AN028-C8051F30X系列软件SPI例子

相关器件

此应用笔记适用于下列器件：
C8051F300, C8051F301, C8051F302和C8051F303。

导言

此应用笔记收集了用软件实现的主模式SPI程序。提供了8个不同的SPI主模式传输例子。示例中包含SPI时钟相位和极性两个子例程：一个是用C语言编写的例程，一个是为了提高速度用汇编语言编写的可被C语言调用的例程。一个例子是在“C”程序中如何调用汇编同时提供了EEPROM接口例子。SPI是摩托罗拉商标。

此文档中描述的SPI功能使SPI处理的软件最小化。在系统中使用C8051F30X器件作为总线上的SPI主器件。

硬件接口

这些例子是使用通用IO引脚作为SPI接口。

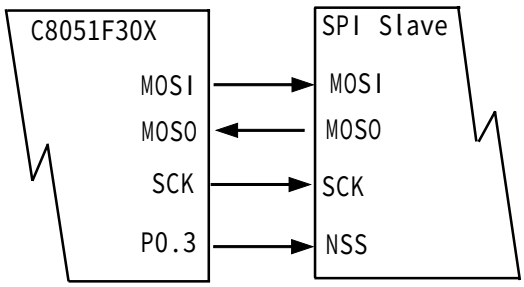
MOSI（主出/从入）：此引脚用于C8051F30X器件串行数据输出，此引脚设置为数据推挽输出。

MISO（主入/从出）：此引脚用于串行数据从从器件输入，此引脚设置为开漏数据引脚。

SCK（串行时钟）：此引脚作为C8051F30X器件的串行时钟输出，设置为数据推挽输出。

此外，如果从器件需要从选择信号，需要第四个通用IO引脚，且必须声明为数据推挽输出。所有这些专用的通用IO口应被数据交叉开关跳过。图1是SPI主（C8051F30X）和SPI从器件的连接图。

图1.硬件配置



CYGNAL Integrated Products, Inc.
4301 Westbank Drive
Suite B-100
Austin, TX 78746
www.cygnal.com

沈阳新华龙电子有限公司
沈阳市和平区青年大街284号58号信箱
Tel：024-23930366 23895360 Fax:23940230
Email: longhua@mail.sy.ln.cn
www.xhl.com.cn

函数描述

在此应用笔记中包含 8 个 SPI 主模式例程。四个不同 SPI 模式（模式 1，模式 2，模式 3 和模式 4）都给出了“ C ”和汇编例子。表 1 立列出了实现每一种模式的源文件。所有的程序使用相同的原型函数可被“ C ”调用。正因为如此，当生成项目时只有一个执行例程被调用。如果在同一系统中需要多个 SPI 模式，则函数可从命名。

当输入一个参数时函数接收单一字符并返回一单一字符。在 MOSI 引脚传输的参数 MSB 优先。函数从 MISO 返回一个接收的数据字节。SCK 相位和极性由 SPI 模式文件决定，当生成项目时次文件被包含。

执行	文件名
模式 0，C 语言	SPI_MODE0.c
模式 0，汇编语言	SPI_MODE0.asm
模式 1，C 语言	SPI_MODE1.c
模式 1，汇编语言	SPI_MODE1.asm
模式 2，C 语言	SPI_MODE2.c
模式 2，汇编语言	SPI_MODE2.asm
模式 3，C 语言	SPI_MODE3.c
模式 3，汇编语言	SPI_MODE3.asm

SPI 时序

当实现软件 SPI 主模式时，确保从器件的时序要求是非常重要的。因为 C8051F30X 器件能够在高速时操作，为了适应SPI从器件的时序这里介绍的程序需要修改。四种 SPI 模式都有各自特殊的串行时钟相位和极性，如图 2 所示。此外，C 和汇编程序有不同的时序要求。图 3 为模式 0 和模式 3 的时序。图 4 为模式 1 和模式 2 的时序。表 2 给出了对应每个时序参数的系统时钟数。

一个以快速时钟运行的系统，为了适应 SPI 从器件的时序要求需要放慢速度或修改 。

图 2.串行时钟相位/极性

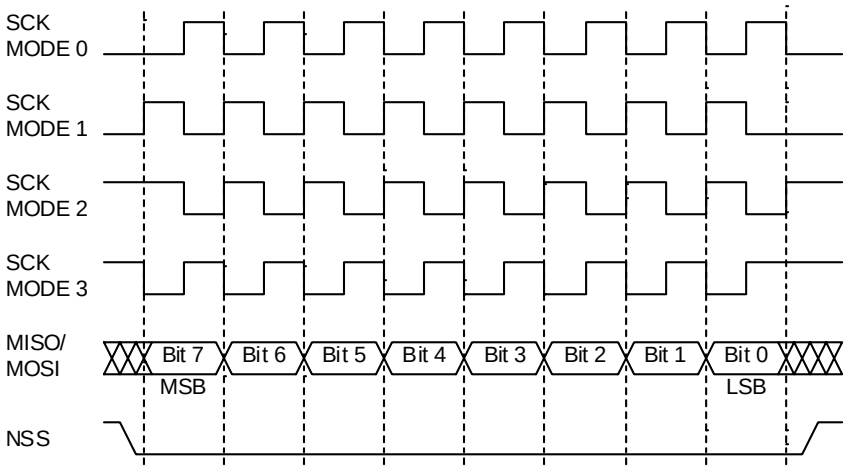


图 3.模式 0 和模式 3 的时序

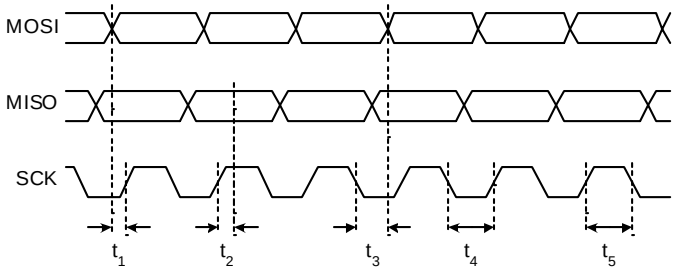


图 4.模式 1 和模式 2 的时序

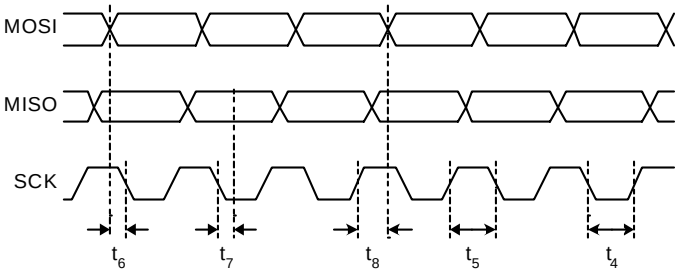


表 2.SPI 时序参数

参数	描述	SPI 模式	C 时序时钟数	汇编时序时钟数
T1	MOSI 有效到 SCK 高（MOSI 建立）	模式 0 模式 3	6 6	2 2
T2	SCK 高到 MISO 锁存	模式 0 模式 3	2 2	2 3
T3	SCK 低到 MOSI 变化（MOSI 保持）	模式 0 模式 3	7 4	5 2
T4	SCK 低时间	模式 0 模式 1 模式 2 模式 3	13 11 8 10	7 7 5 5
T5	SCK 高时间	模式 0 模式 1 模式 2 模式 3	8 10 13 11	5 5 7 7
T6	MOSI 有效到 SCK 低（MOSI 建立）	模式 1 模式 2	6 6	2 2
T7	SCK 低到 MISO 锁存	模式 1 模式 2	2 2	3 2
T8	SCK 高到 MOSI 变化（MOSI 保持）	模式 1 模式 2	4 7	2 5
-	从函数调用返回函数	所有模式	182	113

使用函数

在“C”程序中使用一个例子中的 SPI 函数，含有函数的文件首先必须汇编或编译。所得的目标文件能被加到项目的生成列表中并链接到主（调用）软件。

为了满足所需功能主软件必须正确配置 C8051F30X 器件的通用 IO 引脚。参看第一页的“硬件接口”。SPI_Transfer( )函数的函数原型也需要在所有调用它的文件中声明。

适用于所有例程函数的“C”原型是：

```
extern char SPI_Transfer(char);
```

“extern”限定符告诉链接器函数本身将在一个单独的目标文件中定义。

调用函数用下面的程序行：

```
in_spi = SPI_Transfer(out_spi);
```

in_spi和out_spi是字符型变量分别用于输入和输出 SPI 字节。

例子使用代码

包含两个完整的“ C ”程序示范软件 SPI 的使用。第一个例程，“ SPI_F300_Test.c ”，示范了 SPI 程序的调用方法。第二个例子，“ SPI_EE_F30x.c ”，使用模式 0 或模式 3 实现串行 EEPROM 接口的 SPI 程序。

SPI_F300_Test.c

在“ SPI_F300_Test.c ”文件中，一个 for 循环用于从 0-255 的重复计数。当输出字节时使用 for 循环变量 test_counter，SPI_return 变量是 SPI 输入字节。在函数调用选择从器件前 NSS 信号被拉到低，当调用函数取消选择它后 NSS 被拉到高电平。在 SPI 数据传输后，SPI 输入和输出字节传输到 UART，可以在 PC 终端监控。

测试例子代码，文件名为“ SPI_F300_Test.c ”，“ SPI_defs.h ”，且例子函数文件应该放置在一个单一目录。“ SPI_defs.h ”中包含执行 SPI 时所使用的四个引脚的 sbit 声明。在生成时文件包含 SPI_Transfer( )且“ SPI_F300_Test.c ”文件应该分别被编译或汇编和包含。一旦代码编程到 C8051F30X 器件的 FLASH 中，MISO 和 MOSI 引脚能被连到一起校验移入和移出数据。使用 PC 终端（配置为 115,200 波特，8 位数据位，无奇偶校验，1 个停止位，无溢出控制）通过 RS-232 电平转换器连接到 UART，PC 机终端程序能显示在 MOSI 上送出的数据，同样可以看到 MISO 的接收。这种配置输出结果的一部分类似：

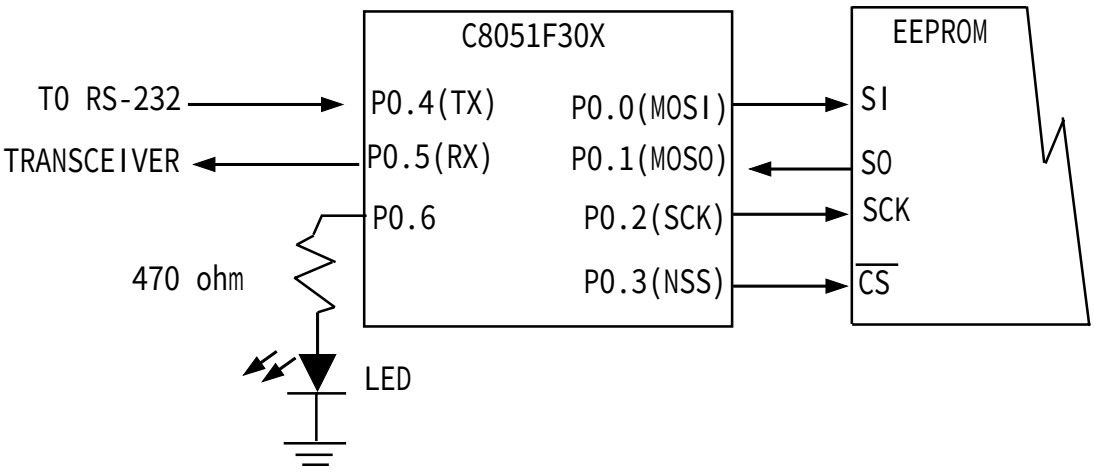
SPI Out = 0xFC, SPI In = 0xFC
SPI Out = 0xFD, SPI In = 0xFD
SPI Out = 0xFE, SPI In = 0xFE
SPI Out = 0xFF, SPI In = 0xFF
SPI Out = 0x00, SPI In = 0x00
SPI Out = 0x01, SPI In = 0x01
SPI Out = 0x02, SPI In = 0x02
SPI Out = 0x03, SPI In = 0x03
SPI Out = 0x04, SPI In = 0x04
SPI Out = 0x05, SPI In = 0x05

连接MOSI和MISO引脚不能完全地测试执行 SPI 的功能性。但是，它可以校验程序处理 MOSI 和 MISO 是否正确。

SPI_EE_F30x.c

“SPI_EE_F30x.c”使用一个 SPI 程序来读、写一个 SPI EEPROM（Microchip 25LC320）。为了与 EEPROM 的 SPI 接口兼容，必须使用模式 O 或模 3 的 SPI 函数。例子代码写入 FLASH 有两种不同的模式，通过读回 EEPROM 的内容校验写入器件的模式，并且与原有模式核对。当写 EEPROM 时，发光二极管（连接到 P0.6）点亮，当读 EEPROM 时，发光二极管熄灭。如果发生读错误，程序将停止。如果未发生错误，发光二极管闪烁表明测试成功。程序的进程可以通过 PC 机的终端（配置为 115,200 波特，8 位数据位，无奇偶校验，1 个停止位，无溢出控制）通过 RS-232 电平转换器连接到 UART 来监控。图 5 为此例子的 C8051F30x 和 EEPROM 的连接图。

图 5.EEPROM 连接



软件例子

```
//-----------------------------------------------------------------------------
// SPI_defs.h
//-----------------------------------------------------------------------------
// Copyright 2001 Cygnal Integrated Products, Inc.
//
//  作者: BD
//  日期: 2001 年 12 月 7 日
//
//  此文件定义 SPI 引脚。
// SPI 引脚映射到 P0.0 - P0.3,  但也可以定义到器件
//  的其它任何可用的 GPIO 引脚。
//
#ifndef SPI_DEFS
#define SPI_DEFS
sbit MOSI = P0^0; //  主出/从入(输出)
sbit MISO = P0^1; //  主入/从出(输入)
sbit SCK = P0^2; //  串行时钟(输出)
sbit NSS = P0^3; //  从选择(输出到片选)
#endif
```



```
//-----------------------------------------------------------------------------
// SPI_MODE0.c
//-----------------------------------------------------------------------------
// Copyright 2001 Cygnal Integrated Products, Inc.
//
// 作者：  BD
// 日期：  2001 年 12 月 14 日
//
// 此文件包含一个模式 0 执行 SPI 器件的‘C’程序。
//
// 目标 t: C8051F30x
// 链接工具：  KEIL C51 6.03 / KEIL EVAL C51
//
//
#include <c8051f300.h> // SFR 声明
#include “SPI_defs.h” // SPI 端口定义
//-----------------------------------------------------------------------------
// SPI_Transfer
//-----------------------------------------------------------------------------
//
// 使用 SPI 协议同时发送和接受一个字节 <SPI_byte>
//  SCK 空闲为低，在 SCK 上升时位锁存。
//
// 此程序的时序如下：
//
//参数                                                    时钟数
// MOSI 有效到 SCK 上升沿                                  6
// SCK 上升到 MISO 锁存                                    2
// SCK 下降到 MOSI 有效                                    7
// SCK 高时间                                              8
// SCK 低时间                                              13
char SPI_Transfer (char SPI_byte)
{
unsigned char SPI_count; // SPI 办理计数器
for (SPI_count = 8; SPI_count > 0; SPI_count--) // 单个字节 SPI 循环
{
MOSI = SPI_byte & 0x80; // 放当前输出位到 MOSI
SPI_byte = SPI_byte << 1; // 移下一位到 MSB
SCK = 0x01; // 设置 SCK 为高
SPI_byte |= MISO; // 在 MISO 上捕捉当前位
SCK = 0x00; // 设置时钟为低
}
return (SPI_byte);
```



```
} //结束  SPI_Transfer

;------------------------------------------------------------------------
; Copyright (C) 2001 CYGNAL INTEGRATED PRODUCTS, INC.
; All rights reserved.
;
; 文件名：   SPI_MODE0.ASM
; 日期：2001 年 12 月 14 日
; 目标 MCU：: C8051F30x
; 描述：这是一个用 C8051F30X 器件端口执行主 SPI 的子程序（函数）
; 此函数可在 C 程序中调用
; 函数原型如下：
;
;
; extern char SPI_Transfer (char);
;
; 注意：  时序如下（模式 0SPI）：
; 参数                                          系统时钟数
; MOSI 有效到  SCK 上升                          2
; SCK  上升到  MISO  锁存                        2
; SCK  下降到  MOSI 有效                         5
; SCK  高时间                                    5
; SCK  低时间                                    7
;
;------------------------------------------------------------------------
NAME SPI_MODE0
?PR?_SPI_Transfer?SPI_MODE0 SEGMENT CODE
PUBLIC _SPI_Transfer
$include (c8051f300.inc) ; 寄存器定义包含文件
$include (SPI_defs.h) ; 包含 SPI 位定义
RSEG ?PR?_SPI_Transfer?SPI_MODE0
_SPI_Transfer:
USING 0
MOV A, R7 ; 在 A 中存储传递变量
MOV R7, #08H ; 装载  R7 计数位
RLC A ; 移位 MSB 到进位位
SPI_Loop: MOV MOSI, C ;将位移出到 MOSI
SETB SCK ; 时钟高
MOV C, MISO ; 将 MISO 移到进位位
RLC A ; 循环移位进位位到  A 中
CLR SCK ; 时钟低
DJNZ R7, SPI_Loop ; 循环直到其它位完成
MOV R7, A ; 在 R7 中存储返回值
?C0001:
```



```
RET ;从程序中返回
END ； 文件结束

//-----------------------------------------------------------------------
// SPI_MODE1.c
//-----------------------------------------------------------------------
// Copyright 2001 Cygnal Integrated Products, Inc.
//
// 作者： BD
// 日期：2001 年 12 月 14 日
//
//  此文件包含一个模式 1 执行主 SPI 器件的‘C’程序。
//
// 目标 t: C8051F30x
// 链接工具： KEIL C51 6.03 / KEIL EVAL C51
//
//
#include <c8051f300.h> // SFR 声明
#include “SPI_defs.h” //SPI 端口定义
//-----------------------------------------------------------------------
// SPI_Transfer
//-----------------------------------------------------------------------
//
// 使用 SPI 协议同时发送和接受一个字节 <SPI_byte>
//  SCK 空闲为低，在 SCK 下降时位锁存。
//
// 此程序的时序如下：
//
//参数                                                    时钟周期数

// SCK 上升沿到 MOSI 有效                                  4
// MOSI 有效到 SCK 下降沿                                  6
// SCK 下降到 MISO 锁存                                    2
// SCK 高时间                                              10
// SCK 低时间                                              11
char SPI_Transfer (char SPI_byte)
{
unsigned char SPI_count; // counter for SPI transaction
for (SPI_count = 8; SPI_count > 0; SPI_count--) // 单一字节 SPI 循环
{
SCK = 0x01; // 设置 SCK 为高
MOSI = SPI_byte & 0x80; // 放当前输出位到 MOSI
SPI_byte = SPI_byte << 1; // 移下一位到 MSB
SCK = 0x00; // 设置 SCK 为低
```



```
SPI_byte |= MISO; //  在 MISO 上捕捉当前位
}
return (SPI_byte);
} //  结束 SPI_Transfer

;------------------------------------------------------------------------
; Copyright (C) 2001 CYGNAL INTEGRATED PRODUCTS, INC.
; All rights reserved.
;
;  文件名：   SPI_MODE1.ASM
;  日期：14 DEC 01
;  目标  MCU : C8051F30x
;  描述：这是一个用 C8051F30X 器件端口执行主 SPI 的子程序（函数）
;  此函数可在 C 程序中调用
;  函数原型如下：
;
; extern char SPI_Transfer (char);
;
;  注意：时序如下  模式 1 SPI)：
;  参数                                    时钟数
; SCK  上升到  MOSI 有效                    2
; MOSI 有效到  SCK  下降                    2
; SCK  下降到  MISO  锁存                   3
; SCK  高时间                               5
; SCK  低时间                               7
;
;------------------------------------------------------------------------
NAME SPI_MODE1
?PR?_SPI_Transfer?SPI_MODE1 SEGMENT CODE
PUBLIC _SPI_Transfer
$include (c8051f300.inc) ;寄存器定义包含文件
$include (SPI_defs.h) ; SPI 位定义
RSEG ?PR?_SPI_Transfer?SPI_MODE1
_SPI_Transfer:
USING 0
MOV A, R7 ;  在 A 中存储传递变量
MOV R7, #08H ;装载  R7 计数位
SPI_Loop: SETB SCK ;  时钟高
RLC A ;将 MSB 移位到进位位 t
MOV MOSI, C ;将位移出到  MOSI
CLR SCK ;时钟低
MOV C, MISO ;  将  MISO 移到进位位
DJNZ R7, SPI_Loop ;  循环直到其他位完成
RLC A ;循环移位进位位到  A 中
```



```
MOV R7, A ;  在 R7 中存储返回值
?C0001:
RET ;从程序中返回
END ;文件结束


//-----------------------------------------------------------------------------
// SPI_MODE2.c
//-----------------------------------------------------------------------------
// Copyright 2001 Cygnal Integrated Products, Inc.
//
// 作者:  BD
// 日期:  2001 年 12 月 14 日
//
// 此文件包含一个模式 2 执行主 SPI 器件的‘C’程序。
//
// 目标 t: C8051F30x
// 链接工具:  KEIL C51 6.03 / KEIL EVAL C51
//
//
#include <c8051f300.h> // SFR 声明
#include “SPI_defs.h” // SPI 端口定义
//-----------------------------------------------------------------------------
// SPI_Transfer
//-----------------------------------------------------------------------------
//
// 使用 SPI 协议同时发送和接受一个字节 <SPI_byte>
//  SCK 空闲为高，在 SCK 下降时位锁存。
//
// 此程序的时序如下:
//
//参数                                    时钟数
// MOSI 有效到 SCK 下降沿                  6
// SCK  下降到 MISO 锁存                   2
// SCK  上升到  MOSI 有效                  7
// SCK  低时间                             8
// SCK  高时间                             13
char SPI_Transfer (char SPI_byte)
{
unsigned char SPI_count; // SPI 办理计数器
for (SPI_count = 8; SPI_count > 0; SPI_count--) //单个字节  SPI 循环
{
MOSI = SPI_byte & 0x80; //放当前输出位到  MOSI
SPI_byte = SPI_byte << 1; //  移下一位到  MSB
```



```
SCK = 0x00; //  设置  SCK  为低
SPI_byte |= MISO; //在 MISO 上捕捉当前位
SCK = 0x01; //  设置  SCK  为高
}
return (SPI_byte);
} //结束 SPI_Transfer


;------------------------------------------------------------------------
; Copyright (C) 2001 CYGNAL INTEGRATED PRODUCTS, INC.
; All rights reserved.
;
; 文件名：  SPI_MODE2.ASM
; 日期：2001 年 12 月 14 日
; 目标 MCU：: C8051F30x
; 描述：这是一个用 C8051F30X 器件端口执行主 SPI 的子程序（函数）
; 此函数可在 C 程序中调用
; 函数原型如下：
;
; extern char SPI_Transfer (char);
;
;注意：时序如下(模式  2 SPI)：
; 参数                                          系统时钟数
; MOSI 有效到  SCK  下降                          2
; SCK  下降到  MISO  锁存                         2
; SCK  上升到  MOSI  有效                         5
; SCK  低时间                                     5
; SCK  高时间                                     7
;
;------------------------------------------------------------------------
NAME SPI_MODE2
?PR?_SPI_Transfer?SPI_MODE2 SEGMENT CODE
PUBLIC _SPI_Transfer
$include (c8051f300.inc) ;寄存器定义包含文件
$include (SPI_defs.h) ;包含 SPI 位定义
RSEG ?PR?_SPI_Transfer?SPI_MODE2
_SPI_Transfer:
USING 0
MOV A, R7 ;  在 A 中存储传递变量
MOV R7, #08H ;  装载  R7 计数位
RLC A ;移位  MSB 到进位位
SPI_Loop: MOV MOSI, C ;将位移出到  MOSI
CLR SCK ;  时钟低
MOV C, MISO ;将  MISO 移到进位位
```



```
RLC A ;循环移位进位位到 A 中
SETB SCK ; 时钟高
DJNZ R7, SPI_Loop ;循环直到其他位完成
MOV R7, A ; 在 R7 中存储返回值
?C0001:
RET ;从程序中返回
END ;文件结束


//-----------------------------------------------------------------------
// SPI_MODE3.c
//-----------------------------------------------------------------------
// Copyright 2001 Cygnal Integrated Products, Inc.
//
// 作者: BD
// 日期: 2001 年 12 月 14 日
//
// 此文件包含一个模式 3 执行主 SPI 器件的‘C’程序。
//
// 目标 t: C8051F30x
// 链接工具: KEIL C51 6.03 / KEIL EVAL C51
//
//
#include <c8051f300.h> //SFR 声明
#include “SPI_defs.h” // SPI 端口定义
//-----------------------------------------------------------------------
// SPI_Transfer
//-----------------------------------------------------------------------
//
// 使用 SPI 协议同时发送和接受一个字节 <SPI_byte>
// SCK 空闲为高，在 SCK 上升时位锁存。
//
// 此程序的时序如下:
//
//参数                                                    时钟数
// SCK 下降沿到 MOSI 有效                  4
// MOSI 有效到 SCK 上升沿                  6
// SCK 上升到 MISO 锁存                    2
// SCK 低时间                              10
// SCK 高时间                              11
char SPI_Transfer (char SPI_byte)
{
unsigned char SPI_count; // SPI 办理计数器
for (SPI_count = 8; SPI_count > 0; SPI_count--) //单个字节 SPI 循环
```



```
{
SCK = 0x00; // 设置 SCK 为低
MOSI = SPI_byte & 0x80; // 放当前输出位到 MOSI
SPI_byte = SPI_byte << 1; // 移下一位到 MSB
SCK = 0x01; // 设置 SCK 为高
SPI_byte |= MISO; //在 MISO 上捕捉当前位
}
return (SPI_byte);
} //结束 SPI_Transfer


;------------------------------------------------------------------------
; Copyright (C) 2001 CYGNAL INTEGRATED PRODUCTS, INC.
; All rights reserved.
;
; 文件名： SPI_MODE3.ASM
; 日期：2001 年 12 月 14 日
; 目标 MCU：: C8051F30x
; 描述：这是一个用 C8051F30X 器件端口执行主 SPI 的子程序（函数）
; 此函数可在 C 程序中调用
; 函数原型如下：
;
; extern char SPI_Transfer (char);
;
; 注意： 时序如下（模式 3SPI）：
; 参数                                    系统时钟数
; SCK 下降到 MOSI 有效                     2
; MOSI 有效到 SCK 上升                     2
; SCK 上升到 MISO 锁存                     3
; SCK 低时间                               5
; SCK 高时间                               7
;
;------------------------------------------------------------------------
NAME SPI_MODE3
?PR?_SPI_Transfer?SPI_MODE3 SEGMENT CODE
PUBLIC _SPI_Transfer
$include (c8051f300.inc) ; 寄存器定义包含文件
$include (SPI_defs.h) ;包含 SPI 位定义
RSEG ?PR?_SPI_Transfer?SPI_MODE3
_SPI_Transfer:
USING 0
MOV A, R7 ; 在 A 中存储传递变量
MOV R7, #08H ;装载 R7 计数位
SPI_Loop: CLR SCK ; 时钟低
```



```
RLC A ;移位  MSB 到进位位
MOV MOSI, C ;将位移出到  MOSI
SETB SCK ;  时钟高
MOV C, MISO ;  将  MISO 移到进位位
DJNZ R7, SPI_Loop ;循环直到其他位完成
RLC A ;移位进位位到 A 中
MOV R7, A ;   在 R7 中存储返回值
?C0001:
RET ;  从程序中返回
END ;文件结束
```

```
//-----------------------------------------------------------------------------
// SPI_F300_Test.c
//-----------------------------------------------------------------------------
// Copyright 2001 Cygnal Integrated Products, Inc.
//
// 作者:   BD
// 日期:   2001 年 12 月 14 日
//
// 此程序示范 C8051F30X 处理主 SPI 程序集是如何在一个 C 程序中使用的。
//
// 此程序设置 C8051F30X 器件的通用 IO 引脚为适当功能，
// 接着用  SPI_Transfer 函数通过 SPI 引脚发送和接收信息。
// 当发送信息时，通过 UART 口与 PC 机的终端程序连接来
// 监视程序进程。
//
// 为了实现代码功能，下列文件之一应该被编译或汇编，
// 最终生成的目标文件必须与从这些文件产生的目标文件连接:
//
// SPI_MODE0.c  模式 0 主 SPI 器件的‘C’语言实现
// SPI_MODE0.asm  模式 0 主 SPI 器件的汇编语言实现
// SPI_MODE1.c Mode 1  模式 1 主 SPI 器件的‘C’语言实现
// SPI_MODE1.asm Mode 1  模式 1 主 SPI 器件的汇编语言实现
// SPI_MODE2.c Mode 2  模式 2 主 SPI 器件的‘C’语言实现
// SPI_MODE2.asm Mode 2  模式 2 主 SPI 器件的汇编语言实现
// SPI_MODE3.c Mode 3  模式 3 主 SPI 器件的‘C’语言实现
// SPI_MODE3.asm Mode 3  模式 3 主 SPI 器件的汇编语言实现
//
// 目标器件: C8051F30x
// 连接工具: KEIL C51 6.03 / KEIL EVAL C51
//
//-----------------------------------------------------------------------------
// 包含文件
```



```
//-----------------------------------------------------------------------
#include <c8051f300.h> // SFR 声明
#include <stdio.h> // 标准 I/O
#include “SPI_defs.h” // SPI 端口定义
//-----------------------------------------------------------------------
// C8051F30X 的 16 位 SFR 定义
//-----------------------------------------------------------------------
sfr16 DP = 0x82; // 数据指针
sfr16 TMR2RL = 0xca; // 定时器 T2 重装值
sfr16 TMR2 = 0xcc; // 定时器 T2 计数器
sfr16 PCA0CP1 = 0xe9; // PCA0 模式 1 捕捉/比较
sfr16 PCA0CP2 = 0xeb; // PCA0 模式 2 捕捉/比较
sfr16 PCA0 = 0xf9; // PCA0 计数器
sfr16 PCA0CP0 = 0xfb; // PCA0 模式 0 捕捉/比较
//-----------------------------------------------------------------------
// 全局变量
//-----------------------------------------------------------------------
#define SYSCLK 24500000 // 系统时钟频率（Hz）
#define BAUDRATE 115200 // UART 波特率（bps）
//-----------------------------------------------------------------------
// 函数原型
//-----------------------------------------------------------------------
void PORT_Init (void); // 端口 I/O 配置
void SYSCLK_Init (void); // 系统时钟初始化
void UART0_Init (void); // UART0 初始化
extern char SPI_Transfer (char); // SPI 传输程序
//-----------------------------------------------------------------------
// 全局变量
//-----------------------------------------------------------------------
//-----------------------------------------------------------------------
// 主程序
//-----------------------------------------------------------------------
void main (void) {
unsigned char test_counter, SPI_return; // 用于测试 SPI 程序
// 禁止 WDT
PCA0MD &= ~0x40; // WDTE = 0 (清除 WDT 使能)
SYSCLK_Init (); // 初始化振荡器
PORT_Init (); // 初始化端口和 GPIO
UART0_Init (); // 初始化 UART0
EA = 1; // 使能全部中断
while (1)
{
for (test_counter = 0; test_counter <= 0xFF; test_counter++)
{
```



```
NSS = 0x00; // 选择 SPI 从器件
SPI_return = SPI_Transfer(test_counter); // 发送/接收 SPI 字节
NSS = 0x01; // 取消选择 SPI 从器件
printf(“\nSPI Out = 0x%02X, SPI In = 0x%02X”, (unsigned)test_counter,
(unsigned)SPI_return);
// 发送 SPI 数据到 UART
// 为校验目的
}
}
}
//-----------------------------------------------------------------------------
// 初始化子程序
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
// PORT_Init
//-----------------------------------------------------------------------------
//
// 配置数据交叉开关和 GPIO 端口
// P0.0 - MOSI (推挽)
// P0.1 - MISO
// P0.2 - SCK (推挽)
// P0.3 - NSS (推挽)
// P0.4 - UART TX (推挽)
// P0.5 - UART RX
// P0.6 -
// P0.7 -
//
void PORT_Init (void)
{
XBR0 = 0x0F; // 在 XBAR 中跳过 SPI 引脚
XBR1 = 0x03; // UART0 TX 和 RX 引脚使能
XBR2 = 0x40; // 使能数据交叉开关和弱上拉
P0MDOUT |= 0x1D; // 允许 TX0, MOSI, SCK 和 NSS 为推挽输出
}
//-----------------------------------------------------------------------------
// SYSCLK_Init
//-----------------------------------------------------------------------------
//
// 此程序初始化系统时钟，用内部 24.5 MHz 时钟
// 作为时钟源
//
void SYSCLK_Init (void)
{
OSCICN = 0x07; // 选择内部振荡器作为系统时钟源
```



```
}
//-----------------------------------------------------------------------------
// UART0_Init
//-----------------------------------------------------------------------------
//
//  使用定时器 T0 作为波特率发生器和 8-N-1 模式
//
void UART0_Init (void)
{
SCON0 = 0x10; // SCON0: 8 位可变速率位
//  忽略停止位
// RX 使能
//  第九位为 0
//  清 RI0 和 TI0 位
if (SYSCLK/BAUDRATE/2/256 < 1)
{
TH1 = -(SYSCLK/BAUDRATE/2);
CKCON &= ~0x13;


CKCON |= 0x10; // T1M = 1; SCA1:0 = xx
}
else if (SYSCLK/BAUDRATE/2/256 < 4)
{
TH1 = -(SYSCLK/BAUDRATE/2/4);
CKCON &= ~0x13;
CKCON |= 0x01; // T1M = 0; SCA1:0 = 01
}
else if (SYSCLK/BAUDRATE/2/256 < 12)
{
TH1 = -(SYSCLK/BAUDRATE/2/12);
CKCON &= ~0x13; // T1M = 0; SCA1:0 = 00
}
else
{
TH1 = -(SYSCLK/BAUDRATE/2/48);
CKCON &= ~0x13;
CKCON |= 0x02; // T1M = 0; SCA1:0 = 10
}
TL1 = 0xff; //  立即设置定时器 T1 溢出
TMOD |= 0x20; // TMOD:  定时器 t1 8 位自动重装
TMOD &= ~0xD0; //  模式
TR1 = 1; //  启动定时器 T1
TI0 = 1; //  表示 TX0 就绪
```



```
}
//-----------------------------------------------------------------------------
// SPI_EE_F30x.c
//-----------------------------------------------------------------------------
// Copyright 2001 Cygnal Integrated Products, Inc.
//
// 作者: BD
// 日期:2001 年 12 月 14 日
//
// 此程序示范 C8051F30X 处理主 SPI 程序集是如何在一个 C 程序中使用的。
//
// 在此例子中, Microchip 的 25LC320 4k X 8 串行 EEPROM 连接到
// 由 C8051F30X 实现的 SPI 主器件
// 用两种测试模式写 EEPROM：
//   1) 所有单元为 0xFF，
//   2) 每个单元用相应地址的 LSB 写
// EEPROM 的内容用测试模式校验。如果测试
// 模式校验无错，则在操作完成后 LED 闪烁。
// 否则, LED 保持为关闭态，可以通过 PC 终端
// 与 UART 连接（传输波特率为 115.2kbps）来监测进程
//
// 为了实现代码功能，下列文件之一应该被编译或汇编，
// 最终生成的目标文件必须与从这些文件产生的目标文件连接：

//
// SPI_MODE0.c 模式 0 主 SPI 器件的‘C’语言实现
// SPI_MODE0.asm 模式 0 主 SPI 器件的汇编语言实现
// SPI_MODE3.c 模式 3 主 SPI 器件的‘C’语言实现
// SPI_MODE3.asm 模式 3 主 SPI 器件的汇编语言实现

//
// EEPROM 的串行口只能用模式 0 和模式 3 SPI 配置操作
//
// 目标器件: C8051F30x
// 连接工具: KEIL C51 6.03 / KEIL EVAL C51
//
//-----------------------------------------------------------------------------
// 包含文件
//-----------------------------------------------------------------------------
#include <c8051f300.h> // SFR 声明
#include <stdio.h> // 标准 I/O
#include “SPI_defs.h” // SPI 端口定义
//-----------------------------------------------------------------------------
// C8051F30X 的 16 位 SFR 定义
```



```
//-----------------------------------------------------------------------------
sfr16 DP = 0x82; //  数据指针
sfr16 TMR2RL = 0xca; //  定时器 T2 重装值
sfr16 TMR2 = 0xcc; //  定时器 T2 计数器
sfr16 PCA0CP1 = 0xe9; // PCA0 模式 1 捕捉/比较
sfr16 PCA0CP2 = 0xeb; // PCA0 模式 2 捕捉/比较
sfr16 PCA0 = 0xf9; // PCA0 计数器
sfr16 PCA0CP0 = 0xfb; // PCA0 模式 0 捕捉/比较
//-----------------------------------------------------------------------------
//  全局变量
//-----------------------------------------------------------------------------
#define SYSCLK 24500000 //  系统时钟频率（Hz）
#define BAUDRATE 115200 // UART 波特率（bps）
#define EE_SIZE 4096 // EEPROM 容量（字节）
#define EE_READ 0x03 // EEPROM 读命令
#define EE_WRITE 0x02 // EEPROM 写命令
#define EE_WRDI 0x04 // EEPROM  写禁止命令
#define EE_WREN 0x06 // EEPROM  写允许命令
#define EE_RDSR 0x05 // EEPROM  读状态寄存器
#define EE_WRSR 0x01 // EEPROM  写状态寄存器
sbit LED = P0^6; // LED  指示器
//-----------------------------------------------------------------------------
//  函数原型
//-----------------------------------------------------------------------------
void PORT_Init (void); //  端口 I/O 配置
void SYSCLK_Init (void); //  系统时钟初始化
void UART0_Init (void); // UART0 初始化
extern char SPI_Transfer (char); // SPI 传输程序
void Timer0_ms (unsigned ms);
void Timer0_us (unsigned us);

unsigned char EE_Read (unsigned Addr);
void EE_Write (unsigned Addr, unsigned char value);
//-----------------------------------------------------------------------------
//  全局变量
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
//  主程序
//-----------------------------------------------------------------------------
void main (void) {
unsigned EE_Addr; // EEPROM 字节地址
unsigned char test_byte;
禁止 WDT
PCA0MD &= ~0x40; // WDTE = 0 (清除 WDT 使能)
```



```
SYSCLK_Init (); // 初始化振荡器
PORT_Init (); // 初始化端口和 GPIO
UART0_Init (); // 初始化 UART0
EA = 1; // 使能全部中断
SCK = 0;
// 用 0xFF 填充 EEPROM
LED = 1;
for (EE_Addr = 0; EE_Addr < EE_SIZE; EE_Addr++)
{
test_byte = 0xff;
EE_Write (EE_Addr, test_byte);
// print status to UART0
if ((EE_Addr % 16) == 0)
{
printf (“\nwriting 0x%04x: %02x “, EE_Addr, (unsigned) test_byte);
}
else
{
printf (“%02x “, (unsigned) test_byte);
}
}
// 用 0xFF 校验 EEPROM
LED = 0;
for (EE_Addr = 0; EE_Addr < EE_SIZE; EE_Addr++)
{
test_byte = EE_Read (EE_Addr);
// 打印状态到 UART0
if ((EE_Addr % 16) == 0)
{

printf (“\nverifying 0x%04x: %02x “, EE_Addr, (unsigned) test_byte);
}
else
{
printf (“%02x “, (unsigned) test_byte);
}
if (test_byte != 0xFF)
{
printf (“Error at %u\n”, EE_Addr);
while (1); // stop here on error
}
}
// 用 EEPROM 地址的 LSB 填充 EEPROM 存储器
LED = 1;
```



```
for (EE_Addr = 0; EE_Addr < EE_SIZE; EE_Addr++)
{
test_byte = EE_Addr & 0xff;
EE_Write (EE_Addr, test_byte);
// 打印状态到 UART0
if ((EE_Addr % 16) == 0)
{
printf (“\nwriting 0x%04x: %02x “, EE_Addr, (unsigned) test_byte);
}
else
{
printf (“%02x “, (unsigned) test_byte);
}
}
// 用 EEPROM 地址的 LSB 校验 EEPROM 存储器
LED = 0;
for (EE_Addr = 0; EE_Addr < EE_SIZE; EE_Addr++)
{
test_byte = EE_Read (EE_Addr);
// 打印状态到 UART0
if ((EE_Addr % 16) == 0)
{
printf (“\nverifying 0x%04x: %02x “, EE_Addr, (unsigned) test_byte);
}
else
{
printf (“%02x “, (unsigned) test_byte);
}
if (test_byte != (EE_Addr & 0xFF))
{
printf (“Error at %u\n”, EE_Addr);
while (1); // stop here on error
}
}
while (1)
{ // 完成后 LED 闪烁
Timer0_ms (100);
LED = ~LED;
  }
}
//-----------------------------------------------------------------------------
子程序
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
```



```
// 初始化子程序
//-----------------------------------------------------------------------
//-----------------------------------------------------------------------
// PORT_Init
//-----------------------------------------------------------------------
//
// 配置数据交叉开关和 GPIO 端口
// P0.0 - MOSI (推挽)
// P0.1 - MISO
// P0.2 - SCK (推挽)
// P0.3 - NSS (推挽)
// P0.4 - UART TX (推挽)
// P0.5 - UART RX
// P0.6 - LED
// P0.7 -
//
void PORT_Init (void)
{
XBR0 = 0x0F; // 在 XBAR 中跳过 SPI 引脚
XBR1 = 0x03; // UART0 TX 和 RX 引脚使能
XBR2 = 0x40; // 使能数据交叉开关和弱上拉
P0MDOUT |= 0x5D; // 允许 TX0, MOSI, SCK 和 NSS 为推挽输出
}
//-----------------------------------------------------------------------
// SYSCLK_Init
//-----------------------------------------------------------------------
//
// 此程序初始化系统时钟，用内部 24.5 MHz 时钟
// 作为时钟源
//
void SYSCLK_Init (void)
{
OSCICN = 0x07; // 选择内部振荡器作为系统时钟源
}
//-----------------------------------------------------------------------
// UART0_Init
//-----------------------------------------------------------------------
//
// 使用定时器 T0 作为波特率发生器和 8-N-1 模式
//
void UART0_Init (void)
{
SCON0 = 0x10; // SCON0: 8 位可变速率位
```



```
// 忽略停止位
// RX 使能
// 第九位为 0
// 清 RI0 和 TI0 位
if (SYSCLK/BAUDRATE/2/256 < 1)
{
TH1 = -(SYSCLK/BAUDRATE/2);
CKCON &= ~0x13;
CKCON |= 0x10; // T1M = 1; SCA1:0 = xx
}
else if (SYSCLK/BAUDRATE/2/256 < 4)
{
TH1 = -(SYSCLK/BAUDRATE/2/4);
CKCON &= ~0x13;
CKCON |= 0x01; // T1M = 0; SCA1:0 = 01
}
else if (SYSCLK/BAUDRATE/2/256 < 12)
{
TH1 = -(SYSCLK/BAUDRATE/2/12);
CKCON &= ~0x13; // T1M = 0; SCA1:0 = 00
}
else
{
TH1 = -(SYSCLK/BAUDRATE/2/48);
CKCON &= ~0x13;
CKCON |= 0x02; // T1M = 0; SCA1:0 = 10
}
TL1 = 0xff; // 立即设置定时器 T1 溢出
TMOD |= 0x20; // TMOD: 定时器 t1 8 位自动重装
TMOD &= ~0xD0; // 模式
TR1 = 1; // 启动定时器 T1
TI0 = 1; // 表示 TX0 就绪
}
//-----------------------------------------------------------------------------
// Timer0_ms
//-----------------------------------------------------------------------------
//
// 配置定时器 T0 在返回前延时<ms>毫秒
//
void Timer0_ms (unsigned ms)
{
unsigned i; // 毫秒计数器
TCON &= ~0x30; // 停止定时器 T0 并清除溢出标志
TMOD &= ~0x0f; // 配置定时器 T0 为 16 位模式
```



```
TMOD |= 0x01;
CKCON |= 0x08; //  定时器 T0 计数系统时钟
for (i = 0; i < ms; i++) //  计数毫秒
{
TR0 = 0; //  停止定时器 T0
TH0 = (-SYSCLK/1000) >> 8; //  设置定时器 T0 在 1ms 溢出
TL0 = -SYSCLK/1000;
TR0 = 1; //  启动定时器 T0
while (TF0 == 0); //  等待溢出

TF0 = 0; //  清除溢出标志
}
}
//-----------------------------------------------------------------------
// Timer0_us
//-----------------------------------------------------------------------
//
//  配置定时器 T0 在返回前延时<ms>微秒
//
void Timer0_us (unsigned us)
{
unsigned i; //  微秒计数器
TCON &= ~0x30; //  停止定时器 T0 并清除溢出标志
TMOD &= ~0x0f; //  配置定时器 T0 为 16 位模式
TMOD |= 0x01;
CKCON |= 0x08; //  定时器 T0 计数系统时钟
for (i = 0; i < us; i++) { //  计数微秒
TR0 = 0; //  停止定时器 T0
TH0 = (-SYSCLK/1000000) >> 8; //  设置定时器 T0 在 1us 溢出
TL0 = -SYSCLK/1000000;
TR0 = 1; //  启动定时器 T0
while (TF0 == 0); //  等待溢出
TF0 = 0; //  清除溢出标志
}
}
//-----------------------------------------------------------------------
// EE_Read
//-----------------------------------------------------------------------
//
//  此程序读和返回一个 EEPROM 字节，
//  字节地址由<Addr>给出
//
unsigned char EE_Read (unsigned Addr)
{
```



```
unsigned char retval; // 返回值
NSS = 0; // 选择 EEPROM
Timer0_us (1); // 至少等待 250ns (CS 建立时间)
// 传送读操作码
retval = SPI_Transfer(EE_READ);
// 传送地址 MSB 在先
retval = SPI_Transfer((Addr & 0xFF00) >> 8); // 传送地址的 MSB
retval = SPI_Transfer((Addr & 0x00FF)); // 传送地址的 LSB
// 初始化哑元传送读数据
retval = SPI_Transfer(0x00);
Timer0_us (1); // 至少等待 250ns (CS 保持时间)

NSS = 1; // 取消选择 EEPROM
Timer0_us (1); // 至少等待 500ns (CS 禁止时间)
return retval;
}
//-----------------------------------------------------------------------------
// EE_Write
//-----------------------------------------------------------------------------
// 此程序写一个 EEPROM 字节<value> ，字节地址由<Addr>给出
//
void EE_Write (unsigned Addr, unsigned char value)
{
unsigned char retval; // 从 SPI 返回值
NSS = 0; // 选择 EEPROM
Timer0_us (1); // 至少等待 250ns (CS 建立时间)
// 传送 WREN（写使能）操作码
retval = SPI_Transfer(EE_WREN);
Timer0_us (1); // 至少等待 250ns (CS 保持时间)
NSS = 1; // 取消选择 EEPROM 到设置 WREN 锁存
Timer0_us (1); //至少等待 500ns (CS 禁止时间)
NSS = 0; // 选择 EEPROM
Timer0_us (1); // 至少等待 250ns (CS 建立时间)
// 传送 WRITE 操作码
retval = SPI_Transfer(EE_WRITE);
// 传送地址 MSB 在先
retval = SPI_Transfer((Addr & 0xFF00) >> 8); // 传送地址的 MSB
retval = SPI_Transfer((Addr & 0x00FF)); // 传送地址的 LSB
// 传送数据
retval = SPI_Transfer(value);
Timer0_us (1); // 至少等待 250ns (CS 保持时间)
NSS = 1; // 取消选定 EEPROM (初始化 EEPROM 写周期）
// 为了完成写操作，现在查询读状态寄存器(RDSR)
do {
```



```
Timer0_us (1); //  至少等待 500ns (CS  禁止时间)
NSS = 0; //  选择 EEPROM 开始查询
Timer0_us (1); //  至少等待 250ns (CS  建立时间)
retval = SPI_Transfer(EE_RDSR);
retval = SPI_Transfer(0x00);
Timer0_us (1); //  至少等待 250ns (CS  保持时间)
NSS = 1; //  取消选择 EEPROM
} while (retval & 0x01); //  查询直到 WIP (Write In Progress)  位为‘0’
Timer0_us (1); //  至少等待 500ns (CS 禁止时间)
}
```

