**MSP430学习总结+MDDS库使用指南**

苏子梁

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 日期 | 描述 | 颜色 |
| 0.0.1 | 2019/2/24 | 对MSP430的基本支持 | 黑 |
| 0.1.1 | 2019/3/16 | 对STM32的支持，窗口和标准按键 | 蓝 |

\*STM32的相关函数由STM32CUBE生成，此值代表上层函数对底层函数的支持程度

目录

[1. 概览 8](#_Toc3707556)

[1.1. 对MSP430的支持 8](#_Toc3707557)

[1.2. 对STM32的支持 8](#_Toc3707558)

[1.3. 库结构 8](#_Toc3707559)

[1.4. 使用方法 11](#_Toc3707560)

[2. MSP430G2结构 11](#_Toc3707561)

[2.1. CPU 11](#_Toc3707562)

[2.2. 时钟 13](#_Toc3707563)

[2.3. 数字IO 14](#_Toc3707564)

[2.4. 定时器 14](#_Toc3707565)

[2.5. USCI模块 15](#_Toc3707566)

[2.6. ADC10模块 16](#_Toc3707567)

[3. MSP430芯片驱动（SOCDriver） 18](#_Toc3707568)

[3.1. 初始化模块 18](#_Toc3707569)

[3.1.1. SystemInit函数 18](#_Toc3707570)

[3.1.2. IdleProcess：空闲处理 18](#_Toc3707571)

[3.2. 时钟与低功耗模式 18](#_Toc3707572)

[3.2.1. 配置 18](#_Toc3707573)

[3.2.2. 枚举类型 19](#_Toc3707574)

[3.2.3. SysClockInit 21](#_Toc3707575)

[3.2.4. PowerPolicySet 21](#_Toc3707576)

[3.2.5. getClockFreq 21](#_Toc3707577)

[3.3. 通用IO口 21](#_Toc3707578)

[3.3.1. 配置 21](#_Toc3707579)

[3.3.2. 枚举和宏 22](#_Toc3707580)

[3.3.3. GPIOInit：以conf.h初始化GPIO 22](#_Toc3707581)

[3.3.4. GPIOWrite：写GPIO 22](#_Toc3707582)

[3.3.5. GPIOWritePin: 写GPIO部分位 23](#_Toc3707583)

[3.3.6. GPIORead: 读GPIO 23](#_Toc3707584)

[3.3.7. GPIOReadPin: 读GPIO部分位 23](#_Toc3707585)

[3.3.8. GPIOSetMode: 设置GPIO模式 23](#_Toc3707586)

[3.3.9. GPIOISR: 中断处理程序 23](#_Toc3707587)

[3.4. 定时器 24](#_Toc3707588)

[3.4.1. 配置 24](#_Toc3707589)

[3.4.2. 枚举 24](#_Toc3707590)

[3.4.3. TimerAInit：时基初始化 26](#_Toc3707591)

[3.4.4. TimerAModuleInit：额外模块初始化 26](#_Toc3707592)

[3.4.5. TimerABaseFreqSet：时基频率设置 26](#_Toc3707593)

[3.4.6. Timer0ABaseMaxCountSet：时基频率设置 26](#_Toc3707594)

[3.4.7. TimerAStart：定时器启动 27](#_Toc3707595)

[3.4.8. TimerAStop：定时器停止 27](#_Toc3707596)

[3.4.9. TimerAModulePWMSet ：PWM模式设置占空比 27](#_Toc3707597)

[3.4.10. TimerAModuleCaptureInit：捕获模式初始化 27](#_Toc3707598)

[3.4.11. TimerAModuleCompareInit：比较模式初始化 27](#_Toc3707599)

[3.4.12. TimerABaseInterruptHook：时基中断处理函数 28](#_Toc3707600)

[3.4.13. Timer1ABaseFreqGet：获取时基溢出频率 28](#_Toc3707601)

[3.5. DMA：直接内存访问 28](#_Toc3707602)

[3.5.1. 配置 28](#_Toc3707603)

[3.5.2. 枚举 29](#_Toc3707604)

[3.5.3. 结构 31](#_Toc3707605)

[3.5.4. DMAInit：初始化DMA 31](#_Toc3707606)

[3.5.5. DMAStaticStart：静态通道启动 31](#_Toc3707607)

[3.5.6. DMADynamicStart：动态通道启动 31](#_Toc3707608)

[3.6. 基本定时器接口 32](#_Toc3707609)

[3.6.1. 配置 32](#_Toc3707610)

[3.6.2. BasicTimerInit：初始化 32](#_Toc3707611)

[3.6.3. BasicTimerAdd：添加一个定时器 32](#_Toc3707612)

[3.6.4. BasicTimerDel：删除一个定时器 32](#_Toc3707613)

[3.7. USCI：硬件SPI、UART和IIC 33](#_Toc3707614)

[3.7.1. 配置 34](#_Toc3707615)

[3.7.2. 枚举 35](#_Toc3707616)

[3.7.3. HSPIInit ：HSPI初始化 35](#_Toc3707617)

[3.7.4. HSPIWaitTransmit ：HSPI等待数据发送完成 35](#_Toc3707618)

[3.7.5. HSPISetCallBack ：HSPI设置回调函数 35](#_Toc3707619)

[3.7.6. HSPISent ：HSPI发送数据 35](#_Toc3707620)

[3.7.7. UARTInit ：UART初始化 35](#_Toc3707621)

[3.7.8. UARTWaitTransmit： UART等待数据发送完成 36](#_Toc3707622)

[3.7.9. UARTSetTxCallBack： UART设置回调函数 36](#_Toc3707623)

[3.7.10. UARTWriteByte: UART发送数据 36](#_Toc3707624)

[3.7.11. IICInit ：IIC初始化 36](#_Toc3707625)

[3.7.12. IICWaitTransmit IIC等待数据发送完成 36](#_Toc3707626)

[3.7.13. IICWaitReceive IIC等待接收一个数据（仅用于无中断模式） 36](#_Toc3707627)

[3.7.14. IICSetTxCallBack IIC设置回调函数 37](#_Toc3707628)

[3.7.15. IICSetRxCallBack IIC设置回调函数 37](#_Toc3707629)

[3.7.16. IICStartWrite IIC写入开始 37](#_Toc3707630)

[3.7.17. IICStop IIC停止 37](#_Toc3707631)

[3.7.18. IICNoAckModeSet IICNoAck模式 37](#_Toc3707632)

[3.7.19. IICWriteByte IIC写入字节 37](#_Toc3707633)

[3.7.20. IICReadByte IIC读取一个字节 37](#_Toc3707634)

[3.8. RTC：实时时钟 37](#_Toc3707635)

[3.8.1. 配置 37](#_Toc3707636)

[3.8.2. RTCInit：初始化 38](#_Toc3707637)

[3.8.3. RTCTimeSet：设置时间 38](#_Toc3707638)

[3.8.4. RTCTimeGet：获取时间 38](#_Toc3707639)

[3.8.5. RTCInterrupt：RTC计时中断函数 38](#_Toc3707640)

[4. 驱动 38](#_Toc3707641)

[4.1. SSPI：软件SPI 38](#_Toc3707642)

[4.1.1. 配置 38](#_Toc3707643)

[4.1.2. SSPIInit：初始化SSPI（无作用） 39](#_Toc3707644)

[4.1.3. SSPISent ：SSPI发送数据 39](#_Toc3707645)

[4.1.4. SSPIWaitTransmit：SSPI等待数据发送完成（无作用） 39](#_Toc3707646)

[4.2. Font：字体 39](#_Toc3707647)

[4.2.1. ASCII，0816 39](#_Toc3707648)

[4.3. LX\_12864 40](#_Toc3707649)

[4.3.1. 配置 40](#_Toc3707650)

[4.3.2. LCDInit ：LCD初始化 40](#_Toc3707651)

[4.3.3. LCDClear：清LCD 40](#_Toc3707652)

[4.3.4. LCDWriteImageRaw ：LCD写入图像 40](#_Toc3707653)

[4.3.5. LCDWriteCharRaw ：LCD写入字符 40](#_Toc3707654)

[4.3.6. LCDWriteStringRaw ：LCD写入字符串 41](#_Toc3707655)

[4.4. DAC8411 41](#_Toc3707656)

[4.4.1. 配置 41](#_Toc3707657)

[4.4.2. DAC8411Init：初始化 41](#_Toc3707658)

[4.4.3. DAC8411Write写数据 42](#_Toc3707659)

[4.4.4. DAC8411AutoWriteInit：自动写入初始化 42](#_Toc3707660)

[4.4.5. DAC8411AutoWrite：自动写入执行函数 42](#_Toc3707661)

[4.5. HT1621液晶驱动器 42](#_Toc3707662)

[4.5.1. 配置 42](#_Toc3707663)

[4.5.2. HT1621Init LCD初始化 42](#_Toc3707664)

[4.5.3. HT1621Refresh屏幕刷新 42](#_Toc3707665)

[4.5.4. HT1621SegmentListAppend改变点亮的段 43](#_Toc3707666)

[4.6. SegLCD\_G2液晶 43](#_Toc3707667)

[4.6.1. LCDDigitDisplay：显示数字 43](#_Toc3707668)

[4.7. TCA6416串行IO扩展 43](#_Toc3707669)

[4.7.1. 配置 43](#_Toc3707670)

[4.7.2. TCA6416A\_Init：初始化 43](#_Toc3707671)

[4.7.3. EXIOWrite写IO 44](#_Toc3707672)

[4.7.4. EXIOWritePin：写IO部分位 44](#_Toc3707673)

[4.7.5. EXIORead：读IO 44](#_Toc3707674)

[4.7.6. EXIOReadPin：读IO部分位 44](#_Toc3707675)

[5. 中间件 44](#_Toc3707676)

[5.1. RouteBuffer.h：环形缓冲 44](#_Toc3707677)

[5.1.1. Buf缓冲区名W：向缓冲区写入 44](#_Toc3707678)

[5.1.2. Buf缓冲区名R：读取缓冲区 45](#_Toc3707679)

[5.1.3. Buf缓冲区名RB：读取缓冲区（阻塞式） 45](#_Toc3707680)

[5.1.4. Buf缓冲区名RB：写缓冲区（阻塞式） 45](#_Toc3707681)

[5.1.5. BufInit初始化环形缓冲区 45](#_Toc3707682)

[5.2. Stream：流 45](#_Toc3707683)

[5.3. AbstractButton：标准按键 45](#_Toc3707684)

[5.3.1. 配置 46](#_Toc3707685)

[5.3.2. 46](#_Toc3707686)

[5.4. GUI：图形用户界面 46](#_Toc3707687)

[5.5. LinkedList：链表 46](#_Toc3707688)

[6. 数学函数 46](#_Toc3707689)

[6.1. Base64 46](#_Toc3707690)

[6.1.1. 配置 46](#_Toc3707691)

[6.1.2. Base64Encode：数据转Base64字符串。 46](#_Toc3707692)

[6.1.3. Base64Decode：数据转Base64字符串。 47](#_Toc3707693)

[6.2. Time：时间戳和时间工具 47](#_Toc3707694)

[6.2.1. ConventTimestampToTime：转换时间戳为时间 47](#_Toc3707695)

[6.2.2. ConventTimestampToUnixTimestamp：转换时间戳为UNIX时间戳 47](#_Toc3707696)

[7. 标准库设计总结 48](#_Toc3707697)

# 概览

MDDS库旨在为跨设备应用提供标准的接口和常被复用的代码，以减小应用开发和移植的难度。

MDDS库为一个包含片上外设驱动、常用外设驱动、中间件、数学函数的大型库。

## 对MSP430的支持

~~MSP430为RISC型单片机，结构与51单片机存在差别，但在c语言编程中基本没有影响。其使用的c语言为标准c语言的拓展，例如中断向量使用预处理#pragma指定。~~

MDDS库为MSP430外设常使用的功能提供了标准函数，并提供一个配置文件以自动完成对外设的初始化。

大部分的上层函数使用MSP430标准函数的接口标准，因此在MSP430上运行有最高的程序效率。

## 对STM32的支持

MDDS库对STM32的支持稍晚，运行效率较低。因STM32Cube使用方便，配置简单，将STM32的配置交由STM32Cube完成，对底层函数的调用使用一个转换层实现。

## 库结构

库包含3个部分，芯片驱动（SOCDriver）、器件驱动（Driver）、中间件（Middleware）。

各个部分由各模块的一对头文件源文件组成，为便于使用，所有头文件包含于includes.h中。除此之外，还有conf.h、conf\_driver.h、conf\_middleware.h、conf\_Math.h用于初始化时的配置。

为保证可移植性，使用长度确定的类型，类型如下表。

|  |  |  |
| --- | --- | --- |
| 类型关键字 | 类型 | 长度 |
| INT8U | 无符号整数 | 8 |
| INT8S | 有符号整数 |
| INT16U | 无符号整数 | 16 |
| INT16S | 有符号整数 |
| INT32U | 无符号整数 | 32 |
| INT32S | 有符号整数 |
| INT64U | 无符号整数 | 64 |
| INT64S | 有符号整数 |
| BL | 布尔型 | ≥1 |
| F32 | 浮点型 | ≥32 |
| F64 | 浮点型 | ≥64 |

支持的其他类型如下：

|  |  |  |
| --- | --- | --- |
| 类型关键字 | 类型 | 长度 |
| int8\_t | 有符号整数 | 8 |
| uint8\_t | 无符号整数 | 8 |
| int16\_t | 有符号整数 | 16 |
| uint16\_t | 无符号整数 | 16 |
| int32\_t | 有符号整数 | 32 |
| uint32\_t | 无符号整数 | 32 |
| size\_t | 无符号整数 | 依设备而定 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 根目录 | 目录和文件 | | 描述 | 完成情况 | 功能实现情况 | 支持设备 |
| MDDS\_  MSP43  0G2 | SOCDriver | sys | 系统核心 | 已测试 | 基本功能 | MSP430 |
| rcc | 时钟与电源 | 已测试 | 基本功能 | MSP430 |
| gpio | 数字IO端口 | 已测试 | 基本功能 | MSP430 |
| USCI | 硬件串行通信 | 已测试 | 基本功能 | MSP430 |
| timer | 计时器 | 已测试 | 基本功能 | MSP430 |
| SOC\_error | 错误处理 | 已测试 | 基本功能 | MSP430 |
| DMA | 直接内存传输 | 已测试 | 基本功能 | MSP430 |
| RTC | 实时时钟 | 已测试 | 基本功能 | MSP430 |
| ADC | 模数转换 | 已测试 | 基本功能 | MSP430 |
| conf\_Driver.h |  | 配置文件 | 已测试 | 基本功能 | MSP430 |
| conf\_  middleware.h |  | 配置文件 | 已测试 | 基本功能 | MSP430 |
| conf\_Math.h |  | 配置文件 | 已测试 | 基本功能 | MSP430 |
| conf.h |  | 配置文件 | 已测试 | 基本功能 | MSP430 |
| Driver | conf\_Driver  \_template.h |  | 配置文件模板 | 未完成 | 基本功能 | MSP430、STM32 |
| Driver |  | 初始化设备 | 已测试 | 基本功能 | MSP430、STM32 |
| font0816 |  | 字体 | 已测试 | 基本功能 | MSP430、STM32 |
| SSPI |  | 软件SPI | 已测试 | 基本功能 | MSP430 |
| LX\_12864 |  | 液晶屏 | 已测试 | 基本功能 | MSP430、STM32 |
| DAC8411 |  | DAC8411 | 已测试 | 基本功能 | MSP430 |
| G2Platform |  | MSPG2实验平台定义 | 空内容 |  | MSP430 |
| HT1621 |  | HT1621 | 已测试 | 基本功能 | MSP430 |
| SegLCD\_G2 |  | MSPG2实验平台液晶定义 | 已测试 | 全部功能 | MSP430 |
| TCA6416 |  | TCA6416 | 除读取已测试 | 基本功能 | MSP430、STM32 |
| Middleware | conf  \_middleware  \_template.h |  | 配置文件模板 | 未完成 | 基本功能 | MSP430、STM32 |
| Middleware |  | 中间件初始化 | 已测试 | 基本功能 | MSP430、STM32 |
| RouteBuffer.h |  | 环形缓冲区 | 已测试 | 大部分功能 | MSP430 |
| Stream |  | 流 | 空内容 |  | MSP430 |
| AbstractButton |  | 标准按键输入 | 已测试 | 全部功能 | MSP430、STM32 |
| AbstractButton  Adapt.h |  | 标准键码转换 | 已测试 | 全部功能 | MSP430、STM32 |
| AbstractButton  Adapt\_template.c |  | 标准键码转换模板 | 已测试 | 全部功能 | MSP430、STM32 |
| GUI |  | 图形用户界面（带事件处理） | 部分完成 | 窗口 | STM32 |
| keycode.h |  | 标准键码 | 已测试 | 基本功能 | MSP430 |
| LinkedList.h |  | 双向链表 | 已测试 | 大部分功能 | MSP430 |
| StateMachine |  | 状态机 | 空内容 |  | MSP430、STM32 |
| Math | conf\_Math  \_template.h |  | 配置文件模板 | 未完成 | 基本功能 | MSP430、STM32 |
| base64 |  | base64 | 未测试 | 全部功能 | MSP430 |
| Time |  | 时间和时间戳 | 测试不通过 | 基本功能 | MSP430 |
| sha3 |  | sha3 | 未测试 | 全部功能 | STM32 |

注：一对.c/.h文件标注时合并且省略扩展名，省略includes文件。

## 使用方法

程序运行时，使用SystemInit();初始化所有外设并启动中断，初始化信息定义于conf.h、conf\_driver.h和conf\_Math.h 。

STM32上使用的初始化函数为AdapterInit();

注意：MSP430中SystemInit()函数不初始化Timer的附加模块，且不启动TimerA，需要手动执行这些操作

# MSP430G2结构

## CPU

MSP430单片机搭载16位极低功耗RISC CPU，活动功耗为250 µA/MIPS，具有JTAG调试接口。其结构如下：

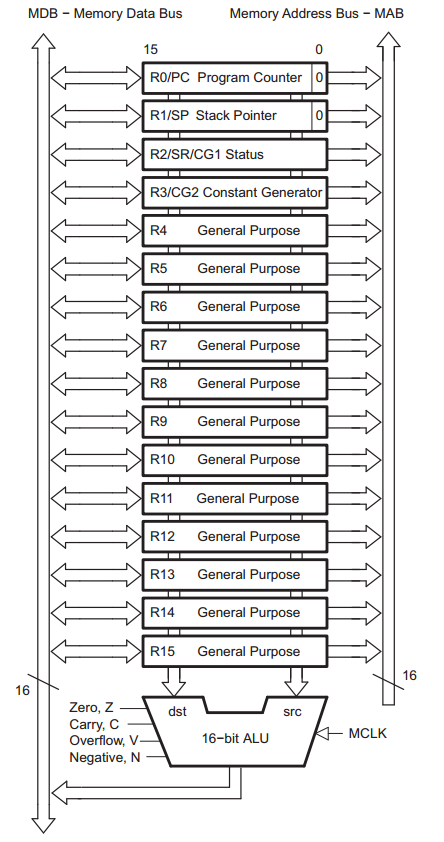


图 2‑1 CPU结构图

需要注意的是，MSP430单片机的中断向量在高位，从0xFFFE递减，这与其他单片机不同。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 中断源 | 中断标志 | 系统中断 | 地址 | 优先级 |
| 上电 | WDTIFG | 复位 | 0FFFEh | 15，最高 |
| 外部复位 |
| 看门狗 |
| NMI | NMIIFG | 非屏蔽 | 0FFFCh | 14 |
| 振荡器故障 | OFIFG\* | 可屏蔽 |
| I/O专用 | P0IFG.0 | 可屏蔽 | 0FFFAh | 13 |
| I/O专用 | P0IFG.1 | 可屏蔽 | 0FFF8h | 12 |
| 可屏蔽 | 0FFF6h | 11 |
| 看门狗定时器 | WDTIFG | 可屏蔽 | 0FFF4h | 10 |
| Timer\_A | CCIFG0 | 可屏蔽 | 0FFF2h | 9 |
| Timer\_A | TAIFG\*\* | 可屏蔽 | 0FFF0h | 8 |
| USART接收 | URXIFG | 可屏蔽 | 0FFEEh | 7 |
| USART发送 | UTXIFG | 可屏蔽 | 0FFECh | 6 |
| ADC，定时器/端口2 | ADCIFG | 可屏蔽 | 0FFEAh | 5 |
| 定时器/端口1 |  | 可屏蔽 | 0FFE8h | 4 |
| P2 | P2IFG.07\*、\*\* | 可屏蔽 | 0FFE6h | 3 |
| P1 | P1IFH.07\*、\*\* | 可屏蔽 | 0FFE4h | 2 |
| Basic Timer | BTIFG | 可屏蔽 | 0FFE2h | 1 |
| P0 | P0IFG.27\*、\*\* | 可屏蔽 | 0FFE0h | 0，最低 |

表 2.1中断向量表

## 时钟

MSP430运行需要的时钟有3个，分别是主时钟MCLK、辅助时钟ACLK、辅助主时钟SMCLK。各个时钟的作用不同。MCLK主要用于CPU时钟，也可提供给外设；ACLK时钟速度往往较低，一般用作外设时钟；SMCLK一般作为外设时钟。

三个时钟的时钟源都可以选择片内和片外，具体参照时钟树。

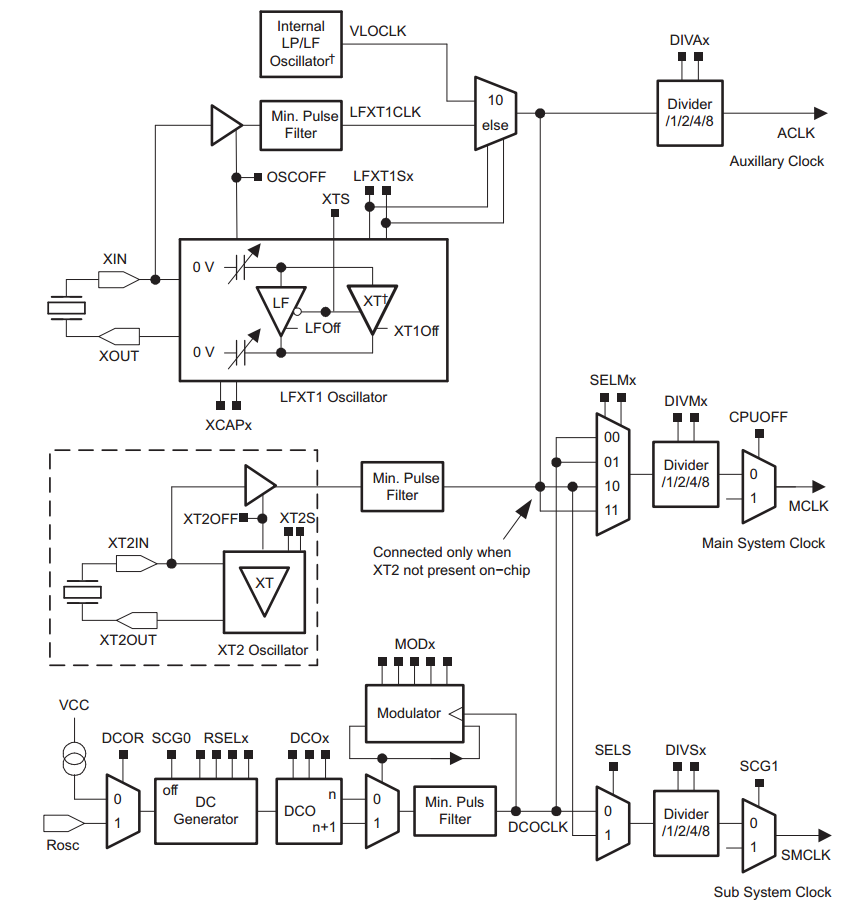


表 2.2 时钟树

值得注意的是LFXT1时钟（MSP430G2系列唯一的外部时钟），具有可配置的电容，一般可以免去外接电容。

## 数字IO

MSP430的数字IO的使用较51单片机更为复杂，具体体现为：

1. 使用前需要设置IO方向
2. 输入寄存器与输出寄存器分开
3. 使用引脚的其他功能需要调整功能选择寄存器（2个）
4. 没有开漏输出，需要调整IO方向进行模拟
5. 具有可配置的上/下拉电阻
6. 所有引脚可配置为中断源

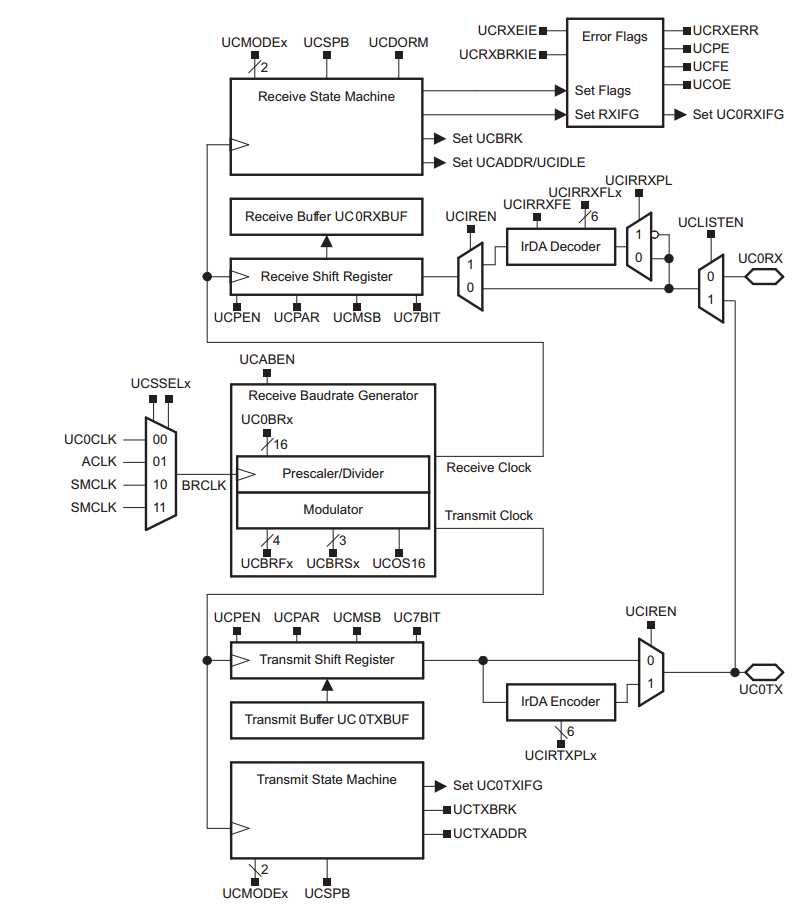
## 定时器

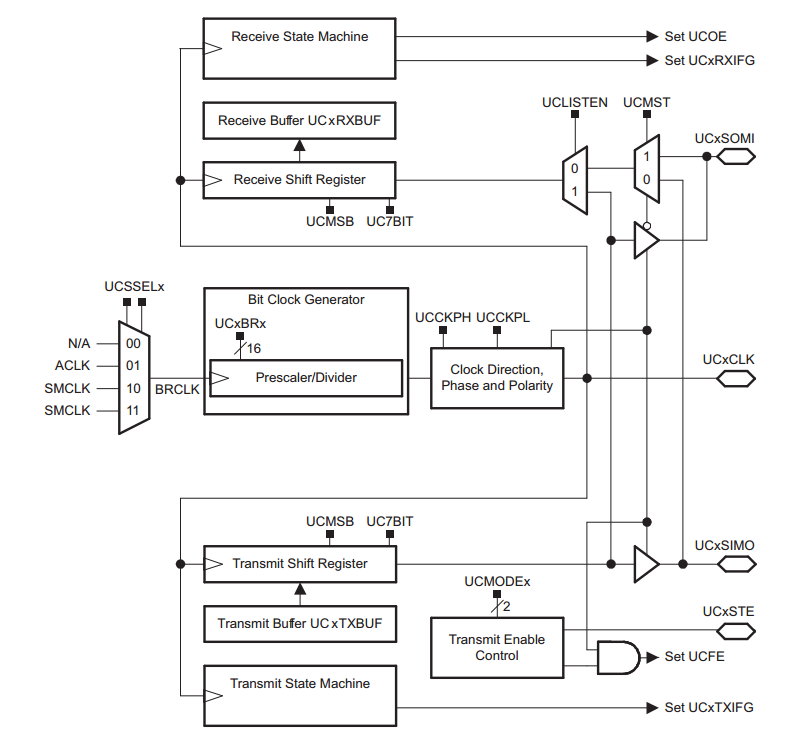
MSP430G2拥有两个TA3定时器，具有多个时钟源，可调整预分频和可变溢出值，且每个定时器带有3个通道（一个常常用作时基的增计数模式寄存器）。每个通道可配置输出和输入（比较模式和捕获模式）。

定时器A的工作可以参考STC15系列单片机的PCA模块，但其需要注意以下不同：

1. 0通道往往被占用，使用时应注意。
2. 通道输出时注意配置IO的功能。
3. 0通道独占一个中断通道，时基和剩下的通道共占一个中断通道

## USCI模块



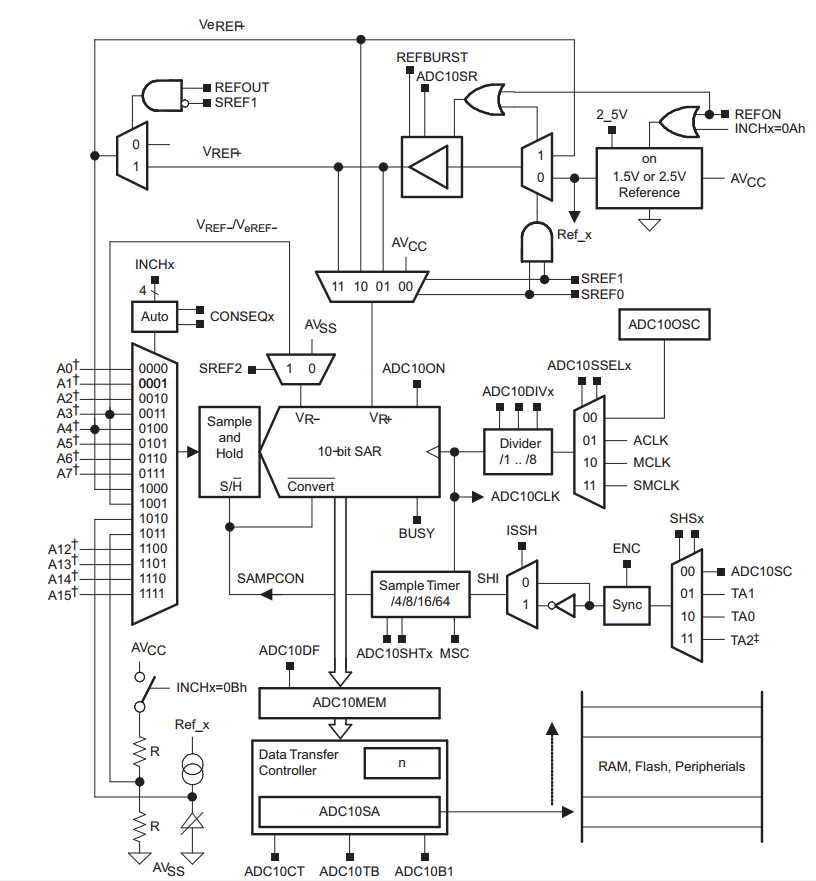


USCI模块可以看作是UART模块和SPI模块的集合，在通常的使用上基本没有难点，按照一般的模块编程即可。使用时应尤其注意：

1. 注意设置UCSYNC位以选择工作模式
2. 通道输出时注意配置IO的功能，尤其注意PxSEL2。

## ADC10模块

ADC10模块是MSP430中一个复杂的模块，与STC15和STM32等许多单片机有较大差别。使用是应注意参照结构图进行适当配置。



需要注意的有以下几点：

1. ADC10具有参考电压发生器，需要时可以使用，但注意其配置较为复杂，需要调整多个寄存器。
2. 配置多个通道时，注意改变转换模式。
3. 多通道采样时，应设置INCHx为最高那个通道，启动时自动减少至通道0
4. ADC10包含一个温度传感器，使用时需要把转换速率调整为一个较低的速度，并把参考电压设置为1.5V或2.5V
5. ADC10包含VCC/2和参考电压通道

# MSP430芯片驱动（SOCDriver）

## 初始化模块

初始化模块包含sys.c及sys.h，依赖于时钟模块。

包含函数：void SystemInit();

### SystemInit函数

功能：关闭看门狗，初始化时钟、GPIO、Timer0A的时基模块，启动中断。

注意：Timer0A需要用户进一步初始化和启动。时钟配置为：内部高频振荡器配置为1MHz。外部晶振fosc>1M时，MCLK初始化为外部晶振，否则初始化为内部高频振荡器;外部晶振fosc=32768Hz时，ACLK初始化为外部晶振，否则初始化为内部高频振荡器;SMCLK与MCLK配置相同。

### IdleProcess：空闲处理

处理基本定时器等事件。

注意：此函数不应在中断中调用

## 时钟与低功耗模式

文件：rcc.c和rcc.h

包含函数：void SysClockInit(enum e\_DCOFreq DCOSel,enum e\_LFXT1Freq LFXT1FreqSel,enum e\_LFXT1Cap LFXT1CapSel,enum e\_ClockSource MCLKSel,enum e\_CLKDiv MCLKDivSel, enum e\_ClockSource ACLKSel,enum e\_CLKDiv ACLKDivSel, enum e\_ClockSource SMCLKSel,enum e\_CLKDiv SMCLKDivSel);

void PowerPolicySet(enum e\_PowerPolicy PowerPolicy);

INT32U getClockFreq(enum e\_ClockType Clk);

### 配置

**#define** CRYSTAL1\_ENABLE 1//外部振荡器使能

**#define** CRYSTAL1\_FREQ 32768//外部振荡器频率

**#define** CRYSTAL1\_CAP C12P//外部振荡器电容

### 枚举类型

#### e\_ClockSource：时钟源选择

LFXT1CLK：外部晶体振荡器

XT2CLK：G2系列不可用

DTOCLK：内部高速振荡器

VLOCLK：内部12KHz振荡器

注意：部分选项不可用，并需要同时调整e\_LFXT1Freq，具体参照时钟树。

#### e\_DCOFreq：DCO频率选择

F16000K：16M已校准

F12000K：12M已校准

F8000K：8M已校准

F1000K：1M已校准

F100K：100k未校准，以下类推

F150K：

F210K:

F300K:

F410K:

F580K:

F750K:

F1200K:

F1600K:

F2300K:

F3400K:

F4250K:

F6000K:

F7800K:

F11250K:

F15250K:

#### e\_LFXT1Freq：外部晶体振荡器工作频率选择

F32768：32768Hz晶体振荡器

\_reserve：保留，未使用

FVLOCLK：内部振荡器12K

EXTCLK：外部时钟输入

F400K\_1M：晶体振荡器，400K~1M

F1M\_3M：晶体振荡器，1M~3M

F3M\_16M：晶体振荡器，3M~16M

F400K\_16M：晶体振荡器，400K~16M

#### e\_LFXT1Cap：外部振荡器可配置电容选择

C1P：1pF

C6P：6pF

C10P：10pF

C12P：12pF（评估板使用）

#### e\_CLKDiv：时钟分频

DIV1：不分频

DIV2：2分频

DIV4：4分频

DIV8：8分频

#### e\_PowerPolicy：电源策略

Off：关闭，所有部件停止，可用外部中断唤醒

Stop：停止，RTC模块运行，内存数据保持，端口保持，其余停止，可用外部中断唤醒

Idle：空闲，CPU停止

LowPower：低功耗（未实现）

Normal：正常（未实现）

HighPerformance：高性能（未实现）

#### e\_ClockType：时钟类型，获取时钟频率时用

MCLK,

ACLK,

SMCLK

### SysClockInit

初始化时钟

参数：

enum e\_DCOFreq DCOSel,//DCO频率选择

enum e\_LFXT1Freq LFXT1FreqSel,// 外部晶体振荡器工作频率选择

enum e\_LFXT1Cap LFXT1CapSel,// 外部振荡器可配置电容选择

enum e\_ClockSource MCLKSel,//主时钟时钟源选择

enum e\_CLKDiv MCLKDivSel,// 主时钟分频

enum e\_ClockSource ACLKSel,//辅助时钟时钟源选择

enum e\_CLKDiv ACLKDivSel,// 辅助时钟分频

enum e\_ClockSource SMCLKSel,//第二时钟时钟源选择

enum e\_CLKDiv SMCLKDivSel//第二时钟分频

返回：无

注意：如果外部晶振初始化失败，所有使用外部时钟的时钟将以VLOCLK为时钟。

### PowerPolicySet

设置电源策略

参数:

e\_PowerPolicy：电源策略

返回：无

### getClockFreq

获取时钟频率

参数：

enum e\_ClockType Clk：时钟类型

返回：

INT32U：时钟频率（HZ）

## 通用IO口

文件：gpio.c和gpio.h

GPIO模块包含初始化、读写、中断处理操作。

注意：GPIOA代表P1，GPIOB代表P2，使用引脚时使用PINx的引脚号代替数字。用户需要实现void GPIOISR(enum e\_GPIOGroup Group,INT8U Pin);函数以处理中断。

### 配置

#define Px\_x\_MODE PINMODE\_OUTPUT //Px.x的模式

#define Px\_OUTPUT 0xfe//初始化时Px口输出状态

### 枚举和宏

#### e\_GPIOPinMode：引脚模式

//basic 基本模式，二选一

PINMODE\_OUTPUT=1,//输出模式

PINMODE\_INPUT=0,//输入模式

//additional 额外的模式

PINMODE\_OUTPUT\_OD=8,//开漏输出（仅输出低电平）

PINMODE\_INPUT\_INT\_R=2,//上升沿中断

PINMODE\_INPUT\_INT\_F=6,//下降沿中断

PINMODE\_SELECT=0x10//第二功能

#### e\_GPIOGroup：GPIO组

GPIOA,//P1

GPIOB//P2

#### 引脚名

#define PIN0 1

#define PIN1 2

#define PIN2 4

#define PIN3 8

#define PIN4 0x10

#define PIN5 0x20

#define PIN6 0x40

#define PIN7 0x80

### GPIOInit：以conf.h初始化GPIO

参数：无

返回值：无

### GPIOWrite：写GPIO

参数：enum e\_GPIOGroup Group:GPIOA或GPIOB

INT8U data:数据（8位）

返回值：无

### GPIOWritePin: 写GPIO部分位

参数：enum e\_GPIOGroup Group:GPIOA或GPIOB

INT8U PinMask:PIN0~7及它们的组合

BL Statu:值(0/1)

返回值：无

### GPIORead: 读GPIO

参数：enum e\_GPIOGroup Group:GPIOA或GPIOB

返回值：INT8U数据

### GPIOReadPin: 读GPIO部分位

参数：enum e\_GPIOGroup Group:GPIOA或GPIOB

INT8U PinMask:PIN0~7及它们的组合

返回值：BL状态（0/1）

### GPIOSetMode: 设置GPIO模式

参数：enum e\_GPIOGroup Group:GPIOA或GPIOB

INT8U PinMask:PIN0~7及它们的组合

enum e\_GPIOPinMode Mode:模式

返回值：无

注意：开漏输出为软件实现，若初始化IO为开漏输出，只可使用写函数完成输出

### GPIOISR: 中断处理程序

由用户定义

参数：enum e\_GPIOGroup Group:GPIOA或GPIOB

INT8U Pin:0~7（注意：为数字）

返回值：无

## 定时器

定时器有0A和1A，包含中断处理和位宽拓展。

A3定时器有3个额外模块，其中0号一般用于时基，不建议使用。

用户可以实现TimerABaseInterruptHook();用于处理时基中断。

注意：因定时器宽度为16位，若捕获/比较需要超出16位的值，需要使用拓展模式。

### 配置

**#define** TIMER\_xA\_EN 0//使能

**#if** TIMER\_xA\_EN>0

**#define** TIMER\_xA\_SOURCE T\_SMCLK//源

**#define** TIMER\_xA\_SOURCE\_DIV DIV1//分配

**#define** TIMER\_xA\_BASE\_INTR\_EN 1//中断使能

**#define** TIMER\_xA\_BASE\_MODE UpMode//计数模式

**#define** TIMER\_xA\_BASE\_FREQ 1000//溢出频率（已经不用于初始化）

**#define** TIMER\_0A\_BASE\_MAX\_COUNT 12000//时基最大计数

**#define** TIMER\_0A\_BASE\_INTR\_CALLBACK 1//启动时基回调函数

**#endif**

### 枚举

#### e\_TimerBaseClockSource//时基时钟源

T\_TACLK,//外部时钟

T\_ACLK,//辅助时钟

T\_SMCLK,//第二时钟

T\_INCLK//2553中为TACLK，其他不知道，可能是取反的TBCLK

#### e\_TimerBaseMode//时基模式

StopMode=0,//定时器暂停

UpMode=1,//增计数 定时器增计数至等于比较寄存器 0 的值

ContinuousMode=2,//连续 定时器连续增计数

UpDownMode=3//增/减计数 定时器增计数到等于比较寄存器 0 的值又减计数到 0

#### e\_TimerAModuleCaptureMode //捕获模式

No,//捕获模式禁止

RisingEdge,//上升沿捕获

FallingEdge,//下降沿捕获

BothEdges//在上升沿与下降沿都捕获

#### e\_TimerAModuleInputSelect //输入选择

CCIxA,//

CCIxB,//

GND,//

VCC//

#### e\_TimerAModuleSynchronizeCaptureSource//异步/同步捕获

Asynchronous,//异步捕获

Synchronous//同步捕获

#### e\_TimerAModuleInputMode//捕获/比较模块工作模式

Compare,//比较模式

Capture//捕获模式

#### e\_TimerAModuleOutputMode// 输出模式

OutBitValue,//0：仅输出OUTx 位的数字即 Outx

Set,//1：置位 比较信号 EQUx 使 Outx 置位

ToggleReset,//2：PWM 翻转/复位 比较信号 EQUx 使 Outx 翻转，EQU0 使 Outx 复位

SetReset,//3：PWM 置位/复位 比较信号 EQUx 使 Outx 置位，EQU0 使 Outx 复位

Toggle,//4：翻转 比较信号 EQUx 使 Outx 翻转

Reset,//5：复位 比较信号 EQUx 使 Outx 复位

ToggleSet,//6：PWM 翻转/置位 比较信号 EQUx 使 Outx 翻转，EQU0 使 Outx 置位

ResetSet//7：PWM 复位/置位 比较信号 EQUx 使 Outx 复位，EQU0 使 Outx 置位

### TimerAInit：时基初始化

参数：

enum e\_TimerBaseClockSource SourceClkSel：时钟源设置

enum e\_CLKDiv ClkDiv：时钟输入分配

BL InterruptEnable：中断启动（如果使用位数拓展，则必须启用）

enum e\_TimerBaseMode TimerBaseMode：时基工作模式

返回：无

### TimerAModuleInit：额外模块初始化

参数：

INT8U Channel,：通道

enum e\_TimerAModuleInputMode InputModeSet,：捕获/比较模块工作模式

enum e\_TimerAModuleCaptureMode CaptureModeSet,：捕获模式

enum e\_TimerAModuleInputSelect InputSelectSet,：输入选择

enum e\_TimerAModuleSynchronizeCaptureSource SynchronizeCaptureSourceSet,：异步/同步捕获

enum e\_TimerAModuleOutputMode OutputModeSet,：输出模式设置

BL InterruptEnable：中断允许

返回：无

### TimerABaseFreqSet：时基频率设置

参数：

INT32U Freq：频率

INT32U ClkFreq：时钟频率（如果使用内部时钟，可以填0自动计算）

返回：无

### Timer0ABaseMaxCountSet：时基频率设置

参数：

INT16U count：计数次数

返回：无

### TimerAStart：定时器启动

参数：无

返回：无

### TimerAStop：定时器停止

参数：无

返回：无

### TimerAModulePWMSet ：PWM模式设置占空比

参数：

INT8U Channel,：通道

INT16U Value：值（0xffff对于100%）

返回：无

### TimerAModuleCaptureInit：捕获模式初始化

参数：

INT8U Channel,：通道

BL ExternModeEN：拓展模式，可以捕获32位值

void (\*pCallBackFunc)(INT32U time)：指向回调函数的指针

返回：无

注意：TimerAModuleInit仍然要调用

### TimerAModuleCompareInit：比较模式初始化

参数：

INT8U Channel,：通道

INT32U Time：时间

void (\*pCallBackFunc)(INT32U time)：指向回调函数的指针

返回：无

注意：TimerAModuleInit仍然要调用，目前比较模式不支持拓展模式，需要回调函数内完成进一步判断，同时需要重新初始化模块。

### TimerABaseInterruptHook：时基中断处理函数

参数：无

返回：无

注意：由用户定义

### Timer1ABaseFreqGet：获取时基溢出频率

返回：INT32U：频率（Hz）

## DMA：直接内存访问

注意：2553单片机不支持DMA功能

DMA模块提供两种模式：静态模式和动态模式，用以适应不同需求。

静态模式下，通道配置固定，不能动态调度，仅能启动。

动态模式下，通道不固定，提供插入方式和排队方式以不同的优先级启动传输。

### 配置

**#define** DMA\_EN 0 //DMA is supported only in MSP430x2xx device family

**#if** DMA\_EN>0

**#define** DMA\_LIST\_SIZE 5

**#define** DMA0\_STATIC 1//DMA功能固定，不能被动态调度

**#if** DMA0\_STATIC>0

**#define** DMA0\_REQUEST {SoftwareTrigger,BlockMode,RisingEdge,0,WordData,Increase,0,WordData,Increase,0}

**#endif**

**#define** DMA1\_STATIC 1//DMA功能固定，不能被动态调度

**#if** DMA1\_STATIC>0

**#define** DMA1\_REQUEST {SoftwareTrigger,BlockMode,RisingEdge,0,WordData,Increase,0,WordData,Increase,0}

**#endif**

**#define** DMA2\_STATIC 1//DMA功能固定，不能被动态调度

**#if** DMA2\_STATIC>0

**#define** DMA2\_REQUEST {SoftwareTrigger,BlockMode,RisingEdge,0,WordData,Increase,0,WordData,Increase,0}

**#endif**

**#endif**

### 枚举

#### e\_DMADynamicPolicy

DMANow,

DMALater

#### e\_DMAStartSignal

SoftwareTrigger ,// DMAREQ bit (software trigger)

TimerACCR2Tirgger ,// TACCR2 CCIFG bit

TimerBTirgger ,// TBCCR2 CCIFG bit

UCA0RxTirgger ,// Serial data received UCA0RXIFG

UCA0TxTirgger ,// Serial data transmit ready UCA0TXIFG

DAC12Tirgger ,// DAC12\_0CTL DAC12IFG bit

ADC12Tirgger ,// ADC12 ADC12IFGx bit

TimerACCR0Tirgger ,// TACCR0 CCIFG bit

TimerBCCR0Tirgger ,// TBCCR0 CCIFG bit

UCA1RxTirgger ,// Serial data received UCA1RXIFG

UCA1TxTirgger ,// Serial data transmit ready UCA1TXIFG

MultiplierTirgger ,// Multiplier ready

UCB0RxTirgger ,// Serial data received UCB0RXIFG

UCB0TxTirgger ,// Serial data transmit ready UCB0TXIFG

LastDMAChannelTirgger ,// DMAxIFG last channel:0:2,1:0,2:1

ExternalTirgger // External trigger DMAE0

#### e\_DMAMode

SingleMode ,// Single transfer Each transfer requires a trigger. DMAEN is automatically cleared when DMAxSZ transfers have been made.

BlockMode ,// Block transfer A complete block is transferred with one trigger. DMAEN is automatically cleared at the end of the block transfer.

HalfSpeedBlockMode ,// Burst-block transfer CPU activity is interleaved with a block transfer. DMAEN is automatically cleared at the end of the burst-block transfer.

HalfSpeedBlockMode1 ,// Burst-block transfer CPU activity is interleaved with a block transfer. DMAEN is automatically cleared at the end of the burst-block transfer.

RepeatedSingleMode ,// Repeated single transfer Each transfer requires a trigger. DMAEN remains enabled.

RepeatedBlockMode ,// Repeated block transfer A complete block is transferred with one trigger. DMAEN remains enabled.

RepeatedHalfSpeedBlockMode ,// Repeated burst-block transfer CPU activity is interleaved with a block transfer. DMAEN remains enabled.

RepeatedHalfSpeedBlockMode1 // Repeated burst-block transfer CPU activity is interleaved with a block transfer. DMAEN remains enabled.

#### e\_DMAAddrIncrement

Unchange,

Unchange1,

Increase,

Decrease

#### e\_DMADataMode

WordData,

ByteData

#### e\_DMATriggerMode

RisingEdge,

HighLever

### 结构

#### s\_DMARequest

**enum** e\_DMAStartSignal Signal;//触发信号

**enum** e\_DMAMode DMAMode;//DMA模式

**enum** e\_DMATriggerMode TriggerMode;//触发模式

**void** \*SrcAddr;//源地址

**enum** e\_DMADataMode SrcDataMode;//源数据模式

**enum** e\_DMAAddrIncrement SrcAddrInc;//源地址增长方向

**void** \*DscAddr;//目标地址

**enum** e\_DMADataMode DscDataMode;//目标数据模式

**enum** e\_DMAAddrIncrement DscAddrInc;//目标地址增长方向

INT16U Size;

### DMAInit：初始化DMA

使用conf。h进行初始化

### DMAStaticStart：静态通道启动

参数：

INT8U Channel：通道

返回：无

### DMADynamicStart：动态通道启动

参数：

s\_DMARequest Req：DMA请求

返回：INT8U：0-》成功，非0-》失败

## 基本定时器接口

基本定时器接口旨在提供一个通用接口供应用使用。

此接口调用片上计时器，依时长不同使用不同的策略。

时长短于定时器溢出时长时，使用比较模块进行延时，注意必须保证空闲模块充足；否则利用定时器溢出进行计数，计时时长舍入到溢出时间的倍数。

### 配置

**#define** BTIMER\_COUNT 3//数量

**#define** BTIMER\_INTERFACE Timer1A//定时器接口

### BasicTimerInit：初始化

参数：无

返回：无

### BasicTimerAdd：添加一个定时器

参数：

INT32U TimeUS：时间（μS）

BL SingleShot：单次

void (\*pCallBack)()：回调函数

返回：

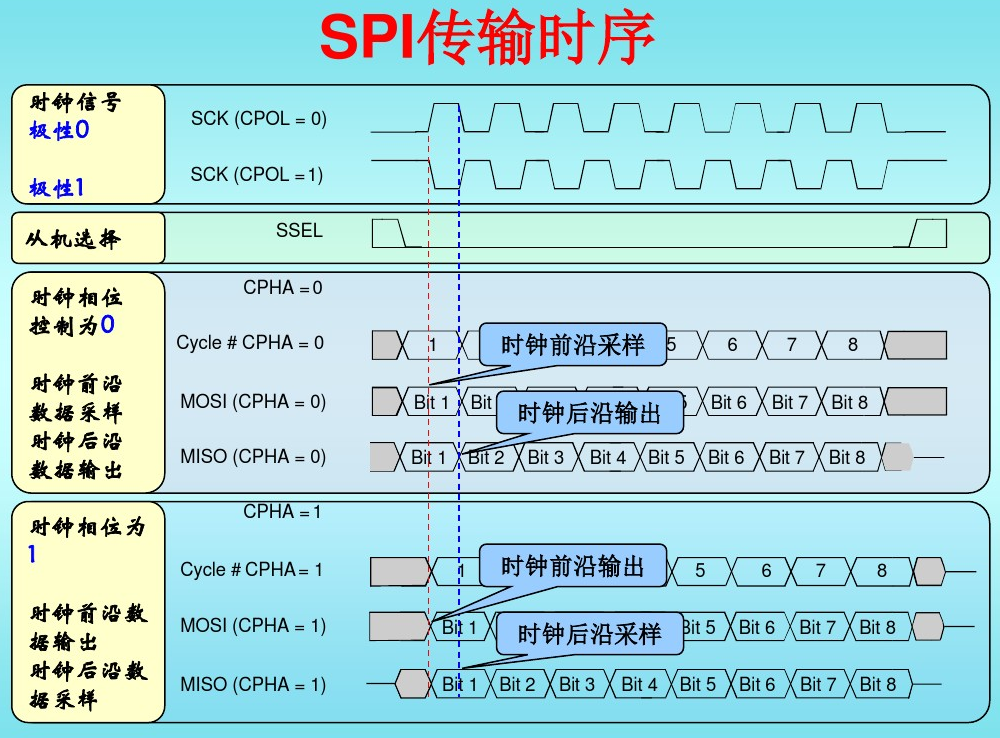
INT8S：定时器号，-1代表失败

### BasicTimerDel：删除一个定时器

参数：

INT8U index：编号

## USCI：硬件SPI、UART和IIC



HSPI使用USCIA0或USCIB0模块，目前支持主机模式的输出（3-wire SPI），支持发送中断。依据使用中断与否采用不同的逻辑。

使用中断：

void HSPISetCallBack(void (\*pFunc)());//设置回调函数

void HSPISent(INT8U dat);//发送数据并立即返回，完成自动后调用回调函数

不使用中断：

void HSPISent(INT8U dat);//发送数据并立即返回

void HSPIWaitTransmit();//等待发送完成

UART（仅USCIA0模块）目前配置为：校验禁止，8位数据，1位停止，UART模式，支持发送中断。依据使用中断与否采用不同的逻辑。

使用中断：

void UARTSetCallBack(void (\*pFunc)());//设置回调函数

void UARTSent(INT8U dat);//发送数据并立即返回，完成自动后调用回调函数

不使用中断：

void UARTSent(INT8U dat);//发送数据并立即返回

void UARTWaitTransmit();//等待发送完成

IIC（仅USCIB0模块）目前配置为：SMCLK、同步通信I2C主机状态，支持发送和接收中断。依据使用中断与否采用不同的逻辑。

使用发送中断（未支持）

不使用发送中断：

void IICStartWrite();//IIC写入开始

void IICWriteByte (INT8U dat);//发送数据并立即返回

BL IICWaitTransmit();//等待发送完成（判断返回值，若返回1说明从机无响应且停止波形以被发送，应停止通信，继续发送数据或停止信号的调用将被忽略）

void IICStop();//发送停止信号

使用接收中断（未支持）

不使用发送中断：

void IICStartRead();//IIC写入开始

BL IICWaitReceive ();//等待发送完成（判断返回值，若返回1说明从机发送了停止波形）

void IICReadByte (INT8U dat);//发送数据并立即返回

void IICNoAckModeSet();//发送NACK信号，注意调用应在IICWaitReceive之前

### 配置

**#define** HSPI\_EN 1 //使能HSPI

**#if** HSPI\_EN>0

**#define** HSPI\_INTERFACE 0 //选择使用模块：0->UCA0或1->UCB0

**#define** HSPI\_CLK\_SEL USCI\_SMCLK //时钟选择：USCI\_NA（无）或USCI\_ACLK（ACLK）或USCI\_SMCLK（SMCLK）

**#define** HSPI\_CLK\_DIV 2 //时钟分频：1~65535

**#define** HSPI\_MSB\_F 1 //MSP在前

**#define** HSPI\_CPOL 1 //CPOL

**#define** HSPI\_CPHA 1 //CPHA

**#define** HSPI\_TX\_INTERRUPT\_EN 1 //使能发送中断

**#define** HSPI\_RX\_INTERRUPT\_EN 0 //

**#endif**

**#define** UART\_EN 0

**#if** UART\_EN>0

**#define** UART\_INTERFACE 0 //选择使用模块：0->UCA0

**#define** UART\_CLK\_SEL USCI\_SMCLK //时钟选择：USCI\_NA（无）或USCI\_ACLK（ACLK）或USCI\_SMCLK（SMCLK）

**#define** UART\_CLK\_DIV 104 //时钟分频：1~65535//9600bps@1M->104

**#define** UART\_MSB\_F 1 //MSP在前

**#define** UART\_TX\_INTERRUPT\_EN 0 //使能发送中断//使用DMA时不要开启

**#define** UART\_RX\_INTERRUPT\_EN 0 //

**#endif**

### 枚举

#### enum e\_USCIClkSource：HSPI时钟源

*USCI\_NA*,//无

*USCI\_ACLK*,//ACLK

*USCI\_SMCLK*//SMCLK

### HSPIInit ：HSPI初始化

参数：

enum e\_HSPIClkSource ClkSourceSel：时钟源选择

INT16U Div：分频数

BL CPHASel：CPHA

BL CPOLSel：CPOL

BL TxInterruptEN：使能中断

返回：无

### HSPIWaitTransmit ：HSPI等待数据发送完成

注意：仅用于无中断模式

### HSPISetCallBack ：HSPI设置回调函数

参数：

void (\*pFunc)()：回调函数

返回：无

### HSPISent ：HSPI发送数据

参数：

INT8U dat：数据

返回：无

### UARTInit ：UART初始化

参数：

enum e\_HSPIClkSource ClkSourceSel：时钟源选择

INT16U Div：分频数

BL TxInterruptEN：使能中断

返回：无

注意：所有数据传输应放在初始化之后，包括其他USCIx0模块，因为初始化可能影响数据传输

### UARTWaitTransmit： UART等待数据发送完成

参数：无

返回：无

注意：仅用于无中断模式

### UARTSetTxCallBack： UART设置回调函数

参数：

void (\*pFunc)()：回调函数

返回：无

### UARTWriteByte: UART发送数据

参数：

INT8U dat：数据

返回：无

### IICInit ：IIC初始化

### IICWaitTransmit IIC等待数据发送完成

返回：BL：失败指示

注意：仅用于无中断模式

### IICWaitReceive IIC等待接收一个数据（仅用于无中断模式）

返回：BL：失败指示

注意：仅用于无中断模式

### IICSetTxCallBack IIC设置回调函数

参数：

void (\*pFunc)()：回调函数

返回：无

### IICSetRxCallBack IIC设置回调函数

参数：

void (\*pFunc)()：回调函数

返回：无

### IICStartWrite IIC写入开始

IICStartRead IIC读取开始

### IICStop IIC停止

### IICNoAckModeSet IICNoAck模式

注意：在等待接收之前调用

### IICWriteByte IIC写入字节

参数：INT8U dat：数据

### IICReadByte IIC读取一个字节

返回：INT8U dat：数据

## RTC：实时时钟

### 配置

**#define** RTC\_EN 1//使能RTC

**#if** RTC\_EN>0

**#define** RTC\_MODE RTC\_Internal//RTC\_Internal/RTC\_External(Don't use)

**#define** RTC\_INTR\_TIME\_IN\_MS 1000//RTC计时中断时间/ms

**#define** RTC\_INTERNAL\_TIMER RTC\_1S//基本定时器接口（弃用）

#endif

### RTCInit：初始化

参数：无

返回：无

### RTCTimeSet：设置时间

参数：

UTime time：时间戳

### RTCTimeGet：获取时间

返回：UTime：时间戳

### RTCInterrupt：RTC计时中断函数

参数：无

返回：无

# 驱动

## SSPI：软件SPI

SSPI在设计上用于驱动串行液晶屏，使用SPI模式3，MSB First ，CPOL=1 ，CPHA=1。

### 配置

**#define** SSPI\_CK GPIOA,PIN4 //SCK引脚，为GPIOx,PINx的二元组

**#define** SSPI\_MOSI GPIOA,PIN2 //MOSI引脚，为GPIOx,PINx的二元组

### SSPIInit：初始化SSPI（无作用）

### SSPISent ：SSPI发送数据

参数：

INT8U dat：数据

返回：无

SPI模式3，MSB First

CPOL=1

CPHA=1

### SSPIWaitTransmit：SSPI等待数据发送完成（无作用）

## Font：字体

字体适用于点阵液晶屏，取模方式为纵向，倒序。以下为0816字形的定义

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0.0 | 1.0 | 2.0 | ··· | 7.0 |
| 0.1 | 1.1 | 2.1 | ··· | 7.1 |
| 0.2 | 1.2 | 2.2 | ··· | 7.2 |
| 0.3 | 1.3 | 2.3 | ··· | 7.3 |
| ·  ·  · | ·  ·  · | ·  ·  · |  | ·  ·  · |
| 0.7 | 1.7 | 2.7 | ··· | 7.7 |
| 8.0 | 9.0 | A.0 | ··· | F.0 |
| 8.1 | 9.1 | A.1 | ··· | F.1 |
| 8.2 | 9.2 | A.2 | ··· | F.2 |
| 8.3 | 9.3 | A.3 | ··· | F.3 |
| ·  ·  · | ·  ·  · | ·  ·  · |  | ·  ·  · |
| 8.7 | 9.7 | A.4 |  | F.7 |

注：标注采用字节.位的方式

### ASCII，0816

字体存放与font0816.c/.h中，首个字符为“！”

包含字体信息三元组，为FONT\_ASCII0816

## LX\_12864

### 配置

**#define** LCD\_DRIVER HSPI //LCD接口驱动，取值有HSPI和SSPI

**#define** LCD\_INTR\_EN 0 //LCD数据传输由中断控制

**#define** LCD\_BUF\_SIZE 40 //LCD缓冲区大小，仅在中断模式时有效

**#define** LCD\_RST\_PIN GPIOA,PIN0 //RST引脚号，为GPIOx,PINx的二元组

**#define** LCD\_CS\_PIN GPIOA,PIN6 //CS引脚号，为GPIOx,PINx的二元组

**#define** LCD\_A0\_PIN GPIOA,PIN7 //A0引脚号，为GPIOx,PINx的二元组

### LCDInit ：LCD初始化

参数：无

返回：无

### LCDClear：清LCD

参数：

INT8U FillData：填充数据

返回：无

### LCDWriteImageRaw ：LCD写入图像

参数：

INT8U Page：页

INT8U Column：列

INT8U width：宽度

INT16U size：数据大小

const INT8U \*dat：数据的指针

返回：无

注意：会完全覆盖原来内容

### LCDWriteCharRaw ：LCD写入字符

参数：

INT8U Page：页

INT8U Column：列

INT8U fontwidth：字体宽度

INT16U fontsize：字体字符数据大小

const INT8U \*fontdat：字体数据的指针

char dat：字节

返回：无

注意：会完全覆盖原来内容，字体信息部分可以直接用字体提供的信息三元组代替

### LCDWriteStringRaw ：LCD写入字符串

参数：

INT8U Page：页

INT8U Column：列

INT8U fontwidth：字体宽度

INT16U fontsize：字体字符数据大小

const INT8U \*fontdat：字体数据的指针

char \*dat：字符串

返回：无

注意：会完全覆盖原来内容，字体信息部分可以直接用字体提供的信息三元组代替

## DAC8411

DAC8411的驱动，不支持硬件SPI。

### 配置

**#define** DAC8411\_EN 0

**#if** DAC8411\_EN>0

**#define** SYNC\_HIGH P1OUT |=BIT0//SYNC信号写1

**#define** SYNC\_LOW P1OUT &=~BIT0//SYNC信号写0

**#define** SCLK\_HIGH P2OUT |=BIT2//SCLK信号写1

**#define** SCLK\_LOW P2OUT &=~BIT2//SCLK信号写0

**#define** DIN\_HIGH P1OUT |=BIT3//DIN信号写1

**#define** DIN\_LOW P1OUT &=~BIT3//DIN信号写1

**#endif**

### DAC8411Init：初始化

参数：无

返回：无

### DAC8411Write写数据

参数：INT16U Data ：数据

返回：无

### DAC8411AutoWriteInit：自动写入初始化

参数：const INT16U \*dat：数据地址

size\_t len：长度

返回：无

### DAC8411AutoWrite：自动写入执行函数

进行实际的DAC写入，常常作为定时器的回调函数

参数：无

返回：无

## HT1621液晶驱动器

### 配置

**#define** HT1621\_EN 1//使能

**#if** HT1621\_EN>0

**#define** HT1621\_INTERFACE EXIO//驱动接口（GPIO或EXIO）

**#define** HT1621\_CS\_PIN GPIOB,PIN4//片选端

**#define** HT1621\_RD\_PIN GPIOB,PIN5//not necessary,always 1，若未定义，则不使用

**#define** HT1621\_WR\_PIN GPIOB,PIN6//写入使能

**#define** HT1621\_DATA\_PIN GPIOB,PIN7//数据引脚

**#endif**

### HT1621Init LCD初始化

参数：无

返回：无

### HT1621Refresh屏幕刷新

参数：无

返回：无

### HT1621SegmentListAppend改变点亮的段

参数：const INT8U \*List：指向数据的指针，数据最高位指示是否点亮，可用宏SEG\_SET、SEG\_RESET

INT8U Lendth：长度

返回：无

## SegLCD\_G2液晶

### LCDDigitDisplay：显示数字

名称：LCDDigitDisplay(unsigned char Position,unsigned char Digit )

功能：让128段式液晶的特定“8字”段显示0~9

说明：大数码“8字”编号左至右为0~5，小数码“8字”编号右至左为6~9；

注意：需要刷新屏幕

传入参数：Digit：想显示的数 0~9，传入其他数字则为消隐

Position：显示的数位（第几个“8字”）

举例：LCD\_DisplayDigit(1,9 )，第1个“8字”段显示9

LCD\_DisplayDigit(1,LCD\_DIGIT\_CLEAR)，第1个 “8字”段消隐

## TCA6416串行IO扩展

### 配置

**#define** TCA6416A\_EN 1//使能

**#if** TCA6416A\_EN>0

**#define** TCA6416A\_INTERFACE IIC //IIC only

**#define** TCA6416A\_PORT\_0\_DIR 0//GPIOA（P0口）模式：0=输出，1=输入，

**#define** TCA6416A\_PORT\_1\_DIR 0x0f//GPIOB（P1口）模式：0=输出，1=输入

**#define** TCA6416A\_PORT\_0\_OUT 0xaa//GPIOA（P0口）数据

**#define** TCA6416A\_PORT\_1\_OUT 0xff//GPIOB（P1口）数据

**#endif**

### TCA6416A\_Init：初始化

参数：无

返回：无

### EXIOWrite写IO

参数：Group:GPIOA或GPIOB

data:数据（8位）

### EXIOWritePin：写IO部分位

参数：Group:GPIOA或GPIOB

PinMask:PIN0~7及它们的组合

Statu:值(0/1)

### EXIORead：读IO

参数：Group:GPIOA或GPIOB

返回值：数据

### EXIOReadPin：读IO部分位

参数：Group:GPIOA或GPIOB

PinMask:PIN0~7及它们的组合

返回值：状态（0/1）

# 中间件

## RouteBuffer.h：环形缓冲

建立环形缓冲区及其配套函数

调用方法：MAKE\_BUFFER(缓冲区名，缓冲区数据类型，缓冲区指针类型，读阻塞操作，写阻塞操作)

作用：建立名为缓冲区名的数组，以及名为pBuf缓冲区名W,pBuf缓冲区名R,Buf缓冲区名Len共3个指针，以及配套函数:Buf缓冲区名W,Buf缓冲区名R,Buf缓冲区名RB,Buf缓冲区名WB共4个

### Buf缓冲区名W：向缓冲区写入

参数：

const 缓冲区数据类型\* Buf：写入数据

缓冲区指针类型 Len：长度

返回：无

注意：若写入数据超出缓冲区，数据后部会被自动截断

### Buf缓冲区名R：读取缓冲区

参数：

const 缓冲区数据类型\* Buf：数据指针

缓冲区指针类型 Len：长度

返回：INT8U：错误代码（0：正常，1：数据量不足，无法读）

### Buf缓冲区名RB：读取缓冲区（阻塞式）

参数：无

返回：缓冲区数据类型：数据

注意：若阻塞，则不断调用读阻塞操作，读阻塞操作为去掉最后分号的语句

### Buf缓冲区名RB：写缓冲区（阻塞式）

参数：缓冲区数据类型：数据

返回：无

注意：若阻塞，则不断调用写阻塞操作，读阻塞操作为去掉最后分号的语句

### BufInit初始化环形缓冲区

用法：BufInit(缓冲区名);

## Stream：流

流用于配置带有自动传输和控制的线性缓冲区，以配合DMA工作。

因MSP430G2553不支持DMA，此部分搁置

## AbstractButton：标准按键

标准按键模块的功能全面，具有按键消抖功能和缓冲功能。

输入层为数组AButtonInput，用于映射各种接口的按键。模块提供勾函数AButtonRefreshHook()以进行按键前处理。

输出层具有两种输出方式，模式取决于回调函数是否为空。若无回调函数，使用AButton AButtonGet()；函数读取按键；否则回调函数会自动调用。

### 配置

**#define** ABUTTON\_EN 1//使能

**#if** ABUTTON\_EN>0

**#define** ABUTTON\_COUNT 4//按键数量，必须<=255

**#define** ABUTTON\_GET\_FUNC\_BUFFER\_LEN 10//按键缓冲区大小，必须<256

**#define** ABUFFON\_REFRESH\_HOOK\_EN 1//使能勾函数进行按键前处理

**#define** ABUTTON\_RAWINPUT\_WORDLENGTH 8//按键输入数组的类型：无符号整型长度。取值有8、16、32、64

**#endif**

### 

## GUI：图形用户界面

## LinkedList：链表

# 数学函数

## Base64

### 配置

**#define** BASE64\_EN 0

### Base64Encode：数据转Base64字符串。

参数：

const unsigned char \*src:源

size\_t len：长度

char \*dsc：目标

返回：size\_t 目标长度

注意：不可用于带MMU系统，产生的base64自带'\000'，目标内存计算方法：（向上对齐到4（len\*4/3））+1

### Base64Decode：数据转Base64字符串。

参数：

const char \*src:源

size\_t len：长度

unsigned char \*dsc：目标

返回：size\_t 目标长度

注意：产生的数据无'\000'，目标内存计算方法：（len\*3/4）-最后‘=’个数

## Time：时间戳和时间工具

随用随加。

现在使用时间戳单位为ms，以2000/01/01 00：00：00.000为0，时间戳类型为64为有符号数

注意：64位数除法似乎在MSP430上没有正常工作

### ConventTimestampToTime：转换时间戳为时间

参数：

UTime Timestamp：时间戳

返回：Time：时间

注意：只支持正数时间戳，且时间戳过大时运行时间可能非常长

UTime ConventTimeToTimestamp(Time t);

### ConventTimestampToUnixTimestamp：转换时间戳为UNIX时间戳

UTime Timestamp：时间戳

返回：INT32S：UNIX时间戳

注意：注意数据溢出

# 标准库设计总结

在几年前，我就已经考虑创建一个具有标准接口的库，可以极其方便地应用在各种架构的单片机上。然后，我了解了Arduino，一个已经移植到多种单片机的开源平台，但是其缺点也是十分明显的：

1. Arduino基于C++，然而像8051之类的单片机没有C++编译器，即使使用C++ to C转换程序，但是C++程序的大内存消耗是8051之类只有128Byte RAM的单片机无法承受的。
2. Arduino提供了非常高的抽象，然而高抽象是以性能和内存为代价的，这一点和C++的高资源消耗类似。

学习STM32之后，我了解了CMSIS，ARM Cortex™ 微控制器软件接口标准，是 Cortex-M 处理器系列的与供应商无关的硬件抽象层（a vendor-independent hardware abstraction layer for the Cortex-M processor series and defines generic tool interfaces）。使用CMSIS，可以为处理器和外设实现一致且简单的软件接口，从而简化软件的重用。然而，CMSIS只适用与ARM Cortex Microcontroller，不适合于其他单片机。

我希望这样一个库可以发挥出各种单片机的大部分性能，同时保证可移植性，并希望其包含诸如数学函数、常见数据结构如链表和二叉树、一些程序结构例如状态机和操作系统内核，以便于快速开发。

经过数次尝试，参考一些跨设备平台例如PlatformIO，我确定了以下的开发原则：

1. 整个库使用C作为开发语言，必要时可以使用汇编以加速执行，但需要提供兼容的C代码（上下文切换等C语言无法完成的任务除外）。
2. 程序由 main() 函数开始执行，使用SystemInit()初始化所有设备，一般情况不需要其他初始化函数。
3. 初始化所需的所有信息由宏定义，包含在一个或数个头文件中
4. 变量定义在源文件中，弃用μCOS-II使用的头文件定义变量方式，以方便调试。
5. 所有源文件只需包含includes.h，一般不需包含其他头文件。
6. 函数设计一般仅包含我可能使用的功能。
7. 库提供少数高度抽象的功能和大部分抽象较少的功能以最大化利用性能。

在这些设计原则指导下，我使用MSP430G2553开发了标准库的第一个版本，具体实现细节请参照《MDDS库使用指南》。其设计上我较为满意，然而其功能并不完整，需要大量时间去添加新功能。

现在库的开发取得了阶段性成果，然而以后的工作还会很多。C语言在宏上的功能仍不能使我满意，例如缺少宏上的循环和宏的拼接，我以后可能会使用新的语言替换掉C语言。