载一个字即 4 个 BYTE 编程
*
```

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

uint16_t Flash_Program1LongWord(uint32_t wNVMTargetAddress, uint32_t dwData)
{
    uint16_t err = FLASH_ERR_SUCCESS;
    //判断是否为字对齐
    if(wNVMTargetAddress & 0x03)
    {
        err = FLASH_ERR_INVALID_PARAM;
        return (err);
    }
    // 清除错误标志
    EFMCMD = FLASH_CMD_CLEAR;
    M32(wNVMTargetAddress) = dwData;//写入数据到对应的地址中
    EFM_LaunchCMD(FLASH_CMD_PROGRAM);//加载编程命令
    return (err);//返回状态
}
```

1.3 加载两个字

函数名	Flash_Program2LongWords
函数原形	Flash_Program2LongWords(uint32_t wNVMTargetAddress, uint32_t dwData0, uint32_t dwData1)
功能描述	加载两个字的大小，编程到 FLASH 中
输入参数	目标写入地址 wNVMTargetAddress 数据 dwData0，dwData1
输出参数	无
返回值	状态
先决条件	无
函数使用实例	Flash_Program2LongWord(0x1000, 0x100,0x100)

```
/*****************************************************************************************
*
* @加载两个字即 8 个 BYTE 编程
*
*****************************************************************************************/
uint16_t Flash_Program2LongWords(uint32_t wNVMTargetAddress, uint32_t dwData0, uint32_t dwData1)
{
    uint16_t err = FLASH_ERR_SUCCESS;

    //判断是否为字对齐
    if(wNVMTargetAddress & 0x03)
    {
        err = FLASH_ERR_INVALID_PARAM;
        return (err);
```

```
    }
    // 清除错误标志
    EFMCMD = FLASH_CMD_CLEAR;

    M32(wNVMTargetAddress) = dwData0; //存放数据到以目标地址为起始的 4 个字节的空间中

    EFM_LaunchCMD(FLASH_CMD_PROGRAM);//0x20000000,加载编程命令
    wNVMTargetAddress = wNVMTargetAddress +4;//地址是字对齐的，地址向后移一个字

    // 清除错误标志
    EFMCMD = FLASH_CMD_CLEAR;

    M32(wNVMTargetAddress) = dwData1;//第二个数据放入处理后的地址的 4 个字节的空间中

    EFM_LaunchCMD(FLASH_CMD_PROGRAM);//加载编程命令
    return (err);//返回处理状态
}
```

1.4 Flash 加载编程

函数名	Flash_Program
函数原形	Flash_Program(uint32_t wNVMTargetAddress, uint8_t *pData, uint16_t sizeBytes)
功能描述	编程到 FLASH 中
输入参数	目标写入地址 wNVMTargetAddress 数据*pData，字节长度 sizeBytes
输出参数	无
返回值	状态
先决条件	无
函数使用实例	Flash_Program(0x1000,&u8DataBuff[0],512)

```
/*****************************************************************************************
*
* @Flash 编程的函数
*@输入参数：uint32_t wNVMTargetAddress  所要存放的 FLASH 首地址
*          uint8_t *pData               所要存放的数据
*          uint16_t sizeBytes           字节长度
*@总的过程就是，给出字节长度，先写入满足两个字的个数，再写入剩余满足一个字的个数，最后写入
*剩余的字节数
*
*****************************************************************************************/

uint16_t Flash_Program(uint32_t wNVMTargetAddress, uint8_t *pData, uint16_t sizeBytes)
{
```

```
    uint16_t err = FLASH_ERR_SUCCESS;
    uint16_t w2LongWordCount = sizeBytes>>3;//处理一下，得到 2 个字的个数
    uint8_t    wLeftBytes = (sizeBytes & 0x07);//低位三个字节的个数
    uint16_t wLeftLongWords = wLeftBytes>>2;//低位中满一个字个数
    uint32_t wTargetAddress = wNVMTargetAddress;
    uint32_t dwData0,dwData1;
    uint32_t *pdwData = (uint32_t*)pData;//传参
    int    i;
    //判断是否字对齐
    if(wNVMTargetAddress & 0x03)
    {
        err = FLASH_ERR_INVALID_PARAM;
        return (err);//返回无效的参数
    }
    //循环写入两个长字（即 8 个字节），共写入 w2LongWordCount * 8 个字节
    for(i = 0; i < w2LongWordCount; i++)
    {
        dwData0 = *pdwData++;
        dwData1 = *pdwData++;
        err = Flash_Program2LongWords(wTargetAddress, dwData0, dwData1);//加载两个字的编程
        if(err) //地址不为 4 个字节对齐，则直接跳转
        {
            goto EndP;
        }
        wTargetAddress += 8;//循环一次写入 8 个字节，即 2 个字，NV32 的 flash 是字对齐的
    }
    // 一个字的编程，即 4bytes
    for(i = 0; i < wLeftLongWords; i++)
    {
        dwData0 = *pdwData++;
        err = Flash_Program1LongWord(wTargetAddress, dwData0);
        if(err)
        {
            goto EndP;
            //break;
        }
        wTargetAddress += 4;
    }
  wLeftBytes = (wLeftBytes-(wLeftLongWords<<2));//在两字和一字的编程都处理完后剩余的低两位字节数
    if(!wLeftBytes) //若无剩余字节数，返回成功
    {
        return (err);
    }
    dwData0 = 0xFFFFFFFFL;
```

```
    pData = (uint8_t*)pdwData+wLeftBytes-1;
    for(i = wLeftBytes; i >0; i--)//关于剩余字节数的编程处理
    {
        dwData0 <<= 8;//8 位，即一个字节
        dwData0 |= *pData--;
    }

    err = Flash_Program1LongWord(wTargetAddress, dwData0);//最后的字节数据写入到一个字的空间中上
EndP:
    return (err);
}
```

1.5 擦除一个扇区

函数名	Flash_EraseSector
函数原形	Flash_EraseSector(uint32_t wNVMTargetAddress)
功能描述	擦除输入目标地址的一个扇区
输入参数	目标写入地址 wNVMTargetAddress
输出参数	无
返回值	状态
先决条件	无
函数使用实例	Flash_EraseSector(0x1000)

```
/*******************************************************************************//*!
*
* @擦除目标地址的一个扇区
*
*******************************************************************************/
uint16_t    Flash_EraseSector(uint32_t wNVMTargetAddress)
{
    uint16_t err = FLASH_ERR_SUCCESS;
    // 判断是否字对齐
    if(wNVMTargetAddress & 0x03)
    {
        err = FLASH_ERR_INVALID_PARAM;//
        return (err);
    }
    // 清除错误标志
      EFMCMD = FLASH_CMD_CLEAR;
      M32(wNVMTargetAddress) = 0xffffffff;
      EFM_LaunchCMD(FLASH_CMD_ERASE_SECTOR);//加载擦除命令
    return (err);
}
```

1.6 整片擦除

函数名	NVM_EraseAll
函数原形	NVM_EraseAll(void)
功能描述	整片擦除
输入参数	无
输出参数	无
返回值	状态
先决条件	无
函数使用实例	NVM_EraseAll()

```
/*****************************************************************************
*
* @整片擦除 Flash
*
*****************************************************************************/
uint16_t NVM_EraseAll(void)
{
    uint16_t err = FLASH_ERR_SUCCESS;
    EFMCMD = FLASH_CMD_CLEAR;
    EFM_LaunchCMD(FLASH_CMD_ERASE_ALL);
    return err;
}
```

1.7 命令加载

函数名	EFM_LaunchCMD
函数原形	EFM_LaunchCMD(uint32_t EFM_CMD)
功能描述	命令加载
输入参数	命令参数 EFM_CMD
输出参数	无
返回值	无
先决条件	无
函数使用实例	EFM_LaunchCMD(FLASH_CMD_ERASE_ALL)

```
/*****************************************************************************//*!
*
* @Flash 命令加载函数，注：此函数须放入 SRAM 中运行
*
*****************************************************************************/
void EFM_LaunchCMD(uint32_t EFM_CMD)
{
```

```
  DisableInterrupts;
        if((EFMCMD&EFM_DONE_MASK)== EFM_STATUS_READY)
        {
      EFMCMD = EFM_CMD;
        }
    while(1)
    {
        if((EFMCMD&EFM_DONE_MASK) == EFM_STATUS_DONE) break;
    }
 EnableInterrupts;
}
```

1.8 擦除 EEPROM

函数名	EEPROM_Erase
函数原形	EEPROM_Erase(uint32_t adr)
功能描述	擦除目标地址 512bytes 的 EEPROM
输入参数	目标地址 adr
输出参数	无
返回值	状态
先决条件	无
函数使用实例	EEPROM_Erase(0x00)

```
/*******************************************************************************
*
* @EEPROM 擦除命令
*@输入参数：地址函数将会擦掉 adr 所在的 512bytes 的 EEPROM
*
*******************************************************************************/
uint16_t EEPROM_Erase(uint32_t adr)
{
    uint16_t err = EEPROM_ERR_SUCCESS;
    uint32_t e_adr;

     if(adr & 0x03)   // 先判断是否字对齐
    {
        err = EEPROM_ERR_INVALID_PARAM;
    return (err);
      }

    if(adr >1024) //大于 1KB，返回溢出
    {
        err=EEPROM_ADR_OverFlow;
```

```
    return(err);
  }

  e_adr=adr+EEPROM_START_ADR;
  err = Flash_EraseSector(e_adr);
  return(err);
}
```

1.9 读取 EEPROM

函数名	EEPROM_Read
函数原形	EEPROM_Read(uint32_t adr)
功能描述	读取目标地址的 EEPROM
输入参数	目标地址 adr
输出参数	无
返回值	状态
先决条件	无
函数使用实例	EEPROM_Read(0x00)

```
/*******************************************************************************
*
* @读取相应地址的 EEPROM
* @输入参数：所要读取的 EEPROM 的地址
*
*******************************************************************************/
uint32_t EEPROM_Read(uint32_t adr)
{
  uint16_t err = EEPROM_ERR_SUCCESS;
  uint32_t e_adr;
  uint32_t data;
  /*判断输入地址是否有效*/
  if(adr & 0x03)
  {
    err = EEPROM_ERR_INVALID_PARAM;
  return (err);
  }

  if(adr >1024) //大于 1KB
  {
    err=EEPROM_ADR_OverFlow;//溢出
    return(err);
  }
```

```
    e_adr=adr+EEPROM_START_ADR;//计算 EEPROM 的起始的地址
    data = M32(e_adr);//取地址的内容
    return(data);

}
```

1.10 写入 EEPROM

函数名	EEPROM_Write
函数原形	EEPROM_Write(uint32_t adr, uint32_t Data)
功能描述	写入目标地址的 EEPROM
输入参数	目标地址 adr,数据 Data
输出参数	无
返回值	状态
先决条件	无
函数使用实例	EEPROM_Write(24,0x55aa)

```
/*******************************************************************************
*
*@ EEPROM 写函数，写对应地址所在的 EEPROM
*@在写之前先读取，判断 EEPROM 是否为空，如果为空，则直接写
*如果非空，则先把整个 512bytes 的扇区读取到 RAM，修改要写的位置
*然后再写入 FLASH,模拟一个写 EEPROM 的过程
*@输入参数：地址，数据
*
*******************************************************************************/
uint16_t EEPROM_Write(uint32_t adr, uint32_t Data)
{

    uint32_t err = EEPROM_ERR_SUCCESS;
    uint32_t e_adr;
    uint32_t r_data;
    uint16_t i;
    uint32_t start_adr;
    uint32_t EEPROM_DATA[128];
    /*判断地址是否有效*/
    if(adr & 0x03)
    {
        err = EEPROM_ERR_INVALID_PARAM;
    return (err);
    }
```

```
    if(adr >1024) //大于 1KB，返回溢出
    {
        err=EEPROM_ADR_OverFlow;
      return(err);
    }

    r_data = EEPROM_Read(adr);

    e_adr=adr+EEPROM_START_ADR;

    if(r_data== EEPROM_BLANK) //如果要写的位置是空的，则直接写 0xffffffff
    {
      err= Flash_Program1LongWord(e_adr,Data);
    }
  else if((r_data&Data) == Data)//如果要写的位置对应的位，和要写的数据一样，或者为 1，可以直接写
     {
      err= Flash_Program1LongWord(e_adr,Data);
    }
    else if(r_data == Data) //如果要写的位和现有的数据是一样的，不进行任何操作，直接返回
        {
      return(err);
    }
    else
    {
        start_adr = e_adr & EEPROM_SECTOR_MASK; //计算出对应 Sector 的头地址

        for( i=0;i<128;i++ ) //如果要写的位置不为空，则先把数据从 FLASH 中取出来，在 RAM 中修改
        {
          EEPROM_DATA[i] = M32(start_adr + 4*i);
        }
        EEPROM_DATA[(adr&EEPROM_ARRAY_ADR_MASK)>>2] = Data; //修改 RAM 中的数据

        err=EEPROM_Erase(adr);

        err=Flash_Program(start_adr,(uint8_t*)EEPROM_DATA,512);//然后再写入
    }
    return(err);

}
```

1.11 字节写入

函数名	EEPROM_WriteByte
函数原形	EEPROM_WriteByte(uint32_t adr, uint8_t Data)
功能描述	字节写入目标地址的 EEPROM
输入参数	目标地址 adr,数据 Data
输出参数	无
返回值	状态
先决条件	无
函数使用实例	EEPROM_WriteByte(132, 0x13)

```
/*******************************************************************************
*
*@字节写入函数
*
*******************************************************************************/
uint16_t EEPROM_WriteByte(uint32_t adr, uint8_t Data) //一个字节一个字节的写
{
    uint32_t err = EEPROM_ERR_SUCCESS;
    uint32_t data_mask;
    uint32_t r_data;
    uint32_t data_m0;
    uint32_t data_m1;
    uint32_t word_adr = adr &0x3fc;
    uint32_t   b_sit= adr & 0x3;
 //先让高位为 FF
    data_m0 = Data << b_sit*8;
    data_mask = 0xFFFFFFFF<<(b_sit+1)*8;
//然后让低位为 FF
     data_m1 = 0xFFFFFFFF >> (32-b_sit*8);
     data_m1 = data_m1 | data_m0 | data_mask ;
    r_data = EEPROM_Read(word_adr);
    data_m1 = data_m1 & r_data;
    err = EEPROM_Write(word_adr , data_m1);
    return(err);
}
```

1.12 字节读取

函数名	EEPROM_ReadByte
函数原形	EEPROM_ReadByte(uint32_t adr)
功能描述	EEPROM 字节读取
输入参数	目标地址 adr
输出参数	无
返回值	状态
先决条件	无
函数使用实例	EEPROM_ReadByte(132)

```
/*******************************************************************************
*
*@Byte 读取
*
*******************************************************************************/
uint8_t EEPROM_ReadByte(uint32_t adr)
{
    uint32_t r_data;
    uint32_t word_adr = adr &0x3fc;
    uint32_t b_sit= adr & 0x3;
    uint8_t   data;

 r_data = EEPROM_Read(word_adr);
    data = (r_data>>b_sit*8)& 0xff;
    return(data);
}
```

1.13 定义长度写入

函数名	EERPOM_Writeup4byte
函数原形	EERPOM_Writeup4byte(uint32_t adr,uint8_t *pData,uint32_t length)
功能描述	定义字节长度，写入数据到 EEPROM
输入参数	EEPROM 定义字节长度写入，数据*pData，定义长度 length
输出参数	无
返回值	状态
先决条件	无
函数使用实例	EERPOM_Writeup4byte(340,u8DataBuff,512)

```
/*****************************************************************************
*
*@写函数，写一个长度为 bytesize 到 EEPROM 中
*@先把 1K 的 EEPROM 读取放入 SRAM 中，然后修改要写的位置
*
*****************************************************************************/
uint16_t EERPOM_Writeup4byte(uint32_t adr,uint8_t *pData,uint32_t length)
{
    uint8_t buf[512];
    uint8_t *pbuf;
    uint32_t e_adr;
    uint32_t e_sec;
    uint32_t e_offset;
    uint32_t a;
    uint32_t err = EEPROM_ERR_SUCCESS;

    if((adr + length )>1024)   //定义超过 1KB 就返回溢出
    {
        err=EEPROM_ADR_OverFlow;
      return(err);
    }

    e_adr=adr+EEPROM_START_ADR; //计算出 EEPROM 起始的地址
    e_sec=e_adr & EEPROM_SECTOR_MASK;
    e_offset=e_adr & 0x1ff;//计算偏移量


    while (length>0){
          //如果起始地址不为 0，即有偏移量，或者长度小于 512，都进入这个循环
        if (e_offset||(length<512))
        {
            pbuf=buf;
            a=512-e_offset;//扇区剩余的大小
            a=(length>a?a:length);//如果 length 大于 a，就取 a，否则值取 length
            memcpy(buf,(uint8_t*)e_sec,512);
            memcpy(&buf[e_offset],pData,a);
            pData+=a;
            length-=a;
            e_offset=0;

        }
        Else
        {           //如果起始地址等于 0，且长度大于 512
            pbuf=pData;//直接把这个数值赋给 BUF
```

```
            pData+=512;//然后写满一个扇区
            length-=512;//全减
        }
        err=Flash_EraseSector(e_sec);//擦除一个扇区
        err=Flash_Program(e_sec,(uint8_t*)pbuf,512);//然后写入 FLASH
        e_sec+=0x200;
    }
    return err;
}
```

第二章 样例程序

2.1 Flash 模拟 EEPROM

```
/*****************************************************************************
*
* @该例程使用了 NV32 的片上 FLASH 来模拟 EEPROM，实现字节的编程
*   也提供了关于 FLASH 初始化，擦写的流程。
*
*****************************************************************************/

#include "common.h"
#include "rtc.h"
#include "flash.h"
#include "sysinit.h"
#include "eeprom.h"
int main (void);
int main (void)
{
    char ch;
    uint32_t i;
    uint8_t   u8DataBuff[512];
    sysinit();
    printf("\nRunning the Flash_demo project.\n");
    LED0_Init();   //初始化 LED0
    LED2_Init();
    Flash_Init();

    /* Erase 99th sector */
    Flash_EraseSector(100*FLASH_SECTOR_SIZE);

    for(i=0;i<512;i++)
    {
        u8DataBuff[i] = (uint8_t)i;
    }

    Flash_Program( 100*FLASH_SECTOR_SIZE,&u8DataBuff[0],512 );

    for( i=0;i<512/16;i++ )
    {
        for(ch =0;ch<16;ch++)
        {
```

```
        }
    }

/*****************************************************************************
*
*EEPROM 使用范例
*EEPROM 第一次使用前可以先 erase,这样可以提高写的
*一次只能写一个双字，如果写 byte,可以先处理一下再写
*小于 10byte,单独调用 EEPROM_Write 效率可能更高，EEPROM_Writeup4byte 建议在写较长的数据时使用
*
*****************************************************************************/

  EEPROM_Erase(0x00);
  EEPROM_Erase(0x200);    //建议在最开始进行 erase 初始化，提高效率，一个扇区
  for(i=0;i<256;i++)
    {
       EEPROM_Write(4*i,1024-4*i); //1KB=1024BYTES
    }

     /*任意位置再写一个*/
     EEPROM_Write(24,0x55aa);
     EEPROM_Write(68,0xaa55);
     EEPROM_Write(128,0x3fec);
     EEPROM_Write(156,0xccbb);
     EEPROM_Write(256,0xbbcc);
     EEPROM_Write(264,0xccdd);
     EEPROM_Write(300,0x3fff);
     EEPROM_Write(512,0x5f5f);
     EEPROM_Write(900,0x9f9f);

 for(i=0;i<256;i++)
  {
    printf("adr:%d =0x%x \n",4*i,EEPROM_Read(4*i));
  }
 //批量写
  printf("/*************************批量测试*************************/ \n");

  //跨界写，1-2 个 Sector 写
  EERPOM_Writeup4byte(340,u8DataBuff,512);
  for(i=0;i<256;i++)
  {
  printf("adr    %d =0x%x \n",4*i,EEPROM_Read(4*i));
  }
```

```
    EEPROM_Erase(0x00);
    EEPROM_Erase(0x200);

    printf("/**************************byte 写测试**************************/ \n");
    EEPROM_WriteByte(132,0x13);
    printf("adr :132 =0x%x \n",EEPROM_Read(132));
    EEPROM_WriteByte(134,0x14);
    printf("adr :134 =0x%x \n",EEPROM_Read(132));
    EEPROM_WriteByte(133,0x12);
    printf("adr :133 =0x%x \n",EEPROM_ReadByte(133));
    EEPROM_WriteByte(135,0x15);
    printf("adr :135 =0x%x \n",EEPROM_ReadByte(135));

        while(1)
            {
                ch = in_char();
                out_char(ch);

            }
}
```