实现带有 TimerA 的 UART 功能

作者：lyzhangxiang

学习单位：安徽师范大学

完成时间：2010-11-13

参考文献：TI文档 Application Report SLAA078A

一、摘要

本应用报告介绍了如何使用 Timer_A 实现 UART 功能。该包括例子是专门为 MSP430x11x 家庭，但他们能适应任何 MSP430 家族成员纳入 Timer_A。使用硬件 UART 的功能在 Timer_A 功能和软件。执行是半双工的，事件驱动，它支持的 8N1 波特率协议从 1200 到 115200 或者更快。

二、简介

异步串行通信，可以添加到 MSP430x11x 应用程序综合 Timer_A 模块硬件的功能。这份报告提供了一些的 UART 函数实例演示了一对一 MSP430F1121 闪存 RS232 接口单片机与 PC 机串行端口。一个描述如何使用 Timer_A1 硬件提供自动启动位检测，波特率生成和数据位锁存是详细。Timer_A 硬件特性的软件，大大降低 CPU 开销通常与微控制器软件 UART 实现。硬件的功能也让 Timer_A 的 UART 操作作为背景的作用，同时与其他实时系统的任务。

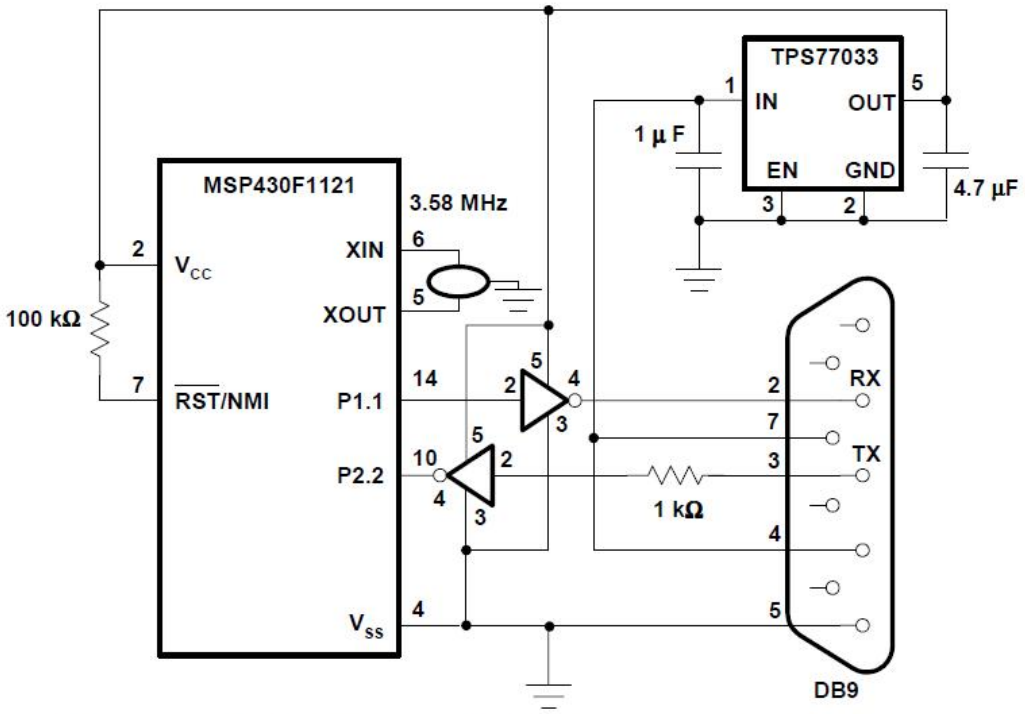

Figure 1. MSP430x11x(1) UART Demonstration Circuit

MSP430x11x Timer_A 的 UART 描述:串行通信的 MSP430F1121 与另一个系统，在这报告电脑使用 RS232 接口。特点是两个系统之间交换通过三线：接收，传输和共同点。协议使用的字符是 8N1：8 个数据位，无奇偶校验，一个停止位。用户可以修改 UART 的功能支持其他协议和波特率，奇偶校验和包括一个 9 个位寻址。 UART 的功能描述使用捕获比较寄存器 0（CCR0），Timer_A 的三个可用寄存器之一。 CCR0 用于起始位检测，波特率生成和数据位锁存。另外两个捕获比较寄存器可用于其他事。该 CCR0 选择是任意的。任何或所有 CCRx 寄存器可用于 UART 的功能。端口引脚 P1.1 和 P2.2 是外设选项选择的相关于 Timer_A CCR0。P1.1 用于传输，P2.2 用于接收。外设选项选定为使用外围设备选项选择引脚寄存器，P1SEL 和 P2SEL。由于 P1.1 置为输出时，该引脚必须配置为使用输出端口方向寄存器 1（P1DIR）。P2.2 作为运行需要输入。这是一个 MSP430 的端口引脚的默认。 Timer_A 配置为运行在连续模式，允许定时器资源用于同时与 UART 等功能可用。中央处理器寄存器 R4 是为 RXTXData -缓冲区使用的 UART 数据或出位的变化。 R5 的 CPU 寄存器用于 BitCnt，有点跟踪登记。 R4 和 R5 的选择是任意的。任何 CPU 寄存器或 RAM 字节可以用于这些功能。

在接收模式下，捕获比较控制寄存器 0（CCTL0）的初始配置，使得在下降沿接收引脚 P2.2 CCR0 捕获。由于接收线路空闲高，一降边指示起始位开始。当 UART 功能已准备好接收数据，没有开销放在 CPU 的功能，即使是准备接收任何一个字符时间。 CPU 资源执行后，起始位下降沿上 P2.2 发生。下降对 P2.2 边沿捕获了自由运行 Timer_A 计数器寄存器（焦油）的电流值，CCR0 与任何其他运行时的活动无关。捕获是通过 Timer_A 硬件，而不是由软件。同时发出一个中断给 CPU。中断的延迟并不大关注确切时间的下降沿触发中断正在 CCR0 存储，独立其他活动。启动后位的边缘检测，软件重新配置，使 CCR0 的 CCTL0 在第一种模式的比较发生在第一个数据位的中间。一个 1.5 位偏移量添加到 CCR0，定位比较下的第一个数据位的中间。接收到的数据是同步的捕获比较输入（SCCI）Timer_A 锁存硬件。SCCI 是可读的 CCTL0 闩锁。在 UART 的功能，SCCI 捕获的逻辑电平同步与 CCR0 输入 P2.2 进行比较。UART 的功能是从 SCCI 中接收锁存的数据。软件不直接测试 P2.2。

```
RX_Bit      bit.w   #SCCI,&CCTL0            ; Get bit waiting in SCCI
            rrc.b   RXTXData                ; Store received bit
```

第一个数据位后，1 位长的偏移量添加到 CCR0 定位在未来捕捉中间的每一位。八位连续的数据被锁存，并收到来自 SCCI 软件到 RXTXData 位的数据。发射模式任务是简单，因为 MSP430 在决定时，传输数据，没有启动位边缘检测的必要性。在缓冲区中存储的数据 RXTXData 传输引脚 P1.1 使用 Timer_A CCR0 输出至硬件。

CCR0 是预先在比较模式下位传输使用输出模式控制位在 CCTL0。模式控制位被预先配置为复位模式（逻辑 0），或设置模式（逻辑 1）发生在未来比较 CCR0。当比较时，在一开始位传输的数据，自动输出 CCR0 硬件数据配置 P1.1 置位并发出一个中断。随着 CCR0 硬件自动输出数据上 P1.1 置位，软件中断延迟和位定时的关注减少。这不是必要的软件准备 CCR0 输出锁存器在一个确切的时间内。 UART 的功能软件不直接对 P1.1 置输出数据，而是预装在 CCR0 输出数据位输出至锁存器的硬件。

```
TX_Bit      rra.b   RXTXData            ; LSB is shifted to carry
            jc      TX_Mark             ; Jump if bit = 1
TX_Space    bis.w   #OUTMOD2,&CCTL0     ; TX Space
            reti                        ;
TX_Comp     bic.w   #CCIE,&CCTL0        ; All Bits RX, disable interrupt
TX_Mark     bic.w   #OUTMOD2,&CCTL0     ; TX Mark
            reti                        ;
```

每个位输出之前，软件循环查询 RXTXData 到进位。适当的跳转提出和 CCTL0 输出模式准备和 1 位长的偏移量添加到 CCR0。

三、波特率计算

Timer_A CCR0 用于波特率产生。根据所需波特率，间隔 Bitime 的计算方法。 Bitime 是 Timer_A 位和 PC 之间的计数长度时间间隔 Timer_A 闩锁在接收和发送出的数据位。 Bitime 是为 Timer_A 计算时钟的波特率分源。

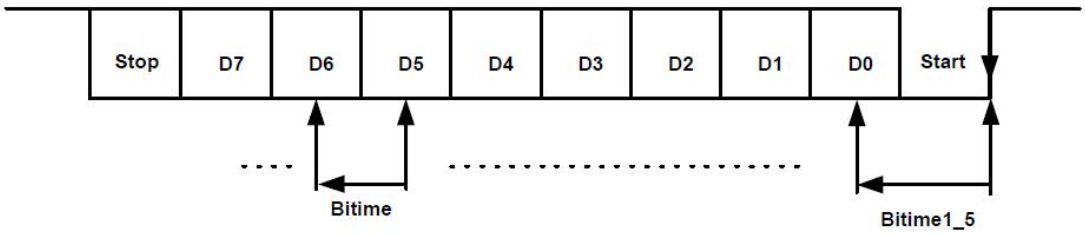

Figure 2. UART8N1 Character

Timer_A 有几个可用的时钟源和除法（参见器件表具体数据）。可用于 MSP430x11x Timer_A 模块的时钟源包括辅助时钟（ACLK），子系统时钟（SMCLK）和两个外部时钟。

例如：考虑与 Timer_A 作为辅助时钟源选择 9600 波特率 ACLK，它被配置为作为一个 3.579545 MHz 的的 XTAL 相同。 Bittime = 3 5799600/545= 372.9一373 实际波特率= 3 579 三百七十三分之五百四十五= 9597 由于只有一个 Bitime 整数值可以被添加到 CCR0，值 373 使用。假设 9600 波特和一个 3.579545 MHz 的时钟源，四舍五入到最接近的整数 Bitime 错误小于 0.03％每位。

四、软件开销

固件工程师可能会关切的是使用中 UART 的硬件/软件系统开销，而不是一个专用的串行端口。UART 的功能在本报告所述用途 timer_A 硬件的功能，最大限度地减少 CPU 的负担。Timer_A 硬件记录起始位优势和锁存和输出自动位的数据。该软件需要有一个 26 位的每接收或发送的最大周期包括中断服务程序。开销是 CPU 的时钟（MCLK）和波特率功能。使用 3.58 - MHz 的 9600 波特率例如上面的 MCLK，开销计算公式为：
软件开销=（26 个 CPU 周期）×（9600）/ 3 579 545 = 6.9％

五、演示电路

图 1 的示例电路与调控，从直接供电由一个 PC 串口 3.3 - V 的 TPS76033 低压降稳压器。出于演示的目的，串口接口是通过使用两个 TI SN74AHC1G04 逆变器。如果一个完全兼容的 RS232 接口是必需的，如 TI 的低功耗 3 - V 的 MAX2331 集成电路都可以使用。复位拉高和 3.58 MHz 的陶瓷谐振器用于产生时钟使用。

六、示例代码

所包含的例子 11x1_uart1.s43 使用图 1 中的电路，并提供了基本的回声功能。接收到一个字符从 PC 和回显。在初始化过程中，端口引脚配置和所有时钟都同步到 LFXT1 晶体振荡器。 LFXT1 是配置为在高速运行（高频）模式。如果在这个例子中，MCLK 的源是一个外部高频晶体，晶体必须是稳定或振荡器故障安全模式会自动使用 FOR MCLK 的 DCOCLK。该 OSCFAULT 可以查询，以确保稳定的晶体前选择此 MCLK 的时钟源。

```
SetupBC     bis.b   #XTS,&BCSCTL1           ; ACLK = LFXT1 HF XTAL
SetupOsc    bic.b   #OFIFG,&IFG1            ; Clear OSC fault flag
            mov.b   #0FFh,R15
SetupOsc1   dec.b   R15                     ; Ddelay to ensure startup
            jnz     SetupOsc1
            bit.b   #OFIFG,&IFG1            ; OSC fault flag set?

           jnz     SetupOsc                ;
           bis.b   #SELM1+SELM0,&BCSCTL2   ; (CPU) MCLK = LFXT1
```

振荡器故障安全模式中描述的 MSP430x1xx 系列用户指南（SLAU049）。该主循环调用子程序 RX_Ready UART 接收准备，然后在低功耗模式 0 等待（LPM0）与 CPU 关闭。只有 Tmer_A1 和 ACLK 活跃。即使在 CPU 中关闭主循环中，UART 接收中断操作功能在后台驱动。后 UART 功能接收到 RXTXData，UART 的中断处理程序的完整的字符返回到主循环活动的 CPU。发送子程序被称为未来，又呼应在 RXTXData 回接收到的字符。循环重复，另一个字符等待收到。该示例使用的编码速度优化技术研究。内 CCR0_ISR，BitCnt 用于自动递增的间接寻址，直接程序流的确切节 ISR 的规定办理接收或发送位。软件是不需要投票登记标志或递减，以确定采取何种行动。自动递增寻址使用一个查找表，直接的方案立即流。

```
add.w   #Bitime,&CCR0       ; Bitime till next bit
br      @BitCnt+            ; Branch To Routine
```

该自动递增寻址的优点是速度和 CPU 的周期计算效率在需要处理为查找表所需的费用在 ISR 码字。

(1)例如 11x1_uart2.s43 -使用 DOC 产生波特率：

这个例子 11x1_uart2.s43 演示如何实现高波特率的 UART MSP430 的功能甚至从超低功耗模式，只使用一个 32 768 的晶体振荡器。 MSP430 的独立片上高速数控振荡器（DCO）可用

于波特率产生。DOC 是一个 100 - kHz 至 5 – MHz 甚至更快，数字可调 RC 型振荡器启动，是在不到 6μs 稳定。随着快速 DOC 启动，高波特率是可能的，甚至从超低功耗模式，要求 DOC 关闭—— 当 DOC 不到 6us 时一个起始位的下降沿被用作中断。
例如 11x1_uart2.s43 提供了一个 9600 波特率波特率 UART 解决方案，使用 DOC 产生波特率。32 768 用于手表晶振的 XTAL，取代了 3.58 MHz 的谐振器图 1。DOC 设置为 1 228 800 赫兹，用来驱动子系统时钟（SMCLK）它被配置作为 Timer_A 来源。DOC 被调整到一个高的速度值使用软件锁频环（FLL）。该软件 FLL 的整合速度 SMCLK（DOC 时钟）以上的速度较慢 ACLK（32 768 Hz 的晶体衍生物）时期。该 DCO 时钟调整，直到达到目标频率（1 228 800 赫兹）。该 DCO 时钟在设定的频率仍然是稳定的，只要 VCC 和温度的稳定。该软件 FLL 可以随时被调用的 DOC 重新调整。看到目前的数据表，MSP430x11x1 混合信号微控制器。在这个例子中，主循环中等待 LPM3，通常小于 2 微安与 UART 的功能充分准备接收。主循环回声操作与前面的例子一样。这个例子使用 BitCnt 登记只是为了跟踪字符接收进程，而不是使用自动递增，同前面的例子。

(2)例如 11x1_uart3.s43 采用 32 kHz 晶振产生波特率：
在某些情况下，32 768 Hz 的手表晶振用作 Timer_A 直接时钟源。UART 功能仍然是可能的，但在传输速率较慢，以防止位定时误差。范例 11x1_uart3.s43 提供了一个 2400 波特率 UART 的解决方案，假设一个 32 768 Hz 的手表晶体是用于的 XTAL（取代图 1 中的 3.58 - MHz 的谐振器）。没有软件 FLL 的是需要设置 DOC 的，因为晶振生成 UART 的位时间。当激活模式下，CPU 配置为在执行 DCO 时钟速度越快，CPU 的代码不在 32 768 赫兹执行代码。此外，在这个例子中，内存的位置用于 RXTXData 和 BitCnt，相对于 CPU 寄存器中使用的其他例子。