MSP-430USCI 模块情况介绍

一、MSP430 的串口通讯模块（USART）

msp430f541x、msp430f543x 多达 4 个通用串行通信接口（USCI）模块，支持多种串行通信模式，不同的 USCI 模块支持不同的模式。

- USCI_Ax 模块支持：
- UART 模式；
- IrDA 通信的脉冲整形；
- LIN 通信的自动波特率检测；
- SPI 模式；
- USCI_Bx 模块支持：
- IIC 模式；
- SPI 模式；

由于本设计解决的是串口通讯问题，所以通信的基本原理是利用 MSP430 的串口通讯模块（USART）来实现单片机和 PC 机之间的串口通信。

1、USART 的硬件构成：

SP430F169 的串行通讯模块（USART）的作用主要是实现对外通信，它可以实现异步通信（UART）和同步通信（SPI）两中通讯功能[5]。图 1-2 是 USART 的通讯模块。

由图 1-2 可以看出 USART 模块分别由波特率部分，接收部分，发送部分，端口 IO 部分组成。USART 接收部分包括接收寄存器，接收移位寄存器以及控制模块组成，它在接收信息的时候产生一些状态信息，并设置相应的中断标志位。USART 的发送部分包括发送寄存器，发送移位寄存器以及控制模块组成，它在发送的时候产生一些状态信息，并可以设置发送中断标志位。USART 的波特率产生部分主要包括时钟的选择，波特率的产生以及波特率的调整部分组成，它通过设置波特率寄存器和波特率调整寄存器来获得需要的波特率。USART 包含一个控制模块，通过控制模块可以选择相应的工作模式，同时设置相应的管脚，比如对异步和同步工作方式的选择，对奇偶校验位和停止位个数等所有设置都是通过操作该模块的寄存器来实现的。对于不同系列的 MSP 单片机其 USART 模块可能有一个也可能有两个，而 MSP430F149 有两个，分别是 USART0 和 USART1。

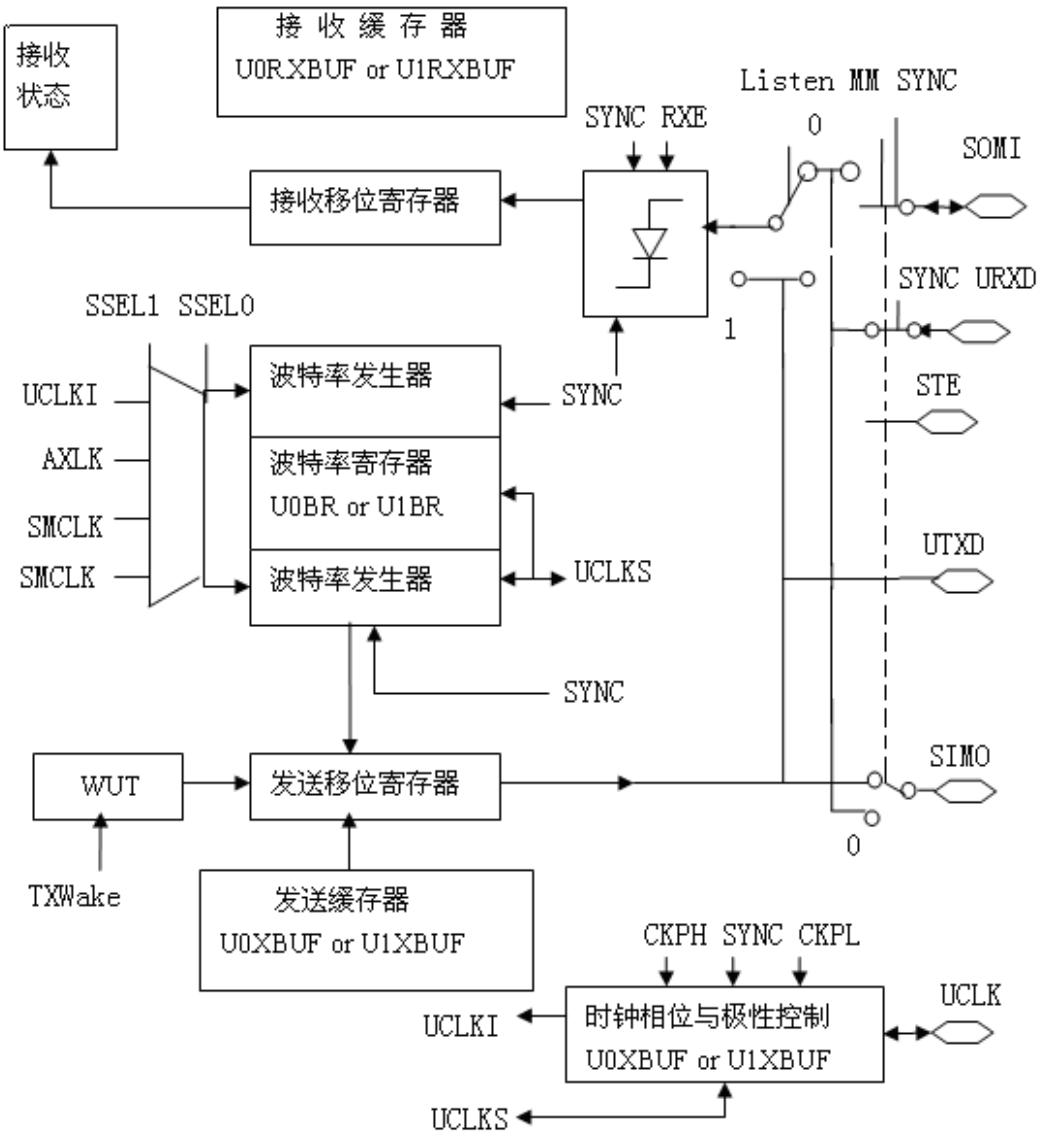


图 1-2 USART 模块组成

大多数 MSP430 芯片都有硬件异步通讯功能，有一些器件有两个通讯端口，也有少数没有。没有硬件串口的芯片可以实现软件（模拟）串口。下面表格为 430 系列芯片串口的情况。

系列芯片	F11 系列	F12 系列	F13 系列	F14 系列	F15 系列	F16 系列
串口数量	0	1	1	2	1	2

芯片系列	F2 系列	C31 系列	C32 系列	C33 系列	F41 系列	F42 系列
串口数量	1	0	0	1	0	1

芯片系列	FW42 系列	FE42 系列	FG43 系列	F43 系列	F44 系列	
串口数量	0	1	1	1	2	

2、USART 的控制寄存器和工作模式

USART 的控制寄存器是其八个寄存器之中的一个，表 1-1 是其位的格式，

表 1-1 控制寄存器

PENV	PEV	SP	CHAR	LISTEN	SYNC	MM	SWRST

USART 的控制寄存器有 8 个有效控制位，通过对这些控制位的设置可以对工作模式，通信协议，校验位等进行选择。用户对 USART 的所有操作都是通过操作该寄存器的控制位来完成的[6]。下面是各个位的简单功能描述,知道这些控制位的功能,有助于我们在后面进行硬件连接和软件设计.

- PENV：校验使能位。该位为 0 不允许校验；为 1 时，允许校验，且在发送时产生校验位，在接收时希望接收到校验位。在地址位多机模式中地址位包括在校验计算中。
- PEV：奇偶校验位。为 0 时，奇校验，为 1 时进行偶校验。
- SP：停止位。接收时停止位只有一个。发送时，该位为 0，只有一个停止位；该位为 1 时，有两个停止位。
- CHAR：字符长度位。该位为 0 表示发送的数据为 7 位，该位为 1 时表示发送的数据为 8 位。
- LISTEN：监听使能位。该位为 0 没有反馈；该位为 1，有反馈，发送的数据送到接收器，可以进行自环测试。
- SYNC：该位为 0 时，USART 为异步通信（UART）模式；该位为 1，USART 为同步通信（SPI）模式。
- MM：多机模式选择。当该位为 0 时，多机模式选择线路空闲多机协议；该位为 1 时，多机模式选择地址位多机协议。
- SWRST：软件复位使能位。也叫控制位。该位影响着其他控制位和状态位的状态，在串行口的使用过程中，这一位比较重要。一次正确的 USART 模块初始化应该是这样的顺序：先在 SWRST=1 的情况下设置串口；然后设置 SWRST=0；最后如果使用中断，则设置相应的中断使能。该位为 0 时：USART 模块被允许。该位为 1 时：如果该位置位，则 USART 状态机和操作运行标志位都被初使化成复位状态（URXIFG=URXIE=UTXIE=0，UTXIFG=1）；同时所受影响的逻辑位保持在复位状态，直到 SWRST 位复位。这意味着，当系统复位后，只有对 SWRST 位复位，USART 的功能才能被重新允许；但是接收和发送标志 URXE 和 UTXE 不受 SWRST 控制位的影响。

2.1 串口通信设置的考虑

MSP430 成功一个实现最简单串口通信，必须同时具备以下几个条件：

- 工作 UART 于模式
- 确定适当的数据格式
- 确定适当的波特率

以上条件须通过正确的设置 MSP430 的 USART 模式及其有关寄存器的参数配置，才能得以实现。

2.1.1 设置 UART 工作模式

由于 MSP430 的 USART 外围模块,可以工作在异步(UART)或同步模式(SPI)下，因此，实现 UART 通信，必须使得模块工作在 UART 模式，仅仅只需要清除 UxCTL 寄存器的位 2（SYNC）到 0，否则将工作于同步模式下。那么，UART 的

工作模式就确立了。

UxCTL寄存器	7	6	5	4	3	2	1	0
	PENA	PEV	SPB	CHAR	LISTEN	SYNC	MM	SWRST

2.1.2 设置数据格式

根据需要的数据格式，进行有关设置，这里以 8-N-1 为例，即，启动 1 位，数据位 8 位，停止 1 位，无校验位。

UxCTL寄存器	7	6	5	4	3	2	1	0
	PENA	PEV	SPB	CHAR	LISTEN	SYNC	MM	SWRST

设置数据 8 位模式，需要设置 UxCTL 寄存器的位 4（CHAR）为 1，否则，数据位是 7 位模式。

无校验模式，需要设置需要清除 UxCTL 寄存器的位 8（PENA）为 0，不允许校验模式；反之，如果为 1，则允许校验模式，那么，还需要确定是偶校验还是奇校验，这个选择也是在这个寄存器的位 6（PEV）设置的。0 是偶校验，1 是奇校验。

2.1.3 波特率设置

a） 时钟源选择

图 1 是波特率发生器时钟源的路径简图， 用户根据实际需要选择途中的三者之一。

途中， SSEL1 和 SSEL0 是 UxTCTL 寄存器的位 4 和位 5， 时钟源的选择如下表所示。

UxTCTL寄存器	7	6	5 4	3	2	1	0
	Unused	CKPL	SSELx	URXSE	TXWAKE	Unused	TXEPT

SSEL1	SSEL0	选中的时钟源	描述
0	0	UCLKx	来自于 UCLKx 引脚的外部时钟输入
0	1	ACLK	来自于基础系统时钟（BSC）
1	0	SMCLK	来自于基础系统时钟的子系统主时钟
1	1	SMCLK	来自于基础系统时钟的子系统主时钟

本例选用 SMCLK 作为波特率发生器的时钟源，频率为 8MHz。

b） 波特率 115200bps 计算和设置

通过设置三个与波特率有关的寄存器，即波特率整数 UxBR1，UxBR0 以及波特率修整寄存器 UxMCTL,使用以下公式计算波特率控制参数：

波特率参数（UxBR1，UxBR0）= 波特率时钟源频率/所需波特率。（结果取整数部分）这里，波特率时钟源频率=8MHz，所需波特率=115200，那么波特率参数（UxBR1，UxBR0）就是 8000(KHz)/115.2(K)=69.44444444,取整后为 69，小数部分 0.44444444 作为修整寄存器的参数，使用如下公式计算

波特率修整参数（UxMCTL）=8×小数部分（结果取整数部分）

那么，应该是 8×0.44444444=3.5,那么，可以取 3 或 4，本例取 4。因此，需要 4 个脉冲插补位，在 UxMCTL 寄存器中，任意选择 4 位后，UxMCTL=0x55。注意，尽量将插补脉冲分散放置到 UxMCTL 8 个中。

至此，串口的参数设置完成，下一步是通过编程来调试并实现串口的通信。

2.2 串口发送和接收编程考虑

MSP430 的 UART 发送和接收是相当有特色的，发送和接收都具有自己独立的发送和接收缓冲器。

这部分内容，我们将讨论串口的发送和接收的这些问题。首先，发送和接收可以工作在中断模式或者查询模式，这取决于用户的应用。考虑到，发送总是由 MSP430 发起的，时间和场合都可以得到完全控制，因此，本例采用发送查询的方式。而对于接收，由于是完全不确定的，因此，要可靠接收每个从另外一方发出的数据，必须采用中断方式，当然，最后还是由用户来决定使用何种工作方式。

2.2.1 UART 发送

2.2.1.1 发送允许

UART 发送具有一个移位寄存器和发送数据缓冲器（UxTXBUF），而移位寄存器对于我们是不可见的，因此也不必理会它是如何工作的，仅仅我们需要关心的是 UxTXBUF 以及其它需要我们干预的东西。启动一次发送同时具备的条件是：

1.发送允许位（UTXEx）被设置

2.写一个数据到 UxTXBUF

对于 MSP13，14，15，16 系列芯片而言，UTXEx 位于模块允许寄存器 1（ME1）的位 7（对于串口 0 而言）和模块允许寄存器（ME2）的位 5（对于有第二个串口的芯片而言）

ME1寄存器	7	6	5	4	3	2	1	0
	UTXE0	URXE0						

ME2寄存器	7	6	5	4	3	2	1	0
			UTXE1	URXE1				

如果使用的芯片是 11，12 系列，由于仅具有一个串口，位于模块允许寄存器（ME2）的位 1。

ME2寄存器	7	6	5	4	3	2	1	0
							UTXE0	URXE0

设置 UTXEx 为‘1’，则允许 UASRT 模块的发送器工作，否则，清零为禁止发送器工作。需要注意的是：当发送数据正在进行时，如果清除 UTXEx 位，那么，此时在发送缓冲器（UxTXBUF）和移位寄存器中的数据不会被清除，而是发送完毕后，模块才会关闭。

2.2.1.2 发送完成标志和中断

当使用查询方式发送数据时，在每次发送之前，为了避免发送错误，必须知道是否现在发送器正在发送或者出于空闲状态，通过查询 UTXIFGx 标志，可以获得发送器的状态，如为 1，表明可以发送新数据，发送缓冲器是空的。否则，必须等待数据缓冲器中数据移入移位寄存器后，由硬件自动产生这个标志，此后新的数据才可以写入 UxTXBUF。

当采用中断发送方式时，在发送完毕后，不论中断是否允许，硬件总是自动设置 UTXIFGx。如果全局中断（GIE）和发送中断（UTXIEx）同时被允许的话，则引起中断产生，当用户程序响应了这个中断后（即进入中断服务程序后），UTXIFGx 会自动被清零，或者写一个数据到 UxTXBUF，也会清除 UTXIFGx 标志。

应当注意的是，在查询方式下，UTXIFGx 不会自动清零，需要用户清除这个标志。UTXIFGx 位于中断标志寄存器（IFGx）的位 7。

对于 MSP13，14，15，16 系列芯片而言，UTXIFGx 位于中断标志寄存器 1（IFG1）的位 7（对于串口 0 而言）和中断标志寄存器（IFG2）的位 5（对于有第二个串口的芯片而言）

IFG1寄存器	7	6	5	4	3	2	1	0
	UTXIFG0	URXIFG0						

IFG2寄存器	7	6	5	4	3	2	1	0
			UTXIFG1	URXIFG1				

如果使用的芯片是 11，12 系列，由于仅具有一个串口，位于模块允许寄存器（IFG2）的位 1。

IFG2寄存器	7	6	5	4	3	2	1	0
							UTXIFG0	URXIFG0

发送中断允许的控制位（UTXIEx）处于也随芯片信号的不同而不同。当被设置为 1 时，允许发送中断，否则不允中断。这些位分布如下：

对于 MSP13,14,15,16 系列芯片而言,UTXIEx 位于中断允许寄存器 1(IE1)的位 7（对于串口 0 而言）和中断允许寄存器（IE2）的位 5（对于有第二个串口的芯片而言）

IE1寄存器	7	6	5	4	3	2	1	0
	UTXIE0	URXIE0						

IE2寄存器	7	6	5	4	3	2	1	0
			UTXIE1	URXIE1				

如果使用的芯片是 11，12 系列，由于仅具有一个串口，位于模块允许寄存器（IE2）的位 1。

IE2寄存器	7	6	5	4	3	2	1	0
							UTXIE0	URXIE0

2.2.2 UART 接收

2.2.2.1 接收允许

UART 接收也具有一个移位寄存器和接收数据缓冲器（UxRXBUF），同样理由不必理会移位寄存器是如何工作的。由于接收是出于被动方式下，因此，总是让接收器处于工作状态下，因此必须设置如下：

发送允许位（URXEx）被设置。

对于 MSP13，14，15，16 系列芯片而言，URXEx 位于模块允许寄存器 1（ME1）的位 6（对于串口 0 而言）和模块允许寄存器（ME2）的位 4（对于有第二个串口的芯片而言）

ME1寄存器	7	6	5	4	3	2	1	0
	UTXE0	URXE0						

ME2寄存器	7	6	5	4	3	2	1	0
			UTXE0	URXE0				

如果使用的芯片是 11，12 系列，由于仅具有一个串口，位于模块允许寄存器（ME2）的位 0。

ME2寄存器	7	6	5	4	3	2	1	0
							UTXE0	URXE0

设置 URXEx 为‘1’，则允许 UASRT 模块接收器工作，否则，清零为禁止工作。

2.2.2.2 接收完成标志或中断

由于查询方式可能造成数据丢失，因此，仅讨论中断方式。当移位寄存器接

收到一个完整的字符后， 无论是否中断允许， 硬件总是自动设置 URXIFGx 标志为 1 （低功耗唤醒是个例外），如果全局中断（GIE）串口中断允许（URXIEx）的话，则产生中断，当用户程序响应这个中断后，UxRXIFG 自动被硬件清零，或者，用户读取 UxRXBUF 的内容后，UxRXIFG 也会被自动清除。

应当注意的是，在查询方式下，UTXIFGx 不会自动清零，需要用户清除这个标志。URXIFGx 位于中断标志寄存器（IFGx）的位 7,分布如下：

对于 MSP13，14，15，16 系列芯片而言，URXIFGx 位于中断标志寄存器 1（IFG1）的位 7（对于串口 0 而言）和中断标志寄存器（IFG2）的位 5（对于有第二个串口的芯片而言）

IFG1寄存器	7	6	5	4	3	2	1	0
	UTXIFG0	**URXIFG0**						

IFG2寄存器	7	6	5	4	3	2	1	0
			UTXIFG1	**URXIFG1**				

如果使用的芯片是 11， 12 系列， 由于仅具有一个串口， 位于模块允许寄存器 （IFG2）的位 1。

IFG2寄存器	7	6	5	4	3	2	1	0
							UTXIFG0	**URXIFG0**

发送中断允许的控制位（URXIEx）处于也随芯片信号的不同而不同。分布如下：

对于 MSP13,14,15,16 系列芯片而言,URXIEx 位于中断允许寄存器 1(IE1)的位 7（对于串口 0 而言）和中断允许寄存器（IE2）的位 5（对于有第二个串口的芯片而言）

IE1寄存器	7	6	5	4	3	2	1	0
	UTXIE0	**URXIE0**						

IE2寄存器	7	6	5	4	3	2	1	0
			UTXIE1	**URXIE1**				

如果使用的芯片是 11，12 系列，由于仅具有一个串口，位于模块允许寄存器（IE2）的位 1。

IE2寄存器	7	6	5	4	3	2	1	0
							UTXIE0	**URXIE0**

2.2.3 低功耗模式下接收唤醒过程

在低功耗模式下，如果希望通过从串口接收来唤醒系统，那么，需要进行有关唤醒功能的预先设置，即设置 UxTCTL 的位 3（URXSE）。如下图所示，当然，实现这个功能有关的中断允许也必须设置好。

UxTCTL寄存器	7	6	5 4	3	2	1	0
	Unused	CKPL	SSELx	URXSE	TXWAKE	Unused	TXEPT

当芯片处于低功耗模式下，一个数据的启动位到达时，将会产生一个串口接收中断，此时，不会产生 URXIFGx 标志，因此，用户中断服务程序中，应当判断出这是低功耗唤醒中断，此时，用户需要做：

1.清除 URXSE，表示响应唤醒中断，清除内部有关标志（用户不必关心内部如何运作细节）。

2.设置 URXSE，表示为下次唤醒做准备。

3.退出低功耗状态。

当完成上述三个步骤后，通常接着又有一次接受中断到来，此时，会发现 URXIFGx 标志被设置，因此，按照正常操作读取数据。

二、通讯方式

1、异步模式（UART）的选择

MSP430F149 单片机支持两种不同的串行协议，异步通信（UART）协议和同步通信（SPI）协议。这两种协议的选择是通过控制寄存器中的 SYNC 位来决定的。

本设计中主要是利用 MSP430 的异步通信（UART）模式原理实现单片机与 PC 机之间的串口通信的。

MSP430 控制寄存器内的信息决定了 USART 的基本操作，选择异步模式(UART)需要通过设置 SYNC=0 来实现；本设计中对于其他控制位的设置还有：设置 CHAR=1，选择字符长度为 8 位；设置 SP=0，选择停止位 1 位；设置 PEV=0，选择奇校验；设置 MM=1，选择地址位多机模式协议。控制位的选择基本上决定了系统的通信方式和通信格式。

2、UART 模块的特点

由于 MSP430 单片机具有两个片内的 UART：串口 0 和串口 1，实现两个串口通信相当容易，只需要设置适当的寄存器就可以使串口工作起来，两个串口都采用中断方式，当接收有数据时，设置一个标志通知主程序有数据到来，当主程序有数局要发送时，设置一个中断标志进入中断发送数据。本设计选用串口 1 与上位机进行通信。

在异步模式下，接收部分自身实现帧的同步，通信双方只要使用相同的波特率即可。异步模式的帧格式有 1 位起始位、7 位或 8 位数据位，校验位，1 位地址位，1 或 2 位停止位构成。在异步模式下，MSP430 支持两种多机模式：线路空闲多机模式和地址位多机模式。线路空闲模式下，数据块被一段空闲的时间分割。在字符的第一个停止位之后收到 10 个以上的 1，表示检测到线路空闲；如果采用两个停止位，则第二个停止位被认为是空闲周期的第一个信号。在使用地址位多机模式时，字符包含一个附加的位作为地址标识，数据快的第一个字符带有一个置位的地址位，用以表明该字符是一个地址。由于已经设置了控制寄存器中的 MM=1，故在本设计中选择了地址位多机模式。

下面是 UART 通信的一些特点：

- 异步通讯模式，包括线路空闲/地址位通信协议。
- 有两个单独的移位寄存器，输入/输出移位寄存器。

- 传输 7 位或 8 位数据，可采用奇偶或无校验。
- 可编程实现波特率调整。
- 分别发，收单独中断。
- 有效地检测到起始位实现从低功耗唤醒。
- 状态标志检测错误或者地址位。

三、基本通信协议：

在 PC 机和多台单片机的通讯中，确定一个明确而合理的通讯协议是关键，包括对数据格式、通讯方式、传送速度、传送步骤、检纠错方式以及控制字符定义等问题做出统一规定。由于已经选择了 UART 的多机通信模式. 为了区别不同的分机，必须为每个分机分配一个唯一的地址，此地址唯一区别各单片机。数据格式采用数据包的形式，一次传输一组数据。数据包格式如表 1-2 所示：

表 1-2 数据包格式

起始标志位	分机地址	操作命令	数据长度	数据内容	和校验	结束标志

- 起始标志位:1 个字节
- 分机地址：1 个字节
- 命令/ 数据：1 个字节
- 数据长度：1 个字节
- 数据内容：n 个字节
- 和检验：2 个字节
- 结束标志位：1 个字节

数据格式中的地址位表示与 PC 机通讯的单片机地址。操作命令则表示此次通讯要完成的操作。在单片机发送上位机接收的时候，协议规定命令 FFH 为上报数据， 此时数据包中的数据长度、数据内容、和检验三个域便填充实际发送数据的个数、数据及和校验；命令 F0H – F3H 则表示单片机给 PC 机的反馈信息，此时数据包中的数据长度、数据内容和检验三个域为空，其中当命令为 F0H 表示接收成功，F1H 表示接收失败并要求重发，F2H 表示单片机有数据上报要求，F3H 表示单片机无数据上报要求。操作命令域在 PC 机发送单片机接收的时候也有相似的协议规定。

1 上位机和下位机实现的功能

上位机功能：

(1)向下位机进行呼叫，接着发送字符串，最后发送结束标志

(2)按照一定的时间间隔对串口进行读操作,如果有数据需要接收,则进行数据接收.

下位机功能：

(1)接收功能：识别上位机发送的地址，如果地址匹配则接收数据,直到结束标志到来,则停止接收。

(2)发送功能：下位机有数据需要发送时，首先向上位机发送本机地址，然后发送数据，最后发送结束标志。

2 下位机程序设计

2.1、下位机通讯协议

分布式控制系统中的下位机的每台单片机均有唯一的地址。通信开始时，先由 PC 机呼叫被叫单片机的地址，单片机在接收到 PC 机的呼叫后，首先判断是不是自己的地址，如果不是就不予理睬。如果是，则发送呼叫应答信号，并根据上位机的命令进行相应的接收或发送。

根据以上要求以及 UART 通信协议基本内容，可以将下位机通信协议设置如下表：

表 3-1 下位机通讯协议格式

地 址	数 据	结束

地址：取值 1—8，即上位机可以呼叫的 8 个下位机之一，发送时地址位有效。

数据：取值 9—127，为 ASCⅡ 可显示字符。

结束：0。

2.2、下位机程序流程图

下位机（单片机）程序设计包括初始化设计、串口中断服务设计和主处理程序设计。本例虽然有多个下位机，但他们除了本机地址的设置不同外，其他硬件电路都是相同的，所以各下位机的软件设计也是相同的。按照通信协议的要求可以设计出如图 3-1 的下位机程序流程。

例程一（详细解释）

```
//*****************************************************
//** 版权:     杭州利尔达科技有限公司
//** 文件名:  main_UART.C
//** 版本:
//** 工作环境: IAR EmbeddedWorkbench 4.11B
//** 作者:      LSD
//** 生成日期: 08.11.10
//** 功能:      UART 的的点对点通讯，串口调试助手上发数据给单片机
//**             单片机把该数据又重新发回给口调试助手
//*****************************************************
#include <msp430x26x.h>

#define  TXD0   BIT4
#define  RXD0   BIT5

void main(void)
{

  WDTCTL   =  WDTPW + WDTHOLD;        //关看门狗
  BCSCTL1 = CALBC1_1MHZ;
  DCOCTL = CALDCO_1MHZ;
```

```
    P4DIR  |= BIT1;
    P4OUT  &= ~BIT1;                //UART_V 打开（注意点：此句为打开通讯模块的电源）


    UCA0CTL1 =  UCSWRST;            //置位 UCSWRST 位，使 UART 模块的寄存器处于初始状态
  //#define UCSWRST                 (0x01)     /* USCI Software Reset */
```

UCSWRST	Bit 0	Software reset enable
	0	Disabled. USCI reset released for operation.
	1	Enabled. USCI logic held in reset state.

```
    UCA0CTL1 |= UCSSEL1;           //UCLK=SMCLK=1MHz;
```

UCSSELx	Bits 7-6	USCI clock source select. These bits select the BRCLK source clock.
	00	UCLK
	01	ACLK
	10	SMCLK
	11	SMCLK

```
  //#define UCSSEL1                 (0x80)     /* USCI 0 Clock Source Select 1 */
  // 与 1000 0000 相或，选择 SMCLK
    UCA0BR0  = 104;
    UCA0BR1  = 0;
```

UxBR0 波特率选择寄存器 0

7	6	5	4	3	2	1	0
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

UxBR1 波特率选择寄存器 1

7	6	5	4	3	2	1	0
2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8

```
    UCA0MCTL = UCBRS0;              //设置波特率 9600
  #define UCBRS0    (0x02)  /* USCI Second Stage Modulation Select 0 */
```

7	6	5	4	3	2	1	0
UCBRFx				UCBRSx			UCOS16
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

UCBRSx	Bits 3-1	Second modulation stage select. These bits determine the modulation pattern for BITCLK. Table 15-2 shows the modulation pattern.

```
  //*MSP430 系列，usart 模块的波特率值设定是通过以下三个参数决定的：UxBR0,UxBR1,UxMCTL
```

波特率＝BRCLK/N

BRCLK:时钟源;
N:波特率产生的分频因子。N=UxBR1+UxBR0+UxMCTL,其中 UxBR1+UxBR0 为整数部分，UxMCTL 为设定小数部分
举个实例：波特率＝2400，时钟源＝32.768kHz
N=32768/2400＝13.65
很明显：UxBR1+UxBR0＝13，即 UxBR1＝0，UxBR0＝13，主要是小数部分对于初学者不是很好理解;
首先把小数部分 0.65×8，即 5.2，取整后为 5。这个 5 的意思就是在 UxMCTL 中的 8 位里要有 5 个 1，剩下来的就是怎么分配这 5 个 1 的位置了。注意一点就是这 5 个 1 要相对分散点。
在这个设定中，UxMCTL 取 0x6B 即：01101011，也可以是其它值。　/

```
  P3DIR |=TXD0;
  P3SEL |=TXD0+RXD0;          //发送和接收引脚为第 2 功能
  UCA0CTL1 &= ~UCSWRST;//关闭复位，开始操作
  IE2 |= UCA0RXIE;            //接收中断使能
#define UCA0RXIE              (0x01)
```

7	6	5	4	3	2	1	0
						UCA0TXIE	UCA0RXIE
						rw-0	rw-0

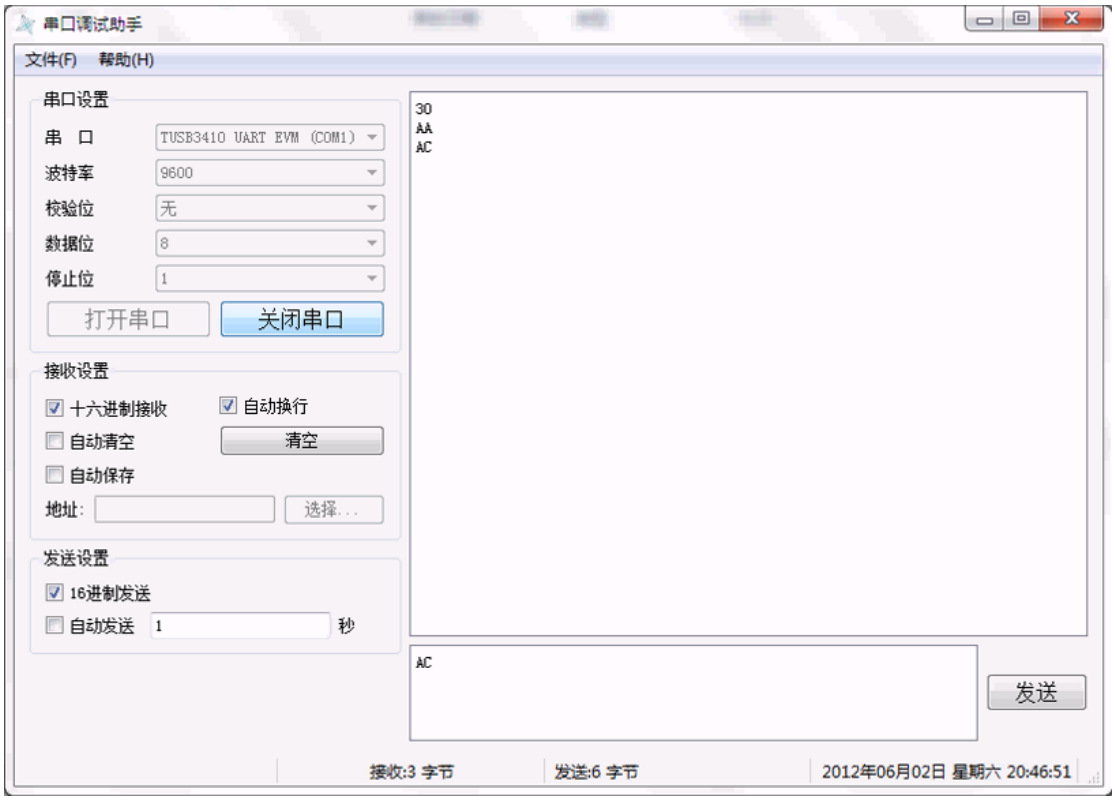
	Bits 7-2	These bits may be used by other modules (see the device-specific data sheet).
UCA0TXIE	Bit 1	USCI_A0 transmit interrupt enable
		0 Interrupt disabled
		1 Interrupt enabled
UCA0RXIE	Bit 0	USCI_A0 receive interrupt enable
		0 Interrupt disabled
		1 Interrupt enabled

```
  _EINT();
  while(1)
  {
    LPM0;                     //进入低功耗
  }

}

//*************************************************************
//**函数(模块)名称:USCI0RX_ISR
//**功能: UART 接收中断内把接收缓存内的数据赋值给发送缓存
//**输入参数: 无
//**输出参数: 无
//**函数返回值说明:无
//**使用的资源：无
//**其它说明: 无
//*************************************************************

#pragma vector = USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void)
```

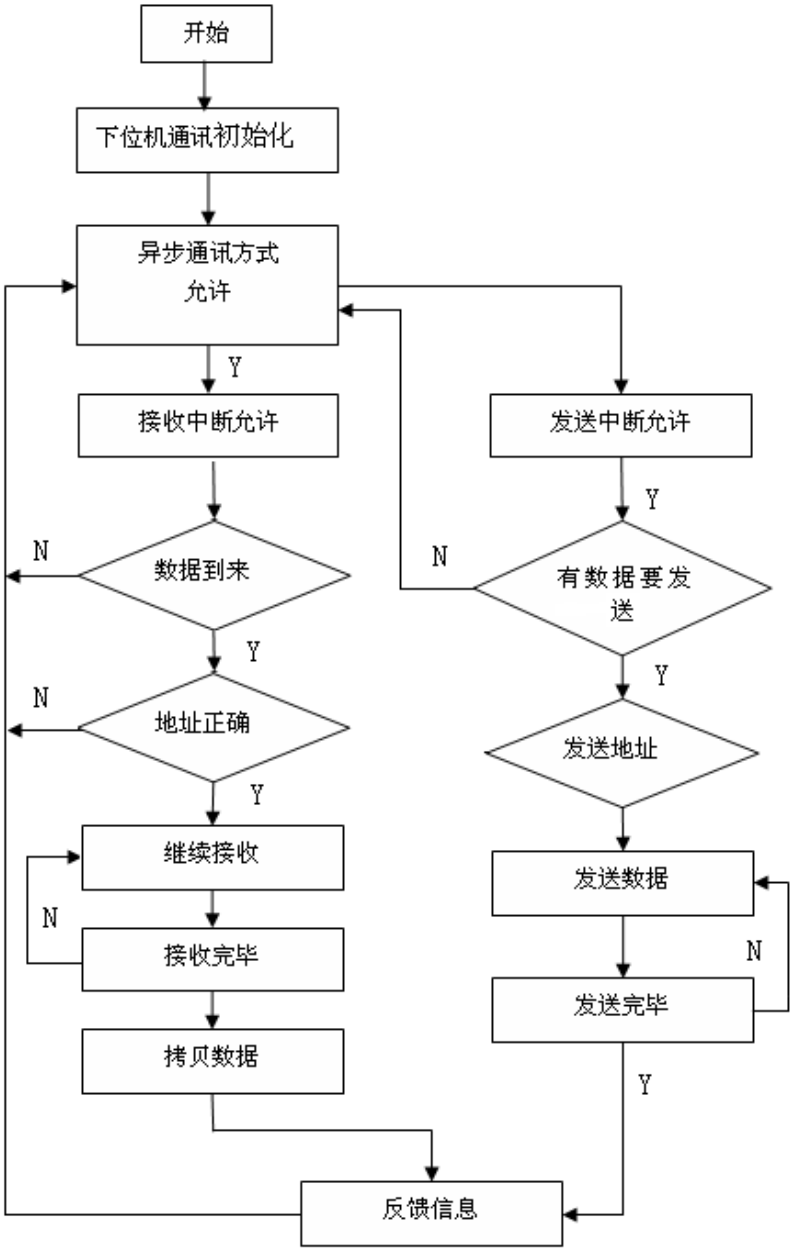
```
{
  while (!(IFG2&UCA0TXIFG));
  UCA0TXBUF=UCA0RXBUF;      //把接收到的数据返回给串口调试助手
}
```

UCRXBUFx	Bits 7-0	The receive-data buffer is user accessible and contains the last received character from the receive shift register. Reading UCAxRXBUF resets the receive-error bits, the UCADDR or UCIDLE bit, and UCAxRXIFG. In 7-bit data mode, UCAxRXBUF is LSB justified and the MSB is always reset.
UCTXBUFx	Bits 7-0	The transmit data buffer is user accessible and holds the data waiting to be moved into the transmit shift register and transmitted on UCAxTXD. Writing to the transmit data buffer clears UCAxTXIFG. The MSB of UCAxTXBUF is not used for 7-bit data and is reset.

电脑上运行串口调试助手，正确设置后向对应串口发送数据，可以看到MSP430 向电脑发送的相同的数据。



例程二



2.3、初始化设置

初始化设置包括时钟初始化，端口初始化和串口初始化。

1、时钟初始化程序及说明：

```
#include <MSP430X14X.h>
#include "UART.h"

void Init_CLK(void)
{
    unsigned int i;
    BCSCTL1 = 0X00;          //将寄存器的内容清零
                               //XT2 震荡器开启
                               //LFTX1 工作在低频模式
```

```
                                        //ACLK 的分频因子为 1
    do
    {
  IFG1&=~OFIFG;                              // 清除 OSCFault 标志
  for(i=0x20;i>0;i--);
    }
    while ((IFG1 & OFIFG) == OFIFG);         // 如果 OSCFault =1
      BCSCTL2=0x00;            //将寄存器的内容清零
    BCSCTL2 += SELM1;         //MCLK 的时钟源为 TX2CLK，分频因子为 1
    BCSCTL2 += SELS;      //SMCLK 的时钟源为 TX2CLK，分频因子为 1

}
```

由上面的程序可以知道，只要设置 BCSCTL1 和 BCSCTL2 寄存器的相应位就可以获得需要的 MCLK、SMCLK、和 ACLK 的时钟信号。

2、端口初始化

```
   void Init_Port(void)
{
    //将所有的管脚在初始化的时候设置为输入方式
    P3DIR = 0;
    //将所有的管脚设置为一般 I/O 口
    P3SEL = 0;
    return;
}
```

上面的初始化程序将 P3 口初始化为一般的 I/O 接口。

3、串口初始化

由于串口 1（UART1）的管脚号为 P3.6 和 P3.7，端口初始化只是将端口设置为一般的 I/O 接口。而本设计里需要将 P3.6 和 P3.7 作为 UART 的输出和输入管脚，所以这里需要对其另外初始化。

UART1 的初始化程序代码为:、

```
void Init_UART1(void)
{
     U1CTL = 0X00;  //将寄存器的内容清零
    UCTL1&=~SWRST; //SWRDT 复位，UART 允许
    UCTL1=CHAR+MM; //8 位数据位，1 位停止位，地址位模式
    URCTL1 |=URXWIE; //只有地址字符使 URXIFG 置位
    UBR0_1 = 0X03;
    UBR1_1 = 0X00;
    UMCTL_1 = 0X4A; //使用 32KHz 晶振时，波特率为 9600bps
    U1TCTL=0x10; //选定 ACLK（32KHz 晶振）为时钟源
    ME2 |= UTXE1 + URXE1; //使能 UART1 的 TXD 和 RXD
    IE2 |= URXIE1;         //使能 UART1 的 RX 中断
    IE2 |= UTXIE1;        //使能 UART1 的 TX 中断
```

```
    P3SEL |= BIT6;    //设置 P3.6 为 UART1 的 TXD
    P3SEL |= BIT7;      //设置 P3.7 为 UART1 的 RXD
    P3DIR |= BIT6;       //P3.6 为输出管脚
  return;
}
```

根据硬件设置的要求，上面的程序设置了串口 1 的参数，比如 8 位数据位，1 位停止位，地址位多机模式波特率发生器选择 ACLK，波特率为 9600 波特/秒等，将 P3.6 和 P3.7 设置为串口 1 的 I/O 管脚。

2.4、串口中断服务程序

串口接收和发送都采用中断方式，设计单片机通信程序时，必须充分发挥单片机的效率，由于单片机多应用于实时性较强的控制场合，因此， 应将及时响应和控制对象的动作放在优先考虑的位置，以尽量减少通信等辅助性操作所占用的 CPU 时间[11]。基于上述考虑，在设计单片机通信程序时，将中断程序分为接收中断服务程序和发送中断服务程序 2 部分。下面为串口通信程序流程图：

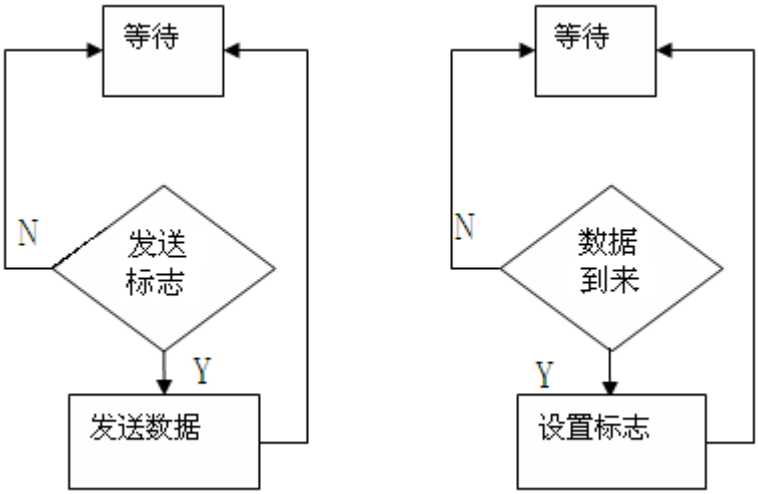


图 3-2 串口通信模块程序流程图

1、接收中断服务程序

当有数据收到时，设置一个标志通知主程序有数据到来，当地址位验证无误后，则开始接收数据。下面为接收中断处理程序代码：

```
interrupt [UART1RX_VECTOR] void UART1_RX_ISR(void)
{
if(URCTL1&URXWIE)     //接收为地址方式时等待正确的地址出现
{
if(URBUF_1==ADDRESS) //地址正确，改变接收为数据方式，准备接收
   {
      URCTL1&=~URXWIE;
      nRX1_Len_temp = 0;
    }
  }
else                                    //接收为数据时
```

```
{
UART1_RX_BUF[nRX1_Len_temp] = RXBUF1;//数据存入 UART1_RX_BUF 里
    nRX1_Len_temp += 1;
    if(UART1_RX_BUF[nRX1_Len_temp ++]== 0) //是否到达停止位
{
      nRX1_Len = nRX1_Len_temp;
      nRev_UART1 = 1;                    //设置标志来通知主程序
    nRX1_Len_temp = 0;
    URCTL1 |=URXWIE;  //数据收完改变为地址方式

     }
 }
```

对于接收中断，程序处于等待状态，当外面有数据到来时则触发接收，进入接收中断服务程序，当地址验证正确开始后面的数据，中断从“RxBUF1”寄存器里读取数据，将读到的数据放到“UART1_RX_BUF[ ]”全局缓冲区里,在接收数据之后设置一个标志“nRev_UART1”来通知主程序，完成后等待下一中断的到来。接收中断服务程序包含了对地址位是否匹配的验证。

2、发送中断服务程序

当主程序有数据要发送时，设置一个中断标志进入中断并发送数据。下面为程序代码：

```
interrupt [UART1TX_VECTOR] void UART1_TX_ISR(void)
{
    if(nTX1_Len != 0)
    {
      nTX1_Flag = 0;           // 表示缓冲区里的数据没有发送完
      TXBUF1==0x01;                        //先写入地址字符
      TXBUF1 = UART1_TX_BUF[nSend_TX1];  //开始传输数据
      nSend_TX1 += 1;
        if(nSend_TX1 >= nTX1_Len)     //数据是否发送完
         {
          nSend_TX1 = 0;
          nTX1_Len = 0;
          nTX1_Flag = 1;       //缓冲区里没有数据要发送了
      }
    }
   }
```

对于发送中断，程序一般处于禁止等待状态。只有当单片机的发送缓冲区历由数据需要发送，并将发送中断置为允许方式后，发送中断才开始工作。发送时从缓冲区里发送数据，遵守通讯协议：首先发送地址位，然后发送需要传输的数据，最后发送校验以及结束标志。在发送中断服务程序里从“UART1_TX_BUF[ ]”全局缓冲区里取出数据给“TXBUF1”寄存器进行发送，发送完后发送中断服务程序等待下一中断的到来。

以上两程序可以看出采用中断有很好的结构，只要在中断服务程序里理接收

和发送数据，然后与主程序进行数据交换，易实现多任务操作，很好利用单片机资源。

2.5、主处理程序

主处理程序包含初始化、设置串口工作方式、对接收到的数据进行处理以及封装需要发送的数据。下面是它的程序代码。

```
#include <msp430x14x.h>
#include "sp3220.h"
//定义串口操作变量
char nRev_UART1;     // 串口 1 的接收标志
char UART1_TX_BUF[60];  // 串口 1 的发送缓冲区

char UART1_RX_BUF[60];      // 串口 1 的接收缓冲区
int nTX1_Len;
char nRX1_Len;
char nRX1_Len_temp;
char nTX1_Flag;
int nSend_TX1;
void main(void)
{
   int nRes_UART1;
    int nRes = 0;
    char UART1_RX_Temp[60];
    int i;
    int n;
      WDTCTL = WDTPW + WDTHOLD;    // 关闭看门狗
       _DINT();                    // 关闭中断
       Init_CLK();               // 初始化时钟
       Init_Port();               // 初始化端口
       Init_UART1();             // 初始化串口 1
      _EINT();                  // 打开中断
      for(;;)                    // 进入处理循环
       {
          if(nRev_UART1 == 1)      //如果有接收中断
       {
          nRev_UART1 = 0;
           for(i = 0;i < nRX1_Len;i++)
             UART1_RX_Temp[i] = UART1_RX_BUF[i]; // 将接收到的数据拷贝到临时缓冲区。
            nRes = ProcessCMD(UART1_RX_Temp,nRX1_Len);
          switch(nRes)
          {
           case 1:
             UART1_TX_BUF[0] = 'O';
             UART1_TX_BUF[1] = 'K';
```

```
                UART1_TX_BUF[2] = 13;
                nTX1_Len = 3;
                // 设置中断标志，进入发送中断程序
                IFG2 |= UTXIFG1;
                 nRX1_Len = 0;
                break;
            case 2:
                for(n = 0;n < nRX1_Len;n++)
                   UART1_TX_BUF[n] = UART1_RX_Temp[n];
                UART1_TX_BUF[nRX1_Len] = 'O';
                UART1_TX_BUF[nRX1_Len+1] = 'K';
                UART1_TX_BUF[nRX1_Len+2] = 13;
                nTX1_Len = nRX1_Len + 3;
                // 设置中断标志，进入发送中断程序
                IFG2 |= UTXIFG1;
                 nRX1_Len = 0;
                break;
            case -1:
                UART1_TX_BUF[0] = 'E';
                UART1_TX_BUF[1] = 'R';
                UART1_TX_BUF[2] = 'R';
                UART1_TX_BUF[3] = 'O';
                UART1_TX_BUF[4] = 'R';
                UART1_TX_BUF[5] = 13;
                nTX1_Len = 6;
                // 设置中断标志，进入发送中断程序
                IFG2 |= UTXIFG1;
                 nRX1_Len = 0;
                break;
            }
        }
    }
}
```

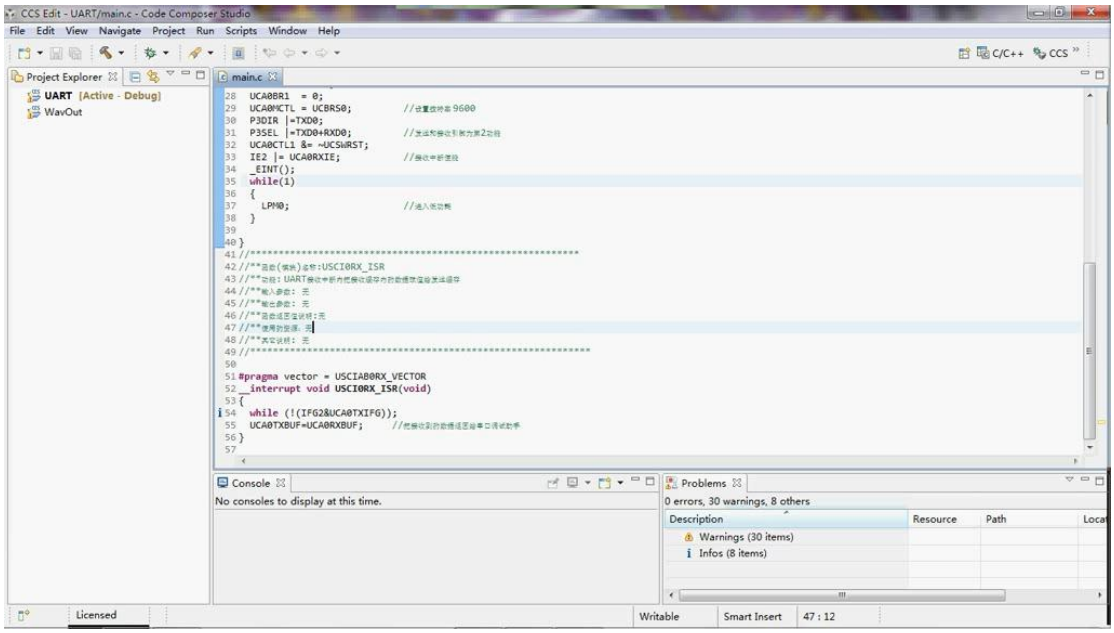
在上面的程序中主要根据“ProcessCMD(UART1_RX_Temp,nRX1_Len)”对得到的结果进行处理，向 PC 发送响应数据，如果接收到的数据有错误，则发送“ERROR”。数据封装完成后,设置“nTx1_Len”的长度,并通过“IFG2 |=UTXIFG1;”发送中断，从而使中断处理程序进行数据发送.上面程序中“ProcessCMD(UART1_RX_Temp,nRX1_Len)”主要处理接收到的数据并返回相应的代码以使主程序处理。该函数的具体代码为：

```
int ProcessCMD(char pBuf[],int nLen)
{
    int nTemp = -1;
    int i;
       if(nLen <= 2) return -1;
```

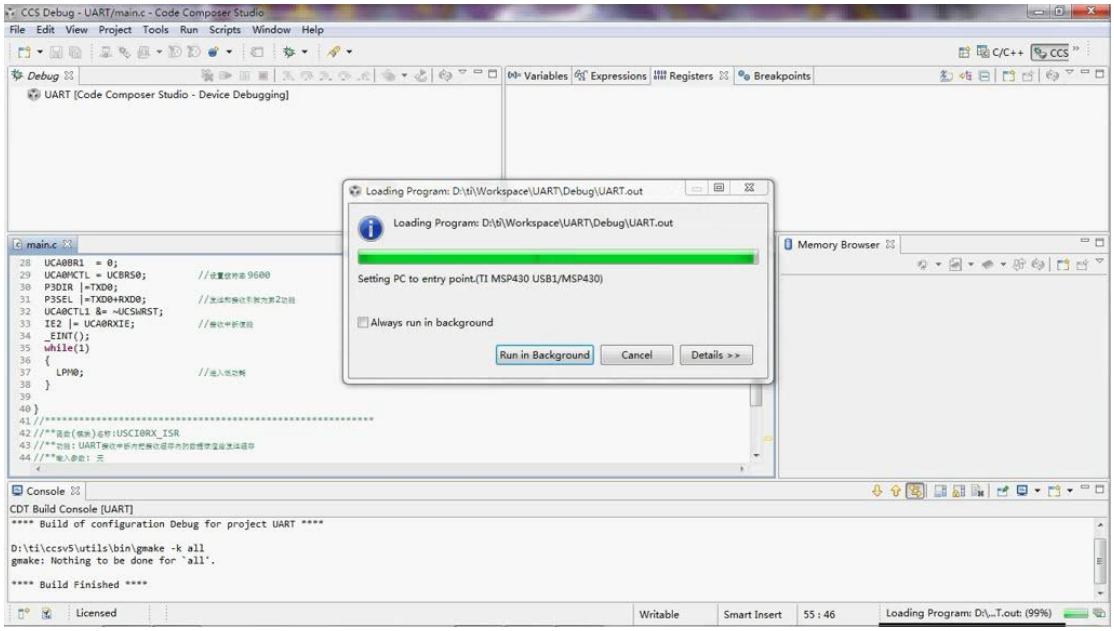
```
    if (nLen == 5)
{
    if((pBuf[0] == 'A') && (pBuf[1] == 'T')
        && (pBuf[2] == 'E') && (pBuf[3] == '0'))
        nTemp = 1;
    if((pBuf[0] == 'A') && (pBuf[1] == 'T')
        && (pBuf[2] == 'E') && (pBuf[3] == '1'))
        nTemp = 2;
}
    return nTemp;
}
```

四、操作范例

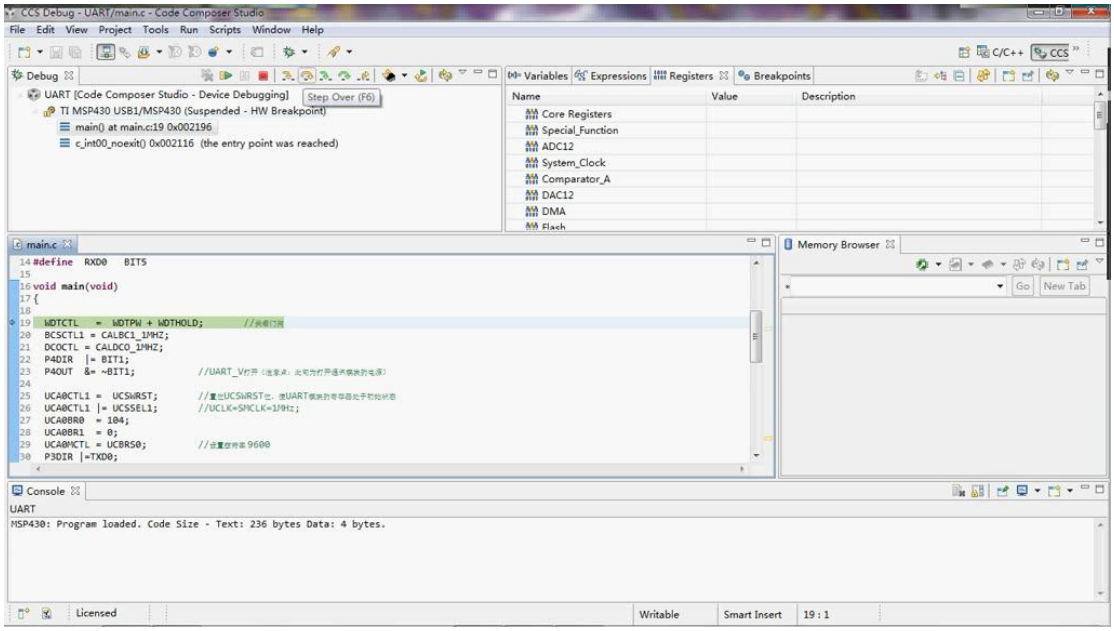
1. 打开 CCS，载入项目。



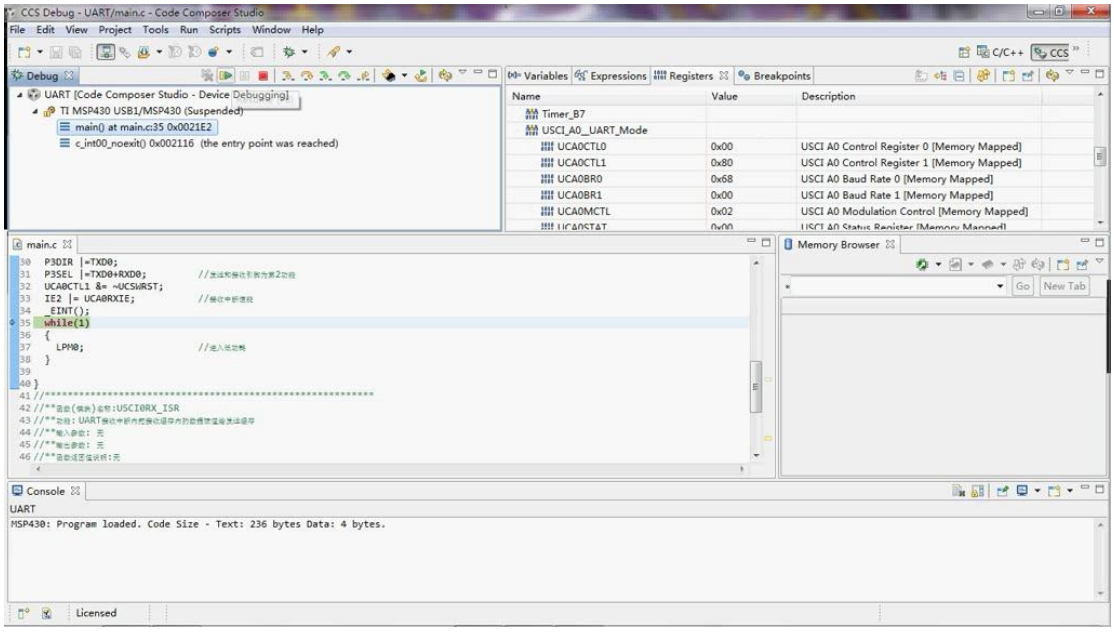
2. 单击绿色甲壳虫（Debug）图标，将程序烧入 MSP430。



3. 进入 Debug 之后，可通过界面上的按键选择单步调试或直接执行至断点。



4. 暂停 Debug 之后，可从右上角的寄存器（Registers）窗口查看对应寄存器的当前情况。



5.将开发板上对应的端口与电脑连接之后，可以使用串口调试工具进行测试。