AN005 — 通过 JTAG 接口对 FLASH 编程

相关器件

本应用笔记适用于下列器件：

C8051F000、C8051F001、C8051F002、C8051F005、C8051F006、C8051F010、C8051F011 和 C8051F012。

引言

本文介绍如何通过 JTAG 接口对 C8051 器件的 FLASH 存储器编程。在本应用笔记的最后提供了示例源代码。

通过 JTAG 接口对 FLASH 编程所需要的信息可以分为三大类：

1. JTAG 接口信息：
   a. 4 脚物理层接口（TCK、TMS、TDI 和 TDO）
   b. 测试访问端口（TAP）状态机
   c. TAP 复位、指令寄存器扫描和数据寄存器扫描基本操作
2. JTAG 间接寄存器操作：
   a. 读间接寄存器
   b. 写间接寄存器
   c. 查询“忙”标志位看读或写操作是否完成
3. FLASH 编程操作：
   a. 读一个 FLASH 字节
   b. 写一个 FLASH 字节
   c. 擦除一个 FLASH 页
   d. 擦除整个 FLASH

图 1 给出通过 JTAG 端口访问 FLASH 的编程层次结构。

图 1. JTAG 闪存编程层次结构

JTAG 接口

本应用笔记为 FLASH 编程提供了足够的 JTAG 接口信息。若需要更多的信息，请参见 JTAG

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
电话：（024）23930366，23940230
电邮：longhua@mail.sy.ln.cn
网址：www.xhl.com.cn

标准 — IEEE 1149.1-1990，该标准可以从“电气与电子工程师协会”得到（更多的信息见 http://standards.ieee.org）。C8051 系列器件的 JTAG 接口完全符合 IEEE 1149.1 规范。已经熟悉 JTAG 接口的读者可跳到第 7 页的“C8051 器件指令寄存器”一节。

测试访问口接口（TAP）

JTAG 口的硬件接口包出括四个信号，如图 2 所示：

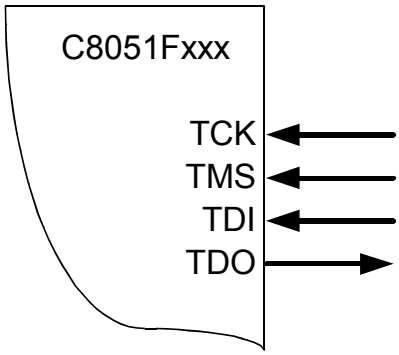

图 2. TAP 接口

1. **TCK**：输入移位时钟。TMS 和 TDI 的数据在 TCK 的上升沿被采样。数据在时钟的下降沿输出到 TDO。
2. **TMS**：输入方式选择。TMS 用于控制 TAP 状态机。
3. **TDI**：输入。输入到指令寄存器（IR）或数据寄存器（DR）的数据出现在 TDI 输入端，在 TCK 的上升沿被采样。
4. **TDO**：输出。来自指令寄存器或数据寄存器的数据在时钟的下降沿被移出到 TDO。

TAP 状态机

如图 3 所示，测试访问口状态机的主要目的是选择指令寄存器或数据寄存器中的一个，使其连接到 TDI 和 TDO 之间。一般来说，指令寄存器用于选择要扫描的数据寄存器。在状态机框图中，位于箭头旁边的数字表示 TCK 变高时 TMS 的逻辑状态。

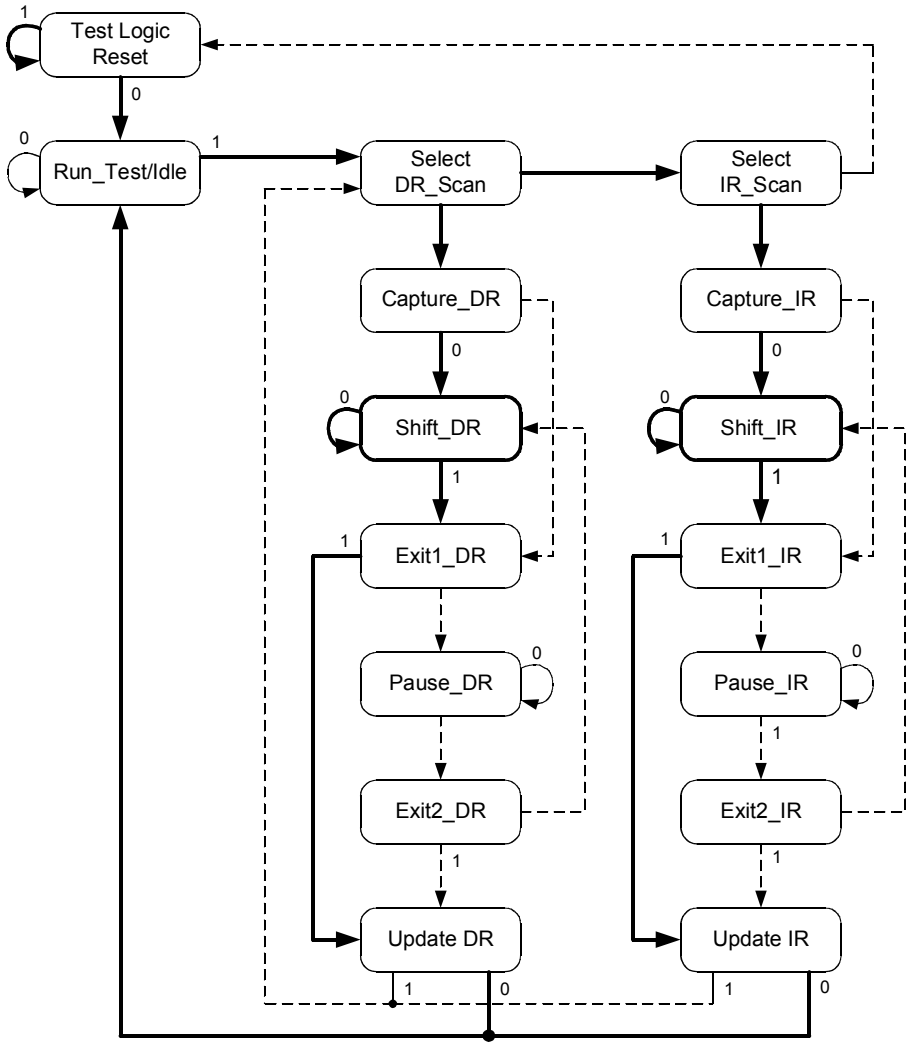

图 3. TAP 状态机

TAP 复位

通过保持 TMS 为高电平（逻辑‘1’）并在 TCK 端输入至少 5 个选通脉冲（变高后再变低）后 TAP 逻辑被复位，如图 4 所示。这使 TAP 状态机的状态从任何其它状态转到测试逻辑复位状态，对 JTAG 口和测试逻辑复位。该状态不复位 CPU 和外设。

TAP 注意事项：

1. **在进入 Shift_DR 或 Shift_IR 状态时，TDO 上的数据从 TCK 的下降沿开始有效。**
2. **在进入 Shift_DR 或 Shift_IR 状态时数据不移位。**
3. **在离开 Shift_DR 或 Shift_IR 时数据被移位。**
4. **最先移出的是数据的最低位（LSB）。**

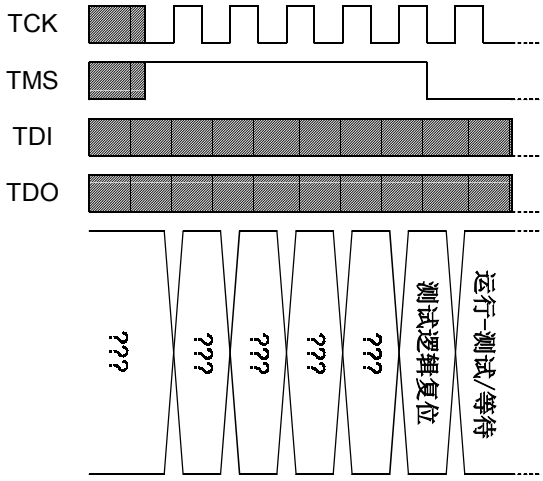

图 4. TAP 复位时序

IR 和 DR 扫描

除了测试逻辑复位之外，状态机还控制两个基本操作：指令寄存器（IR）扫描和数据寄存器（DR）扫描。在一次扫描操作中，出现在 TDI 的数据在 TCK 的上升沿被采样，在 TCK 的下降沿数据被输出到 TDO。在一次指令寄存器扫描操作中，指令寄存器在 Shift_IR 状态被传送。在一次数据寄存器扫描操作中，数据寄存器在 Shift_DR 状态被传送。数据移位时总是 LSB 在先。

在 C8051 器件中，指令寄存器的长度总是 16 位。数据寄存器的长度是变化的，取决于所选择的寄存器。图 5 给出了指令寄存器访问的时序图，图 6 给出了数据寄存器访问的时序图。

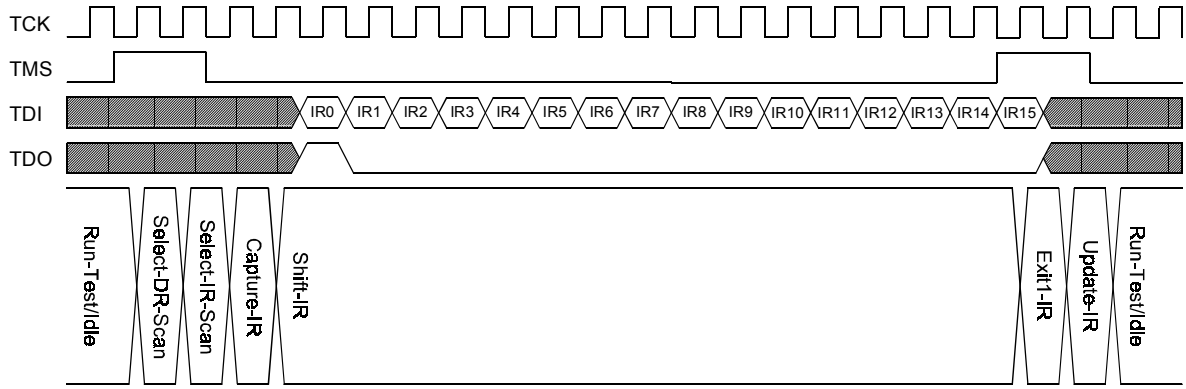

图 5. 指令寄存器访问时序图

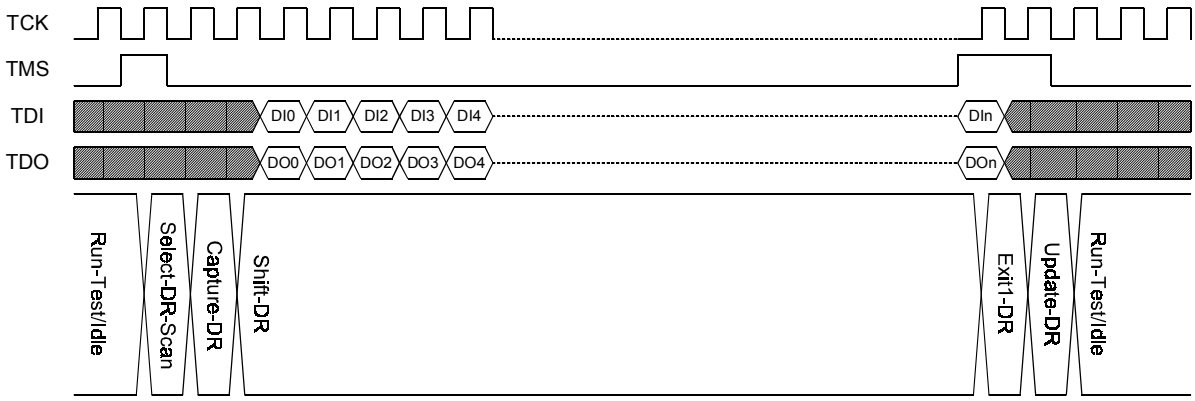

图 6. 数据寄存器访问时序图

IDCODE 举例

为了更好地说明一个典型的 JTAG 操作是如何工作的，我们举一个读 IDCODE 寄存器的例子。

读 IDCODE 是一个两步的过程。首先启动一个指令寄存器扫描操作，将 IDCODE 的地址装入指令寄存器，从 TDI 移入 16 位，如图 7 所示。一旦指令寄存器装入完成，则启动数据寄存器扫描操作，从器件中读出 32 位的 IDCODE 输出到 TDO，如图 8 所示。

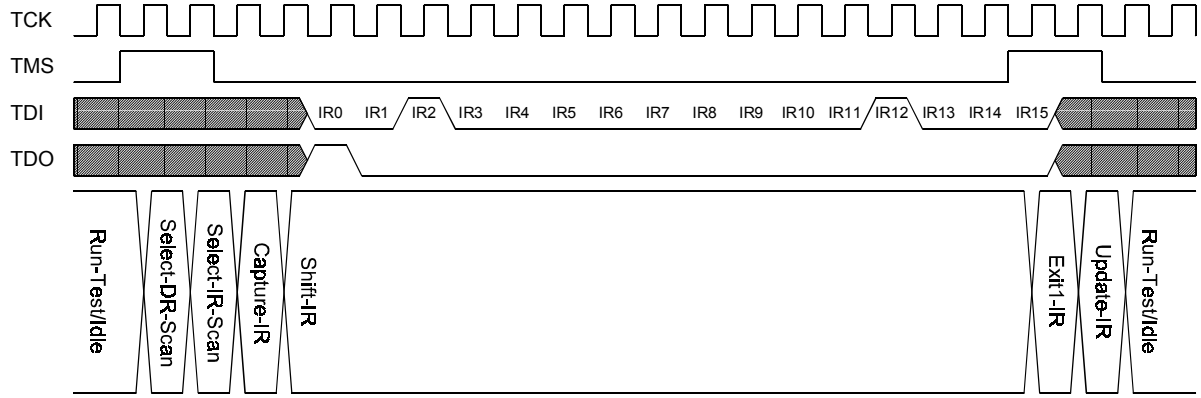

图 7. 读 IDCODE 的指令寄存器扫描时序

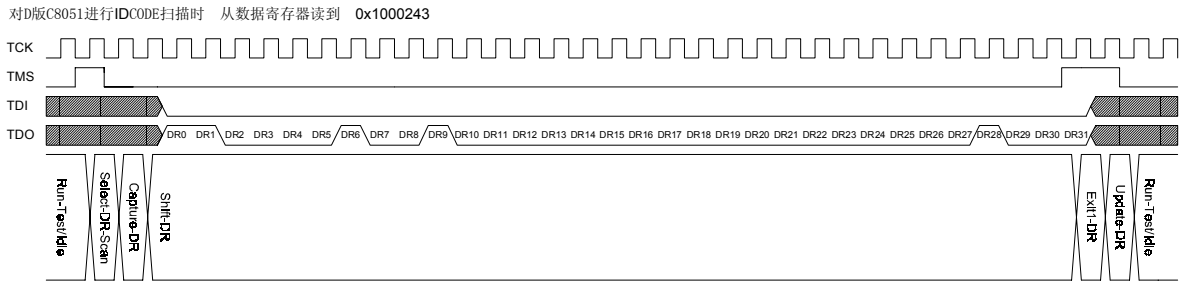

图 8. 读 IDCODE 的数据寄存器扫描时序

C8051 器件的指令寄存器

C8051 器件中的指令寄存器的长度总是 16 位，指令格式如下：

指令寄存器格式

15:12	11:0
StateCntl	**DRAddress**

StateCntl 字段控制调试硬件的状态。在一次 FLASH 编程操作中，系统首先停止运行，CPU 内核被保持在暂停执行状态，使看门狗定时器不起作用。

StateCntl 字段译码

StateCntl*	器件状态
0000	正常
0001	停止
0010	系统复位
0100	CPU 内核挂起
1111	**正常**
***未列出的状态为保留状态**	

DRAddress 字段译码

寄存器	DRAddress*
EXTEST	0x000
SAMPLE/PRELOAD	0x002
IDCODE	0x004
BYPASS	0xFFF
FLASHCON	**0x082**
FLASHDAT	**0x083**
FLASHADR	**0x084**
FLASHSCL	**0x085**
***未列出的状态为保留状态**	

间接寄存器

4 个 FLASH 寄存器（FLASHCON、FLASHADR、FLASHDAT 和 FLASHSCL）都是采用相同的间接方式进行访问的。这种间接访问机制处理 JTAG 时钟域（受 TCK 控制）和 CPU 时钟域（受 SYSCLK 控制）之间的信息传送。不应将这些间接寄存器与标准 8051 的间址寄存器 R0 和 R1 相混淆。

间接寄存器访问概述

为了读或写一个间接寄存器，指令寄存器中必须装入正确的数据寄存器地址（DRAddress）。

然后通过向所选择的数据寄存器写入适当的间接操作码（IndOpCode）来启动读和写操作。在进行写操作时，‘写’操作码位于待写数据之后。

对于输入命令，数据寄存器的格式如下：

间接写数据寄存器格式

19:18	17:0
IndOpCode	**WriteData（待写数据）**

间接操作码的各位译码如下：

IndOpCode 字段译码

IndOpCode	操作
0x	查询
10	读
11	写

数据寄存器输出数据的格式如下：

间接读数据寄存器格式

19	18:1	0
0	读出的数据	**Busy（忙）**

间接读

读（Read）操作启动一次从由 DRAddress 选择的寄存器中读取数据的过程。读过程可以通过向间接寄存器移入两位来启动（‘读’的 IndOpCode 位）。在读操作被启动后，可以通过查询 Busy 位来确定操作何时完成和何时可以读取数据。图 9 给出了描述如何对一个间接寄存器进行读操作的流程图。

间接写

写操作启动一次向由 DRAddress 选择的寄存器内写数据的过程。可以写长度不大于 18 位的任意长度的寄存器。如果待写寄存器的长度小于 18 位，WriteData（写数据）应左对齐（MSB 占据位 17）。这样允许较短的寄存器可以用较少的 JTAG 时钟周期写入。例如，写一个 8 位的间接寄存器只需移 10 位（2 位‘写’操作码+8 个数据位）即可完成。在启动一个写操作之后，应查询‘Busy’位来确定该操作何时完成。图 10 给出了描述如何对一个间接寄存器进行写操作的流程图。

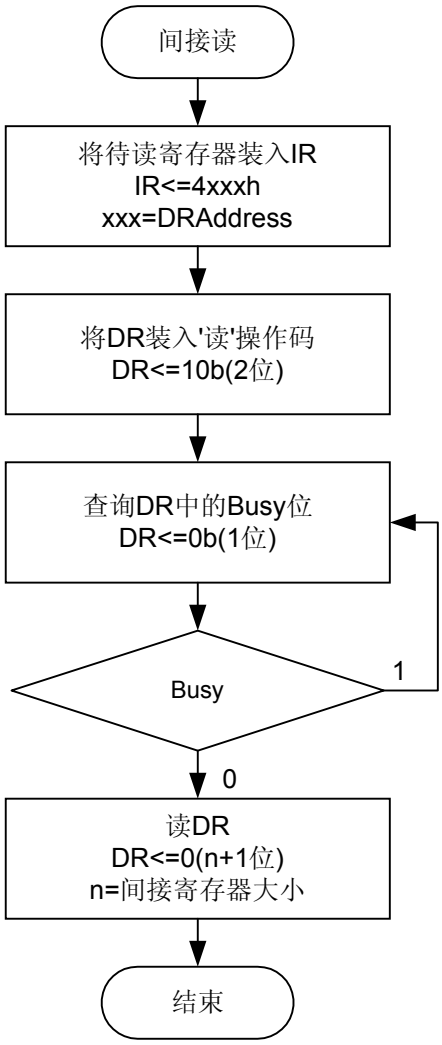

图 9. 间接读流程图

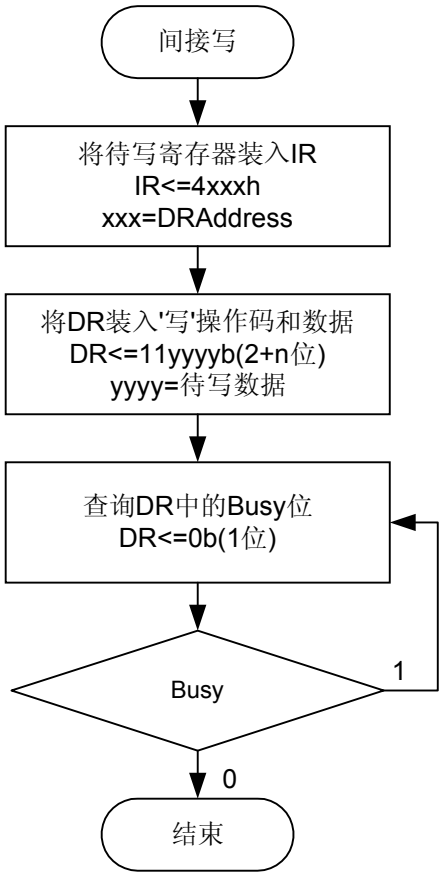

图 10. 间接写流程图

查询‘Busy’位

‘Busy’位用于指示当前的读或写操作是否完成。它在操作启动后变高（‘1’），在操作完成后回到低电平（‘0’）。由于‘Busy’位占据返回数据的 LSB，因此对‘Busy’位的查询可以在一个 DR 移位周期内完成（在退出 Shift_DR 状态时）。

在进行间接读时，一旦‘Busy’位变低，ReadData 即可被移出。注意，ReadData 总是右对齐的。这就允许长度小于 18 位的寄存器可以用较少的 JTAG 时钟周期读取。例如：一个 8 位的读操作可以用 9 个 DR 移位（8 个数据位+1 个‘Busy’位）完成。

图 11 给出了查询‘Busy’位的数据寄存器扫描时序图。

当一个读或写操作正在进行时，不应改变指令寄存器的内容。

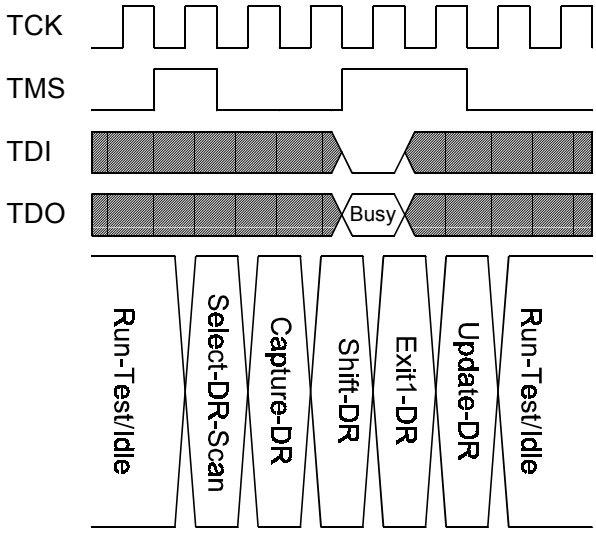

图 11. 查询‘Busy’位的数据寄存器扫描时序图

FLASH 编程

FLASH 寄存器说明

通过 4 个间接寄存器访问 FLASH：FLASHCON、FLASHADR、FLASHDAT 和 FLASHSCL。用前节所述的间接读或间接写来对每一个寄存器访问。

FLASHCON

FLASHCON 是一个 8 位寄存器，它控制 FLASH 逻辑如何响应对 FLASHDAT 寄存器的读和写操作。FLASHCON 寄存器包含一个读方式（ReadMode）设置和一个写方式（WriteMode）设置，意义如下：

FLASHCON 格式

7:4	3:0
WriteMode	**ReadMode**

ReadMode 字段译码

ReadMode*	操作
0000	FLBusy 查询
0010	启动 FLASH 读； FLASHADR 加 1
***未列出的状态为保留状态**	

WriteMode 字段译码

WriteMode*	操作
0000	FLBusy 查询
0001	启动 FLASH 写； FLASHADR 加 1
0010	如果 FLASHDAT = 0xA5，则启动对当前页的页擦除操作； 如果 FLASHDAT = 0xA5，且 FLASHADR 为 0x7DFE 或 0x7DFF，则启动对整个 FLASH 的擦除操作；
*未列出的状态为保留状态	

FLASHADR

FLASHADR 是一个 16 位寄存器，它包含待读或待写的 FLASH 字节的地址。FLASHADR 在完成一个读或写操作后自动加一。

FLASHDAT

FLASHDAT 是一个 10 位的寄存器，它包含 8 位数据，一个 FLFail 位和一个 FLBusy 位，如下所示：

FLASHDAT 读格式

9:2	1	0
FLData	**FLFail**	**FLBusy**

写 FLASHDAT 只需要 8 位，因为最后一个被锁存的位处于 MSB 位置。

读 FLASHDAT 需要 11 个 *DR_SHIFT* 周期（8 个用于 FLData，一个用于 FLFail，一个用于 FLBusy，一个用于 Busy）。

查询 FLBusy 至少需要 2 个 *DR_SHIFT* 周期，一个用于 FLBusy，一个用于 Busy。

FLASHSCL

FLASHSCL 是一个 8 位寄存器，用它设置 FLASH 操作时序所需要的预分频值。当使用内部的 2MHz 系统时钟时，该寄存器应配置如下：

FLASHSCL 配置

7:4	3:0
1000	**0110**

FLASH 访问流程

在对 FLASH 编程之前，需要对器件复位并禁止看门狗。否则看门狗定时器可能在 FLASH 操作期间启动系统复位，导致预想不到的后果。

禁止看门狗定时器（WDT）

图 12 给出了禁止看门狗定时器的流程图。该过程描述如下：

1. 通过向指令寄存器（IR）装入 0x2FFF 使系统复位。
2. 通过向 IR 装入 0x1004 进行 IDCODE 扫描，然后用 0x00000000 进行 32 位的 DR 扫描。
3. 在 IR 地址后设置 StateCntl 为 ‘0x04’，使内核挂起，FLASH 离线。

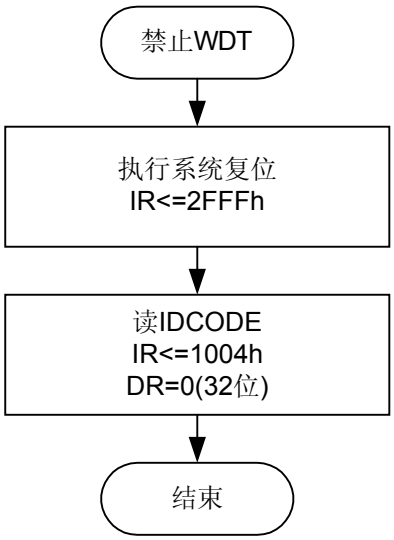

图 12. 旁路看门狗定时器的流程图

读一个 FLSH 字节

图 13 给出了读一个 FLASH 字节的流程图。该过程描述如下：

1. 向 FLASHSCL 装入 0x86，设置正确的 FLASH 时序，使用内部的 2MHz 系统时钟。这可以通过对 FLASHSCL 进行间接写来完成。
2. 向 FLASHADR 装入待读字节的 16 位地址，这可以通过对 FLASHADR 进行 16 位的间接写来完成。
3. 向 FLASHCON 装入启动读操作的操作码（0x01），这可以通过对 FLASHCON 进行 8 位的间接写来完成。
4. 通过读 FLASHDAT 启动读操作。这是一个 0 位的间接读（DR 扫描只包含 2 位读操作码）。注意这只是启动读过程。
5. 向 FLASHCON 装入查询 FLBusy 的操作码（0x00），这是一个对 FLASHCON 的 8 位间接写操作。
6. 查询 FLBusy，直到它变为低电平，表示读操作已经完成。这是一个一位的间接读。图 14 给出查询 FLBusy 的 DR 扫描时序图。

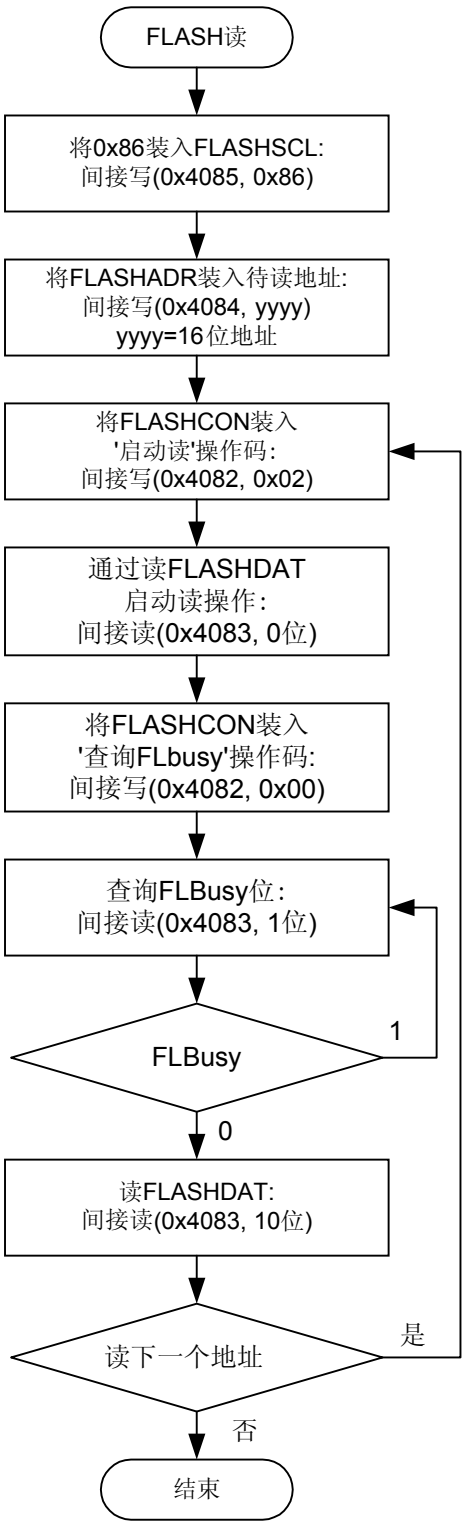

图 13. 读一个 FLASH 字节的流程图

7. 读 FLASHDAT。这是一个 10 位的间接读（8 个数据位，一个 FLFail 位和一个 FLBusy 位）。图 15 给出读 FLASHDAT 的 DR 扫描时序图。

如果进行连续读，则该过程可以从步骤 3 重新开始，因为 FLASHADR 在进行一次读或写操作后自动加 1。

如果试图对一个已被设置为读锁定的扇区进行读操作，则 FLFail 位置‘1’。

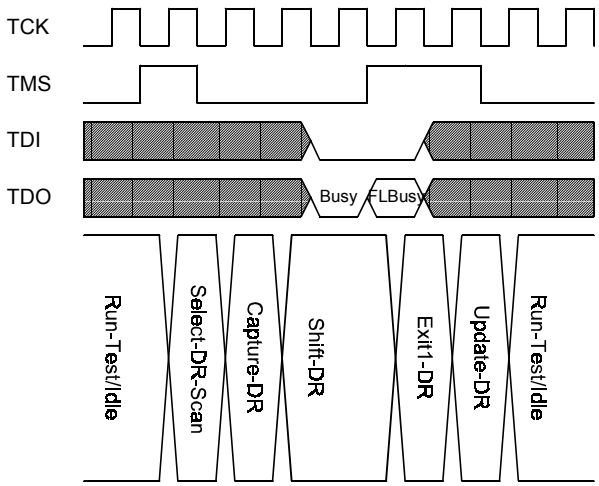

图 14. 查询 FLBusy 的 DR 扫描时序图

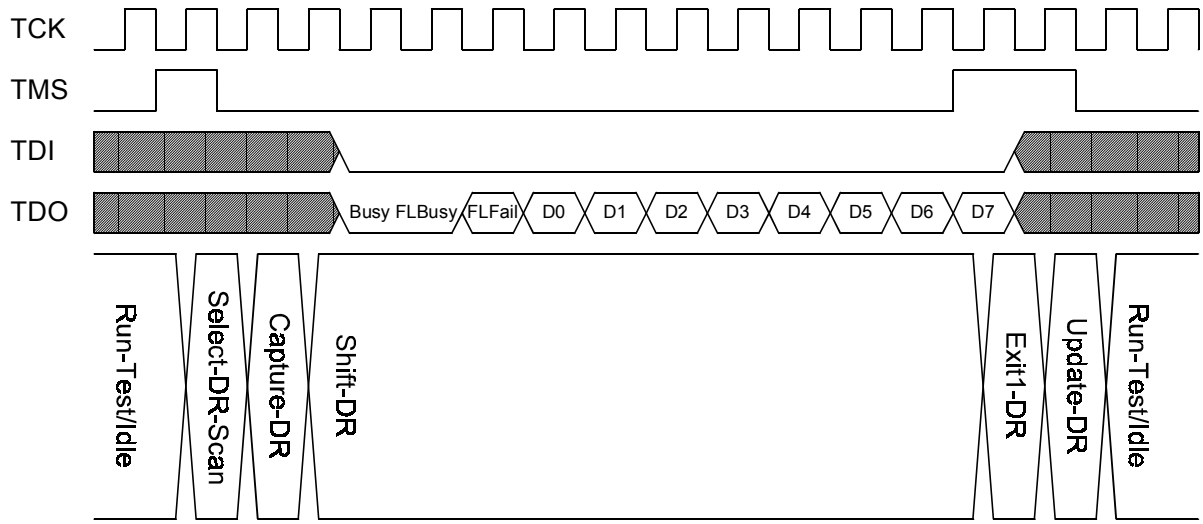

图 15 读 FLASHDAT 的 DR 扫描时序图

写一个 FLSH 字节

图 16 给出了写一个 FLASH 字节的流程图。该过程如下：

1. 向 FLASHSCL 装入 0x86，设置正确的 FLASH 时序，使用内部的 2MHz 系统时钟。这可以通过对 FLASHSCL 进行间接写来完成。
2. 向 FLASHADR 装入待写字节的 16 位地址。
3. 向 FLASHCON 装入启动写操作的操作码（0x10）。
4. 向 FLASHDAT 装入待写数据。这是一个 8 位的间接写。
5. 向 FLASHCON 装入查询 FLBusy 的操作码（0x00）。
6. 查询 FLBusy。这是通过启动一个对 FLASHDAT 寄存器的一位间接读来完成的。

如果进行连续写，则该过程可以从步骤 3 重新开始。FLASHADR 在进行一次读或写操作后自动加 1。

如果试图对一个已被设置为写锁定的扇区进行写操作，则 FLFail 位置‘1’。

图 17 给出了步骤 4 以前的写 FLASHDAT 的 DR 扫描时序图。

擦除一个 FLASH 页

FLASH 存储器是由一系列的页组成的，每页 512 字节。擦除一个 FLASH 页的过程与写一个 FLASH 字节类似，区别在于 FLASHCON 寄存器应设置为 0x20，FLASHDAT 需设置为 0xA5。FLASHADR 可以设置为要擦除的页内的任何一个地址。如果 FLASHADR 被设置为锁定字节地址中的任何一个（0x7dfe 或 0x7dff），则该擦除操作将擦除整个 FLASH 存储器，位于 0x7e00 和 0x7fff 之间的保留区除外。

与读和写操作不同，在擦除操作完成后 FLASHADR 并不自动加 1。

图 18 给出了 FLASH 页擦除过程的流程图。

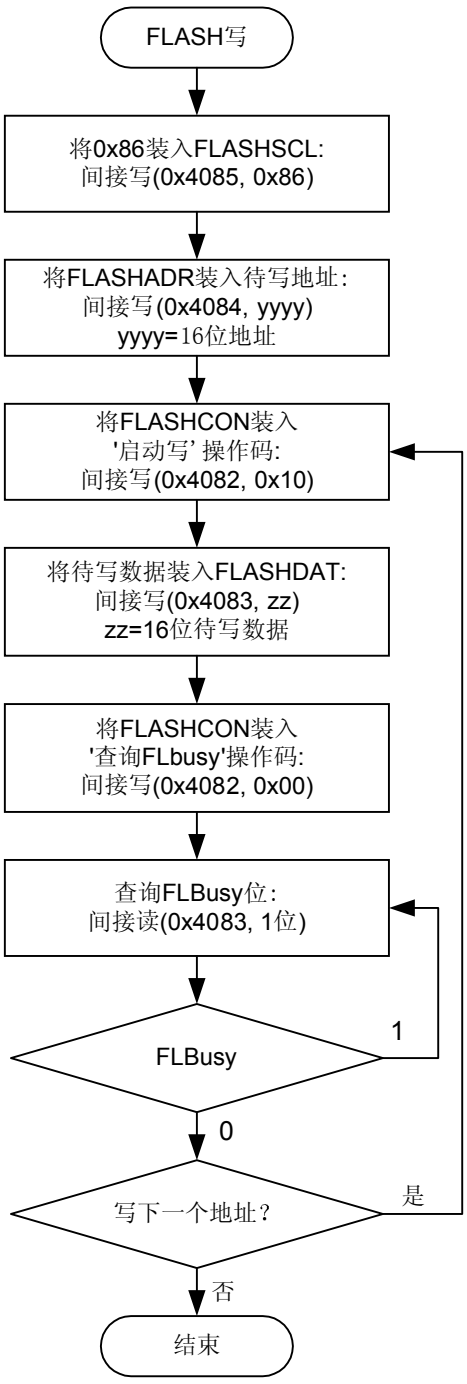

图 16. 写一个 FLASH 字节的流程图

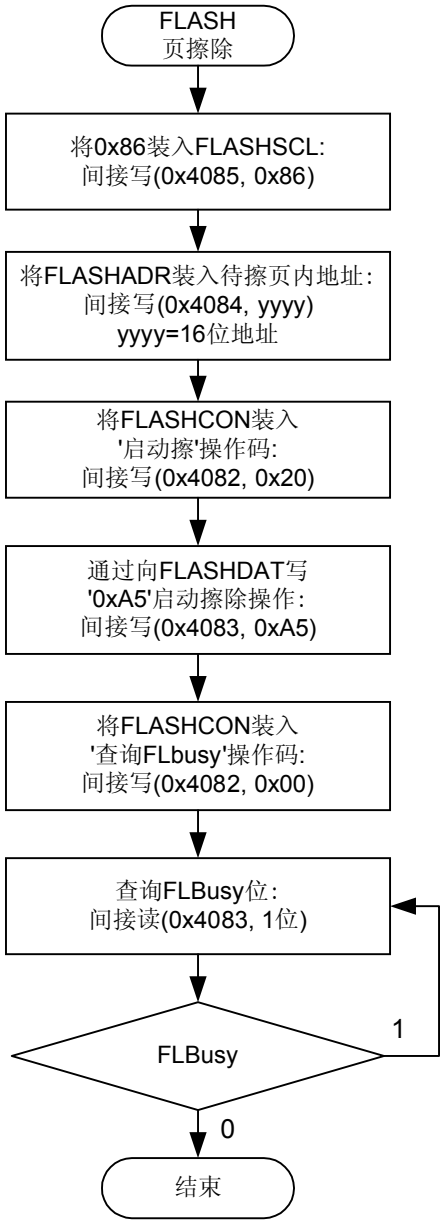

图 18. 擦除一个 FLASH 页的流程图

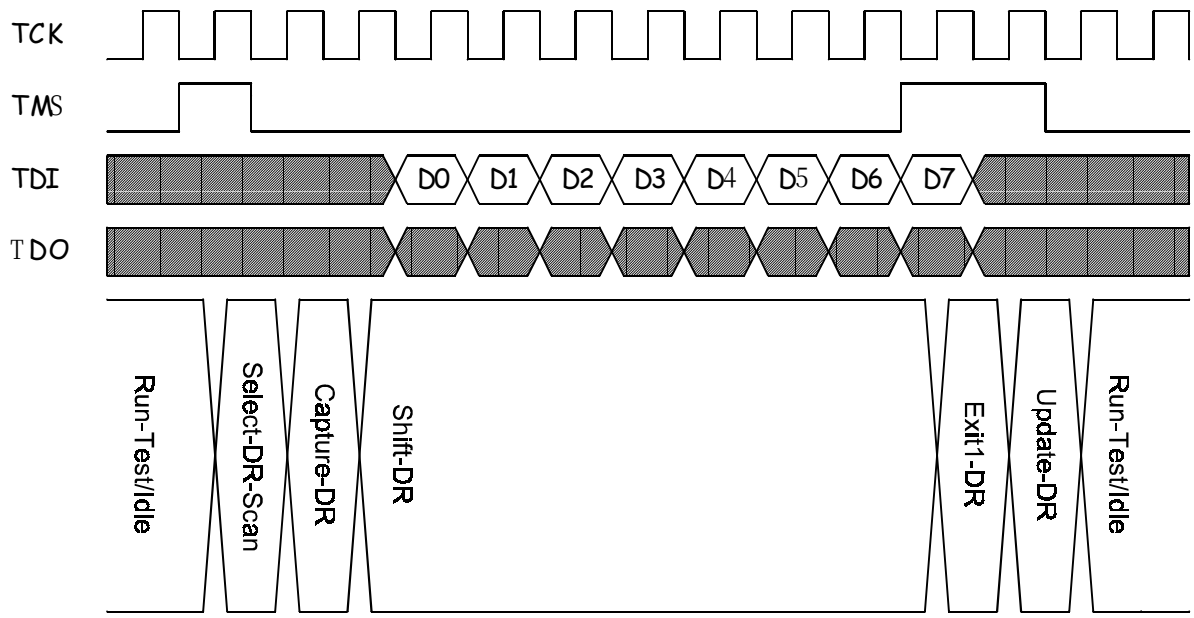

图 17. 写 FLASHDAT 的 DR 扫描时序图。

对 JTAG 链中的器件编程

如果一个 C8051 器件与其它器件一起组成了一个边界扫描链，或多个 C8051 器件的 JTAG 口如图 19 所示那样连接在一起，则除了正在被编程的器件外，每个指令寄存器扫描操作都被配置为将所有其它器件置于 BYPASS（旁路）方式。这可以通过向待编程器件之前或之后的所有器件的指令寄存器中移入‘1’来完成。

数据寄存器扫描操作向待编程器件之前或之后的器件各插入一位以累计器件中的 BYPASS 寄存器的个数。

1. IR 扫描操作前面有 m 个‘1’，后面有 n 个‘1’，其中 m 是待编程器件之前的指令寄存器位数，n 是待编程器件之后的指令寄存器位数。
2. DR 扫描操作中，前面有 x 个‘0’，后面有 y 个‘0’，其中 x 是待编程器件之前的 JTAG 器件的个数，y 是待编程器件之后的 JTAG 器件的个数。

图 20 给出了 IR 扫描操作和 DR 扫描操作的示例。

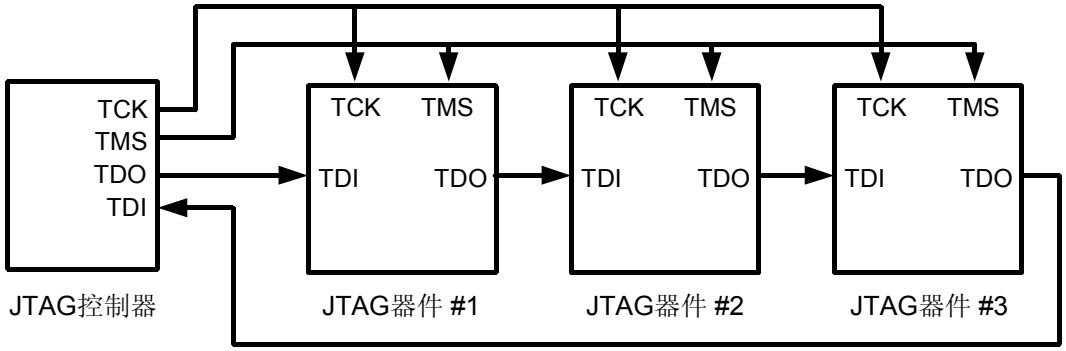

图 19. 典型 JTAG 链连接

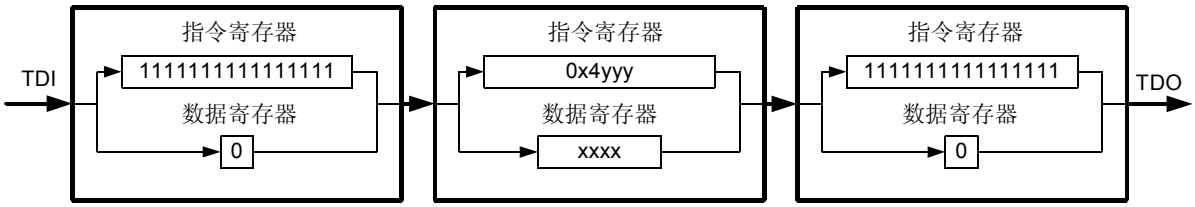

图 20. 隔离待编程的 C8051 器件

软件示例

```
//-----------------------------------------------------------------------
// JTAG_FLASH.c
//-----------------------------------------------------------------------
// 该程序包含一些通过 C8051Fxxx 器件测试方式下的 JTAG 口对 FLASH 进行读、写、擦除
// 操作的基本例程。测试方式的 JTAG 引脚接到 C8051F000 主器件的端口引脚。
//
// 目标器件:    C8051F000，C8051F010
//
// 工具链:      KEIL Eval 'c'
//

//-----------------------------------------------------------------------
// 包含文件
//-----------------------------------------------------------------------
#include <c8051f000.h>                  // SFR 声明

//-----------------------------------------------------------------------
// 全局常量
//-----------------------------------------------------------------------
sbit   LED = P1^6;                      // 绿色 LED: '1' = 亮，'0' = 灭

// 连接到待编程器件 JTAG 引脚的通用 I/O 引脚
sbit   TCK = P3^7;                      // JTAG 测试时钟
sbit   TMS = P3^6;                      // JTAG 方式选择
sbit   TDI = P3^5;                      // JTAG 数据输入
sbit   TDO = P3^4;                      // JTAG 数据输出

#define   TRUE 1
#define   FALSE 0

// JTAG 指令寄存器地址
#define   INST_LENGTH 16                // 指令寄存器位数
#define   BYPASS      0xffff
#define   EXTEST      0x0000
#define   SAMPLE      0x0002

#define   RESET       0x2fff            // 系统复位指令

#define   IDCODE      0x1004            // IDCODE 指令地址/HALT
#define   IDCODE_LEN  32                // ID 码的位数

#define   FLASHCON    0x4082            // FLASH 控制指令地址
#define   FLCN_LEN      8               // FLASHCON 的位数

#define   FLASHDAT      0x4083          // FLASH 数据指令地址
#define   FLD_RDLEN     10              // FLASHDAT 读的位数
#define   FLD_WRLEN     8               // FLASHDAT 写的位数

#define   FLASHADR      0x4084          // FLASH 地址指令地址
```

```
#define   FLA_LEN      16                  // FLASHADR 的位数

#define   FLASHSCL     0x4085              // FLASH 预分频指令地址
#define   FLSC_LEN     8                   // FLASHSCL 的位数

//-------------------------------------------------------------------------
// 函数原型
//-------------------------------------------------------------------------

void init (void);
void JTAG_StrobeTCK (void);
void JTAG_Reset (void);
unsigned int JTAG_IR_Scan (unsigned int instruction, int num_bits);
unsigned long JTAG_DR_Scan (unsigned long dat, int num_bits);
void JTAG_IWrite (unsigned int ireg, unsigned long dat, int num_bits);
unsigned long JTAG_IRead (unsigned int ireg, int num_bits);
int FLASH_ByteRead (unsigned int addr, unsigned char *pdat);
int FLASH_ByteWrite (unsigned int addr, unsigned char dat);
int FLASH_PageErase (unsigned int addr);

//-------------------------------------------------------------------------
---------------
// 主程序

void main (void) {

   unsigned long id;
   unsigned char dest;
   int pass;

   id = 0x12345678L;

   init ();                                 // 初始化端口

   JTAG_Reset ();                           // 复位测试方式下的 JTAG 状态机
   JTAG_IR_Scan (RESET, INST_LENGTH);       // 复位 DUT

   JTAG_IR_Scan (IDCODE, INST_LENGTH);      // 向 IR 装入 IDCODE 并停止 DUT
   id = JTAG_DR_Scan (0x0L, IDCODE_LEN);    // 读 DCODE
                                           // IDCODE 应 = 0x10000243,
                                           // 对 8051F000 Rev D

   // 我们擦除 FLASH 页 0x1000 - 0x11ff，读 0x1000（内容为 0xff），
   // 写 0x66 到 0x1000，重读 0x1000（其值应变为 0x66）。
   while (1) {
      pass = FLASH_PageErase (0x7c00);      // 在写之前先擦除页
      while (!pass);                        // 处理写锁定的情况

      dest = 0x5a;                          // 设置测试变量为非 0xff 的值

      pass = FLASH_ByteRead (0x7c00, &dest); // dest 应返回 0xff
      while (!pass);                        // 处理读锁定的情况
```



```
      dest = 0x66;
      pass = FLASH_ByteWrite (0x7c00, dest); // 写 0x66 到 0x1000
      while (!pass);                         //处理读锁定的情况

      pass = FLASH_ByteRead (0x7c00, &dest);   // dest 应返回 0x66
      while (!pass);                         //处理读锁定的情况

      pass = FLASH_PageErase (0x7c00);
      while (!pass);

      pass = FLASH_ByteRead (0x7c00, &dest);
      while (!pass);
   }
}

//-----------------------------------------------------------------------
// 函数和过程
//-----------------------------------------------------------------------

//-----------------------------------------------------------------------
// init
//-----------------------------------------------------------------------
// 该函数禁止看门狗定时器并初始化通用 I/O 引脚
//
void init (void) {

   WDTCN = 0xde;                          // 禁止看门狗定时器
   WDTCN = 0xad;

   XBR2 |= 0x40;                          // 允许交叉开关
   PRT1CF |= 0x40;                        // 允许 P1.6 (LED) 上拉输出
   PRT3CF |= 0xe0;                        // 设置 P3.7-5 为推挽输出
   P3 &= 0x1f;                            // TCK、TMS 和 TDI 全为低电平

}

//-----------------------------------------------------------------------
// JTAG_StrobeTCK
//-----------------------------------------------------------------------
// 该函数向目标系统的 TCK 引脚发出选通脉冲（置高电平后再变为低电平）。
//
void JTAG_StrobeTCK (void) {

   TCK = 1;
   TCK = 0;
}

//-----------------------------------------------------------------------
// JTAG_Reset
//-----------------------------------------------------------------------
// 该函数将目标系统的 JTAG 状态机置于测试逻辑复位状态，通过在保持 TMS 高电平时向 TCK
// 发出 5 个选通脉冲来实现。让 JTAG 状态机保持在 Run_Test/Idle 状态。
//
void JTAG_Reset (void) {
```

```
   TMS = 1;

   JTAG_StrobeTCK ();                    // 转到测试逻辑复位状态
   JTAG_StrobeTCK ();
   JTAG_StrobeTCK ();
   JTAG_StrobeTCK ();
   JTAG_StrobeTCK ();

   TMS = 0;

   JTAG_StrobeTCK ();                    // 转到 Run_Test/Idle 状态
}

//-------------------------------------------------------------------------
// JTAG_IR_Scan
//-------------------------------------------------------------------------
// 该函数将<num_bits>长度的<instruction>装入目标系统的 JTAG 指令寄存器。
// 使其停在 Test/Idle 状态。返回值是从 IR 读到的 n 位值。
// 假定 JTAG 状态机从 Run_Test/Idle state 开始运行。
//
unsigned int JTAG_IR_Scan (unsigned int instruction, int num_bits) {

   unsigned int retval;                  // JTAG 指令读
   int   i;                              // JTAG IR 位计数器

   retval = 0x0;

   TMS = 1;
   JTAG_StrobeTCK ();                    // 转到 SelectDR
   TMS = 1;
   JTAG_StrobeTCK ();                    // 转到 SelectIR
   TMS = 0;
   JTAG_StrobeTCK ();                    // 转到 Capture_IR
   TMS = 0;
   JTAG_StrobeTCK ();                    // 转到 Shift_IR state

   for (i=0; i < num_bits; i++) {

      TDI = (instruction & 0x01);        // 移位 IR, LSB 先移
      instruction = instruction >> 1;

      retval = retval >> 1;
      if (TDO) {
         retval |= (0x01 << (num_bits - 1));
      }

      if (i == (num_bits - 1)) {
         TMS = 1;                        // 转到 Exit1_IR 状态
      }

      JTAG_StrobeTCK();
   }

   TMS = 1;
   JTAG_StrobeTCK ();                    // 转到 Update_IR
```



```
   TMS = 0;
   JTAG_StrobeTCK ();                    // 转到 RTI state

   return retval;
}

//-----------------------------------------------------------------------
// JTAG_DR_Scan
//-----------------------------------------------------------------------
// 该函数将<data>的<num_bits>位移入到数据寄存器，返回从数据寄存器读到的数据，
// 最多 32 位。使状态机停在 Run_Test/Idle 状态。
// 假定 JTAG 状态机从 Run_Test/Idle state 开始运行。
//


unsigned long JTAG_DR_Scan (unsigned long dat, int num_bits) {

   unsigned long   retval;               // JTAG 返回值
   int   i;                              // JTAG DR 位计数器

   retval = 0x0L;

   TMS = 1;
   JTAG_StrobeTCK ();                    // 转到 SelectDR
   TMS = 0;
   JTAG_StrobeTCK ();                    // 转到 Capture_DR
   TMS = 0;
   JTAG_StrobeTCK ();                    // 转到 Shift_DR state

   for (i=0; i < num_bits; i++) {

      TDI = (dat & 0x01);                // 移位 DR, LSB 先移
      dat = dat >> 1;

      retval = retval >> 1;
      if (TDO) {
         retval |= (0x01L << (num_bits - 1));
      }

      if ( i == (num_bits - 1)) {
         TMS = 1;                        // 转到 Exit1_DR 状态
      }

      JTAG_StrobeTCK();
   }
   TMS = 1;
   JTAG_StrobeTCK ();                    // 转到 Update_DR
   TMS = 0;
   JTAG_StrobeTCK ();                    // 转到 RTI 状态

   return retval;
}

//-----------------------------------------------------------------------
// JTAG_IWrite
```

```
//-------------------------------------------------------------------------
// 该函数执行一个间接写 <ireg>寄存器的操作，写数据为<dat>，数据长度为<num_bits>
// 它在写操作后进行查询，操作完成后返回。注意：此处的查询操作是看 JTAG 寄存器
// 的写操作是否完成，而不是指 FLASH 写操作。对 FLASH 写操作的查询在更高一级处理。
// 有效间接寄存器为：
//  FLCN – FLASH 控制
//  FLSC – FLASH 预分频器
//  FLA – FLASH 地址
//  FLD – FLASH 数据
// 停留在 Run_Test/Idle 状态
//
void JTAG_IWrite (unsigned int ireg, unsigned long dat, int num_bits) {

   int done;                                // TRUE = 写完成；否则为 FALSE

   JTAG_IR_Scan (ireg, INST_LENGTH);    // 将<ireg>装入 IR

   dat |= (0x03L << num_bits);           // 在数据后面加‘写’操作码

   // load DR with <dat>
   JTAG_DR_Scan (dat, num_bits + 2);    // 启动 JTAG 写

   // 将 DR 装入'0'，并检查 BUSY 位是否变为'0'.
   do {
      done = !(JTAG_DR_Scan (0x0L, 1)); // 查询 JTAG_BUSY 位
   } while (!done);
}

//-------------------------------------------------------------------------
// JTAG_IRead
//-------------------------------------------------------------------------
// 该函数执行一个间接读 <ireg>寄存器的操作，数据长度为<num_bits>。
// 它在读操作后进行查询，操作完成后返回。注意：此处的查询操作是看 JTAG 寄存器
// 的读操作是否完成，而不是指 FLASH 读操作。对 FLASH 读操作的查询在更高一级处理。
// 有效间接寄存器为：
//  FLCN – FLASH 控制
//  FLSC – FLASH 预分频器
//  FLA – FLASH 地址
//  FLD – FLASH 数据
// 使状态机停留在 Run_Test/Idle 状态
//
//
unsigned long JTAG_IRead (unsigned int ireg, int num_bits) {

   unsigned long retval;                // 读操作返回的数值
   int done;                            // TRUE = 写完成，否则为 FALSE

   JTAG_IR_Scan (ireg, INST_LENGTH);   // 将<ireg>装入 IR

   // 将 DR 装入读操作码(0x02)
```



```
   JTAG_DR_Scan (0x02L, 2);              // 启动 JTAG 读

   do {
      done = !(JTAG_DR_Scan (0x0L, 1)); // 查询 JTAG_BUSY 位
   } while (!done);

   retval = JTAG_DR_Scan (0x0L, num_bits + 1); // 允许查询操作读其余的位
   retval = retval >> 1;                       // 移出 JTAG_BUSY 位

   return retval;
}

//---------------------------------------------------------------------
// FLASH_ByteRead
//---------------------------------------------------------------------
// 该函数读地址为<addr>的字节并存于<pdat>指向的地址
// 如果操作成功则返回 TRUE，否则返回 FALSE（被设置为读保护的页）。
//
int FLASH_ByteRead (unsigned int addr, unsigned char *pdat)
{
   unsigned long testval;                // 保存 FLASHDAT 读的结果
   int done;                             // TRUE/FALSE 标志
   int retval;                           // 操作成功为 TRUE

   JTAG_IWrite (FLASHSCL, 0x86L, FLSC_LEN);// 根据 SYSCLK 频率设置 FLASHSCL
                                           // (2MHz = 0x86)

   // 将 FLASHADR 设置为要读的地址
   JTAG_IWrite (FLASHADR, (unsigned long) addr, FLA_LEN);

   JTAG_IWrite (FLASHCON, 0x02L, FLCN_LEN); // 设置 FLASHCON 以执行
                                            //  FLASH 读操作(0x02)

   JTAG_IRead (FLASHDAT, FLD_RDLEN);        // 启动读操作

   JTAG_IWrite (FLASHCON, 0x0L, FLCN_LEN);  // 设置 FLASHCON 以执行查询操作

   do {
      done = !(JTAG_IRead (FLASHDAT, 1));     // 查询 FLBUSY，等待操作完成
   } while (!done);

   testval = JTAG_IRead (FLASHDAT, FLD_RDLEN); // 读结果数据

   retval = (testval & 0x02) ? FALSE: TRUE;    // FLFail 是 LSB 的前一位

   testval = testval >> 2;                  // 将 data.0 移入 LSB 位置

   *pdat = (unsigned char) testval;         // 将数据存入返回位置

   return retval;                           // 返回 FLASH 成功/失败
}

//---------------------------------------------------------------------
```

```
// FLASH_ByteWrite
//-------------------------------------------------------------------------
// 该函数将数据<dat>写入 FLASH，地址为<addr>
// 如果操作成功则返回 TRUE，否则返回 FALSE（被设置为写保护的页）。
//
int FLASH_ByteWrite (unsigned int addr, unsigned char dat)
{
   unsigned long testval;                 //保存 FLASHDAT 读的结果
   int done;                              // TRUE/FALSE 标志
   int retval;                            //操作成功为 TRUE

   JTAG_IWrite (FLASHSCL, 0x86L, FLSC_LEN); //根据 SYSCLK 频率设置 FLASHSCL
                                            // (2MHz = 0x86)

   // 设置 FLASHADR 为要写的地址
   JTAG_IWrite (FLASHADR, (unsigned long) addr, FLA_LEN);

   JTAG_IWrite (FLASHCON, 0x10L, FLCN_LEN); // 设置 FLASHCON 以执行
                                            // FLASH 写操作(0x10)

   // 启动写操作
   JTAG_IWrite (FLASHDAT, (unsigned long) dat, FLD_WRLEN);

   JTAG_IWrite (FLASHCON, 0x0L, FLCN_LEN);  // 设置 FLASHCON 以执行查询操作

   do {
      done = !(JTAG_IRead (FLASHDAT, 1));     // 查询 FLBusy，等待操作完成
   } while (!done);

   testval = JTAG_IRead (FLASHDAT, 2);        // 读 FLBusy 和 FLFail

   retval = (testval & 0x02) ? FALSE: TRUE;  // FLFail 是 LSB 的前一位

   return retval;                             // 返回 FLASH 成功/失败
}

//-------------------------------------------------------------------------
// FLASH_PageErase
//-------------------------------------------------------------------------
// 该函数擦除包含<addr>的页。该函数假定目前没有进行 FLASH 操作。
// 该函数返回时保证没有 FLASH 操作在进行。如果操作成功则返回 TRUE，
// 否则返回 FALSE（页保护）。
//
int FLASH_PageErase (unsigned int addr)
{
   unsigned long testval;                  // 保存 FLASHDAT 读的结果
   int done;                              // TRUE/FALSE 标志
   int retval;                            // 操作成功为 TRUE

   JTAG_IWrite (FLASHSCL, 0x86L, FLSC_LEN); // 根据 SYSCLK 频率设置 FLASHSCL
                                            // (2MHz = 0x86)
```



```
   // 将 FLASHADR 设置为待擦除页内的地址
   JTAG_IWrite (FLASHADR, (unsigned long) addr, FLA_LEN);

   JTAG_IWrite (FLASHCON, 0x20L, FLCN_LEN);  // 设置 FLASHCON 以执行
                                             // FLASH 擦除操作(0x20)

   JTAG_IWrite (FLASHDAT, 0xa5L, FLD_WRLEN); // 设置 FLASHDAT 为 0xa5
                                             // 以启动擦除过程

   JTAG_IWrite (FLASHCON, 0x0L, FLCN_LEN);   // 设置 FLASHCON 以执行查询操作

   do {
      done = !(JTAG_IRead (FLASHDAT, 1));    // 查询 FLBusy，等待操作完成
   } while (!done);

   testval = JTAG_IRead (FLASHDAT, 2);       // 读 FLBusy 和 FLFail

   retval = (testval & 0x02) ? FALSE: TRUE;  // FLFail 是 LSB 的前一位

   // 根据 FLFail 位设置返回值
   return retval;                            // 返回 FLASH 成功/失败
}
//***文件结束***
```