AN015 — 软件 UART 示例

相关器件

本应用笔记适用于下列器件：

C8051F000、C8051F001、C8051F002、C8051F005、C8051F006、C8051F010、C8051F011、C8051F012、C8051F015、C8051F016、C8051F017、C8051F220、C8051F221、C8051F226、C8051F230、C8051F231、C8051F236。

引言

本应用笔记讨论基于 C8051Fxxx 系列器件的软件 UART 的实现方法。本文给出两个完整的例子：一个用 PCA 作为波特率发生器的 C 语言程序和一个用定时器 0 作为波特率发生器的汇编语言程序。

关键特性

这两个软件设计示例在节省硬件资源和 CPU 带宽的前提下几乎完全模拟硬件 UART 的功能。下面是两个例子中都具备的关键特性：

- 一个与硬件 UART 相似的接口，有用户级发送和接收中断。
- 支持中断或查询方式访问。
- 全双工通信，使用 18.432MHz 时钟源时最大波特率可达 57.6 kbps。
- 基于状态的中断驱动实现方案，需要的 CPU 开销最小。
- 最少的硬件资源占用：
  - C 语言示例程序使用两个 PCA 模块。
  - 汇编语言示例程序使用定时器 0 工作于方式 3。

实现选项

在实现一个软件 UART（SW UART）时最重要的是在硬件资源占用和速度/效率之间权衡。使用较多硬件的设计可消耗较小的 CPU 带宽并允许较高的位速率。这种权衡将在下面讨论。

波特率源

在传输每一位时都必须产生一个中断；在全双工 115.2 kbps 的速率下，每 4.3 微秒就要产生一个中断。产生这些中断（波特率源）的方法不同，则实现时所需的 CPU 开销会有很大的差异。可选择的波特率源包括：8 位定时器、16 位定时器和可编程计数器阵列（PCA）。注意：对于全双工操作，需要两个波特率源（发送和接收各一个）。

使用 8 位定时器的方案允许将一个 16 位硬件定时器用于产生发送和接收波特率。定时器 0 工作于方式 3 时具有这种能力。注意：当定时器 0 工作于方式 3 时，定时器 1 的功能将减少；但是定时器 1 仍可用作硬件 UART（HW UART）的波特率发生器。使用 8 位定时器节省硬件资源，但是会有 CPU 软件开销和时间延迟的问题。这些问题在例 2 中讨论。

上述方法的一个替代方案是使用 16 位自动重装载定时器。在这种情况下，SW UART 占用两个 16 位硬件定时器：一个用于发送，一个用于接收。任何可用的定时器都能满足要求，但定时器 2 和定时器 3 的自动重装载能力可以减少软件开销并可消除任何中断延迟问题。此外，16 位定时器还可支持更宽的波特率范围。

Silicon Laboratories Inc.
4635 Boston Lane
Austin, TX 78735
Email: mcuinfo@silabs.com
Internet: www.silabs.com

新华龙电子有限公司
深圳市福田区华强北路现代之窗大厦 A 座 13F C 室(518013)
电话：0755-83645240 83645242 83645244 83645251
电邮：shenzhen@xhl.com.cn
网址：www.xhl.com.cn

可编程计数器阵列（PCA）也为 SW UART 提供了一个很好的解决方案，所提供的 C 语言示例能说明这一点。PCA 包含一个专用的 16 位计数器/定时器和五个 16 位的捕捉/比较模块。每个模块都可以被设置为在 PCA 计数器与相对应的比较模块的内容一致时产生中断。由于 PCA 计数器在产生中断时并不停止运行，所以该方案可以避免中断延迟累加的问题。PCA 实现方案不适用于 C8051F2xx 器件。

其它考虑

上述的每种定时器源都可以用 SYSCLK 或一个外部信号作为时钟。在所提供的例子中，波特率源用 SYSCLK 作为时钟，而 SYSCLK 源自外部的 18.432MHz 晶体。任何波特率/晶体频率组合都是允许的，但软件开销限制了波特率与 SYSCLK 的最大比值。

起始位检测也是 SW UART 接收器的一个重要问题。C8051F00x 和 C8051F01x 器件提供了很多外部中断源，其中有几个可被配置为下降沿触发。本文的两个示例程序都使用外部中断来检测起始位。

例 1：可编程计数器阵列实现方案

例 1 使用两个 PCA 模块产生接收和发送波特率（分别为模块 0 和 1）。这两个模块被配置为软件定时器方式，以产生波特率中断。对 PCA 的介绍见 AN007。

程序结构

在软件定时器方式，当 PCA 计数器与某个比较模块中的值一致时，PCA 可产生一个中断。由于 PCA 计数器不停止运行，比较模块可在每个位时间被更新以精确产生下一个位时间。另外，PCA 还提供了在起始位检测中很有用的捕捉功能。

PCA 模块可通过交叉开关与外部信号连接。这些信号（对模块 n 称为 CEXn）可用于触发 PCA 计数器捕捉。在 SW UART 接收器中用到这一特性。起始位识别是用模块 0 实现的，模块 0 被配置为在 RX 引脚出现一个下降沿时捕捉 PCA 计数器的内容。该功能有两个优点：（1）起始位检测容易实现；（2）由于捕捉是在检测到下降沿后立即完成的，所以位采样时间不受中断响应延迟的影响。

实现

例 1 中的发送和接收操作是在 PCA 中断服务程序的两个独立的状态机中实现的，如图 1 所示。

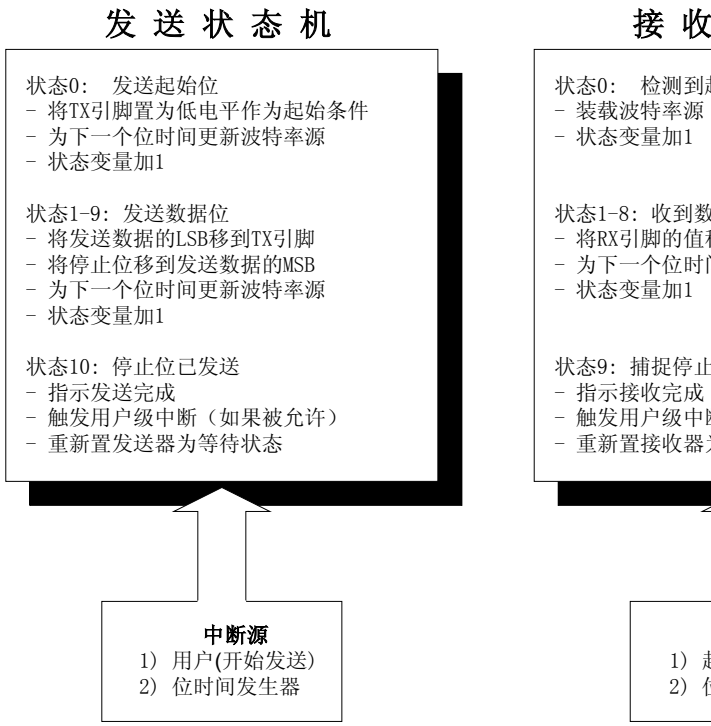


图 1. 发送和接收状态机

接收状态机

当 SW UART 被初始化时，PCA 模块 0 被配置为下降沿捕捉方式。它的输入（CEX0）通过交叉开关接到一个通用 I/O 引脚（P0.2, SW_RX）。在状态机处于状态 0 时，如果在 SW_RX 引脚检测到下降沿，则会产生一个中断。由于模块工作在捕捉方式，PCA 计数器的内容被装入到模块 0 捕捉寄存器。注意该值与中断响应延迟无关。在检测到起始位后，模块 0 被切换到软件定时器方式，并且 3/2 个位时间被加到模块 0 捕捉寄存器中。额外的 1/2 位时间只在检测到起始位之后使用，其作用是使采样发生在下一个位周期的中间（见图 2）。当 PCA 计数器计到模块 0 捕捉寄存器中的数值时，产生第一个位采样中断（此处为 LSB）。

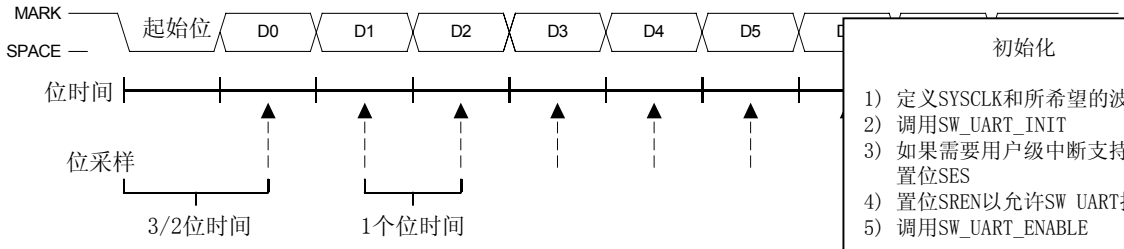


图 2. SW UART 位时序

状态 1-8 在模块匹配中断发生时执行。在每个状态，从 SW_RX 采样数据位并将其移入到变量 RXSHIFT。PCA 模块 0 的内容在每个状态都被更新，以提供下一个位时间中断（一个位时间被加到比较寄存器）。状态变量也被加 1。

状态 9 捕捉停止位，设置 SRI，最后使接收器返回到等待状态。

发送状态机

用户可以通过强制 PCA 模块 1 产生中断（设置 CCF1=1）来启动一次发送。在状态 0，TX 引脚被强制为低电平以产生起始条件。此时读 PCA 计数器的值并将该值加上一个位时间后装入到模块 1 捕捉寄存器。注意：从产生起始位到读出 PCA 计数值之间要经过几个系统时钟周期。这是例 1 中唯一的中断延迟影响位时间的地方。这个影响是可以忽略的（最坏的情况是大约 1/16 位时间，这是在 18.432MHz 系统时钟频率、57.6 kbps 波特率的情况下）。

状态 1-9 在发生模块匹配中断时执行。在每个状态，TDR 的 LSB 被移出，而一个代表停止位的‘1’被移入 TDR 的 MSB。一个位时间被加到模块 1 捕捉寄存器以产生下一个位时间。经过 9 次移位后，数据字节+停止位被发送出去。最后发送结束标志（STI）被置位，发送忙标志（STXBSY）被清除，TX 状态变量被复位。

程序接口

SW UART 支持查询和中断驱动方式接口。查询支持是通过禁止用户级中断（SES=0）来实现的。可以通过查询发送和接收标志（分别为 STI 和 SRI）看传输是否结束。例 1 的初始化和查询方式的编程过程如图 3 所示。

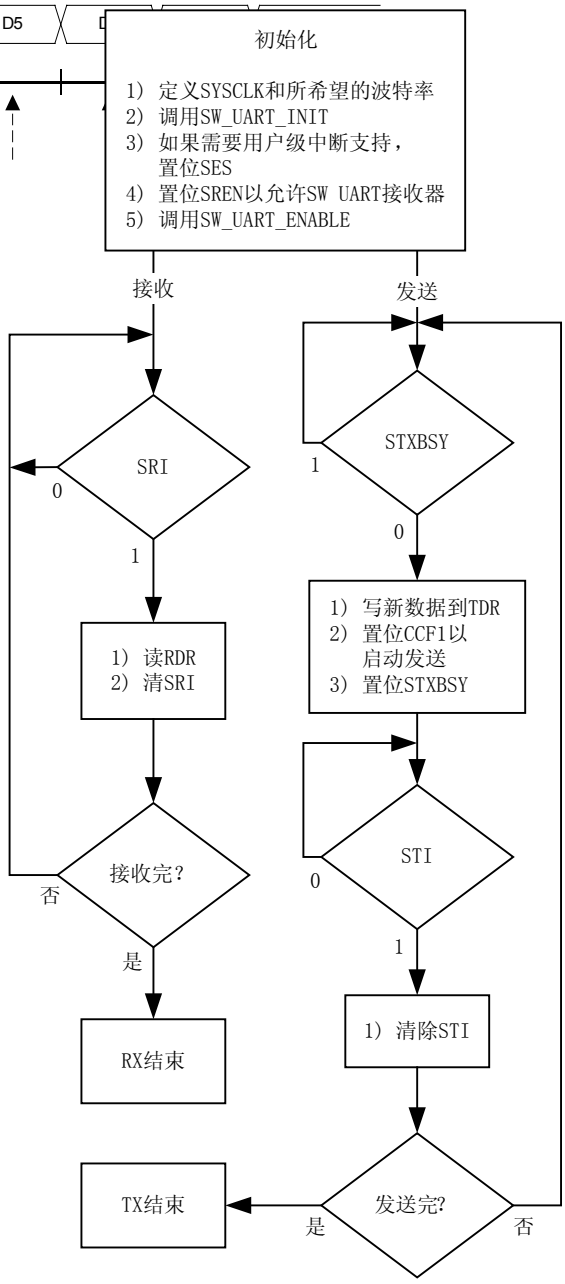


图 3. 例 1 的用户级查询方式接口

初始化程序 SW_UART_INIT 配置 SW UART 中用到的 PCA、中断和状态变量。SW_UART_ENABLE 程序允许 SW UART。必须将 SREN 位置‘1’以允许接收。注意，常数 TIME_COUNT 是用软件由常数 BAUD_RATE 和 SYSCLK 计算而得。

为了使用中断方式，设置 SES=1。中断方式的编程过程如图 4 所示。

如果用户级中断支持被允许（SES=1），则每当一次发送或接收过程结束就会产生一个 IE7 中断。与硬件 UART 一样，用户软件必须通过检查发送/接收结束标志来确定中断源。如果发送和接收同时完成，用户软件将只收到一个中断，IE7 中断服务程序必须能处理这种情况。有两种处理方式：（1）在一次中断服务程序执行中同时处理发送和接收，（2）处理一个（STI 或 SRI）并强制产生一个中断使中断服务程序被重新调用以服务另一个中断。建议采用第二种方案，这样可使中断服务程序的执行时间最短。

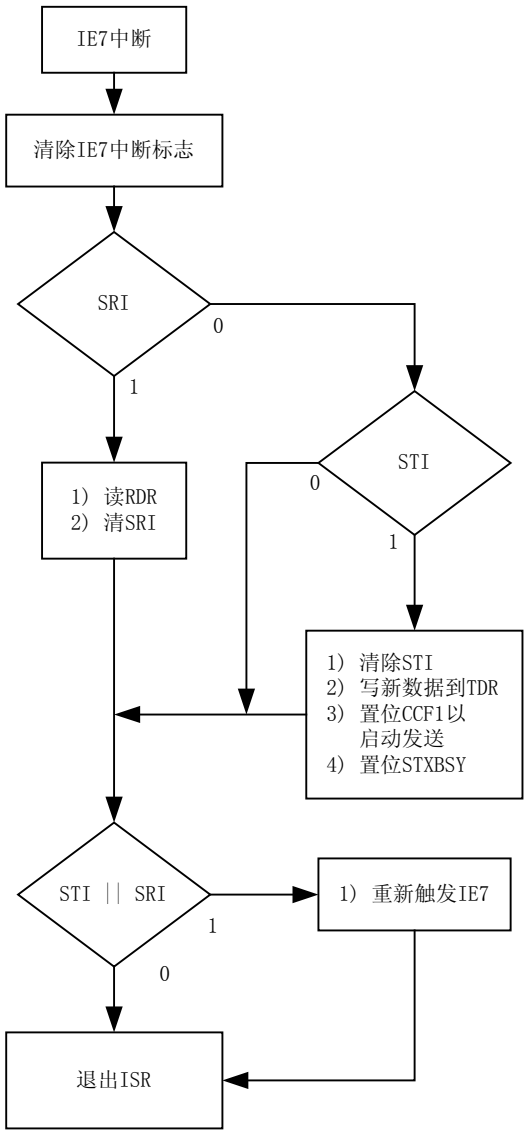


图 4. 例 1 的用户级中断方式接口

示例程序中提供了 SW UART 与 HW UART 接口的测试代码。跳线连接方式如图 5 所示。

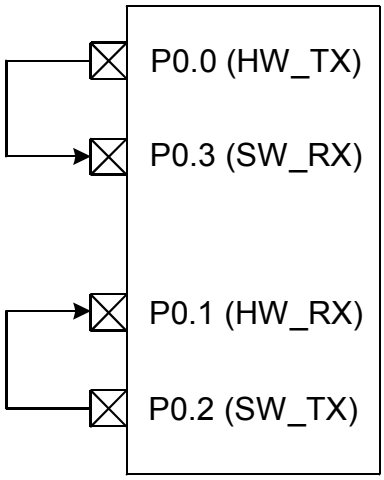


图 5. 例 1 的测试配置

测试代码配置并允许 HW UART，使其工作于方式 1 并使用定时器 1 作为波特率发生器。测试代码还完成对定时器 1 的设置。通过改变 BAUD_RATE 和 SYSCLK 常数可以测试不同的波特率和晶体。硬件和软件 UART 的波特率计数值用软件根据这些常数计算而得。测试程序在两个方

向上各发送 15 个字符。

为了测试查询方式下的 SW UART，给下行加上注释符：

; INTERRUPT_TEST();

去掉下行的注释符：

POLLED_TEST();

将上面的步骤反过来可测试中断方式下的 SW UART。去掉下行的注释符

INTERRUPT_TEST();

给下行加上注释符

; POLLED_TEST();

例 2 中执行时间最长的状态需要 113 个系统时钟周期（TX 状态 1-9）。对于 18.432MHz 的晶体，SW UART 发送或接收操作在最坏的情况下需要 6μs/位（113*TSYSCLK）。在 57.6 kbps 的波特率下，发送或接收占大约 35%的 CPU 带宽（全双工时占 70%）。对于例 1 的软件，用 Keil 编译器编译时全双工的软件开销可用下式估算：

全双工软件开销（%）≈BAUD_RATE/81000。

按照上面的公式，波特率高于 80 kbps 时不能支持全双工操作。只有在 SW UART 正在进行传输过程时才会有软件开销的限制。

例 2：8 位定时器实现方案

在例 2 中，SW UART 使用定时器 0 工作于方式 3。在该方式下，定时器 0 被分成两个 8 位定时器：一个用于发送，一个用于接收。TL0 用作接收定时器；TH0 用作发送定时器。

当定时器 0 工作于方式 3 时，定时器 1 不能设置 TF1 标志，不能产生中断，也不能用外部信号作为时钟。但是，如果被配置为方式 2（自动重装载的 8 位定时器），则定时器 1 仍可被用作 HW UART 的波特率发生器。在定时器 0 工作于方式 3 时，可以通过工作方式设置来禁止/允许定时器 1。定时器 1 在方式 3 时被禁止，在其它方式被允许。

用定时器 1 作为 HW UART 的波特率源时，本方案大概是对硬件资源的最有效利用。缺点是软件开销增加（相对于 16 为定时器方案）。定时器 0 的方式 3 没有自动重装载能力；手动装载定时器需要在每次执行中断服务程序（ISR）时进行 16 位的数据传送。另外，中断延迟会影响位时间精度。可以将定时器的预装值加上一个校正常数以补偿典型中断延迟，但是无法考虑中断延迟之间的差别。

较低的波特率可能需要用大于 8 位长度的定时器对位时间计时。在 SYSCLK 为 18.432MHz 且定时器 0 工作于 SYSCLK/1 方式的情况下，低于 72kbps 的波特率需要大于 256 的定时器计数值。可选方案包括：

1）使用定时器 0 工作在 SYSCLK/12 方式。这样可以用 8 位长度得到较低的波特率，但要得

到标准波特率/SYSCLK 的组合将更加困难。

2）使用定时器 0 工作在 SYSCLK/1 方式，但要在定时器 ISR 中手动操作高位定时器字节。注意：不管波特率为何，对每次发送和接收，使用该方法时每隔 256 个 SYSCLK 将产生一个中断（每次低 8 位溢出时产生一个中断）。例 2 的软件是选项 2 的实例。

程序结构

例 2 中的发送和接收操作是在定时器 0 和定时器 1 中断服务程序的两个独立的状态机中实现的（见图 1）。定时器 0 的中断服务程序管理接收状态机；定时器 1 的中断服务程序管理发送状态机。/INT0 的中断服务程序启动接收状态机，但在接收状态不为 0 时被禁止。SW UART 接收器用外部中断源/INT0 捕捉起始位，/INT0 被配置为下降沿触发。在等待一个起始位时，/INT0 被允许，但在传输过程中/INT0 被禁止。/INT0 通过交叉开关连到通用 I/O 引脚。有关交叉开关配置的详细信息见 AN001。

由于在中断服务程序中所有的定时器装载都是手动完成的，所以必须对中断延迟进行补偿。从每个定时器预装值中减去一个‘溢出常数’，以补偿这个中断延迟和程序从定时器溢出到重装定时器初值所需要的执行时间。这些常数与 SYSCLK 频率或波特率无关，但是没有考虑到中断延迟的变化。

实现

注：对于本例的讨论，假设波特率低到用 8 位定时器不能满足要求。直接地址字节 BCRHI 和 BCTHI 分别用于手动管理发送和接收定时器的高字节。

发送状态机

当 SW UART 被初始化并被允许时，TX 中断有效但仍被禁止。用户通过允许发送中断来启动传送过程（注意，定时器 0 的高字节 TH0 产生 TX 中断）。

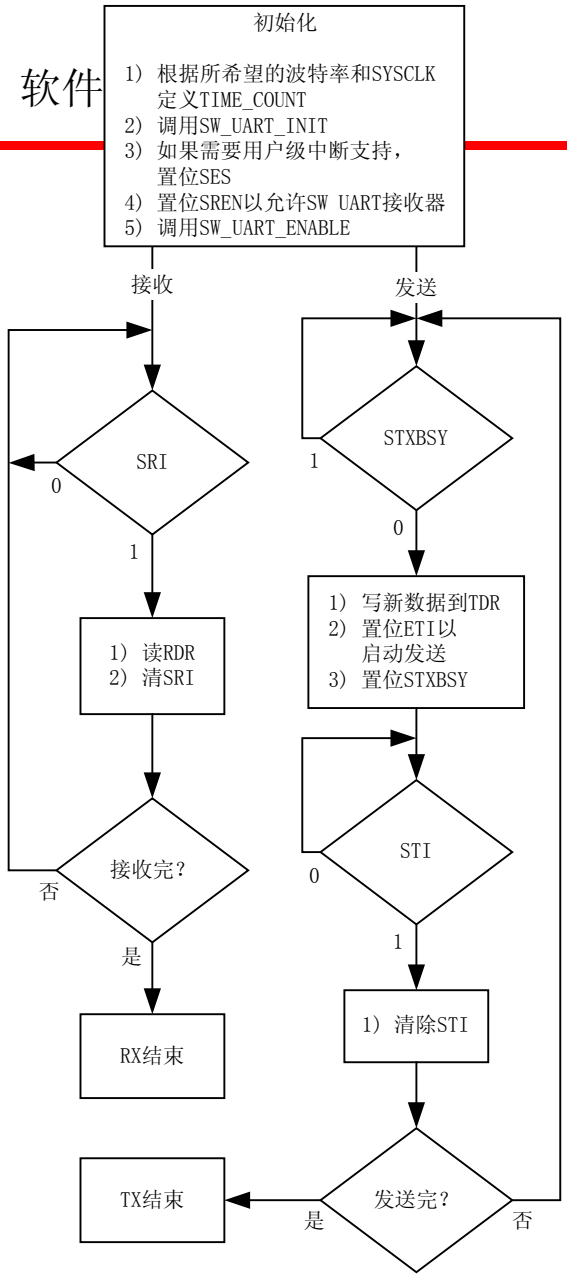
在状态 0，TX 引脚被强制为低电平以产生起始条件，定时器被装入一个位时间以产生下一个中断。

```
mov     BCTHI, #HIGH(TX_BT)      ; 将高字节装入 BCTHI
mov     TH0, #-LOW(TX_BT)        ; 将低字节装入 TH0
```

注意，BCTHI 的装入值是无符号的位时间高字节，而 TH0 的装入值是位时间低字节的负数。这是因为定时器 0（与所有硬件定时器一样）是一个加 1 计数器，而 BCTHI 向下计数。TH0 在从 0xFF 计到 0x00 时发生溢出并产生一个中断；BCTHI 在每次中断时减 1，当减到 0 时表示计满一个位时间。

对于状态 1-9，每当 BCTHI 达到 0 时有一个状态被执行。在每个状态，发送数据寄存器（TDR）的 LSB 被移出到 TX 引脚。TX 定时器被装入一个位时间值，一个‘1’被移入 TDR 的 MSB 以代表状态 9 的停止位（在传输完成后 TDR 中应保持 0xFF）。

状态 10 将发送结束标志(STI)置‘1’，清除发送忙标志(STXBSY)，并触发一个 IE7 中断(如果用户级中断支持被允许)。



接收状态机

在状态 0，/INT0 用做 RX 输入(被配置为下降沿有效，高优先级)。发生/INT0 中断意味着检测到起始条件。/INT0 的中断服务程序用 3/2 位时间装载 RX 定时器(TL0 + BCRHI)。BCRHI 在每次 TL0 溢出时减 1。

状态 1-8 在 BCRHI 达到 0 时执行。在每个状态，SW_RX 引脚被采样并被移入变量 RXSHIFT 的 LSB。RX 定时器被重新装载以产生下一个采样时间。状态 9 捕捉停止位，但没有提供帧错误检测(未检测停止位极性)。如果用户级中断被允许，该状态允许并触发 IE7 中断。

程序接口

例 2 支持查询和中断驱动方式接口。查询方式的初始化过程和程序流程示于图 6。在该例中 TIME_COUNT 常数必须明确定义。

图 7 给出了中断方式下 IE7 中断服务程序的流程。注意，接收操作首先被服务，因为它对延迟最敏感。

为了处理发送和接收同时完成的情况，本例的程序在完成对一个功能的服务后又重新自触发去服务另一个功能。

图 6.例 2 的用户级查询方式接口

本例的程序中提供了测试代码。为了测试查询方式的代码，在主程序中去掉下行的注释符

ajmpPolledRX_PolledTX

给下行加上注释符：

; ajmp InterruptRX_InterruptTX

为了运行中断方式的测试代码，将上面的步骤反过来。给下行加上注释符

; ajmp PolledRX_PolledTX

去掉下行的注释符

ajmp PolledRX_PolledTX

在 SW_GPIO_TX 和 SW_GPIO_RX 之间加一个跨接片即可很容易地对 SW UART 进行快速评估。注意，这种评估方法只对中断方式的测试代码有用。

在 SYSCLK 为 18.432MHz 时，例 2 中给出的软件工作在全双工方式时最大波特率可达到 57.6kbps。

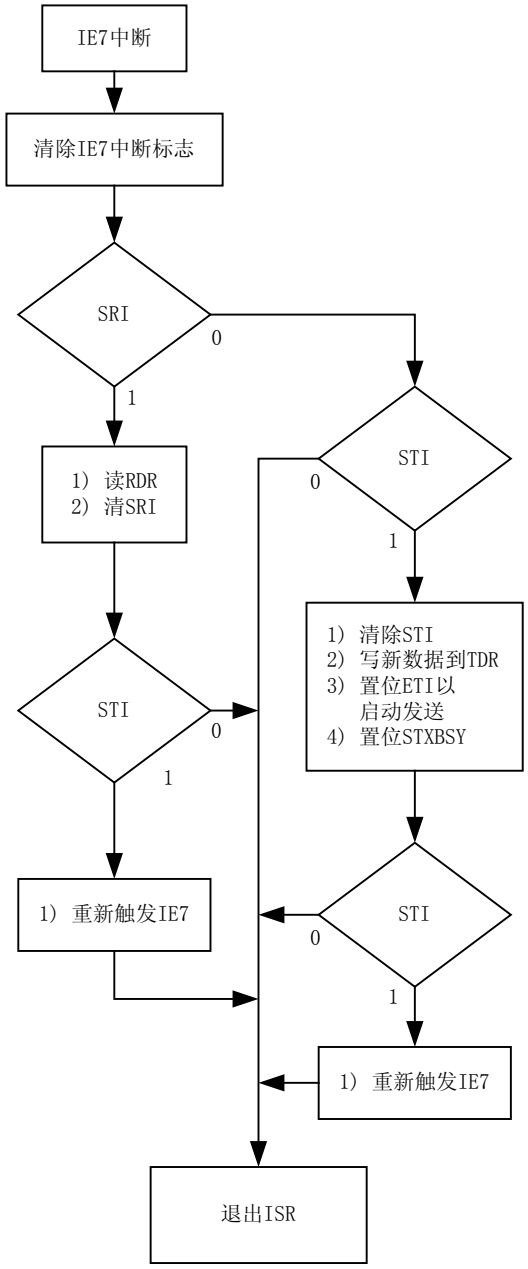


图 7. 例 2 的用户级中断接口

示例代码

```
//-----------------------------------------------------------------------
//
// Copyright 2001   Cygnal Integrated Products, Inc.
//
// 文件名       : AN015_1.c
// 目标器件     : C8051F00x, C8051F01x
// 编写日期     : 03/10/01
// 作者         : JS
//
// 软件 UART 程序，使用 PCA 作为波特率发生器。
// PCA 模块 0 用作接收波特率源和起始位检测器。为了检测起始位，模块 0 被配置为
// 负沿捕捉方式。对于所有其它的 SW_UART 操作，模块 0 被配置为软件定时器。模块
// 匹配中断用于产生波特率。模块 1 用软件定时器方式产生发送波特率。
// 程序假设一个外部晶体连接在 XTAL1 和 XTAL2 引脚之间。
// 外部晶体的频率应在 SYSCLK 常数中定义。
//
// 初始化过程：
// 1) 根据外部晶体频率定义 SYSCLK。
// 2) 定义所期望的波特率 BAUD_RATE。
// 3) 调用 SW_UART_INIT()。
// 4) 置位 SREN 以允许 SW_UART 接收器。
// 5) 置位 SES，只在需要用户级中断支持时。
// 6) 调用 SW_UART_ENABLE()。
//
// 发送：
// 1) 查询 STXBSY 是否为 0。
// 2) 写数据到 TDR。
// 3) 置位 CCF1 以启动发送。
// 4) 发送完成时置位 STI。如果用户级中断被允许，产生一个 IE7 中断。
//
// 接收：
// 1) 如果在查询方式，查询 SRI。如果在中断方式，在 IE7 中断服务程序中查询 SRI。
// 2) 从 RDR 读数据。
//
// 示例代码中包含查询和中断方式的测试代码。测试代码假设 HW_UART 和 SW_UART 引脚
// 在外部相连：
// P0.0 (HW_UART TX) ->  P0.3 (SW_UART RX)
// P0.1 (HW_UART RX) ->  P0.2 (SW_UART TX)
//
// 为了使用查询方式的测试代码，将位于主程序底端的对 INTERRUPT_TEST() 的调用行
// 前面加上注释符，并去掉调用 POLLED_TEST() 的程序行前面的注释符。为了测试中断
// 方式，在 POLLED_TEST() 调用行前面加上注释符，并去掉调用 INTERRUPT_TEST()
//的程序行前面的注释符。
//
```

```
// 测试程序配置硬件 UART 用定时器 1 作为波特率源。定时器 1 的预装值用 SYSCLK 和
// BAUD_RATE 常数自动计算。
//
//-----------------------------------------------------------------------
// 包含文件
//-----------------------------------------------------------------------
#include <c8051f000.h>                  // SFR 声明

//-----------------------------------------------------------------------
// 全局常量
//-----------------------------------------------------------------------

#define  BAUD_RATE   57600          // 用户定义的 SW_UART 波特率
#define  SYSCLK      18432000       // 系统时钟取自外部 18.432MHz 晶体

#define  TIME_COUNT  SYSCLK/BAUD_RATE/4 // 对应一个位时间的 PCA 计数值
                                        // （PCA 被配置为对 SYSCLK/4
                                        // 计数）

#define   TH_TIME_COUNT TIME_COUNT*3/2   // 3/2 位时间，在接收到一个起
                                        // 始位之后使用。在起始位边沿之
                                        // 后 RX 应在一个位时间内保持低
                                        // 电平，第一个位采样在下一个位
                                      // 时间的中间开始。

#define HW_TIME_COUNT   SYSCLK/BAUD_RATE/16 // 用于产生 HW_UART 波特率
                                            // 的定时计数值。根据前面定
                                            // 义的常数 SYSCLK 和
                                            // BAUD_RATE 自动计算。

//-----------------------------------------------------------------------
//全局变量
//-----------------------------------------------------------------------

bit    SRI;                             // SW_UART 接收完成标志
bit    STI;                             // SW_UART 发送完成标志
bit    STXBSY;                          // SW_UART 发送忙标志
bit    SREN;                            // SW_UART 接收允许
bit    SES;                             // SW_UART 用户级中断支持允许

sbit   SW_RX = P0^2;                    // SW_UART 接收引脚
sbit   SW_TX = P0^3;                  // SW_UART 发送引脚

char   TDR;                           // SW_UART 发送数据寄存器
char   RDR;                           // SW_UART 接收数据寄存器(锁存器)

// 测试变量
char k, m;                            // 测试计数器
char idata SW_BUF[20];                // SW_UART 测试接收缓冲区
```

```
bit    HW_DONE;                          // HW 发送结束标志(发送完 15 个字符)
bit    SW_DONE;                          // SW 发送结束标志(发送完 15 个字符)

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

void   SW_UART_INIT();                   // SW_UART 初始化程序
void   SW_UART_ENABLE();                 // SW_UART 允许程序
void   PCA_ISR();                        // SW_UART 中断服务程序
void   INTERRUPT_TEST(void);             // SW_UART 中断方式测试程序
void   POLLED_TEST(void);                // SW_UART 查询方式测试程序
void   USER_ISR(void);                   // SW_UART 测试中断服务程序
void   HWU_INIT(void);                   // HW_UART 初始化程序
void   HW_UART_ISR(void);                // HW_UART 中断服务程序

//-----------------------------------------------------------------------
// 主程序
//-----------------------------------------------------------------------
// - 禁止看门狗定时器
// - 配置外部晶体；稳定后将 SYSCLK 切换到外部晶体。
// - 配置交叉开关和端口。
// - 初始化并允许 SW_UART。
// - 调用测试程序。
//
void MAIN (void){
   int delay;                            // 延时计数器

   OSCXCN = 0x66;                        // 允许外部晶体
   WDTCN = 0xDE;                         // 禁止看门狗定时器
   WDTCN = 0xAD;

   // Port Setup
   XBR0 = 0x0C;                          // HW_UART 连到 P0.0 和 P0.1;
                                      // CEX0 连到 P0.2.
   XBR2 = 0x40;                       // 允许交叉开关，允许上拉

   PRT0CF = 0x09;                     // 将 P0.0 (HW TX)和 P0.3 (SW TX)
                                      // 配置为推挽输出

   delay=256;                         // 在查询 XTLVLD 之前延时>1 ms
   while(delay--);

   while (!(OSCXCN & 0x80));          // 等待外部晶体起振
   OSCICN = 0x0C;                     // 切换到外部振荡器
   OSCICN = 0x88;                     // 禁止内部振荡器；允许时钟丢失检测器

   while (!(OSCXCN & 0x80));          // 等待外部晶体起振
```

```
    OSCICN = 0x08;                  // 切换到外部振荡器

//  POLLED_TEST();                  // 调用查询方式 SW_UART 测试程序
    INTERRUPT_TEST();               // 调用中断方式 SW_UART 测试程序

    while(1);                       // 原地循环
}

//-----------------------------------------------------------------------
// 函数
//-----------------------------------------------------------------------

//-----------------------------------------------------------------------
// INTERRUPT_TEST: SW_UART 中断方式测试
// 测试代码用 SW_UART 中断方式向 HW_UART 发送 15 个字符并从 HW_UART 接收 15 个
// 字符（在中断方式）。
// - 初始划并允许 SW_UART & HW_UART
// - 清 0 所有测试变量和计数器
// - 从 HW_UART 向 SW_UART 发送 15 个字符，同时从 SW_UART 向 HW_UART 发送 15 个
//   字符
//
void INTERRUPT_TEST(void){

    SW_UART_INIT();                 // 初始化 SW_UART
    SW_UART_ENABLE();               // 允许 SW_UART
    SREN = 1;                       // 允许 SW_UART 接收器
    SES = 1;                        // 允许用户级中断支持

    HWU_INIT();                     // 配置 HW_UART，用于测试程序

    k=m=0;                          // 清 0 用户 ISR 计数器

    HW_DONE=0;                      // 清 0 传输完成标志
    SW_DONE=0;                      //

    IE |= 0x10;                     // 允许 HW_UART 中断。
    STI=1;                          // 指示发送完成以启动下一次发送。

    EIE2 |= 0x20;                   // 通过允许并强制一个 IE7 中断来
    PRT1IF |= 0x80;                 // 启动软件发送（SW_TX）。

TI = 1;                             // 通过强制 TX 中断启动一次
                                    // HW_UART 发送。

    while(!(HW_DONE&SW_DONE));      // 等待发送结束。
}

//-----------------------------------------------------------------------
// POLLED_TEST: SW_UART 查询方式测试
// 测试代码用 SW_UART 查询方式向 HW_UART 发送 15 个字符并从 HW_UART 接收 15 个
```

```
// 字符。
// - 初始化并允许 SW_UART & HW_UART
// - 清 0 所有测试变量和计数器
// - 从 HW_UART 发送 15 个字符，由 SW_UART 接收。
// - 从 SW_UART 发送 15 个字符，由 HW_UART 接收
//
void POLLED_TEST(void){

    SW_UART_INIT();                     // 初始化 SW_UART
    SW_UART_ENABLE();                   // 允许 SW_UART
    SREN = 1;                           // 允许 SW_UART 接收器
    SES=0;                              // 禁止用户级中断支持

    HWU_INIT();                         //配置 HW_UART，用于测试程序
    k=m=0;                              // 清 0 测试计数器变量
    HW_DONE=0;                          // 清 0 传送完成标志
    SW_DONE=0;                          //
    IE |= 0x10;                         // 允许 HW_UART 中断

    TI = 1;                             // 通过强制 TX 中断来启动一次
                                    // HW_UART 发送。

    // 用 SW_UART 接收 15 个字符；用 HW_UART 发送。
    while(SREN){                        // 在 SW_UART 被允许时执行。
        if (SRI){                       // 如果接收完成：
            SRI = 0;                    // 清除接收标志
            SW_BUF[k++] = RDR;          // 读接收缓冲器
            if (k==15){                 // 如果已收到 15 个字符：
                SREN = 0;               // 禁止 SW_UART 接收器。
            }                           // 指示已收到 15 个字符。
        }
    }

    //用 SW_UART 发送 15 个字符；用 HW_UART 接收。
    while(STXBSY);                      // 查询忙标志
STXBSY = 1;                         // 占用 SW_UART 发送器
    TDR=m++;                            // 装发送数据
    CCF1=1;                             // 通过强制 PCA 模块 1 中断来
                                        // 启动第一次 SW_UART 发送。
    while(!SW_DONE){                    // SW_UART 发送
                                    // HW_UART 接收

        if (STI){                       // 如果发送完成：
            STI = 0;                    // 清除发送标志。
            if (m<16){                  // 发送 15 个字符。
                STXBSY = 1;             // 占用 SW_UART 发送器
            TDR = m++;                  // 发送，变量增 1
                CCF1 = 1;               // 强制模块 1 中断以启动发送
```

```
            }
          else                                  // 如果这是第 15 个字符，
                SW_DONE=1;                      // 表示这是发送的最后一个字符。
          }
      }
}

//-----------------------------------------------------------------------
// HWU_INIT: HW_UART 初始化程序
// 初始化 HW_UART，用于 SW_UART 测试。
// - HW_UART 工作于方式 1
// - 定时器 1 用做波特率源，以 SYSCLK 为时钟。
//
void HWU_INIT(void) {
    PCON |= 0x80;                       // SMOD=1（HW_UART 使用定时器 1
                                        // 溢出，不分频）。
    TMOD = 0x20;                        // 配置定时器 1 为 HW_UART 所用
    CKCON |= 0x10;                      // 定时器 1 用 SYSCLK 作为时钟
    TH1 = -HW_TIME_COUNT;               // 定时器 1 初始值
    TL1 = -HW_TIME_COUNT;               // 定时器 1 重载值
    TR1 = 1;                            // 启动定时器 1
    RI=0;                               // 清除 HW_UART 接收和发送完成标志
    TI=0;                               //
    SCON = 0x50;                        // 配置 HW_UART 为方式 1，允许接收
}

//-----------------------------------------------------------------------
// SW_UART_INIT: SW_UART 初始化程序
// 初始化 SW_UART。
// - 配置 PCA：模块 0 为负沿捕捉方式；模块 1 为软件定时器方式；
//   PCA 时基 = SYSCLK/4；禁止 PCA 中断；禁止 PCA 计数器
// - 清除 PCA 模块 0 和模块 1 中断
// - 复位 TX 和 RX 状态变量
//
void   SW_UART_INIT(void){
    PCA0CPM0 = 0x10;                    // 模块 0 为负沿捕捉方式；
                                        // 禁止模块 0 中断。
    PCA0CPM1 = 0x48;                    // 模块 1 为软件定时器方式；
                                        // 禁止模块 1 中断。
    PCA0CN = 0;                         // PCA 保持禁止状态
    PCA0MD = 0x02;                      // PCA 时基 = SYSCLK/4；
                                        // 禁止 PCA 计数器中断。
    CCF0 = 0;                           // 清除 PCA 模块 0 和模块 1
    CCF1 = 0;                           // 捕捉/比较中断。
    SRI = 0;                            // 清除接收完成标志。
```

```
    STI = 0;                            // 清除发送完成标志。
SW_TX = 1;                             // 将 TX 线初始化为高电平
    STXBSY = 0;                         // 清除 SW_UART 忙标志
}

//-----------------------------------------------------------------------
// SW_UART_ENABLE: SW_UART 允许程序
// 允许 SW_UART
// - 允许 PCA 模块 0 中断
// - 允许 PCA 模块 1 中断
// - 启动 PCA 计数器
//
void   SW_UART_ENABLE(void){
    PCA0CPM0 |= 0x01;                   // 允许 PCA 模块 0（接收）中断。
    PCA0CPM1 |= 0x01;                   // 允许 PCA 模块 1（发送）中断。

    CR = 1;                             // 启动 PCA 计数器
    EIE1 |= 0x08;                       // 允许 PCA 中断
    EA = 1;                             // 全局中断允许
}
//-----------------------------------------------------------------------
// 中断服务程序
//-----------------------------------------------------------------------
//
// PCA_ISR: PCA 中断服务程序。
// 该 ISR 由发送和接收函数触发，在每个发送和接收位都被触发一次
// - 检查模块 0 中断标志(CCF0)；如果置位，服务接收状态。
// - 检查模块 1 中断标志(CCF1)；如果置位，服务发送状态。
//
void   PCA_ISR(void) interrupt 9 {

    static char SUTXST = 0;             // SW_UART TX 状态变量
    static char SURXST = 0;             // SW_UART RX 状态变量
    static unsigned char RXSHIFT;       // SW_UART RX 移位寄存器

    unsigned int PCA_TEMP;              // 临时存储变量，
                                        // 用于处理 PCA 模块的高和低字节

    // 首先检查接收中断变量，如果 CCF0 置位则对其服务。
    if (CCF0){
        CCF0 = 0;                       // 清除中断标志
        switch (SURXST){

            // 状态 0:收到起始位。
            // 在该状态，是 SW_TX 上的负边沿触发的中断，表示检测到起始位，
            // 同时 PCA0CP0 寄存器捕捉 PCA0 的值。
            // - 检查接收允许和起始位
            // - 将 PCA 模块 0 切换到软件定时器方式
```

```
// - 加 3/2 位时间到模块 0 捕捉寄存器以采样 LSB。
// - RX 状态变量加 1。
case 0:
    if (SREN & ~SW_RX){          //检查接收允许和起始位

        PCA_TEMP = (PCA0CPH0<<8);  // 将模块 0 的内容读到
        PCA_TEMP |= PCA0CPL0;      // PCA_TEMP。

        PCA_TEMP += TH_TIME_COUNT; // 加 3/2 位时间到 PCA_TEMP

        PCA0CPL0 = PCA_TEMP;       // 更新 PCA0CPL0 和
        PCA0CPH0 = (PCA_TEMP >> 8);// PCA0CPH0

        PCA0CPM0 = 0x49;           // 将模块 0 切换到软件定时器
                                   // 方式，允许中断。

        SURXST++;                  // 更新 RX 状态变量。
    }
    break;

// 状态 1-8：收到数据位
// - 采样 SW_RX 引脚
// - 将新数据位移入 RXSHIFT
// - 加 1 个位时间到模块 0 捕捉寄存器
// - RX 状态变量增 1
case 1:
case 2:
case 3:
case 4:
case 5:
case 6:
case 7:
case 8:

    RXSHIFT = RXSHIFT >> 1;    // 右移一位
    if (SW_RX)                 // If SW_RX=1，
        RXSHIFT |= 0x80;       // 将‘1’移入 RXSHIFT 的 MSB

    PCA_TEMP = (PCA0CPH0 << 8);// 将模块 0 内容读到 PCA_TEMP。
    PCA_TEMP |= PCA0CPL0;      //

    PCA_TEMP += TIME_COUNT;    // 加一个位时间到 PCA_TEMP

    PCA0CPL0 = PCA_TEMP;       // 更新 PCA0CPL0 和 PCA0CPH0
    PCA0CPH0 = (PCA_TEMP >> 8);

    SURXST++;                  // 更新 RX 状态变量。
    break;

// 状态 9：已收到 8 个数据位，捕捉停止位。
// - 将 RXSHIFT 传送到 RDR。
```

```
    // - 置位 SRI （表示接收完成）。
    // - 设置模块 0，为下一次传输做准备。
    // - 复位 RX 状态变量
    // - 触发 IE7（如果用户级中断支持被允许）
    case 9:

        RDR = RXSHIFT;              // 将接收到的数据传送到接收寄存器。
        SRI = 1;                    // 置‘1’接收完成标志。

        PCA0CPM0 = 0x11;            // 切换模块 0 到负沿捕捉方式；
                                    // 允许中断以检测起始位。

        SURXST = 0;                 // 复位 RX 状态变量

        if (SES){                   //如果用户级中断支持被允许
            EIE2 |= 0x20;           // 允许 IE7.
            PRT1IF |= 0x80;         // 触发 IE7.
        }
        break;

    }
}

// 检查发送中断，如果 CCF1 置位则对其服务。
else if (CCF1){
    CCF1 = 0;                       // 清除中断标志
    switch (SUTXST){

    // 状态 0：发送过程已启动
    // 在此，用户已将要发送的字节装入到 TDR，强制模块 1 中断以启动发送。
    // - 发送起始位（使 SW_TX 变低）
    // - 读 PCA0，加一个位时间后存到模块 1 捕捉寄存器。
    // - TX 状态变量增 1。
        case 0:

            SW_TX = 0;                  // 使 TX 引脚变低作为起始位

            PCA_TEMP = PCA0L;           // 将 PCA 计数器的值读到
            PCA_TEMP |= (PCA0H << 8); // PCA_TEMP

            PCA_TEMP += TIME_COUNT; // 加一个位时间

            PCA0CPL1 = PCA_TEMP;    // 将更新后的匹配值存到模块 1
            PCA0CPH1 = (PCA_TEMP >> 8);// 的捕捉比较寄存器

            PCA0CPM1 |= 0x48;           // 允许模块 1 软件定时器

            SUTXST++;                   // 更新 TX 状态变量
            break;

        // 状态 1-9：发送数据位
```

```
// - 将 TDR 的 LSB 输出到 TX
// - 将 TDR 右移一位
// - 将一个‘1’移入 TDR 的 MSB 作为状态 9 的停止位
// - 加一个位时间到模块 1 捕捉寄存器
case 1:
case 2:
case 3:
case 4:
case 5:
case 6:
case 7:
case 8:
case 9:

    SW_TX = (TDR & 0x01);// 将 TDR 的 LSB 输出到 SW_TX 引脚。
    TDR >>= 1;                // TDR 右移一位。
    TDR |= 0x80;              // 将一个‘1’移入 TDR 的 MSB
                              // 作为状态 9 的停止位

    PCA_TEMP = (PCA0CPH1 << 8);// 将模块 1 内容读到 PCA_TEMP
    PCA_TEMP |= PCA0CPL1;

    PCA_TEMP += TIME_COUNT;     // 加一个位时间到 PCA_TEMP

    PCA0CPL1 = PCA_TEMP;        // 更新 PCA0CPL1
    PCA0CPH1 = (PCA_TEMP >> 8);// 和 PCA0CPH1

    SUTXST++;                   // 更新 TX 状态变量。
    break;

// 状态 10：最后一位数据已发送完。发送停止位并结束传输过程。
// - 发送停止位
// - 置‘1’发送结束标志，清除忙标志
// - 复位 TX 状态
// - 设置模块 1，为下一次传输做准备。
// - 触发 IE7（如果用户级中断支持被允许）
case 10:

    STI = 1;                  // 表示发送完成。
    SUTXST = 0;               // 复位 TX 状态。
    SW_TX = 1;                // SW_TX 应保持高电平。

    PCA0CPM1 = 0x01;          // 禁止模块 1 软件定时器，
                              // 保持中断为允许状态，以备下一
                          // 次传输

    if (SES){                 //如果用户级中断支持被允许：
        EIE2 |= 0x20;         // 允许 IE7
        PRT1IF |= 0x80;       // 触发 IE7
    }
```

```
                    STXBSY = 0;                 // SW_UART TX 空闲
                    break;
                }
        }
}

//-------------------------------------------------------------------
// USER_ISR: 用户 SW_UART 中断服务程序(IE7 ISR)
// 如果中断测试方式被允许，该 ISR 将发送 15 个字符并接收 15 个字符。
// 每次 SW_UART 发送或接收完成都要触发该例程。
// - 检查接收完成指示标志并服务。
// - 检查发送完成指示标志并服务。
// - 检查 ISR 执行期间是否有发送和接收过程完成；如有，再触发一次中断。
//
void USER_ISR(void) interrupt 19 {          // IE7 中断服务程序

    PRT1IF &= ~(0x80);                      // 清除 IE7 中断标志

    if (SRI){                               // 如果接收完成:
        SRI = 0;                            // 清除接收标志。
        SW_BUF[k++] = RDR;                  // 读接收缓冲器
        if (k==16){                         // 如果已收到 15 个字符:
            SREN=0;                         // 禁止 SW_UART 接收器。
            }                               // 表示收到 15 个字符。
        }

    else if (STI){                          // 如果发送完成:
        STI = 0;                            // 清除发送标志。

        if (m<15){                          // 如果已发送的字符不足 15 个:
            STXBSY = 1;                     // 占用 SW_UART 发送器。
         TDR = m++;                         // 变量加 1，发送。
         CCF1 = 1;                          // 强制模块 1 中断以启动发送
      }
      else
         SW_DONE=1;                         // 表示已发送完最后一个字符。
    }

    if (STI|SRI)                            // 如果 SRI 或 STI 置位，再次触发
        PRT1IF |= 0x80;                     // 中断服务。
}

//-------------------------------------------------------------------
// HW_UART_ISR: 硬件 UART 中断服务程序
// 发送字符 1-15，并接收 15 个字符。
// - 检查接收中断并服务。
// - 检查发送中断并服务。
//
void   HW_UART_ISR(void) interrupt 4 {
```

```
    static char i=0;                        // 发送数据变量
    static char j=0;                        // 接收数据的下标
    static idata char HW_BUF[20];           // 接收数据缓冲区

    if (RI){                                // 如果接收完成：

        RI=0;                               // 清除接收标志
        HW_BUF[j++] = SBUF;                 // 读接收缓冲区
        if (j==15)                          // 如果已收到 15 个字符：
            REN=0;                          // 禁止 HW_UART 接收器
    }

    else if (TI){                           // 如果发送完成：

        TI = 0;                             // 清除发送标志
        if (i<15)                           // 如果还有要发送的字符：
            SBUF=i++;                       // 变量加 1，发送
        else                                // 如果已发送完 15 个字符，
            HW_DONE=1;                      // 指示 HW TX 结束。

    }
}

// 例 1 结束
```

```
;-----------------------------------------------------------------------
;   COPYRIGHT 2001 CYGNAL INTEGRATED PRODUCTS, INC.
;
;   文件名       : an015_2.ASM
;   目标 MCU     : C8051F000
;   说明         : 软件 UART 示例源码
;
; 实现笔记：
;  - 使用定时器 0 工作于方式 3（两个 8 位定时器）
;  - 定时器 0 运行/溢出用于 RX 状态机
;  -定时器 1 溢出用于 TX 状态机
;  - 8-N-1 格式，无帧错误检测
;  - 使用 IE7 作为用户级中断
;  - 基于状态表的实现方案，使用单字节 PC 偏移量
;  - 使用/INT0 下降沿检测起始位
;
;-----------------------------------------------------------------------

;-----------------------------------------------------------------------
; 等价定义
;-----------------------------------------------------------------------
```

```
$MOD8F000

; SW UART 常量
SW_TX_GPIO  EQU  P0.4       ; SW UART TX GPIO 引脚（可以是任一 GPIO 引脚）
SW_RX_GPIO  EQU  P0.2       ; SW UART RX GPIO 引脚（必须是/INT0）

TIME_COUNT  EQU  320
                            ; 注：320 是保证可靠全双工操作的极限值
                            ; 对于 SYSCLK = 18.432 MHz:
                            ;  115200 = 160
                            ;  57600 = 320
                            ;  38400 = 480
                            ;  19200 = 960
                            ;  9600 = 1920
                            ;  4800 = 3840
                            ;  2400 = 7680

TX_CORR       EQU    41     ; (41) 在发送周期的定时器预装校正值
RX_CORR       EQU    47     ; (47) 在接收周期的定时器预装校正值
THALF_CORR    EQU    113    ; (113) 对于 3/2 RX 的定时器预装校正值

TX_BT      EQU    TIME_COUNT - TX_CORR    ; 实际的 16 位位计数器周期值
                                          ;  TX
RX_BT      EQU TIME_COUNT - RX_CORR       ; 实际的 16 位位计数器周期值
                                          ;  RX
THALF_BT   EQU TIME_COUNT*3/2 - THALF_CORR;实际的 16 位1.5 位计数器周期值
                                            ;  RX

RX_BUFSIZE    EQU    16                     ; 接收缓冲区字符个数
;-----------------------------------------------------------------------
; 变量
;-----------------------------------------------------------------------

BSEG
          org 0h

SRI:      DBIT      1             ; SW UART 接收完成标志
STI:      DBIT      1             ; SW UART 发送完成标志
STXBSY:   DBIT      1             ; SW UART 发送忙标志
SREN:     DBIT      1             ; SW UART 接收允许
SES:      DBIT      1             ; SW UART 用户级中断支持允许

DSEG at 30h

TDR:      DS        1             ; SW UART 发送数据寄存器
RDR:      DS        1             ; SW UART 接收数据寄存器
RXSHIFT:  DS        1             ; SW UART 接收移位寄存器
SURXST:   DS        1             ; SW UART 接收状态变量
SUTXST:   DS        1             ; SW UART 发送状态变量
BCRHI:    DS        1             ; 用于SW UART接收的 16 位位定时器的MSB
BCTHI:    DS        1             ; 用于SW UART发送的 16 位位定时器的MSB
```

```
;测试变量
RX_TAIL:   DS         1              ; 接收消息缓冲区的写指针
TX_VAL:    DS         1              ; 待发送数值
;--------------------
; 间接地址空间变量

ISEG at 80h

RX_BUF:    DS         RX_BUFSIZE     ; 接收消息缓冲区

;--------------------
; 堆栈

STACK_TOP:    DS      1           ; 符号表中的占位符，定义硬件堆栈的起始地址

;-----------------------------------------------------------------------
; 宏定义
;-----------------------------------------------------------------------

;-----------------------------------------------------------------------
; 复位和中断向量表
;-----------------------------------------------------------------------

CSEG
           org    00h
           ljmp   Reset              ; 系统复位初始化向量

           org    03h
           ljmp   INT0_ISR           ; 软件 UART 接收起始位检测

           org    0bh
           ljmp   Timer0_ISR         ; 软件 UART 接收状态机中断

           org    1bh
           ljmp   Timer1_ISR         ; 软件 UART 发送状态机中断

           org    9bh
           ljmp   IE7_ISR            ; 用户级软件 UART 中断

;-----------------------------------------------------------------------
; 主程序代码
;-----------------------------------------------------------------------

           org 0B3h

Main:
           ajmp   PolledRX_PolledTX        ; 这些行中留一行不被注释掉

;          ajmp   InterruptRX_InterruptTX ;

           sjmp   $                        ; 原地跳转

;-----------------------------------------------------------------------
```

```
; 主例程
;-----------------------------------------------------------------------

;-----------------------------------------------------------------------
; PolledRX_PolledTX
;-----------------------------------------------------------------------
; 该例程演示查询方式访问 SW UART。
;
; 发送器发送从$00 到$ff 的序列
;
; 接收器接收字符并存入一个循环缓冲区。
;
PolledRX_PolledTX:
          acall  SW_UART_Init      ; 初始化 SW UART（保持在禁止状态）

          setb   SREN              ; 允许 SW UART 接收器
          clr    SES               ; 禁止用户级中断支持
          acall  SW_UART_Enable    ; 允许 SW UART

          ; 发送消息 - 查询方式
          jb     STXBSY, $         ; 等待 SW TX 可用

          ; 发送字符$00 到$ff
          clr    a
TX_LOOP:  setb   STXBSY            ; 占用 SW UART 发送器
          mov    TDR, a            ; 写字符到发送数据寄存器
          setb   ET1               ; 启动 SW TX 操作
          inc    a                 ; 设置下一个发送值
          jnb    STI, $            ; 等待发送结束
          clr    STI               ; 清除发送完成标志
          jnz    TX_LOOP
TX_LOOP_END:

          mov    RX_TAIL, #RX_BUF ; 初始化尾指针

          ; 接收消息 - 查询方式
RX_LOOP:  mov    r0, RX_TAIL       ; 待写字符的间接地址
          jnb    SRI, $            ; 等待接收字符
          clr    SRI               ; 清除接收完成标志
          mov    @r0, RDR          ; 保存接收值
          inc    RX_TAIL           ; 尾指针加 1
          mov    a, RX_TAIL        ; 处理尾指针回绕
          add    a, #-(RX_BUF + RX_BUFSIZE)
          jnc    RX_LOOP
          mov    RX_TAIL, #RX_BUF ; 尾指针回绕

          sjmp   RX_LOOP           ; 一直重复下去

;-----------------------------------------------------------------------
; InterruptRX_InterruptTX
;-----------------------------------------------------------------------
```

```
; 该例程演示中断访问 SW UART。
;
; 接收器接收字符并存入一个循环缓冲区。
; 发送和接收例程都位于 IE7_ISR 处理程序中
;
InterruptRX_InterruptTX:

          acall  SW_UART_Init     ; 初始化 SW UART（保持在禁止状态）
          setb   SES              ; 允许用户级中断支持
          setb   SREN             ; 允许 SW UART 接收器

          mov    RX_TAIL, #RX_BUF ; 初始化尾指针

          acall  SW_UART_Enable   ; 允许 SW UART

          setb   STI              ; 通过允许并触发 IE7 强制启动 SW UART
          orl    EIE2, #00100000b ; 发送器
          orl    PRT1IF, #10000000b;

          sjmp   $

;-----------------------------------------------------------------------
; 中断向量
;-----------------------------------------------------------------------
;-----------------------------------------------------------------------
; 复位中断向量
;
; 该例程初始化器件和所有外设及变量。
;   - 启动外部振荡器（一旦 XTLVLD 变高，sysclk 将被切换到外部振荡器）
;   - 禁止看门狗定时器
;   - 定义交叉开关和 GPIO 输出方式
;   - 初始化硬件堆栈指针
;   - 初始化中断优先级和中断允许
;     - /INT0
;     - 定时器 0
;     - 定时器 1

Reset:
   mov    OSCXCN, #01100110b   ; 允许晶体振荡器，不分频
                               ; XFCN = '110'（对于 18.432MHz 的晶体）
                               ; 在 XTLVLD 变高后选择外部振荡器，
                               ; XTLVLD 变高表示外部振荡器已经启动
                               ;  并稳定（几百毫秒之后）

       mov    WDTCN, #0deh         ; 禁止看门狗定时器
       mov    WDTCN, #0adh

       ; 初始化交叉开关和端口 I/O
       mov    XBR0, #00000100b     ; 允许 HW UART,在 P0.0 (TX)和 P0.1 (RX)
       mov    XBR1, #10000100b     ; 允许/INT0 在 P0.2; /SYSCLK 在 P0.3
```

```
        mov     XBR2, #01000000b    ; 允许交叉开关/允许上拉
        orl     PRT0CF, #00011101b  ; 允许 P0.0、0.2、0.3、0.4 为推挽方式
                                    ; P0.4 为 SW UART 发送引脚
                                    ; P0.2 为 SW UART 接收引脚
        orl     PRT1CF, #01000000b  ; 允许 P1.6 （目标板 LED）为推挽方式

        mov     SP, #STACK_TOP      ; 初始化堆栈指针（在所用 RAM 单元之后）

        ; 在检查外部振荡器是否稳定之前等待>1 ms
        clr     a
        mov     r0, a               ; 清 r0

        djnz    r0, $               ; 延时~380 us
        djnz    r0, $               ; 延时~380 us
        djnz    r0, $               ; 延时~380 us


OSC_WAIT:
        mov     a, OSCXCN           ; 等待晶体振荡器稳定
        jnb     acc.7, OSC_WAIT

        orl     OSCICN, #00001000b  ; 选择外部振荡器作为系统时钟源
        anl     OSCICN, #NOT(00000100b); 禁止内部振荡器
        orl     OSCICN, #10000000b  ; 允许时钟丢失检测器，必须在选择外部
                                    ;  振荡器作为系统时钟源之后完成。

        setb    EA                  ; 允许全局中断

        ljmp    Main

;-----------------------------------------------------------------------
; Timer0_ISR / INT0_ISR
;
; 这些中断启动并驱动 SW UART 接收状态机
;
SWRX_STATE_TABLE:                   ; 每个表项为 1 个字节
    DB      SWRX_S0 - SWRX_STATE_TABLE ; 等待/起始位检测
    DB      SWRX_S1 - SWRX_STATE_TABLE ; b0
    DB      SWRX_S2 - SWRX_STATE_TABLE ; b1
    DB      SWRX_S3 - SWRX_STATE_TABLE ; b2
    DB      SWRX_S4 - SWRX_STATE_TABLE ; b3
    DB      SWRX_S5 - SWRX_STATE_TABLE ; b4
    DB      SWRX_S6 - SWRX_STATE_TABLE ; b5
    DB      SWRX_S7 - SWRX_STATE_TABLE ; b6
    DB      SWRX_S8 - SWRX_STATE_TABLE ; b7
    DB      SWRX_S9 - SWRX_STATE_TABLE ; 停止位捕捉

INT0_ISR:
Timer0_ISR:
        push    PSW                 ; 资源保护
        push    acc
```

```
        mov     a, BCRHI                ; 如果 BCRHI 不为 0，我们需要继续
                                        ;  等待定时器溢出
        jz      SWRX_PROCESS_STATE
        dec     BCRHI
        ajmp    Timer0_ISR_EXIT

SWRX_PROCESS_STATE:
        push    DPH                     ; 资源保护
        push    DPL

        mov     a, SURXST               ; 从表中读状态偏移量
        mov     DPTR, #SWRX_STATE_TABLE
        movc    a, @A+DPTR              ; 'a' 中现在为状态偏移量(PC)
        jmp     @A+DPTR                 ; 执行该状态

Timer0_ISR_END:                         ; 所有 RX 状态都返回到此

        pop     DPL                     ; 资源恢复
        pop     DPH
Timer0_ISR_EXIT:
        pop     acc                     ; 资源恢复
        pop     PSW
        reti

;SWRX_S0: RX 等待状态
; 在该状态，已经检测到/INT0 上有一个下降沿。
; 我们首先检查 SW UART 接收器是否被允许。如果是，再检查一次 RX 引脚看是否仍为低
; 电平（起始位有效）。如果是，我们初始化定时器 0 使其计 3/2 位时间以接收 LSB。
; 在此，我们还要禁止/INT0 中断。
;   - 检查 SREN = '1':  如果为'1':
;       - 将 3/2 位时间值装入 TL0
;       - 启动定时器
;       - 允许 TF0 中断
;       - 禁止/INT0 中断
;       - 状态变量加 1，转到 S1
;   - 如果 SREN = '0' (SW UART RX 被禁止)
;       - 退出，下一个状态为 S0
;
SWRX_S0:
        jnb     SREN, SWRX_S0_END       ; 检查 SW UART RX 是否被允许
                                        ;  如果未允许，退出并停在等待状态

        jb      SW_RX_GPIO, SWRX_S0_END ; 检查是否为真正的起始位

        clr     EX0                     ; 禁止/INT0

        clr     TR0                     ; 停止定时器 0（低）
        clr     TF0                     ; 清除任何中断

        mov     BCRHI, #HIGH(THALF_BT); 设置定时器 0（低）+BCRHI，从现在
```

```
        mov     TL0, #-LOW(THALF_BT)    ;  起 1.5 位的时间(我们假设起始位
                                        ;  有效)

        setb    ET0                     ; 允许定时器 0 中断
        setb    TR0                     ; 启动定时器 TL0

        inc     SURXST                  ; 下一个状态为 SWRX_S1(我们假设起始位
                                        ;  有效)

SWRX_S0_END:
        ajmp    Timer0_ISR_END

;SWRX_S1 到 SWRX_S8: 接收 b0..b7
; 到此为止，我们已经确定起始位有效，我们将在位间隔时间查询 RX_GPIO，
; 将结果移入 RXSHIFT。
;   - 如果 BCRHI 不为 0，我们需要继续等待定时器溢出
;       - BCRHI 减 1
;       - 等待定时器重新溢出
;       - 保持现状态
;   - 如果 BCRHI 为 0:
;       - 停止定时器
;       - 将 RX_GPIO 状态传送到进位位
;       - 右移进位位到 RXSHIFT
;       - 设置定时器以接收下一位
;       - 允许定时器
;       - 更新状态变量
;
SWRX_S1:
SWRX_S2:
SWRX_S3:
SWRX_S4:
SWRX_S5:
SWRX_S6:
SWRX_S7:
SWRX_S8:
        clr     TR0                     ; 停止定时器 0（低）
        clr     TF0                     ; 清除任何中断

        mov     BCRHI, #HIGH(RX_BT); 将位时间值装入 16 位虚拟计数器
        mov     TL0, #-LOW(RX_BT)

        setb    TR0                     ; 启动 RX 位定时器

        mov     C, SW_RX_GPIO           ; 在右移之前将 RX 状态移入进位标志

        mov     a, RXSHIFT
        rrc     a                       ; 右移，将进位标志移入移位寄存器
        mov     RXSHIFT, a              ; 重存

        inc     SURXST                  ; 状态变量加 1
```

```
SWRX_S2_END:
       ajmp   Timer0_ISR_END

;SWRX_S9：捕捉停止位
; 到此为止，我们已经将所有数据位移入 RXSHIFT，我们已准备好采样停止位。
; 在此，我们表明已收到一个字符，并将状态机复位到等待状态。在本例中，我们实际上
; 没有捕捉停止位；我们假设停止位是有效的。此处应是我们加入帧错误检测的地方。
;   -如果 BCRHI 不为 0，我们需要等待定时器重新溢出
;       - BCRHI 减 1
;       - 等待定时器重新溢出
;       - 保持现状态
;
;   - 如果 BCRHI 为 0：
;       - 停止定时器
;       - 将数据从移位寄存器传送到数据寄存器
;       - 置位 SRI
;       - 禁止定时器中断
;       - 允许/INT0 中断
;       - 复位状态变量到等待状态
;       - 检查用户级中断是否被允许(EIS)，如是：
;           - 允许 IE7
;           - 切换 P1.7 以触发 IE7
SWRX_S9:
       clr    TR0                  ; 停止定时器 TL0
       mov    RDR, RXSHIFT         ; 将数据从移位寄存器保存到数据寄存器

       setb   SRI                  ; 置‘1’SW UART SRI 位表示接收完成
       clr    ET0                  ; 禁止定时器 TL0 中断
       clr    IE0                  ; 禁止挂起的/INT0 中断
       setb   EX0                  ; 允许/INT0 中断
       mov    SURXST, #00          ; 复位 RX 状态到等待状态

       jnb    SES, SWRX_S9_END     ; 检查用户级中断是否被允许
       orl    EIE2, #00100000b     ; 允许 IE7;
       orl    PRT1IF, #10000000b   ; 触发 IE7
SWRX_S9_END:
       ajmp   Timer0_ISR_END

;-----------------------------------------------------------------------
; Timer1_ISR（注意，该 ISR 实际上由定时器 0 的高半部分调用，定时器 0 工作于方式 3）
;
; 该中断驱动 SW UART 发送状态机
;
SWTX_STATE_TABLE:                   ; 每个表项为 1 个字节；共 11 个表项
   DB     SWTX_S0  - SWTX_STATE_TABLE    ; 起始位
   DB     SWTX_S1  - SWTX_STATE_TABLE    ; b0
   DB     SWTX_S2  - SWTX_STATE_TABLE    ; b1
   DB     SWTX_S3  - SWTX_STATE_TABLE    ; b2
```

```
    DB      SWTX_S4  - SWTX_STATE_TABLE    ; b3
    DB      SWTX_S5  - SWTX_STATE_TABLE    ; b4
    DB      SWTX_S6  - SWTX_STATE_TABLE    ; b5
    DB      SWTX_S7  - SWTX_STATE_TABLE    ; b6
    DB      SWTX_S8  - SWTX_STATE_TABLE    ; b7
    DB      SWTX_S9  - SWTX_STATE_TABLE    ; 停止位开始
    DB      SWTX_S10 - SWTX_STATE_TABLE    ; 停止位结束

Timer1_ISR:
       push   PSW                    ; 资源保护
       push   acc

       mov    a, BCTHI               ; 如果 BCTHI 不为 0，我们需要
                                     ;  等待定时器重新溢出...
       jz     SWTX_PROCESS_STATE
       dec    BCTHI
       ajmp   Timer1_ISR_EXIT

SWTX_PROCESS_STATE:
       push   DPH                    ; 资源保护
       push   DPL

       mov    a, SUTXST              ; 从表中读状态偏移量
       mov    DPTR, #SWTX_STATE_TABLE
       movc   a, @A+DPTR             ; acc 现在包含状态偏移量
       jmp    @A+DPTR                ; 执行状态 x

Timer1_ISR_END:                      ; 所有 TX 状态返回到此

       pop DPL                       ; 资源恢复
       pop DPH
Timer1_ISR_EXIT:
       pop acc                       ; 资源恢复
       pop PSW

       reti

;SWTX_S0: TX 起始位状态
; 到此为止，用户程序已经将待发送字符装入 TDR 并通过置位 TF1 调用了定时器 1 中断
; 处理程序。
;   - 清除 STI
;   - 使 TX_GPIO 变低(起始位开始)
;   - 为下一个位时间配置 TH0、BCTHI，下一位将是 LSB
;   - 允许 TH0
;   - 设置下一个状态为 SWTX_S1
;
SWTX_S0:
       mov    BCTHI, #HIGH(TX_BT) ; 将位时间值装入 16 位虚拟计数器
       mov    TH0, #-LOW(TX_BT)
       clr    SW_TX_GPIO             ; 起始位开始
       clr    TF1                    ; 清除任何挂起的中断
```

```
        inc     SUTXST                  ; 下一个状态是 SWTX_S1
SWTX_S0_END:
        ajmp    Timer1_ISR_END

;SWTX_S1 到 SWTX_S9: 发送 b0..b7 和停止位
; 至此，我们开始将 TDR 中的字符逐位移出到 TX_GPIO 引脚，每移一位状态变化一次。
; 我们移入一个额外的‘1’到 MSB 作为停止位。
;   -如果 BCTHI 不为 0，我们需要等待定时器重新溢出。
;       - BCRHI 减 1
;       - 等待定时器重新溢出
;       - 保持现状态
;
;   - 如果 BCRHI 为 0:
;       - 停止定时器
;       - 为下一位设置定时器
;       - TDR 右移
;       - 允许定时器
;       - 输出位
;       - 状态变量加 1
;
SWTX_S1:
SWTX_S2:
SWTX_S3:
SWTX_S4:
SWTX_S5:
SWTX_S6:
SWTX_S7:
SWTX_S8:
SWTX_S9:
        mov     BCTHI, #HIGH(TX_BT); 将位时间值装入 16 位虚拟计数器
        mov     TH0, #-LOW(TX_BT)

        mov     a, TDR                  ; 将下一位右移到进位标志
        setb    C                       ; 将停止位移入 MSB
        rrc     a
        mov     TDR, a                  ; 重存
        mov     SW_TX_GPIO, C           ; 输出一位到 GPIO 引脚
        clr     TF1                     ; 清除任何挂起的中断

        inc     SUTXST                  ; 进入下一状态

SWTX_S1_END:
        ajmp    Timer1_ISR_END

;SWTX_S10 停止位完成/复位到等待状态
; 到此为止，我们已经移出了停止位，我们已准备好对状态机复位和指示发送完成，
; 包括触发一个用户级中断（如果被允许）。
;   -如果 BCTHI 不为 0，我们等待定时器重新溢出。
;       - BCRHI 减 1
;       - 等待定时器重新溢出
```

```
;       - 保持现状态
;   - 如果 BCRHI 为 0:
;       - 停止定时器
;       - 置位 STI
;       - 清除 STXBSY
;       - 检查 IE7 支持，如果被允许则触发中断
;       - 设置状态变量为 S0
;
SWTX_S10:
       clr    ET1                    ; 禁止定时器 1 中断
       setb   TF1                    ; 强制一个定时器 1 中断。这允许
                                     ;  立即触发一次发送操作

       mov    SUTXST, #00h           ; 复位状态变量到等待状态
       setb   STI                    ; 置‘1’STI，表示发送完成
       clr    STXBSY                 ; 清除 TXBSY，表示发送器可用
       jnb    SES, SWTX_S10_END      ; 触发用户级中断 IE7（如果被允许）
       orl    EIE2, #00100000b       ; 允许 IE7
       orl    PRT1IF, #10000000b     ; 触发 IE7
SWTX_S10_END:
       ajmp   Timer1_ISR_END

;-----------------------------------------------------------------------
; IE7_ISR
;
; 这是 SW UART 的用户级中断处理程序。注：该程序必须检查 SRI 和 TRI，在两者
; 都为 1 时必须先处理一种情况，然后重新触发 IE7 以处理另一情况（或在同一次调用中
; 处理）。如果只有一种情况需要中断处理则不需要这样做，例如，RX 由中断处理，而 TX
; 用查询处理。
;
; 注意，如果 TX 情况用查询方式处理，则此处不应清除 STI。
;
; 在本例中，如果 SRI 置位，表示 SW UART 已收到一个字符，该字符被保存到一个循环
; 缓冲区（RX_BUF）。如果 STI 置位，表示发送完成，保存在 TX_VAL 的字符被发送。
;
;
IE7_ISR:
       push   PSW
       push   acc

       anl    PRT1IF, #NOT(10000000b); 清 IE7
       jbc    SRI, SW_RX_HANDLE      ; 首先处理接收，因为接收对延迟最敏感
       jbc    STI, SW_TX_HANDLE      ; 处理发送
IE7_ISR_END:
       pop    acc
       pop    PSW
       reti                          ; 所有 IE7_ISR 例程返回到此
SW_RX_HANDLE:
       push   ar0                    ; 资源保护
```

```
        mov     r0, RX_TAIL             ; 使 r0 指向保存地址
        mov     @r0, RDR                ; 将数据读到缓冲区
        inc     RX_TAIL                 ; 更新尾指针
        mov     a, RX_TAIL              ; 必要时指针回绕
        add     a, #-(RX_BUF+RX_BUFSIZE)
        jnc     SW_RX_HANDLE_END
        mov     RX_TAIL, #RX_BUF        ; 指针回绕

SW_RX_HANDLE_END:
        jnb     STI, NO_TX_PENDING      ; 如果有挂起的 TX 中断
        orl     PRT1IF, #10000000b      ;  触发该中断（IE7）

NO_TX_PENDING:
        pop     ar0
        ajmp    IE7_ISR_END

SW_TX_HANDLE:
    Setb    STXBSY                  ; 占用 SW UART 发送器
        mov     TDR, TX_VAL             ; 将字节装入 TDR
        setb    ET1                     ; 启动 SW UART 发送器
        inc     TX_VAL                  ; 下一个字节

SW_TX_HANDLE_END:
        jnb     SRI, NO_RX_PENDING      ; 如果有挂起的 RX 中断，
        orl     PRT1IF, #10000000b      ;  触发该中断（IE7）
NO_RX_PENDING:
        ajmp    IE7_ISR_END             ; 中断返回

;-------------------------------------------------------------------------
; 子程序
;-------------------------------------------------------------------------

;-------------------------------------------------------------------------
; SW UART 子程序(非用户代码)
;-------------------------------------------------------------------------

;-------------------------------------------------------------------------
; SW_UART_Init
;
; 初始化:
;  - /INT0 为下降沿触发
;  - 定时器 0 工作于方式 3（两个 8 位定时器），TL0、TH0（TF0、TF1）定时器的
;    中断处理程序被禁止。
;  - RX/TX 状态机和状态变量
;  - SW UART TX 状态机和 RX 状态机工作在高优先级
SW_UART_Init:
        ; 初始化/INT0
        clr     EX0                     ; 禁止/INT0 中断
        setb    IT0                     ; /INT0 为下降沿触发
        clr     IE0                     ; 清除/INT0 中断标志
        setb    PX0                     ; /INT0 为高优先级中断
```

```
        ; 初始化定时器 0
        clr     ET0                     ; 禁止定时器 0 中断
        clr     ET1                     ; 禁止定时器 1 中断
        clr     TR0                     ; 停止定时器 0
        clr     TR1                     ; 停止定时器 1
        clr     TF0                     ; 清除中断标志
        clr     TF1
        orl     TMOD, #00000011b        ; 定时器 0 工作于方式 3（两个 8 位定时器）
        anl     TMOD, #NOT(00001100b); GATE0=0, C/T0 = 0
        orl     CKCON, #00001000b       ; 定时器 0 使用系统时钟作为时间基准
        setb    PT0                     ; 定时器 0 中断为高优先级
        setb    PT1                     ; 定时器 1 中断为高优先级

        ; 用户级中断（IE7）由状态机初始化

        ; 初始化状态机和状态变量
        clr     a                       ; 初始化状态机
        mov     SURXST, a               ;  RX 状态变量
        mov     SUTXST, a               ;  TX 状态变量
        mov     BCRHI, a                ;  RX 位定时器 MSB
        mov     BCTHI, a                ;  TX 位定时器 MSB
        clr     SES                     ; 禁止用户级中断支持
        clr     SREN                    ; 禁止 SW UART 接收器
        clr     TXBSY                   ; 清除 TXBSY 标志
        clr     SRI                     ; 清除 RX 完成标志
        clr     STI                     ; 清除 TX 完成标志

        ret

;-----------------------------------------------------------------------
; SW_UART_Enable
;
; 通过允许中断处理程序，使发送和接收状态机从等待状态转到相对应的下一状态
; 来允许 SW_UART。/INT0 使 RX 状态机从等待状态转到起始状态。由用户代码
; 设置定时器 1（setb TF1）使发送状态机从等待状/起始状态转到 TX_LSB。
;
; 用户级中断（IE7）是在状态机查询了 EIS（外部中断支持）后被允许的。
;
SW_UART_Enable:
        clr     IE0                     ; 清除挂起的/INT0 中断
        setb    TF1                     ; 产生一个定时器 1 中断
        setb    EX0                     ; 允许/INT0 中断
        clr     ET1                     ; 禁止定时器 1 中断
        setb    TR1                     ; 启动定时器 1

        ret
;-----------------------------------------------------------------------
; SW_UART_Disable
```

```
;
; 通过禁止所有的状态机中断，包括用户级中断 IE7（如果状态寄存器指示该中断已被
; 允许），来禁止 SW UART。
SW_UART_Disable:
       clr    EX0                    ; 禁止/INT0 中断
       clr    ET0                    ; 禁止定时器 0 中断
       clr    ET1                    ; 禁止定时器 1 中断
       jnb    SES, SW_UART_Dis_End; 检查 IE7 是否被允许
       anl    EIE2, #NOT(00100000b); 禁止 IE7 中断
SW_UART_Dis_End:
       ret

;-----------------------------------------------------------------------
; 文件结束。

; 例 2 结束。

END
```