第三章　时钟系统（CS）

3.1　本章引言

时钟系统（Clock System）模块支持低成本和低功耗。通过使用 4 个内部时钟信号，用户可以在低功耗和性能之间做到最好的平衡。

时钟模块可以配置成无需任何外部组件，使用一个外部电阻器或完全使用 DCO 旁路模式。

时钟模块有四个系统时钟信号可以使用：

- ACLK：辅助时钟。当运行在 DCO 时，ACLK 是固定在 32kHz。如果设备是设置在 DCO 旁路模式，ACLK 运行在旁路时钟频率的 1/512。
- MCLK：主时钟。MCLK 可以被 1,2,4,8 或 16 分频。MCLK 通常被 CPU 和系统使用。
- SMCLK：子系统主时钟。SMCLK 可以被 1,2,4,8 或 16 分频。SMCLK 可以被各个外围模块通过软件选择使用。
- SD24CLK：SD24 时钟提供一个 1.024MHz 固定频率的时钟给 Sigma-Delta ADC(SD24)。该时钟只为 SD24 的请求所使用。如果 SD24 功能必须在 DCO 旁路模式下工作，那么外部时钟频率必须是 16.384Mhz。

这个驱动程序包含在 cs.c 文件里，cs.h 头文件包含该应用程序使用的 API 定义。

3.2　函数总览

宏

#define CS_DCO_FREQ　16384000

1	**void GS_setupDCO (uint8_t mode)**
	使用选中的模式配置 DCO
2	**void CS_initClockSignal (uint8_t clockSource, uint8_t clockSourceDivider)**
	使用分频器初始化时钟信号
3	**uint32_t CS_getACLK (void)**
	获取当前 ACLK 的频率（单位 Hz）
4	**uint32_t CS_getSMCLK (void)**
	获取当前 SMCLK 的频率（单位 Hz）
5	**uint32_t CS_getMCLK (void)**
	获取当前 MCLK 的频率（单位 Hz）
6	**uint8_t CS_getFaultFlagStatus (uint8_t mask)**
	获取 DCO 故障（或错误）标志状态

CS API 函数分为三组：配置时钟模块的、侦测时钟速度的和 CS 错误标记处理的。

CS 一般配置和初始化的函数有:

- **CS_setupDCO()**
- **CS_initClockSignal()**

侦测时钟速度的函数有：

- **CS_getACLK()**
- **CS_getSMCLK()**
- **CS_getMCLK()**

CS 错误标志处理的函数有：

- **CS_getFaultFlagStatus()**

函数 CS_getMCLK,CS_getSMCLK,CS_getACLK 只在使用内部或外部电阻或者 16.384MHz 的旁路时钟频率下使用 DCO 才是精确有效的。

详细描述

uint32_t CS_getACLK(void)

获取当前 ACLK 频率（单位 Hz）。当设备安装在 DCO 旁路模式下，它不能正常工作。此外，使用这个 API 前应该调用 CS_setupDCO()，以便 DCO 被校准，这样计算才是精确的。

返回值：当前 ACLK 频率，单位 Hz,当旁路模式下返回 0。

uin8_t CS_getFaultFlagStatus(uint8_t mask)

获取 DCO 错误标志状态。当 DCO 在外部电阻模式和 DCO 检测到异常时候，DCO 故障标志将被置位（写 1）。异常可能是，ROSC 端子保持开路或者短接到地，或者连接在 ROSC 端子的电阻远离推荐的值。如果故障仍然存在，DCO 就会自动切换到内部电阻模式作为一种处理故障的安全机制。

mask:mask 参数有 CS_DCO_FAULT_FLAG.

返回值：CS_DCO_FAULT_FLAG。说明错误标志被置位。

uint32_t CS_getMCLK(void)

获取当前 MCLK 频率（单位 Hz）。当设备安装在 DCO 旁路模式下，它不能正常工作。此外，使用这个 API 前应该调用 CS_setupDCO()，以便 DCO 被校准，这样计算才是精确的。

返回值：当前 MCLK 频率，单位 Hz,当旁路模式下返回 0。

uint32_t CS_getSMCLK(void)

获取当前 SMCLK 频率（单位 Hz）。当设备安装在 DCO 旁路模式下，它不能正常工作。此外，使用这个 API 前应该调用 CS_setupDCO()，以便 DCO 被校准，这样计算才是精确的。

返回值：当前 SMCLK 频率，单位 Hz,当旁路模式下返回 0。

void CS_initClockSignal(uint8_t clockSource, uint8_t clockSourceDivider)

使用分频器初始化一个时钟信号。

如果 DCO 是在旁路模式下，频率将是 CLKIN/分频器。如果 DCO 不在旁路模式下，频率将是 16.384MHz/分频器。

该函数有两个参数：clockSource 和 clockSourceDivider。

clockSource 时钟信号初始化值可以选的值有：CS_MCLK，CS_SMCLK。

clockSourceDivider 分频器设置可选的值有： CS_CLOCK_DIVIDER_1 ，CS_CLOCK_DIVIDER_2 ， CS_CLOCK_DIVIDER_4 ， CS_CLOCK_DIVIDER_8 ，CS_CLOCK_DIVIDER_16。

返回值：无。

void CS_setupDCO(uint8_t mode)

使用参数所选模式配置 DCO。如果选择旁路模式，则需要在 CLKIN 管脚接入外部数字时钟信号来作为所有的设备（CPU、外设等）的时钟信号。ACLK 频率是不可以被编程的，且固定在旁路时钟频率除以 512。使用外部吊足模式，需要在 ROSC 管脚连接一个 20KΩ的电阻器。与使用内部电阻相比，使用外部电阻模式，在绝对误差和温度漂移上，可以提供更高的时钟精度。请根据你所选的设备型号对应的数据手册的详细情况来选择不同的模式。

该函数只有一个参数：mode。

mode：该参数可以选择的量有 CS_INTERNAL_RESISTOR，CS_EXTERNAL_RESISTOR，CS_BYPASS_MODE。

返回值：空。

注释：DCO 的配置可以使用内部电阻和外部电阻，还有一个就是旁路模式，旁路模式就是可以通过一个时钟输入管脚将外部时钟信号灌入系统。

3.3 例程

下面的例程演示如何配置 CS 模块，设置为 SMCLK=DCO/2，MCLK=DCO/8。

```
//先配置 DCO 频率为 16.384MHz.
CS_setupDCO(CS_INTERNAL_RESISTOR);
//配置完 DCO 才可以配置 MCLK 和 SMCLK
CS_initClockSignal(CS_MCLK,   CS_CLKOCK_DIVIDER_8);
CS_initClockSignal(CS_SMCLK,   CS_CLKOCK_DIVIDER_2);
```

第四章　EUSCI 通用异步接收器/发送器（EUSCI_A_UART）

4.1　本章引言

MSP430i2xx 系列的 EUSC_A_UART 驱动库特性包括：

- 奇偶校验或非奇偶校验
- 独立的发送和接收移位寄存器
- 分立的发送和接收缓冲寄存器
- 低位优先或高位优先数据发送和接收
- 内置空闲线和地址位通信协议的多处理器系统
- 具有从 LPMx 模式自动唤醒接收器启动的边缘检测
- 用于错误检测和抑制的状态标志
- 地址检测的状态标志
- 用于接收和发送独立的中断能力

在 UART 模式中，eUSCI 在一定的位速率下，异步与另外一个设备进行字符的发送和接收。每个字符的时间长度是基于所选择的 eUSCI 的波特率所固定的。因此，发送和接收函数要使用相同的波特率进行通信。

这个驱动程序包含在 eusci_a_uart.c 文件里，eusci_a_uart.h 头文件包含该应用程序使用的 API 定义。

4.2　函数总览

1	**bool EUSCI_A_UART_init (uint16_t baseAddress,EUSCI_A_UART_initParam *param)**
	先进的 UART 模块初始化程序，把初始化参数通过初始化函数写进时钟预分频器。
2	**void EUSCI_A_UART_transmitData (uint16_t baseAddress, uint8_t transmitData)**
	从 UART 模块发送出去一个字节
3	**void EUSCI_A_UART_receiveData (uint16_t baseAddress)**
	接收一个字节数据。
4	**void EUSCI_A_UART_enableInterrupt (uint16_t baseAddress,uint8_t mask)**
	使能 UART（独立的）中断源
5	**void EUSCI_A_UART_disableInterrupt (uint16_t baseAddress,uint8_t mask)**
	关闭 UART（独立的）中断源
6	**void EUSCI_A_UART_getInterruptStatus (uint16_t baseAddress,uint8_t mask)**
	获取当前 UART 中断状态
7	**void EUSCI_A_UART_clearInterruptStatus (uint16_t baseAddress,uint8_t mask)**
	清除 UART 中断源。（备注：清除中断状态，重置中断状态标识）
8	**void EUSCI_A_UART_enable (uint16_t baseAddress)**
	启用 UART 模块
9	**void EUSCI_A_UART_disable (uint16_t baseAddress)**
	关闭 UART 模块
10	**uint8_t EUSCI_A_UART_queryStatusFlags (uint16_t baseAddress, uint8_t mask)**
	获取当前 UART 状态标志.(查询 UART 当前状态标识)
11	**void EUSCI_A_UART_setDormant(uint16_t baseAddress)**
	把 UART 模块设置在休眠模式
12	**void EUSCI_A_UART_resetDormant(uint16_t baseAddress)**
	把 UART 模块从休眠模式唤醒
13	**void EUSCI_A_UART_transmitAddress(uint16_t baseAddress,uint8_t transmitAddress)**
	根据所选的多处理器模式，传送下一个字节标记为地址
14	**void EUSCI_A_UART_transmitBreak(uint16_t baseAddress)**
	发送终止
15	**uint32_t EUSCI_A_UART_getReceiveBufferAddress(uint16_t baseAddress)**
	返回 RX 缓冲区的 UART 的 DMA 模块的地址。
16	**uint32_t EUSCI_A_UART_getTransmitBufferAddress(uint16_t baseAddress)**
	返回 TX 缓冲区的 UART 的 DMA 模块的地址。
17	**void EUSCI_A_UART_selectDeglitchTime(uint16_t baseAddress,uint16_t deglitchTime)**
	设置抗尖峰脉冲时间

EUSI_A_UART_API 提供了一组函数，用来实现一个中断驱动 EUSI_A_UART 的驱动程序。该 EUSI_A_UART 初始化的各种模式和功能是由 EUSCI_A_UART_init()完成。这个函数初始化结束

时 EUSI_A_UART 保持禁用（所有的相关配置工作完成后才会通过函数启动 UART）。EUSCI_A_UART_enable()使 EUSI_A_UART，模块现在已经可以准备传输和接收了。

建议通过 EUSCI_A_UART_init() 初始化 EUSI_A_UART, 使能所需的中断，然后通 EUSCI_A_UART_enable()启用 EUSI_A_UART。

EUSI_A_UART API 函数分为三组:那些处理 EUSI_A_UART 模块配置和控制的,用于发送和接收数据的，管理中断和状态的。

配置和控制 EUSI_UART 的函数有:

- EUSCI_UART_init()
- EUSCI_UART_enable()
- EUSCI_UART_disable()
- EUSCI_UART_setDormant()
- EUSCI_UART_resetDormant()
- EUSCI_UART_selectDeglithTime()

通过 EUSI_UART 发送和接收数据的函数有：

- EUSCI_UART_transmitData()
- EUSCI_UART_receiveData()
- EUSCI_UART_transmitAddress()
- EUSCI_UART_transmitBreak()

管理 EUSI_UART 中断和状态的函数有：

- EUSCI_UART_enableInterrupt()
- EUSCI_UART_disableInterrupt()
- EUSCI_UART_geInterruptStatus()
- EUSCI_UART_clearInterrupt()
- EUSCI_UART_queryStatusFlags()

void EUSCI_A_UART_clearInterruptStatus (uint16_t baseAddress,uint8_t mask)

清除 UART 中断源。

UART 中断源被清除，所以它不再断言（计算机专业术语，大概意思就是：中断源被清除后，相关的不再执行，根据我们所接触过的类似的情况，可以看出来，清除中断标志后，才可以重新接收新一次的中断信号，本次中断信号已经完成，不再有效）。

该函数共两个参数：baseAddress 和 mask。

参数

baseAddress	是 EUSCI_A_UART 模块的基地址
mask	将被清除中断源的位掩码。掩码值可以是以下量的逻辑或： EUSCI_A_UART_RECEIVE_INTERRUPT_FLAG EUSCI_A_UART_TRANSMIT_INTERRUPT_FLAG EUSCI_A_UART_STARTBIT_INTERRUPT_FLAG EUSCI_A_UART_TRANSMIT_COMPLETE_INTERRUPT_FLAG

该函数修改 UCAxIFG 寄存器。

返回值：无。

void EUSCI_A_UART_disable (uint16_t baseAddress)

关闭 UART 模块。

该函数只有 1 个参数：baseAddress。

参数

baseAddress	是 EUSCI_A_UART 模块的基地址

该函数修改 UCAxCTL1 寄存器的 UCSWRST 位。

返回值：无。

void EUSCI_A_UART_disableInterrupt (uint16_t baseAddress, uint8_t mask)

关闭独立的 UART 中断源。

禁用表示的 UART 中断源。只有启用的源可以反映给处理器进行中断。禁用的源不会影响到处理器的工作。

该函数共两个参数：baseAddress 和 mask。

参数

baseAddress	是 EUSCI_A_UART 模块的基地址
mask	掩码位对应的中断源将会被关闭。掩码值可以是以下量的逻辑或： EUSCI_A_UART_RECEIVE_INTERRUPT 接收中断

	EUSCI_A_UART_TRANSMIT_INTERRUPT	发送中断
	EUSCI_A_UART_RECEIVE_ERRONEOUSCHAR_INTERRUPT	收到错误的字符中断使能
	EUSCI_A_UART_BREAKCHAR_INTERRUPT	接收间隔字符中断使能
	EUSCI_A_UART_STARTBIT_INTERRUPT	起始位接收中断启用
	EUSCI_A_UART_TRANSMIT_COMPLETE_INTERRUPT	发送完成中断使

该函数修改 UCAxCTL1 寄存器的 UCAxIE 位。

返回值：无。

void EUSCI_A_UART_enable (uint16_t baseAddress)

启用 UART 模块，这将能够操作 UART 模块。

该函数只有 1 个参数：baseAddress。

参数

baseAddress	是 EUSCI_A_UART 模块的基地址

该函数修改 UCAxCTL1 寄存器的 UCSWRST 位。

返回值：无。

void EUSCI_A_UART_enableInterrupt (uint16_t baseAddress, uint8_t mask)

启用独立的 UART 中断源。

只有这个源使能启用后才能够反映到处理器中断；关闭源将不再影响到处理器。

注释：通过该函数启动中断源，才可以把中断事件产生的中标标志置位反映到处理器响应中断，进行中断，如果关闭了中断源，系统将不会把中断标志置位的信号传递给处理器，也不会响应中断。这就是为什么，总是看到，启动中断程序前都进行了中断标志位清零操作。

该函数共两个参数：baseAddress 和 mask。

参数

baseAddress	是 EUSCI_A_UART 模块的基地址
mask	掩码位对应的中断源将会被关闭。掩码值可以是以下量的逻辑或：

	EUSCI_A_UART_RECEIVE_INTERRUPT 接收中断
	EUSCI_A_UART_TRANSMIT_INTERRUPT 发送中断
	EUSCI_A_UART_RECEIVE_ERRONEOUSCHAR_INTERRUPT 收到错误的字符中断使能
	EUSCI_A_UART_BREAKCHAR_INTERRUPT 接收间隔字符中断使能
	EUSCI_A_UART_STARTBIT_INTERRUPT 起始位接收中断启用
	EUSCI_A_UART_TRANSMIT_COMPLETE_INTERRUPT 发送完成中断使

该函数修改 UCAxCTL1 寄存器的 UCAxIE 位。

返回值：无。

uint8_t EUSCI_A_UART_getInterruptStatus (uint16_t baseAddress, uint8_t mask)

获取当前 UART 中断状态。

这将为 UART 模块返回基于其传递的标志的中断状态。

该函数共两个参数：baseAddress 和 mask。

参数

baseAddress	是 EUSCI_A_UART 模块的基地址
mask	返回掩码的中断标志位状态。掩码值可以是以下量的逻辑或： EUSCI_A_UART_RECEIVE_INTERRUPT_FLAG EUSCI_A_UART_TRANSMIT_INTERRUPT_FLAG EUSCI_A_UART_STARTBIT_INTERRUPT_FLAG EUSCI_A_UART_TRANSMIT_COMPLETE_INTERRUPT_FLAG

该函数修改 UCAxIFG 寄存器。

返回值

下面量的逻辑或：

EUSCI_A_UART_RECEIVE_INTERRUPT_FLAG

EUSCI_A_UART_TRANSMIT_INTERRUPT_FLAG

EUSCI_A_UART_STARTBIT_INTERRUPT_FLAG

EUSCI_A_UART_TRANSMIT_COMPLETE_INTERRUPT_FLAG

指示掩码标志状态。

uint32_t EUSCI_A_UART_getReceiveBufferAddress (uint16_t baseAddress)

为 DMA 模块返回 UART 的 RX 缓冲器地址。

这可以结合使用 DMA 直接接收到的数据存储到内存。

参数

baseAddress	是 EUSCI_A_UART 模块的基地址

返回值：RX 缓冲器地址。

uint32_t EUSCI_A_UART_getTransmitBufferAddress (uint16_t baseAddress)

为 DMA 模块返回 UART 的 TX 缓冲器地址。

这可以结合使用 DMA 直接接收到的数据存储到内存。

参数

baseAddress	是 EUSCI_A_UART 模块的基地址

返回值：TX 缓冲器地址。

bool EUSCI_A_UART_init (uint16 t baseAddress, EUSCI A UART initParam * param)

先进的 UART 模块初始化程序。

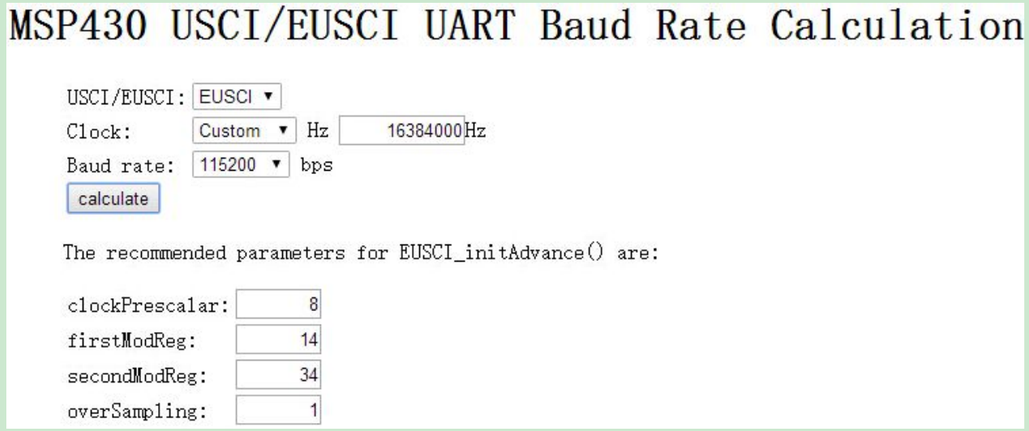
被写进 clockPrescalar(前置分频器),firstModReg，secondModRge 和 overSampling（过采样）的参数应该提前计算好再传递给初始化函数。

注释：oversampling 指的是在对模拟信号进行采样的时候，采样频率比被采样信号的最大频率成分的两倍要高，即满足奈奎斯特采样定理（fs>=fmax）。

在成功初始化 UART 模块前，这个函数将完成初始化该模块，单 UART 模块仍然是关闭的，必须使用函数 EUSCI_A_UART_enable()使能启动。对于计算 clockPrescalar，firstModReg，secondModReg 和 overSampling 请使用下面链接：

http://software-dl.ti.com/msp430/msp430_public_sw/mcu/msp430/MSP430BaudRateConverter/index.html

根据网页提供的工具可以方便的计算出初始化用的参数。如下图所示。



参数

baseAddress	是 EUSCI_A_UART 模块的基地址
mask	param 是初始化结构体的指针。

该函数修改寄存器 **UCAxCTL0** 的 **UCPEN,UCPAR,UCMSB,UC7BIT,UCSPB,UCMODEx 和 UCSYNC** 位。

返回值 STATUS_SUCCESS 或 STATUS_FAIL。

注释：如果初始化成功了，返回 **STATUS_SUCCESS**；如果初始化写入失败了，返回 **STATUS_FAIL**。

uint8 t EUSCI_A_UART_queryStatusFlags (uint16 t baseAddress, uint8 t mask)

获取当前 UART 状态标志。

该函数返回 UART 模块作为参数传送的标志位的状态。

注释：多次提到传送标志位的概念，意思就是，该函数中作为参数传送过去的标志位对应的当前状态将会作为返回值返回。

参数

baseAddress	是 EUSCI_A_UART 模块的基地址
mask	是将被返回的中断标志位状态掩码。 EUSCI_A_UART_LISTEN_ENABLE EUSCI_A_UART_FRAMING_ERROR EUSCI_A_UART_OVERRUN_ERROR EUSCI_A_UART_BREAK_DETECT EUSCI_A_UART_ADDRESS_RECEIVED EUSCI_A_UART_IDLELINE EUSCI_A_UART_BUSY

该函数修改寄存器 UCAxSTAT 的位。

返回值：下面量的逻辑或。

- EUSCI_A_UART_LISTEN_ENABLE
- EUSCI_A_UART_FRAMING_ERROR
- EUSCI_A_UART_OVERRUN_ERROR
- EUSCI_A_UART_BREAK_DETECT
- EUSCI_A_UART_ADDRESS_RECEIVED
- EUSCI_A_UART_IDLELINE
- EUSCI_A_UART_BUSY

uint8_t EUSCI_A_UART_receiveData (uint16_t baseAddress)

接收已经发送到 UART 模块的一个字节。该函数从 UART 接收数据寄存器读取一个字节数据。

参数

baseAddress	是 EUSCI_A_UART 模块的基地址

该函数修改寄存器 UCAxRXBUF。

返回值：返回从 UART 模块收到的字节，强制转换为 uint8_t。

注释：函数的类型就是返回值的类型，函数内部返回值必须以函数类型返回。

void EUSCI_A_UART_resetDormant (uint16_t baseAddress)

从休眠模式重启 UART 模块。不休眠，所有接收到字符就置位 UCRXIFG。

参数

baseAddress	是 EUSCI_A_UART 模块的基地址

该函数修改寄存器 UCAxCTL1 的 UCDORM 位。

返回值：无

void EUSCI_A_UART_selectDeglitchTime (uint16 t baseAddress, uint16 t deglitchTime)

设置抗尖峰脉冲时间。

参数

baseAddress	是 EUSCI_A_UART 模块的基地址
deglitchTime	抗尖峰脉冲时间可选的值有： EUSCI_A_UART_DEGLITCH_TIME_2ns EUSCI_A_UART_DEGLITCH_TIME_50ns EUSCI_A_UART_DEGLITCH_TIME_100ns EUSCI_A_UART_DEGLITCH_TIME_200ns

返回值：无

void EUSCI_A_UART_setDormant (uint16_t baseAddress)

设置 UART 在休眠模式。在空闲线（串口线路闲置状态）或 UCRXIFG 置位前。

在 UART 自动波特率检测模式，只有断点和同步字段组合才可置位 UCRXIFG（在这种模式下，触发中断的条件）。

参数

baseAddress	EUSCI_A_UART 模块的基地址

该函数修改寄存器 UCAxCTL1。

返回值：无

void EUSCI_A_UART_transmitAddress (uint16_t baseAddress, uint8_t transmitAddress)

根据选择的多处理器模式，发送要被发送的标记为地址的下一个字节。

参数

baseAddress	EUSCI_A_UART 模块的基地址
transmitAddress	被发送的下一个字节

该函数修改寄存器 UCAxTXBUF 和 UCAxCTL1。

返回值：无

void EUSCI_A_UART_transmitBreak (uint16_t baseAddress)

传输中断（停止，暂停，间断）。

传输中断（停止）作为下一个写入发送缓冲器。在 UART 的自动波特率检测模式下，EUSCI_A_UART_AUTOMATICBAUDRATE_SYNC(0x55) 必须被写进 UCAxTXBUF 来生成所需的中断/同步字段。否则,默认同步(0x00)必须写入传输缓冲区。另外确保模块为发送下一个数据做好准备。

参数

baseAddress	是 EUSCI_A_UART 模块的基地址

该函数修改寄存器 **UCAxTXBUF 和 UCAxcTL1**。

返回值：无

void EUSCI_A_UART_transmitData (uint16_t baseAddress, uint8_t transmitData)

从 UART 模块发送一个字节。

该函数放置提供的数据在 UART 发送数据寄存器里，并开始发送。

参数

baseAddress	是 EUSCI_A_UART 模块的基地址
transmitData	从 UART 模块将要被发送出去的数据

该函数修改寄存器 UCAxTXBUF。

返回：无。

4.3 例程

例程将展示怎样使用 EUSCI_A_UART API 来初始化 EUSCI_A_UART 并开始发送字符。

```
// 使用 SMCLK 频率为 16384000Hz 配置 UART 模块波特率为 115200
// 可以在以下网址计算器计算出配置参数:
// http://software-dl.ti.com/msp430/msp430 public sw/mcu/msp430/MSP430BaudRateConverter/index.html
EUSCI_A_UART_ initParam uartConfig = {
EUSCI_A_UART_ CLOCKSOURCE SMCLK, // SMCLK Clock Source
8, // BRDIV = 8
14, // UCxBRF = 14
34, // UCxBRS = 34
EUSCI_A_UART_ NO PARITY, // No Parity
EUSCI_A_UART_ MSB FIRST, // MSB First
EUSCI_A_UART_ ONE STOP BIT, // One stop bit
EUSCI_A_UART_ MODE, // UART mode
EUSCI_A_UART_ OVERSAMPLING_BAUDRATE_GENERATION // Oversampling Baudrate
};
WDT_hold(WDT_BASE);
// 设置 DCO 使用内部电阻，DCO 将被配置在 16.384MHz.
CS_setupDCO(CS_INTERNAL RESISTOR);
// SMCLK 设置与 DCO 相同的速度。SMCLK = 16.384MHz
CS_initClockSignal(CS_SMCLK, CS_CLOCK DIVIDER_1);
// 设置 P1.2 和 P1.3 管脚作为 UART 管脚。P1.4 管脚作为 LED 输出
GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P1, GPIO_PIN2 | GPIO_PIN3, GPIO_PRIMARY_MODULE_FUNCTION);
GPIO_setAsOutputPin(GPIO_PORT_P1, GPIO_PIN4);
GPIO_setOutputLowOnPin(GPIO_PORT_P1, GPIO_PIN4);
// 配置和使能 UART 外设
EUSCI_A_UART_ init(EUSCI_A0_BASE, &uartConfig);
EUSCI_A_UART_ enable(EUSCI_A0_BASE);
EUSCI_A_UART_ enableInterrupt(EUSCI_A0_BASE, EUSCI_A_UART_ RECEIVE_INTERRUPT);
while(1) {
EUSCI_A_UART_ transmitData(EUSCI_A0_BASE, TXData);
//进入休眠并等待退出 LPM
__bis_SR_register(LPM0_bits | GIE);
}
```

4.4 详解

UART 主要是通过一个结构体来初始化的。

从 eusci_a_uart.h 我们可以看到该结构体为

```
typedef struct EUSCI_A_UART_initParam
{
uint8_t selectClockSource;
uint16_t clockPrescalar;
uint8_t firstModReg;
uint8_t secondModReg;
uint8_t parity;
uint16_t msborLsbFirst;
uint16_t numberofStopBits;
uint16_t uartMode;
uint8_t overSampling;
} EUSCI_A_UART_initParam;
```

根据库函数头文件的介绍，第一个时钟源选择变量，一共有两个值可以选择。分别是使用 SMCLK 和 ACLK 作为 UART 时钟源。

第二个，第三个，第四个，我们可以不用管，直接利用网页的工具进行计算，如果想知道怎么手工计算，请查看技术手册。

第五个 **parity** 是奇偶校验，一共三个选项，无奇偶校验、奇校验和偶校验。默认情况是无奇偶校验。

第六个是 **msborLsbFirs**，高位优先或低位优先，默认低位优先。

第七个是 **numberofStopBits**，停止位数量，可以选择 1 个停止位或 2 个选择位。默认 1 个停止位。

第八个是 **uartMode**，共 4 个模式，默认是 **EUSCI_A_UART_MODE** 模式，还可以选择 **EUSCI_A_UART_IDLE_LINE_MULTI_PROCESSOR_MODE**（空闲线多处理器模式）、**EUSCI_A_UART_ADDRESS_BIT_MULTI_PROCESSOR_MODE**（地址位多处理器模式）和 **EUSCI_A_UART_AUTOMATIC_BAUDRATE_DETECTION_MODE**（自动波特率检测模式）。

第九个是 **overSampling**，有两个值可以选择，分别用来指示使用过采样波特率发生器还是使用低频率波特率发生器。分别是 **EUSCI_A_UART_OVERSAMPLING_BAUDRATE_GENERATION** 和 **EUSCI_A_UART_LOW_FREQUENCY_BAUDRATE_GENERATION**。

详情参见技术手册相关章节。并在本帖后回复讨论。

关于本章节函数的参数变量的选择，详情见 eusci_a_uart.h。

本章节的作业：根据例程，编写串口发送和接收程序，发送 LED_ON 字符串点亮接收 MCU 的 LED,发送 LED_OFF,关闭接收 MCU 的 LED，发送 LED_TEST 字符串，返回当前的 LED 状态。