

MSP-EXP430Launchpad 实验指南

前言

MSP430G2 系列是德州仪器近期推出的一款产品，在秉承 MSP430 超低功耗，高集成度的优点的同时，具有高性价比的特点。该系列被称为 ValueLine，旨在以 8 位单片机的价格实现 16 位单片机的性能。MSP-EXP430Launchpad 是 TI 推出的又一套用于 MSP430 和电路实验的开发板。在该套不到名片大小的开发板上集成了一片超低功耗 16 位 MSP430 单片机，USB 口仿真器电路以及各引脚接口等。利用 LaunchPad 开发板，仅需一台笔记本电脑，就可以在基于图形界面的编程软件 CCS 上进行嵌入式软硬件系统的开发和调试，真正做到将实验室装进口袋里，让使用者可以随时随地，不受场地和设备的限制进行 430 的开发工作。除了学生自主创新实践外，LaunchPad 开发板还可以用于本科低年级课程，如嵌入式 C 语言，电子技术基础，微机原理，单片机等课程的自主实验环节以及课程设计。该套开发板为单片机爱好者提供了一个很好的学习平台。

该实验指导书在 Launchpad 的基础上进行了功能模块的扩展，以期更好地进行实验教学和学习。本书共有六章，分为两大部分。第一部分为第 1 章至第 3 章，对 MSP430G2 系列单片机的外设进行介绍，CCSv5.1 的安装和使用，同时给出了几个基于 Launchpad 的实际开发案例。第二部分为第 4 章至第 6 章，对一体化实验系统以及各扩展模块的硬件电路进行了详细的介绍，在此基础上通过六个基础实验以及六个综合实验帮助读者更好地理解 and 掌握 430 的开发和应用。

由于时间和篇幅的原因，本书中第三章关于微控制器外设寄存器的更为详细的描述没有在此书中涉及，但包含在随书光盘的电子文档中，供有需要的读者阅读和参考。

该实验指导书、程序和相关教学材料由西安电子科技大学 MSP430 联合实验室赵建老师和 TI 单片机大学计划黄争经理共同策划和审阅，在编写过程中，联合实验室老师和同学付出了辛勤的劳动，在此表示衷心的感谢。此外，也感谢 TI 大学计划部崔萌和王沁工程师对全书进行的修改，整理和完善工作。由于时间和水平有限，书中可能存在错误和不妥之处，敬请广大读者批评指正。

西安电子科技大学 MSP430 实验室

2012 年 10 月

目录

MSP-EXP430Launchpad 实验指南	1
前言	1
第一章 MSP430G2 系列单片机	6
第一节 MSP430 系列单片机概述	6
1.1.1 MSP430 系列单片机及低功耗特性	6
1.1.2 MSP430 单片机的其他特点	7
第二节 LaunchPad Launch!	9
第三节 MSP430G2 系列单片机的应用与开发	11
1.3.1 多路电源开关	11
1.3.2 风速测试仪	11
1.3.3 窗帘电机控制器	12
1.3.4 数字频率计	12
1.3.5 自行车里程表	13
第二章 CCS 快速上手	14
第一节 CCS 简介	14
功能总览	14
第二节 CCS 的使用	16
2.2.1 CCS 的安装	16
第三节 CCS 的调试	20
2.3.1 利用 CCSv5.1 导入已有工程	20
2.3.2 利用 CCSv5.1 新建工程	21
第三章 MSP430G2 系列单片机硬件资源应用技术	42
第一节 时钟与休眠模式	42
3.1.1 时钟系统简介	42
3.1.2 时钟系统的操作	43
3.1.3 基本时钟模块特性	43
3.1.4 VLO 时钟	43
3.1.5 LFX1 时钟	43
3.1.6 XT2 晶振	44
3.1.7 DCO	44
3.1.8 调整 DCO 的频率	45
3.1.9 运用外部电阻 R_{osc} 调整 DCO 的值	45
3.1.10 DCO 调制器	45
3.1.11 基本时钟系统的错误检测	46
3.1.12 从外部晶振获取 MCLK	46
3.1.13 基本寄存器表	46
第二节 通用 IO 口	50
3.2.1 IO 口	50

3.2.2 IO 寄存器	50
3.2.3 IO 口中断	51
3.2.4 线与逻辑.....	51
3.2.5 兼容性.....	52
第三节 10 位 ADC	53
3.2.1 ADC10 的特点	53
3.2.2 ADC10 转换模式.....	55
3.3.3 ADC10 寄存器	58
3.3.4 内部温度传感器.....	63
第四节 16 位定时/计数器.....	65
3.4.1 Timer_A 定时/计数器的主计数器模块结构和原理	65
3.4.2 Timer_A 定时/计数器比较模块	66
3.4.3 Timer_A 捕获模块	69
3.4.4 Timer_A 定时器中断	69
第五节 FLASH 控制器	73
3.5.1 Flash 存储介绍	73
3.5.2 Flash 存储器的分段	73
3.5.3 Flash 的操作	74
3.5.4 Flash 寄存器	84
第六节 通信接口 (USCI 和 USART)	91
3.6.1 串行同步和串行异步通信原理的简述	91
3.6.2 USCI 模块的相关寄存器定义.....	93
3.6.3 USART 的相关寄存器定义	100
3.6.4 USCI 模块初始化和收发操作步骤流程	105
3.6.5 USART 模块的初始化和收发操作步骤流程	121
第七节 比较器 A+.....	128
3.7.1 比较器 A+的介绍.....	128
3.7.2 比较器 A+的操作.....	128
3.7.3 比较器 A+寄存器.....	132
第八节 Grace 软件技术	136
3.8.1 创建 Grace 工程	136
3.8.2 使用 Grace 配置 I/O 口及外设	137
第九节 MSP430G2 系列单片机调试接口 JTAG 和 SBW	144
3.9.1 JTAG 简介	144
3.9.2 JTAG 接口	144
3.9.3 SBW 接口	145
第十节 触摸按键.....	146
3.9.1 电阻式按键.....	146
3.9.2 电容式按键.....	146
第四章 MSP430G2 系列单片机一体化实验系统.....	149
4.1.1 实验系统标准版结构组成.....	149
4.1.2 实验系统标准版使用对象和使用特点	153
第二节 MSP430G2 系列单片机一体化实验系统简化板	154
第三节 实验模块配置.....	156

4.3.1 模拟滤波器实验功能模块.....	156
4.3.2 程控放大器实验功能模块.....	161
4.3.3 晶体管特性测试模块.....	167
4.3.4 光照度检测实验功能模块.....	172
4.3.5 三种温度测量模块.....	179
4.3.6 矩阵键盘及数码管实验模块.....	185
4.3.7 触控 RGBLED 模块.....	188
4.3.8 频率计及 D/A 转换模块.....	194
4.3.9 三种通信接口模块.....	199
4.3.10 声音强度检测模块.....	206
第五章 MSP430G2 系列单片机基础实验.....	212
第一节 I/O 基础实验.....	212
5.1.1 矩阵键盘按键扫描实验.....	212
5.1.2 控制数码管显示数字实验.....	214
5.1.3 按键扫描并控制数码管显示键值实验.....	217
5.1.4 点阵 LCD 显示器控制实验.....	220
5.1.5 触摸按键实验.....	221
5.1.6 RGBLED 触控实验.....	225
第二节 AD 转换基础实验.....	227
5.2.1 输入电压检测实验.....	227
5.2.2 光照度检测实验.....	228
5.2.3 Pt100 温度测量实验.....	231
5.2.4 声音强度检测实验.....	234
第三节 D/A 转换基础实验.....	237
5.3.1 D/A 转换器件实现电压输出实验.....	237
5.3.2 PWM 实现电压输出实验.....	238
第四节 定时/计数器基础实验.....	240
5.4.1 定时信号产生实验.....	240
5.4.2 信号频率测量实验.....	241
5.4.3 模拟滤波器实验.....	245
第五节 通信接口基础实验.....	252
5.5.1 SPI 接口基础实验.....	252
5.5.2 程控放大器实验.....	256
5.5.3 RS-232 接口通信实验 (UART 接口).....	266
5.5.4 RS-485 接口通信实验 (UART 接口).....	269
5.5.5 : 18B20 温度测量 (单总线).....	271
第六章 MSP430G2 系列单片机应用实践.....	275
第一节 用三种温度传感器实现的温度巡检/控制器.....	275
6.1.1 目的与要求.....	275
6.1.2 电路设计和系统连接.....	275
6.1.3 信号与数据处理.....	275
6.1.4 软件设计与调试.....	277
6.1.5 总结与扩展.....	280
第二节 键盘输入控制的程控放大器.....	281

6.2.1 目的与要求.....	281
6.2.2 电路设计与系统连接.....	281
6.2.3 软件设计与调试.....	282
6.2.4 总结与扩展.....	289
第三节 声音强度检测仪.....	290
6.3.1 声音强度采样方法.....	290
6.3.2 声音强度数据处理.....	290
6.3.3 光照度检测模块程序设计.....	291
第四节 RS-232 接口与 PC 机通信.....	294
6.4.1 目的与要求.....	294
6.4.2 电路设计与系统连接.....	294
6.4.3 软件设计与调试.....	294
第五节 RS-485 接口多机通信.....	304
6.5.1 目的与要求.....	304
6.5.2 电路设计与系统连接.....	304
6.5.3 软件设计与调试.....	305
第六节 PS2 接口键盘输入与 LCD 显示.....	309
6.6.1 目的与要求.....	309
6.6.2 电路设计与系统连接.....	309
6.6.3 信号与数据处理.....	309
6.6.4 软件设计与调试.....	314
6.6.5 总结与扩展.....	335

第一章 MSP430G2 系列单片机

第一节 MSP430 系列单片机概述

本章主要内容包括 MSP430G2 系列单片机的特点性能介绍, launchpad 评估板资源的使用方法介绍及单片机入门的一些基础知识, 其中单片机入门知识有关软件的部分希望能够引起读者们的注意, 因为目前就单片机开发而言, 硬件方案 IC 制造商总是能够提出多样、成套的解决方案, 而工程师在整合方案之余, 绝大部分精力都投入到了编程之中, 故而在学习单片机之初养成良好的编程习惯, 不失为是工程师之路的一个良好地开端。

1.1.1 MSP430 系列单片机及低功耗特性

如今的单片机已经不再是二十年前所谓的 SCM, 而是从 MCU 向 SOC 转型中间期的一个时代, 在开始本章节正是内容之前, 不妨先解释一下上述提到的几个英文缩写:

SCM: 单片微型计算机 (Single Chip Microcomputer), 主要是寻求最佳的单片形态嵌入式系统的最佳体系结构。Intel 公司的 8031 就是这个缩写的代名词。

MCU: 微控制器 (Micro Controller Unit), 主要的技术发展方向是: 不断扩展满足嵌入式应用时, 对象系统要求的各种外围电路与接口电路, 突显其对象的智能化控制能力。它所涉及的领域都与对象系统相关, Philips 公司是最早提出并将这种概念付诸实践的。

SOC: 嵌入式系统 (System on Chip) 即寻求应用系统在芯片上的最大化解解决, 尤其是对于便携式、轻量小型化的手持设备, 对单片集成系统的需求更是强烈。每当工程师们讨论到手持设备, MSP430 系列单片机总是第一个被提上议程。

MSP430 (Mixed Signal Processor 混合信号处理器) 系列最为夺目的亮点之一就是超低功耗运行, 这一硬件上的特点甚至带来另一种以中断为主的编程模式 (第三节中详细讨论)。关于 MSP430 系列单片机到底有多省电, 可以做如下一个实验:

实验器材: 橘子一个、锌电极三支 (Zn, 可以把废旧的锌锰干电池外皮剪下压平)、铜电极三支 (Cu, 可以用铜制钥匙充当)、导线若干, 内带秒表程序 MSP430+段码液晶小系统

实验步骤:

1. 制作原电池。将橘子切成三块, 分别在果肉两侧插入锌铜两个电极, 锌为负端, 铜为正端。再用导线将三个原电池串联, 固定牢靠后用万用表测量电池电压。

实测数据每个水果电池电压约为 0.8V, 三个串联达 2.4V, 虽然距离 MSP430 单片机标准供电电压 3.3V 还有很大差距, 但是在这些条件下 MSP430 单片机运行是绰绰有余的。

2. 将 MSP430 单片机系统的 VCC、GND 分别和水果电池组的正负极连接, 观察段码液晶是否可以受到单片机的正常控制。

水果原电池能提供的电流十分有限, 短路实测最大电流不足 100uA, 而 MSP430 单片机运行秒表程序功耗仅仅不足 2uA, 段码液晶约 3uA, 系统总功耗 5uA。对于这个橘子而言, 给 MSP430 单片机供电实在是屈才了。

3. 将该实验装置放在通风良好, 温度 20 摄氏度的环境下, 每天三次记录单片机系统的运行状态, 检测系统能够维持的时间。

经实验, 只要水果没有腐烂变质或者风干 (可以适当补充水分), 该系统能坚持一个月以上。当液晶上的显示变得模糊不清时, 立刻换上新的电源, 系统立刻重获新生, 这时品尝橘子, 几乎没什么酸味了。5uA 到底是个什么概念呢, 举一个更直接的例子, 常用的 CR2032 纽扣电池, 可以为这个系统提供 5 年以上的续航能力。

如此种卓越的超低功耗特性, 自然给 MSP430 单片机开拓了广阔的应用空间:

第一, 正如前文所说, MSP430 单片机就是为了便携式设备而量身打造的。随着技术的

日新月异，便携式设备不断的向小型化、轻量化、高精度、多功能化的方向发展，在集成度、电池尺寸、设备大小的限制下，还对处理器运算能力和片上资源有了更高的要求。MSP430 单片机强大 CPU（16 位处理器，每秒处理指令最高 25 兆）和片上丰富的接口电路和模拟电路资源，可以实现模拟数字信号混合处理，大部分设计“单芯片”完成，大幅提高集成度和生产效率的同时有效的控制了成本。

第二，超低功耗的特定使得产品电池寿命终身化这个命题得以实现。按照电子产品预期使用寿命 5~8 年计算，电池设计寿命 8~10 年，如果产品耗电量足够低的话，一款产品终生只用配上一块电池，无需充电和更换。这样一方面减少了用户的麻烦，另一发明设计和制造的成本也降了下来。例如一款基于 MSP430 单片机的野外气象观测的传感器节点，可以连续工作数年不更换电池，直至产品寿命终结。

第三，MSP430 单片机低功耗还体现在另一方面，即十分微弱的能量，也能够驱动 MSP430 单片机工作，在 1.8V 以上的电压下 CPU 都可以正常工作，最新系列的 MSP430 单片机甚至可以把这个数值再降低几乎一半至 1.1V。这样很多间接电能也能直接给 MSP430 单片机供电，如小块太阳能电池、信号线自取点、温差能、电缆上磁场能、人体机械运动震动的能量、酸碱碱性溶液中的能量等等。基于这种间接电能设计的无源设备产品，也是符合当前低碳环保设计理念的新方向。日本最近研制了一种尿液检查卡片，就是以尿液作为电解质发电，驱动低功耗系统进行尿液样本分析。

TI 新品发布会上，Scott Roller 用“无人能及，实实在在”八个字来形容 430 系列产品，并将“独特的超低漏电工艺技术”和 MSP430 单片机超低功耗系统理念、MSP430 Ware 软件列为 MSP430 单片机系列”实现超低功耗的三大独门秘籍。

对于单片机开发人员而言，芯片内部工艺或许与设计不是那么相关，但是 MSP430 单片机低功耗系统在软件方面的操作就至关重要了，因为即使 CPU 能够以极低的功耗执行指令，而程序总是做一些冗余的动作，那样进行下来，还是会有很多功耗白白浪费，那么 MSP430 为软件上控制低功耗又提供了那些可能呢？

首先，MSP430 单片机建立的时钟系统概念，使得 CPU、片上模块、休眠唤醒三者时钟彼此独立。总众所周知，时钟频率的高低是决定系统功耗的一个重要因素，但是不同模块的运行速度各不相同，CPU 的繁忙程度也并非总是很高，这就造成低速时钟无法满足需求，高速时钟又会带来功率浪费的尴尬局面。MSP430 单片机时钟系统提供了三种时钟，通过软件寄存器设置，不同时钟可以分别开启关闭，可以分别设置倍频分频系数，为各种模块和 CPU 提供多样的选择。基于这样的时钟系统，MSP430 单片机可以实现不同深度的系统休眠，如此梯度化的休眠方式，让整个系统以间歇工作的方式，最大限度的节约能量。

其次，MSP430 单片机采用模块化设计，在使用时每个模块都可以由软件单独开启关闭，用到某一模块时打开它，任务完毕之后关闭它，这样也能节省不少的电能。这样的设计还有诸多其他好处，每一种模块都具有独立完整的结构，在不同型号单片机中，同款模块的功能结构使用方法都是完全一样的。同一家族不同型号的 MSP430 单片机，实际上就是不同功能模块的组合。这样的设计，对于学习者而言，使得 MSP430 单片机学习一通百通，对研发者而言，更换更高级的 MSP430 单片机芯片时，程序移植得心应手。

此外，对于新手而言，如何在保证程序稳定性、健壮性的前提下提高程序执行效率、删除冗余语句，还是一个遥不可及要求。TI 公司就专门组织了老练的程序工程师编写了程序库 MSP430 ware，MSP430 430ware 中囊括了几乎所有模块的所有功能函数和海量的例程，把工程师从底层繁琐的代码编写工作中解放了出来，同时还保证了底层函数的效率和稳定。

1.1.2 MSP430 单片机的其他特点

除了超低功耗这一突出优势外，MSP430 系列单片机还有不少其他不俗的表现：

1. MSP430 单片机内核采用 16 位 RISC (reduced instruction set computer, 精简指令集计算机) 处理器, 单指令周期, 运算能力和速度优势明显, 某些型号的 MSP430 单片机内部带有硬件乘法器, 在 DMA 控制器的配合下, 性能堪比 DSP (Digital Signal Processing, 数字信号处理, 一种专精于复杂信号运算的智能器件)。

2. MSP430 单片机采用冯 诺依曼结构, 寄存器和数据段 (即 RAM) 与代码段 (即 ROM, FLASH 或 FRAM) 统一编制。这样代码在 RAM 里同样可以运行, 每款 MSP430 单片机都有 FLASH (或 FRAM) 控制器, 通过它可以对 ROM 的区的代码进行擦写。这种机制可以很方便的实现设备在线升级功能, 无需重新烧写程序, 固件更新时单片机仍处正常工作状态(在第三章中详细讨论)。

3. MSP430 单片机属于工业级芯片, 能够在-40~85 摄氏度的范围内工作, 并且带有 PWM 发生器等控制输出。适合各类工业测量、工业控制、电机控制等领域。

4. MSP430 单片机作为混合信号处理器, 模拟设备也是一大特色, 运算放大器、比较器、ADC、DAC 应有尽有。以 ADC 为例, 有高速的 ADC10、ADC12; 有高精度的 SD16、SD24; 还有低成本的 SLOPE, 可以满足各式各样的测试任务。

5. MSP430 单片机具有多种通信接口, 涵盖 UART、I2C、SPI、USI 等等, 5 系列单片机还带有 USB 控制器、射频控制器、Zigbee 控制器。适用于各种协议下的数据中继器、转发器、转换器的应用中。

6. TI 公司实力雄厚, MSP430 单片机系列产品生命力旺盛, 自 1996 年问世以来每年都有新的型号推出, 更新、更强、更省电的单片机不断的推出。2012 年六又推出了新款的“金刚狼”系列, 再度挑战低功耗极限。

第二节 LaunchPad Launch!

前言所述,主要谈到MSP430系列单片机在超低功耗领域傲人的优势,具体谈到MSP430G2系列单片机,其核心竞争力在于超高性价比。G2系列又称“超值系列”,那么它的超值到底体现在哪些方面呢?

首先,G2系列单片机的售价低,G2全系列单片机共有44款,根据片上资源的丰富程度价格由低到高,最低价格0.34美元,配置最全的2553也仅售0.99美元,这样的价位在单片机范围内可以算是物美价廉了。

G2系列单片机虽然价格不高,但不等于它的功能不强,G2系列单片机作为MSP430系列中的一员拥有大部分MSP430单片机片上外围模块:

- ◆ 16MHz 主频 16 位 CPU
- ◆ 片上程序存储器 FLASH (512B/1KB/2KB/4KB/8KB/16KB) 及 FLASH 控制器
- ◆ 片上随机存储器 SRAM(128B/256B/512B)
- ◆ 通用并行输入输出端口 GPIO (4 位/16 位/20 位/24 位)
- ◆ 支持电容触摸式 I/O
- ◆ 看门狗定时器 WDT
- ◆ 上电复位模块 BOR
- ◆ 多功能通信模块 USI (I2C/SPI) USIC_A(UART/LIN/IrDA/SPI) USIC_B(I2C&SPI)
- ◆ 比较器模块 Comparator A+
- ◆ 片内温度传感器 Temp Sensor
- ◆ 十位逐次逼近型 ADC10
- ◆ 斜率型 slope ADC

板载着一颗MSP430G2单片机的launchpad评估实验开发板除了上述片上资源外,板上还有诸多额外的硬件资源:

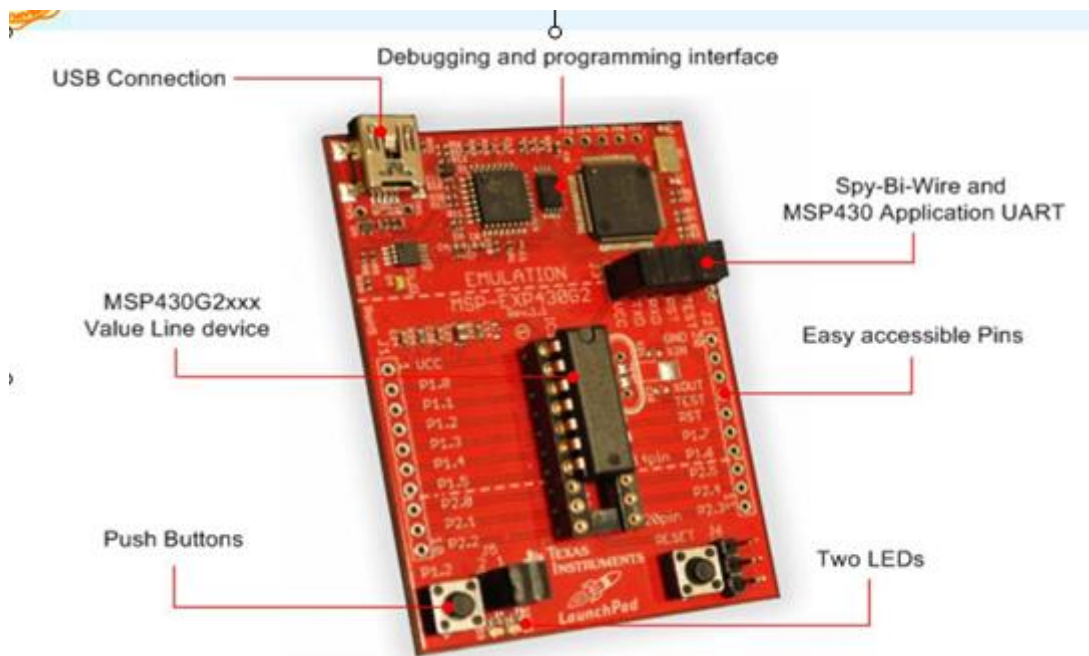


图 1.2.1 MSP430launchpad 实物图

- ◆ 板载 USB 调试与编程仿真器接口, 无驱动可安装
- ◆ 支持所有采用 DIP14 和 DIP20 封装 MSP430G2XX 和 MSP430F20 器件

- ◆ 红绿两粒 LED，两个按键
- ◆ 配套两款电容触摸板
- ◆ 所有管脚在板子两边引出

这其中最吸引人的当然要数板载 USB 仿真器了，从事 MSP430 单片机的朋友无一抱怨 MSP430 单片机的仿真器价格过高，一只 TI 官方认证的仿真器动辄上千元，而仿制的仿真器也要三百至四百元，顶着侵权的罪名同时还有时刻仿真器死机的风险。现在有了 launchpad，USB 仿真器即插即用，一根 USB 线，等待 30 秒，系统搞定驱动。这无疑是 MSP430 单片机开发者的福音，扫清了开发者道路上的羁绊。

这么好用的一块开发板得多少钱呢？现在 TI 正在 MSP430G2 系列单片机推广期，一块 launchpad 开发板仅售 4.30 美元，折合人民币不到 30 元，此外 TI 和国内各大高校和各大电子技术论坛都有合作，还有不少机会可以免费获得 launchpad。

第三节 MSP430G2 系列单片机的应用与开发

MSP430G2 系列单片机在各类学生科技实践中已经有了一定的影响，现简单展示几个 MSP430G2 系列单片机在科技活动中的应用成果，这些实例会在本后后面几个章节展开介绍。

1.3.1 多路电源开关

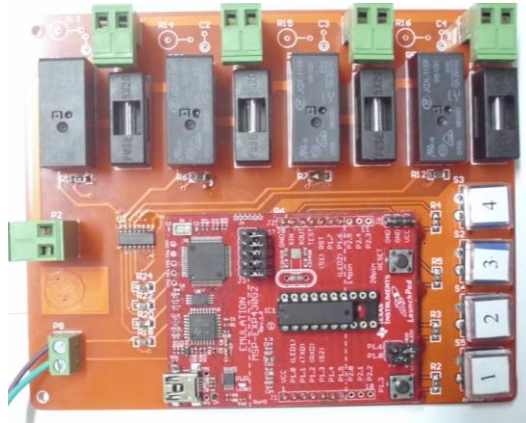


图 1.3.1 继电器控制器实物图

该作品特点：

- ◆ MSP430G2231 为主控芯片
- ◆ 四路继电器相互隔离
- ◆ 四个按键控制继电器开闭
- ◆ 继电器开闭情况由 LED 显示

继电器是小电流控制大电流的常见器件，作为小电流控者，单片机是一个常见的角色，各式各样的大功率用电器往往扮演被控者。这样的一种应用本身十分简单，无非就是起到一个开关功能，但是提供了一个基本的模块，拓展后有广泛的空间，如应用在家居环境，单片机可以控制冰箱、风扇、洗衣机的电源，用作工业环境单片机可以控制风机、电炉、传送带，用作农业可以控制进料通道、畜栏开闭。

1.3.2 风速测试仪



图 1.3.2 风速测试仪实物图

该作品特点：

- ◆ MSP430G2211 为主控芯片
- ◆ 驱动一个电源风扇
- ◆ 调理转速信号为 PWM 波
- ◆ 以脉冲计数方式测量风扇转速
- ◆ 风扇转速显示在 LCD 上

风冷系统是较水冷、气冷、氮冷几种主流冷却系统中性价比最高的冷却方式，是现在大多数电子设备散热的首选，该作品根据风扇自身的转速反馈信号测算出风扇的实际转速，若能再结合温度检测模块，则可构成一个完整的散热系统。

1.3.3 窗帘电机控制器

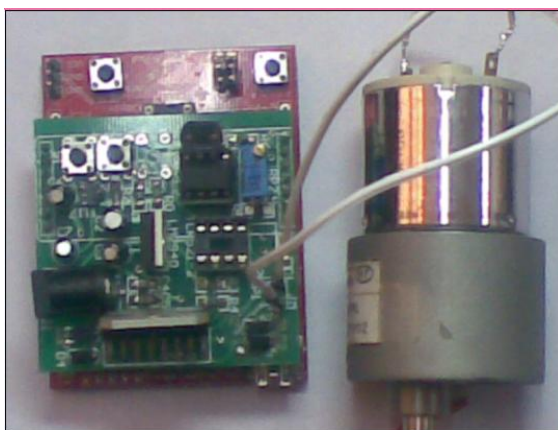


图 1.3.3 窗帘电机控制器

该作品特点：

- ◆ MSP430G2211 位主控芯片
- ◆ 单片机驱动直流电机
- ◆ 手动按键控制和自动感光控制两种方式
- ◆ 自动感光模式下光线强时自动关闭窗帘

智能家居概念提出至今，各式各样的家居智能产品在技术上已没有什么难题，核心障碍是如何做到人性化，如何想用户之所想，代用户之所劳，而家用窗帘控制器就解决的很多想拉窗帘而懒于行动的人烦恼。

1.3.4 数字频率计



图 1.3.4 频率计实物图

该作品特点：

- ◆ MSP430G2231 位主控芯片
- ◆ 定时计数法测频率
- ◆ 兼容正弦波、三角波、锯齿波、矩形波
- ◆ 频率测量结果显示在 LCD 上

频率是电学领域常见参数，频率测量是测控领域的一个重要课题，除了本作品这种定时测量方法外，还有很多复杂精密的方案，低频的频率测量只需要普通 I/O 口就行了，频率稍高的则要用到定时器或者其他分频方法，目前频域测量的尖端已经触及太赫兹领域，在普通人看来是一个不可能的数量级但是测控界确实已经做到了。

1.3.5 自行车里程表

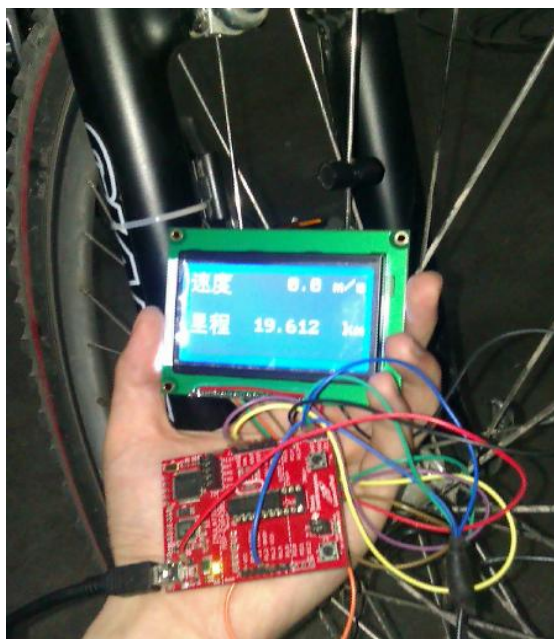


图 1.3.5 自行车里程表

该作品特点：

- ◆ MSP430G2231 位主控芯片
- ◆ 定时计数法测计算车轮转速
- ◆ LCD 显示当前速度、里程
- ◆ 手动复位里程计数
- ◆ 系统掉电数据不丢失

户外运动一直也是这些年来人们业余活动的一个热点，但是在人们尽情享受自行车越野带来的欢乐时，安全却成了常常被户外运动新手忽视的一个关键，而对于缺乏对外界地形、气候等户外经验的人来说，带上一个电子助手是一个不错的选择，当然这个助手除了能解决户外问题，还得低功耗，但是这对于 MSP430 单片机而言并非什么难事，实际上就在 2011 年 TI 杯全国 MCU&C2000 大赛中，西安电子科技大学的一个参赛队就以一部用 MSP430 单片机为核心，具有谷歌地图定位，记录海拔、时间、温湿度测量显示的“户外助手”作品，获得了专业组特等奖。

第二章 CCS 快速上手

第一节 CCS 简介

Code Composer Studio 是一种针对 TI 的 DSP、微控制器和应用处理器的集成开发环境。CCStudio 包括一套用于开发和调试嵌入式应用程序的工具。它包括用于各种 TI 设备系列的编译器、源代码编辑器、项目生成环境、调试程序、探查器、模拟器和其他许多功能。CCStudio 提供一个单一用户界面，指导用户完成应用程序开发流程的每一步骤。类似的工具和界面使用户能够比以前更快地开始使用，并且能够向他们的应用程序添加功能，这些都归功于成熟的生产能力工具。

本章节着重讲解 CCS 的使用的基本方法和功能，旨在与读者读完后能够快速上手，开始自己的 MSP430 单片机学习、练习之路，当然这里提到的使用方法也同样适用与 DSP 等其他 CCS 支持的处理器。

功能总览

1. 调试

CCStudio 的集成调试程序具有用于简化开发的众多功能和高级断点。条件断点或硬件断点以全 C 表达式、本地变量或寄存器为基础。高级内存窗口允许您检查内存的每一级别，以便您可以调试复杂的缓存一致性问题。CCStudio 支持复杂的多处理器或多核系统的开发。全局断点和同步操作提供了对多个处理器和多核的控制。

2. 分析

CCStudio 的交互式探查器使快速测量代码性能并确保在调试和开发过程中目标资源的高效使用变得更容易。探查器使开发人员能够轻松分析其应用程序中指令周期内或其他事件内的所有 C/C++ 函数，例如缓存未命中/命中率、管道隔栏和分支。分析范围可用于在优化期间将精力集中在代码的高使用率方面，帮助开发人员开发出经过优化的代码。分析可用于任何组合的汇编、C++ 或 C 代码范围。为了提高生产能力，所有分析设备在整个开发周期中都可使用。

3. 脚本

某些任务，例如测试，需要运行数小时或数天而不需要用户交互。要完成此类任务，IDE 应能自动执行一些常见任务。CCStudio 拥有完整的脚本环境，允许自动进行重复性任务，例如测试和性能基准测试。一个单独的脚本控制台允许您在 IDE 内键入命令或执行脚本。

图像分析和虚拟化

CCStudio 拥有许多图像分析及图形虚拟化功能。其中包括以图形方式在能够自动刷新的屏幕上查看变量和数据的能力。CCStudio 还能以本机格式 (YUV、RGB) 查看主机 PC 或在目标电路板中加载的图像和视频数据。

4. 编译器

TI 已经开发了专门为了最大程度地提高处理器的使用率和性能而优化的 C/ C++ 编译器。TI 编译器使用各种各样经典的、面向应用的、成熟的、因设备而异的优化，专为所有支持的结构而优化。其中部分优化包括：

消除公共子表达式；软件流水；强度折减；自动增量寻址；基于成本的寄存器分配；指令预测；硬件循环；函数内联；矢量化

TI 编译器还执行程序级别优化，在应用程序级别评估代码性能。通过程序级别视图，编译器能够像具有完整系统视图的汇编程序开发人员一样生成代码。编译器充分利用此应用程序级别视图，找出能够显著提升处理器性能的折衷。

TI ARM 和 Microcontroller C/C++ 编译器经过专门针对代码大小和控制代码效率的优

化。它们具备行业领先的性能和兼容性。

5. 模拟

模拟器向用户提供一种在能够使用开发板之前开始开发的方式。模拟器还具有更加透彻地了解应用程序性能和行为的优势。提供了几种模拟器，让用户能够权衡周期精确性、速度和外围设备模拟，一些模拟器特别适合算法基准测试，而另一些特别适合更加详细的系统模拟。

6. 仿真

TI 设备包含高级硬件调试功能。这些功能包括：

IEEE 1149.1 (JTAG) 和边界扫描；

对寄存器和内存的非侵入式访问；

实时模式，用于调试与不得禁用的中断进行交互的代码。实时模式允许您在中断事件挂起后台代码，同时继续执行时间关键中断服务例程；

多核操作，例如同步运行、步进和终止。其中包括跨核触发，该功能可以让一个核触发另一个核终止；

高级事件触发 (AET)，可在选定设备上使用，允许用户依据复杂事件或序列，例如无效数据或程序内存访问，终止 CPU 或触发其他事件。它能够以非侵入式方式测量性能及统计系统事件数量（例如缓存事件）；

CCStudio 提供有关选定设备的处理器跟踪，帮助客户发现以前“看不到的”复杂实时缺陷。跟踪能够探测很难发现的缺陷-事件之间的争用情况、间歇式实时干扰、堆栈溢出崩溃、失控代码和不停用处理器的误中断。跟踪是一种完全非侵入式调试方法，依赖处理器内的调试单元，因此不会干扰或更改应用程序的实时行为。跟踪可以微调复杂开关密集型多通道应用程序的代码性能和缓存优化。处理器跟踪支持程序、数据、计时和所选处理器与系统事件/中断的导出。可以将处理器跟踪导出到 XDS560 跟踪外部 JTAG 仿真器或选定设备上，或导出到芯片缓存嵌入式跟踪缓存 (ETB) 上。

7. 实时操作系统支持

CCS 具有两个版本的 TI 实时操作系统：

DSP/BIOS5.4x 是一种为 DSP 设备提供预清空多任务服务的实时操作系统。其服务包括 ISR 调度、软件中断、信号灯、消息、设备 I/O、内存管理和电源管理。此外，DSP/BIOS5.x 还包括调试诊断和加工，包括低系统开销打印和统计数据收集。

BIOS6.x 是一种高级可扩展实时操作系统，支持 ARM926、ARM Cortex M3、C674x、C64x+、C672x 和基于 28x 的设备。它提供 DSP/BIOS 5.x 没有的若干内核和调试增强，包括更快、更灵活的内存管理、事件和优先级继承互斥体。

注意：BIOS6.x 包括 DSP/BIOS5.x 兼容层，从而使应用程序源代码的迁移非常轻松。

第二节 CCS 的使用

2.2.1 CCS 的安装

(1) 运行下载的安装程序 ccs_setup_5.1.1.00031.exe，当运行到如图 2.2.1 处时，选择 Custom 选项，进入手动选择安装通道。

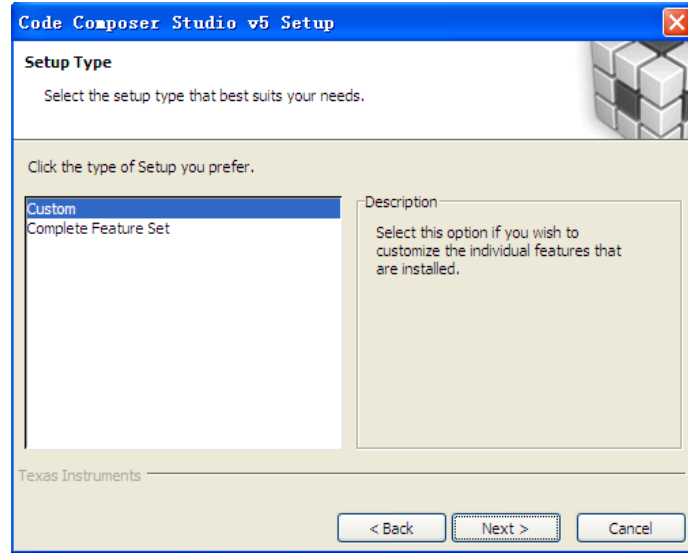


图 2.2.1 安装过程 1

(2) 单击 Next 得到如图 2.2.2 所示的窗口，为了安装快捷，在此只选择支持 MSP430 Low Power MCUs 的选项。单击 Next，保持默认配置，继续安装。（如果需要作为 C28x 32-bit Real-time MCUs 或者其他处理器的集成开发环境，则勾选相应的选项。）

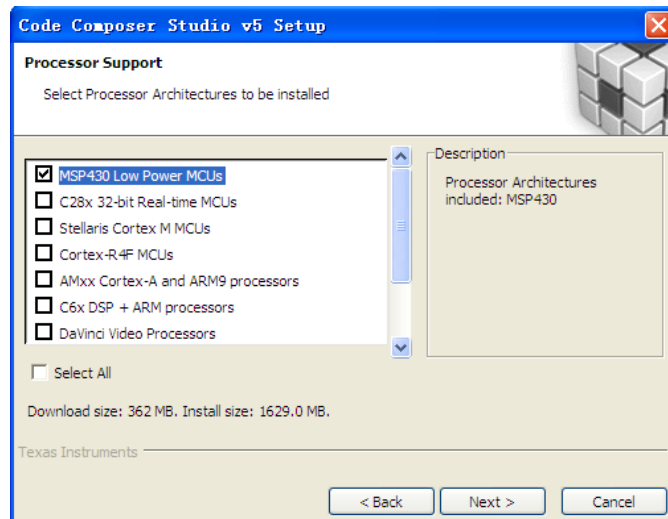


图 2.2.2 安装过程 2

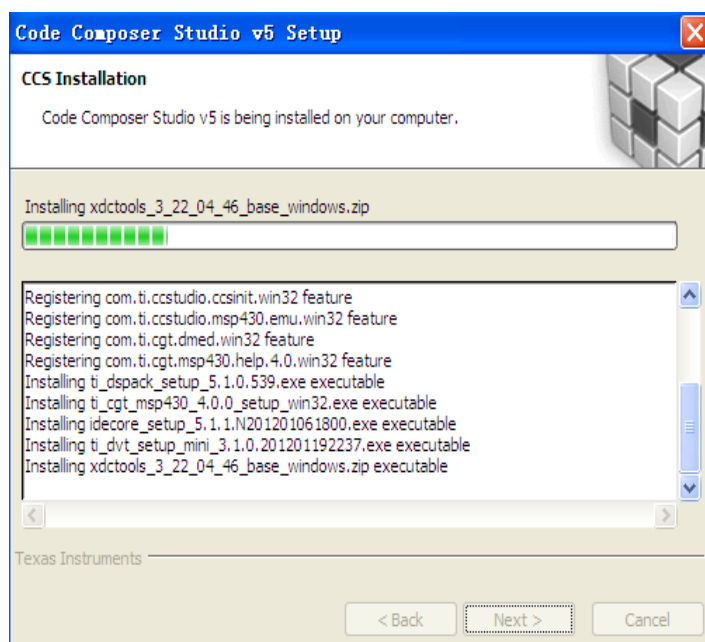


图 2.2.3 软件安装中

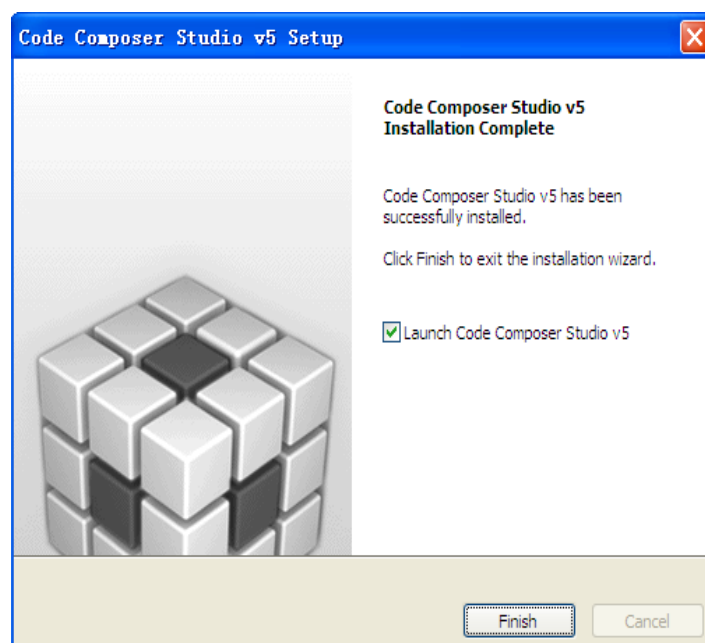


图 2.2.4 软件安装完成

(3) 单击 **Finish**，将运行 CCS，弹出如图 2.2.5 所示窗口，打开“我的电脑”，在某一磁盘下，创建以下文件夹路径：**C:\MSP-EXP430G2-Launchpad**，单击 **Browse**，将工作区间链接到所建文件夹。CCS 首先要求的是定义一个工作区，即用于保存开发过程中用到的所有元素（项目和指向项目的链接，可能还有源代码）的目录。默认情况下，会在 C:\Users\<用户>\Documents 或 C:\Documents and Settings\<用户>\My Documents 目录下创建工作区，但可以任意选择其位置。每次执行 CCS 都会要求工作区目录。如果计划对所有项目使用一个目录，只需选中“Use this as the default and do not ask again（默认使用此目录且不再询问）”选项。以后随时可以在 CCS 中更改工作区。

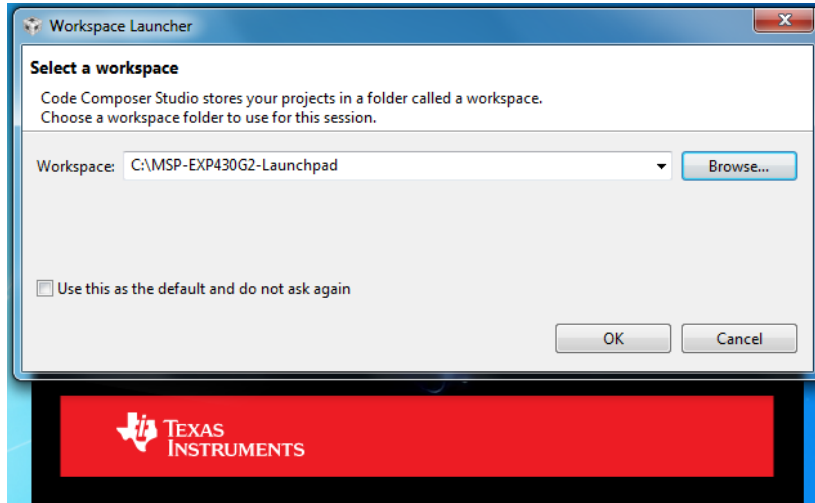


图 2.2.5 Workspace 选择窗口

(4) 单击 OK，第一次运行 CCS 需进行软件许可的选择，如图 2.2.6 所示。

在此，选择 CODE SIZE LIMITED(MSP430)选项，在该选项下，对于 MSP430，CCS 免费开放 16KB 的程序空间；若您有软件许可，可以参考以下链接进行软件许可的认证：http://processors.wiki.ti.com/index.php/GSG:CCSv5_Running_for_the_first_time，单击 Finish 即可进入 CCSv5.1 软件开发集成环境，如图 2.2.7 所示。

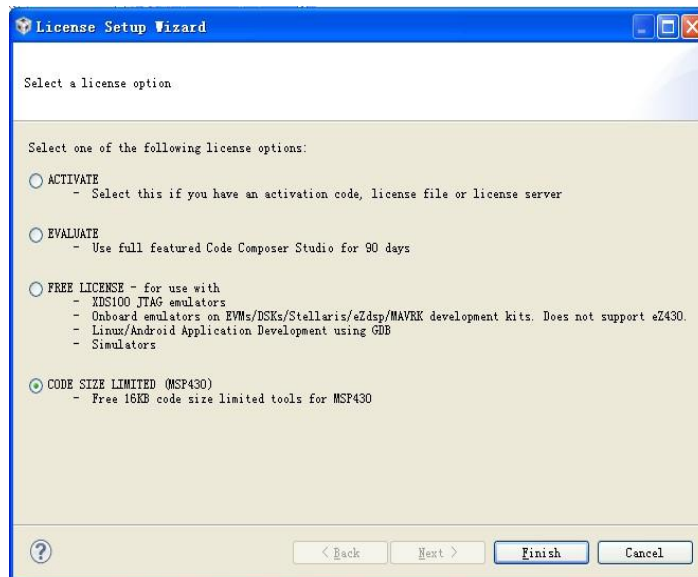


图 2.2.6 软件许可选择窗口

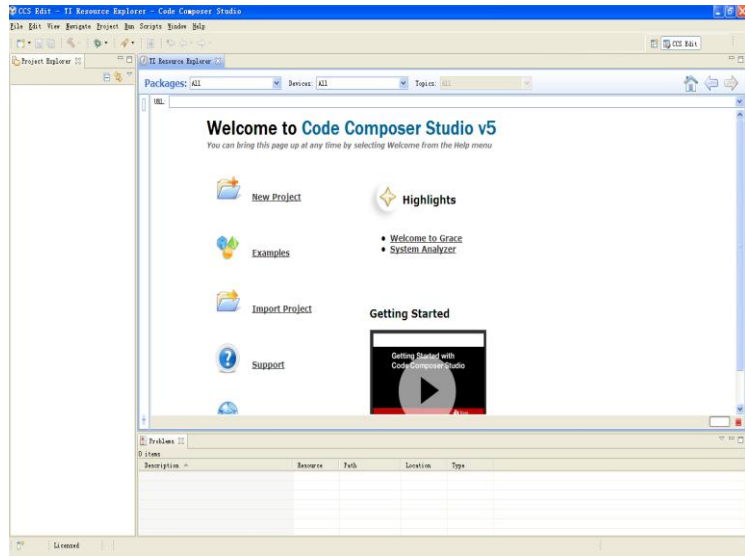


图 2.2.7 CCSv5 软件开发集成环境界面

第三节 CCS 的调试

2.3.1 利用 CCSv5.1 导入已有工程

(1) 首先打开 CCSv5.1 并确定工作区间 C:\MSP-EXP430G2-Launchpad，选择 File-->Import 弹出图 2.3.1 对话框，展开 Code Composer Studio 选择 Existing CCS/CCE Eclipse Projects。

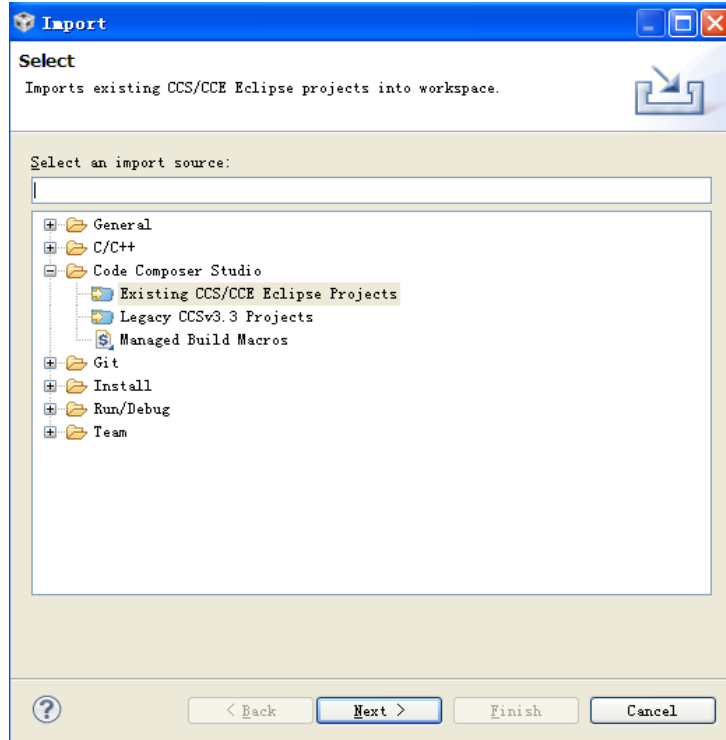


图 2.3.1 导入新的 CCSv5 工程文件

(2) 单击 Next 得到图 2.3.2 对话框。

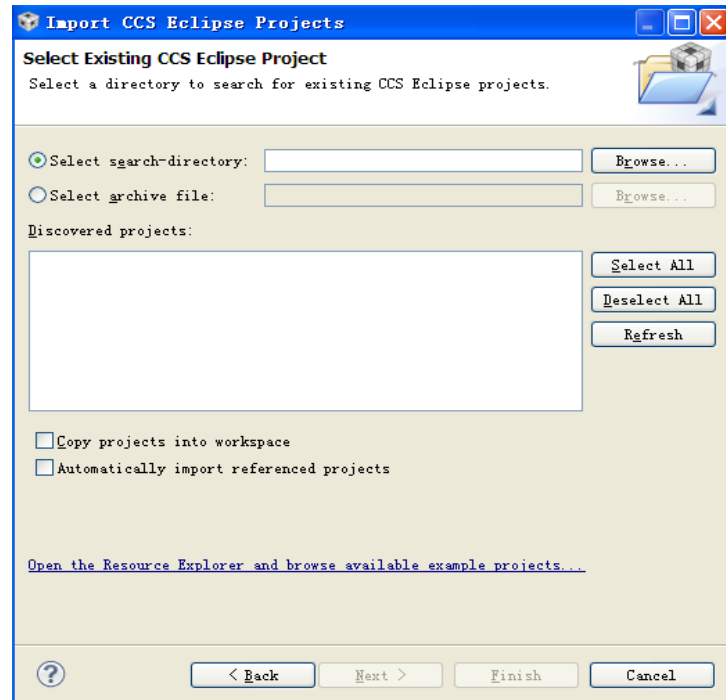


图 2.3.2 选择导入工程目录

(3) 单击 Browse 选择需导入的工程所在目录，在此，我们选择：

C:\MSP-EXP430G2-Launchpad\MSP-EXP430G2-Launchpad (需在此之前, 将实验代码复制到工作区间下), 得到图 2.3.3。

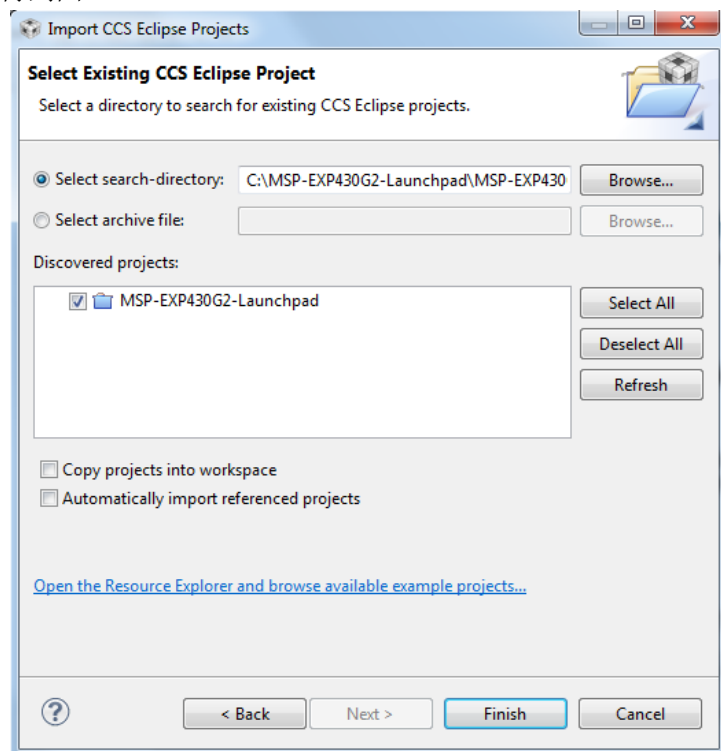


图 2.3.3 选择导入工程

(4) 单击 **Finish**, 即可完成既有工程的导入。

(5) 此外也可以通过 **project-->Import Existing CCS/CCE Eclipse Project** 导入已有工程, 如图 2.3.4 所示, 其他步骤与步骤(2)-(4)相同

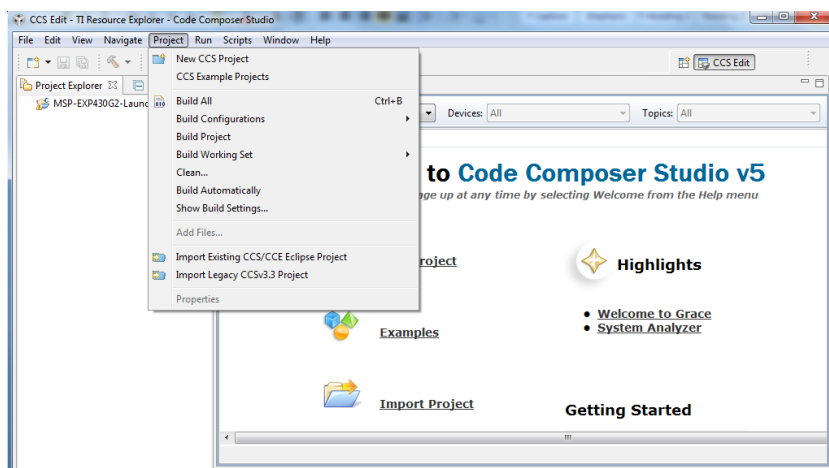


图 2.3.4 导入工程

2.3.2 利用 CCSv5.1 新建工程

(1) 首先打开 CCSv5.1 并确定工作区间, 然后选择 **File-->New-->CCS Project** 弹出图 2.3.5 对话框。

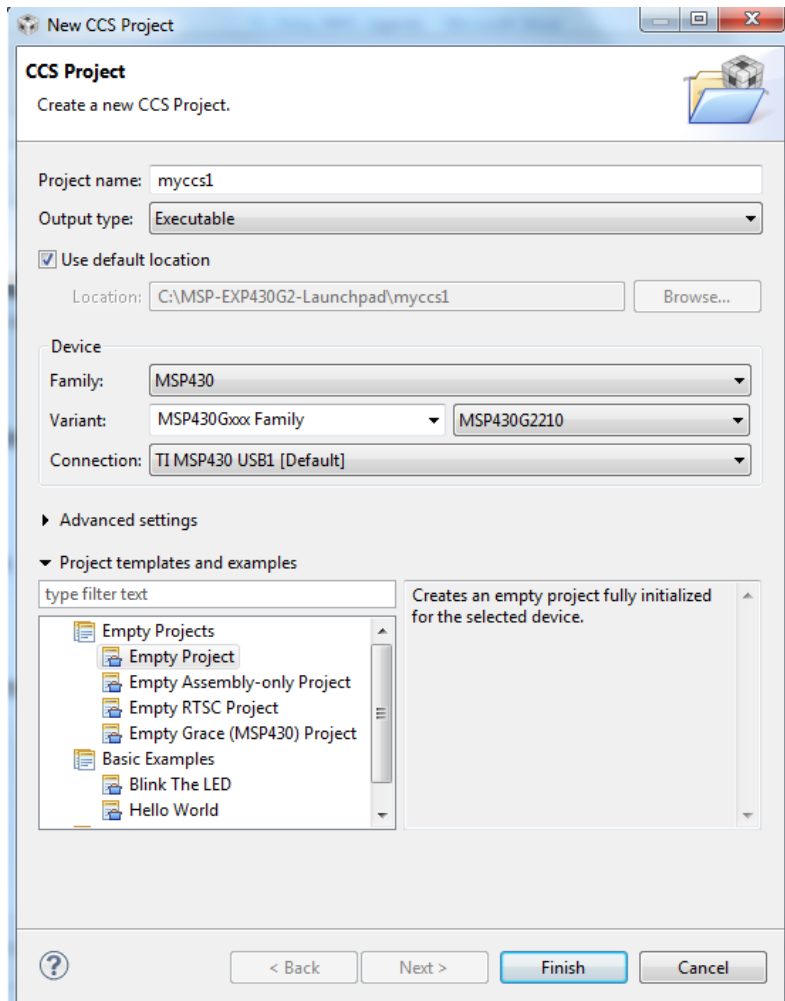


图 2.3.5 新建 CCS 工程对话框

(2) 在 Project name 中输入新建工程的名称，在此输入 myccs1。

(3) 在 Output type 中有两个选项：Executable 和 Static library，前者为构建一个完整的可执行程序，后者为静态库。在此保留：Executable。

(4) 在 Device 部分选择器件的型号：在此 Family 选择 MSP430；Variant 选择 MSP430Gxxxx family，芯片选择 MSP430G2210；Connection 保持默认。

(5) 选择空工程，然后单击 Finish 完成新工程的创建。

(6) 创建的工程将显示在 Project Explorer 中，如图 2.3.6 所示。

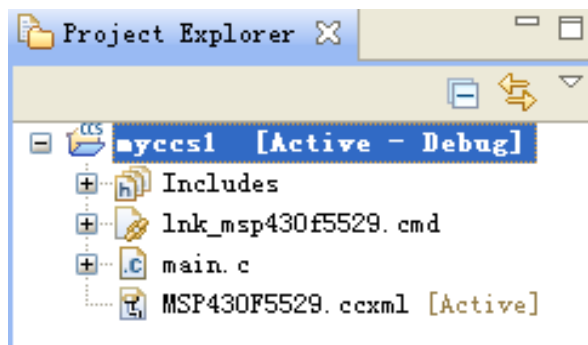


图 2.3.6 初步创建的新工程

特别提示：若要新建或导入已有.h 或.c 文件，步骤如下：

(7) 新建.h 文件：在工程名上右键点击，选择 New-->Header File 得到图 2.3.7 对话框。

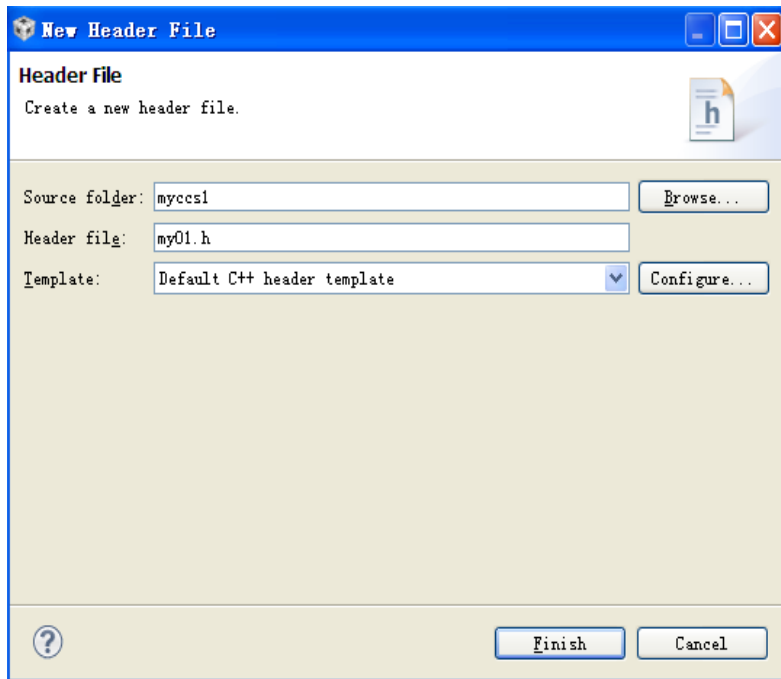


图 2.3.7 新建.h 文件对话框

在 Header file 中输入头文件的名称，注意必须以.h 结尾，在此输入 my01.h。

(8) 新建.c 文件：在工程名上右键单击，选择 New-->source file 得到图 2.3.8 对话框。

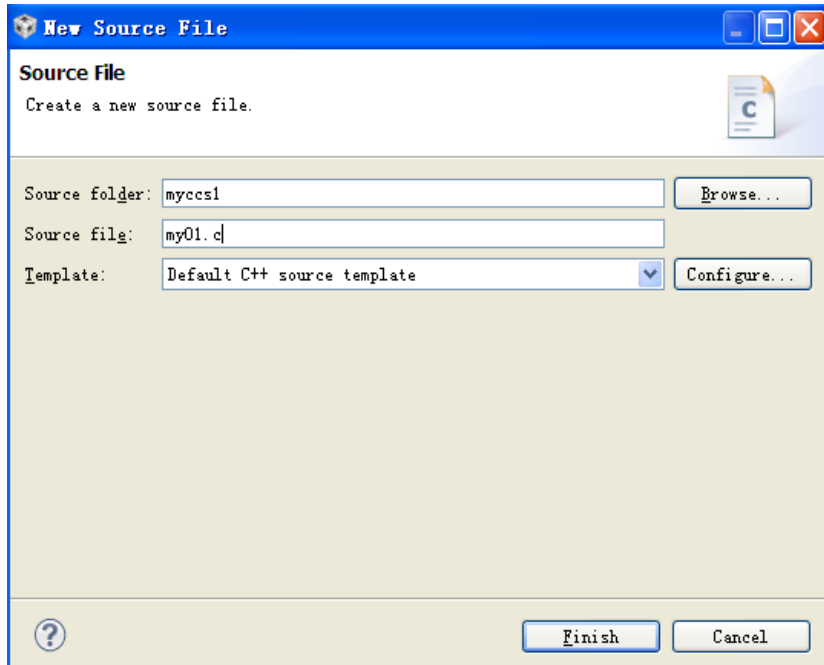


图 2.3.8 新建.c 文件对话框

在 Source file 中输入 c 文件的名称，注意必须以.c 结尾，在此输入 my01.c。

(9) 导入已有.h 或.c 文件：在工程名上右键单击，选择 Add Files 得到如 2.3.9 对话框。

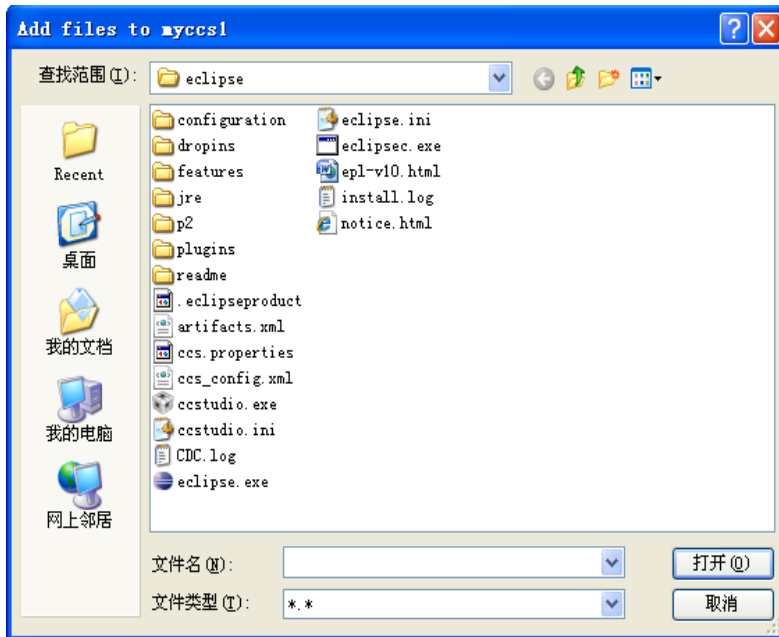


图 2.3.9 导入已有文件对话框

找到所需导入的文件位置，单击打开，得到图 2.3.10 对话框。

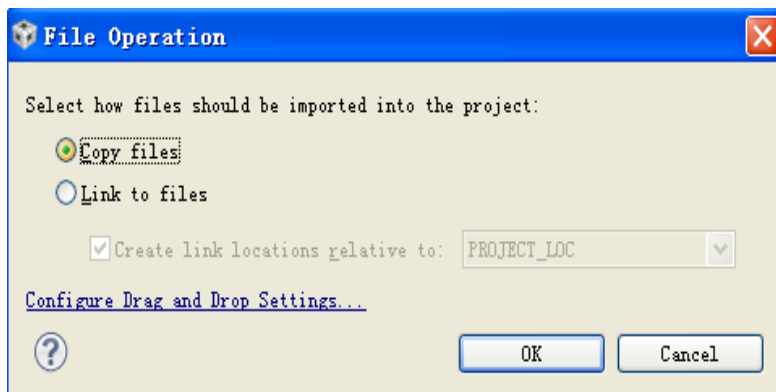


图 2.3.10 添加或连接现有文件

选择 Copy files，单击 OK，即可将已有文件导入到工程中。

若已用其它编程软件（例如 IAR），完成了整个工程的开发，该工程无法直接移植入 CCSv5，但可以通过在 CCSv5 中新建工程，并根据步骤（7）、（8）和（9）新建或导入已有.h 和.c 文件，从而完成整个工程的移植。

2.3.3 利用 CCSv5.1 调试工程

2.3.3.1 创建目标配置文件

（1）在开始调试之前，有必要确认目标配置文件是否已经创建并配置正确。在此以实验一为例进行讲解：首先导入 Launchpad 实验的工程，导入步骤请参考 2.3.1 节，如图 2.3.11 所示，其中 MSP430G2231.ccxml 目标配置文件已经正确创建，即可以进行编译调试，无需重新创建；若目标配置文件未创建或创建错误，则需进行创建。为了讲解目标配置文件创建过程，在此对 LAB1 的工程再次创建目标配置文件。

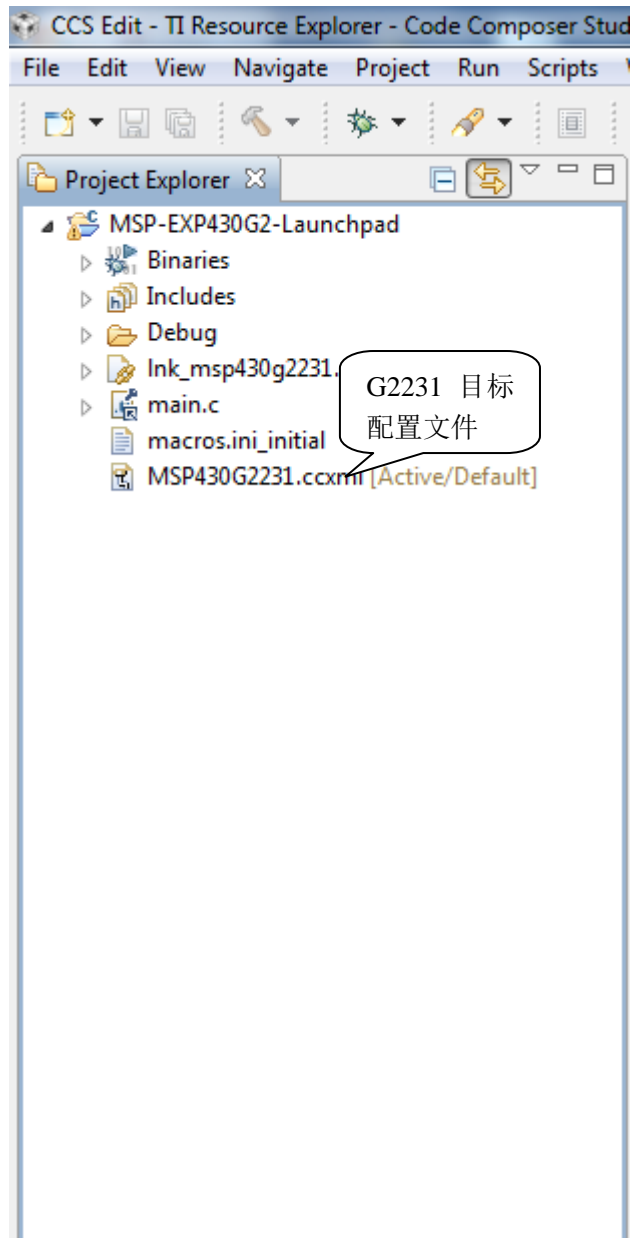


图 2.3.11 导入工程目标配置文件

(2) 创建目标配置文件步骤如下：右键单击项目名称，并选择 NEW --> Target Configuration File。

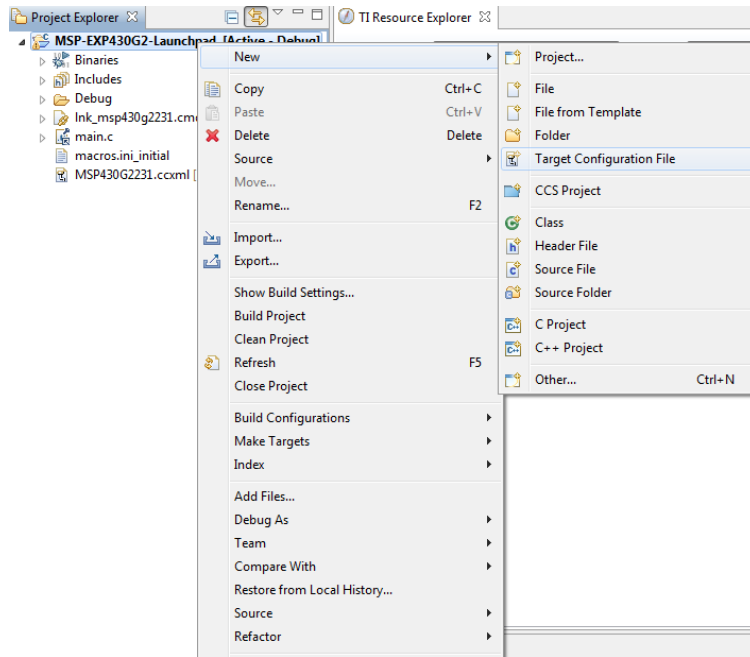


图 2.3.12 创建新的目标

(3) 在 File name 中键入后缀为 .ccxml 的配置文件名，将配置文件命名为 MSP-EXP430G2231.ccxml，如图 2.3.13 所示。

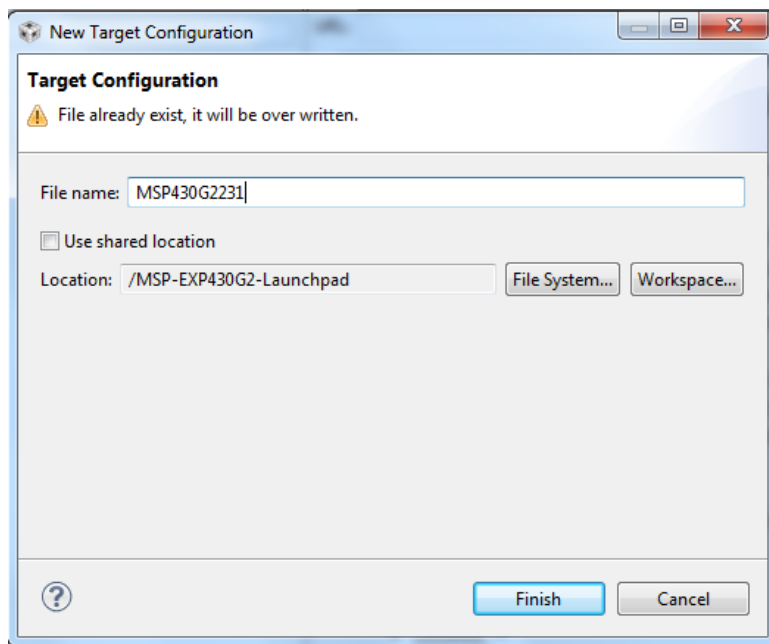


图 2.3.13 目标配置文件名

(4) 单击 Finish，将打开目标配置编辑器，如图 2.3.14 所示。

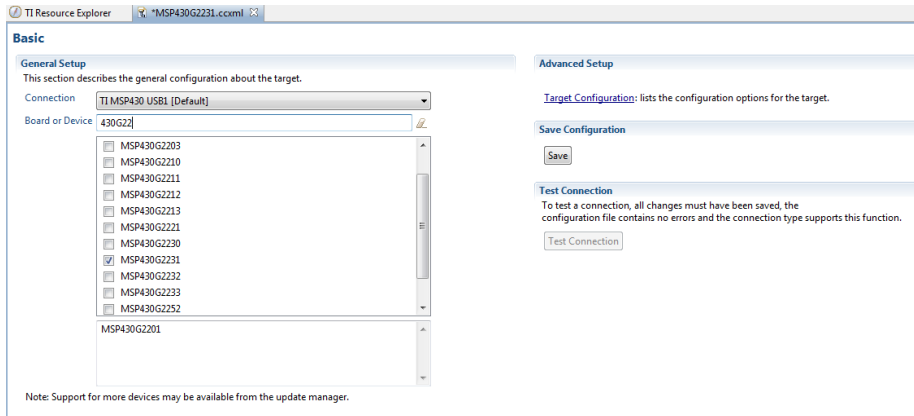


图 2.3.14 目标配置编辑器

(5) 将 Connection 选项保持默认: TI MSP430 USB1 (Default), 在 Board or Device 菜单中选择单片机型号, 在此选择 MSP430G2231。配置完成之后, 单击 Save, 配置将自动设为活动模式。如图 2.3.15 所示, 一个项目可以有多个目标配置, 但只有一个目标配置在活动模式。要查看系统上所有现有目标配置, 只需要去 View --> Target Configurations 查看。

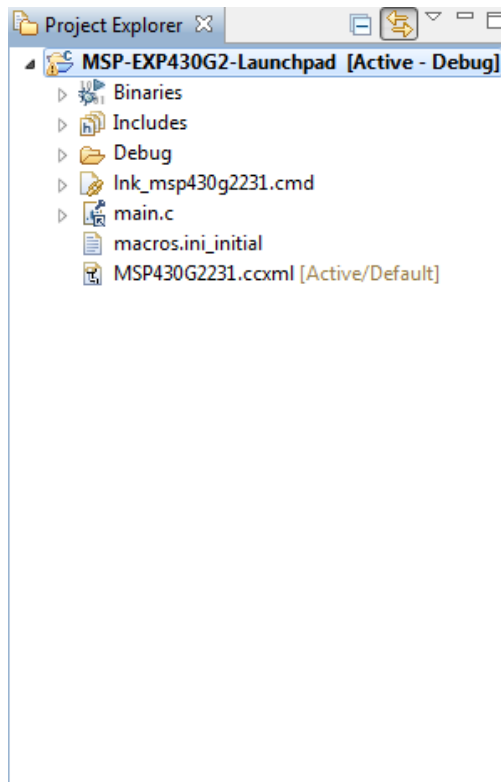


图 2.3.15 项目与配置后的目标文件

2.3.3.2 启动调试器

(1) 首先将 LAB1 工程进行编译通过: 选择 Project-->Build Project, 编译目标工程。在第一次编译实验工程时, 系统会提示自动创建 rts430x1.lib 库文件, 您可以选择等待创建完成, 但可能会花费较长的时间。或者, 为了方便, 推荐在编译之前将本实验文件夹内的 rts430x1.lib 库文件复制到 CCSV5.1 的库资源文件夹内, 其复制路径为: ---\tools\compiler\msp430\lib(---为 CCSv5.1 的安装路径)。

编译结果, 如图 2.3.16 所示, 表示编译没有错误产生, 可以进行下载调试; 如果程序有错误, 将会在 Problems 窗口显示, 根据显示的错误修改程序, 并重新编译, 直到无错误提示。

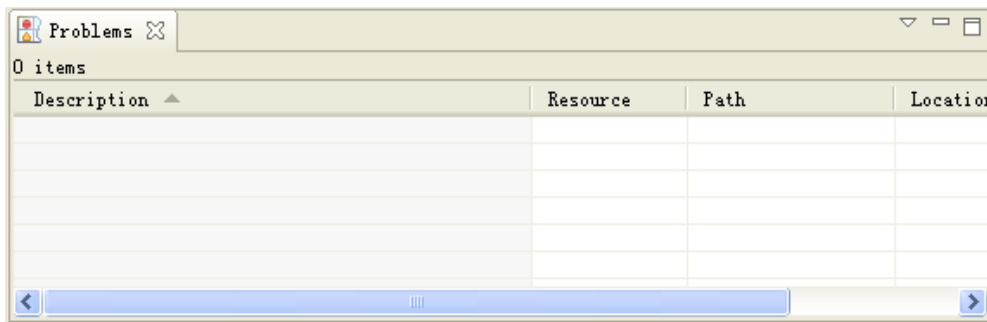



图 2.3.16 LAB1 工程调试结果

(2) 单击绿色的 Debug 按钮  进行下载调试，得到图 2.3.17 所示的界面。

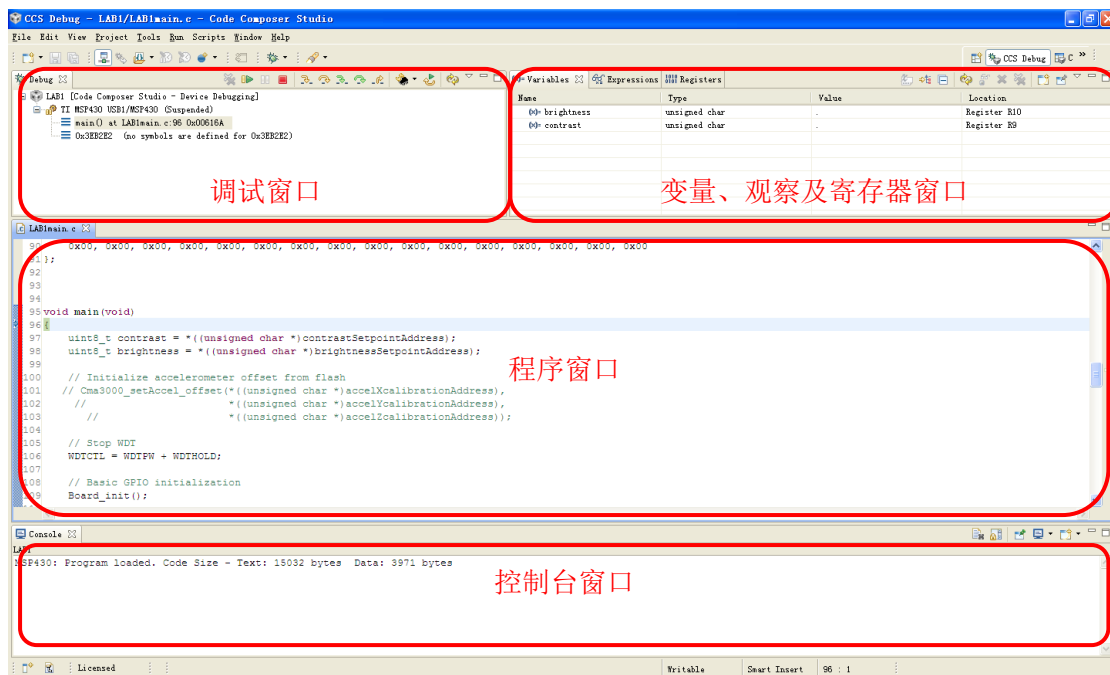










图 2.3.17 调试窗口界面

(3) 单击运行图标  运行程序，观察显示的结果。在程序调试的过程中，可通过设置断点来调试程序：选择需要设置断点的位置，右击鼠标选择 Breakpoints→Breakpoint，断点设置成功后将显示图标 ，可以通过双击该图标来取消该断点。程序运行的过程中可以通过单步调试按钮    配合断点单步的调试程序，单击重新开始图标  定位到 main() 函数，单击复位按钮  复位。可通过中止按钮  返回到编辑界面。

(4) 在程序调试的过程中，可以通过 CCSV5.1 查看变量、寄存器、汇编程序或者是 Memory 等的信息显示程序运行的结果，以和预期的结果进行比较，从而顺利地调试程序。单击菜单 View→Variables，可以查看到变量的值，如图 2.3.18 所示。

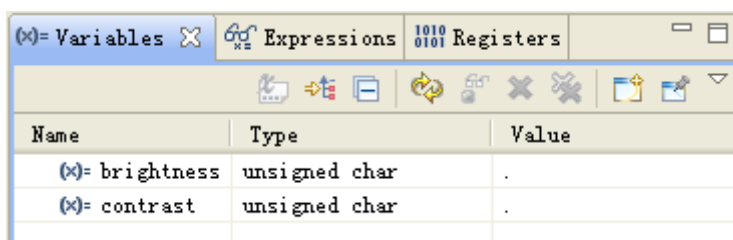


图 2.3.18 变量查看窗口

(5) 点击菜单 View→Registers, 可以查看到寄存器的值, 如图 2.3.19 所示。

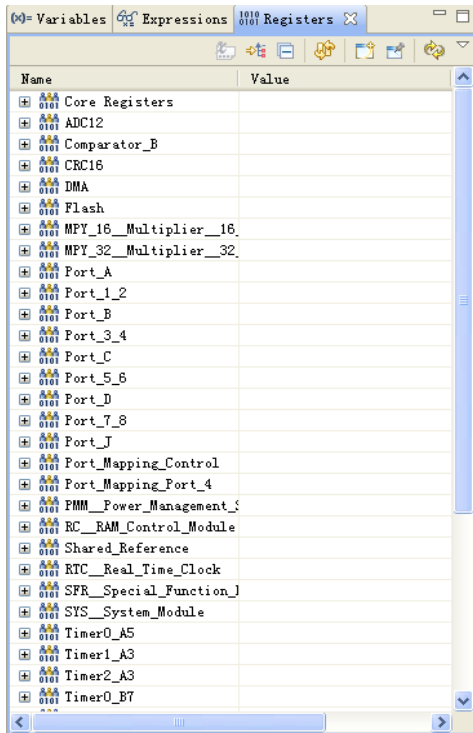



图 2.3.19 寄存器查看窗口

(6) 点击菜单 View→Expressions, 可以得到观察窗口, 如图 2.3.20 所示。可以通过  Add new 添加观察变量, 或者在所需观察的变量上右击, 选择 Add Watch Expression 添加到观察窗口。

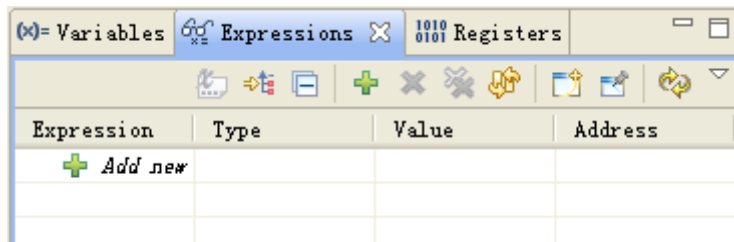


图 2.3.20 观察窗口

(7) 点击菜单 View→Disassembly, 可以得到汇编程序观察窗口, 如图 2.3.21 所示。



图 2. 3. 21 汇编程序观察窗口

(8) 点击菜单 View→Memory Browser，可以得到内存查看窗口，如图 2. 3. 22 所示。

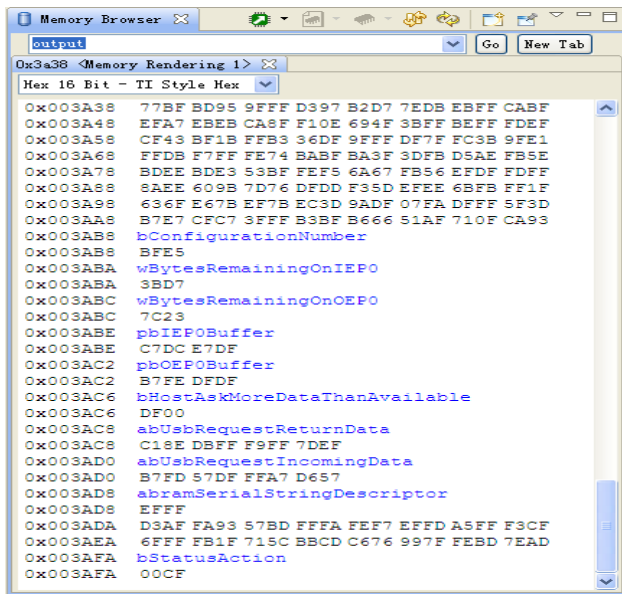


图 2. 3. 22 内存查看窗口

(9) 点击菜单 View→Break points，可以得到断点查看窗口，如图 2. 3. 23 所示。

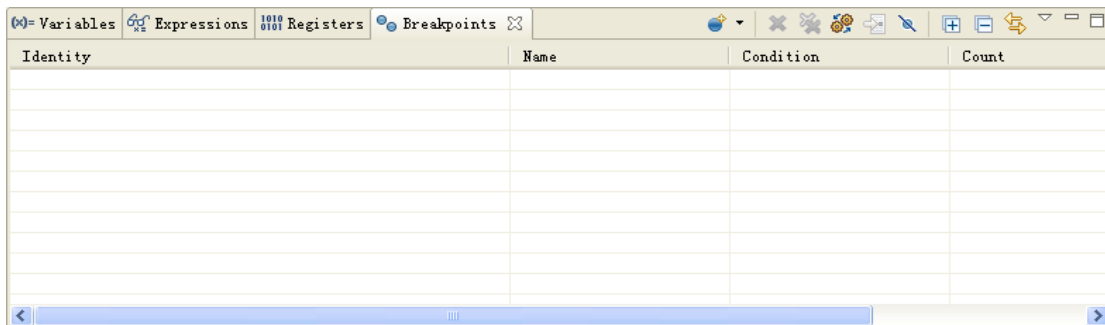


图 2. 3. 23 断点查看窗口

2. 3. 4 CCSv5.1 资源管理器介绍及应用

2.3.4.1 CC5.1 资源管理器介绍

(1) CCSv5.1 具有很强大的功能，并且其内部的资源也非常丰富，利用其内部资源进行 MSP430 单片机开发，将会非常方便。现在演示 CCSv5.1 资源管理器的应用。如图 2.3.24 所示，通过 Help→Welcome to CCS 打开 CCSv5.1 的欢迎界面。

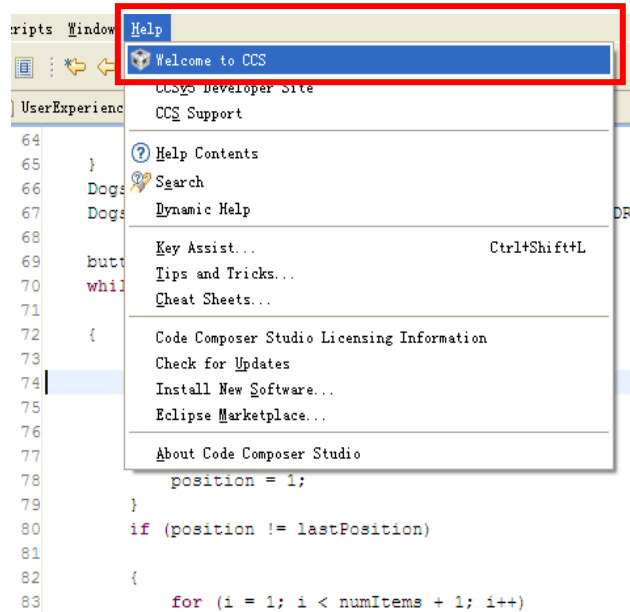


图 2.3.24 欢迎界面打开途径

(2) 具体 TI 欢迎界面如图 2.3.25 所示，利用 New Project 链接可以新建 CCS 工程，具体新建步骤可以参考 2.3.2 节：利用 CCSv5.1 新建工程；利用 Examples 链接可以搜索到示例程序资源；利用 Import Project 链接可以导入已有 CCS 工程文件，具体导入步骤可以参考 2.3.1 节：利用 CCSv5.1 导入已有工程；利用 Support 链接可以在线获得技术支持；利用 Web Resources 链接可以进入 CCSv5.1 网络教程，学习 CCSv5.1 有关知识。

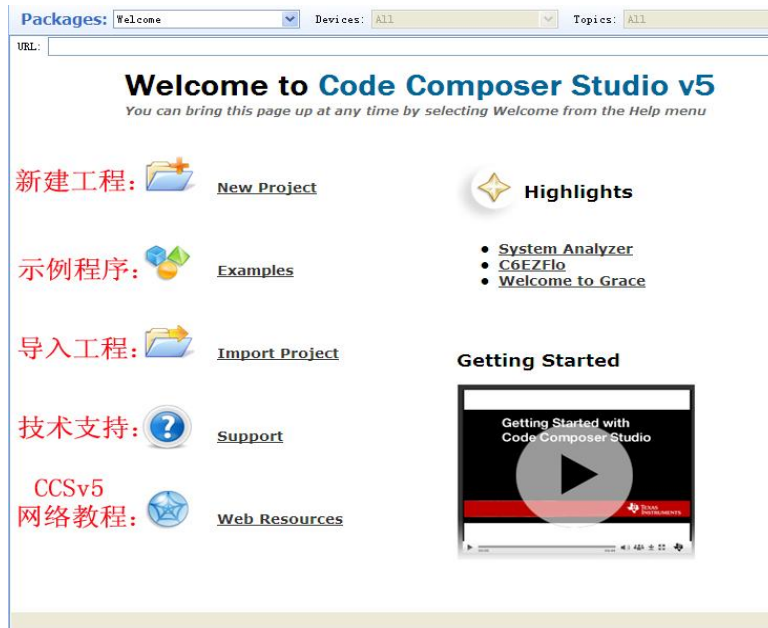


图 2.3.25 TI 欢迎窗口界面

(3) 在“Packages”下拉菜单下选择 ALL，进入 CCSv5.1 资源管理器，如图 2.3.26 所示。在左列资源浏览器中，包含 MSP430Ware。MSP430Ware 将所有的 MSP430 MCU 器件的代码范例、数据表与其他设计资

源整合成一个便于使用的程序包，基本上包含了成为一名 MSP430 MCU 专家所需要的一切。

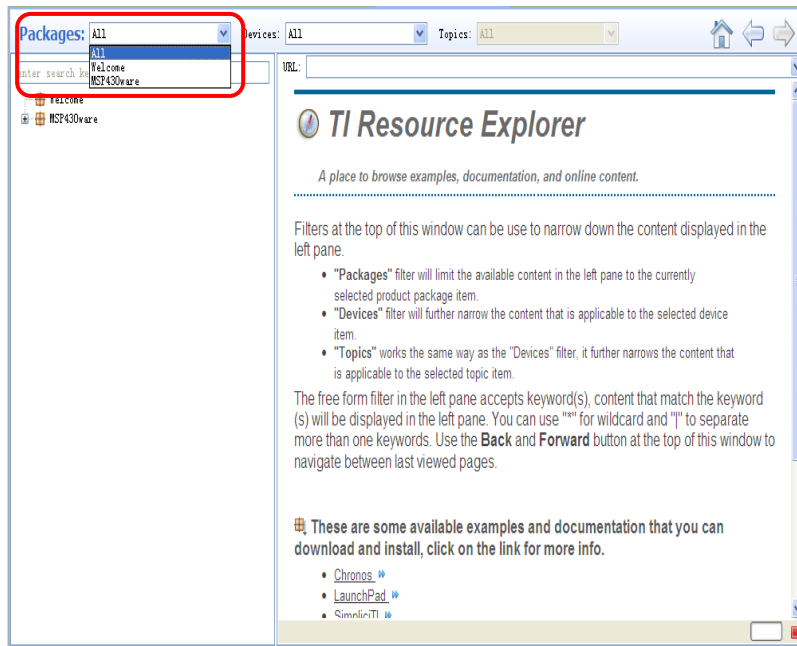


图 2. 3. 26 CCSv5. 1 资源管理器窗口

(4) 如图 2. 3. 27 所示，展开 MSP430ware，其包含三个方面内容：MSP430 单片机资源、开发装置资源以及 MSP430 资源库。



图 2. 3. 27 MSP430ware 界面

(5) 展开 MSP430 单片机资源，得到如图 2. 3. 28 所示的界面，展开 MSP430F5xx/6xx，其中包含 F5xx/6xx 系列的用户指导、数据手册、勘误表以及示例代码。



图 2. 3. 28 单片机资源管理图

(6) 展开 Code Examples，在下拉选项上选择 MSP430F552x，在右面窗口中，将得到 MSP430F552x 有关各内部外设的应用程序资源，如图 2. 3. 29 所示。若您打算在 ADC 模块的基础上，开发 MSP430，首先可以选择一个有关 ADC 的工程，作为讲解，在此选择第二个工程：MSP430F55xx_adc_01. c。单击该工程名称，

将会弹出一个对话框，选择单片机型号，在此选择 MSP430F5529，单击 OK。之后您将在工程浏览器中，看到导入的工程：MSP430F55xx_adc_01，您可以在在此基础上进行单片机的开发。

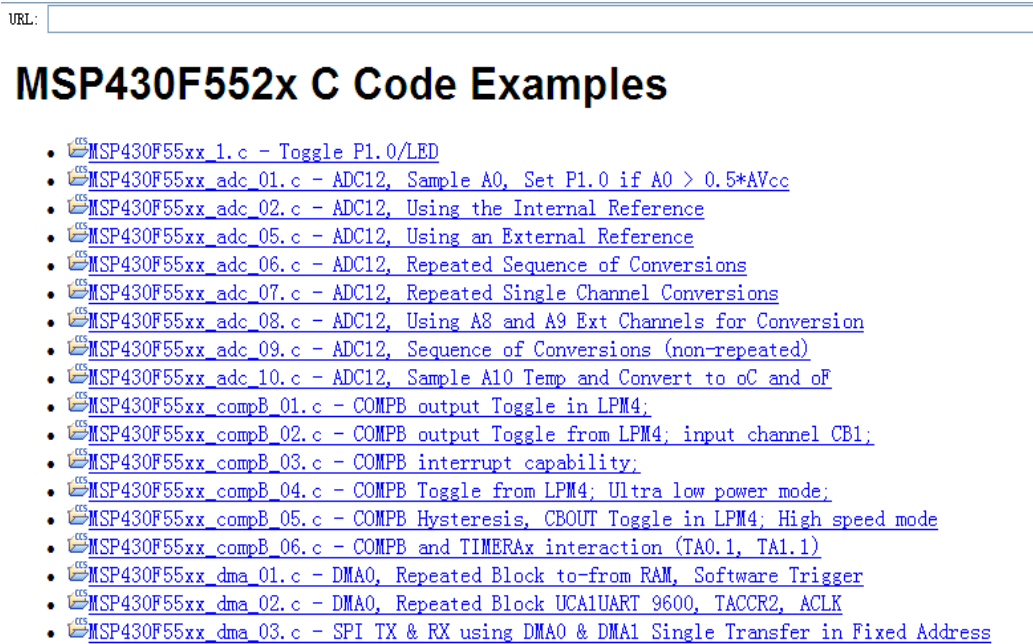


图 2.3.29 MSP430F552x 应用程序资源

(7) 展开 Development Tools 开发装置资源，得到如图 2.3.30 所示的界面，其中包含 MSP-EXP430F5529 开发板资源。



图 2.3.30 开发装置资源管理图

(8) 单击 User Experience Project (Code Limited)，在右面窗口中将得到如图 2.3.31 所示窗口。示例程序导入步骤分为四步，在保证开发板仿真器连接正确的前提下（在此利用开发板内置仿真器），单击第一步，将示例工程导入 CCS，您将在资源浏览器中，看到导入的工程：MSP-EXP430F5529 User Experience_16KB，并且第一步和第三步后面蓝色的对号变亮。单击第二步，对示例工程进行编译，编译完成后，将发现第二步后面蓝色的对号变亮。单击第四步，将示例工程下载到开发板。

User Experience Project (Code Limited)

This is the out-of-the-box software code limited version for MSP-EXP430F5529.

These are the steps to import the project, build the project, and debug the project.

Step 1:  Import the example project into CCS → 将示例工程导入 CCS

Click on the link above to import the project. The imported project is available in the **Project Explorer** view, expand the project node to browse the imported source files. To modify source code, double clicks on the source file within the project to open the source file editor.

Step 2:  Build the imported project → 编译示例工程

To change build options, right click on the project and select **Properties** from the context menu. To build the project, select the link above, or select the **Build** toolbar button, or select the **Project | Build Project** menu item.

Step 3:  Debugger Configuration → 调试器配置

Connection: **none**
Click on the link above to change the device connection. Additionally, this option is also available in the project properties.

Step 4:  Debug the imported project → 下载示例工程

Click on the link above to launch a debug session for the **User Experience Project (Code Limited)** project and switch to the **CCS Debug Perspective**. Additionally, these are other methods to start a project debug session. Select the project in the **Project Explorer** view and click on the bug toolbar button. To relaunch a previous debug session, click on the small arrow beside the bug toolbar button and select one of the debug session from the history.

图 2.3.31 MSP-EXP430F5529 原板载程序资源

(9) 展开 Libraries 资源库，得到如图 2.3.32 所示的界面，其中包含 MSP430 驱动程序库以及 USB 的开发资源包。“MSP430 驱动程序库”为全新高级 API，这种新型驱动程序库能够使用户更容易地对 MSP430 硬件进行开发。就目前而言，MSP430 驱动程序库可支持 MSP430F5xx 和 F6xx 器件。MSP430USB 开发资源包包含了开发一个基于 USB 的 MSP430 项目所需的所有源代码和示例应用程序，该开发资源包只支持 MSP430USB 设备。



图 2.3.32 资源库管理图

2.3.4.2 430Ware 使用指南

(1) 430Ware 是 CCS 中的一个附带应用软件，在安装 CCSV5 的时候可选择同时安装 430Ware，在 TI 官网上也提供单独的 430Ware 安装程序下载。（<http://www.ti.com/tool/msp430ware>）在 430Ware 中可以容易地找到 MSP430 所有系列型号的 Datasheet，User's guide 以及参考例程，此外 430Ware 还提供了大多数 TI 开发板（持续更新中）的用户指南，硬件设计文档以及参考例程。针对 F5 和 F6 系列还提供了驱动程序文件，以方便用户进行上层软件的开发。

(2) 在 CCS 中单击“view”->“TI Resource Explorer”，在主窗口体会显示如图 2.3.33 所示的界面。其中，Package 右侧的下拉窗口中可以观察目前 CCS 中安装的所有附加软件。在 package 旁的下拉菜单中

选择 MSP430Ware，进入 430Ware 的界面。

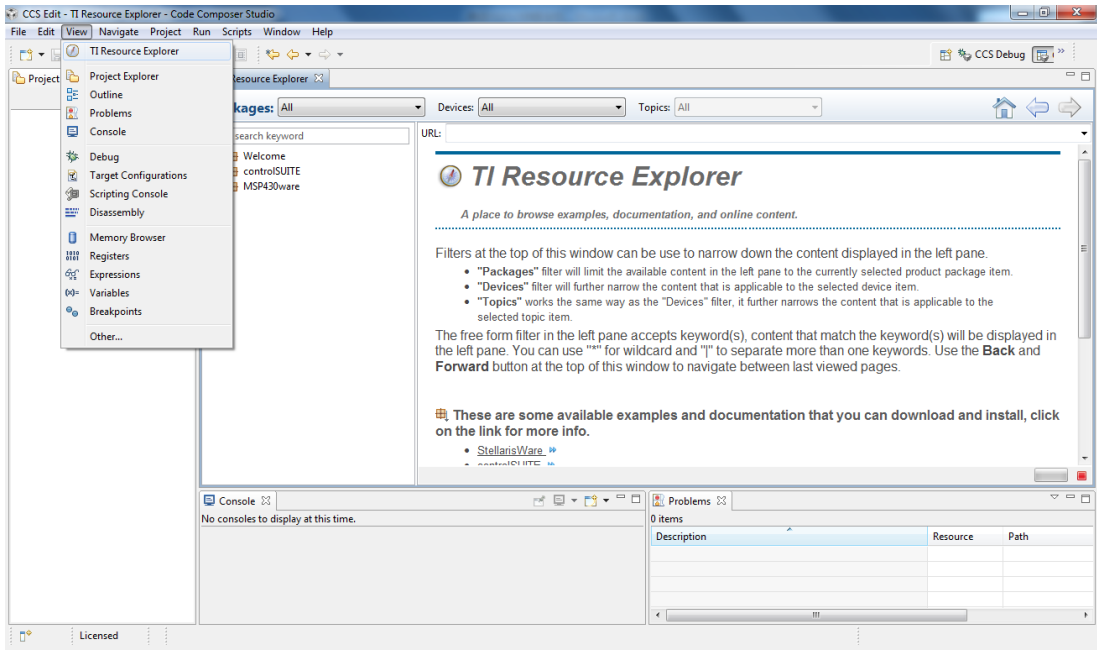


图 2.3.33 TI Resource Explorer 界面

(3) 在 430Ware 的界面左侧可以看到 3 个子菜单，分别是 Device，里面包含 MSP430 所有的系列型号；Development Tools，里面包括 TIMSP430 较新的一些开发套件的资料；和 Libraries，包含了可用于 F5 和 F6 系列的驱动库函数以及 USB 的驱动函数。

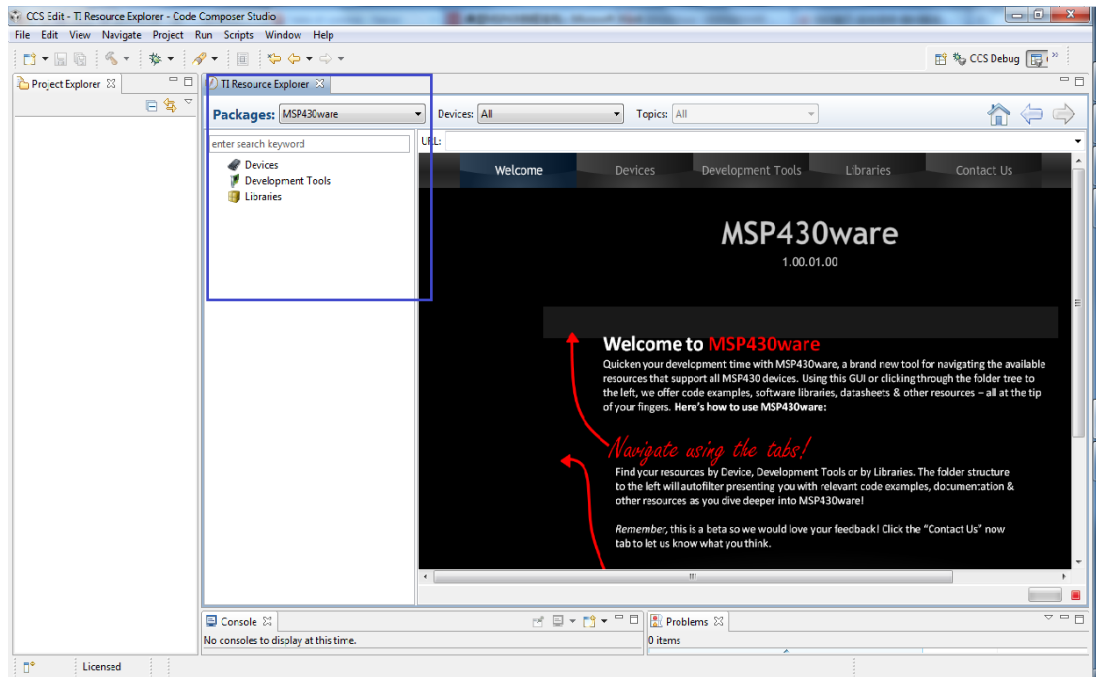


图 2.3.34 TI Resource Explorer 界面

(4) 单击图 2.3.35 所示界面菜单前的三角下拉键，查看下级菜单，可以看到在 Devices 的子目录下有目前所有的 MSP430 的型号，找到正在使用的型号，例如 MSP430G2xx，同样单击文字前的三角下拉键，在子目录可以找到该系列的 User's Guide，在用户指南中有对该系列 MSP430 的 CPU 以及外围模块，包括寄存器配置，工作模式的详细介绍和使用说明；同时可以找到的是该系列的 Datasheet，数据手册是与具体的型号相关，所以在 datasheet 的子目录中会看到具体不同型号的数据手册；最后在这里还可以找到的

是参考代码。

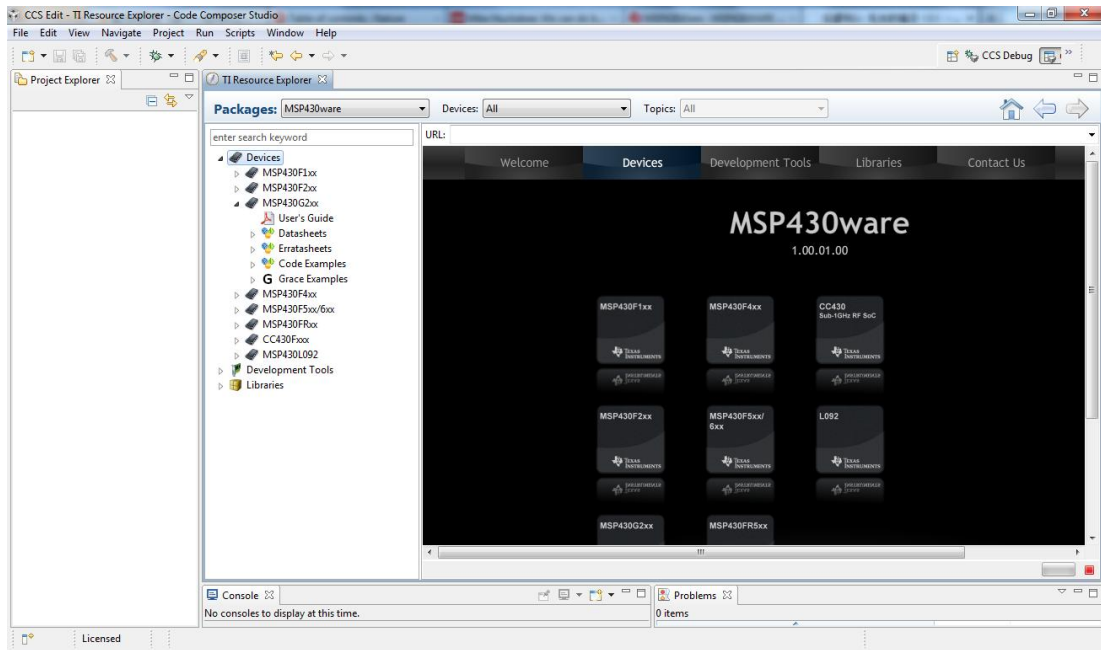


图 2. 3. 35 MSP430ware 界面

(5) 在 430Ware 中提供不同型号的 CCS 示例程序，以及基于 Grace 的示例程序供开发者参考。如图 2. 3. 36 所示，选择具体型号后，在右侧窗口中看到提供到的参考示例程序。为更好地帮助用户了解 MSP430 的外设，430Ware 中提供了基于所有外设的参考例程，从示例程序的名字中可以看出该示例程序涉及的外设，同时在窗口中还可以看到关于该例程的简单描述，帮助用户更快地找到最合适的参考程序。如图 2. 3. 37 所示，单击选中的参考例程，在弹出的对话框中选择连接的目标芯片型号。

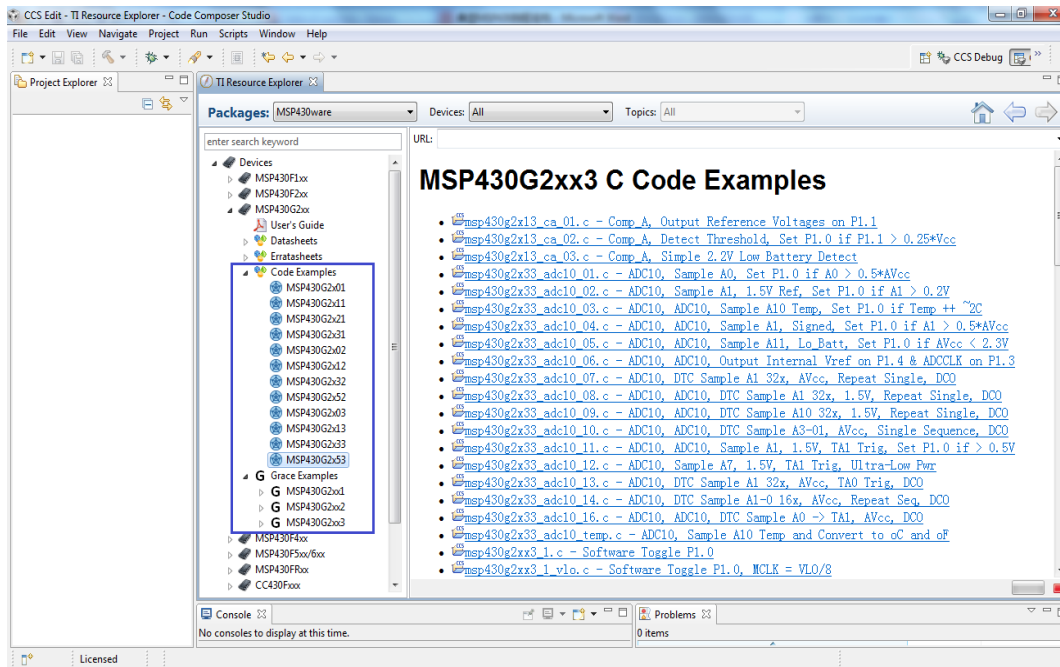


图 2. 3. 36 MSP430ware 界面

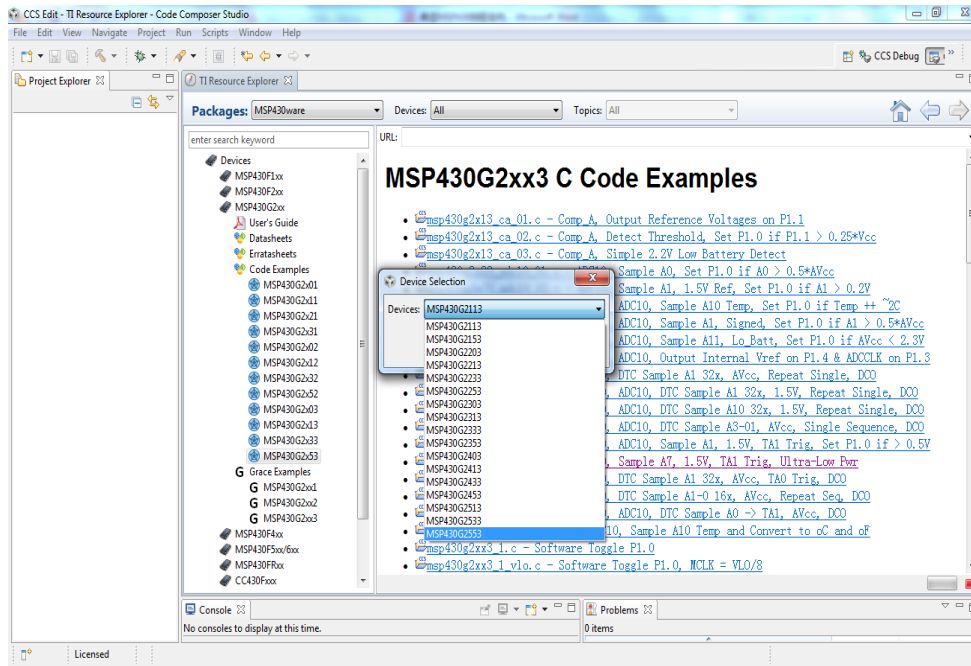


图 2.3.37 MSP430ware 界面

(6) 经过上一步操作后，CCS 会自动生成一个包含该示例程序的工程，用户可以直接进行编译，下载和调试。在 Development Tools 的子目录中可以找到 TI 基于 MSP430 的开发板，部分资源已经整合在软件中，另外还有部分型号在 430Ware 中也给出了链接以方便用户的查找和使用。在该目录下可以方便地找到相应型号的开发板的用户指南，硬件电路图以及参考例程。

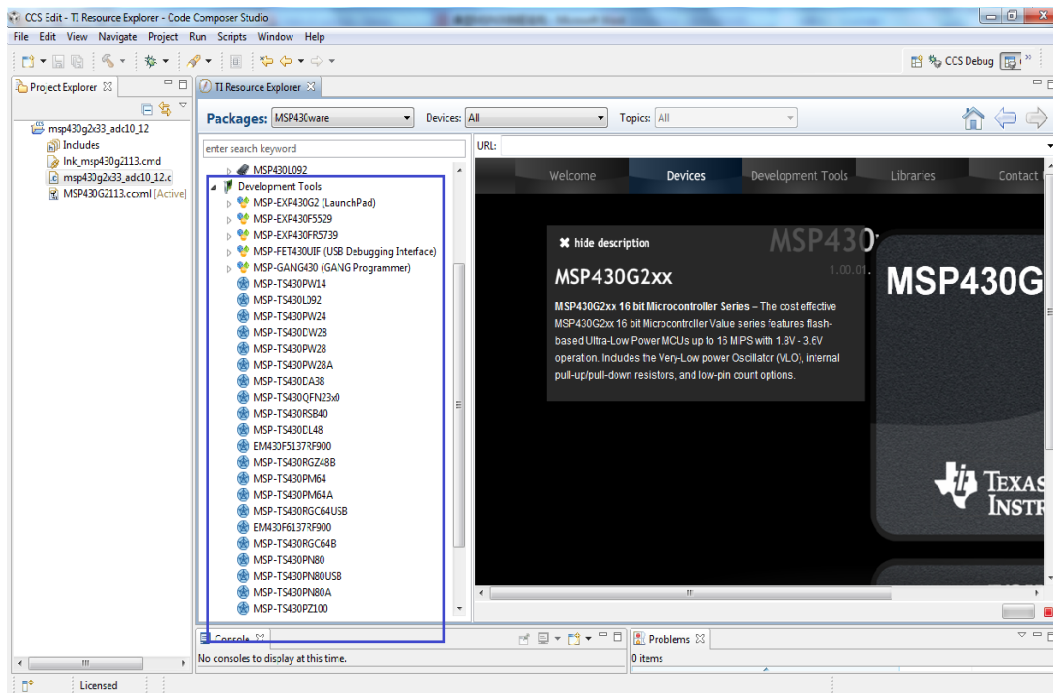


图 2.3.38 MSP430ware 界面

(7) 为简化用户上层软件开发，TI 给出了 MSP430 外围模块的驱动库函数，这样用户不用过多地去考虑底层寄存器的配置。这些支持可以在 430Ware 的 Libraries 子目录中方便地找到。目前对 DriverLib 的支持仅限于 MSP430F5 和 F6 系列。

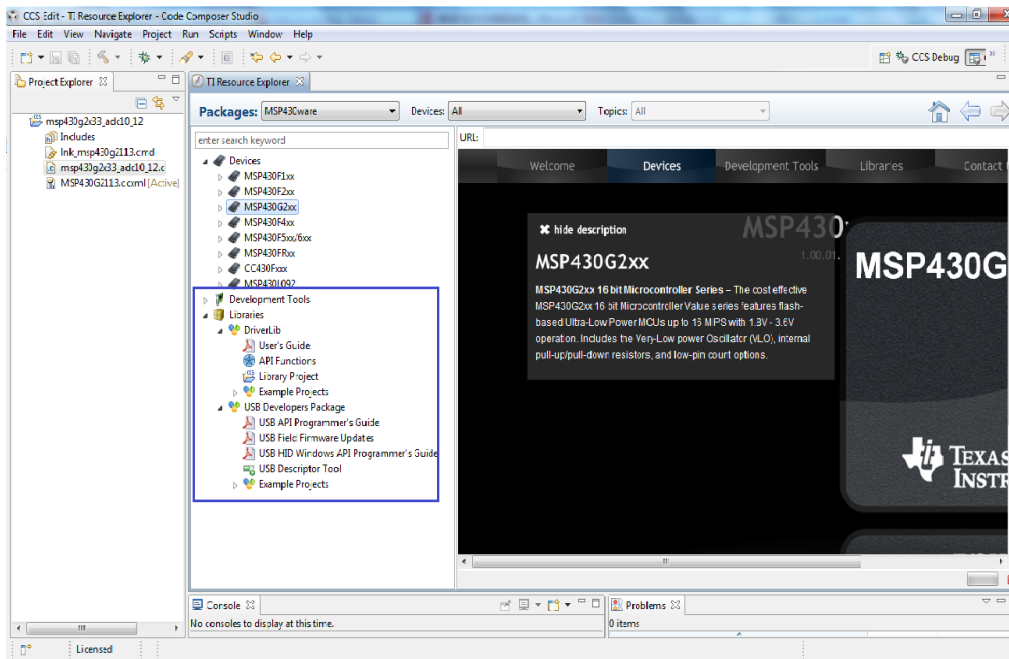


图 2.3.39 MSP430ware 界面驱动库函数

通过上述描述可以看出，430Ware 是一个非常有用的工具，利用 430Ware 可以很方便地找到进行 430 开发所需要的一些帮助，包括用户指南，数据手册和参考例程。

(8) 管理断点

作为任何调试器都会拥有的最基本功能，CCSv5 中的断点添加了一系列选项，帮助增加调试进程的灵活性。

硬件断点可从 IDE 直接进行设置；

软件断点仅受到设备可用内存的限制；

软件断点可设置为无条件或有条件停止；

除了停止目标之外，软件断点还可执行其他功能：文件 I/O 传输、屏幕更新等。

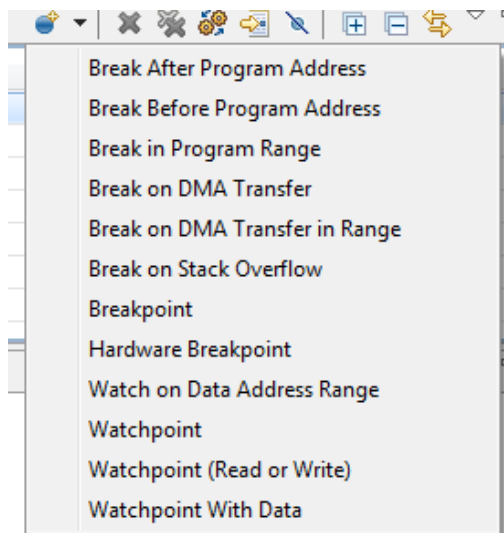




图 2.3.40 断点选项

要设置断点，只需在源代码或反汇编视图中双击代码行即可。硬件  或软件  断点的图标会指示其状态和放置位置。

注意：在优化代码中，有时无法将断点设置到 C 源代码中确切的某一行。这是因为优化器可能会将代码紧缩起来，从而影响汇编指令和 C 源代码之间的相关性。

所有断点（软件、硬件、已启用、已禁用）都可在断点查看器中看到。

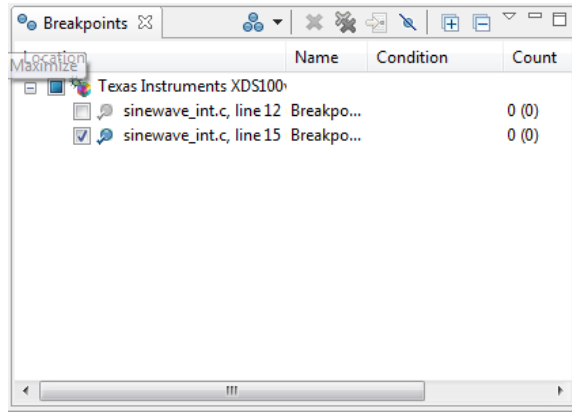


图 2.3.41 断点查看器

要配置断点，只需右键单击蓝点，或者在断点视图中右键单击并选择“Breakpoint Properties...”（断点属性...）”。

(9) 图形显示工具

CCSv5 中提供了一个高级图形和图像可视化工具。它可通过图形形式显示数组，并且可采用多种格式。要添加图形，只需转到菜单“Tools -> Graph（工具 -> 图形）”，然后从各种显示选项中选择。基于时间的图形：“Single Time（单曲线图）”和“Dual Time（双曲线图）”基于频率的图形：所有 FFT 选项 图形窗口中的顶部工具栏可控制多种功能，例如更新速率（冻结、连续、目标停止时或手动）、缩放、配置属性等。



图 2.3.42 图形工具栏

默认情况下，图形窗口会在目标停止时立即更新、使用自动缩放并以样本数显示 X 轴，以整数值显示 Y 轴。所有这些选项都可进行设置。


注意：请记住，图形更新时所传输的数据量可能会影响目标硬件的实时操作。

教程：下面的过程显示了包含正弦波发生器输出内容的图形。

在源代码窗口中，右键单击断点蓝点（已在上一部分设置）并选择“Breakpoint Properties...”（断点属性...）”。

变量 `output[]` 包含 16 个正弦波发生器输出样本，因此整个缓冲区必须立即显示在图形窗口中。单击“Tools -> Graph -> Single Time（工具 -> 图形 -> 单曲线图）”，然后将选项配置如下：

属性	值
采集缓冲区大小	16
Dsp 数据类型	16 位带符号整数
Q_value	15
开始地址	output

屏幕底部应该出现一个图形窗口。如果需要，可通过单击  按钮更改图形属性。

单击“Target -> Run（目标 -> 运行）”。该图形应该以 16 个样本为一组分批更新。要查看 `output` 数组的实际值，请单击“Watch（监视）”选项卡（应当在屏幕右上角部分），然后单击“New（新建）”。键入 `output` 并展开此数组以显示其中的所有值。这些值以 16 位

带符号整数输出，因此可通过调整 Q 值使其标准化：在“Watch（监视）”窗口中选择所有值，右键单击并选择“Q-values -> Q-value(15)（Q 值 -> Q 值(15)）”。

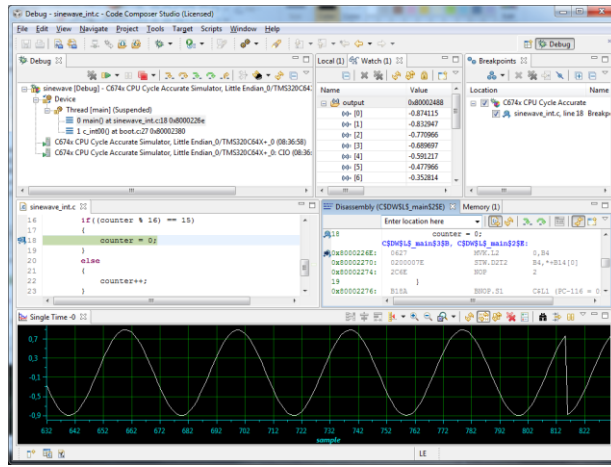


图 2.3.43 正弦波示例

(7) 图像显示工具

要显示图像，只需转到菜单“Tools -> Image（工具 -> 图像）”。

屏幕底部将打开两个视图：“Image（图像）”和“Properties（属性）”。


CCSv5 显示的信息既可以是来自 PC 主机中的文件，也可以是目标开发板中加载的图像。在属性页面中，只需将“Image source（图像源）”选项设置为“File（文件）”或“Connected Device（连接的设备）”即可。

与图形查看器类似，需要设置其他所有属性才能使显示内容有意义。彩色障板、线条尺寸和数据宽度等几种选项会影响图像的正确显示。

教程：要显示加载至目标的图像，请执行以下操作：

转到菜单“View -> Memory（查看 -> 内存）”打开内存视图。

在地址框中键入有效的目标地址：0xC0000000

将图像文件 <sample_24bpp.dat> 加载至 0xC0000000：单击内存操作图标  旁边的三角形，然后单击“Load（加载）”。浏览至下面的目录，然后单击“Next（下一步）”。
C:\Program Files\Texas Instruments\ccsv4\c6000\examples

键入与内存窗口中相同的起始地址，并将“Type-size（类型大小）”设置为 32 位。按下图所示设置属性：

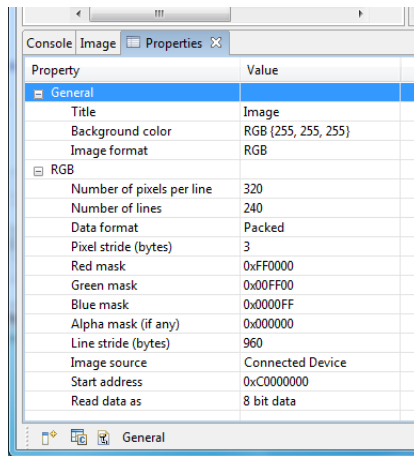


图 2.3.44 图像属性

选择“Image (图像)”选项卡，然后右键单击并选择“Refresh (刷新)”。应该会显示下面的图像。

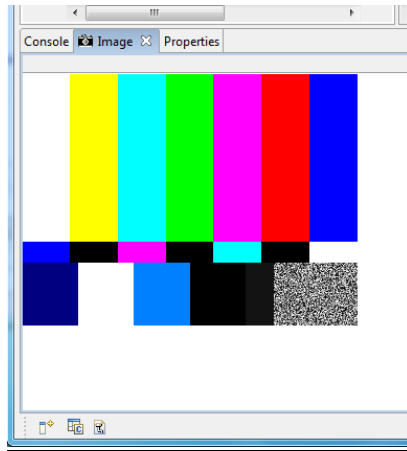


图 2.3.45 图像显示

另外还提供了其他一些调试器功能和视图。强烈建议您尝试一下所有这些选项，了解调试器的所有功能。

第三章 MSP430G2 系列单片机硬件资源应用技术

在这一章中结合 MSP430G2 系列单片机，对 LaunchPad 能够提供的各种硬件资源进行全面介绍，先解析相关的硬件器件，包括组成框图和寄存器，再介绍具体的使用方法和注意事项。

第一节 时钟与休眠模式

3.1.1 时钟系统简介

MSP430G2 系列单片机的时钟系统需要支持系统低功耗运行的需要。通过对三个内部时钟信号的运用，用户可很容易的选择功耗最低，效率最高的系统时钟方案。在软件的控制下，MSP430G2 系列单片机运行时可以不接外接晶振，也可接一只外接电阻或者接一到两只外接晶振，也可以外接频率发生器，如图 3.1.1 所示为 G2 系列的时钟系统。

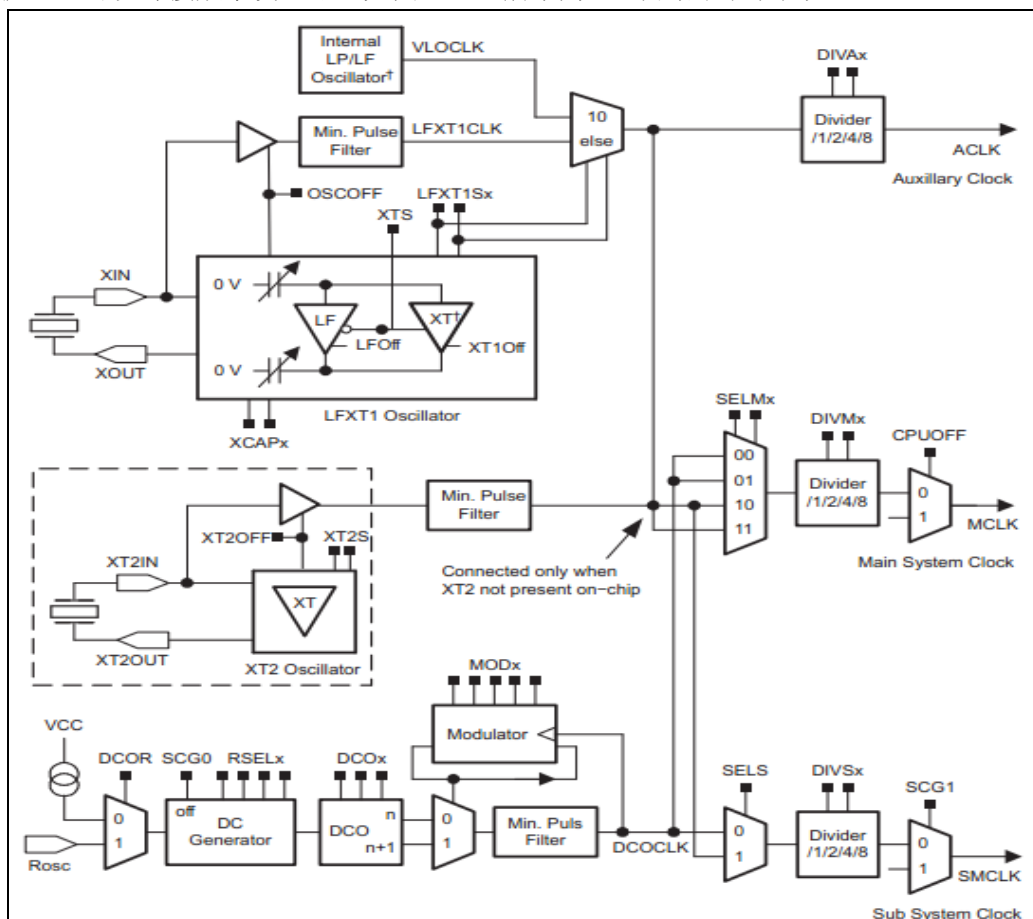


图 3.1.1 时钟模块组成框图

基本时钟模块包括二至四个时钟源：

◆LFX1CLK：外接低频或高频振荡器，如手表晶振，频率发生器，外部时钟源（400kHz 至 16MHz）。

◆XT2CLK：外接高频晶振，范围在 400kHz 至 16MHz

◆DCOCLK: 内部时钟震荡

◆VLOCLK: 内部低频低功耗晶振, 标准频率是 12kHz

三种时钟信号可被 CPU 和外设所使用:

◆ACLK: 辅助时钟, ACLK 可被软件配置成从 LFXT1CLK 或 VLOCLK 输入震荡 ACLK 可以被 1, 2, 4, 8 分频。ACLK 可以被选用作外围模块的时钟输入。

◆MCLK: 主时钟, 主时钟可以从 LFXT1CLK 或 VLOCLK, XT2CLK 或 DCOCLK 输入 MCLK 用在 CPU 系统之中。

◆SMCLK: 辅助主时钟。SMCLK 可被选择从 LFXT1CLK, VLOCLK, XT2CLK 或 DCOCLK 输入。SMCLK 可以被 1, 2, 4, 8 分频。SMCLK 可被选用为外围模块的时钟。

如图 3.1.1 所示为 G2 系列的时钟系统。需要说明的是并非每一款 G2 系列的单片机都具有如上的时钟系统, 以下做详细说明:

MSP430G22x0: 无 LFXT1, 无 XT2, 不支持 ROOSC

MSP430G21x1, MSP430G2xx2, MSP430G2xx3: LFXT1 不支持 HF 模式, 无 XT2, 不支持 ROOSC。

MSP430x21x2: 无 XT2

3.1.2 时钟系统的操作

在一个 PUC 信号之后, MCLK 和 SMCLK 开始从 DCO (大约在 1.1MHz), ACLK 从 LFXT1CLK (内部电容为 6pF) 获取时钟。

状态控制位 SCG0, SCG1, OSCOFF 和 CPUOFF 配置 MSP430 的操作模式并时能或停止基本时钟模块, 控制寄存器 DCOCTL1, BCSCTL1, BCSCTL2 和 BCSCTL3 寄存器配置基本时钟模块。

3.1.3 基本时钟模块特性

低频时钟是为了节省系统能耗实现定时;

高频时钟是为了对事物有很快的响应;

时钟的稳定性取决于温度和供电电压。

以上的需求决定了用户可以选择三种时钟信号进行使用, ACLK, MCLK 和 SMCLK。对于 ACLK 主要使用在一些速度不高的外设, 以实现低功耗; SMCLK 可从 DCO 或外部引入主要是为了满足高速外设的使用需求; MCLK 主要向 CPU 提供时钟。一般情况下 MCLK 频率最高, ACLK 频率最低。

3.1.4 VLO 时钟

内部低频时钟产生器提供一个 12kHz 的振荡频率, 并且不需要外接晶振。通过 XTS=0 和 LFXT1Sx=10. 可启用 VLOCLK。在 LPM4 状态下, OSCOFF 位停止 VLO 时钟。当选择 VLO 发生频率时, 接到 LFXT1 上的晶振会被禁止使用。VLO 如果不用的话将不会被使能。

3.1.5 LFX1 时钟

在 MSP430G22x0 中不含有 LFXT1 系列振荡器, 如图 3.1.2 所示为 LFX1CLK 的逻辑图 LFXT1 振荡器支持极低功耗的手表晶振。手表晶振可以连接到 XIN 和 XOUT 之间且不需要外

接任何电容。我们可以通过软件设置内部电容 XCAP_x 的值，它可以被配置成 1pF, 6pF, 10pF 或 12.5pF。

LFXT1 振荡器也支持高频晶振模式既可外接高频晶振，也可外接高频振荡器。通过控制位 LFXT1S_x=11, OSCOFF=0, XCAP_x=00。若选择为高频模式后，从 LFXT1 输入的频率较低，LFXT10F 位会停止 LFXT1CLK。

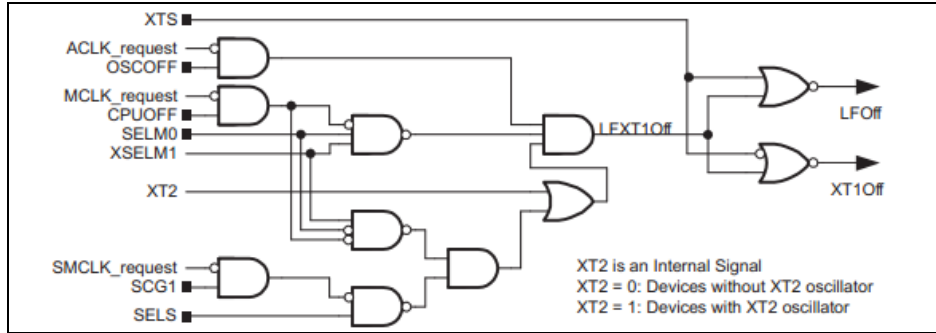


图 3.1.2 LFX1CLK 时钟控制逻辑

3.1.6 XT2 晶振

在 MSP430 系列单片机的一些型号芯片中还有第二个晶振 XT2，XT2 从 XT2CLK 获取振荡信号，它的特性和 LFXT1 的 HF 模式较为相似。XT2S_x 选择 XT2 的频率变化范围，如果不将 XT2CLK 用作 MCLK 和 SMCLK。XT2OFF 位关闭 XT2 振荡器，XT2 的逻辑结构如图 3.1.3 所示。

XT2 可以从 XT2IN 引入，此时需置 XT2S_x=11 和 XT2OFF=0。当用外部信号输入时，时钟频率必须满足 XT2 的需求，否则 CPU 将其关闭。

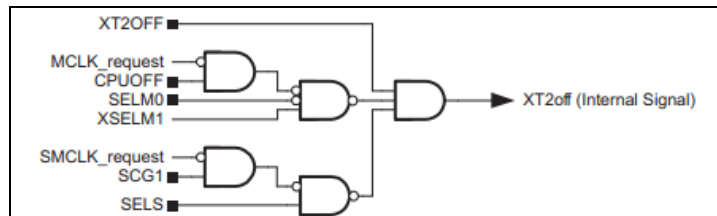


图 3.1.3 XT2 时钟控制逻辑

3.1.7 DCO

DCO 是由 MSP430 内部产生的，可通过 DCO_x, MOD_x 和 RSEL_x 进行调节，关闭 DCO 模块。当 SMCLK 和 MCLK 不需要 DCO 提供频率时，软件可通过置位 SCG0 停止 DCOCLK。

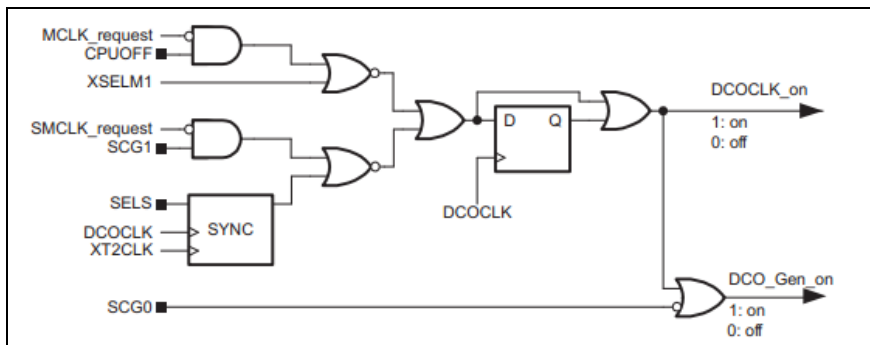


图 3.1.4 DCO 控制逻辑

3.1.8 调整 DCO 的频率

在 PUC 复位信号之后，系统设置 RSEL_x=7 和 DCO_x=3，使 DCO 在一个中等大小的频率上起震。此时，SMCLK 和 MCLK 都从 DCOCLK 获取频率，代码在 PUC 信号后 2μs 之内开始运行，DCOX 和 RSEL_x 的变化和梯度图如图 3.1.5 所示。

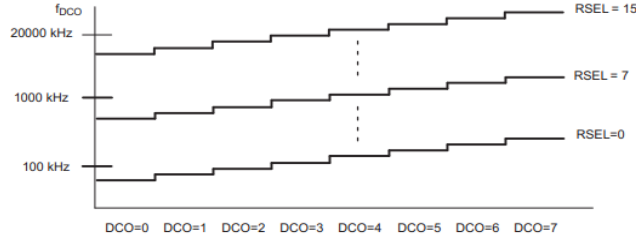


图 3.1.5 DCO_x 和 RSEL_x 所决定的 DCO 频率图

几乎每一款 MSP430G2 系列单片机都有时钟校准系统。可将 DCO 校准成精准的时钟，将校准值储存在内存之中，运行校准指令之后，将内存中校准的数据存入 DCOCTL 和 BCSCTL1 寄存器。

运行以下代码可将 DCO 校准至精准的 1MHz

```
DCOCTL=0;
BCSCTL1=CALBC1_1MHZ;
DCOCTL=CALDCO_1MHZ;
```

3.1.9 运用外部电阻 R_{osc} 调整 DCO 的值

MSP430G2 系列单片机可以通过在 R_{osc} 引脚外接震荡电阻调整振荡频率，如果 DCOR=1，则将其接 DV_{cc}，RSEL_x 可以从 0—7 进行选择。

3.1.10 DCO 调制器

调制器将 f_{DCO} 和 f_{DCO+1} 进行混频，产生一个在 f_{DCO} 和 f_{DCO+1} 之间的频率。调制器在 32 个 DCOCLK 周期之内，对 f_{DCO} 和 f_{DCO+1} 进行混频，当 MOD_x=0 时调制器关闭。

调制器的公式为： $t=(32-MOD_x)*t_{DCO}+MOD_x*t_{DCO+1}$

由于 f_{DCO} 的频率低于已知的频率，而 f_{DCO+1} 高于已知的频率，经混频后频率误差接近于 0 且不会累加。

混频器的设置和 DCO 的控制可以通过软件进行设置。

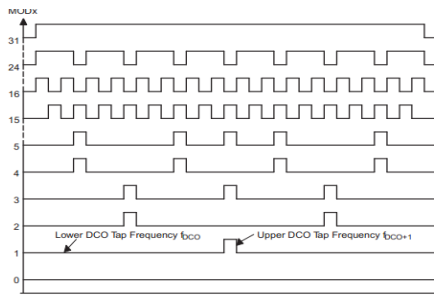


图 3.1.6 DCO 混频时序图

3.1.11 基本时钟系统的错误检测

时钟系统可检测来自外接晶振错误，错误分为以下三类

- ◆LF 模式下的低频振荡错误
- ◆HF 模式下的 LFXT1 错误
- ◆XT2 错误

当发生错误时，时钟发生器错误位 LFXT1OF 和 XT2OF 会被置位，如果错误不被恢复，则错误发生位会一直被置位。

当 LFXT1OF 或 XT2OF 置位后，OFIFG 位会被置位并引起 POR 复位。当 OFIFG 置位后，MCLK 会从 DCO 输入时钟，OFIE 是时钟错误允许中断，若发生错误且 OFIE 置位，则会产生一次 NMI 中断，当中断产生以后，OFIE 会被自动复位。OFIFG 位必须通过软件清除。

主频时钟发生错误后，会从 DCO 获取频率，但 SELM 的值不会发生变化。

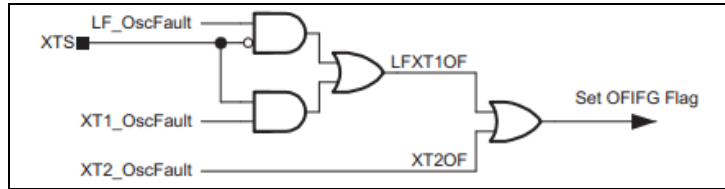


图 3.1.7 时钟错误控制逻辑

3.1.12 从外部晶振获取 MCLK

在 PUC 复位信号之后，基本时钟模块使用 DCOCLK 作为 MCLK。如果需要更高的频率可以选用 LFXT1 或 XT2。

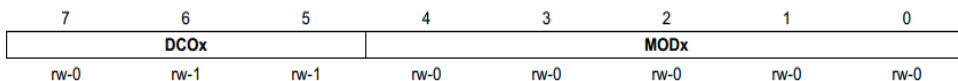
- 从 DCO 切换到外部晶振的步骤如下：
- 开启外部晶振模式选择合适的模式；
- 清除 OFIFG 位；
- 等待 50us；
- 测试 OFIFG 位是否被复位。

3.1.13 基本寄存器表

表 3.1 寄存器名称地址表

Register	Short Form	Register Type	Address	Initial State
DCO control register	DCOCTL	Read/write	056h	060h with PUC
Basic clock system control 1	BCSCTL1	Read/write	057h	087h with POR ⁽¹⁾
Basic clock system control 2	BCSCTL2	Read/write	058h	Reset with PUC
Basic clock system control 3	BCSCTL3	Read/write	053h	005h with PUC ⁽²⁾
SFR interrupt enable register 1	IE1	Read/write	000h	Reset with PUC
SFR interrupt flag register 1	IFG1	Read/write	002h	Reset with PUC

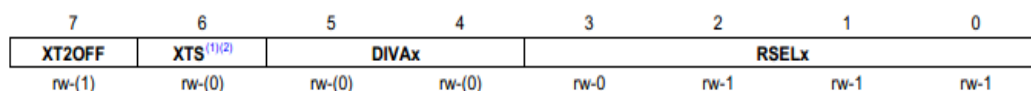
DCOCTL: DCOCTL 寄存器，它分为两部分：DCOx 和 MODx



DCOx: DCO 频率选择，这些位可以在由 RSELx 决定的八个离散的 DCO 频率中选择。

MODx: 调制系数，这个系数决定在 32 个周期中 fDCO+1 占多少，fDCO 占多少。

BCSCTL1: 基本时钟系统控制寄存器



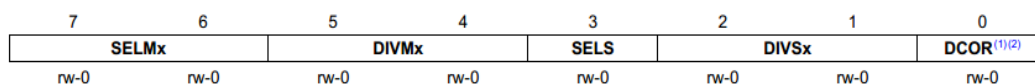
XT2OFF: 0 XT2 开启
1 XT2 如果不被用作 MCLK 和 SMCLK 则关闭

XTS: 0 低频模式
1 高频模式

DIVAx: ACLK 分频控制
00 /1
01 /2
10 /4
11 /8

RSELx: DCO 频率范围选择，RSELx=0 时频率最低，当 DCOR=1 时 RSEL3 被忽略。

BCSCTL2 寄存器:



SELMx: MCLK 频率选择
00 DCOCLK
01 DCOCLK
10 XT2 如果出现在片上则是 XT2CLK，当 XT2 不出现在片上则选择 LFXT1CLK 或 VLOCLK
11 LFXT1CLK 或 VLOCLK

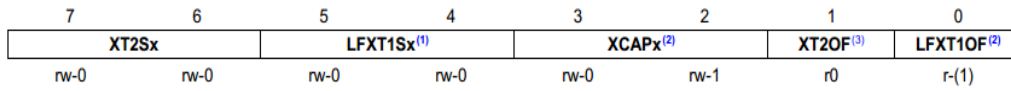
DIVM: MCLK 分频
00 /1
01 /2
10 /4
11 /8

SELS: SMCLK 频率选择
0 DCOCLK
1 片上若有 XT2 则选择 XT2CLK，若没有 XT2 则选择 VLOCLK 或 XT2

DIVSx: SMCLK 的分频
00 /1
01 /2
10 /4

11 /8
DCOR: DCO 震荡电阻选择
 0 内部电阻
 1 外部电阻

BCSCTL3 寄存器:



XTSx: XT2 范围选择
 00 0.4~1MHz 晶振或振荡器
 01 1~3MHz 晶振或振荡器
 10 3~16MHz 晶振或振荡器
 11 外部 0.4~16MHz 振荡器

LFXT1Sx: 低频时钟选择和 LFXT1 范围选择。当 XTS=0 时这些位在 LFXT1 和 VLO 之选择，当 XTS=1 时选择 LFXT1 的频率范围
 XTS=0 时

00 LFXT1 的 32768Hz 晶振
 01 保留
 10 VLOCLK (仅在 MSP430F21x1 中有)
 11 外部数字时钟源

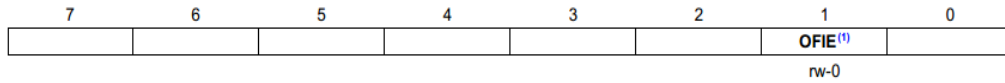
XTS=1 时
 00 0.4MHz~1MHz 晶振或振荡器
 01 1MHz~3MHz 晶振或振荡器
 10 3~16MHz 晶振或振荡器
 11 0.4~16MHz 振荡器

XCAPx: 内部电容选择
 00 ~1pF
 01 ~6pF
 10 ~10pF
 11 ~12.5pF

XT20F: XT2 错误标示位
 0 无错误
 1 有错误

LFXT10F: LFXT1 错误
 0 无错误
 1 有错误

IE1: 中断允许寄存器 1

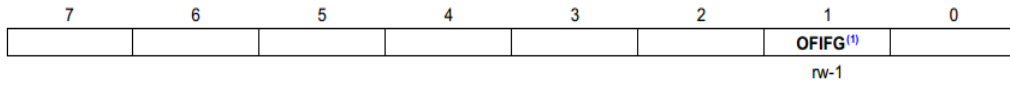


OFIE: 振荡器错误中断允许

0 中断不允许

1 中断允许

IFG1: 中断标志寄存器 1



OFIFG: 0 无中断申请

1 晶振错误中断申请

第二节 通用 I/O 口

3.2.1 I/O 口

I/O 口是微处理器系统对外界沟通的最基本部件，从基本的键盘、LED 到复杂的外设芯片等，都是通过 I/O 口的输入输出操作来进行的。

在 MSP430 系列中，不同的单片机 I/O 口数量不同。体积最小的 MSP430F20xx 系列中只有 10 个 I/O 口，适合在超小型设备中应用；功能最丰富的 MSP430FG46xx 系列中多达 80 多个 I/O 口，足够应付外部设备繁多的复杂应用。在 MSP430G2553 单片机中，共有 16 个 I/O 口，属于 I/O 口较少的系列。

3.2.2 I/O 寄存器

和大部分单片机类似，MSP430 系列单片机也是将 8 个 I/O 口编为一组。每个 I/O 口有四个控制寄存器，P1 和 P2 还有额外的 3 个中断寄存器。寄存器情况可参见表 3.2。

表 3.2 I/O 寄存器表

寄存器名	寄存器功能	读写类型	复位初始值
PxIN	Px 口输入寄存器	只读	无
PxOUT	Px 口输出寄存器	可读可写	保持不变
PxDIR	Px 口方向寄存器	可读可写	0（全部输入）
PxSEL	Px 口第二功能选择	可读可写	0（全部为 I/O 口）
PxIE	Px 口中断允许	可读可写	0（不允许中断）
PxIES	Px 口中断沿选择	可读可写	保持不变
PxIFG	Px 口中断标示位	可读可写	0（未发生中断）

PxDIR 寄存器用于设置每一位的 I/O 口方向

MSP430 单片机的 I/O 口是双向 I/O 口：0=输入 1=输出。在使用 I/O 口时首先要选择寄存器来设置每个 I/O 口方向。

例如下面的语句：

```
P1DIR |= BIT1+BIT3+BIT4;
```

```
P1DIR |= ~(BIT5+BIT6+BIT7);
```

以上语句将 P1.1, P1.3 和 P1.4 的方向置为输出，P1.5, P1.6 和 P1.7 的方向置为输入。

PxDIR 寄存器在复位的过程中会被清零，没有被设置的 I/O 口方向均为输入状态，所以第二句可以省略。注意，将未用的 I/O 口置为输出可减小漏电流。

对于所有已经设成输出的 I/O 口可通过 PxOUT 寄存器设置其输出电平；对于所有已经被设成输入的 I/O 口，可通过 PxIN 寄存器读回其输入电平。

PxSEL 寄存器用于设置每一位 I/O 口的功能：0=普通 I/O 口， 1=第二功能。

在 MSP430 系列单片机中，很多内部功能模块也需要和外界进行数据交换，为了不增加芯片的管脚数量，大部分都和 I/O 口管脚复用，这就导致 MSP430 系列单片机的大多数 I/O 管

脚都具有第二功能。通过 PxSEL 可以指定某些 IO 口作为第二功能使用。如 MSP430G2553 中 P1.1 和 RXD, P1.2 和 TXD 复用。

3.2.3 IO 口中断

在 MSP430 系列所有的单片机中, P1 口、P2 口总共 16 个 IO 口均能引发中断。

PxIE 用于设置每一位 IO 口的中断允许: 0=不允许 1=允许

PxIES 用于选择每一个 IO 口的中断触发沿: 0=上升沿 1=下降沿

在使用 IO 口中断之前, 需要先将 IO 口设置为输入状态, 并允许该位 IO 中断, 再通过 PxIES 寄存器选择上升沿触发还是下降沿触发。如下例, 将 P1.5, P1.6 和 P1.7 设置为中断源, 下降沿触发。

```
P1DIR&=~(BIT5+BIT6+BIT7);
```

```
P1IES |=BIT5+BIT6+BIT7;
```

```
P1IE |=BIT5+BIT6+BIT7;
```

```
_EINT();
```

PxIFG 是 IO 口中断标志寄存器: 0=中断标志不成立, 1=中断条件曾经成立

无论中断是否被允许, 也无论是否可以执行中断服务程序, 只要对应 IO 口满足中断条件, PxIFG 中相应位都会立即置 1 并保持, 只能通过软件人工清除。这种机制的目的在于最大可能的保证不会漏掉每一次中断。在 MSP430 系列单片机中, P1 口的 8 个中断各共用了一个中断入口, 因此该寄存器另一重要作用在于判断是哪一位 IO 产生了中断。

注意, 在退出中断前, 一定要人工清除中断标志, 否则该中断会不停被执行。类似的原理, 即使 IO 口没有出现中断条件, 人工向 PxIFG 寄存器写“1”, 也会引发中断。更改中断沿选择寄存器也相当于跳变, 也会引发中断。所以, 更改 PxIES 寄存器应该在关闭中断后进行, 并且在打开中断后及时清除中断标示位。

3.2.4 线与逻辑

MSP430 系列单片机的 IO 口是 CMOS 型, 特点是当 IO 处于输入状态时时, 呈高阻状态; 当 IO 处于输出状态时, 高低电平都具有较强的输出能力。若输出高电平的 IO 口和输出低电平的 IO 口直接相连, 则会因短路造成损坏, 不像 8051 的 IO 那样能实现“线与”功能。但可以通过 IO 方向的切换来模拟。以 P1.0 口为例, 硬件上加一个上拉电阻至 Vcc, 软件中先将 P1OUT 的 BIT0 置 0, 再通过软件切换方向来改变输出。

```
#define IO_H P1DIR &=~BIT0
```

```
#define IO_L P1DIR |=BIT0
```

```
#define IO_R (P1IN&BIT0)
```

用这种方法来模拟 IO 操作, 高电平不是由 IO 直接输出的, 而是通过上拉电阻拉高的, 因此这种高电平的电流输出能力很弱; 而低电平由 IO 直接输出, 驱动能力较强。当高电平遇到低电平的时候会被拉低。当若干 IO 连在一起时, 只要有 1 根输出 0, 整体就输出 0, 总输出相当于各 IO 相与。“线与”逻辑在 I2C 总线、多机通信中都有重要用途。

3.2.5 兼容性

为了电池供电应用，MSP430 系列单片机工作电压较低（1.8V~3.6V）。大部分应用取 3V 左右，因此单片机的 I/O 口属于 3V 逻辑。且 MSP430 单片机的任何一个管脚的输入电压不能超过 $V_{cc}+0.3V$ ，不能低于 $-0.3V$ ，否则将启动内部泄放电路。泄放电路最大值只能吸收 2mA 的电流，超过可能毁损 I/O 口。在 MSP430 单片机与 5V 逻辑连接时，必须考虑电平转换问题，分以下几种情况进行讨论：

1. 5V 逻辑器件输出至 MSP430 单片机，这是一种最简单的情况，将 5V 逻辑通过 10K 和 20K 电阻分压后转换成 3V 逻辑。若 5V 逻辑属于弱上拉型，也可以直接连接，利用 MSP430 单片机内部泄放电路将电压钳至 3V。当然，最为可靠的方法还是使用 74LVC 系列缓冲器，如 74LVC244 等，它可以 3V 供电，并具有 5V 的输入承受能力。

MSP430 单片机输出至 5V 逻辑器件输入，这种情况首先要看接收器件的高电平门限，一般的接收芯片或设备手册都会给出。某些器件具有 2.5V 以下的门限可直接连接无需额外电路。若接收方门限较高，可在两者之间加一片 74HCT24 缓冲器。74HCT 器件具有 2.2V 的固定逻辑门限，在 5V 电源时能够识别 3V 的逻辑输入。

双向数据传输中，不仅要转换电平，还需要切换方向，最好选用电平转换芯片 74LVC4245 实现。

驱动 5V 以上的逻辑电路，利用漏级开路门电路可以实现逻辑电平的转换。

无论出现何种转换方案，3V 逻辑和 5V 逻辑电路中都是不值得推荐的。这不仅破坏了 MSP430 系统的简洁设计原则，还额外增加了功耗，增加了电源管理难度。所以在设计 MSP430 系统时应尽量使用 3V 逻辑。

和所有 CMOS 电路一样，MSP430 单片机的 I/O 口输入状态也呈高阻态。若悬空，则等效于天线，会因附近电场而随机的感应出中间电平。MSP430 单片机内部带有施密特触发器和总线保持器，悬空或输入中间电平不会造成错误或损坏，但会因为输入级 CMOS 门截止不良，额外增加系统耗电。所以在超低功耗应用中每个 I/O 口都应具有确定电平，对于未用的 I/O 口，可接地或设置为输出状态，以保证电平确定。

第三节 10 位 ADC

MSP430 系列单片机内部集成了 ADC，这为工程师在设计硬件电路时提供了很大的方便。同时，不同的单片机中集成了不同类型的 ADC，有精度高但速度慢的 SD16，有适用于多通道采集的 ADC12，也有适用于高速度采集的 ADC10。在 MSP430G2 系列单片机内部通常集成的是 10 位 ADC。

3.2.1 ADC10 的特点

ADC10 是 MSP430 单片机的片上模数转换器，其转换位数为 10 比特，该模块内部是一个 SAR 型的 AD 内核，可以在片内产生参考电压，并且具有数据传输控制器。数据传输控制器能够在 CPU 不参与的情况下，完成 AD 数据向内存任意位置的传输。它具有如下特点：

- 最大转换速率大于 200ksps
- 转换精度为 10 位
- 采样保持器的采样周期可通过编程设置
- 可利用软件或者 TimerA 设置转换初始化
- 编程选择片上电压参考源，选择 1.5V 或者 2.5V
- 编程选择内部或者外部电压参考源
- 8 个外部输入通道
- 具备对内部温度传感器、供电电压 VCC 和外部参考源的转换通道
- 转换时钟源可选择
- 多种采样模式：单通道采样、序列通道采样、单通道重复采样、序列通道重复采样
- 提供自动数据传输方法
- ADC 的内核和参考源可分别单独关闭

1. ADC10 的结构与原理

图 3.3.1 是 ADC10 模块的结构框图。从图中可以看出 ADC10 有 16 个采样通道 A0~A15，其中 A0 到 A7 是外部采样通道，A10 是对内部温度传感器的采样通道。

ADC10 模块工作的核心是 ADC10 的核，即图中的 10-bit SAR。ADC10 的核将模拟量转换成 10 位数字量并储存在 ADC10MEM 寄存器里。这个核使用 VR+和 VR-来决定转换模拟值的高低门限。当输入电压超过 VR+时它会停在 03FFh 上，当输入门限低于 VR-时它会停在 0 上。采样值的计算公式为：

$$NADC=1023* (VIN-VR-) / (VR+-VR-)$$

下面将对 ADC10 模块的配置和操作进行介绍。

(1) 转换时钟选择

ADC10CLK 需要可作为转换时钟也可作为产生采样周期的时钟，ADC10CLK 可通过 ADC10SSELx 位进行选择，通过 ADC10DIVx 进行分频。可供选择的 ADC10CLK 时钟源是 SMCLK，MCLK，ACLK 或者是内部晶振 ADC100SC。ADC100SC 最大可达到 5MHZ，但是会根据具体片子的不同而有所差别，详见数据手册。用户必须确保 ADC10CLK 的时钟处于开启状态，否则转换

将无法开始。

```
ADC10CTL1 |= ADC10SSEL_3+ADC10DIV_0;//时钟源选择 SMCLK, 1 分频
```

```
ADC10CTL1 |= ADC10SSEL_1+ADC10DIV_7;//时钟源选择 ACLK, 8 分频
```

(2) ADC10 的输入和复用

八个外部和四个内部模拟信号可被选择为输入的通道，他们通过复用器共同作为转换器的前端。不被选中的通道将被与转换核心剥离。当模拟信号输入到 ADC 的 CMOS 输入门时寄生电流会从 VCC 到地变化。禁止端口输入端缓冲可以减少寄生电流。8 位的 ADC10AE 寄存器可以对对应开启或关闭 8 个外部采样通道。

```
ADC10AEO |= 0x15; //开启外部通道 0, 通道 2 和通道 4
```

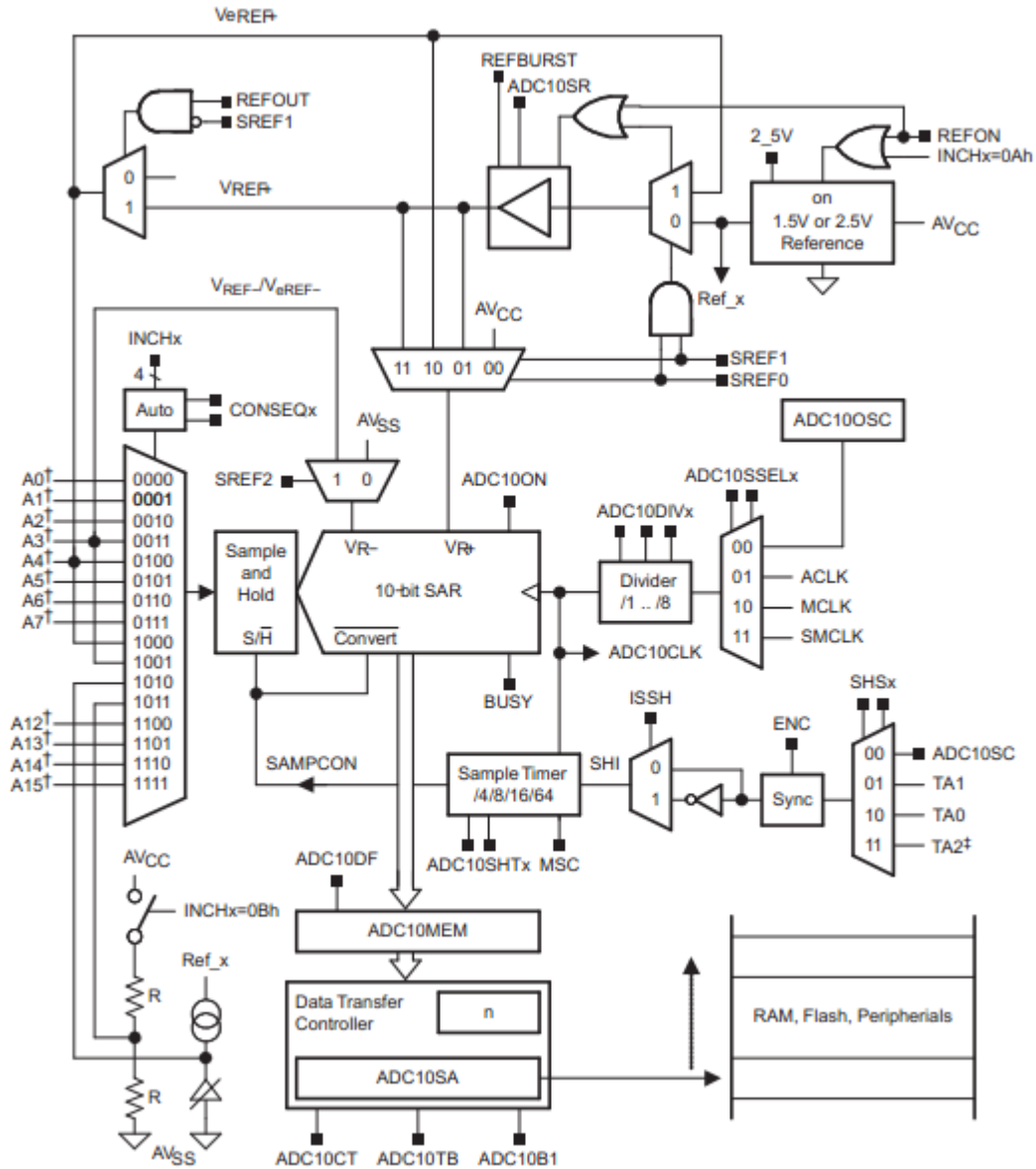


图 3.3.1 ADC10 模块的结构框图

(3) 电压参考产生

寄存器 SREFx 可选择 ADC10 模块的电压参考源。ADC10 模块含有内部电压参考源。使用

内部参考源时,令 SREF_x=001(B),则 ADC10 模块电压参考源选择内部电压参考,同时将 REFON 置 1 使能内部参考源。当 REF2_5V=1 时,内部参考源电压是 2.5V。当 REF2_5V=0 时,参考值是 1.5V,如果将 REFOUT=1 可向外输出参考源电压。

ADC10CTL0 |= SREF_1+REFON+REF2_5V; //选择并使能内部参考源,电压 2.5V

(4) 采样和转换时间

ADC10 转换可以被 SHI 信号的上升沿所触发,SHI 信号可以被 SHS_x 位所选择为:ADC10SC 位、TIMER_A.OUT1、TIMER_A. OUT0 和 TIMER_A. OUT2。

采样延时可通过 SHT_x 位进行选择,时间可以是 4,8,16,64 个 ADC10CLK 周期。当 SAMPCON 置高时,采样定时器开始计时采样时间等于 SAMPCON 由高到低的时间,采样完后开始转化,大约需要 13 个 ADC10CLK 的时间进行转化。如图 3.3.2 所示

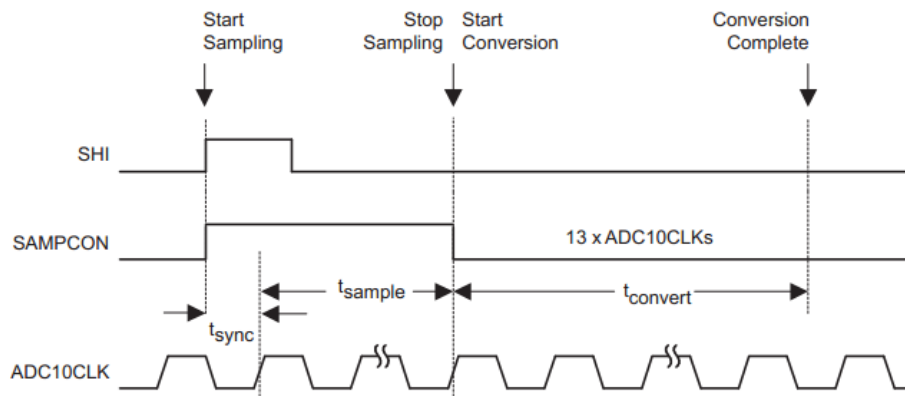


图 3.3.2 ADC10 采样和转换时间

3.2.2 ADC10 转换模式

ADC10 有四个操作模式可以通过一个两位的寄存器 CONSEQ_x 被选择。具体情况如下表:

表 3.3 ADC10 的转换模式选择

CONSEQ _x	模式	操作
00	单通道单次采样	一个通道被采样转换一次
01	序列通道采样	多个通道被依次采样转换
10	单通道重复采样	一个通道被多次采样转换
11	序列通道重复采样	多个通道被重复采样转换

1. 单通道单次采样模式

单个被 INCH_x 所选中的通道 x 被采样并转换一次。ADC 结果被写入 ADC10MEM 寄存器。如图 3.3.3 所示为单通道单次转换状态转移图。可用 ADC10SC 触发一次转换,当切换至其他触发源时,在两次转换之间,ENC 需要被切换。

在采样模式 0 下,首先令 ADC100N=1 即开启 ADC10 模块,接着确定采样通道 x,然后等待触发。当 SHS=0, ENC=1 时,ADC10SC 可触发采样开始。或者用 TIMERA 进行触发采样,当 ENC 的上升沿到来之后,它将等待 TIMERA 所产生的 PWM 波进行触发。ADC10 采样完成后,经过 12 个 ADC10CLK 的时钟周期进行采样结果转换,再经过 1 个 ADC10CLK 的时钟周期,ADC10

将转换后的结果存入寄存器 ADC10MEM，同时 ADC10 的中断标志位 ADC10IFG 被置 1。在转换过程中的任意时刻，如果将 ENC 置 0 则会关闭 ADC10 模块。

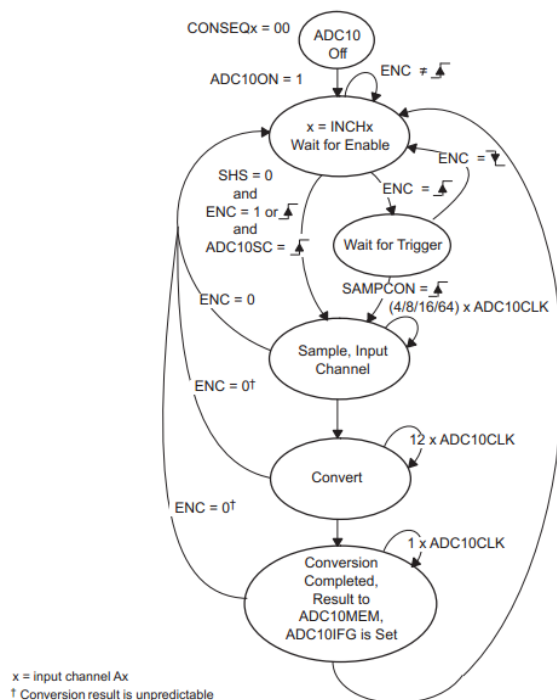


图 3.3.3 单通道单次采样状态转移图

2. 序列通道采样模式

如图 3.3.4 序列通道采样状态转移图所示，ADC10 在这种工作模式下，INCHx 中值意义和在单通道单次采样模式下不同，INCHx 写入的通道时序列通道中的最高通道，采样开始后采样通道从选中的通道依次转换到通道 0，并且只完成一次序列通道采样。每一个 ADC 采样结果都会被存入 ADC10MEM 寄存器。当转换到通道零时，转换序列会停止。可以使用 ADC10SC 来触发一个序列的转换。当其他的触发源被使用时，ENC 必须被复位。

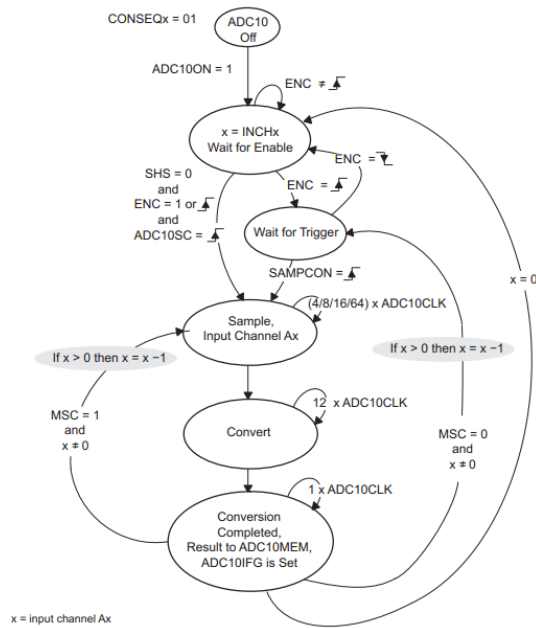


图 3.3.4 序列通道采样状态转移图

3. 单通道重复采样模式

在这种模式下，一个被 INCH_x 可以被选中通道 x 被多次采样转换。每次转换结果，都将被存入 ADC10MEM 寄存器。若旧的值未被读取，则新的值将被旧的值所覆盖。

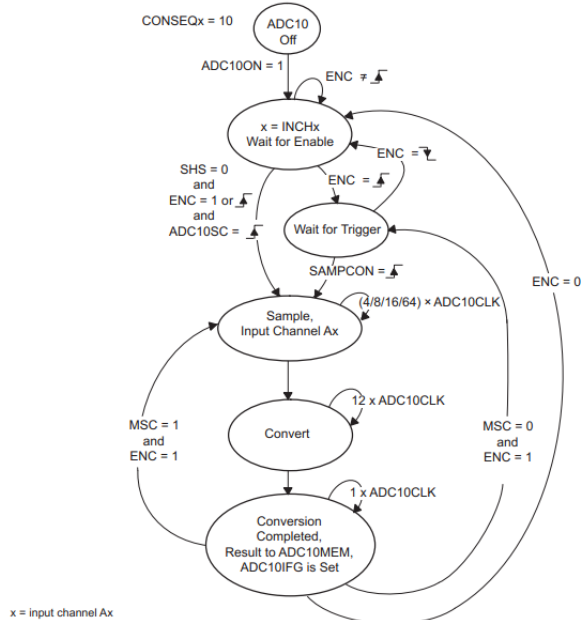


图 3.3.5 单通道重复采样状态转移图

4. 序列通道重复采样模式

这种模式和序列通道采样模式基本相同，采样通道从 INCH_x 所选中的通道开始依次转换到通道 0，然而在这种工作模式下 ADC10 可进行重复序列通道采样。ADC10 每次转换结果都会被写入 ADC10MEM 寄存器的值。每次采样完成之后，会自动将通道值重新置入 INCH_x。

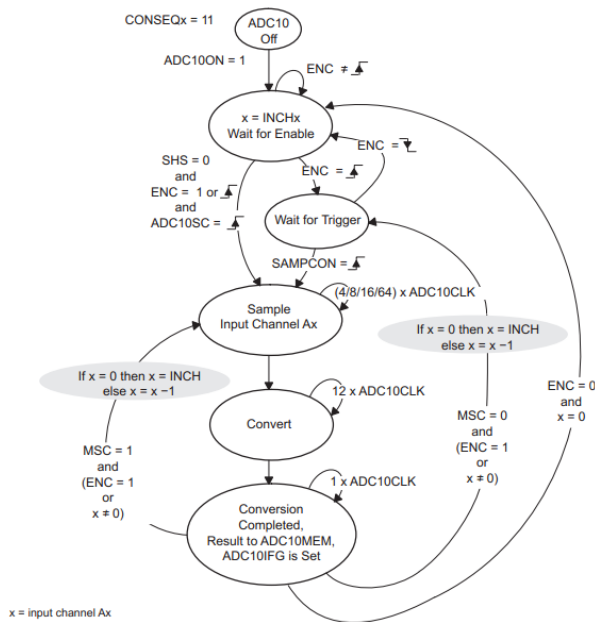


图 3.3.6 序列道重复采样状态转移

3.3.3 ADC10 寄存器

ADC10 的寄存器如表 2 所示。

表 3.4 ADC10 寄存器

寄存器	形式	寄存器类型	地址	初始状态
ADC10 输入使能寄存器 0	ADC10AE0	可读写	04Ah	上电复位
ADC10 输入使能寄存器 1	ADC10AE1	可读写	04Bh	上电复位
ADC10 控制寄存器 0	ADC10CTL0	可读写	01B0h	上电复位
ADC10 控制寄存器 1	ADC10CTL1	可读写	01B2h	上电复位
ADC10 数据传输控制寄存器 0	ADC10DTC0	可读写	048h	上电复位
ADC10 数据传输控制寄存器 1	ADC10DTC1	可读写	049h	上电复位
ADC10 数据传输起始地址	ADC10SA	可读写	01BCh	上电后为 0200h
ADC10 内存	ADC10MEM	只读	01B4h	无变化

下面对常用的部分寄存器进行介绍。

1. ADC10CTL0

ADC10CTL0 寄存器结构如图 3.3.7 所示。只有 ENC=0 时 ADC10CTL0 的内容才能被修改。

15	14	13	12	11	10	9	8
SREFx		ADC10SHTx			ADC10SR	REFOUT	REFBURST
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
MSC	REF2_5V	REFON	ADC10ON	ADC10IE	ADC10IFG	ENC	ADC10SC
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

图 3.3.7 ADC10CTL0 寄存器结构

SREFx: 基准源选择位

000	$V_{R+}=V_{CC}$	$V_{R-}=V_{SS}$
001	$V_{R+}=V_{REF+}$	$V_{R-}=V_{SS}$
010	$V_{R+}=V_{REF+}$	$V_{R-}=V_{SS}$
011	$V_{R+}=BufferedV_{REF+}$	$V_{R-}=V_{SS}$

	100	$V_{R+}=V_{CC}$	$V_{R-}=V_{REF-}/V_{eREF-}$
	101	$V_{R+}=V_{REF+}$	$V_{R-}=V_{REF-}/V_{eREF-}$
	110	$V_{R+}=V_{eREF+}$	$V_{R-}=V_{REF-}/V_{eREF-}$
	111	$V_{R+}=BufferedV_{eREF+}$	$V_{R-}=V_{REF-}/V_{eREF-}$
ADC10SHTx:		ADC10 采样和保持时间设置位	
	00	4 个 ADC10CLK 周期	
	01	8 个 ADC10CLK 周期	
	10	16 个 ADC10CLK 周期	
	11	64 个 ADC10CLK 周期	
ADC10SR:		ADC10 采样率设置位。该位的选择参考缓冲器驱动能力的最大采样率。设置 ADC10SR 可降低基准缓冲器的电流消耗。	
	0	参考缓冲器支持高达 200 ksp/s 的采样速度	
	1	参考缓冲器支持高达 50 ksp/s 的采样速度	
REFOUT:		参考源输出控制位	
	0	参考源输出关闭	
	1	参考源输出开启	
REFBURST:		可利用此位降低功耗	
	0	参考源一直开启	
	1	只有在采样和转换时开启参考源	
MSC:		多重采样和转换。这一位只用于序列或重复采样模式。	
	0	SHI 信号的上升沿触发每个采样和转换。	
	1	SHI 信号的第一个上升沿触发采样定时器，但是进一步的采样和转换只有在前一次的转换完成时才进行。	
REF2_5V:		参考源电压选择位，更改时 REFON 必须开启	
	0	参考源电压 1.5V	
	1	参考源电压 2.5V	
REFON:		参考源开关	
	0	参考源关闭	
	1	参考源开启	
ADC100N:		ADC10 模块开关	
	0	ADC10 关闭	
	1	ADC10 开启	
ADC10IE:		ADC10 模块使能开关	
	0	禁止 ADC10 中断	
	1	允许 ADC10 中断	
ADC10IFG:		中断标志位。采样转换完成后，ADC10IFG 被置为 1。单片机响应 ADC10 中断以后，ADC10IFG 自动被置位 0，也可软件置 0。	
	0	无中断请求	

	1	有中断请求
ENC:	采样使能开关	
	0	ADC10 使能关闭
	1	ADC10 使能开启
ADC10SC:	采样开始	
	0	停止采样
	1	开始采样

2. ADC10CTL1

ADC10CTL1 寄存器结构如图 3.3.8 所示。只有 ENC=0 时 ADC10CTL1 的内容才能被修改。

15	14	13	12	11	10	9	8
INCHx				SHSx		ADC10DF	ISSH
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ADC10DIVx			ADC10SELx		CONSEQx		ADC10BUSY
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r-0

图 3.3.7 ADC10CTL0 寄存器结构

INCHx:	通道选择
0000	A0
0001	A1
0010	A2
0011	A3
0100	A4
0101	A5
0110	A6
0111	A7
1000	V_{REF+}
1001	V_{REF-}/V_{REF-}
1010	内部温度传感器
1011	$(V_{CC}-V_{SS})/2$
1100	$(V_{CC}-V_{SS})/2$, MSP430F22xx 系列单片机才有此通道
1101	$(V_{CC}-V_{SS})/2$, MSP430F22xx 系列单片机才有此通道
1110	$(V_{CC}-V_{SS})/2$, MSP430F22xx 系列单片机才有此通道
1111	$(V_{CC}-V_{SS})/2$, MSP430F22xx 系列单片机才有此通道
SHSx:	采样保持源选择
00	ADC10SC 位
01	Timer_A.OUT1
10	Timer_A.OUT0
11	Timer_A.OUT2
ADC10DF:	数据格式
0	二进制数

	1	二进制数的补码
ISSH:	输入采样信号反转	
	0	输入的采样信号不反转
	1	输入的采样信号反转
ADC10DIVx:	ADC10 时钟分频选择	
	000	1 分频
	001	2 分频
	010	3 分频
	011	4 分频
	100	5 分频
	101	6 分频
	110	7 分频
	111	8 分频
ADC10SSELx:	ADC10 时钟源选择	
	00	ADC100SC
	01	ACLK
	10	MCLK
	11	SMCLK
CONSEQx:	ADC10 工作模式选着	
	00	单通道单次采样模式
	01	序列通道采样模式
	10	单通道重复采样模式
	11	序列通道重复采样模式
ADC10BUSY:	ADC10 模块忙标志	
	0	ADC10 模块无操作
	1	ADC10 模块正在进行采样或转换

3. ADC10AEO

ADC10 输入使能寄存器有 ADC10AEO 和 ADC10AE1 两个，ADC10AEO 对应 8 个外部通道，ADC10AE1 可对应使能 A12 到 A15，ADC10AE1 只存在于 MSP430F22x 中，这里不作介绍。ADC10AEO 寄存器结构如下图所示：

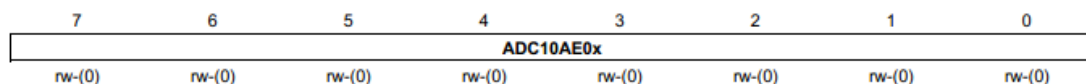


图 3.3.8 ADC10AEO 寄存器结构

从图中可知 ADC10AEO 有 8 位，从低位到高位分别对应使能通道 A0 到 A7，只要对应的位置 1，则对应的通道被使能。

下面给出两个 ADC10 寄存器配置的例子。

例 1: 用单通道重复采样模式对 A1 采样，将采样结果 20 次求平均，然后将结果存在变量 ADC10_Result 内。假设选择 2 分频后的 ACLK 做采样时钟，选择内部参考源 2.5V，ADC10SC

触发采样，禁止 ADC10 中断，采样时间设置为 64 个采样时钟。

```
int ADC10_Result;
ADC10CTL1 |= CONSEQ_2;           //单通道重复采样模式
ADC10CTL0 |= SREF_1+REFON+REF2_5V;//选择内部参考源 2.5V，打开基准源
ADC10CTL0 |= ADC10SHT_3+MSC;    //过采样率设置为 64 个采样周期，打开 AD 转换
ADC10CTL1|= ADC10SSEL_1+ADC10DIV_1+SHS_0; //ACLK2 分频为采样时钟，用
                                   ADC10SC 触发采集
ADC10CTL1 |=INCH_1;             //选择通道 A1
ADC10CTL0 |= ADC10ON;           //开启 ADC10
ADC10AE0 |= 0x02;               //开启外部通道 A1
for(int i=0;i<20;i++)
{
    ADC10CTL0 |=ENC+ADC10SC;     //开始转换
    while((ADC10CTL0 &ADC10IFG)==0); //等待 ADC10IFG 标志变高（转换完成）
    ADC10_Result+=ADC10MEM;      //读取采样结果
}
ADC10_Result= ADC10_Result/20;
```

例 2：用序列通道采样模式对通道 A1、A5、A7 采样，将采样结果在中断中读出，并将 A1、A5、A7 采样的结果分别存入 ADC10_A1[]、ADC10_A5[]和 ADC10_A7[]数组中。选择 SMCLK 无分频作为采样时钟，选择内部参考源 1.5V，ADC10SC 触发采样，采样时间设置为 16 个采样时钟周期。

```
unsigned int u=7;
unsigned int ADC_timers =0;
unsigned int ADC_Result[8];
unsigned int ADC10_A1[20];
unsigned int ADC10_A5[20];
unsigned int ADC10_A7[20];
void main ()
{
    WDTCTL=WDTPW+WDTHOLD;       //关闭看门狗
    ADC10CTL1 |= CONSEQ_1;
    ADC10CTL0 |= SREF_1+REFON+ADC10IE;//选择内部参考源 1.5V，开启 AD 允许中断
    ADC10CTL0 |= ADC10SHT_2+MSC; //打开 AD 转换，过采样率设置为 16 个采样
                                   周期
    ADC10CTL1 |= ADC10SSEL_3+SHS_0; //选择 SMCLK 无分频作为采样时钟，ADC10SC
                                   触发采集
    ADC10CTL1 |=INCH_7;         //最高通道设为通道 7
    ADC10CTL0 |= ADC10ON;       //打开 ADC10 模块
```

```

ADC10AE0 |= 0x1;           //开启通道 A1、A5、A7
__bis_SR_register(GIE);    //开总中断
while(1)
{
    ADC10CTL0 |=ENC+ADC10SC; //开始转换
    for(int i=0;i<10000;i++); //加入延时等待 ADC10 一轮序列通道采样
                                完成
}
}
/*****
* 名 称:
* 功 能: AD 采样中断, 没完成一次采样中断一次, 将采样值存入数组 AD_Result[10]
        然后将想要的采样值读出
*****/
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR (void)
{
    ADC_Result[u]=ADC10MEM;
    u--;
    if(u>8)
    {
        u=7;
        ADC10_A7[AD_timers]=ADC_Result[7];
        ADC10_A5[AD_timers]=ADC_Result[5];
        ADC10_A1[AD_timers]=ADC_Result[1];
        ADC_timers++;
    }
}
}

```

3.3.4 内部温度传感器

ADC10 内部集成了一个温度传感器, 使用此温度传感器时, 只要对 ADC10 的通道 10 进行采样, 就可的出此温度传感器的输出值。图 3.3.9 是 ADC10 内部温度传感器输出电压和温度的理想线性关系。但是按照图 3.3.9 中的线性关系计算得到的温度值会有很大误差, 实际使用时要对其进行校准, 让单片机在一个基准温度 T_0 下, 采样计算的此时温度传感器输出电压值 V_0 , 以此点为基准进行校准, 温度 $TEMP_c = (V_{TEMP} - V_0) / 0.00355 + T_0$ 。

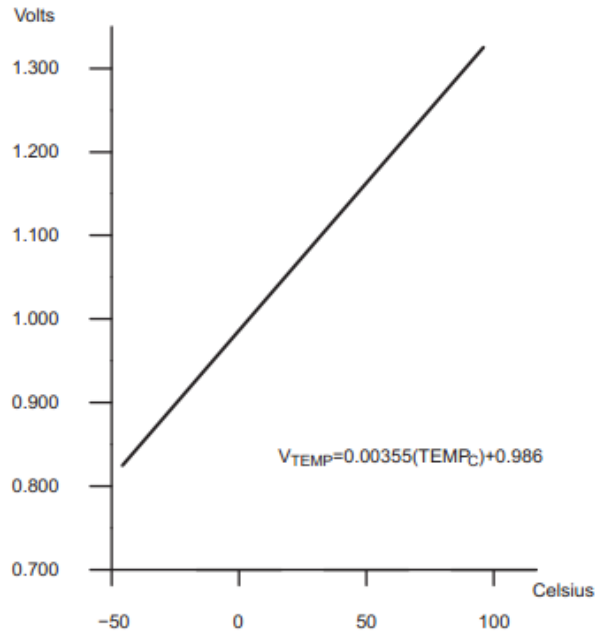


图 3.3.9 内部温度输出电压与温度的线性关系

```

void main ()
{
    int ADC10_Result;
    int TEMP;
    ADC10CTL1 |= CONSEQ_2;           //单通道重复采样模式
    ADC10CTL0 |= SREF_1+REFON; //选择内部参考源 1.5V, 打开基准源
    ADC10CTL0 |= ADC10SHT_3+MSC;    //过采样率设置为 64 个采样周期, 打开 AD 转换
    ADC10CTL1 |= ADC10SSEL_3+SHS_0; //ACLK2 分频为采样时钟, 用 ADC10SC 触发采集
    ADC10CTL1 |= INCH_1;           //选择通道 A1
    ADC10CTL0 |= ADC10ON;          //开启 ADC10
    while(1)
    {
        ADC10CTL0 |= ENC+ADC10SC;  //开始转换
        while((ADC10CTL0 & ADC10IFG)==0); //等待 ADC10IFG 标志变高 (转换完成)
        ADC10_Result=ADC10MEM;     //读取采样结果
        TEMP= (ADC10_Result-746)/(0.000355*678)+286; //计算温度值, 扩大 10 倍, 单片机在 28.6°C 下 ADC10 采样值为 746, 选择此点进行校准。同时要将 ADC10 采样值转换为电压值, 1V 电压时, 采样值为 678
    }
}

```


第四节 16 位定时/计数器

在 MSP430 系列单片机中，都带有一个 16 位定时/计数器 TIMER_A，用以精确定时或计数。在普通计数器的基础上还添加了 3 路捕获比较模块，能够在无需 CPU 的干预的情况下自动根据触发条件捕获定时器的计数值或自动产生输出波形。

3.4.1 Timer_A 定时/计数器的主计数器模块结构和原理

Timer_A 分为 2 部分：主计数器和比较捕获模块。主计数器负责定时，计时或计数。计数值（TAR 寄存器的值）被送到各比较捕获模块中，它们可以在无需 CPU 干预的情况下根据触发条件与计数器值自动完成某些测量和输出功能。只需定时或计数功能时，可以只使用主计数器部分。在 PWM 调制，利用捕获测量脉宽，周期等应用之中还需要捕获比较模块配合。

与 Timer_A 定时器中的主计数器相关的控制位都位于 TACTL 寄存器中，主计数器的计数数值存放与 TAR 寄存器中。每个比较捕获寄存器 TACCRx (x=0, 1, 2)。在一般定时器应用中，TACCRx 可提供额外的定时中断触发条件；在 PWM 输出模式之下，TACCRx 可用于设置周期和占空比；在捕获模式下，TACCRx 存放捕获结果。

主计数器结构如图 3.4.1 所示，它包括时钟源选择，预分频，计数器与计数模式选择几个部分。

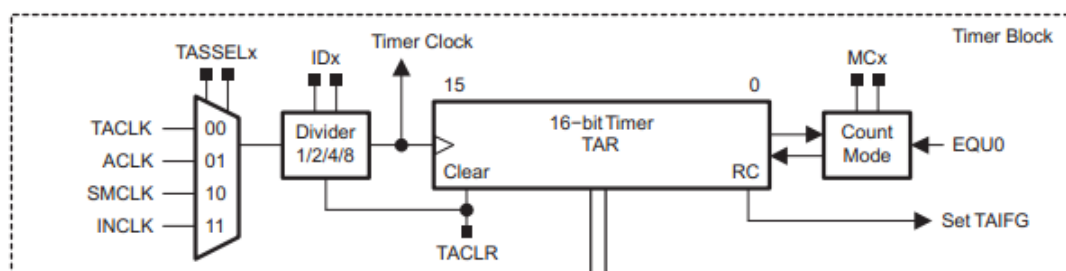


图 3.4.1 TA 主计数器结构

相关控制字有：

TASSELx: Timer_A 计数器的时钟源选择（位于 TACTL 寄存器）

00=外部管脚（TACLK） 01=ACLK 10=SMCLK 11=外部管脚（TACLK 取反）

快捷宏定义：TASSEL_0 TASSEL_1 TASSEL_2 TASSEL_3

IDx: Timer_A 计数器预分频系数（位于 TACTL 寄存器）

00=无分频 01=2 分频 10=4 分频 11=8 分频

快捷宏定义：ID_0 ID_1 ID_2 ID_3

通过上面两组控制位，可以设置定时计数时钟源。在低功耗应用以及需要长时间定时或计时的情况下，可以用 ACLK 作为作为时钟，加上预分频，最长的定时计时周期可达 16s。在高分辨率短时间应用之中，可以选择 SMCLK 作为时钟源。

若选择 TACLK 作为，定时器实际上成为计数器，累积从 TACLK 管脚上输入的脉冲，上升沿计数。若选择 TACLK 取反作为时钟源，TACLK 的下降沿计数。

TACLx: Timer_A 计数器清零控制位（位于 TACTL 寄存器）

0=不清零 1=清零

将该控制位置 1，可以立即将 Timer_A 计数器清零，无需通过软件赋值操作来实现。计数器复位后该标志位自动归零，因此读该标志位将永远读回 0。

MCx: Timer_A 计数器的技术模式（位于 TACTL 寄存器）

00=停止 01=增计数 10=连续增计数 11=增减计数

快捷宏定义: MC_0 MC_1 MC_2 MC_3

TAIFG: Timer_A 计数器溢出标志位（位于 TACTL 寄存器）

0=未发生溢出 1: 发生了溢出

Timer_A 计数器提供了 3 种计数模式：增计数，连续计数和增减计数。在增计数模式下，每个时钟周期 TAR 加 1。在 TAR 值超过 TACCRO 寄存器（捕获比较模块 0 模式值）时自动清零，同时将 Timer_A 溢出标志位 TAIFG 置 1。如果 TA 中断被允许，还会引发中断。改变 TACCRO 寄存器可以改变定是周期，且不存在初值装载问题，非常适合产生周期定时中断，只要改变 TACCRO 的值即可随意调整定时周期。

在连续计数模式下，其工作方式与 8051 的定时器基本相同。每个时钟周期 TAR 加 1，计数器值超过 0xFFFF 后溢出，TAR 回到 0，同时将 TAIFG 置 1，或引发中断。如果中断内给 TAR 重新赋初值，也可以产生不同周期的定时中断。用增计数模式产生定时中断比连续模式更加简单，一般不用连续模式更简单；连续模式一般在捕获下使用较多，让计数器自由运行，利用捕获功能在事件发生时自动记录下计数值，通过对比几个值可以确定时间发生的准确时间或者准确的时间间隔。

在增减模式下，计数器从 0 开始递增，计到 TACCRO 后，自动切换为递减模式，减到 0 后又恢复为递增模式，如此往复，在 TAR 从 1 变为 0 的时刻产生 TAIFG 中断标志。在一般应用中，不用增减模式来定时或计数，而多用于 PWM 发生器。借助增减模式，捕获比较模块能够产生带死区的对称 PWM 驱动波形，可以直接驱动半桥电路，无需专门的死区产生电路。

例如，在 MSP430 单片机中，为 Timer_A 配置时钟源及工作模式，使 Timer_A 在无需 CPU 的干预下，每隔 1.3125s 溢出一次（假设 SMCLK=MCLK=1.048576MHz，ACLK=32.768kHz）。

首先 1.3125 时间较长，若使用 SMCLK 作为时钟源 16 位计数器不够用，应该使用低频 ACLK 作为时钟源。再考虑周期性定时，3 种模式都能实现，其中增计数模式最简单，无需重复置初值等操作。最后计算 TACCRO 的值应该 1.3125 乘以 ACLK 频率得到设置值 43008。计数从 0 开始，实际应设置为 43007。

```
TACTL |=TASSEL_1+ID_0+MC_1;
```

```
TACCRO=43008-1;
```

3.4.2 Timer_A 定时/计数器比较模块

当 CAP 控制位设为 0 时，捕获比较模块工作在比较模式。此时 TACCRx 的值由软件写入，并通过比较器与主计数器的计数值进行比较。每次相等产生 EQU 信号，该信号触发输出逻辑，通过 OUTMODE 控制位可以配置输出逻辑，通过不同的输出逻辑配合产生不同的输出波形。整个过程无需 CPU 干预，软件中只需改变 TACCRx 的值即可改变输出波形的某些参数。对于不同型号的芯片，波形输出 TA_x 所对应的管脚有所不同，读者可参考相应型号所对应的芯片手

册。例如在 MSP430F42x 系列单片机中，TA0 对应 P1.0，TA1 对应 P1.2，TA2 对应 P2.0。某些芯片会将多个管脚对应一个输出方便布线，使用时可利用 IO 口的 PxSE1 寄存器激活输出功能。如图 3.4.2 所示为 TA 输出模块的逻辑图。

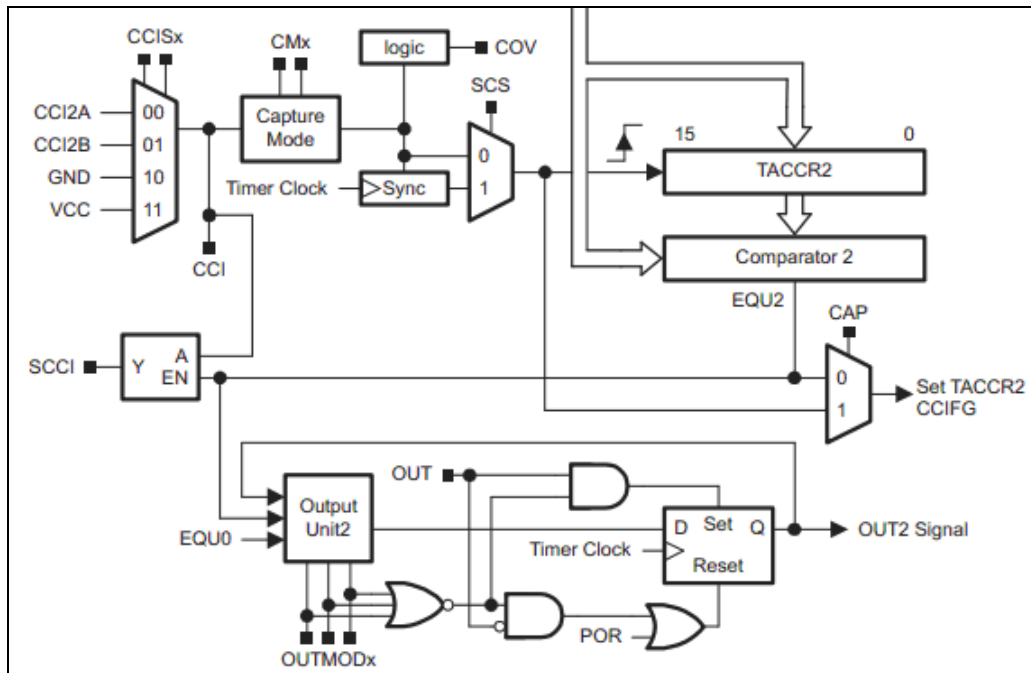


图 3.4.2 TA 输出模块逻辑图

OUTMODx: Timer_A 比较模块的输出模式位 (TACCTL0/1/2 寄存器)

OUT: Timer_A 比较模块的输出电平控制位 (位于 TACCTL0/1/2 寄存器)

Timer_A 比较模块的 8 种输出模式如表 3.4.2 所示:

表 3.5 Timer_A 比较模块输出模式表

OUTMODEx 控制位	输出模式控制	说明
000 (模式 0)	电平输出	Tax 管脚输出电平由 OUT 控制位决定
001 (模式 1)	延迟置位	当主计数器计至 TACCRx 值时, TAx 置 1
010 (模式 2)	取反/清零	当主计数器计至 TACCRx 值时, TAx 管脚取反 当主计数器计至 TACCRO 值时, TAx 管脚置 0
011 (模式 3)	置位/清零	当主计数器计至 TACCRx 值时, TAx 管脚置 1 当主计数器计至 TACCRO 值时, TAx 管脚置 0
100 (模式 4)	取反	当主计数器计至 TACCRx 值时, TAx 管脚取反
101 (模式 5)	延迟清零	当主计数器计至 TACCRx 值时, TAx 管脚置 0
110 (模式 6)	取反/置位	当主计数器计至 TACCRx 值时, TAx 管脚取反 当主计数器计至 TACCRO 值时, TAx 管脚置 1
111 (模式 7)	清零/置位	当主计数器计至 TACCRx 值时, TAx 管脚置 0 当主计数器计至 TACCRO 值时, TAx 管脚置 1

(1) 模式 0 (电平输出): 在输出模式 0 下, TAx 管脚与普通的输出 IO 口一样, 可以由软件操作 OUT 控制位来控制 TAx 管脚的高低电平。

(2) 模式 1 与模式 5 (单脉冲输出): 利用比较模块的模式 1 和模式 5, 可以替代单稳态电路, 产生单脉冲波形。

在输出模式 1 下，当主计数器计至 TACCRx 值时，TAx 管脚置 1。如果通过 OUT 控制位事先将 TAx 的输出设为低，经过 TACCRx 个周期后，TAx 将自动变高。这样做可以输出一个低电平脉冲。通过改变 TACCRx 的值，可以改变低电平脉冲的周期，且脉冲过程无需 CPU 的干预。

在输出模式 5 下，当主计数器计至 TACCRx 值时，TAx 管脚置 0。如果通过 OUT 控制位事先将 TAx 输出设置为高，经过 TACCRx 个周期后，TAx 将自动变低。这样做可以输出一个高电平脉冲。通过改变 TACCRx 的值可以改变该点脉冲的周期，且脉冲过程无需 CPU 的干预。

(3) 模式 3 和模式 7 (PWM 输出)：脉宽调制是最常用的功率调整手段之一。所谓脉宽调制，顾名思义，是指在脉冲方波周期一定的情况之下，通过调整脉冲的宽度，改变负载通断时间的比例，以达到功率调节的目的。

PWM 波形中，负载接通时间与一个周期总时间之比叫做占空比。占空比越大，负载功率就越大。如果 PWM 频率足够高以至于不足以表现出负载断续，从宏观上看，负载实际功率将是连续的。

在 PWM 调整负载功率的过程中，负载断开时晶体管无电流通过，不发热。负载接通时晶体管饱和，虽然通过有较大电流，但压降很小，发热功率也很低。所以使用 PWM 控制负载时，开关器件的总发热量很小。相比于串联耗散式的调整方法，效率会高很多，适合大功率，高效率的负载调整应用。但 PWM 的缺点是负载功率高频波动很大，不适合要求输出平稳无纹波要求的场合。

此外，PWM 控制本身属于开环控制，具有调节功能，但不具有稳定负载的能力，也不保证输出结果正比于占空比。例如在电机调速实验中，通过 PWM 控制可以改变电动机的功率，但不能稳定电动机的转速，电动机的转速会受负载力矩的影响。

需要得到高精度，高稳定性，快速且无超调的控制结果时，需要反馈式控制系统。在 MSP430 单片机中，通过 ADC 采集功能测量实际被控量作为反馈信号，结合强大 CPU 计算能力实现各种反馈控制算法，最终通过 PWM 控制输出量，可以用单芯片构成各种反馈式控制系统。

在输出模式 7 下，每次 TA 计数值 TACCRx 时，TAx 管脚会自动置低，当 TA 计至 TACCRO 时，TAx 管脚会自动置高，实际输出波形就是调制 PWM 方波。改变 TACCRO 的值即可改变 PWM 的周期，改变 TACCRx 的值即可改变从 TAx 管脚输出信号的占空比，TACCRx 越大，占空比越大。

在模式 3 下，与模式 7 刚好相反。TACCRx 越大，占空比越小。对于某些低电平接通负载的电路，用模式 3 更符合习惯。模式 3 和模式 7 也常一起用，用于产生两路对称的波形。

TACCRO 被用于 PWM 周期设定，通过 Timer_A 产生的若干路 PWM 波形的周期都是一样的。且对于含有 3 个捕获比较模块的 TimerA，最多只能产生两路 PWM 波形。某些型号的单片机中含有 5 个比较捕获模块的 TimerA，最多产生 4 路独立的 PWM 波形。

(4) 模式 4 (可变频率输出)：输出模式 4 下，TA 计数每次达到 TACCRx 的值时，TAx 管脚电平自动取反。因此，改变 TA 计数周期可以改变 TAx 管脚的输出频率；同时若改变 TACCRx 的值可以改变波形的相位。

改变 TACCRO 的值，即可同时改变 3 路输出波形的频率，改变 TACCR1 与 TACCR2 的值，

可以改变 TA1 与 TA2 输出波形与 TA0 波形之间的相位差。由于 TACCR1 与 TACCR2 最大值只能等于 TACCRO，所以最大值只能滞后 0—180 度。如果需要超过 180 度的移相，可以通过改变管脚的初始值来实现。

3.4.3 Timer_A 捕获模块

在捕获模式下，用某个指定管脚的输入电平跳变触发捕获电路，将此刻主计数器的计数值自动保存到响应的捕获值寄存器中。该过程纯硬件实现，无需 CPU 的干预，不存在中断响应等时间延迟。可以用于测频率、测周期、测脉宽、测占空比、门控计数等需要获得精确时间量的场合。

将 TACCTLx 中的 CAP 位置 1 使能捕获模块。通过 CMx 选择触发沿（详见寄存器表）。触发沿可以根据实际需要进行选择。例如，测量方波周期时用上升沿或者下降沿都可以，测量脉宽时用上升下降触发方式较为方便。

捕获过程通过纯硬件进行实现，实时性很强，CPU 可以通过查询或中断内读取捕获值，根据两次捕获值之间的差即可计算出周期脉宽等信息。即使读取略有延迟也不影响捕获结果。这比单纯靠外部中断保存计数器值的方法高效得多，且不要求 CPU 直接进行读取。但也可能遇到第一次捕获值尚未来得及被 CPU 读取的情况下，第二次捕获又成立了，这种情况称为捕获溢出。会导致计算错误。为此，模块中留有一个标志位用于指示溢出。COV 为 1 时发生了捕获溢出。该标示位必须通过软件清除。SCS 位为 0 时，捕获过程的锁存直接由硬件电路控制，不受时钟的约束。当该标示位为 1 时，捕获触发信号经过 D 触发器与定时器时钟同步。假设捕获触发条件触发在值为 N 的时间段内，异步模式将捕获到数值 N，同步模式下捕获到数值 N+1。一般建议工作在同步模式下，以免数字逻辑部分出现竞争产生毛刺。

CCI: Timer_A 捕获模块的输入信号电平（异步）

SCCI: Timer_A 捕获模块的输入信号电平（同步）

3.4.4 Timer_A 定时器中断

Timer_A 定时器的下列四种事件均能产生中断：

(1) 主计数器（TACCRO）计满后复位，TAIFG 标志被置 1。中断发生在计数值从 TACCRO 跳至 0 时刻。

(2) 捕获通道 0 发生捕获事件，或让主计数器值 TAR 计至 TACCRO（计数值从 TACCRO-1 跳至 TACCRO 的时刻），TACCTL0 寄存器内的 CCIFG 标志被置 1。

(3) 捕获通道 1 发生捕获事件，或主计数值 TAR 计至 TACCR1（计数器从 TACCR1-1 跳至 TACCRO 的时刻），TACCTL1 寄存器内的 CCIFG 标志被置 1。

(4) 捕获通道 2 发生捕获事件，或主计数器 TAR 计至 TACCR2（计数值从 TACCR2-1 跳至 TACCR2 的时刻），TACCTL2 寄存器内的 CCIFG 标志被置 1。

这 4 种事件占用了两个中断源，其中，事件 2（计至 TACCR2 或捕获通道 0 发生捕获事件）独占一个中断源 TIMERA0_VECTOR，其余 3 种事件共用另一个中断源 TIMERA1_VECTOR。对于需要紧急处理的捕获事件建议使用通道 0，因为它独占一个中断源，在终端内无需分之判断，反应最快。

相应的标志位有：

TAIFG: Timer_A 计数器计满复位标志（位于 TACTL 寄存器）

TAIE : Timer_A 主计数器计满中断允许位（位于 TACTL 寄存器）

CCIFG: Timer_A 捕获/比较模块中断标志（位于 TACCTL0/1/2 寄存器）

比较模式下，当主计数器计至 TACCRx 时，该标志位置 1。在捕获模式下，当捕获条件发生时，该标志位置 1。TACCTL0 内的 CCIFG 标志会在中断执行后自动清零，其余模块共用了中断入口，它们的 CCIFG 标志位会根据 TAIV 寄存器的值在执行相应的中断后自动清除。

CCIE: Timer_A 比较捕获模块中断允许位（位于 TACCTL0/1/2 寄存器）

TAIV: Timer_A 中断向量寄存器

几个事件共用了 TIMERA1_VECTOR 中断向量，需要在中断服务程序中通过软件判断 TAIV 寄存器的值来确定具体中断原因。

表 3.6 Timer_A 中断向量寄存器表

TAIV 值	中断源	中断标志	优先级
00H	无中断发生		
02H	捕获/比较模块 1	TACCTL1 内的 CCIFG	最高
04H	捕获/比较模块 2	TACCTL2 内的 CCIFG	
06H	*捕获/比较模块 3	TACCTL3 内的 CCIFG	
08H	*捕获/比较模块 4	TACCTL4 内的 CCIFG	
0AH	主计数器计满溢出	TAIFG	最低

注：带*号的中断源只对 Timer_A5 单片机有效

3.4.5 寄存器图

TACTL:

15	14	13	12	11	10	9	8
Unused						TASSELx	
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
IDx		MCx		Unused	TACLr	TAIE	TAIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

未使用: 15~10 位

TASSELx: 9~8 位 Timer_A 时钟源选择

00 TACLK

01 SMCLK

02 ACLK

03 INCLK

IDx: 7~6 位 时钟分频选择

00 /1

01 /2

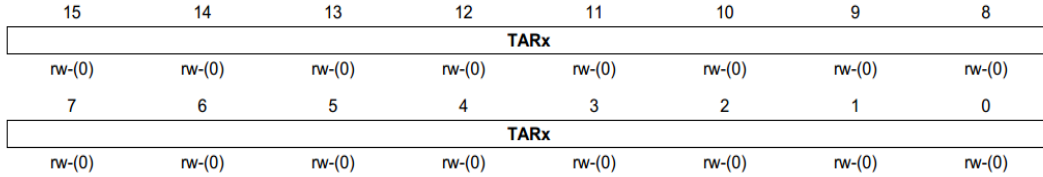
02 /4

03 /8

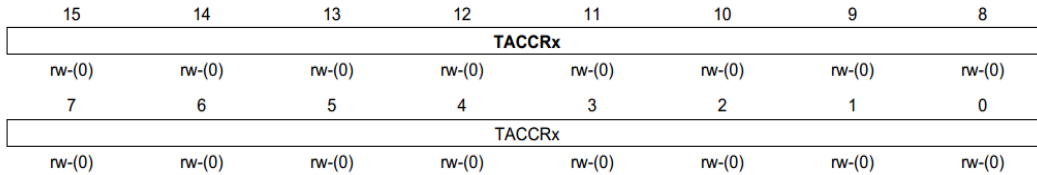
MCx: 5~4 位 模式控制

- 00 停止模式
- 01 增计数模式，计至 TACCRO
- 02 连续技术模式，计至 0xFFFF
- 03 增减计数模式，增计数计至 TACCRO 减至 0

- 未使用: 3 位 未使用
- TACL: 2 位 清零位，计数器清零，分频和模式位也清零
- TAIE: 1 位 中断允许位，中断允许时置 1
- TAIFG: 0 位 中断允许标志位，有中断时置 1



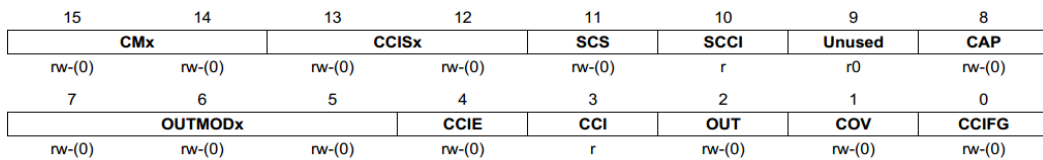
TARx: 15~0 位 Timer_A 计数器寄存器



TACCRx: TimerA 比较捕获寄存器

比较模式: TACCRx 与 TAR 比较，当相等时改变输出管脚的电平

捕获模式: 当捕获源到来时将 TAR 复制到 TACCRx



TACCTLx 捕获比较控制寄存器

- CMx:** 15~14 位 捕获模式
 - 00 无捕获
 - 01 上升沿捕获
 - 02 下降沿捕获
 - 03 上升下降沿捕获

- CCISx:** 13~12 位 捕获比较输入选择
 - 00 CCIxA
 - 01 CCIxB
 - 02 GND
 - 03 Vcc

- SCS:** 11 位 同步异步选择

	0	异步捕获
	1	同步捕获
SCCI:	10 位	捕获同步信号输入端
未使用:	9 位	
CAP:	8 位	捕获模式
	0	比较模式
	1	捕获模式
OUTMODEx:	7~5 位	输出模式配置
	000	模式 0
	001	模式 1
	010	模式 2
	011	模式 3
	100	模式 4
	101	模式 5
	110	模式 6
	111	模式 7
CCIE:	4 位	
	0	不允许中断
	1	中断允许
CCI:	3 位	捕获比较输入
OUT:	2 位	在模式 0, 该位直接控制输出电平
COV:	1 位	捕获溢出控制位
	0	无捕获溢出
	1	捕获溢出
CCIFG:	0 位	捕获比较中断
	1	无中断发生
	0	中断发生

第五节 FLASH 控制器

本节将介绍 MSP430G2553 单片机 Flash 控制器的结构和原理,FLASH 控制器的相关寄存器定义和 FLASH 的读写操作。

3.5.1 Flash 存储介绍

MSP430 Flash 存储器可以按位、按字节和按字进行寻址和编程的。Flash 块带有一个可以控制编程和擦除操作的控制器,该控制器内有 4 个寄存器、1 个时序信号发生器和 1 个提供编程和擦除电压的电压发生器。

MSP430 Flash 存储器的特征包括:

- (1) 内部编程电压的产生
- (2) 可按位、字节或字可编程
- (3) 超低功耗操作
- (4) 段擦除和全部擦除主 Flash 区
- (5) 边界 0 和边界 1 读模式

Flash 控制器的结构框图如图 3.5.1 所示:

注意:在对 Flash 进行读和写操作期间供电电压 V_{CC} 的最小值 2.2V,如果低于这个电压值,则写和擦除的结果将是不可预知的。

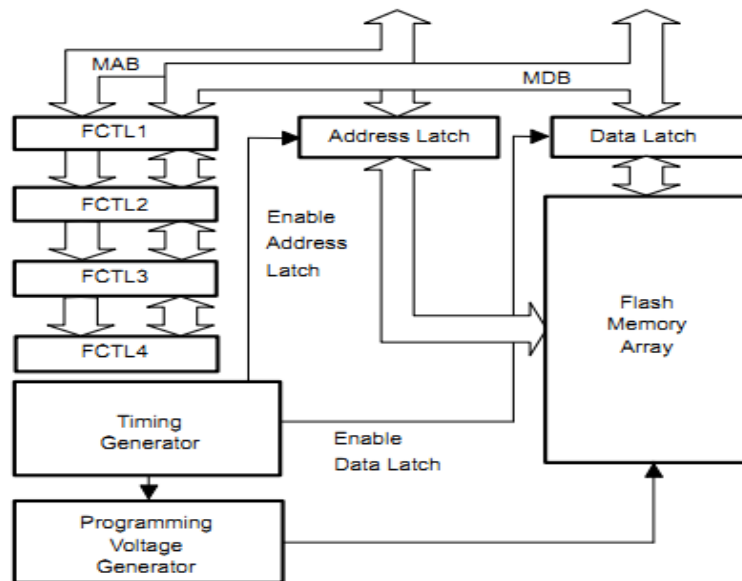


图 3.5.1 Flash 存储器块结构框图

3.5.2 Flash 存储器的分段

MSP430 Flash 被分割成不同的段(segment)。虽然单一的位、字节或字都可以被写入到 Flash 中,但段是 Flash 可擦除的最小单位。

Flash 的段分别隶属于主 Flash (MainFlash) 区和信息 Flash (InfoFlash) 区。在对主 Flash 区和信息 Flash 区进行操作时几乎没有差别,所以代码和数据可以放置于这两个区中

的任意一个。这两个区的区别在于它们段的大小和所处的物理地址范围不同。

信息 Flash 区有 4 个 64 字节的段。主 Flash 区至少有两个 512 字节的段。

段又进一步被划分成块(block)，下图显示了一个带有 8 个主 Flash 区段和 4 个信息 Flash 区段的 32KB Flash 分段结构图：

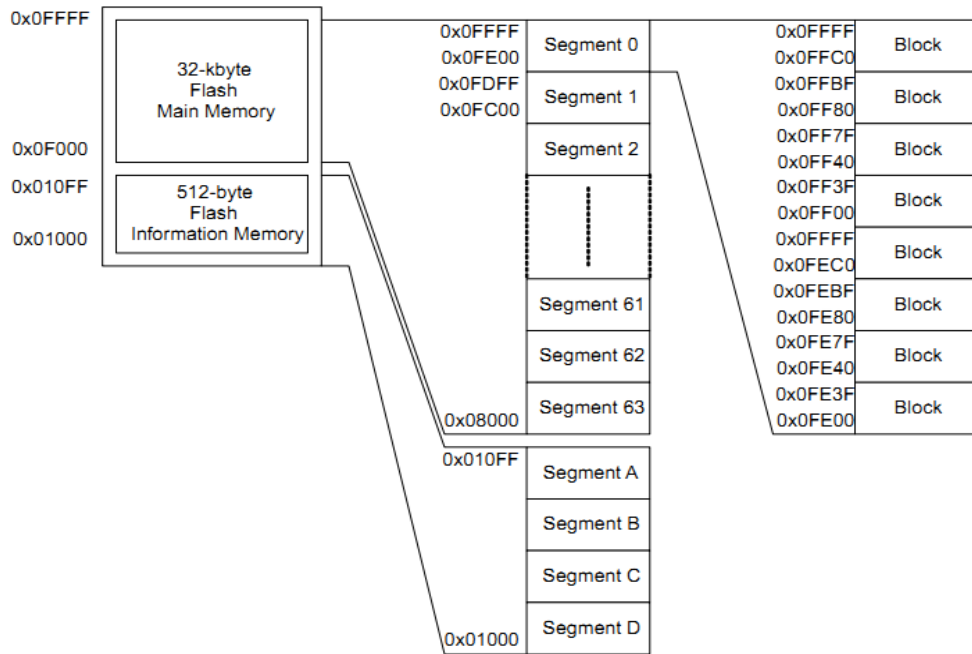


图 3.5.2 32-KB Flash 段划分结构

InfoA 段：

可以使用 LOCKA 位将信息 Flash 区的 InfoA 段独立于其他的段被单独锁住。当 LOCKA=1 时,Flash 被锁定,用户不能对 InfoA 段进行写和擦除的操作,并且整个信息 Flash 区在全部擦除主 Flash 区和编程期间不响应其他的擦除操作;而当 LOCKA=0 时,Flash 解除锁定,用户可以像对 flash 中其他的段一样对 InfoA 段进行擦除和写入,并且整个信息 Flash 区在全部擦除主 Flash 区和编程期间被擦除。

对 LOCKA 位写 1 该位的状态将被取反,写 0 时不起作用。这使得现有的 Flash 编程程序被使用而不发生改变。

InfoA 段的开锁和上锁分别使用下面的语句就可以了：

```
FCTL3=FWKEY; //开锁
```

```
FCTL3=FWKEY+LOCKA; //上锁
```

3.5.3 Flash 的操作

Flash 的默认状态下的模式是读模式。在该模式下,Flash 是不能被擦除或者写入的,此时 Flash 的时序信号发生器和电压发生器处于关闭状态,对 Flash 的操作几乎就和 ROM 一样。

MSP430 Flash 可在线编程的 (in-system programmable, ISP), 无需额外部电压的供应。MSP430 CPU 可以对自身的 Flash 进行写和擦除操作,具体的模式是通过 BLKWRT、WRT、MERAS 和 ERASE 位来进行选择的, 分别有：

- (1) 写字节/字模式
- (2) 写块模式
- (3) 单段擦除模式
- (4) 全部擦除主 Flash 区模式（针对所有主 Flash 区的段，保留信息 Flash 区的内容）
- (5) 全部擦除所有 Flash 模式（针对所有的段）。

在编程和擦除期间禁止向 Flash 进行读写操作。在读、写 Flash 期间 CPU 执行的代码必须放置于 RAM 中。用户可以从 Flash 内部或 RAM 对 Flash 进行任意的更新。

1. Flash 时序信号发生器

如图 3.5.3 所示，用户对 Flash 进行的写和擦除操作是由 Flash 时序信号发生器来控制的。Flash 时序信号发生器的工作频率 f_{FTG} 必须在 257KHz~476KHz 范围内（查看具体器件的数据手册）。

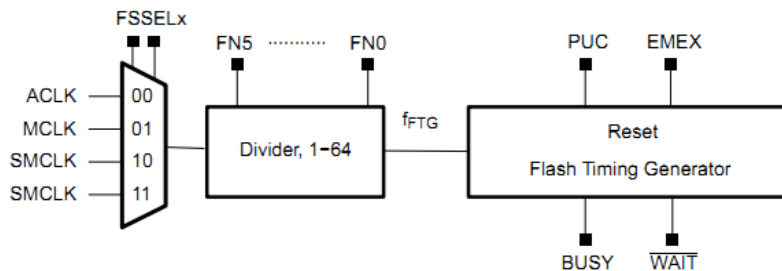


图 3.5.3 Flash 存储器时序信号发生器结构框图

Flash 时序信号发生器的选择：

Flash 时序信号发生器可以选择活动时钟（ACLK）、子系统时钟（SMCLK）或者主时钟（MCLK）作为自己的时钟源，被选择的时钟源可以通过 FN_x 位分频以满足 Flash 工作频率 f_{FTG} 要求。如果读、写操作期间 f_{FTG} 的频率偏离如上所述的说明规定，那么写和擦除操作的结果将是不可预知的，或者 Flash 将在被迫能可靠工作的极限范围之外工作。

如果写或擦除操作期间时钟信号检测不到时钟信号，操作将被终止，这时 FAIL 标志位被置位，操作的结果同样是不可预料的。

当写或者擦除操作正在进行时，用户是不能通过禁用所选择的时钟源以使 MSP430 进入低功耗模式的，时钟源要等到操作完成后才能被禁用。

2. Flash 的擦除

Flash 位擦除的电平为高电平。每个位可单独地被从 1 复位成 0 但需要一个擦除周期才能将其从 0 重新置位成 1。Flash 可被擦除的最小单位是段。用户可以通过 ERASE 和 MERAS 位的设置来选择三种擦除模式，如表 3.6 所示：

表 3.6 擦除模式的选择

MERAS	ERASE	擦除模式
0	1	单段擦除模式
1	0	全部擦除主 Flash 区模式（针对所有主存储区的段）
1	1	LOCKA=0：擦除主 Flash 区和信息 Flash 区的内容 LOCKA=1：只擦除主存储区的内容

任何的擦除操作都在所要擦除的地址范围内进行的空写操作。空写开启 Flash 时序信号发生器和擦除操作。图 3.5.4 说明了擦除操作的时序，在进行空写操作后，BUSY 位即刻被

置位并且在擦除周期内一直保持置位状态。BUSY、MERAS 和 ERASE 位在擦除周期结束后被自动清零。擦除周期的时序不依赖于器件上 Flash 的大小。所有的 MSP430F2XX 和 MSP430G2XX 器件擦除的次数是一样的。

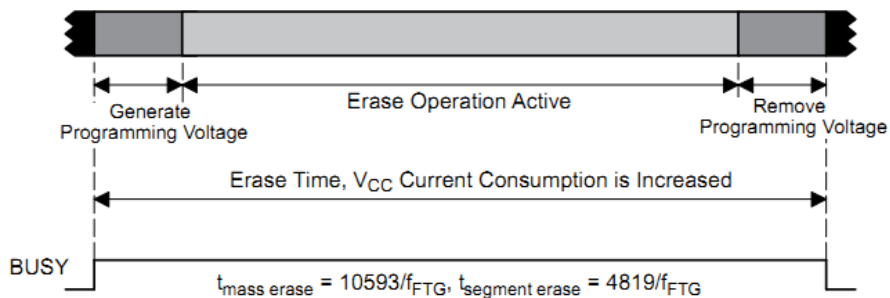


图 3.5.4 擦除周期的时序

对一个不在擦除范围内的地址进行的空写操作不会启动擦除周期，是会对 Flash 产生影响的，也无论如何不会改变标志位的。这个错误的空写操作也将被忽略。

(1) 从 Flash 存储器内部启动一个擦除操作

任意一个擦除周期都可以从 Flash 内部或者从 RAM 启动。当在一个 Flash 的段上进行擦除操作时，所有的时序是由 Flash 控制器来控制的，CPU 的执行将被延迟。当擦除周期结束时，CPU 用一条跟随着空写操作的指令来恢复代码的执行。

当从 Flash 内部启动一个擦除周期，有可能会擦除在擦除之后要执行的代码，如果发生这种情况的话，那么在擦除周期之后，CPU 要执行的内容将是不可预知的。

下面就是从 Flash 内部启动一个擦除操作的流程图：

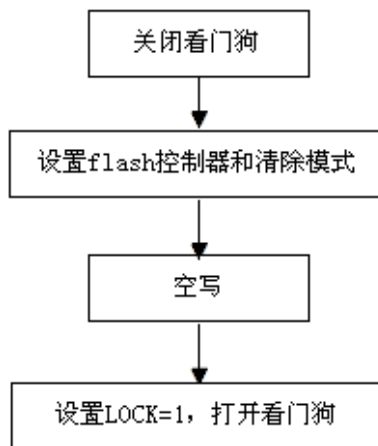


图 3.5.5 从 RAM 启动清除周期流程图

写成 C 代码如下(以擦除 InfoD 段为例)：

```

#define SegD 0x01100 //准备对 InfoD 段进行擦除
WDTCTL=WDTPW+WDTHOLD; //关闭看门狗
FCTL2=FWKEY+FSSEL1+FN0; //选择 SMCLK 作为时钟源，二分频
FCTL3=FWKEY; //开锁
FCTL1=FWKEY+ERASE; //准备擦除
*((* int)SegD)=0x00; //段擦除—空写
FCTL3=FWKEY+LOCK; //上锁
  
```

WDTCTL=WDTPW+WDTCNTCL; //打开看门狗

(2) 从 RAM 启动一个写擦除操作

任何的清除周期都是从 RAM 开始的。在这种情况下，CPU 的执行是不被延迟的，它仍然可以继续执行 RAM 中的代码，但是必须通过轮询 BUSY 位的状态，确定清除周期已经结束 CPU 才能重新访问任意的 Flash 地址。如果 Flash 在 BUSY=1 时被访问，将引起访问冲突，这时 ACCVIFG 标志位也将被置位，这样清除的结果也是不可预知的。

从 RAM 启动 Flash 的擦除流程如图 3.5.6 所示：

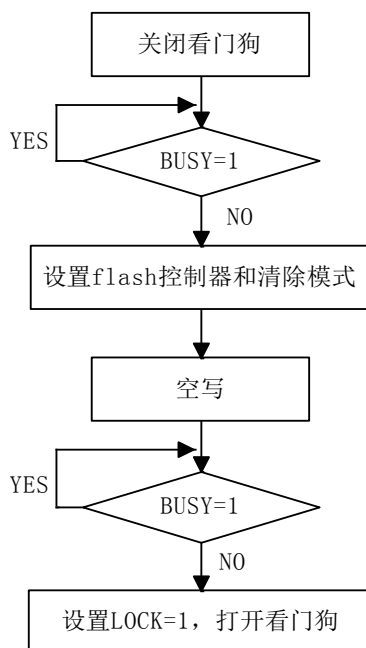


图 3.5.6 从 RAM 启动擦除周期的流程图

写成 C 代码如下(以擦除 InfoD 段作为例)：

```

WDTCTL=WDTPW+WDTCTL; //关闭看门狗
while (BUSY==(FCTL3&BUSY)); //检测忙，若忙，则等待
FCTL2=FWKEY+FSSEL1+FN0; //选择 SMCLK/2 作为时钟源，二分频
FCTL3=FWKEY; //开锁
*((* int)SegD)=0x00; //空写
while (BUSY==(FCTL3&BUSY)); //检测忙，若忙则等待
FCTL3=FWKEY+LOCK; //上锁
WDTCTL=WDTPW+WDTCNTCL; //打开看门狗
  
```

写 Flash:

通过 WRT 和 BLKWRT 位的设置来选择写模式的方法，如表 3.5.2 所示

表 3.6 写模式的选择

BLKWRT	WRT	写模式
0	1	写字节/字
1	1	写块

两种写模式使用各自不同的一些列写命令来实现的，但是写块的模式速度大约是写字节/字模式的两倍，因为电压发生器在写块的整个过程中一直保持开启状态。任意一条修改目

的操作数的命令都可以用来修改写字节/字模式或写块模式的 Flash 的位置。在两次擦除之间，同一个 Flash 字（低字节+高字节）一定不能被写入两次以上。否则，将对 Flash 产生损害。

BUSY 位在写操作正在进行时被置位，在操作完成时被清零。如果写操作是从 RAM 启动的，在 BUSY=1 时，CPU 一定不能访问 Flash，否则将产生访问冲突，这时 ACCIFG 也将被置位，对 Flash 写的后果也是不可预知的。

(1) 写字节/字

一个写字节/字操作可以从 Flash 内部或 RAM 里启动。当从 Flash 内部启动写字节/字操作时，所有的时序是由 Flash 控制器控制的，CPU 的执行被延迟直至写操作完成。写操作完成后，CPU 用一条跟随着下一个写操作的指令恢复代码的执行。字节/字的写时序如图 3.5.7 所示：

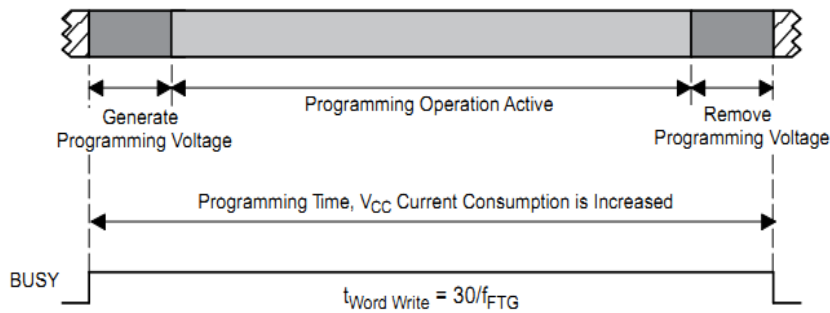


图 3.5.8 写字节/字时序

当从 RAM 中执行一个字节/字写操作的时候，CPU 将继续执行 RAM 中的代码。BUSY 位在 CPU 再次访问 Flash 之前必须清零，否则将产生访问冲突，ACCIFG 将被置位，写操作的后果是不可预知的。

在字节/字模式下，每写一次字节/字需要花费 30 个 f_{FTG} 周期，而其中有 27 个周期整个 64 字节的块上需要施加内部产生的编程电压。随着每一个字节或字的写入，块上施加编程电压的时间量不断累积，但是任何一个块来说，编程电压累积的时间量都不能超过累积编程时间 t_{CPT} 。如果达到了累积编程时间，那么在对块内的任意地址执行任何进一步的写操作之前，就必须对块进行擦除。

(2) 从 Flash 内启动一个写字节/字操作

从 Flash 发起一个字/字节的写操作的流程图如下：

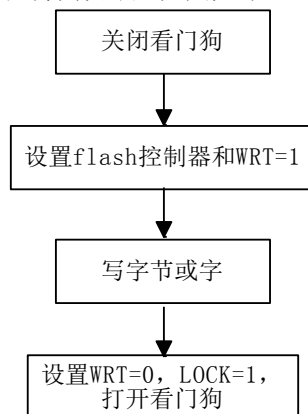


图 3.5.9 从 Flash 启动一个写字节/字操作

写成 C 代码如下：

```
#define Addr 0x01100
WDTCTL=WDTPW+WDTHOLD; //关闭看门狗
FCTL2=FWKEY+FSSEL1+FN0; //选择 SMCLK/2 作为时钟源，二分频
FCTL3=FWKEY; //开锁
FCTL1=FWKEY+WRT; //写使能
*((* int )Addr)=0x1234; //向地址 0x01100 写入字 0x1234，假设该地址单元已被擦除过
FCTL1=FWKEY; //写关闭
FCTL3=FWKEY+LOCK; //上锁
WDTCTL=WDTPW+WDTCNTCL; //打开看门狗
```

(3) 从 RAM 启动一个字节/字操作

从 RAM 启动一个字节/字操作的流程图如下：

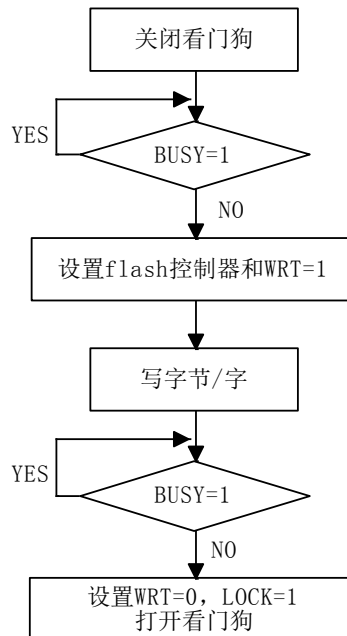


图 3.5.10 从 RAM 启动一个写字节/字操作

写成 C 代码如下：

```
#define Addr 0x01100
WDTCTL=WDTPW+WDTHOLD; //关闭看门狗
While(BUSY==(FCTL3&BUSY)); //检测忙，若忙，则等待
FCTL2=FWKEY+FSSEL1+FN0; //选择 SMCLK/2 作为时钟源，二分频
FCTL3=FWKEY; //开锁
FCTL1=FWKEY+WRT; //写使能
*((* int )addr)=0x1234; //向地址 0x01100 写入 0x1234，假设该地址单元已被擦除过
While(BUSY==(FCTL3&BUSY)); //检测忙，若忙，则等待
```

```

FCTL1=FWKEY;           //写关闭
FCTL3=FWKEY+LOCK;     //上锁
WDTCTL=WDTPW+WDTCTL; //打开看门狗

```

(4) 写块

当有许多连续的字节/字需要写入 Flash 时，使用写块模式可以加速对 Flash 的写入进程。在对大小为 64 字节的块进行写操作期间，Flash 的编程电压始终保持开启状态。但是任何块上编程电压累积的时间量都不能累积编程时间 t_{CPT} 。

写块操作不能从 Flash 而只能从 RAM 启动。BUSY 位在整个写块期间始终保持置位状态。此外，每对块写一个字节/字必须检查 WAIT 位的状态。当 WAIT 位被置位，说明上一次写操作已经完成，则块的下一个字节/字就可以被写入到 Flash 中。当连续写入块时，BLKWRT 位必须在当前块写入后清零，经过必要的恢复时间 t_{end} 之后，BLKWRT 可以被再次置位以启动下一个写块操作。BUSY 位在每一个写块操作完成之后清零预示着下一个块将要被写入。图 3.5.11 说明了写块操作的时序：

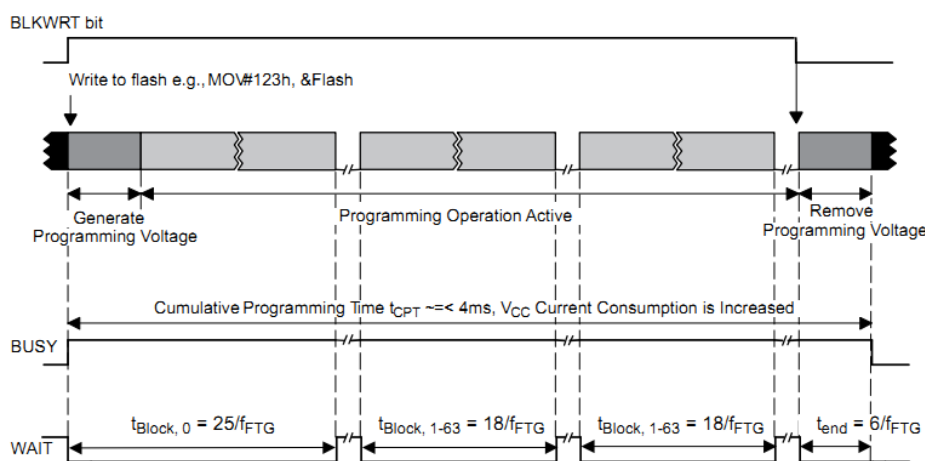


图 3.5.11 写块周期的时序

(5) 写块例子的流程图

写块的流程图和例子如下：

写成 C 代码如下：

```

//从 RAM 启动写，假设 Flash 已经被擦除
char Write_Counter=32 ;           //写计数
int Addr=0x01100;                //从地址 01100h 开始写
WDTCTL=WDTPW+WDTCTL;           //关闭看门狗
while (BUSY==(FCTL3&BUSY));     //检测忙，若忙，则等待
FCTL2=FWKEY+FSSEL1+FN0;        //选择 SMCLK/2 作为时钟源
FCTL3=FWKEY;                     //开锁
FCTL1=FWKEY+BLKWRT+WRT;        //使能写块模式
while (WAIT==(FCTL3&WAIT));     //检测 Flash 是否已准备好被写入下一个字节/字，若否，
//则等待准备好
Addr++;                           //指向下一个地址

```



```

Write_Counter --;           //待写入值计数减 1
if(Write_Counter ==0)
{
    FCTL1=FWKEY;           //写关闭
}
*((int *)Addr)=Write_Value; //这里 Write_Value 是用户要写入的值
While(BUSY==(FCTL3&BUSY)); //检测忙, 若忙, 则等待
FCTL3=FWKEY+LOCK;        //上锁
WDTCTL=WDTPW+WDTCNTCL; //打开看门狗

```

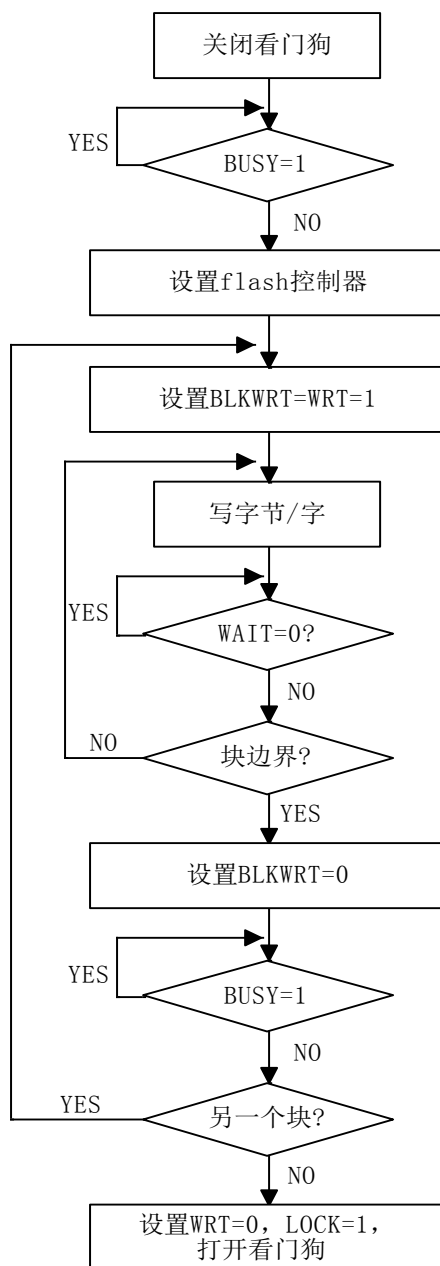


图 3.5.12 写块流程图

4. 读写或擦除期间 Flash 的访问

若在 BUSY=1 时从 RAM 启动的任何写或擦除操作，CPU 可能无法对 Flash 的任意位置进行读写操作。否则，将产生访问冲突，ACCIFG 将被置位，产生的后果也将是不可预知的。若在 WRT=0 尝试对 Flash 进行写操作，ACCIFG 中断标志将被置位，Flash 将不受影响。

从 Flash 内启动一个写字节/字或者任意的擦除操作时，Flash 控制器将在 CPU 获取下一条指令时返回操作码 03FFFh（它是一条跳转指令——JMP PC），这将导致 CPU 一直循环直至 Flash 操作完成为止。当操作完成并且 BUSY=0，Flash 控制器将允许 CPU 获取合适的操作码并且恢复程序的执行。

BUSY=1 时 Flash 的访问情况如表 3.7 所示：

表 3.7 当 BUSY=1 时 Flash 的访问

Flash 操作	Flash 访问	WAIT 位	结果
任意的清除 或写字节/字	读	0	ACCIFG=0，读取到的值是 03FFFh。
	写	0	ACCIFG=1，写操作被忽略。
	取指令	0	ACCIFG=0，CPU 取操作码 03FFFh，也就是 JMP PC 指令。
写块	任意操作	0	ACCIFG=1，LOCK=1
	读	1	ACCIFG=0，读到的值是 03FFFh。
	写	1	ACCIFG=0，写入要写入的内容。
	取指令	1	ACCIFG=1，LOCK=1

内存擦除周期开始之前应禁用看门狗定时器（在看门狗模式下）。在这种情况下，复位将终止擦除操作因此产生的结果也将是不可预知的。在擦除周期结束以后，看门狗可能重新被使用。

5. 停止一个写或擦除周期

任意的写或擦除操作都可以在该操作正常结束之前通过设置紧急退出位 EMEX 使其停止。通过设置 EMEX 位，用户可以立即停止当前执行的操作和 Flash 控制器。所有的 Flash 操作都将停止，Flash 将回到读模式状态，而且 FCTL1 寄存器中的所有位都将被清 0，这将产生不可预知的结果。

6. 边界读模式

边界读模式可以被用来验证 Flash 内容的完整性。在边界读模式下，可以检测到被边界编程 Flash 位的位置。产生这种情况的事件包括在擦除/编程操作期间不恰当的 f_{FTG} 设置或者是供电电压低于最小的 V_{CC} 值。找到这种存储位置的一种办法是对 Flash 的一个区（例如，一个 Flash 的段）周期性地计算一次校验和并且在边界读使能的情况下重复这一过程。如果它们不匹配的话，就肯预示着 Flash 区域的编程不足。可以通过禁用边界读模式、把 Flash 段复制到 RAM、擦除 Flash 段和从 RAM 将其写回到 Flash 中来刷新受影响的 Flash 段。

检查 Flash 中内容的程序必须从 RAM 中执行。从 Flash 中执行代码将自动禁用边界读模式。边界读模式由 MRG0 和 MRG1 寄存器的位来控制。寄存器 MRG1 的设置是用于检测那些包含一个“1”编程不足的 Flash 单元（被擦除的位），而寄存器 MRG0 的设置用于检测那些包含一个“0”的编程不足的 Flash 单元（被编程的位）的不足。在这些位中每次只能有一位被置位。因此，一个完整的边界读校验需要两次通过 Flash 内容完整性的检查。在边界读模式期间，Flash 的访问速度 (MCLK) 必须限定在 1MHz。

7. 配置和访问 Flash 控制器

FCTLx 是 16 位的、受密码保护的、可读/可写的寄存器。任何的读或写访问必须使用字指令并且写访问时必须在高字节包含 0A5h。对 FCTLx 寄存器写入其高字节不是 0A5h 的任何值都是违反安全密钥的，也都将使 KEYV 标志位置位和触发系统的复位。另外，从 FCTLx 寄存器的读取的值的高字节都是 096h。

在 Flash 被擦除或写字节/字操作期间对 FCTL1 寄存器的任何写入都将产生访问冲突也都将使 ACCIFG 位置位。虽然允许在写块模式和 WAIT=1 情况下写寄存器 FCTL1，但在写块模式和 WAIT=0 的情况下写 FCTL1 却会产生访问冲突，ACCIFG 也将被置位。

在 BUSY=1 的情况下，任何对 FGCTL2 寄存器写入将产生访问冲突。

任何一个 FCTLx 寄存器都可以在 BUSY=1 的情况下被读取，并且是读取不会产生访问冲突的。

8. Flash 控制器中断

Flash 存储器有 KEYV 和 ACCVIFG 两个中断源。ACCVIFG 在产生访问冲突时被置位。当 ACCVIFG 位在 Flash 写或擦除操作后被重新使能。ACCVIFG 位置位将产生一个中断请求，ACCVIFG 中断源于 NMI（非可屏蔽）中断向量，不需要置位 GIE 位来产生中断请求。ACCVIFG 也可以通过软件校验来确定是否发生了访问冲突。ACCVIFG 位必须要软件复位。

违反密钥的标志位 KEYV 在任意一个 Flash 控制寄存器被写入不正确的密码时被置位。当这种情况发生时，单片机将立即产生一个 PUC（上电清除）信号使器件复位。

9. Flash 存储器编程

有三种方式可以对一个 MSP430 Flash 器件编程，这三种方式都支持在线编程：

- 1) 通过 JTAG 编程
- 2) 通过引导加载程序(bootstrap loader)编程
- 3) 通过用户自定义的方法编程

(1) 通过 JTAG 对 Flash 编程

我们可以通过 JTAG 端口对 MSP430 器件编程。JTAG 接口需要 4 个信号（在 20 和 28 个引脚的器件上需要 5 个信号）：地、可选择端、 V_{CC} 和 \overline{RST} / NMI 。

JTAG 端口由保险丝保护。烧断保险丝将使 JTAG 端口被完全禁用，此过程是不可逆的，即不可能再通过 JTAG 实现对器件的进一步访问。具体的细节，请查看 the MSP430 Programming Via JTAG Interface User's Guide 这篇文章。

(2) 通过引导加载程序(BSL)对 Flash 编程

大多数的 MSP430 Flash 中都储存有一个引导加载程序。引导加载程序使用户可以通过使用 UART 串口对 Flash 进行读和编程。另外，通过引导加载程序对 MSP430 的 Flash 的访问是受一个 256 位的密码的。更多的细节请查看 the MSP430 Programming Via the Bootstrap Loader User's Guide 这篇文章。

(3) 通过用户自定义的方法对 Flash 编程

MSP430 CPU 可以通过在线的和外部自定义编程的方法写自身的 Flash，如图 3.5.13 所示。用户可以选择通过 UART、SPI 等任何一种方式向 MSP430 提供数据。用户开发的软件也可以接收数据和编程 Flash。由于这种方法是用户开发的，所以完全可以通过自定义来满

足编程、擦除和更新 Flash 的要求。

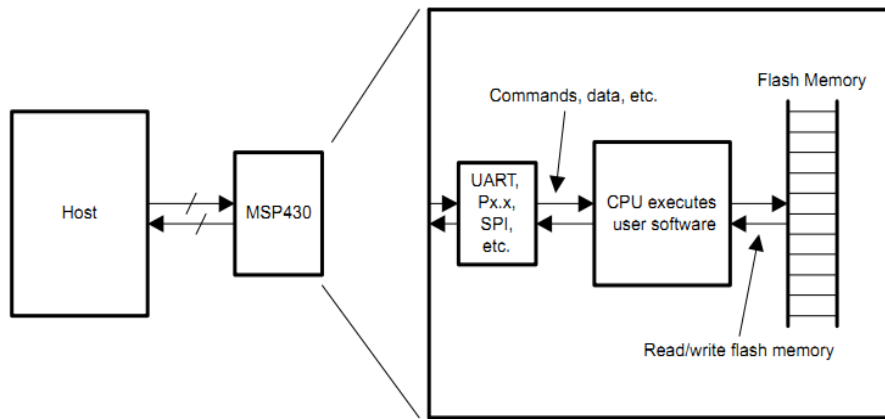


图 3.5.13 用户开发的编程方案

3.5.4 Flash 寄存器

Flash 寄存器如表 3.8 所示

表 3.8 Flash 寄存器

寄存器	缩写形式	寄存器类型	地址	初始状态
Flash 控制寄存器 1	FCTL1	可读/写	0x0128	0x9600 (上电清除时)
Flash 控制寄存器 2	FCTL2	可读/写	0x012A	0x9642 (上电清除时)
Flash 控制寄存器 3	FCTL3	可读/写	0x012C	0x9658 (上电清除时) (1)
中断使能 1	IE1	可读/写	0x0000	复位 (上电清除时)
中断标志 1	IFG1	可读/写	0x0002	

注：(1) KEYV 在上电复位 (POR) 时复位。

(2) 不是所有的器件中都有这个寄存器。请看具体器件的数据手册。

(1) Flash 控制寄存器 1——FCTL1

15	14	13	12	11	10	9	8
FRKEY, 读出值总为 096h FWKEY, 写入值必须为 0A5h							
7	6	5	4	3	2	1	0
BLKWRT	WRT	Reserved			MERAS	ERASE	Reserved
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

FRKEY: 第 15-8 位 FCTLx 密码。读出值总是 096h。

FWKEY: 写入值必须为 0A5h, 否则将产生一个上电清除 (PUC) 信号。

BLKWRT: 第 7 位 写块模式。写块模式时 WRT 也必须被置位。BLKWRT 在 EMEX 被置位时自动清 0。

0 写块模式关

1 写块模式开

WRT: 第 6 位 写模式。该位用于选择任意一种写模式。WRT 在 EEMEX 被置位时自动清 0。

0 写模式关闭

1 写模式开启

Reserved: 第 5 位 保留位。总是读出 0。

MERAS: 第 2 位 全部擦除和擦除。这些位一起被用来选择擦除模式

ERASE: 第 1 位 EMEX 被置位时，MERAS 和 ERASE 被自动清 0。

MERAS	ERASE	擦除周期
0	0	不擦除。
0	1	只擦除单段。
1	0	擦除主存储器中所有的段。
1	1	LOCKA=0: 擦除主 Flash 存储器和信息 Flash 存储器。 LOCKA=1: 只擦除主 Flash 存储器。

Reserved: 第 0 位 保留位。总是读为 0。

(2) Flash 控制寄存器 2——FCTL2

15	14	13	12	11	10	9	8
FWKEY , 读出值总为 096h 写入值必须为 0A5h							
7	6	5	4	3	2	1	0
FSSSELx	FNx						
rw-0	rw-1	rw-0	rw-0	rw-0	rw-0	rw-1	rw-0

FWKEY: 第 15-8 位 FCTLx 密码。读出值总为 096h。写入值必须为 0A5h 否则将产生一个上电清除 (PUC) 信号。

FSSSELx: 第 7-6 位 Flash 控制器时钟源选择

- 00 ACLK
- 01 MCLK
- 10 SMCLK
- 11 SMCLK

FNx: 第 5-0 位 Flash 控制器时钟分频器。这六位为 Flash 控制器时钟选择分频器。分频值为 $FNx+1$ 。例如，当 $FNx=00h$ ，分频值是 1。当 $FNx=03h$ ，分频值是 64。

(3) Flash 控制寄存器 3——FCTL3

15	14	13	12	11	10	9	8
FWKEYx , 读出值总为 096h 写入值必须 0A5h							
7	6	5	4	3	2	1	0
FAIL	LOCKA	EMEX	LOCK	WAIT	ACCVIFG	KEYV	BUSY
r(w)-0	r(w)-1	rw-0	rw-1	r-1	rw-0	rw-(0)	r(w)-0

FWKEYx: 第 15-8 位 FCTLx 密码。读出值总为 096h。写入值必须为 0A5h 否则将产生一个上电清除 (PUC) 信号。

FAIL: 第 7 位 操作失败位。该位在 f_{FTG} 时钟源失效、Flash 操作在 $EEIEX=1$ 时被中断终止的情况下被置位。FAIL 必须通过软件复位。

0 无失效

- 1 失效
- LOCKA:** 第 6 位 InfoA 段和信息锁。对该位写 1 将改变 InfoA 段和信息锁的状态。写 0 不产生影响。
- 0 在全部擦除主 Flash 区期间，InfoA 段开锁，所有的信息 Flash 区被擦除。
- 1 在全部擦除主 Flash 区期间，InfoA 段锁定，所有的信息 Flash 区受保护无法被擦除。
- EMEZ:** 第 5 位 紧急退出位。
- LOCK:** 第 4 位 锁定位。这个位可以使 Flash 开锁以进行写入和擦除操作。LOCK 位可以在一个写字节/字或者擦除操作和操作正常完成期间随时被置位。在写块模式下，如果 LOCK 被置位，且 BLKWRT=WAIT=1，那么 BLKWRT 和 WAIT 位将被清 0，写块模式将正常终止。
- 0 开锁
- 1 锁定
- WAIT:** 第 3 位 等待位。指示 Flash 正在被写入。
- 0 Flash 还未准备好被写入下一个字节/字
- 1 Flash 已经准备好被写入下一个字节/字
- ACCVIFG:** 第 2 位 访问冲突标志位
- 0 无中断挂起
- 1 中断挂起
- KEYV:** 第 1 位 Flash 安全密钥冲突。该位指示错误的 FCTLx 密码被写入到任意一个 Flash 控制寄存器中，在被置位时产生一个上电清除(PUC)信号。KEYV 必须通过软件清 0。
- 0 FCTLx 密码写入正确
- 1 FCTLx 密码写入不正确
- BUSY:** 第 0 位 忙位。这个位指示 Flash 时序信号发生器的状态。
- 0 时序信号发生器处于非忙状态
- 1 时序信号发生器处于忙状态

(4) 中断使能寄存器 1——IE1

7	6	5	4	3	2	1	0
		ACCVIE					

rw-0

第 7-6 位 这些位可以被用于其它的块。

- ACCVIE:** 第 5 位 Flash 访问冲突中断使能。该位将使能 ACCVIFG 中断。
- 0 中断不使能
- 1 中断使能

第 4-0 位 这些位可能被用于其它的块。

注：寄存器位的规约：

每一个寄存器的每一个单独的位都用一个键来指示该位的可访问性和初始值：

表 3.9 寄存器位的规约

键	位可访问性
rw	可读/写
r	只读
r0	读出值为 0
r1	读出值位 1
w	只写
w0	写入值为 0
w1	写入值为 1
(w)	无寄存器位执行；写入 1 将产生一个脉冲。寄存器位读出值为 0。
h0	硬件清 0
h1	硬件置位
-0, -1	上电清除 (PUC) 后的初始值
-(0), -(1)	上电复位 (POR) 后的初始值

下面是关于 Flash 使用的代码样例，其中包括了 Flash 初始化、段擦除、写字节、读字节、写字、读字、改字节和批量写等操作的子程序，读者可以在此基础上作适当修改，以用于不同的应用中或移植到其他系列的 msp430 单片机中。

```
#include "msp430g2553.h"
/*****
g2553 有 4 个数据段, 每个数据段有 64bytes, 共 256bytes
D:0x1000 -- 0x003F
C:0x1040 -- 0x107F
B:0x1080 -- 0x10BF
A:0x10C0 -- 0x10FF
*****/
#define uint unsigned int
#define uchar unsigned char
#define SegA 0x010C0
#define SegB 0x01080
#define SegC 0x01040
#define SegD 0x01000
#define SegSize 64
/*****Flash 初始化*****/
void FlashInit()
{
    FCTL2=FWKEY+FSSSEL_2+FN1;    //选择 SMLCK 作为时钟源, 二分频
}
/*****Flash 检测忙*****/
void FlashCheckBusy()
{
    while(BUSY==(FCTL3&BUSY));    //检测是否忙
}
```

```

}
/*****Flash 段擦除*****/
void FlashErase(int SegX)
{
    _DINT();          //关闭总中断
    FlashCheckBusy(); //检测 Flash 是否处于忙状态
    FCTL3=FWKEY;      //lock=0 开锁
    FCTL1=FWKEY+ERASE; //使能段擦除
    *((int *)SegX)=0x00; //段擦除--空写
    FlashCheckBusy(); //检测 Flash 是否处于忙状态
    FCTL3=FWKEY|LOCK; //上锁
    return;
}
/*****Flash 写字节*****/
void FlashWriteChar(uint addr, char wdata)
{
    _DINT();          //关闭总中断
    FlashCheckBusy(); //检测 Flash 是否处于忙状态
    FCTL3=FWKEY;      //lock=0 开锁
    FCTL1=FWKEY+WRT;  //写使能
    *((uchar *)addr)=wdata; //将 wdata 存入 addr 变量地址中
    FCTL1=FWKEY;      //写关闭
    FCTL3=FWKEY+LOCK; //上锁
    return;
}
/*****Flash 读字节*****/
char FlashReadChar(uint addr)
{
    char rdata;
    rdata=(char*)addr; //读取 addr 所指地址的值
    return rdata;
}
/*****Flash 写字*****/
void FlashWriteWord(uint addr, uint wdata)
{
    _DINT();          //关闭总中断
    FlashCheckBusy(); //检测忙, 若忙, 则等待
    FCTL3=FWKEY;      //lock=0 开锁

```



```

FCTL1=FWKEY+WRT; //写使能
*((uint *)addr)=wdata; //向地址 addr 处写入 wdata
FCTL1=FWKEY; //写关闭
FCTL3=FWKEY+LOCK; //上锁
return;
}
/*****Flash 读字*****/
uint FlashReadWord(uint addr)
{
    uint rdata;
    rdata=(uint *)addr; //读取变量 addr 地址的值
    return rdata;
}
/*****Flash 修改字节*****/
void FlashModifyChar(uint SegX, char AddrNum, char wdata)
{
    char i, TempArray[SegSize];
    for(i=0; i<SegSize; i++) //读入内存
    {
        TempArray[i]=*(uint *) (SegX+i);
    }
    TempArray[AddrNum]= wdata; //在数组中的某一位置 AddrNum 写入 wdata
    FlashErase(SegX); //段擦除
    FCTL3=FWKEY; //lock=0 开锁
    FCTL1=FWKEY+WRT; //准备写
    for(i=0; i<SegSize; i++) //向段中重新写数组
    {
        *(uint *) (SegX+i)=TempArray[i];
    }
    FCTL1=FWKEY; //写关闭
    FCTL3=FWKEY+LOCK; //上锁
}
/*****Flash 批量写*****/
void FlashBurstWrite(int SegX, int *pStr)
{
    int i;
    FlashErase(SegX); //段擦除
    FCTL3=FWKEY; //lock=0, 开锁

```

```

FCTL1=FWKEY+WRT;          //写使能
for(i=0;i<2*sizeof(pStr);i++) //将数组内容写入段中
{
    *(uchar *) (SegX+i)=*(pStr+i);
}
FCTL1=FWKEY;              //写关闭
FCTL3=FWKEY+LOCK;        //上锁
}
main()
{
    char ReadChar;
    uint ReadWord;
    int p[]={ 'a', 'b', 'c', 'd' };
    WDTCTL=WDTPW+WDTHOLD; //关闭看门狗
    P1DIR=0xff; //P1 口设为输出, 闲置的 I/O 不悬空
    P2DIR=0xff; //P2 口设为输出, 闲置的 I/O 不悬空
    P1OUT=0xff; //P1 口输出 1
    P2OUT=0xff; //P2 口输出 1
    FlashInit(); //Flash 初始化
    FlashErase(SegD);
    FlashWriteChar(0x01007, 0x12); //向地址 01008h 写入 12h
    ReadChar=FlashReadChar(0x01007); //读取地址 01008h 的值
    FlashWriteWord(0x01008, 0x3456); //向地址 01009h 和 0100Ah 依次写入 56h 和
                                     34h
    ReadWord=FlashReadWord(0x01008); //读取从地址 01009h 起的一个字
    FlashWriteChar(0x01017, ReadChar); //向地址 01018h 写入 12h
    FlashWriteWord(0x01018, ReadWord); //向地址 01019h 和 0101Ah 依次写入 56h 和
                                     34h
    FlashBurstWrite(SegC, p); //向 SegD 段从地址 0110h 依次写入 a、b、c、
                               d
    FlashModifyChar(SegB, 0x02, 0xef); //将地址 0112h 和 0113h 内容改为 e 和 f
    _BIS_SR(CPUOFF); //关闭 CPU
}

```

第六节 通信接口（USCI 和 USART）

串行通信接口是处理器与外界进行数据传输最常用的方式之一。顾名思义，串行通信是指使用一条数据线，将数据一位一位地依次传输，每一位数据占据一个固定的时间长度。与并行通信相比，串行通信速度较慢，但占用更少的 I/O 资源，只需要少数几条线就可以在系统间交换信息，特别适用于计算机与计算机、计算机与外设之间的远距离通信。

串行通信可以分为同步通信和异步通信两种类型。如果带有同步时钟，则称为同步串行通信，如常用的 SPI 和 I2C 接口就属于同步串行通信接口。如果没有同步时钟，依靠严格的时间间隔来传输每一位，则称为异步串行通信。MSP430 系列单片机有两种串行通信接口，较早的 USART 模块和较新的 USCI 模块。其中，1 系列和 4 系列单片机多为 USART 模块，而 2 系列、5 系列、6 系列和 4 系列较新的型号多配置 USCI 模块。各型号包含的模块个数为 0—4 个不等，如较为简单的单片机如 MSP430G2211 不含串行通信模块，此时可以用定时器软件模拟串口功能；MSP430F425 有一个串行通信模块；MSP430F149 有两个串行通信模块；MSP430F5438 有 4 个串行通信模块，等等。具体请参看选型手册。USART 又可配置为 UART、SPI 或 I2C 模式；USCI 中，USCI_Ax 支持 UART、LIN（波特率自检）、IrDA 编解码及 SPI 功能，USCI_Bx 支持 SPI 和 I2C 功能，具体将在以下章节详细介绍。

3.6.1 串行同步和串行异步通信原理的简述

1. 同步通信和异步通信

同步通信方式，是把许多字符组成一个信息组，这样，字符可以一个接一个地传输。但是，在每组信息（通常称为信息帧）的开始要加上同步字符，在没有信息要传输时，要填上空字符，因为同步传输不允许有间隙。同步方式下，发送方除了发送数据，还要传输同步时钟信号，信息传输的双方用同一个时钟信号确定传输过程中每一位的位置。见下图 3.6.1 所示。

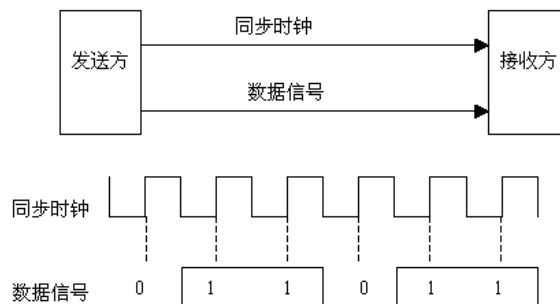


图 3.6.1 串行同步通信示意图

在异步通信方式中，两个数据字符之间的传输间隔是任意的，所以，每个数据字符的前后都要用一些数位来作为分隔位。

从图 3.6.1 中可以看到，按标准的异步通信数据格式（叫做异步通信帧格式），1 个字符在传输时，除了传输实际数据字符信息外，还要传输几个外加数位。具体说，在 1 个字符开始传输前，输出线必须在逻辑上处于“1”状态，这称为标识态。传输一开始，输出线由标识态变为“0”状态，从而作为起始位。起始位后面为 5~8 个信息位，信息位由低往高

排列，即先传字符的低位，后传字符的高位。信息位后面为校验位，校验位可以按奇校验设置，也可以按偶校验设置，或不设校验位。最后是逻辑的“1”作为停止位，停止位可为 1 位、1.5 位或者 2 位。如果传输完 1 个字符以后，立即传输下一个字符，那么，后一个字符的起始位便紧挨着前一个字符的停止位了，否则，输出线又会进入标识态。在异步通信方式中，发送和接收的双方必须约定相同的帧格式，否则会造成传输错误。在异步通信方式中，发送方只发送数据帧，不传输时钟，发送和接收双方必须约定相同的传输率。当然双方实际工作速率不可能绝对相等，但是只要误差不超过一定的限度，就不会造成传输错误。图 3.6.2 是异步通信时的标准数据格式。

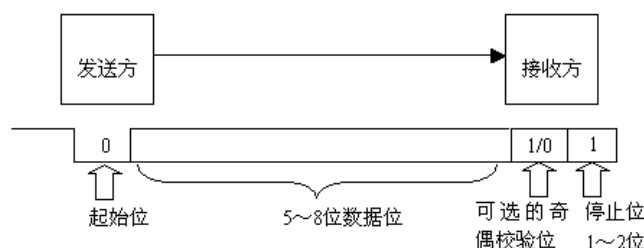


图 3.6.2 异步通信示意图

比较起来，在传输率相同时，同步通信方式下的信息有效率要比异步方式下的高，因为同步方式下的非数据信息比例比较小。

2. 传输率

所谓传输率就是指每秒传输多少位，传输率也常叫波特率。在计算机中，每秒传输多少位和波特率的含义是完全一致的，但是，在最初的定义上，每秒传输多少位和波特率是不同的，前者是指每秒钟传输的数位是多少，而波特率是指每秒钟传输的离散信号的数目。所谓离散信号，就是指不均匀的、不连续的也不相关的信号。在计算机里，只允许高电平和低电平两种离散信号，它们分别表示 1 和 0，于是，造成了波特率与每秒传输数位这两者的吻合。但在其他一些场合，就未必如此。比如，采用脉冲调制时，可以允许取 4 种相位，而每种相位代表 2 个数位，这种情况下，按每秒传输多少位 (bps) 计算的传输率便是波特率的两倍。国际上规定了一个标准波特率系列，标准波特率也是最常用的波特率，标准波特率系列为 110、300、600、1200、1800、2400、4800、9600、19200.....。

大多数接口的波特率可以通过编程来指定。

作为例子，我们可以考虑这样一个异步传输过程：设每个字符对应 1 个起始位、7 个数据位、1 个奇 / 偶校验位和 1 个停止位，如果波特率为 1200，那么，每秒钟能传输的最大字符数为 $1200 / 10 = 120$ 个。

作为比较，我们再来看一个同步传输的例子。假如也用 1200 的波特率工作，每个字符为 7 位，用 4 个同步字符作为信息帧头部，但不用奇 / 偶校验，那么，传输 100 个字符所用的时间为 $7 \times (100 + 4) / 1200 = 0.6067$ ，这就是说，每秒钟能传输的字符数可达到 $100 / 0.6067 = 165$ 个。

3. 异步通信的差错类型

异步通信过程中，可能发生通信错误，一般有 3 种错误：

1. 帧格式错：在应该接收到停止位的时候，接收到逻辑的“0”，便产生帧格式错误。

2. 奇偶错：接收到的奇偶校验位错。

3. 覆盖错：通信接口接收到数据并存放数据输入寄存器中，但是 CPU 没有及时来取，后面新接收的数据覆盖了前面收到的数据，叫做覆盖错。

发生帧格式错和奇偶错的原因可能为下面几种：

◆ 发送和接收双方采用了不同的传输率，或虽然双方约定了相同的传输率，但传输率不可能绝对相等。在通信的速率比较高的情况下，如果双方的传输率误差达到一定的程度，也会造成通信出错；

- ◆ 通信双方采用了不相同的帧格式；
- ◆ 干扰。

3.6.2 USCI 模块的相关寄存器定义

1. USCI 模块原理图

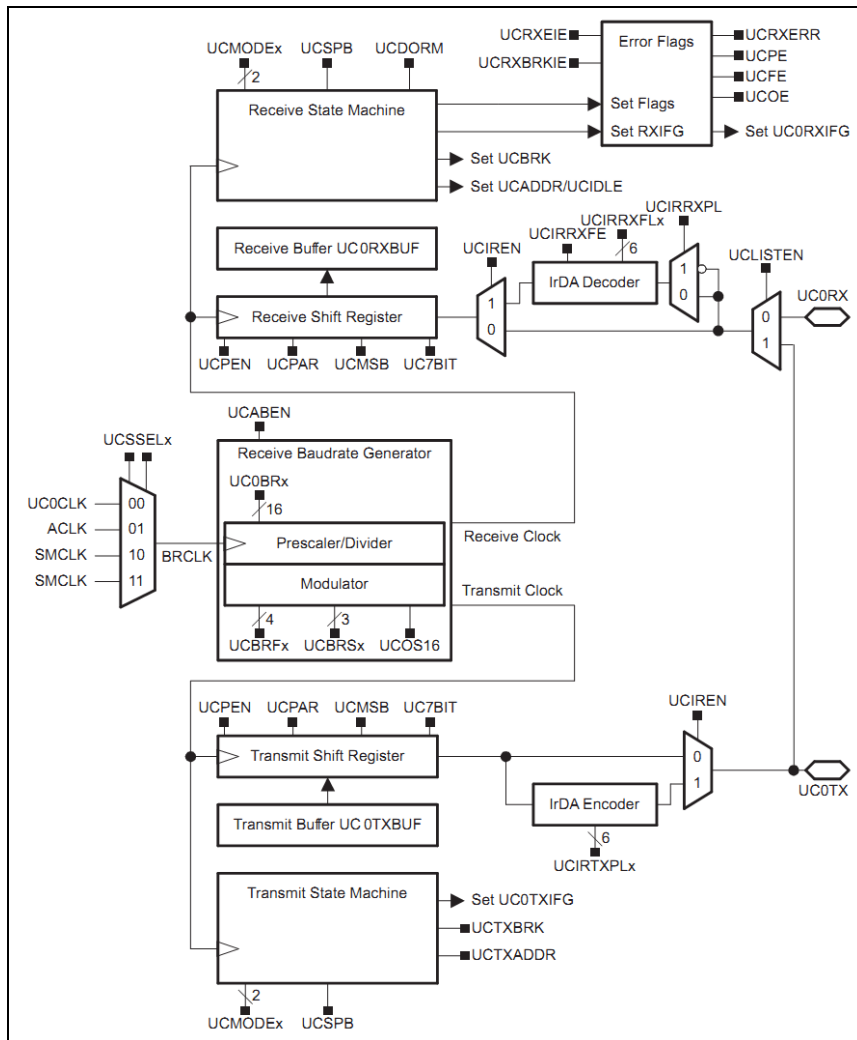


图 3.6.3 USCI_Ax 方框图: UART 模式 (UCSYNC = 0)

USCI_Ax 模块结构图 3.6.3 所示。该模块包含 4 个部分：

- ◆ 波特率部分：控制串行通信数据接收和发送的速度。
- ◆ 接收部分：接收串行输入的数据。
- ◆ 发送部分：发送串行输出的数据。

◆接口部分：完成并/串、串/并转换。

2. USCI 串行通信模块寄存器

硬件 USCI 方式可实现串行通信时，允许 7 或 8 位串行位流以预先编程的速率或外部时钟确定的速率输入、输出给 MSP430 单片机。用户对 USCI 的使用是通过对硬件原理和通信协议理解，在进行一系列寄存器设置之后，由硬件自动实现数据的输入输出。

USCI_x 分为 USCI_A_x 和 USCI_B_x，其中只有 USCI_A_x 可配置为 UART，MSP430 单片机中有的型号有两个通信模块 USCI0 和 USCI1，因此它们有两套寄存器，参见表 3.10 和表 3.11。USCI1 的中断相关寄存器有所不同，使用时请注意参照手册配置。

表 3.10 USCI_A0 控制和状态寄存器

Register	Short Form	Register Type	Address	Initial State
USCI_A0 control register 0	UCA0CTL0	Read/write	060h	Reset with PUC
USCI_A0 control register 1	UCA0CTL1	Read/write	061h	001h with PUC
USCI_A0 Baud rate control register 0	UCA0BR0	Read/write	062h	Reset with PUC
USCI_A0 baud rate control register 1	UCA0BR1	Read/write	063h	Reset with PUC
USCI_A0 modulation control register	UCA0MCTL	Read/write	064h	Reset with PUC
USCI_A0 status register	UCA0STAT	Read/write	065h	Reset with PUC
USCI_A0 receive buffer register	UCA0RXBUF	Read	066h	Reset with PUC
USCI_A0 transmit buffer register	UCA0TXBUF	Read/write	067h	Reset with PUC
USCI_A0 Auto baud control register	UCA0ABCTL	Read/write	05Dh	Reset with PUC
USCI_A0 IrDA transmit control register	UCA0IRTCTL	Read/write	05Eh	Reset with PUC
USCI_A0 IrDA receive control register	UCA0IRRCTL	Read/write	05Fh	Reset with PUC
SFR interrupt enable register 2	IE2	Read/write	001h	Reset with PUC
SFR interrupt flag register 2	IFG2	Read/write	003h	00Ah with PUC

表 3.11 USCI_A1 控制和状态寄存器

Register	Short Form	Register Type	Address	Initial State
USCI_A1 control register 0	UCA1CTL0	Read/write	0D0h	Reset with PUC
USCI_A1 control register 1	UCA1CTL1	Read/write	0D1h	001h with PUC
USCI_A1 baud rate control register 0	UCA1BR0	Read/write	0D2h	Reset with PUC
USCI_A1 baud rate control register 1	UCA1BR1	Read/write	0D3h	Reset with PUC
USCI_A1 modulation control register	UCA10MCTL	Read/write	0D4h	Reset with PUC
USCI_A1 status register	UCA1STAT	Read/write	0D5h	Reset with PUC
USCI_A1 receive buffer register	UCA1RXBUF	Read	0D6h	Reset with PUC
USCI_A1 transmit buffer register	UCA1TXBUF	Read/write	0D7h	Reset with PUC
USCI_A1 auto baud control register	UCA1ABCTL	Read/write	0CDh	Reset with PUC
USCI_A1 IrDA transmit control register	UCA1IRTCTL	Read/write	0CEh	Reset with PUC
USCI_A1 IrDA receive control register	UCA1IRRCTL	Read/write	0CFh	Reset with PUC
USCI_A1/B1 interrupt enable register	UC1IE	Read/write	006h	Reset with PUC
USCI_A1/B1 interrupt flag register	UC1IFG	Read/write	007h	00Ah with PUC

(1) UCAxCTL1, USCI_Ax 控制寄存器 0

7	6	5	4	3	2	1	0
UCPEN	UCPAR	UCMSB	UC7BIT	UCSPB	UCMODEx		UCSYNC
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

UCPEN: 第 7 位 校验允许位

- 0 校验禁止;
- 1 校验允许。

校验允许时，发送端发送校验，接收端接收该校验。多机模式中，地址位包含校验操作。

UCPAR: 第 6 位 奇偶校验选择位，该位在校验允许时有效

- 0 奇校验;
- 1 偶校验。

UCMSB: 第 5 位 大小端存储方式选择，控制发送和接收寄存器的存储方向。

- 0 低位在前
- 1 高位在前

UC7BIT: 第 4 位 数据长度，选择 7 位或 8 位数据长度。

- 0 8 位数据
- 1 7 位数据

UCSPB: 第 3 位 停止位选择，选择停止位位数。

- 0 1 位停止位
- 1 2 位停止位

UCMODEx: 第 2-1 位 USCI 模式，当 UCSYNC = 0 时，UCMODEx 位选择异步通信模式。

- 00 UART 模式;
- 01 线路空闲多机模式;
- 10 地址多机模式
- 11 带自动波特率检测的 UART 模式

UCSYNC: 第 0 为 同步模式使能

- 0 异步模式
- 1 同步模式

(2) UCAxCTL1, USCI_Ax 控制寄存器 1

7	6	5	4	3	2	1	0
UCSSELx	UCRXEIE	UCBRKIE	UCDORM	UCTXADDR	UCTXBRK	UCSWRST	
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-1

UCSSELx: 第 7-6 位 USCI 时钟源选择，这些位选取 BRCLK 的时钟源。

- 00 UCLK
- 01 ACLK
- 10 SMCLK
- 11 SMCLK

UCRXEIE: 第 5 位 接收数据错误中断允许

- 0 拒收错误字符，UCAxRXIFG 不置位;
- 1 接收错误字符并置位 UCAxRXIFG。

UCBRKIE: 第 4 位 接收暂停字符中断允许

- 0 接收暂停字符不置位 UCAxRXIFG;
- 1 接收暂停字符置位 UCAxRXIFG。

UCDORM: 第 3 位 休眠，令 USCI 进入睡眠状态。

- 0 不休眠，所有接收字节都置位 UCAxRXIFG；
- 1 休眠，只有前导为空闲线路的字节才置位 UCAxRXIFG。在带自动波特率检测的 UART 模式下只有同步场和间断的组合才能置位 UCAxRXIFG。

UCTXADDR: 第 2 位 发送地址，如果选择了多机模式，发送的下一帧将被标识为地址。

- 0 下一帧发送的是数据；
- 1 下一帧发送的是地址。

UCTXBRK: 第 1 位 发送隔断。通过向发送缓冲区写下一字节来产生隔断。在带自动波特率检测的 UART 模式下必须向 UCAxTXBUF 写 055h 来产生要求的隔断/同步区域。其他情况则要向发送缓冲写 0h。

- 0 下一帧不是隔断；
- 1 发送的下一帧是隔断或者是隔断/同步。

UCSWRST: 第 0 位 软件复位允许。

- 0 禁止 USCI 复位释放来允许操作；
- 1 允许 USCI 逻辑保持在复位状态。

(3) UCAxBR0, USCI_Ax 波特率控制寄存器

7	6	5	4	3	2	1	0
UCBRx							
rw	rw	rw	rw	rw	rw	rw	rw

(4) UCAxBR1, USCI_Ax 波特率控制寄存器 1

7	6	5	4	3	2	1	0
UCBRx							
rw	rw	rw	rw	rw	rw	rw	rw

UCBRx: 第 7-0 位 波特率发生器的时钟预分频设置。(UCAxBR0 + UCAxBR1 × 256) 这个 16 位的值构成预分频的值。

(5) UCAxMCTL, USCI_Ax 调制控制寄存器

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

UCBRFx: 第 7-4 位 第一调制阶段选择。当 UCOS16 = 1 时，这些位决定 BITCLK16 的调制模式。当 UCOS16 = 0 时这些位被忽略。

UCBRsX: 第 3-1 位 第二调制阶段选择。这些位决定 BITCLK 的调制模式。表 15-2 给出调制模式。

UCOS16: 第 0 位 过采样模式使能

- 0 禁止
- 1 使能

(6) UCAxSTAT, USCI_Ax 状态寄存器

7	6	5	4	3	2	1	0
UCLISTEN	UCFE	UCOE	UCPE	UCBRK	UCRXERR	UCADDR UCIDLE	UCBUSY
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	r-0

UCLISTEN: 第 7 位 侦听允许，UCLISTEN 位选择回路模式。

- 0 禁止；
- 1 允许，UCAxTXD 被从内部反馈到接收器。

UCFE: 第 6 位 帧差错标志

- 0 无错误;
- 1 字符在低停止位下被接收。

UCOE: 第 5 位 覆盖错误标志. , 当 UCAxRXBUF 中前一帧数据还未读出新的子节就被写入时这一位被置位。当 UCxRXBUF 被读后 , UCOE 会被软件自动清除, 否则, 它将无法正常工作。

- 0 无错误;
- 1 发生覆盖错误。

UCPE: 第 4 位 奇偶校验错误标志. 当 UCPEN = 0, UCPE 被读为 0.

- 0 无错误;
- 1 接收数据奇偶校验出错。

UCBRK: 第三位 暂停检测标志

- 0 没有暂停条件;
- 1 暂停条件发生。

UCRXERR: 第 2 位 接收错误标志位, 这一位表明接收数据出现错误。当 UCRXERR = 1, 一个或多个错误发生 (UCFE, UCPE, UCOE) 也被置位。当 UCAxRXBUF 被读取后 UCRXERR 被清除。

- 0 没有接收错误;
- 1 有接收错误。

UCADDR: 第 1 位 在地址位多机模式下地址被接收。

- 0 接收内容为数据
- 1 接收内容为一地址

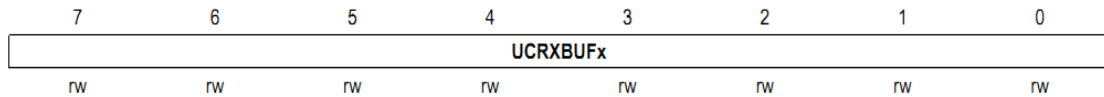
UCIDLE: 在空闲线路多机模式下的空闲线路检测状态

- 0 未检测到空闲线路
- 1 检测到空闲线路

UCBUSY: 第 0 位 USCI 忙标志. 这一位表明是否有发送或接收操作正在进行。

- 0 USCI 闲置
- 1 USCI 正在发送或接收

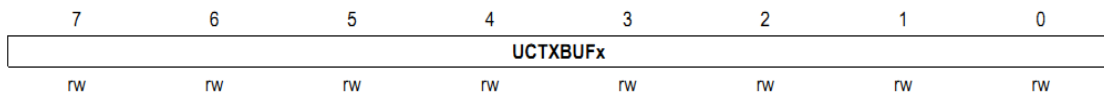
(7) UCAxRXBUF, USCI_Ax 接收缓冲寄存器



UCRXBUF_x: 第 7-0 位 接收缓存从接收移位寄存器最后接收的字符, 可由用户访问。

读取 UCAxRXBUF 会复位接收错误位, UCADDR、UCIDLE 和 UCAxRXIFG。在 7 位数据位模式下, UCAxRXBUF 是低位在前的。

(8) UCAxTXBUF, USCI_Ax 发送缓冲寄存器



UCTXBUF_x: 第 7-0 位 发送数据缓存使用户可访问的, 它可以保持数据直到数据被传送至发送移位寄存器, 然后由 UCAxTXD 传输。对发送缓存进行写操作可以复位 UCAxTXIFG , 在 7 位数据模式下 UCAxTXBUF

的 MSB 位没有使用并被复位。

(9) UCAxIRTCTL, USCI_Ax IrDA 发送控制寄存器

7	6	5	4	3	2	1	0
UCIRTXPLx						UCIRTXCLK	UCIREN
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

UCIRTXPLx: 第 7-2 位 发送脉冲宽度。脉冲宽度 $t_{PULSE} = (UCIRTXPLx + 1) / (2 \times f_{IRTXCLK})$

UCIRTXCLK: 第 1 位 IrDA 发送脉冲时钟选择

- 0 BRCLK
- 1 当 UCOS16 = 1, BITCLK16; 否则, BRCLK

UCIREN: 第 0 位 IrDA 编码/解码使能。

- 0 IrDA 编码/解码禁止
- 1 IrDA 编码/解码使能

(10) UCAxIRRCTL, USCI_Ax IrDA 接收控制寄存器

7	6	5	4	3	2	1	0
UCIRRXFLx						UCIRRXPL	UCIRRXFE
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

UCIRRXFLx: 第 7-2 位 接收脉冲宽度。接收最小脉宽由下式给出: $t_{MIN} = (UCIRRXFLx + 4) / (2 \times f_{IRTXCLK})$

UCIRRXPL: 第 1 位 IrDA 接收输入 UCAxRXD 极性

- 0 当接收到一个光脉冲时 IrDA 发送器传递一个正脉冲
- 1 当接收到一个光脉冲时 IrDA 发送器传递一个负脉冲

UCIRRXFE: 第 0 位 IrDA 接收滤波器使能。

- 0 接收滤波器禁止
- 1 接收滤波器使能

(11) UCAxABCTL, USCI_Ax 自动波特率控制寄存器

7	6	5	4	3	2	1	0
Reserved	UCDELIMx			UCSTOE	UCBTOE	Reserved	UCABDEN
r-0	r-0	rw-0	rw-0	rw-0	rw-0	r-0	rw-0

保留 第 7-6 位 保留

UCDELIMx: 第 5-4 位 间断/同步字符长度

- 00 1 位长度
- 01 2 位长度
- 10 3 位长度
- 11 4 位长度

UCSTOE: 第 3 位 同步字段超时错误

- 0 无错误
- 1 同步字段长度超过可测量的时间。

UCBTOE: 第 2 位 间隔字段超时错误

- 0 无错误

1 间隔字段长度超过 22 位次

保留 第 1 位 保留

UCABDEN: 第 0 位 自动波特率检测允许

0 波特率检测禁止，间隔和同步域长度不被测量。

1 波特率检测允许，间隔和同步域长度被测量并且相应的对波特率设置做出改变。

(12) IE2, 中断使能寄存器 2

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

未用 第 7-4 位 未用

第 3-2 位 这些位在其他的 USCI 模式下可能被使用。

UCA0TXIE: 第 1 位 USCI_A0 发送中断允许

0 中断禁止

1 中断允许

UCA0RXIE: 第 0 位 USCI_A0 接收中断允许

0 中断禁止

1 中断允许

(13) IFG2, 中断标志寄存器 2

7	6	5	4	3	2	1	0
						UCA0TXIFG	UCA0RXIFG
						rw-1	rw-0

未用 第 7-4 位 未用

第 3-2 位 这些位在其他的 USCI 模式下可能被使用。

UCA0TXIFG: 第 1 位 USCI_A0 发送中断标志位，当 UCA0TXBUF 为空时，UCA0TXIFG 被置位。

0 无中断挂起

1 中断挂起

UCA0RXIFG: 第 0 位 USCI_A0 接收中断标志，当 UCA0RXBUF 接收到一整帧数据，UCA0RXIFG 将被置位。

0 无中断挂起

1 中断挂起

(14) UC1IE, USCI_A1 中断使能寄存器

7	6	5	4	3	2	1	0
Unused						UCA1TXIE	UCA1RXIE
rw-0	rw-0	rw-0	rw-0			rw-0	rw-0

未用 第 7-4 位 未用

第 3-2 位 这些位在其他的 USCI 模式下可能被使用。

UCA1TXIE: 第 1 位 USCI_A1 发送中断允许

0 中断禁止

1 中断允许

UCA1RXIE: 第 0 位 USCI_A1 接收中断允许

0 中断禁止

1 中断允许

(15) UC1IFG, USCI_A1 中断标志寄存器

7	6	5	4	3	2	1	0
Unused						UCA1TXIFG	UCA1RXIFG
rw-0	rw-0	rw-0	rw-0			rw-1	rw-0

未用 第 7-4 位 未用

第 3-2 位 这些位在其他 USCI 模式下可能被使用。

UCA1TXIFG: 第 1 位 USCI_A1 发送中断标志位, 当 UCA1TXBUF 为空时, UCA1TXIFG 被置位。

0 无中断挂起

1 中断挂起

UCA1RXIFG: 第 0 位 USCI_A1 接收中断标志, 当 UCA1RXBUF 接收到一整帧数据, UCA1RXIFG 将被置位。

0 无中断挂起

1 中断挂起

3.6.3 USART 的相关寄存器定义

1. USART 模块原理图

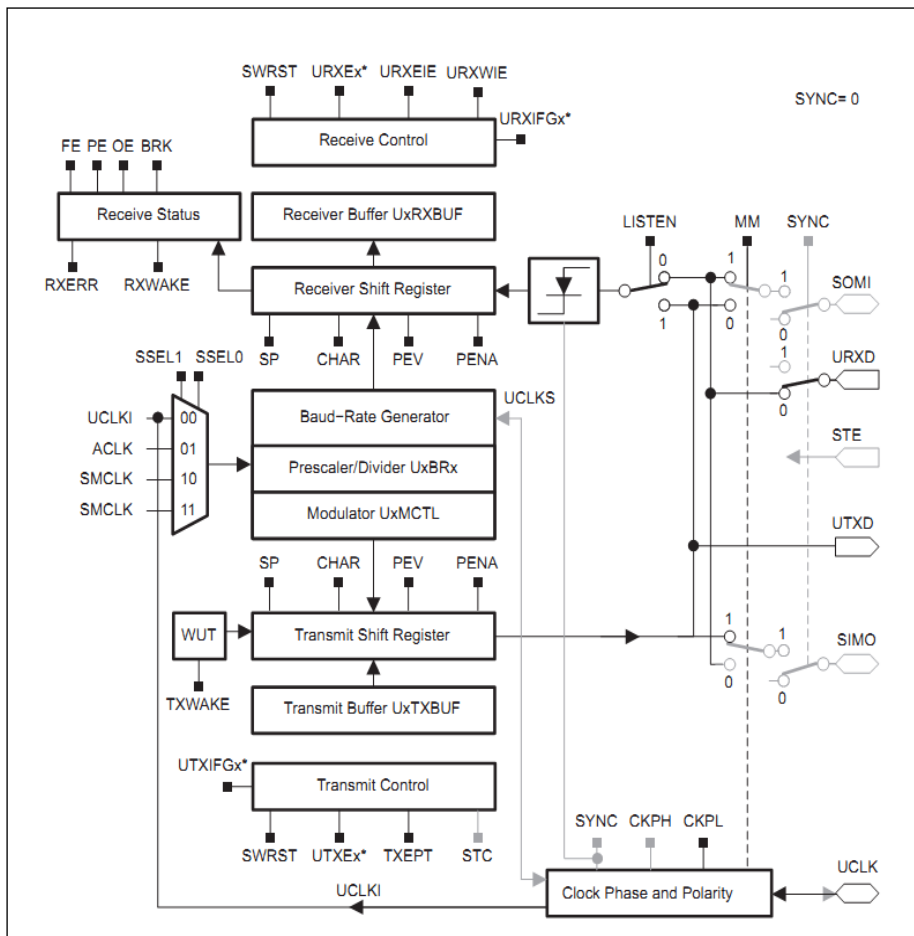


图 3.6.4 异步串行通信模块原理图

USART 模块结构图 3.6.4 所示。该模块包含 4 个部分：

- ◆波特率部分：控制串行通信数据接收和发送的速度。
- ◆接收部分：接收串行输入的数据。
- ◆发送部分：发送串行输出的数据。
- ◆接口部分：完成并/串、串/并转换。

2. USART 通信模块寄存器

硬件 USART 方式可实现串行通信时,允许 7 或 8 位串行位流以预先编程的速率或外部时钟确定的速率输入、输出给 MSP430 单片机。用户对 USART 的使用是通过对硬件原理和通信协议理解,在进行一系列寄存器设置之后,由硬件自动实现数据的输入输出。

MSP430 器件中有的型号有两个通信模块 USART0 和 USART1,因此它们有两套寄存器,参见表 3.12 和表 3.13。

表 3.12 串口 0 (USART0) 的控制寄存器

Register	Short Form	Register Type	Address	Initial State
USART control register	U0CTL	Read/write	070h	001h with PUC
Transmit control register	U0TCTL	Read/write	071h	001h with PUC
Receive control register	U0RCTL	Read/write	072h	000h with PUC
Modulation control register	U0MCTL	Read/write	073h	Unchanged
Baud rate control register 0	U0BR0	Read/write	074h	Unchanged
Baud rate control register 1	U0BR1	Read/write	075h	Unchanged
Receive buffer register	U0RXBUF	Read	076h	Unchanged
Transmit buffer register	U0TXBUF	Read/write	077h	Unchanged
SFR module enable register 1†	ME1	Read/write	004h	000h with PUC
SFR interrupt enable register 1†	IE1	Read/write	000h	000h with PUC
SFR interrupt flag register 1†	IFG1	Read/write	002h	082h with PUC

表 3.13 串口 1 (USART1) 的控制寄存器

Register	Short Form	Register Type	Address	Initial State
USART control register	U1CTL	Read/write	078h	001h with PUC
Transmit control register	U1TCTL	Read/write	079h	001h with PUC
Receive control register	U1RCTL	Read/write	07Ah	000h with PUC
Modulation control register	U1MCTL	Read/write	07Bh	Unchanged
Baud rate control register 0	U1BR0	Read/write	07Ch	Unchanged
Baud rate control register 1	U1BR1	Read/write	07Dh	Unchanged
Receive buffer register	U1RXBUF	Read	07Eh	Unchanged
Transmit buffer register	U1TXBUF	Read/write	07Fh	Unchanged
SFR module enable register 2	ME2	Read/write	005h	000h with PUC
SFR interrupt enable register 2	IE2	Read/write	001h	000h with PUC
SFR interrupt flag register 2	IFG2	Read/write	003h	020h with PUC

注：两套串口相互独立，且寄存器结构相同，这里我们仅介绍一套串口，用 U_x 表示

(1) U_xCTL 串口控制寄存器

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

PENA: 校验允许位

- 0 校验禁止;
- 1 校验允许。

校验允许时，发送端发送校验，接收端接收该校验。地址位多机模式中，地址位包含校验操作。

PEV: 奇偶位校验位，该位在校验允许时有效

- 0 奇校验;
- 1 偶校验。

SPB: 停止位选择。决定发送的停止位数，但接收时接收器只检测 1 位停止位。

- 0 1 位停止位
- 1 2 位停止位。

CHAR: 字符长度

- 0 7 位
- 1 8 位。

LISTEN: 反馈选择。选择是否讲发送数据由内部反馈给接收器

- 0 无反馈
- 1 有反馈，发送信号由内部反馈给接收器。

SYNC: USART 模块的模式选择

- 0 UART 模式（异步）
- 1 SPI 模式（同步）。

MM: 多机模式选择位

- 0 线路空闲多机协议;
- 1 地址多机协议。

SWRST: 控制位

该位的状态影响着其他一些控制位和状态位的状态。在串行口的使用过程中，这一位是比较重要的控制位。一次正确的 USART 模块初始化应该是这样的顺序：先在 SWRST=1 情况下设置串行口；然后设置 SWRST=0；最后如果需要中断，则设置相应的中断使能。

(2) UxTCTL 串口发送寄存器

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

CKPL: 时钟极性控制位

- 0 UCLKI 信号与 UCLK 信号极性相同;
- 1 UCLKI 信号与 UCLK 信号极性相反;

SSELx: 时钟源选择位

这两位确定波特率发生器的时钟源

- 0 外部时钟 UCLKI。
- 1 辅助时钟 ACLK。
- 2 子系统时钟 SMCLK。

3 子系统时钟 SMCLK。

URXSE: 接收触发沿控制位

- 0 没有接收触发沿检测；
- 1 有接收触发沿检测。

TXWAKE: 传输唤醒控制

- 0 下一个要传输的字符为数据
- 1 下一个要传输的字符为地址。

TXEPT: 发送器空标志

在异步模式与同步模式时不一样

- 0 正在传输数据或者发送缓冲器 (UTXBUF) 有数据；
- 1 表示发送移位寄存器和 UTXBUF 空或者 SWRST=1。

(3) UxRCTL 串口接收寄存器

7	6	5	4	3	2	1	0
FE	PE	OE	BRK	URXEIE	URXWIE	RXWAKE	RXERR

FE: 帧错误标志

- 0 没有帧错误；
- 1 帧错误

PE: 校验错误标志位

- 0 校验正确；
- 1 校验错误。

OE: 溢出标志位

- 0 无溢出
- 1 有溢出。

BRK: 打断检测位

- 0 没有被打断；
- 1 被打断。

URXEIE: 接收出错中断允许位

- 0 不允许中断，不接收出错字符并且不改变 URXIFG 标志位；
- 1 允许中断，出错字符接收并且能够置位 URXIFG。

URXWIE: 接收唤醒中断允许位

当接收到地址字符时，该位能够置位 URXIFG，当 URXEIE=0, 如果接收内容有错误，该位不能置位 URXIFG。

- 0 所有接收的字符能够置位 URXIFG；
- 1 只有接收到地址字符才能置位 URXIFG。

RXWAKE: 接收唤醒检测位

在地址位多机模式，接收字符地址位置位时，该机被唤醒，在线路空闲多机模式，在接收到字符前检测到 URXD 线路空闲时，该为被唤醒，RXWAKE 置位。

- 0 没有被唤醒，接收到的字符是数据；
- 1 唤醒，接收的字符是地址。

RXERR: 接收错误标志位

- 0 没有接收错误;
- 1 有接收错误。

(4) UxBRO 波特率控制寄存器 0

7	6	5	4	3	2	1	0
27	26	25	24	23	22	21	20

(5) UxBR1 波特率控制寄存器 1

7	6	5	4	3	2	1	0
215	214	213	212	211	210	29	28

UxBRO 和 UxBR1 两个寄存器用于存放波特率分频因子的整数部分。

其中 UxBRO 位低字节，UXBR1 为高字节。两字节和起来为一个 16 位字，成为 UBR。在异步通信时，UBR 的允许值不小于 3。如果 UBR<3，则接收和发送会发生不可预测的错误。

(6) UxMCTL 波特率调整寄存器

7	6	5	4	3	2	1	0
m7	m6	m5	m4	m3	m2	m1	m0

如果波特率发生器的输入频率 BRCLK 不是所需的波特率的整数倍，带有一小数，则整数部分写入 UBR 寄存器，小数部分由调整控制寄存器 UxCTL 的内容反映。波特率由以下公式计算：

$$\text{波特率} = \text{BRCLK} / (\text{UBR} + (\text{M7} + \text{M6} + \dots + \text{M0}) / 8)$$

其中 M0, M1, …, M6 及 M7 为控制器 UxMCTL 中的各位。调整寄存器的 8 为分别对应 8 次分频，如果 M=1，则相应次的分频增加一个时钟周期；如果 Mi=0，则分频计数器不变。

(7) UxRXBUF 串口接收缓冲寄存器

7	6	5	4	3	2	1	0
27	26	25	24	23	22	21	20

接收缓存从接收移位寄存器最后接收的字符，可由用户访问。

当接收和控制条件为真时，接收缓存装入当前接收到的字符，如下表所列：

条件		结果			
URXEIE	URXWIE	装入 URXBUF	PE	PE	BRK
0	1	无差错地址字符	0	0	0
1	1	所有地址字符	X	X	X
0	0	无差错字符	0	0	0
1	0	所有字符	X	X	X

(8) UxTXBUF 串口发送缓冲寄存器

7	6	5	4	3	2	1	0
27	26	25	24	23	22	21	20

发送缓存内容可以传送至发送移位寄存器，然后有 UxTXDx 传输。对发送缓存进行写操作可以复位 UxTXIFGx。

(9) ME1 ME2 串口模块控制寄存器

ME1 模块允许寄存器 1

7	6	5	4	3	2	1	0
UTXE0	URXE0						

ME2 模块允许寄存器 2

7	6	5	4	3	2	1	0
		UTXE1	URXE1				

UTXE0: 串口 0 的发送允许

URXE0: 串口 0 的接收允许

UTXE1: 串口 1 的发送允许

URXE1: 串口 1 的接收允许

0 禁止

1 允许

(10) IFG1 IFG2 串口中断标志控制寄存器

IFG1 中断标志寄存器 1

7	6	5	4	3	2	1	0
UTXIFG0	URXIFG0						

IFG2 中断标志寄存器 2

7	6	5	4	3	2	1	0
		UTXIFG1	URXIFG1				

UTXIFG0: 串口 0 的发送中断标志

URXIFG0: 串口 0 的接收中断标志

UTXIFG1: 串口 1 的发送中断标志

URXIFG1: 串口 1 的接收中断标志

0 无中断请求标志

1 有中断请求标志

(11) IE1 IE2 串口中断控制寄存器

IE1 中断控制寄存器 1

7	6	5	4	3	2	1	0
UTXIE0	URXIE0						

IE2 中断控制寄存器 2

7	6	5	4	3	2	1	0
		UTXIE1	URXIE1				

UTXIE0: 串口 0 的发送中断允许

URXIE0: 串口 0 的接收中断允许

UTXIE1: 串口 1 的发送中断允许

URXIE1: 串口 1 的接收中断允许

0 中断请求禁止

1 中断请求允许

3.6.4 USCI 模块初始化和收发操作步骤流程

1. USCI 模块的串行异步通信操作

USCI_Ax 在某一特定波特率下异步的发送和接收数据。每一字符的发送或接收所用时间与 USCI 选取的波特率有关，发送和接收功能使用相同的波特率。

在异步通信模式下, MSP430 单片机通过两个引脚, 即接收引脚 UCAxRXD 和发送引脚 UCAxTXD 与外界相连。当 UCSYNC 位被清除时, UART 模式被选择。

异步通信的特点如下:

- ◆ 传输 7 位或 8 位数据, 可采用奇校验或偶校验或者无校验。
- ◆ 两个独立移位寄存器: 输入移位寄存器和输出移位寄存器。
- ◆ 独立的发送和接收缓冲寄存器。
- ◆ 从低位在前或高位在前的数据发送和接收。
- ◆ 对多机系统内置有线路空闲/地址位通信协议。
- ◆ 通过有效的起始位检测将 MSP430 从低功耗唤醒。
- ◆ 可编程实现分频因子为整数或小数的波特率。
- ◆ 状态标志检测和抑制错误。
- ◆ 状态标志检测地址位。
- ◆ 独立的发送和接收中断能力。

(1) USCI 初始化和复位

USCI 在 PUC 后或者通过设置 UCSWRST 位。在 PUC 后, UCSWRST 位自动置位, 保持 USCI 在复位状态。当置位时, UCSWRST 位复位 UCAXRXIE, UCAXTXIE, UCAXRXIFG, UCRXERR, UCBRK, UCPE, UCOE, UCFE, UCSTOE 和 UCBTOE 位并置 UCAXTXIFG 位。清除 UCSWRST 位释放 USCI 的操作。

注解: 初始化或者从新配置 USCI 模块, 推荐的 USCI 初始化/重配置的过程如下:

- a. 设置 UCSWRST
- b. UCSWRST=1 时初始化所有的 USCI 寄存器 (包括 UCAXCTL1)
- c. 配置端口
- d. 软件清除 UCSWRST
- e. 通过 UCAXRXIE 和/或 UCAXTXIE 允许中断 (可选)

(2) 字符格式

UART 的字符格式展示在图 3.6.5 中, 包括一个开始位, 7 或 8 个数据位, 一个奇/偶/无校验位, 一个地址位 (地址位模式), 一个或两个停止位。UCMSB 位控制传输的方向以及选择 LSB 或 MSB 的优先。必须满足 UART 通信才能 LSB-FIRST。

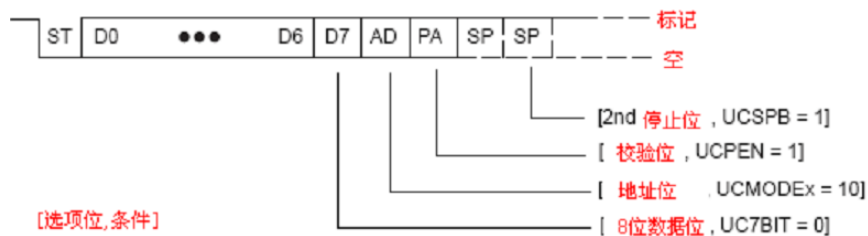


图 3.6.5 字符格式

(3) 异步通信格式

当两个设备异步通信时, 多机的模式是协议所要求的。当三个或更多的设备通信时, USCI 支持空闲线和地址位多机的通信格式。

空闲线路多机模式:

当 UCMODEX=01 时，空闲线路多机模式被选中。块数据在发送和接收线上被一段空闲时间隔开，如图 3.6.6 所示。当 10 个或更多的持续标志被在一个或两个字符的停止位之后接收到时一条空闲接收线被监测。在接收到一次空闲线之后波特率发生器被切断，直到下一次开始边沿被监测到。

当一次空闲线被监测到时，UCIDLE 位被表示为多机形式，UCDORM 位被用来控制数据接收。当 UCDORM=1 时，所有的非地址字符被组装但不会传输到 UCAXRXBUF 中，此时中断不会发生。当接收一个地址字符时，这个字符被传输到 UCAXRXBUF 中，UCAXRXIFG 被置位，当 UCRXEIE=1 时任何可用的错误标志被置位。当 UCRXEIE=1 时并且一个带有帧错误或奇偶错误的地址字符被接收到时，字符就不会传输到 UCAXRXBUF 中，而且 UCAXRXIFG 不会置位。

如果接收到一个地址，使用者需软件确认此地址。同时，为继续接收数据必须复位 UCDORM。如果 UCDORM 保持置位就只能接收到地址字符。在接收字符的过程中一旦 UCDORM 被清除，接收中断标志将在接收完成后被置位，UCDORM 位不会被 USCI 的硬件自动改变。在空闲线多机形式进行地址传输，为了在 UCAXTXD 上发生地址字符确认，一个精确的空闲周期会被 USCI 产生。如果下一个字符先于 11 位的空闲线被装载到 UCAXTXBUF 中，这通过双缓存 UCTXADDR 标志展示出来，当开始位发生时，UCTXADDR 将被自动清除。

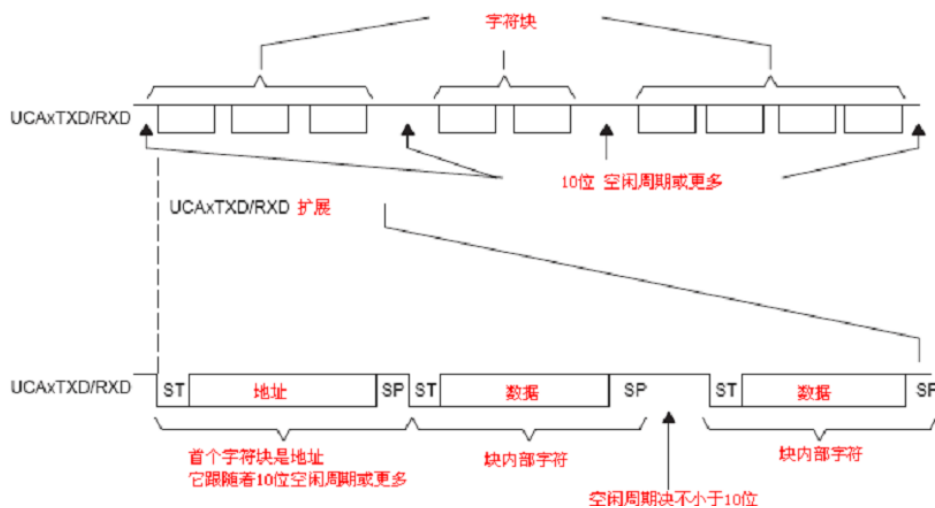


图 3.6.6 空闲线路格式

发送空闲帧：

以下操作为发送一个空闲帧来展示一个地址字符及其关联的数据：

①置位 UCTXADDR，然后写地址字符到 UCAXTXBUF 中，UCAXTXBUF 必须准备发送新数据 (UCAXTXIFG=1)。这可以产生一个 11 位的空闲周期以及随后的地址字符。UCTXADDR 在地址字符从 UCAXTXBUF 传输到移位寄存器中后自动复位。

②写预期数据到 UCAXTXBUF 中，UCAXTXBUF 必须准备发送新数据 (UCAXTXIFG=1)。写到 UCAXTXBUF 中的数据传输到移位寄存器中并且一旦移位寄存器准备新数据就开始发送。空闲周期不能超过地址和数据传输的间隙或者数据和数据传输的间隙，否则传输的数据将被误解为地址。

地址位多机模式：

当 UCMODEX=10，地址位多机模式被选中。如图 3.6.7 所示，每个待处理的字符包含一

个用作地址指示的位。字符块的第一个字符带有一个设置指示字符地址的地址位。USCI 的 UCADDR 位在包含地址位的字符中，被传输到 UCAXRXBUF 中时即置位。

在地址位多机模式中 UCDORM 位被用来控制数据接收。当 UCDORM 置位时，地址位为 0 的字符数据被接收器组装起来但不传输到 UCAXRXBUF 中，同时没有中断发生。当接收一个包含已置地址位的字符时，此字符被传输到 UCAXRXBUF 中，UCAXRXIFG 被置位，同时当 UCRXEIE=1 时所有可用错误标志被置位。当 UCRXEIE=1 时接收一个包含已置地址位的字符，但是没有帧错误和极性错误发生，此时这个字符不会传输到 UCAXRXBUF 中而且 UCAXRXIFG 不会被置位。

如果接收到一个地址，用户需要软件使地址生效同时复位 UCDORM 去继续接收数据。如果 UCDORM 保持置位，仅仅是已置地址位的地址字符被接收到。UCDORM 位不会被 USCI 模块自动改变。

当 UCDORM=1 时所有已接收的字符将置位中断标志 UCAXRXIFG。如果在接收数据期间清除 UCDORM 接收中断标志将在接收完成之后置位。

在地址位多处理模式的地址传输中，字符的地址位被 UCTXADDR 位控制，UCTXADDR 的值装载到字符的地址位后从 UCAXTXBUF 传送到发送移位寄存器中。当开始位发生时 UCTXADDR 被自动清除。

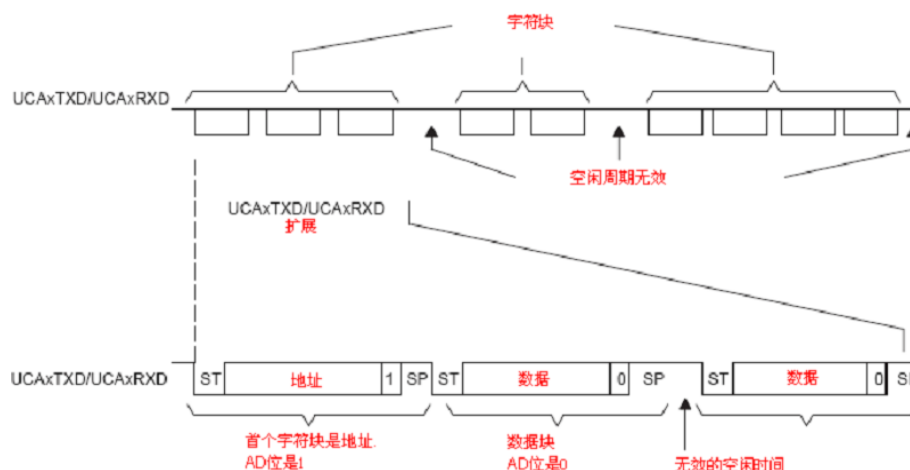


图 3.6.7 地址位多机格式

暂停接收的产生:

当 UCMODEX=00, 01 或 10 时，不管是奇偶位，地址模式还是字符设置，接收器在监测到一次暂停时数据位，奇偶位，停止位都要变低。当一次暂停被监测到时，UCBRK 位置位。如果暂停中断允许位 UCBRKIE 置位，接收中断标志 UCAXRXIFG 也会置位。这样的话，UCAXRXBUF 的值在所有数据位为 0 后变成 0。

设置 UCTXBRK 位来发生暂停，然后写 0 到 UCAXTXBUF 中，UCAXTXBUF 必须准备新的数据 (UCAXTXIFG=1)。这就会发生暂停，同时所有位变低。当开始位发生时，UCTXBRK 自动清除。

(4) 自动波特率检测

当 UCMODEX=11 时 UART 模式的自动波特率监测被选中。在这种情况下，数据帧在一个包含暂停和异步域的异步序列之后。当 11 个或更多的 0 被接收到时监测到暂停。如果暂停

时间超过 22 位的传输时间暂停超时错误标志 UCSTOE 将被置。暂停的异步域如图 3.6.8 所示。

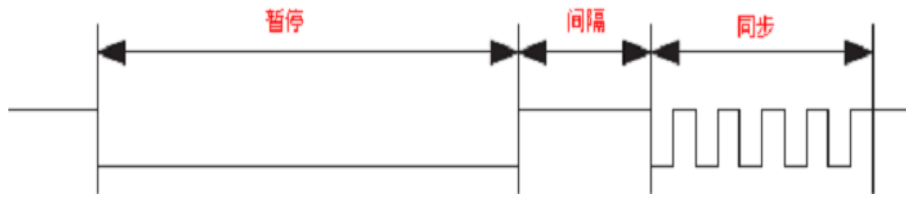


图 3.6.8 自动波特率监测-暂停/同步序列

为了 LIN 一致，字符格式应该设置为 8 位数据位，LSB 在前，无奇偶校验位和停止位。没有可用的地址位。

在一个字节域内同步域所包含的数据 055H 如图 3.6.9 所示。同步的时间范围在第一个下降沿和最后一个下降沿之间。如果自动波特率监测通过置位 UCABDEN 而被允许发送波特率发生器就可以用来测量。否则，这个模式只被接收而不测量。测量的结果被传送到波特率控制寄存器 UCAXBRO，UCAXBRI 和 UCAXMCTL 中。如果同步域的长度超过了测量时间同步超时错误标志 UCSTOE 将置位。

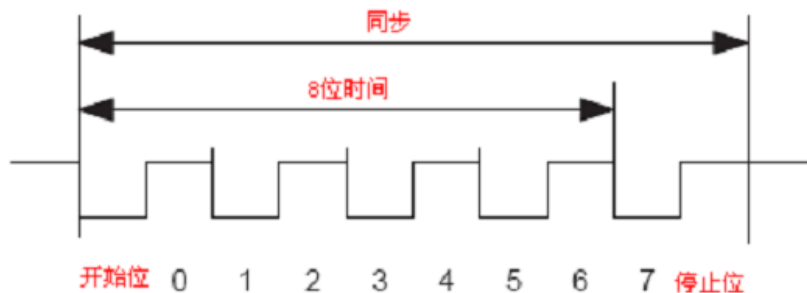


图 3.6.9 自动波特率监测-同步域

在这种模式中 UCDORM 位被用来控制数据接收。当 UCDORM 置位时，所有字符被接收而不传输到 UCAXRXBUF 中去，而且中断不发生。当一个暂停/同步域被监测到时 UCBRK 标志被置位。随后的暂停/同步域的字符被传送到 UCAXRXBUF 中同时 UCAXRXIFG 中断标志置位。任何可用的错误标志也会置位。如果 UCBRKIE 位置位，暂停/同步的接收置位 UCAXRXIFG。通过读接收缓存 UCAXRXBUF，UCBRK 位被用户软件置位。

当一个暂停/同步域被接收时，为继续接收数据用户必须软件置位 UCDORM。如果 UCDORM 保持置位状态，仅仅在接受下一个暂停/同步域时字符才能被接收。UCDORM 位不会被 USCI 硬件自动更改。

当 UCDORM=0 时所有的接收字符将使接受中断标志 UCAXRXIFG 置位。如果在接收字符的过程中 UCDORM 被清除，接收中断标志在完成接受后将置位。

发送暂停/同步域:

以下为发送一个暂停/同步域的程序流程

- ①置位 UCTXBRK 且 UMODEX=11。
- ②写 0x55 到 UCAXTXBUF。UCAXTXBUF 必须准备新数据 (UCAXTXIFG=1)。

伴随着中断分割符和同步字符将会产生一个 13 位的暂停域。暂停域的长度被 UCDELIMX 位控制。当同步字符从 UCAXTXBUF 传输到移位寄存器中时 UCTXBRK 将自动复位。

③写数据字符到 UCAXTXBUF 中。UCAXTXBUF 必须准备新数据 (UCAXTXIFG=1) 。
 写到 UCAXTXBUF 中的数据传输到移位寄存器中，一旦移位寄存器有新数据就立即发送。

(5) IrDA 编码和解码

当 UCIREN 置位，IrDA 解码器和译码器被允许，同时提供修整 IrDA 通信的硬件位，IrDA 译码。

IrDA 编码：

如图 3.6.10，在来自 UART 的发送字节流中，译码器为每个 0 位发送一个脉冲，脉冲持续时间由 UCIRTXPLX 位决定。UCIRTXPLX 位指定被 UCIRTXCLK 选中的时钟周期的数目。

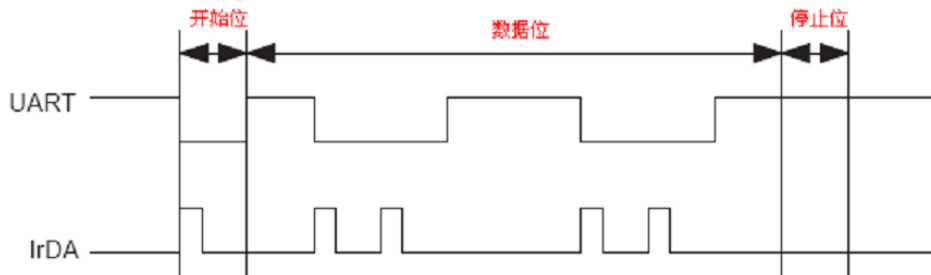


图 3.6.10 UART VS IRDA 数据格式

由 IrDA 标准要求去设置 3/16 位的周期脉冲，通过置 UCIRTXCLK=1 选中 BITCLK16 时钟，同时脉冲长度由 UCIRTXPLX=6-1=5 来设置 6 个半时钟循环。

当 UCIRTXCLK=0，基于 BRCLK 的脉冲长度 T_{pulse} 由下面的公式计算：

$$UCIRTXPLX = t_{PULSE} \times 2 \times f_{BRCLK} - 1$$

当 UCIRTXCLK=0 时计数器 UCBRX 必须被设置成一个大于或等于 5 的值。

IrDA 解码：

当 UCIRRXPL=0 时，解码器监测到高脉冲。否则就监测到低脉冲。除模拟抗尖峰脉冲滤波器外，可编程数字滤波器也能通过设置 UCIRRXFE 被允许。当 UCIRRXFE 置位，只有比可编程滤波器长的脉冲能通过。短脉冲被丢弃。可编程滤波器长度 UCIRRXFLX 的方程式如下：

$$UCIRRXFLX = (t_{PULSE} - t_{WAKE}) \times 2 \times f_{BRCLK} - 4$$

这里：

T_{pulse} ：接受脉冲的最小宽度；

T_{wake} ：从任何低功耗中唤醒。当 MSP430 单片机处于活动模式时为 0。

(6) 自动错误检测

干扰脉冲抑制会阻止 USCI 的突然启动。在 UCAXRXD 上的任何小于抗尖峰脉冲时间 T_t (大约 150NS) 的脉冲都被忽略。参见设备数据手册的参数。

当在 UCAXRXD 上有超过 T_t 的低脉冲多数情况都会被当开始位。如果多数情况没有监测到一个有效的开始位 USCI 将停止接收字符同时等待 UCAXRXD 上的下一个低电平。为阻止错误这种方法也用在每一位数据的接收上。

在接收字符过程中 USCI 模块自动监测帧错误，奇偶校验错误，溢出错误以及暂停条件。

在相应的情况被监测到时 UCFE, UCPE, UCOE 以及 UCBRK 等位置位。当 UCFE, UCPE 和 UCOE 等错误标志置位时, UCRXERR 也置位。错误条件在表 3.14 中描述。

表 3.14 接收错误情况

错误情况	错误标志	描述
帧错误	UCFE	当一个低停止位被监测到时此错误发生。当使用两个停止位, 两位都会被检查是否有帧错误。当检测到帧错误时, UCFE 置位。
奇偶校验错误	UCPE	一个奇偶校验错误是一次字符中校验位的实际值和设置值的不匹配。包括在奇偶位中的计算。当一次错误被监测到时, UCPE 置位。
接收溢出	UCOE	在读前一个字符之前另一个字符被装载到 UCAXRXBUF 中引发一次溢出错误。当溢出错误发生时, UCOE 置位
暂停情况	UCBRK	当不使用自动波特率监测时, 在所有的数据, 校验位和停止位为低时暂停被监测到。同时 UCBRK 置位。如果暂停中断允许位 UCBRKIE 置位暂停发生时还会置位其中断标志 UCAXRXIFG。

当 UCRXEIE=0 时以及帧错误或奇偶校验错误被监测到时, 不会有字符被接收到 UCAXRXBUF 中。当 UCRXEIE=1 时, 字符被接收到 UCAXRXBUF 中, 同时相应的错误位置位。

当 UCFE, UCPE, UCOE, UCBRK 或 UCRXEER 置位时, 其状态保持到用户软件复位或 UCAXRXBUF 中的数据被读出。UCOE 必须通过读 UCAXRXBUF 复位, 否则它不会起作用。

(7) USCI 接收使能

USCI 模块通过清除 UCSWRST 位, 接收器做好准备和在空闲状态下被允许。接收波特率发生器保持在准备状态但不计时, 不产生任何时钟。

开始位的下降沿允许波特率发生器同时 UART 状态机器检查一个有效的开始位, 如果没有监测到有效的开始位 UART 机器状态返回到空闲态同时波特率发生器再次被关掉; 如果一个有效的开始位被监测到字符就能被接收。

当 UCMODEX=01 空闲线多机模式被选中, UART 机器状态在接收一个字符后检查空闲线。如果监测到开始位, 另一个字符会被接收。否则在接收 10 个字符后 UCIDLE 标志置位, 同时 UART 的机器状态返回到空闲态并且波特率发生器被关掉。

接收数据干扰脉冲抑制: 干扰脉冲抑制会阻止 USCI 的突然启动。在 UCAXRXD 上的任何小于抗尖峰脉冲时间 T_t (大约 150NS) 的脉冲都被忽略, 同时进一步的情况会发生, 如图 3.6.11 所示。参见设备数据手册的参数。

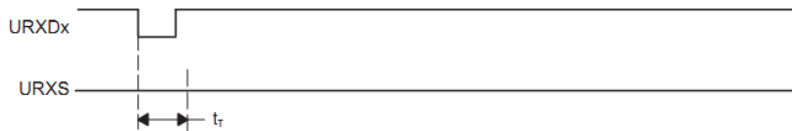


图 3.6.11 干扰脉冲抑制, USCI 不开始接收

当干扰脉冲时间大于 T_t 时, 或者一个有效的开始发生在 UCAXRXD 上。USCI 开始接收操作而且多数情况在图 3.6.12 中展示。如果多数情况没有监测到一个有效的开始位 USCI 将停止接收字符。

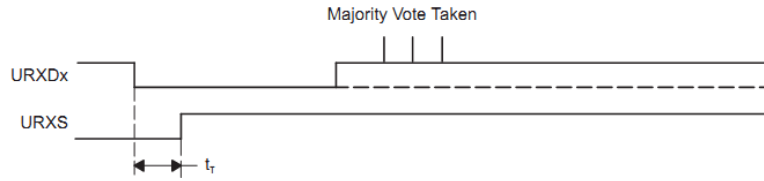


图 3.6.12 干扰脉冲抑制，USCI 启动

(8) USCI 发送使能

USCI 模块通过清除 UCSWRST 位，接收器做好准备和在空闲状态下被允许。接收波特率发生器保持在准备状态但是不计时，不产生任何时钟。

一次传输通过写数据到 UCAXTXBUF 中初始化。当初始化发生后，波特率发生器被允许。同时，在发送移位寄存器为空后的下一个 BITCLK 周期 UCAXTXBUF 中的数据被移到发送移位寄存器，当新数据被写进 UCAXTXBUF 时，UCAXTXIFG 置位。

先前的数据发送结束时，只要在 UCAXTXBUF 中的新数据有效发送就会不断进行。在前一个数据发送后如果没有新数据在 UCAXTXBUF 中，发送器返回到空闲态并且波特率发生器被关掉。

(9) UART 波特率产生

USCI 波特率发生器能从非标准源频率中产生一个标准的波特率。通过 UCOS16 位提供两种操作选择

低频波特率生成：

当 UCOS16=0 时低频模式被选中。这种模式允许波特率从低频时钟源中产生（比如 32768HZ 晶振产生 9600 波特率）。通过使用较低的输入频率，能减少模块的能量消耗。

在低频模式中波特率发生器使用一个计数器和一个调节器产生位时钟时序。这种组合支持波特率产生的小数部分。在这种模式下，最大的 USCI 波特率能达到时钟源频率的 1/3。每一位的时序展示在图 3.6.13 中，对接收的每一位，多数表决被采用以决定位值。这些采样过程发生在 $N/2-1/2$, $N/2$, 和 $N/2+1/2$ 个 BRCLK 周期处，N 是每个 BITCLK 周期中 BRCLK 的数目。

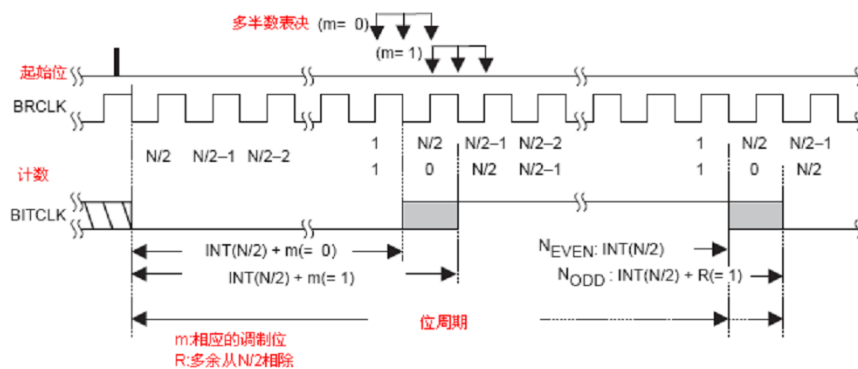


图 3.6.13 UCOS16=0 时 BITCLK 波特率时序

基于 UCBRSX 的调制如表 3.6.6 所示。表中 A1 表示了 $m=1$ 并且相应的 BITCLK 周期比 $m=0$ 时的 BITCLK 大一个 BRCLK 的情况。调制经 8 位循环一次，但是在每一个新的起始位会重启。

表 3.15

BITCLK 调制模式

UCBR _{Sx}	Bit 0 (Start Bit)	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7
0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0
2	0	1	0	0	0	1	0	0
3	0	1	0	1	0	1	0	0
4	0	1	0	1	0	1	0	1
5	0	1	1	1	0	1	0	1
6	0	1	1	1	0	1	1	1
7	0	1	1	1	1	1	1	1

过采样波特率生成:

当 UCOS16 = 1 时过采样模式被选择。这一模式支持以更高的时钟频率取样一个 UART 比特流。这导致多数表决总是相距 1/16 位时钟周期。当 IrDA 编码和解码器使能时，这一模式也非常易于用 3/16 位时间支持 IrDA 脉冲。

这一模式使用一个预定标器和一个调幅器来产生 BITCLK16 时钟信号，这一时钟比 BITCLK 快 16 倍。一个额外的分频器和调幅器从 BITCLK16 阶段产生 BITCLK。这一组合支持 BITCLK16 和 BITCLK 波特率发生的小数分频。在这一模式下，最大的 USCI 波特率位 1/16 UART 时钟源频率 BRCLK。当 UCB_{Rx} 置 0 或 1 时，第一个预定标和调幅器被旁路并且 BRCLK 等于 BITCLK16。BITCLK16 的调制基于表 3.16 给出的设置。表中表示 BITCLK16 的相应周期比 m=0 时长一个 BRCLK 周期。每个新的位定时重新启动调制过程。

表 3.16 BITCLK16 调制模式

UCBR _{Fx}	No. of BITCLK16 Clocks After Last Falling BITCLK Edge															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
00h	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
01h	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
02h	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
03h	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1
04h	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	1
05h	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1
06h	0	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1
07h	0	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1
08h	0	1	1	1	1	0	0	0	0	0	0	0	1	1	1	1
09h	0	1	1	1	1	1	0	0	0	0	0	0	1	1	1	1
0Ah	0	1	1	1	1	1	0	0	0	0	0	1	1	1	1	1
0Bh	0	1	1	1	1	1	1	0	0	0	0	1	1	1	1	1
0Ch	0	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1
0Dh	0	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1
0Eh	0	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
0Fh	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

(10) 设定波特率

对于给定的 BRCLK，波特率有除数因子 N 决定。

$$N = \frac{f_{BRCLK}}{\text{Baud rate}}$$

除数因子 N 通常不是一个整数，因此至少需要一个除法器和一个调制器去调整波特率。如果 N 等于或大于 16，过采样波特率产生模式能通过设置 UCOS16 被选中。

低频波特率模式设置:

在低频模式下，除法器的整数部分由计数器实现:

$$UCBRx = \text{INT}(N)$$

同时小数部分由调制器通过下面的公式实现：

$$UCBRs_x = \text{round} \left((N - \text{INT}(N)) \times 8 \right)$$

对于任何给定的位，通过计算增加或减少 UCBRSX 设定可能会降低最大位误差。为了决定是否是这种情况，在每一次的 UCBRSX 的设定中必须对误差进行详细的计算。

过采样波特率模式设置：

在过采样模式中计数器被设置为：

$$UCBR_x = \text{INT} \left(\frac{N}{16} \right)$$

同时前级调节器被设置为：

$$UCBRf_x = \text{round} \left(\left(\frac{N}{16} - \text{INT} \left(\frac{N}{16} \right) \right) \times 16 \right)$$

当要求更高的精度时，UBRSX 调节器也可以通过从 0-7 调节实现。为了对任何给定位的在最坏情况下的最大误差进行查明，必须对 USCBSX 的 0-7 位的误差进行详细的计算，而且这 0-7 位的随着 UCBRFx 设定增加或减少。

(11) 发送位时序

每个字符的时序是单个位时序的总和。使用波特率发生器的调节特性可以减少累计的位误差。单个位误差可以通过以下的步骤被计算出来。

低频波特率模式设置：

在低频模式中，计算 Tbit, TX[I] 位的长度是基于 UCBRX 和 UCBRSX 的设定：

$$T_{\text{bit}, \text{TX}[i]} = \frac{1}{f_{\text{BRCLK}}} (UCBR_x + m_{\text{UCBRS}_x}[i])$$

这里： $m_{\text{UCBRS}_x}[i]$ = 调制位 i 来自表 3.15 中

过采样波特率模式设置：

在过采样波特率模式中计算位 I Tbit 的长度，TX[I] 的设定基于波特率发生器 UCBRX，UCBRFX 和 UCBRSX：

$$T_{\text{bit}, \text{TX}[i]} = \frac{1}{f_{\text{BRCLK}}} \left((16 + m_{\text{UCBRS}_x}[i]) \times UCBR_x + \sum_{j=0}^{15} m_{\text{UCBRF}_x}[j] \right)$$

这里

$$\sum_{j=0}^{15} m_{\text{UCBRF}_x}[j]$$

总和来自相应的表 3.16 中的内容

$m_{\text{UCBRS}_x}[i]$ 调制位 i 来自表 3.15 中

结束位 Tbit, TX[i] 需要的时间相当于所有的以前的和当前位的时间：

$$t_{\text{bit}, \text{TX}[i]} = \sum_{j=0}^i T_{\text{bit}, \text{TX}[j]}$$

为了计算位误差，时间和理想的位时间 Tbit, 比较，TX[i]：

$$t_{\text{bit}, \text{ideal}, \text{TX}[i]} = \frac{1}{\text{Baud rate}} (i + 1)$$

对于一个理想位的时间 (1/波特率) 在标准化误差中的结果为：

$$\text{Error}_{\text{TX}}[i] = (t_{\text{bit,TX}}[i] - t_{\text{bit,ideal,TX}}[i]) \times \text{Baudrate} \times 100\%$$

(12) 接收位时序

接收时序误差包括两个误差源。第一个是位到位的时序误差，和发送位时序误差相似。第二个是介于一个上升沿事件和被 UCSI 模块接受的上升沿。图 3.6.14 所示的是在 UCAXRXD 脚上的数据和内部波特率时钟之间的异步时钟误差。这导致一个另外的异步误差。异步误差 T_{sync} 是介于 -0.5BRCLKS 和 $+0.5\text{BRCLKS}$ 之间。这取决于所选择的波特率发生模式。

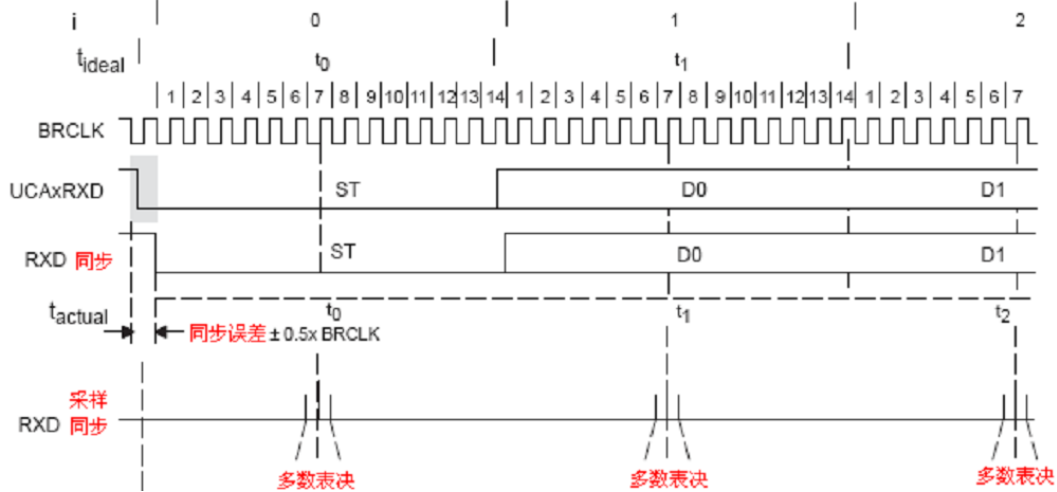


图 3.6.14 接收误差

理想的采样时间 $T_{\text{bit,ideal,RX}}[i]$ 在一位周期之间:

$$t_{\text{bit,ideal,RX}}[i] = \frac{1}{\text{Baud rate}} (i + 0.5)$$

真实采样时间 $T_{\text{bit,rx}}[i]$ 等于先前所有的位的总和，通过下面所列的公式的发送时序部分可以看出，再加上当前位 i 的 $1/2$ 个 BITCLK 以及异步误差 T_{sync} 。对于低频波特率模式结果在以下的 $T_{\text{bit,RX}}[i]$ 中。

$$t_{\text{bit,RX}}[i] = t_{\text{SYNC}} + \sum_{j=0}^{i-1} T_{\text{bit,RX}}[j] + \frac{1}{f_{\text{BRCLK}}} \left(\text{INT} \left(\frac{1}{2} \text{UCBRx} \right) + m_{\text{UCBRSx}}[i] \right)$$

这里:

$$T_{\text{bit,RX}}[i] = \frac{1}{f_{\text{BRCLK}}} (\text{UCBRx} + m_{\text{UCBRSx}}[i])$$

$m_{\text{UCBRSx}}[i]$ 调制的 i 位来自表 3.15 中。

对于过采样波特率模式 $T_{\text{bit,rx}}[i]$ 的 i 位采样时间由以下公式计算:

$$t_{\text{bit,RX}}[i] = t_{\text{SYNC}} + \sum_{j=0}^{i-1} T_{\text{bit,RX}}[j] + \frac{1}{f_{\text{BRCLK}}} \left((8 + m_{\text{UCBRSx}}[i]) \times \text{UCBRx} + \sum_{j=0}^{7+m_{\text{UCBRSx}}[i]} m_{\text{UCBRFx}}[j] \right)$$

这里

$$T_{\text{bit,RX}}[i] = \frac{1}{f_{\text{BRCLK}}} \left((16 + m_{\text{UCBRSx}}[i]) \times \text{UCBRx} + \sum_{j=0}^{15} m_{\text{UCBRFx}}[j] \right)$$

$\sum_{i=0}^{7+m_{UCBRs_x[i]}} m_{UCBRFx[i]}$: 来自 0-7 列的总和+来自表 3.16 中相应的列

$m_{UCBRs_x[i]}$ 来自表 3.15 中的调制

以下公式的结果是标准化的误差中的一个理想位时序（1/波特率）。

$$\text{Error}_{RX}[i] = (t_{\text{bit},RX}[i] - t_{\text{bit,ideal},RX}[i]) \times \text{Baudrate} \times 100\%$$

(13) 典型波特率和错误

对于使用 32768 的晶振源 ACLK 和典型的 SMCLK 频率，通过 UCBRX, UCBSRX 和 UCBRFx 设置的标准的波特率数据被列于表 3.17 和 3.18 中。

表 3.17 常用波特率，设置，误差，UCOS16=0

BRCLK Frequency [Hz]	Baud Rate [Baud]	UCBRx	UCBSRx	UCBRFx	Maximum TX Error [%]		Maximum RX Error [%]	
32,768	1200	27	2	0	-2.8	1.4	-5.9	2.0
32,768	2400	13	6	0	-4.8	6.0	-9.7	8.3
32,768	4800	6	7	0	-12.1	5.7	-13.4	19.0
32,768	9600	3	3	0	-21.1	15.2	-44.3	21.3
1,048,576	9600	109	2	0	-0.2	0.7	-1.0	0.8
1,048,576	19200	54	5	0	-1.1	1.0	-1.5	2.5
1,048,576	38400	27	2	0	-2.8	1.4	-5.9	2.0
1,048,576	56000	18	6	0	-3.9	1.1	-4.6	5.7
1,048,576	115200	9	1	0	-1.1	10.7	-11.5	11.3
1,048,576	128000	8	1	0	-8.9	7.5	-13.8	14.8
1,048,576	256000	4	1	0	-2.3	25.4	-13.4	38.8
1,000,000	9600	104	1	0	-0.5	0.6	-0.9	1.2
1,000,000	19200	52	0	0	-1.8	0	-2.6	0.9
1,000,000	38400	26	0	0	-1.8	0	-3.6	1.8
1,000,000	56000	17	7	0	-4.8	0.8	-8.0	3.2
1,000,000	115200	8	6	0	-7.8	6.4	-9.7	16.1
1,000,000	128000	7	7	0	-10.4	6.4	-18.0	11.6
1,000,000	256000	3	7	0	-29.6	0	-43.6	5.2
4,000,000	9600	416	6	0	-0.2	0.2	-0.2	0.4
4,000,000	19200	208	3	0	-0.2	0.5	-0.3	0.8
4,000,000	38400	104	1	0	-0.5	0.6	-0.9	1.2
4,000,000	56000	71	4	0	-0.6	1.0	-1.7	1.3
4,000,000	115200	34	6	0	-2.1	0.6	-2.5	3.1
4,000,000	128000	31	2	0	-0.8	1.6	-3.6	2.0
4,000,000	256000	15	5	0	-4.0	3.2	-8.4	5.2
8,000,000	9600	833	2	0	-0.1	0	-0.2	0.1
8,000,000	19200	416	6	0	-0.2	0.2	-0.2	0.4
8,000,000	38400	208	3	0	-0.2	0.5	-0.3	0.8
8,000,000	56000	142	7	0	-0.6	0.1	-0.7	0.8
8,000,000	115200	69	4	0	-0.6	0.8	-1.8	1.1
8,000,000	128000	62	4	0	-0.8	0	-1.2	1.2
8,000,000	256000	31	2	0	-0.8	1.6	-3.6	2.0
12,000,000	9600	1250	0	0	0	0	-0.05	0.05
12,000,000	19200	625	0	0	0	0	-0.2	0
12,000,000	38400	312	4	0	-0.2	0	-0.2	0.2
12,000,000	56000	214	2	0	-0.3	0.2	-0.4	0.5

BRCLK Frequency [Hz]	Baud Rate [Baud]	UCBRx	UCBRSx	UCBRFx	Maximum TX Error [%]		Maximum RX Error [%]	
12,000,000	115200	104	1	0	-0.5	0.6	-0.9	1.2
12,000,000	128000	93	6	0	-0.8	0	-1.5	0.4
12,000,000	256000	46	7	0	-1.9	0	-2.0	2.0
16,000,000	9600	1666	6	0	-0.05	0.05	-0.05	0.1
16,000,000	19200	833	2	0	-0.1	0.05	-0.2	0.1
16,000,000	38400	416	6	0	-0.2	0.2	-0.2	0.4
16,000,000	56000	285	6	0	-0.3	0.1	-0.5	0.2
16,000,000	115200	138	7	0	-0.7	0	-0.8	0.6
16,000,000	128000	125	0	0	0	0	-0.8	0
16,000,000	256000	62	4	0	-0.8	0	-1.2	1.2

接收误差是时间累计起来的与在每位中间的理想扫描时间相对。最糟糕的误差由给定的带有校验和一个包括异步误差的停止位的接收产生。

发送误差是时间累计起来的与位周期的理想时间相对。最糟糕的误差由给定的带校验和停止位的传输产生。

表 3.18 常用波特率，设置，误差，UCOS16=1

BRCLK Frequency [Hz]	Baud Rate [Baud]	UCBRx	UCBRSx	UCBRFx	Maximum TX Error [%]		Maximum RX Error [%]	
1,048,576	9600	6	0	13	-2.3	0	-2.2	0.8
1,048,576	19200	3	1	6	-4.6	3.2	-5.0	4.7
1,000,000	9600	6	0	8	-1.8	0	-2.2	0.4
1,000,000	19200	3	0	4	-1.8	0	-2.6	0.9
1,000,000	57600	1	7	0	-34.4	0	-33.4	0
4,000,000	9600	26	0	1	0	0.9	0	1.1
4,000,000	19200	13	0	0	-1.8	0	-1.9	0.2
4,000,000	38400	6	0	8	-1.8	0	-2.2	0.4
4,000,000	57600	4	5	3	-3.5	3.2	-1.8	6.4
4,000,000	115200	2	3	2	-2.1	4.8	-2.5	7.3
4,000,000	230400	1	7	0	-34.4	0	-33.4	0
8,000,000	9600	52	0	1	-0.4	0	-0.4	0.1
8,000,000	19200	26	0	1	0	0.9	0	1.1
8,000,000	38400	13	0	0	-1.8	0	-1.9	0.2
8,000,000	57600	8	0	11	0	0.88	0	1.6
8,000,000	115200	4	5	3	-3.5	3.2	-1.8	6.4
8,000,000	230400	2	3	2	-2.1	4.8	-2.5	7.3
8,000,000	460800	1	7	0	-34.4	0	-33.4	0
12,000,000	9600	78	0	2	0	0	-0.05	0.05
12,000,000	19200	39	0	1	0	0	0	0.2
12,000,000	38400	19	0	8	-1.8	0	-1.8	0.1
12,000,000	57600	13	0	0	-1.8	0	-1.9	0.2
12,000,000	115200	6	0	8	-1.8	0	-2.2	0.4
12,000,000	230400	3	0	4	-1.8	0	-2.6	0.9
16,000,000	9600	104	0	3	0	0.2	0	0.3
16,000,000	19200	52	0	1	-0.4	0	-0.4	0.1
16,000,000	38400	26	0	1	0	0.9	0	1.1
16,000,000	57600	17	0	6	0	0.9	-0.1	1.0
16,000,000	115200	8	0	11	0	0.9	0	1.6
16,000,000	230400	4	5	3	-3.5	3.2	-1.8	6.4
16,000,000	460800	2	3	2	-2.1	4.8	-2.5	7.3

(14) 在低功耗模式下使用 USCI 模块的 UART 模式

低功耗模式下，USCI 模块支持自动激活 SMCLK 时钟。当 SMCLK 作为 USCI 模块的时钟源，因为设备处于低功耗模式 SMCLK 未激活，如果必要 USCI 模块将自动激活，不管时钟源的控制位如何。时钟将保持活动状态直到 USCI 模块返回空闲状态。在 USCI 返回空闲状态后，时钟源的控制将受制于其控制位。自动激活模式不适用于 ACLK。

当 USCI 模块激活一个不活动的时钟源，整个设备以及使用此时钟源的外围配置的时钟

将激活。例如，在 USCI 模块强制激活 SMCLK 时使用 SMCLK 的定时器将计数。

(15) USCI 中断

USCI 有一个发送中断向量和接收中断向量。

USCI 发送中断操作:

UCAXTXIFG 中断标志被发送器置位表示 UCAXTXBUF 准备接收另一个字符。如果 UCAXTXIE 和 GIE 也置位将产生一个中断请求。如果字符被写到 UCAXTXBUF 中 UCAXTXIFG 将自动复位。

USCI 接收中断操作:

一个字符被接受并装载到 UCAXRXBUF 中去时 UCAXRXIFG 中断标志置位一次。如果 UCAXRXIE 和 GIE 置位将产生一个中断请求。复位 PUC 信号或者 UCSWRST=1 时 UCAXRXIFG 和 UCAXRXIE 将被系统复位。当 UCAXRXBUF 被读时 UCAXRXIFG 将自动复位。

其他的中断控制特性包括:

- ◆在 UCAXRXEIE=0 时错误字符不会置位 UCAXRXIFG。
- ◆在 UCDORM=1 时，在多处理机模式下非地址字符不会置位 UCAXRXIFG。
- ◆当 UCBRKIE=1 时一个中断条件将置位 UCBRK 位和 UCAXRXIFG 标志。

USCI 中断用法:

USCI_AX 和 USCI_BX 使用同一个中断向量。接收中断标志 UCAxRXIFG 和 UCBxRXIFG 和同一个中断向量连接，发送中断标志 UCAxTXIFG 和 UCBxTXIFG 分享另一个中断向量。

2. USCI 模块串行异步通信例程

以 MSP430G2553 的 USCI 模块串行异步通信操作为例，介绍串口寄存器配置及收发程序处理过程。

(1) 初始化串口

```
void UART_init(void)
```

```
{  
    UCAOCTL1 |=UCSWRST;  
    UCAOCTL1 |= UCSSEL_2;           //SMCLK  
    UCAOBR0 = 0x68;                 //32.768/9600=  
    UCAOBR1 = 0x00;                 // 1000kHz/9600 = 104.166 =0X68 波特率9600  
    UCAOMCTL = UCBRS_2;            // Modulation UCBRSx = 1  
    UCAOCTL0 &=~UCPEN;  
    UCAOCTL0 &=~UCSPB;  
    UCAOCTL0 &=~UC7BIT;  
    UCAOCTL1 &=~UCSWRST;  
    P1SEL |=BIT1+BIT2;  
    P1SEL2 |=BIT1+BIT2;  
    //P1DIR |=BIT2;第二功能无需配置? ?  
    IE2 |= UCA0RXIE+UCA0TXIE;  
    unsigned int j;
```

```

    for(j=0;j<2000;j++);
}

```

(2) 串口发送函数

```

/*****
    宏定义
*****/
unsigned char RX_BUFF[RXBUF_SIZE]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}; //接收缓冲区
unsigned int UART_InLen=16; //输入数据长度
unsigned int RX_IndexR = 0; //接收FIFO的读指针
unsigned int RX_IndexW = 0; //接收FIFO的写指针
unsigned char TX_BUFF[TXBUF_SIZE]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}; //发送缓冲区
unsigned int UART_OutLen =16; //发送FIFO内待发送的字节数
unsigned int TX_IndexR = 0; //发送FIFO的读指针
unsigned int TX_IndexW = 0; //发送FIFO的写指针
uint8 ch=0;
/*****
    串口发送一字节函数，查询方式
*****/
void UART_sent(uint8 Chr)
{
    IFG2&=~UCA0TXIFG;
    UCA0TXBUF=Chr;
    while ((IFG2&UCA0TXIFG)==0); // USCI_A0 TX buffer ready?
}
/*****
    串口发送一帧数据函数，中断方式
*****/
char UART0_PutFrame(unsigned char *Ptr,unsigned int Lenth)
{
    int i;
    if(IE2&UCA0TXIE)
    {
        return(0);
    }
    if(Lenth>TXBUF_SIZE)
    {
        return(0);
    }
}

```

```

    }
    for(i=0;i<Lenth;i++)
    {
        delay_us(100);
        TX_BUFF[i]=Ptr[i];
    }
    TX_IndexR=0;
    UART_OutLen=Lenth;
    IFG2|=UCA0TXIFG;
    IE2|=UCA0TXIE;
    return(1);
}

/*****
* 名 称: USCIxTX_ISR()
* 功 能: 串口发送中断, 每发完1字节会发生一次中断
*****/
#pragma vector=USCIAB0TX_VECTOR
__interrupt void USCI0TX_ISR (void)
{
    if (UART_OutLen > 0)
    {
        UART_OutLen--;
        UCA0TXBUF = TX_BUFF[TX_IndexR];
        while (!(IFG2&UCA0TXIFG)); //查看上一字节是否发送完毕
        if (++TX_IndexR >= TXBUF_SIZE)
        {
            TX_IndexR = 0;
        }
    }
    else IE2 &=~UCA0TXIE;
}

```

(3) 串口接收函数

```

/*****
串口接收一字节函数, 查询方式, 此处虽定义但并未使用, 不保证其正确性
*****/
uint8 UART_receive(void)
{
    while ((IFG2&UCA0RXIFG)==0); // 收到一字节?
}

```



```

    IFG2&=~UCAORXIFG;
    return(UCAORXBUF);
}
#pragma vector=USCIABORX_VECTOR
__interrupt void USCIORX_ISR(void)
{
    UART_InLen++;
    RX_BUFF[RX_IndexW] = UCAORXBUF;          //保存在缓冲区中
    if (++RX_IndexW >= RXBUF_SIZE)
    {
        RX_IndexW = 0;
    }
}

```

3.6.5 USART 模块的初始化和收发操作步骤流程

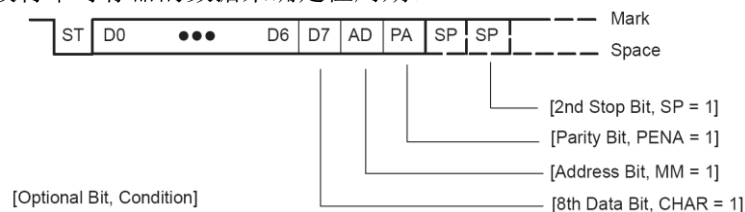
1. USART 模块的串行异步通信操作

MSP430 单片机串行异步通信模式通过两个引脚，即接收引脚 URXD 和发送引脚 UTXD 与外界相连。异步通信的特点如下：

- ◆ 异步模式，包括线路空闲/地址位通信协议。
- ◆ 两个独立移位寄存器：输入移位寄存器和输出移位寄存器。
- ◆ 传输 7 位或 8 位数据，可采用奇校验或偶校验或者无校验。
- ◆ 从最低位开始的数据发送和接收。
- ◆ 可编程实现分频因子为整数或小数的波特率。
- ◆ 独立的发送和接收中断。
- ◆ 通过有效的起始位检测将 MSP430 从低功耗唤醒。
- ◆ 状态标志检测错误或者地址位。

(1) 异步通信字符格式

异步通信字符格式由 4 部分组成：起始位、数据位、奇偶校验位和停止位，如图 3.6.15 所示。其中用户可以通过软件设置数据位、停止位的位数，还可以设置奇偶位的有无。通过选择时钟源的波特率寄存器的数据来确定位周期。



3.6.15 异步通信字符格式

接收操作以收到有效起始位开始。起始位由检测 URXD 端口的下降沿开始，然后以 3 次采样多数表决的方法取值。如果 3 次采样至少两次是 0 才表明是下降沿，然后开始接收初始化操作，这一过程实现错误起始位的拒收和帧中各数据的中心定位功能。

MSP430 单片机可以处于低功耗模式，通过上述过程识别正确起始位后，MSP430 单片机

可以被唤醒，然后按照通用串口接口可以控制寄存器中设定的数据格式，开始接收数据，直到本帧采集完毕。

异步模式下，传送数据是以字符为单位传送的。因为每个字符在起始位处被唤醒，所以传输时多个字符可以一个接一个地连续传送，也可以断续发送和接收。

(2) 异步多机通信模式

在异步模式下，USART 支持两种多机通信模式，即线路空闲和地址位多机模式。在信息以一个多帧数据块，从一个指定地源传送到一个或者多个目的位置。在同一个串行链路上，多个处理器之间可以用这些格式来交换信息，实现了在多处理器通信系统间的有效数据传输。它们也用于使系统的激活状态压缩最低，以节省电流消耗或处理器所用资源。控制寄存器的 MM 位用来确定这两种模式。这两种模式采用唤醒发送、地址特性和激活等功能。

线路空闲多机模式：

在这种模式下，数据块被空闲时间分割。在字符的第一个停止位之后，收到 10 个以上的 1，则表示检测到接收线路空闲，如图 3.6.16 所示。

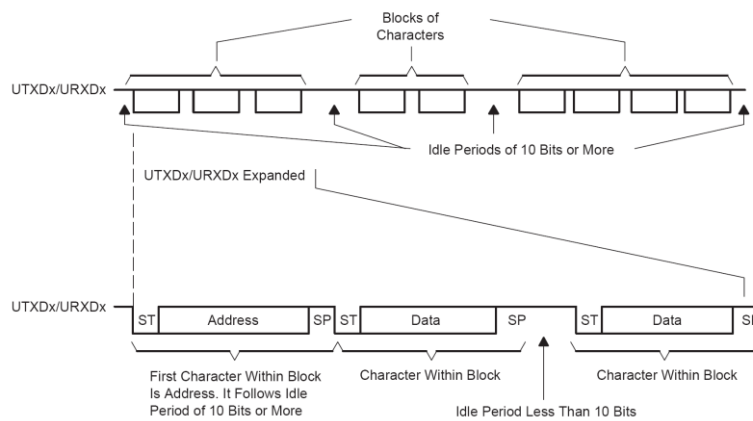


图 3.6.16 线路空闲多机模式

如果采用两位停止位，第二个停止位被认为空闲周期的第一个标志。空闲周期的第一个字符是地址字符。RXWAKE 位于可用于地址字符的标志。当接收到的字符是地址字符时，RXWAKE 被置为，并送入接收缓存。

用发送空闲帧来识别地址字符的步骤如下：

- TXWAKE=1，将任意数据写入 UTXBUF (UTXIFG=1)。当发送移位寄存器为空时，UTXBUF 的内容将被送入发送移位寄存器，同时 TXWAKE 的值移入 WUF。

- 如果此时 WUT=1，则将发送的起始位、数据位及校检位等被抑止，发送一个正好 11 位的空闲周期。

- 在地址字符识别空闲周期后移出串口的下一个数据是 TXWAKE 置位后写入 UTXBUF 中的第二个字符。当地址识别被发送后，写入 UTXBUF 中的第一个字符被抑制，并在以后被忽略。这时需随便往 UTXBUF 中写入一个字符，以便能将 TXWAKE 的值移入 WUF 中。

地址位多机模式：

地址位多机模式的格式如图 3.6.17 所示。在这种模式下，字符包含一个附加的位作为地址标志。数据块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。当接收字符是地址时，RXWAKE 置位，并且将接收的字符送入接收缓存 URXBUF。

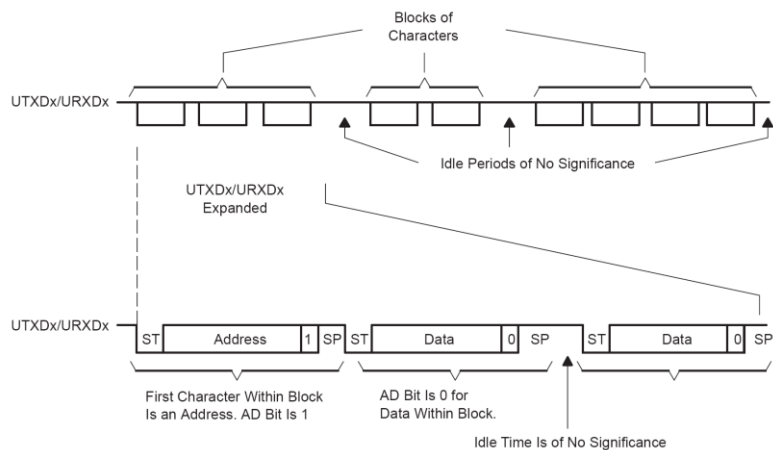


图 3.6.17 地址位多机模式

在 USART 的 URXWIE=1 时，数据字符在通常方式的接收器内拼装成字节，但他们不会被送入接收缓存，也不产生中断。只有当接收到一个地址位为 1 的字符时，接收器才被激活，接收到的字符被送往 URXBUF，同时 URXIFG 被置位。如果有错误，则相应的错误标志被设置。应用软件在判断后作出相应的处理。在地址位多机模式下，通过写 TXWAKE 位控制字符的地址位。每当字符由 UTXBUF 传送到发送器时，TXWAKE 位装入字符的地址位，再由 USART 将 TXWAKE 位清除。

串行操作自动错误检测：

USART 模块接收字符时，能够自动进行校验错误、帧错误、溢出错误和打断状态检测。各种检测错误的含义标志如下：

- ◆FE：标志帧错误：当一个接受字符的停止位为 0 并被装入接收缓存，则认为接收到一个错误帧。
- ◆PE：奇偶校验错误：当接收的 1 的个数与它的效验位不相符。
- ◆OE：溢出错误标志：当然一个字节写入 UxRXBUF 时，前一个字符还没有被读出，则溢出。
- ◆BRK：打断检测标志：当发生一次打断同时 UxRXEIE 置位，该位为 1。

当上述任何一种错误出现，位 RXERR 置位，表明有一个或者多个出错标志（FE、PE、OE 和 BRK 等）被置位。

如果 URXEIE=0 并且有错误被检测到，那么接收缓存不接收任何数据。如果 URXEIE=1，接收缓存接收字符，相应的错误检测标志置位，直到用户软件复位或者接收缓存内容被读出，才能复位。

波特率的产生：

在异步通信中，波特率是很重要的指标，表示为每秒钟传送二进制数码的位数。波特率反映了异步串行通信的速度。波特率发生器产生同步信号表明各位的位置。波特率部分由时钟输入选择和分频、波特率发生器、调整器和波特率寄存器组成。串行通信时，数据接收和发送的速率就由这些构件控制。图 3.6.18 为其较为详细的结构。

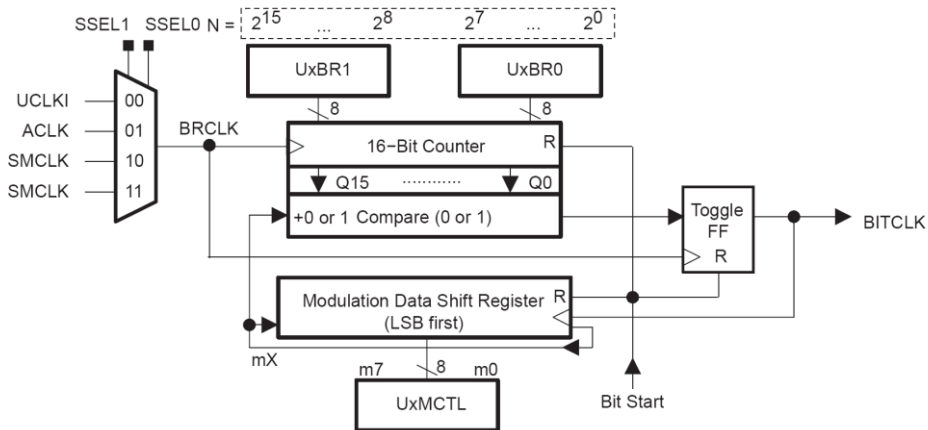


图 3.6.18 波特率生成器结构

整个模块的时钟来自内部 3 个时钟或外部输入时钟，由 SSEL1 和 SSEL0 选择，以决定最终进入模块的时钟信号 BRCLK 的周期。时钟信号 BRCLK 送入一个 16 位的分频器，通过一系列的硬件控制，最终输出移位寄存器使用的移位时钟 BITCLK 信号。这个信号（BITCLK）的产生，由图 3.6.18 的下半部分可以看出，是分频器在起作用。当计数器减到 0 时，输出触发器翻转，送给 BITCLK 信号周期的一半就是定时器（分频计数器）的定时时间。

下面介绍波特率的设置与计算。采集每位数据的时候，在每位数据的中间都要进行 3 次采样，以多数表决的原则进行数据标定与移位接收操纵，如此依次采集。由此看出，分频因子要么很大，要么是整数，否则，由于采集点的积累偏移，会导致每帧后面的几位数据采样点不在其数据的有效范围内。

MSP430 单片机的波特率发生器使用一个分频计数器和一个调整器，能够用低时钟频率实现高速通信，从而在系统低功耗的情况下实现高性能的串行通信。使用分频因子加调整的方法可以实现每一帧的各位有不同的分频因子，从而保证每个数据帧的 3 个采样状态都处于有效的范围内。

分频因子 N 由分频计数器的时钟（BRCLK）的频率和所需的波特率来决定：

$$N = \text{BRCLK} / \text{波特率}$$

如果使用常用的波特率与常用晶体产生的 BRCLK，则一般得不到整数的 N，还有小数部分。分频计数器实现分频因子的整数部分，调整器使得小数部分尽可能准确。分频因子定义如下：

N 为目标分频因子；UBR 为 UxBR0 中的 16 位数据值；MX 为调整寄存器（UXMCTL）中的各数据位。

波特率可由下式计算：

$$\text{波特率} = \text{BRCLK} / N = \text{BRCLK} / (\text{UBR} + (\text{M7} + \text{M6} + \dots + \text{M0}) / 8)$$

例如：BRCLK=32.768KHz，要产生 BITCLK=2400Hz，分频器的分频系数为 32768 / 2400 = 13.65。所以设置分频器的计数值为 13。接下来用调整寄存器的值来设置小数部分的 0.65。调整器是一个 8 位寄存器，其中每一位分别对应 8 次分批情况，如果对应位 0，则分频器按照设定的分频系数分频计数；如果对应位为 1，则分频器按照设定的分频系数加 1 分频计数。按照这个原则 0.65*8=5，也就是说，8 次分频计数过程中应有 5 次加 1 计数，3 次不加 1 计数。调整寄存器的数据由 5 个 1 和 3 个 0 组成。调整器的数据每 8 次周而复始循环使用，最

低位最先调整，比如设置调整寄存器的值为 6BH (01101011)，当然也可以设置其他值，必须有 5 个 1，而且 5 个 1 要相对分散。即分频器按顺序 13、14、14、13、14、13、14、14 来分频。在 8 位调整器调整位都使用后，再重复这一顺序。

常用波特率设置表：

表 3.19 常用波特率设置

Baud Rate	Divide by		A: BRCLK = 32,768 Hz						B: BRCLK = 1,048,576 Hz				
	A:	B:	UxBR1	UxBR0	UxMCTL	Max. TX Error %	Max. RX Error %	Synchr. RX Error %	UxBR1	UxBR0	UxMCTL	Max. TX Error %	Max. RX Error %
1200	27.31	873.81	0	1B	03	-4/3	-4/3	±2	03	69	FF	0/0.3	±2
2400	13.65	436.91	0	0D	6B	-6/3	-6/3	±4	01	B4	FF	0/0.3	±2
4800	6.83	218.45	0	06	6F	-9/11	-9/11	±7	0	DA	55	0/0.4	±2
9600	3.41	109.23	0	03	4A	-21/12	-21/12	±15	0	6D	03	-0.4/1	±2
19,200		54.61							0	36	6B	-0.2/2	±2
38,400		27.31							0	1B	03	-4/3	±2
76,800		13.65							0	0D	6B	-6/3	±4
115,200		9.1							0	09	08	-5/7	±7

2. USART 模块异步串行通信操作例程

(1) 初始化串口

初始化串口 0，允许接收和发送，允许接收中断，禁止发送中断，8bit 字符发送时钟 ACLK = 32.768KHz，波特率 4800

注意：对串口寄存器操作的时候务必保证 SWRST=1，设置完成后 SWRST=0。

// 串口初始化函数

```
void InitUSART(void)
{
    UOCTL   |= CHAR;           // 8bit 字符
    UOTCTL   |= SSEL0;        // ACLK
    UOBR1    = 0x00           // 4800 波特率
    UOBRO    = 0x06
    UOMCTL   = 0x6f
    UOCTL    &= ~SWRST;      // 设置完成
    ME1      |= UTXE0 + URXE0; // 接收发送允许
    IE1      |= URXIE0;       // 接收发送中断
    P3SEL    |= (0x80 + 0x40); // 引脚切换到特殊功能上
    _EINT(); // 不要忘了开中断
}
```

(2) 串口发送函数

采用软件查询式发送，将 1 个字节写入发送寄存器，然后等待发送完成的标志。本方法适合波特率较高的场合（大于 4800）

// 所涉及的全局变量

```
unsigned char TBuff[8]; // 发送缓冲区
unsigned char RBuff[8]; // 接收缓冲区
```

```

unsigned char Flag_Receive = 0;    // 标志: 接收到有效数据
// 串口发送函数(不需要开发送中断)发送一个数组(共 8 个字节)
void USART_Send(unsigned char *pData)
{
    unsigned char i;
    for(i=0; i<8; i++)
    {
        TXBUF0 = pData[i];        // 装入发送寄存器
        while ((IFG1 & UTXIFG0) == 0); // 判断:发送是否完成
        IFG1 &= ~(UTXIFG0);
    }
}

```

(3) 串口接收函数

在 RAM 开辟接收缓冲区，等到接收的数据组满足要求或者为一帧数据时才处理。中断中接收。

```

// 所涉及的全局变量
unsigned char TBuff[8];        // 发送缓冲区
unsigned char RBuff[8];        // 接收缓冲区
unsigned char Flag_Receive = 0; // 标志: 接收到有效数据
// 串口接收函数(需要开接受中断)
#pragma vector=UARTORX_VECTOR
__interrupt void USARTO_RXIRQ (void)
{
    unsigned char Temp;
    // 暂存接收数据
    Temp = RXBUF0;
    // 8 字节接收缓冲队列
    RBuff[0] = RBuff[1];
    RBuff[1] = RBuff[2];
    RBuff[2] = RBuff[3];
    RBuff[3] = RBuff[4];
    RBuff[4] = RBuff[5];
    RBuff[5] = RBuff[6];
    RBuff[6] = RBuff[7];
    RBuff[7] = Temp;
    // 判断数据有效(有效则设置标志位, 等待处理)
    if (RBuff[0] == 0xd0){Flag_Receive = 1; return; }else{Flag_Receive = 0;}
}

```

(4) 串口校验 CRC16 函数

常用函数，使用的 CRCKey = 0xA001

输入一个数组和长度，则计算出该部分的 CRC 值。

// CRC-16 循环冗余效验函数

```
unsigned int Caculate_CRC16(unsigned char *DAT, unsigned char Lenth)
{
    // *DAT 指向要计算 CRC 的数组, Lenth 为数据的有效长度
    unsigned int CRC = 0xffff;    // CRC 的初始值为 FFFF
    unsigned char i;
    unsigned char j;
    for(i=0; i<Lenth; i++)
    {
        CRC = CRC ^ DAT[i];        // 和当前字节异或一次
        for(j=0; j<8; j++)        // 循环 8 次
        {
            if(CRC & 0x01)        // 判断最低位, 如果为 1
            {
                CRC = CRC >> 1;    // 右移一位
                CRC = CRC ^ 0xA001; // 和多相式异或
            }
            else                    // 判断最低位, 如果为 0
            {
                CRC = CRC >> 1;    // 右移一位
            }
        }
    }
    return(CRC);
}
```

第七节 比较器 A+

比较器 A+是一个模拟电压比较器。本节将讲述 MSP430G2553 单片机中比较器 A+的操作。

3.7.1 比较器 A+的介绍

比较器 A+模块支持精度斜率的模数转换、电源电压的管理以及对外部模拟信号的监控。

Comparator_A+所具有的特征：

- (1) 同相端和方向端输入多路器
- (2) 可软件选择的比较器输出 RC 滤波器
- (3) 输出可作为 Timer_A 的捕获输入
- (4) 端口输入缓存的软件控制
- (5) 中断的能力
- (6) 可供选择的参考电压产生器
- (7) 比较器和参考电压发生器可掉电。
- (8) 输入多路器

比较器 A+结构框图如图 3.7.1 所示：

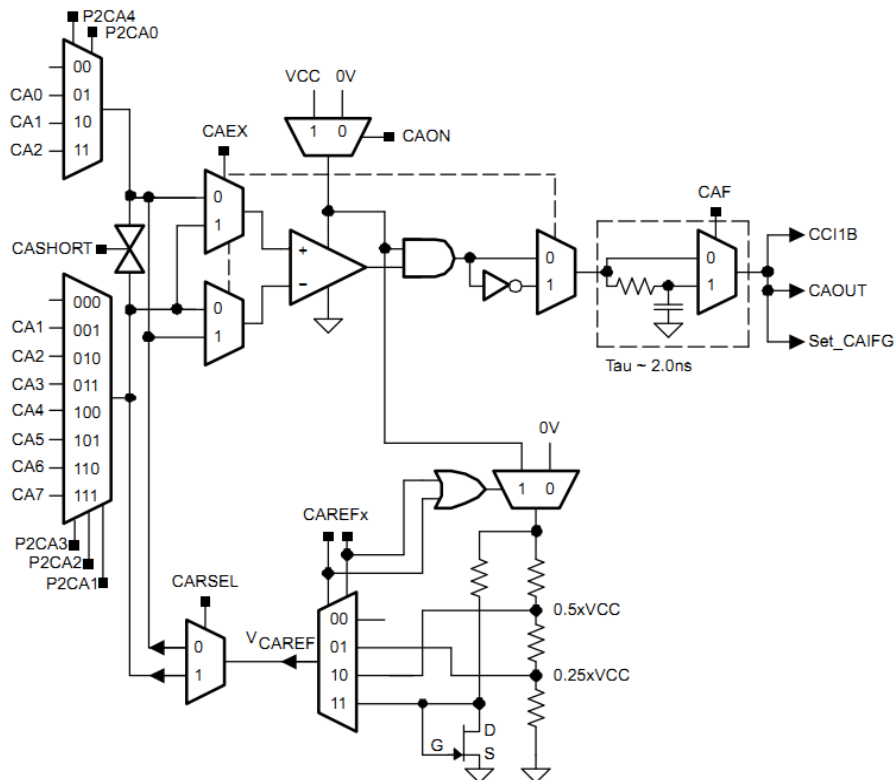


图 3.7.1 比较器 A+功能框图

3.7.2 比较器 A+的操作

比较器 A+模块是用户通过软件配置的。下面的篇幅中将着重论述比较器 A+的配置和操作：

1. 比较器

比较器的作用是比较其同相输入端和反相输入端的模拟电压。若同相输入端的电压高于反相输入端的电压，比较器输出 CAOUT 为高电平，否则输出 CAOUT 为低电平。比较器可以使用 CAON 位进行开启或关闭。在不使用比较器的情况下应该将其关闭以减少电流的消耗。当比较器被关闭后，CAOUT 总是呈现低电平状态。

2. 输入模拟开关

用户可以通过对 P2CAx 位的设置控制模拟输入开关对比较器两输入端与相关端口引脚的连接。比较器的两个输入端可以单独控制。P2CAx 位允许：

- (1) 对输入到比较器同相端和反相端的外部信号的应用。
- (2) 内部参考电压到相关输出端口引脚的路由。

比较器内部的输入开关被构造为 T 型开关，这样利于抑制信号通路上的失真。

注意：比较器的输入连接

比较器处于开启状态时，输入端应该连接到信号、电源或地，否则，悬空的电平将产生无法预料的中断并且电流的消耗也会增加。

CAEX 位用于控制输入多路器，而多路器正是用于交换输入到比较器同相输入端和反相输入端的信号。此外，当比较器的输入端信号被交换时，比较器的输出信号也将反相。这使得用户可以确定比较器的输入失调电压的存在或者对其进行补偿。

(3) 输入短路开关

通过 CASHORT 位的配置可以使比较器的输入端短路。这可以用于搭建一个简单的比较器的采样和保持电路，如图 3.7.2 所示：

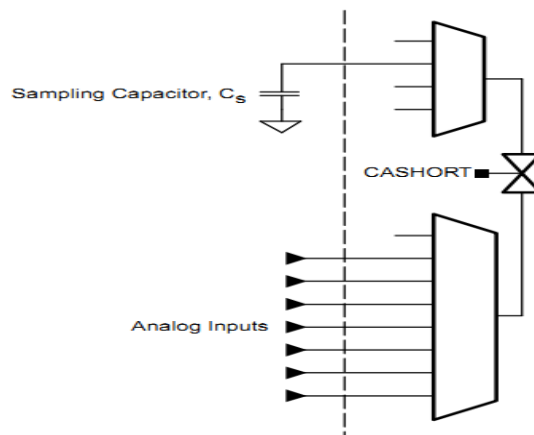


图 3.7.2 比较器 A+ 的采样和保持

采样时间与采样电容 C_S 的电容值、与短路开关串联的输入开关电阻 R_I 的阻值以及外部源电阻 R_S 的阻值的大小成正比。总内部电阻 R_I 大小范围是 $2 \sim 10 \text{ K}\Omega$ 。采样电容 C_S 的电容值应该大于 100 pF ，其充电的时间常数 T_{au} 可以通过下面的等式来计算：

$$T_{au} = (R_I + R_S) \times C_S$$

采样时间应该是 T_{au} 值的 $3 \sim 10$ 倍，这取决于对采样精度的要求。若使用 3 倍于 T_{au} 的采样时间，采样电容充电电压的最大值将达到输入信号电压值的 95%，若使用 5 倍于 T_{au} 的采样时间，采样电容充电电压的最大值将超过输入信号电压值的 99%，若使用 10 倍于 T_{au} 的采样时间，采样电压对 12 位的精度要求来说是足够的。

(4) 输出滤波器

用户可以选择是否需要将比较器的输出通过其内置的滤波器滤波后再使用。当 CAF 控制位被置位，比较器输出使用片上 RC 滤波器进行滤波。

只要比较器的两输入端上有小幅的电压差，输出端就会有振荡现象产生。这种现象产生的原因是源于信号线、电源线和系统的其他部分之间的寄生效应和串扰的影响，如图 3.7.3 所示。比较器输出端的振荡将降低比较器的精度和比较结果的分辨率。选择输出滤波器可以减小由比较器振荡产生的误差。

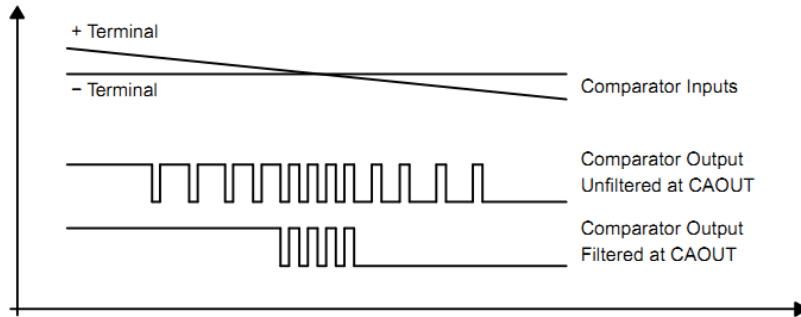


图 3.7.3 比较器输出端的 RC 滤波器响应

(5) 参考电压发生器

参考电压发生器产生连接到比较器任意一个输入端的参考电压 V_{CAREF} 。CAREF_x 位控制电压发生器的输出。CARSEL 位选择连接参考电压 V_{CAREF} 的比较器的输入端。如果外部信号被连接到比较器的输入端，此时应当关断内部参考电压发生器以减少电流的消耗。参考电压发生器可以产生对 V_{CC} 进行分压后的三种电压值的电压或者大小固定约为 0.55V 的晶体管阈值电压。

(6) 比较器 A+，端口禁用寄存器——CAPD

比较器输入和输出的功能是通过复用相关的数字 CMOS 门构成的 I/O 端口引脚实现的。当模拟信号被施加于 CMOS 门上，寄生电流就会从 VCC 流动到 GND。若输入电压接近 CMOS 门的跳变电平时就会产生这种寄生电流。禁用端口引脚的缓冲器可以消除寄生电流的流动因此也可以减少总的电流消耗。

当 CAPD_x 位被置位时，相应引脚的输入和输出缓冲器也将被禁用，如图 3.7.4 所示。当电流消耗比较严重的时候，任何连接模拟信号的端口引脚都应该通过设置相应的 CAPD_x 位使其缓冲器被禁用。

不管与它相关的 CAPD_x 位的状态如何，用 P2CA_x 位选择比较器多路器上输入信号的输入端引脚都将自动禁用那个引脚的输入和输出缓冲器。

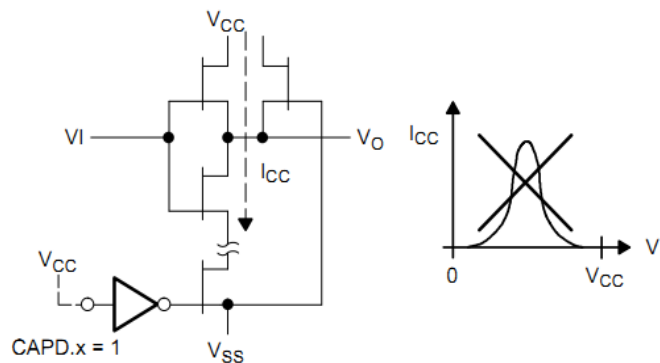


图 3.7.4 CMOS 反相器/缓冲器的传输特性和功耗

(7) 比较器 A+中断

如图 3.7.5 所示，比较器 A+与一个中断标志和一个中断向量有关。中断标志 CAIFG 在比较器输出的上升沿和下降沿被置位，通过 CAIES 位可以选择具体的边沿。如果 CAIE 和 GIE 都被置位，那么 CAIFG 标志就产生一个中断请求。CAIFG 标志在中断请求服务完毕后自动清 0 或者被软件清 0。

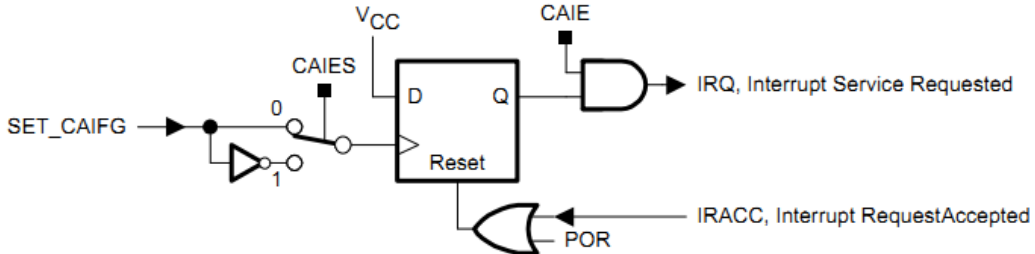


图 3.7.5 比较器 A+中断系统

(8) 比较器 A+用于热敏电阻的温度测量

如图 3.7.6 所示， R_{ref} 为参考电阻， R_{meas} 为热敏电阻，单片机的 I/O 口 Px.x 输出高电平对电容充电直至充满，此时电容两端的电压为 V_{CC} ，高于 $0.25 \times V_{CC}$ ，比较器输出高电平；然后 Px.x 输出低电平，使电容通过电阻 R_{ref} 进行放电。当电容两端电压低于 $0.25 \times V_{CC}$ 时，比较器从高电平跳变为低电平，该下降沿被定时器捕获，得到捕获值 N_{ref} 。同样，单片机的 I/O 口 Px.y 输出高电平对电容充电，直至充满，此时电容两端的电压为 V_{CC} ，高于 $0.25 \times V_{CC}$ ，比较器输出高电平；然后 Px.y 输出低电平，使电容通过电阻 R_{meas} 进行放电。当电容两端电压低于 $0.25 \times V_{CC}$ 时，比较器从高电平跳变为低电平，此下降沿被定时器捕获，得到捕获值 N_{meas} 。

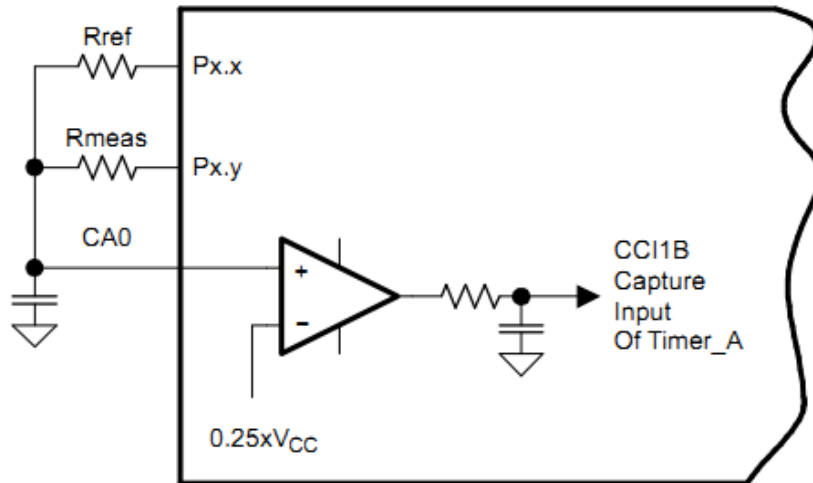


图 3.7.6 温度测量系统

MSP430 单片机上用于计算由 R_{meas} 检测到的温度的资源有：

- 1) 两个对电容充放电的数字 I/O 引脚。
- 2) I/O 输出高电平 (V_{CC}) 对电容进行充电，输出低电平使电容放电。
- 3) 当 I/O 口闲置时，用 CAPDx 位将其切换到高阻抗输入状态。
- 4) 一个 I/O 口的输出通过参考电阻 R_{ref} 对电容进行充放电。

- 5) 另一个 I/O 口的输出通过电阻 R_{meas} 对电容充放电。
- 6) 比较器同相端连接到电容的正极。
- 7) 比较器反相端连接参考电压，例如 $0.25 \times V_{CC}$ 。
- 8) 输出滤波器用来尽可能地减小开关噪声。
- 9) 比较器的输出 CAOUT 连接到 Timer_A 的捕获口 CCI1B，通过电平跳变沿捕获获取电容器的放电时间。

使用这种方法用户至少可以对一个电阻元件的阻值进行测量。其余的电阻元件可以连接到其他未被使用的 I/O 引脚上，当这些 I/O 口闲置时请将其切换到高阻态。热敏电阻的温度测量是基于比例转换的原则。两个电容放电次数的计算如下图所示：

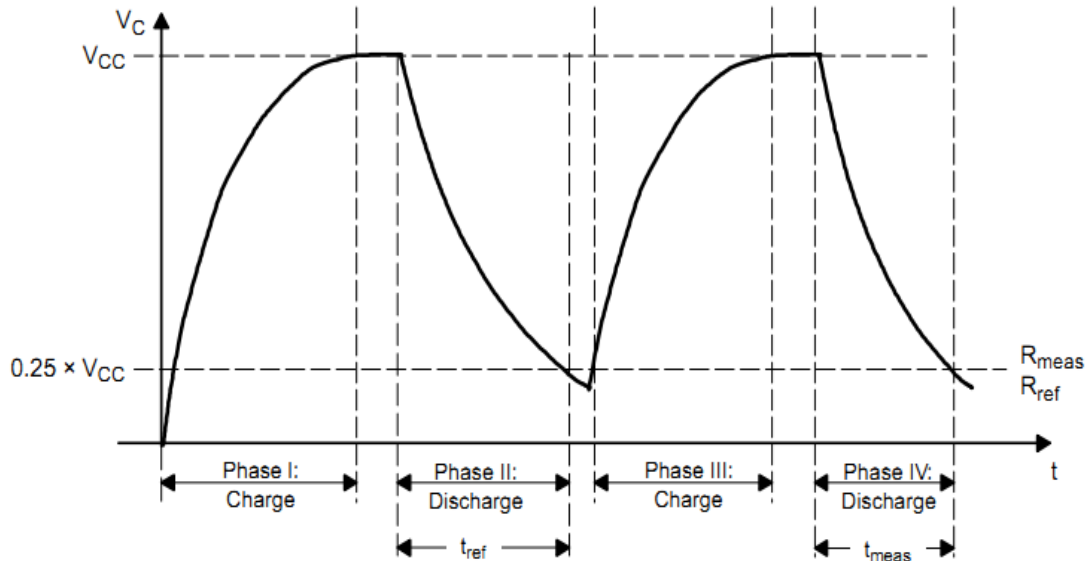


图 3.7.7 温度测量系统的电容充放电波形

在转换期间， V_{CC} 电压和电容值应该保持恒定，但这并不是关键，因为它们都在比例化简中被消去了：

$$\frac{N_{meas}}{N_{ref}} = \frac{-R_{meas} \times C \times \ln \frac{V_{ref}}{V_{CC}}}{-R_{ref} \times C \times \ln \frac{V_{ref}}{V_{CC}}}$$

$$\frac{N_{meas}}{N_{ref}} = \frac{R_{meas}}{R_{ref}}$$

$$R_{meas} = R_{ref} \times \frac{N_{meas}}{N_{ref}}$$

得到当前温度下热敏电阻的阻值 R_{meas} 后，根据热敏电阻厂家提供的温度阻值对照表可即得当前环境下的温度值。

3.7.3 比较器 A+ 寄存器

比较器 A+ 的寄存器如表所示：

表 3.20

比较器 A+的寄存器

寄存器	缩写形式	寄存器类型	地址	初始状态
比较器 A+控制寄存器 1	CACTL1	可读/写	059h	上电复位时被清 0
比较器 A+控制寄存器 2	CACTL2	可读/写	05Ah	上电复位时被清 0
比较器 A+端口禁用	CAPD	可读/写	05Bh	上电复位时被清 0

(1) 比较器 A+控制寄存器 1——CACTL1

	7	6	5	4	3	2	1	0
CAEX		CARSEL	CAREF _x	CAON	CAIES	CAIE	CAIFG	

CAEX: 第 7 位 比较器 A+交换, 该位交换比较器的两个输入端信号和使比较器的输出反相。

CARSEL: 第 6 位 比较器 A+参考电压的选择。该位选择参考电压 V_{CAREF} 连接到比较器的哪个端。

当 CAEX=0:

0 V_{CAREF} 连接到比较器的同相输入端

1 V_{CAREF} 连接到比较器的反相输入端

当 CAEX=1:

0 V_{CAREF} 连接到比较器的反相输入端

1 V_{CAREF} 连接到比较器的同相输入端

CAREF: 第 5-4 位 比较器 A+参考电压。这些位用于选择参考电压 V_{CAREF} 。

00 关闭内部参考电压源, 连接外部参考电压。

01 $0.25 \times V_{CC}$

10 $0.50 \times V_{CC}$

11 选择二极管参考电压

CAON: 第 3 位 比较器 A+开启。该位启动比较器。关闭比较器则其电流消耗停止。参考电路被独立地使能和禁用。

CAIES: 第 2 位 比较器 A+中断边沿选择。

0 上升沿

1 下降沿

CAIE: 第 1 位 比较器中断使能

0 禁用

1 使能

CAIFG: 第 0 位 比较器 A+中断标志

0 无中断挂起

1 中断挂起

(2) 比较器 A+控制寄存器——CACTL2

	7	6	5	4	3	2	1	0
CASHORT		P2CA4	P2CA3	P2CA2	P2CA1	P2CA0	CAF	CAOUT
	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r-(0)

CASHORT: 第 7 位 输入短路。该位将比较器的同相输入端和反相输入端短路。

0 比较器两输入端不短路

1 比较器两输入端短路

P2CA4: 第 6 位 输入选择位。该位和 P2CA0 一起，在 CAEX=0 时，选择同相端输入，在 CAEX=1 时，选择反相端输入。

P2CA3: 第 5-3 位 输入选择位。当 CAEX=0 时，这些位选择反相端输入，当 CAEX=1，P2CA2 这些位选择同相端的输入。

P2CA1:

- 000 无连接
- 001 CA1
- 010 CA2
- 011 CA3
- 100 CA4
- 101 CA5
- 110 CA6
- 111 CA7

P2CA0: 第 2 位 输入选择。该位和 P2CA4 一起，在 CAEX=0 时，选择同相端输入，在 CAEX=1 时，选择反相端输入。

- 00 无连接
- 01 CA0
- 10 CA1
- 11 CA2

CAF: 第 1 位 比较器 A+输出滤波器

- 0 对比较器 A+输出不作滤波处理
- 1 不对比较器 A+输出作滤波处理

CAOUT: 第 0 位 比较器 A+输出。该位反映了比较器的输出值。写该位不起作用。

(3) 比较器 A+端口禁用寄存器——CAPD

7	6	5	4	3	2	1	0
CAPD7	CAPD6	CAPD5	CAPD4	CAPD3	CAPD2	CAPD1	CAPD0
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

CAPD_x: 第 7-0 位 比较器 A+端口禁用。这些位单独禁用比较器 A+有关的端口引脚的输入缓冲器。例如，CA0 在 P2.3 引脚上，CAPD_x 位可以单独使能或禁用每一个 P2. x 引脚的缓冲器。CAPD0 禁用 P2.0，CAPD1 禁用 P2.1 等等。

- 0 使能输入缓冲器
- 1 禁用输入缓冲器

注：寄存器位的规约：

每一个寄存器的每一个单独的位都用一个键来指示该位的可访问性和初始值：

表 3.21 寄存器位的规约

键	位可访问性
rw	可读/写
r	只读

r0	读出值为 0
r1	读出值位 1
w	只写
w0	写入值为 0
w1	写入值为 1
(w)	无寄存器位执行；写入 1 将产生一个脉冲。寄存器位读出值为 0。
h0	硬件清 0
h1	硬件置位
-0, -1	上电清除 (PUC) 后的初始值
-(0), -(1)	上电复位 (POR) 后的初始值

例：利用 MSP430G2553 单片机内置的比较器 CA5 作为外部输入信号的输入端口，连接到内置比较器的反相输入端，同时将 0.5VCC 参考电压连接到比较器的同相输入端，P1.6 口接有绿色 LED 等，观察灯光变化情况。

```
#include "msp430g2553.h"
void main()
{
    WDTCTL=WDTPW+WDTHOLD; //关闭看门狗
    CACTL1=CAON+CAIE+CAREF_2+CAEX+CARSEL; //打开比较器，允许中断，选择//0.5VCC
                                         为参考电压

    CACTL2&=~P2CA2;
    CACTL2=P2CA3+P2CA1+CAF; //选通 CA5 作为输入，输出作滤波处理
    P1DIR&=~BIT5;           //P1.5 作为输入口
    P1SEL|=BIT5;           //P1.5 作为第二功能即作为比较器输入口使用
    P1DIR|=BIT6;           //P1.6 作为输出控制发光二极管的亮灭
    _EINT();               //打开总中断
    while(1);
}
#pragma vector=COMPARATORA_VECTOR //比较器中断声明的固定格式
__interrupt void COMPARE(void)    //比机器中断处理程序，无返回值
{
    P1OUT^=BIT6;                //将 P1.6 引脚取反，即使发光二极管闪烁
}
```

第八节 Grace 软件技术

经过前面几章的讲解，相信大家已经对 MSP430G2xxx 系列单片机有了一定的了解。初学者可能会为它功能强大的功能和丰富的外设而惊叹，同时又对难以迅速而有效地记住并熟练配置各种寄存器而感到担忧。

TI 是一家非常注重用户体验的公司，他们当然也考虑到了这个问题，在推出其产品的同时，也配套推出了简单易用的图形化 I/O 与外设配置软件——Grace，Grace 是基于 GUI 的 I/O 与外设配置的软件工具，适用于 MSP430F2XX/G2XX 器件。Grace 使得开发人员能够生成经全面注释的易读型 C 代码并快速完成外设的配置，利用 Grace 代码的生成，快速启动开发工作，使开发者可以在数分钟内完成 MSP430 单片机的外设模块的配置，并可以生成 C 语言代码，极大的缩减了开发者花在配置外设上的时间，缩短了开发周期。

Grace 软件在 TI 官网上以两种形式发布，一是作为 Code Composer Studio (CCS) 的一个插件，如果您使用的是 CCS5.1 以上版本，那么 Grace 插件已经被包含在软件中了；如果您使用的是 CCS4.2，那么您可以单独在 TI 官网下载 Grace 插件，安装后即可使用；二是 Standalone 版本，它可以在您电脑上独立运行，您只需把最终生成的代码复制到您的工程中即可。

下面就以 CCS 5.2 版本为例，使用 Grace 软件实现 LED 闪烁：

3.8.1 创建 Grace 工程

如图 3.8.1 所示，打开 CCS，Project→New CCS Project，弹出如下对话框 Project name 由用户来命名，但是不能含有中文。

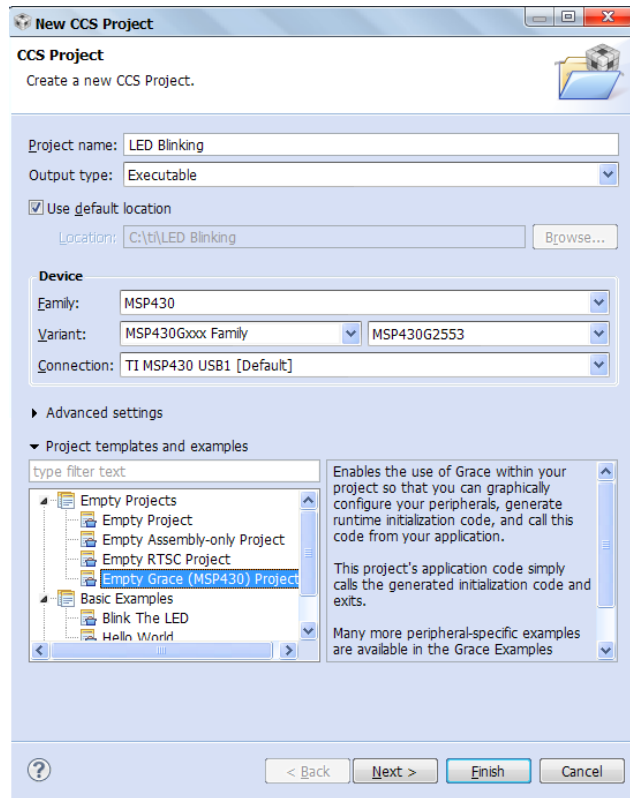


图 3.8.1 创建 Grace 工程

在这里填 LED Blink。Family 选择默认的 MSP430，因为 CCS 5.2 还可以用于 TI ARM 处理器的开发，所以如果你安装了 ARM 模块，还需要在这里进行选择。Variant 选择您所使用的芯片，本例中选择 MSP430G2553。Project templates and examples 选择第一个 Empty Projects 中的 Empty Grace(MSP430) Project。其余项保持默认即可，点击 Finish 完成 Grace 工程的创建。

3.8.2 使用 Grace 配置 I/O 口及外设

本应中用到 IO 口及 TA 定时器。现在分别对它们进行配置。

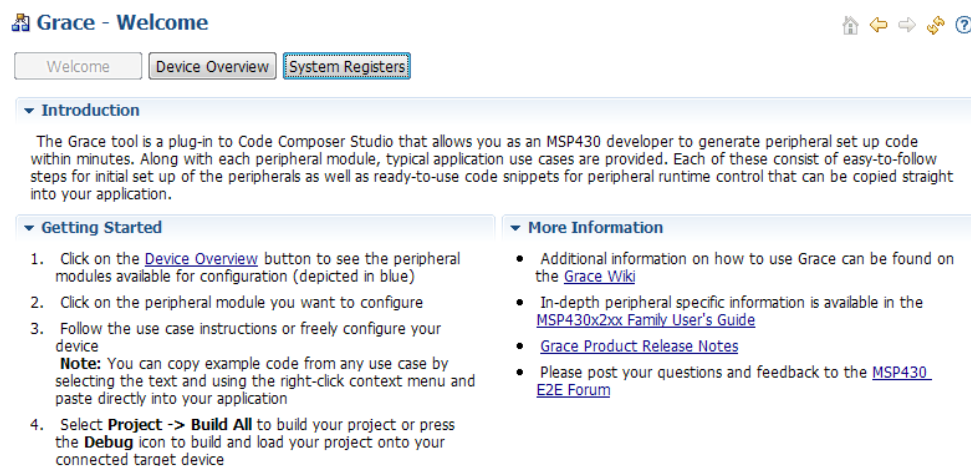


图 3.8.2 Grace 的起始界面

1. 点击 Device Overview 按钮，等待数秒钟，Grace 会把片上可供配置的外设模块(用蓝色标记)列出来，如果您想要使用哪个模块，只需在该模块上单击鼠标右键，选择 Use xxxx，正在使用的模块左下角会有一个绿色的标记。如图 3.8.3 所示：

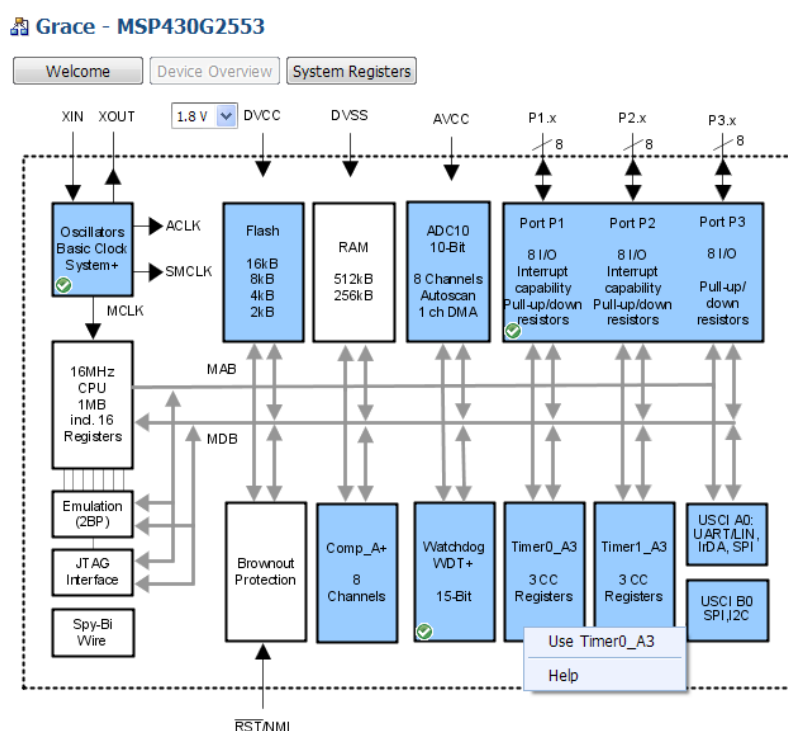


图 3.8.3 片上资源框图

2. 或者直接点击某个模块，出现图 3.8.4 所示界面，选中 Enable Timer0_A3 in my configuration:

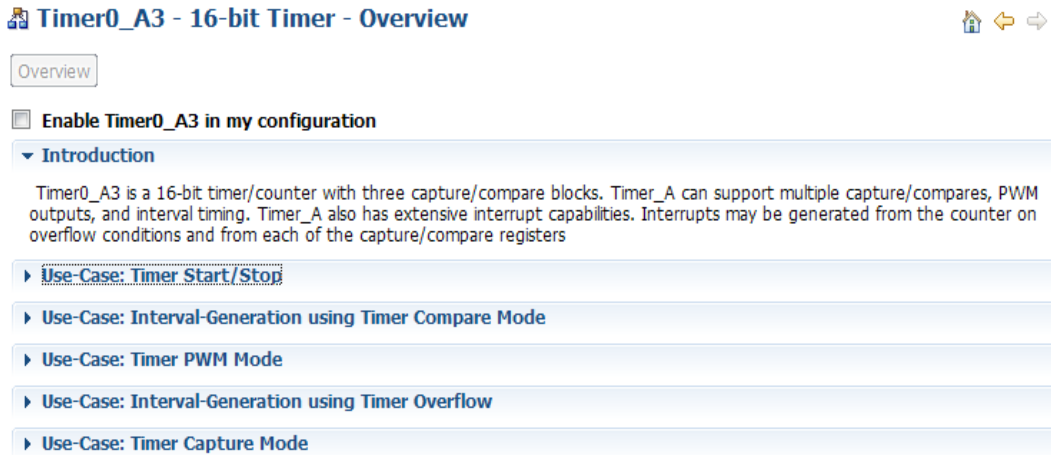


图 3.8.4 选择 Timer0 模块

3. 等待数秒钟后，出现图 3.8.5 所示界面。在 Basic User 里面可以进行一些简单的设置，如果您需要使用更复杂、更高级的功能，点击 Power User。

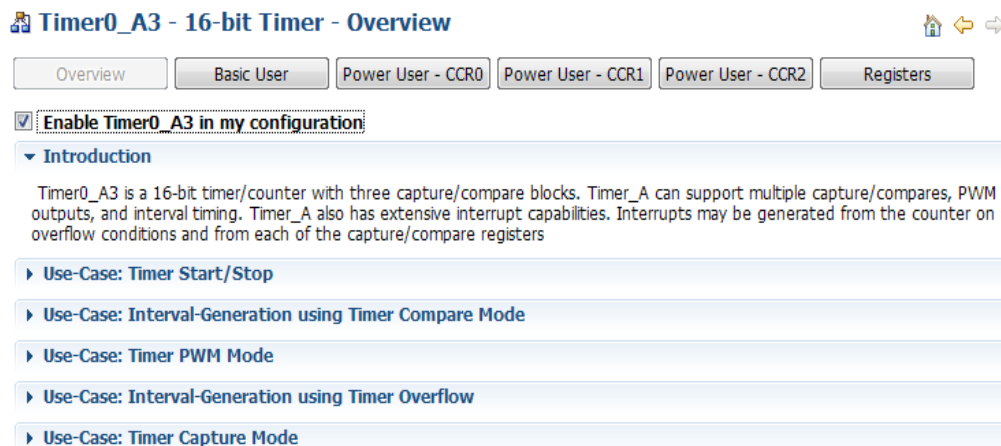
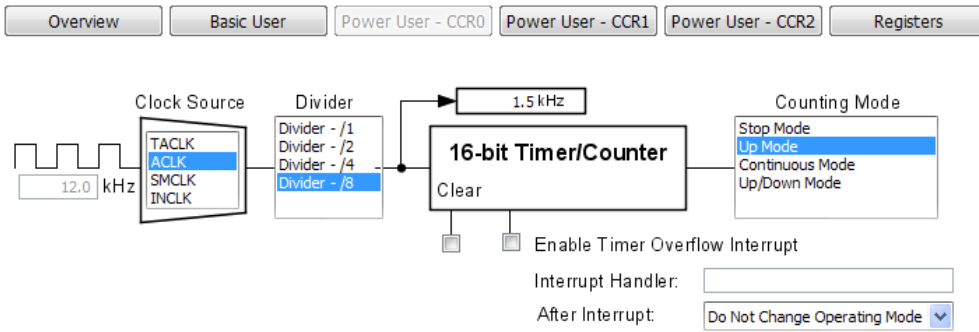


图 3.8.3 使能 Timer0 模块

4. 现在就可以对 TA 定时器进行配置了。要让 LED 每秒钟闪一次，即需要产生一个频率为 1Hz 的中断，在这里选择 ACLK 为时钟源，再进行 8 分频，可以看到一个 1.5kHz 的时钟就产生并被送到 TA 定时器了。选择计数模式为增计数模式，在 Desired Timer Period 或 Capture Register 里面设置定时时间或时钟脉冲数，Grace 会自动算出定时周期和频率，这点非常方便。由于只需要产生一个定时间隔，所以选择输出比较模式，禁用输出管脚。如果要输出 PWM 波，则可选择对应的管脚，然后在 Output Mode 中选择相应的输出模式。注意 CCR0 寄存器不能设为模式 2、3、6 和 7，如果设置了其中任何一个，Grace 会自动报错并提示出错的原因。这里可以看到左边的 Input Selection 和 Capture Mode 两栏变为灰色，即不能进行选择，因为在输出捕获模式下没有意义。只有选择输入捕获模式，灰色部分才会被激活。再在下面使能捕获/比较中断，填写中断句柄，该句柄相当于是自定义的一个中断服务函数，在中断函数中被调用。最后选择中断之后的状态，低功耗休眠或唤醒 CPU。在本应用中的配置如图 3.8.6 所示：

Timer0_A3 - 16-bit Timer - Power User Mode - CCR0



Timer Capture/Compare Block #0

Desired Timer Period: ms Time(r) Period: 1 Hz

Capture Register: Clock Ticks Time(r) Frequency: 1 Hz

Input Selection: Capture Mode: Mode: Output Pins:

Output Mode: Set OUT bit High/Low

Enable Capture/Compare Interrupt

Interrupt Handler: After Interrupt:

图 3.8.6 配置 Timer0 模块

也可以点击 Register 查看寄存器每一位的值。当不清楚某一位的功能时，只需把鼠标移动到这一位上面，Grace 会自动弹出提示信息。如图 3.8.7 所示：

Timer0_A3 - 16-bit Timer - Register Controls



Overview Basic User Power User - CCR0 Power User - CCR1 Power User - CCR2 Registers

TA0CTL, Timer_A Control Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Unused								TASSELx ACLK	IDx Divider - /8	MCx Up Mode	Unused	TACLRL	TALR	TAIFG [R/W]	

TA0CCTL0, Capture/Compare Block #0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CMx No Capture	CCISx CCIxA	SCS	SCCl	Unused	CAP	OUTMODx PWM output mode: 0 -		CCIE <input checked="" type="checkbox"/>	CCI [R]	OUT	COV [R/W]	CCIFG [R/W]			

TA0CCTL1, Capture/Compare Block #1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CMx No Capture	CCISx CCIxA	SCS	SCCl	Unused	CAP	OUTMODx PWM output mode: 0 -		CCIE <input type="checkbox"/>	CCI [R]	OUT	COV [R/W]	CCIFG [R/W]			

TA0CCTL2, Capture/Compare Block #2

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CMx No Capture	CCISx CCIxA	SCS	SCCl	Unused	CAP	OUTMODx PWM output mode: 0 -		CCIE <input type="checkbox"/>	CCI [R]	OUT	COV [R/W]	CCIFG [R/W]			

TA0CCR0 **TA0CCR1** **TA0CCR2**

150 150 150

[R/W] Read/Write register not available in GUI

图 3.8.7 Timer0 寄存器

5. 下面点击左下角的 Grace 标签或右上角的 图标，回到图图 3.8.3 下面配置 I/O 口。

点击 I/O 口模块，根据所使用的芯片封装点击相应的按钮。在此选择 Pinout 20-TSSOP/20-TDIP，一个 MSP430G2553 芯片就出现了，每个管脚外侧都会显示 IO 口的当前设置，点击蓝色向下箭头，就可以重新配置 IO 口。在些配置 P1.0 为 GPIO Output，用来驱动 LED。图 3.8.8 所示：

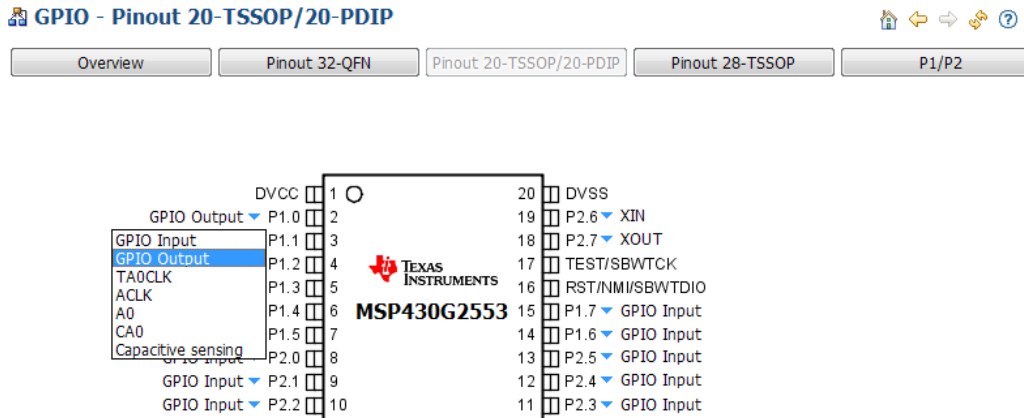



图 3.8.3 IO 口配置

6. 这样 Grace 就配置完了，然后点出 CCS 工具栏上的  图标，生成代码，这时会提示错误，这是因为刚才定义的中服务函数没有定义。右键单击 main 函数里面的 CSL_init() 函数，选择 Open Declaration，打开 CSL_init.c 文件，或者在 Project Explorer 中找到并打开 CSL_init.c 文件。如图 3.8.9 所示：

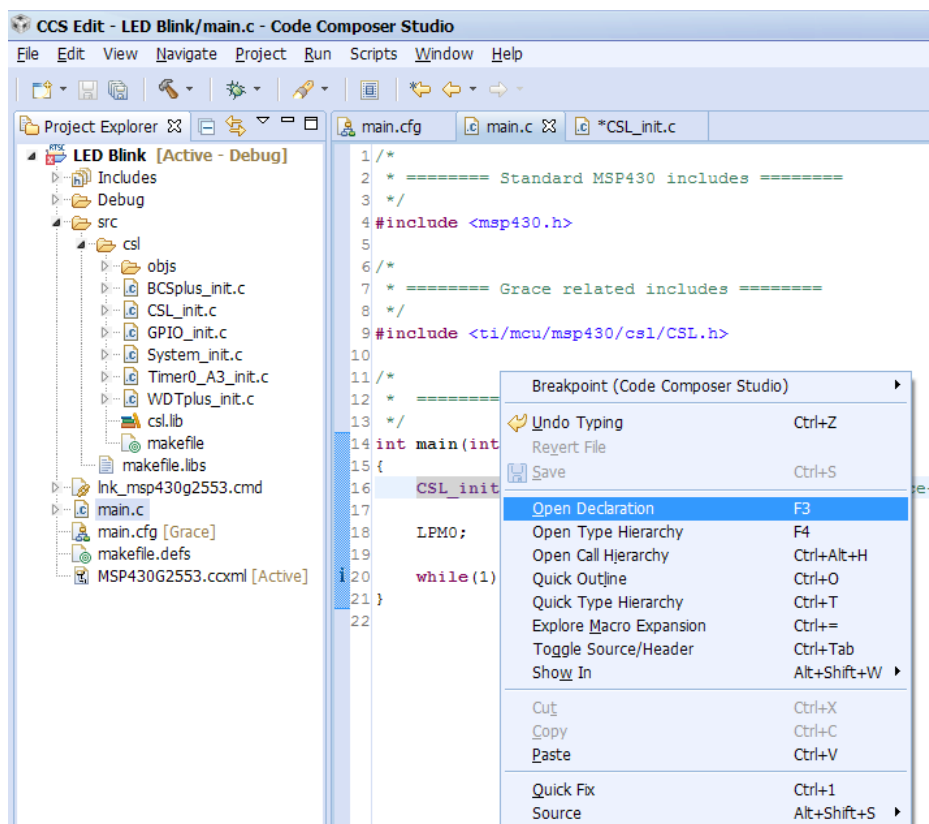



图 3.8.9 软件调试

在如图 3.8.10 所示位置（蓝色部分）添加代码，然后编译，没有错误，然后连接 LaunchPad，点击  图标开始下载仿真。

```

/*
 * ===== Interrupt Function Definitions =====
 */

/* Interrupt Function Prototypes */
extern void TAO_ISR(void);

void TAO_ISR(void)
{
    P1OUT ^= BIT0;
}

/*
 * ===== Timer0_A3 Interrupt Service Routine =====
 */
#pragma vector=TIMER0_A0_VECTOR
__interrupt void TIMER0_A0_ISR_HOOK(void)
{

    /* Capture Compare Register 0 ISR Hook Function Name */
    TAO_ISR();


    /* No change in operating mode on exit */
}

```

图 3.8.10 添加代码

点击全速运行，可以看到 LED 大概每秒钟闪一次，任务完成了。

Grace 的 Standalone 版本和 CCS 版本差别不大，下面以 ADC 内部温度传感器为例，重点讲解一下二者的区别。

(1) 打开 Grace 后点击  图标新建一个 Grace 工程，输入工程名，同样不能包含非 ASCII 码字符，选择要保存的位置，选择您所使用的芯片，如图 3.8.11 所示。然后点击 OK 完成。

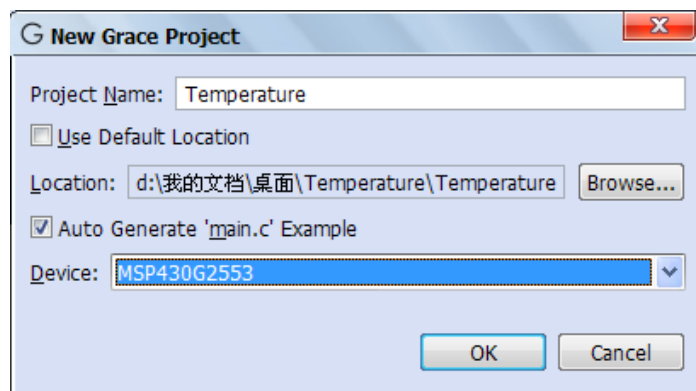


图 3.8.11 新建 Grace 工程

(2) 同样来到图 3.8.3 的界面，选择时钟模块。MSP430G2553 单片机在 Flash 中存了四个数字时钟振荡器 DCO(Digital Controlled Oscillator)的频率校准点，分别为 1MHz、8MHz、12MHz、16MHz，用户可以根据需要选择，从而省去了手工配置 DCO。当然 Grace 的强大之处在于，它不仅有这四个频率可供选择，而且用户可以自定义频率，Grace 会根据用户输入的值自动配置好相应寄存器。注意，如果 CPU 要运行在比较高的频率，在图 3.8.3 中，

DVCC 应该选择 3.3V，不然 Grace 会报错。本例中 MCLK 配置为 16MHz，SMCLK 2MHz，ACLK 的时钟源为 VL0(Very Low Power Low Frequency Oscillator，大约 12kHz)。并且可以在 P1.0 上输出 ACLK，如图 3.8.12 所示。Grace 的高明之处在于，在某模块设置 IO 口后，在 GPIO 模块，相应的 IO 口会同时改变。

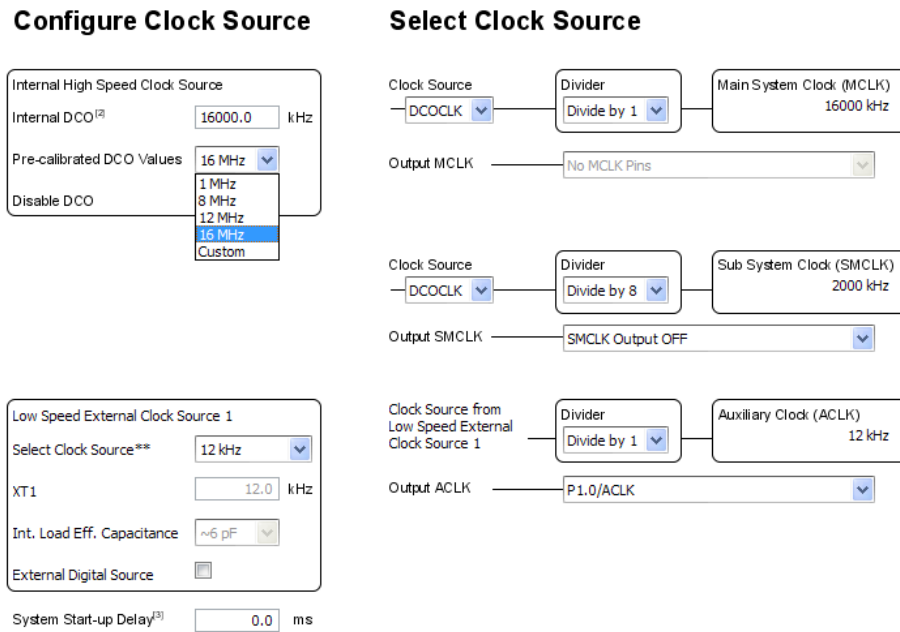


图 3.8.12 基础时钟模块

(3) 配置 ADC10 模块。选择采样通道，注意，必须同时选中通道对应的外部 IO 口。现在如果我们选择通道 0 和 P1.0，Grace 会报错，因为刚才我们已经把 P1.0 用作 ACLK 的输出了。Grace 的这个优点避免了配置冲突的错误。

ADC10 - 10-bit SAR - Power User Mode

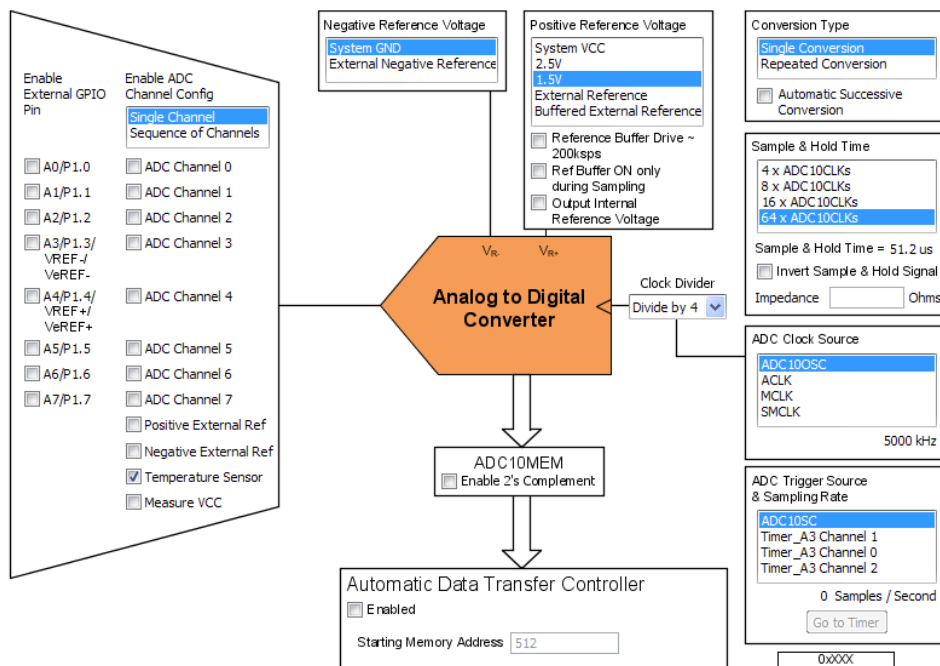



图 3.8.13 ADC10 模块

然后设置正负参考电压、转换类型、采样保持时间、ADC 时钟源和触发信号，最后打开中断，输入中断句柄，如图所示。时钟源选择 ADC100SC，这样配置的好处是可以使单片机进入更低功耗的休眠而不影响 ADC 转换。如果触发信号选择定时器，Go to Timer 按钮就会生效，点击可以进入定时器进行相应的配置。使用定时器触发 ADC 转换，甚至可以省去中断，降低了软件的复杂度，还可进一步降低功耗。

(4) 配置完成后，保存并点击  生成代码。然后就可以在..\Temperature\cs1(..\表示相对目录，即工程所在的文件夹)里面看到生成的代码了。然后将其复制到您的应用中就可以了。

第九节 MSP430G2 系列单片机调试接口 JTAG 和 SBW

3.9.1 JTAG 简介

JTAG 是一种国际标准测试协议，用于对 MSP430 进行在线仿真。任一款 msp430 都支持 JTAG 协议。标准的 JTAG 接口是 4 线：TMS、TCK、TDI、TDO，分别为模式选择、时钟、数据输入和数据输出线。

JTAG 最初是用来对芯片进行测试的，JTAG 的基本原理是在器件内部定义一个 TAP (Test Access Port; 测试访问口) 通过专用的 JTAG 测试工具对进行内部节点进行测试。JTAG 测试允许多个器件通过 JTAG 接口串联在一起，形成一个 JTAG 链，能实现对各个器件分别测试。现在，JTAG 接口还常用于实现 ISP (In-System Programmable; 在线编程)，对 FLASH 等器件进行编程。

JTAG 编程方式是在线编程，传统生产流程中先对芯片进行预编程现再装到板上因此而改变，简化的流程为先固定器件到电路板上，再用 JTAG 编程，从而大大加快工程进度。JTAG 接口可对 PSD 芯片内部的所有部件进行编程。

具有 JTAG 口的芯片都有如下 JTAG 引脚定义：

TCK——测试时钟输入；

TDI——测试数据输入，数据通过 TDI 输入 JTAG 口；

TDO——测试数据输出，数据通过 TDO 从 JTAG 口输出；

TMS——测试模式选择，TMS 用来设置 JTAG 口处于某种特定的测试模式。

可选引脚 TRST——测试复位，输入引脚，低电平有效。

含有 JTAG 口的芯片种类较多，如 CPU、DSP、CPLD 等。

JTAG 内部有一个状态机，称为 TAP 控制器。TAP 控制器的状态机通过 TCK 和 TMS 进行状态的改变，实现数据和指令的输入。

3.9.2 JTAG 接口

对于运用 MSP430 进行开发的工程师而言，我们只需学会将仿真器上的引脚分别连接到芯片上即可。

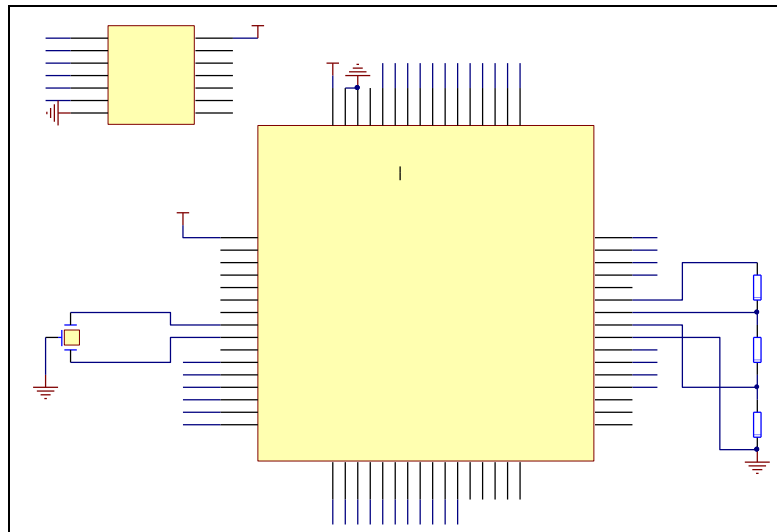


图 3.9.1 JTAG 连接图 1

如图所示为标准的 JTAG 连接方法：仿真器上的 TDO, TDI, TMS, TCK, GND, RESET 分别连接到单片机上的 TDO/TDI, TDI/TCLK, TMS, , TCK, GND, RST/NMI 端。

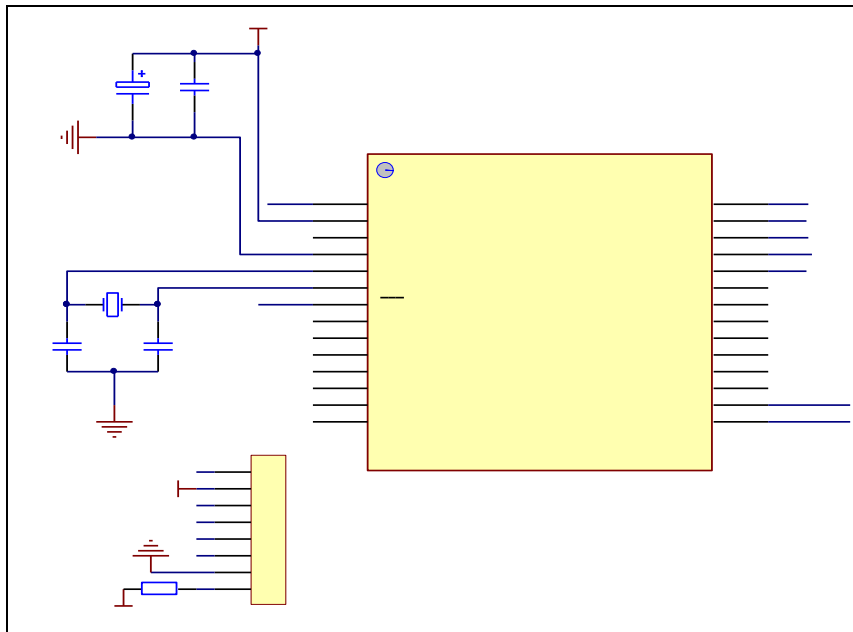


图 3.9.2 JTAG 连接图 2

也有如图 3.9.2 所示的下载仿真接口，该接口多出了一个 TEST 口，用于帮助仿真器区分单片机处于仿真状态还是运行状态，将它与单片机的 TEST 脚直接连起来。

3.9.3 SBW 接口

对于一些引脚较少的单片机来说，使用大量引脚来完成仿真下载工作是不合适的。比如，MSPG2231 只有 14 个管脚，而 JTAG 则需要至少 5 个管脚，这显然不合理。因此 TI 提出了两线制下载的解决方案 SBW，它之运用两根线就可完成下载功能，及其的简洁方便。具体连接方法以 MSP430G2553 单片机为例：只需将单片机的 16 脚 SBWTDIO 和 17 脚 SBWTCK 分别与仿真其上的 TDO 和 TCK 连接起来即可。

特别需要注意的是：无论在 JTAG 还是在 SBW 上都应该注意，如果单片机靠仿真器上提供的电压供电，则需要将 VCC 接到仿真器上 2 号脚上，如果有外接电源，则将仿真器的第 4 脚接到 VCC 上，最好的方法是设置一个开关进行选择如图 3.9.3 所示。

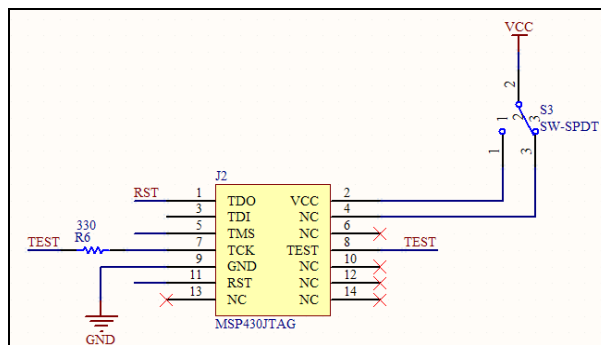


图 3.9.3 JTAG 供电

第十节 触摸按键

触摸按键可分为两类：电阻式触摸按键与电容式感应按键，即滑动式按键和点触式按键。

3.9.1 电阻式按键

电阻式的触摸按键原理非常类似于触摸屏技术，需要由多块导电薄膜上面按照按键的位置印制成的，因此这种按键需要在设备表面贴一张触摸薄膜。电阻式触摸屏一直由于其低廉的价格而深受厂商的喜爱，但是由于导电薄膜的耐用性较低，并且也会降低透光性，因此已经被越来越多的厂家所抛弃。

3.9.2 电容式按键

电容式触摸按键主要是为了克服电阻屏的耐用性所提出的，电容式触摸按键的结构与电阻式的相似，但是其采用电容量为判断标准。简单来说，就是一个 IC 控制的电路，该电路包括一个能放置在任何介质面板后的简单阻性环形电极组件，因此，按键的操作界面可以是一整块普通绝缘体（如有机玻璃一般材料都可），不需要在界面上挖孔，按键在介质下面，人手接近界面和下面的电极片形成电容，靠侦测电容量的变化来感应。温度，静电，水，灰尘等外界因素一般不会影响，界面没有太多要求，可以加上背光，音效等，靠人手感应，整个界面没有按键的存在，便于清洁，让产品在外观上更加高档美观，由于按键没有接点，使用寿命也是非常的长久，一般来说是半永久性。

1. 电容式感应按键的原理

电容感应原理是利用人体的感应电容来检测是否有手指存在，在手指没有按下时，按键上由于分布电容的存在，因此按键对地存在一定的静态电容，当人的手指按下或者接近按键时，人体的寄生电容将耦合到这个静态电容上，使按键的最终电容值变大，该变化的电容信号再输入到单片机进行信号转换，将变化的电容量转换成某种电信号的变化量，再由一定的算法来检测和判断这个变化量的程度，当这个变化量超过一定的域值时就认为手指按下。

在电路上任意两个相邻的极板之间形成一个等效电容，因此在电容感应的应用中就人为的创造了这样的两个电极，以此来获取一个等效电容。如图 3- 的示意图，在没有手指按下时，各按键对地有一个等效的静态电容 $C_{基}$ ，当有手指按下时，人体电容 ΔC 就叠加到按键上，使按键对地的电容增加，这样获得的电容再输入到芯片中进行处理，即 $C=C_{基}+\Delta C$ 。

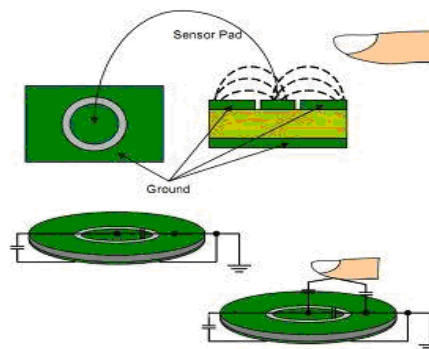


图 3.10.1 电容感应原理图

2. 触摸按键的实现

电容式感应触摸按键实际只是 PCB 上的一小块覆铜焊盘，当没有手指触摸时，焊盘和地信号产生的电容成为“基准电容”。

触摸按键最关键的设计是在 PCB 排版上，为了获得稳定的且较高的灵敏度，应遵从以下原则：基电容容值小、基电容稳定抗噪、尽量获得大的 ΔC 。

常见的作法有：优化触摸铜盘大小尺寸、减小铜盘与地之间的距离、减小铜盘与手的距离、触摸盘背面铺地用 30%~60%的网格铺设等。

表面电容按键设计方法：

一个表面电容按键由一个与控制器相连接的单端铜电极组成。它不需要有太高的灵敏度，只需能够识别手指的触摸与否。

(1) 按键形状

任何形状的按键均可用于电容感应式触摸中，如图 3- 所示。不同的形状不会影响感应的性能，仅与板子的美观程度有关。

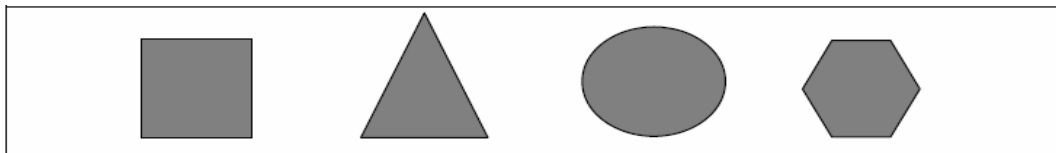


图 3.10.2 按键形状图

(2) 按键尺寸

在其它条件都相等的情况下，通常按键越大越好。两个使用相同走线与控制器连接的按键，如果它们的大小不同，则灵敏度也将不同。

小的按键因其表面积小，触摸电容 (CT) 也很小，相应地灵敏度会较差，按键过大并不会显著提高 CT。但是，将按键面积增大至与触摸物(手指，拇指等)相当，则会显著提高 CT。对于手指感应，推荐按键的直径至少为 0.4 英寸(10mm)。较小的按键也可工作，但性能较差。大的按键灵敏度较高，但感应式触摸物的有效接触面积决定了按键尺寸的上限。

(3) 按键间距

按键可以相互邻近，但如果节距太小，相互之间太过靠近，则会带来不期望的相互影响。

(4) 按键-地的间隙

如果可能，地层不应与感应元件放在同一层。如果地层与感应元件太近，将会增加电容并影响对手指存在与否的检测。

(5) 电容感应应用中的走线

触摸感应控制器与传感器之间的走线会增加按键电容并降低信号，从而降低传感器的灵敏度。走线长度会降低灵敏度，因为它会增加感应电路的并联电容。走线长度也会增加噪声，因为走线同时会受内部电路与外部噪声环境的影响。

(6) 走线长度

缩短从控制器至传感器之间的走线长度可降低其它元件与走线产生耦合的风险。触摸感应控制器与传感器之间的走线应尽可能短。

(7) 走线宽度

线宽会增加整个系统的铜覆盖面积，从而增加传感器的电容。同时也会增加与其它层上元件的耦合。因此，走线应尽可能细小，且远离地层。

(8) 走线放置

无论何时，电容感应中的走线放置必须以与其它设计元件，包括其它电容感应走线的相互影响最小为准则。同时，将走线放置在用户 PCB 的背面可降低手指对走线的影响，以确保传感器引脚上的所有电容值变化是来自于手指(或其它导体)与有效传感面积的相互作用，而不是来自于手指与走线的相互作用。

(9) 分组放置

触摸感应控制器同一端口上的引脚由软件库一起驱动，并通过软件映射在同一组。同一组中的引脚走线可最大限度地靠近在一起。

第四章 MSP430G2 系列单片机一体化实验系统

通过前三章的学习，相信读者对 MSP430G2xx 系列单片机的特点、其开发软件的使用方法、及其单片机内部资源的使用方法有了很好的掌握。学习单片机的最好途径就是实践。对于初学者来说，接下来的问题就是：如何运用掌握的知识进行实践？进行什么样的实践？

为帮助读者进行单片机学习实践，本章介绍一种 MSP430G2 系列单片机一体化实验系统。该系统采用美国 TI 公司最新推出的 MSP430G2 超低功耗系列 LaunchPad 作为核心板，引入模块化设计理念，采用系统底板加功能模块的结构形式，基本配置 14 种实验模块，可完成十余个实验，让使用者方便地进行单片机学习实践。同时，使用者也可以根据实验系统预留的接口和提供的 MSP430G2553 单片机核心板，设计自己开发的功能模块和实用性功能设备。

本实验系统包括一个实验系统标准板、一个实验系统简化板和 14 种配置的实验模块，本章将具体对它们的适用对象、基本功能、适用方法、结构原理、连接方式等进行具体的介绍。

第一节 MSP430G2 系列单片机一体化实验系统标准板

4.1.1 实验系统标准版结构组成

MSP430G2 系列单片机一体化实验系统标准版由系统底板、功能模块区和扩展实验区组成。

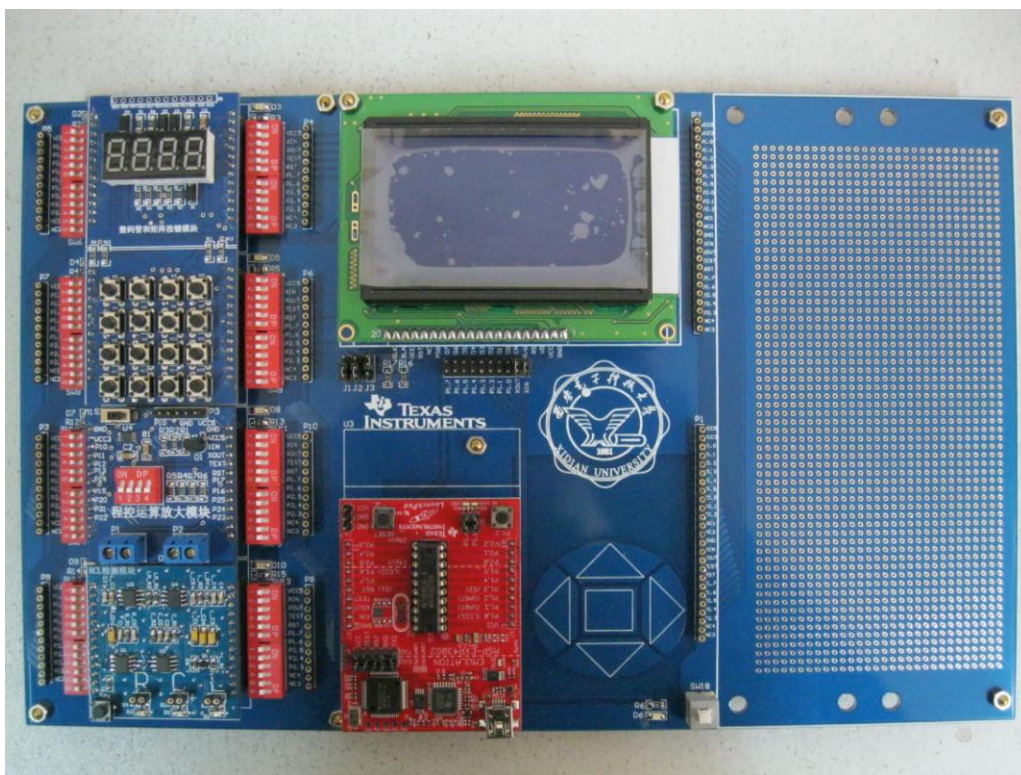


图 4.1.1 MSP430G2 系列单片机一体化实验系统标准板实物图

4.1.1.1 功能模块区

如图 4.1.1 所示，实验板的左端是实验板的功能模块区，功能模块区留有四个实验模块插槽，插槽的两端各有 2 个 6 位的拨码开关，拨码开关用于选通插槽端口和 LaunchPad 的连接。另外模块插槽两边还各引出一排扩展口，用于外接扩展电路，也可用来测试 LaunchPad 上接口的状态。最后每个模块插槽对应两个电源指示灯。功能模块区的电路连接图如图 4.1.2 所示。

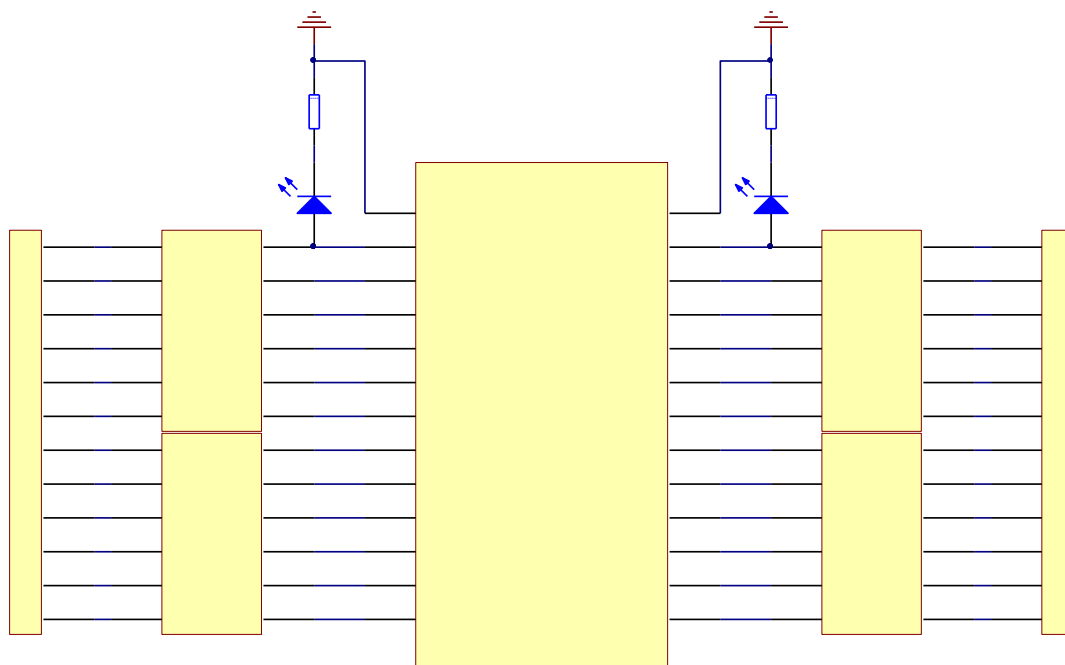


图 4.1.2 实验板功能模块区中单个模块插槽部分的电路连接

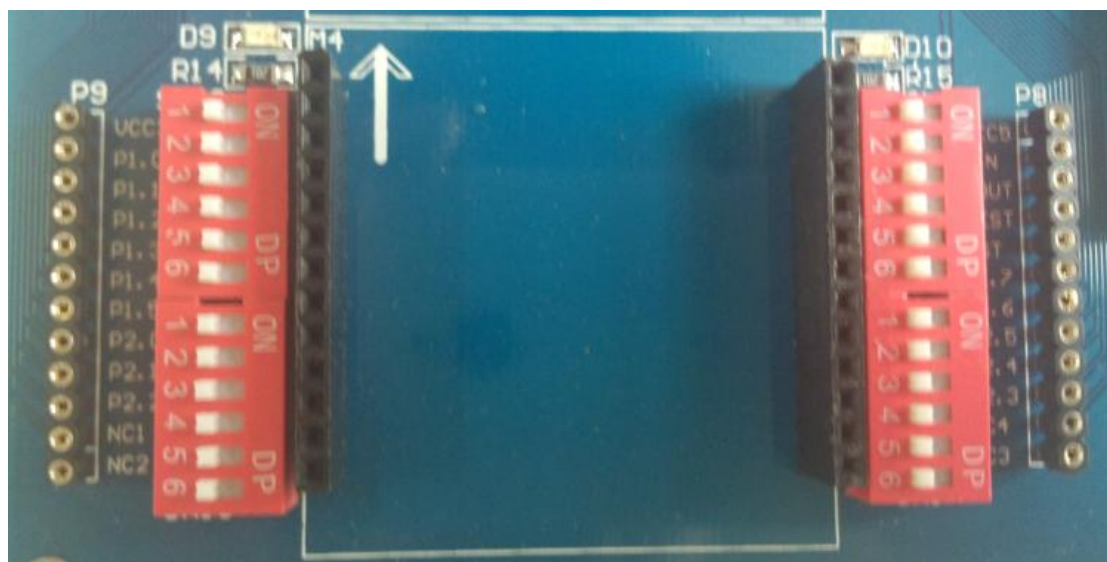


图 4.1.3 实验板功能模块区中单个模块插槽部分的实物图

如图 4.1.3 所示，拨码开关和扩展口和它们之间的丝层字符相对应。在使用模块时，将模块按图 4.1.3 中丝层箭头指向对齐插入模块插槽中，然后开启要联通的拨码开关，向右拨动为开启，例模块要用到 VCC3.3 和 P1.0，就将拨码开关 SW1 中的前两位开关合上。系统上

电后，模块就可以供给 LaunchPad 核心板使用。

4.1.1.2 系统底板区

图 4.1.1 所示实验板的中间部分就是系统底板区，系统底板区包括 LaunchPad 插槽、LCD12864 插槽及其跳线和触摸按键三个部分。

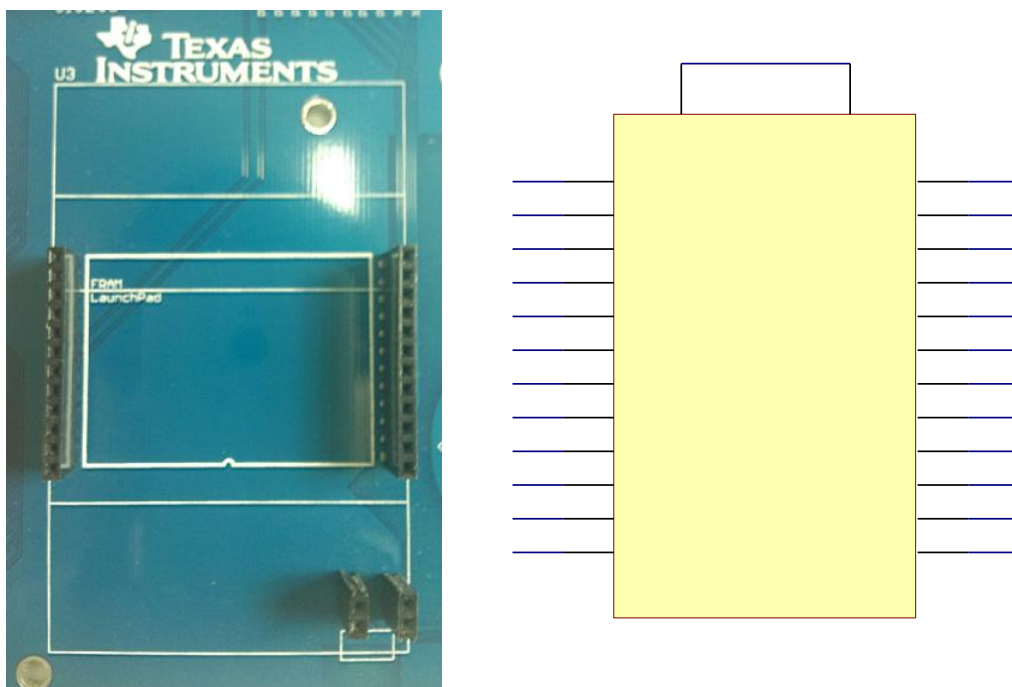


图 4.1.4 LaunchPad 插槽实物图和电路连接图

图 4.1.3 是 LaunchPad 插槽的实物图和电路连接图。使用 LaunchPad 时，将 LaunchPad 按图 4.1.1 所示 USB 口朝下，以 LaunchPad 插槽低端为基准对齐插入。

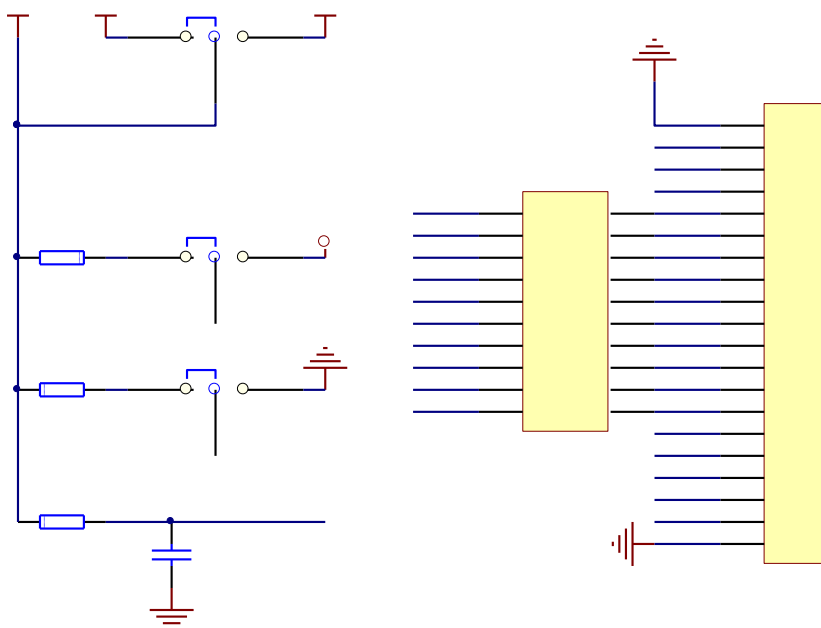


图 4.1.5 LCD12864 显示电路连接

图 4.1.3 是 LCD12864 显示电路的原理图。LCD12864 有 5V 供电和 3.3V 供电两种，有串行和并行两种工作方式，在串行工作方式下，只有 R/W 和 E 两根线和 LaunchPad 核心板相连，同时 RS 电平置高、PSB 电平置低；在并行方式下，图 4.1.4 中 LCD12864 从 RS 到 DB7 这 11 根线都要和 LaunchPad 核心板相连，同时 PSB 电平要置高。使用者可以根据自己的需要改变跳线的连接，来选着 LCD12864 的工作方式。有关 LCD12864 的内容后面章节会详细介绍。

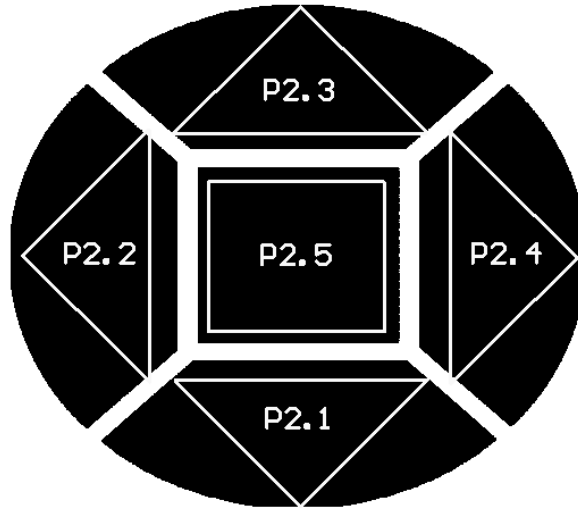


图 4.1.6 触摸按键示意图

图 4.1.5 是触摸按键的示意图，图中标明了每个触摸按键对应 LaunchPad 核心板的 IO 口。

4.1.1.3 扩展实验区

图 4.1.1 所示实验板的右端就是扩展实验区。扩展实验区由两排引出的扩展口、固定在底板上的面包板和底板上的焊盘矩阵组成。两排引出的扩展口电路连接如下图所示，它们的接口与 LaunchPad 和模块插槽接口一一对应。

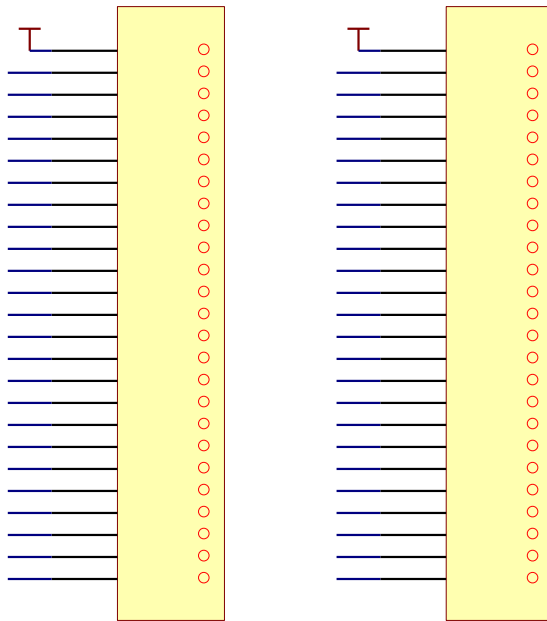


图 4.1.7 扩展区扩展口电路连接

扩展实验区的面包板和焊盘矩阵可以供给使用者搭建电路、设计自己的作品，使用者可将搭建的电路通过其旁边的两排扩展口与 LaunchPad 连接，这样使用者可以方便的使用此实验系统标准版对自己的电路和作品做测试工作。

4.1.2 实验系统标准版使用对象和使用特点

实验系统标准版，可通过提供的 MSP-EXP430G2 LaunchPad 和不同功能的 14 种实验模块，完成 14 种模块各自的实验。使用者也可以将几种模块一起组合使用。同时实验系统标准版留出了扩展区，便于使用者设计自己的电路和实验，当然使用者也可以利用提供的实验模块结合自己的电路完成使用者自己设定的系统性、创新性实验的实验。

MSP430G2 系列单片机一体化实验系统具有开放性、方便使用者使用、操作简单等特点，它不但能够很好地让使用者进行单片机学习实践，也更能激发使用者的创造力和想象力。

第二节 MSP430G2 系列单片机一体化实验系统简化板

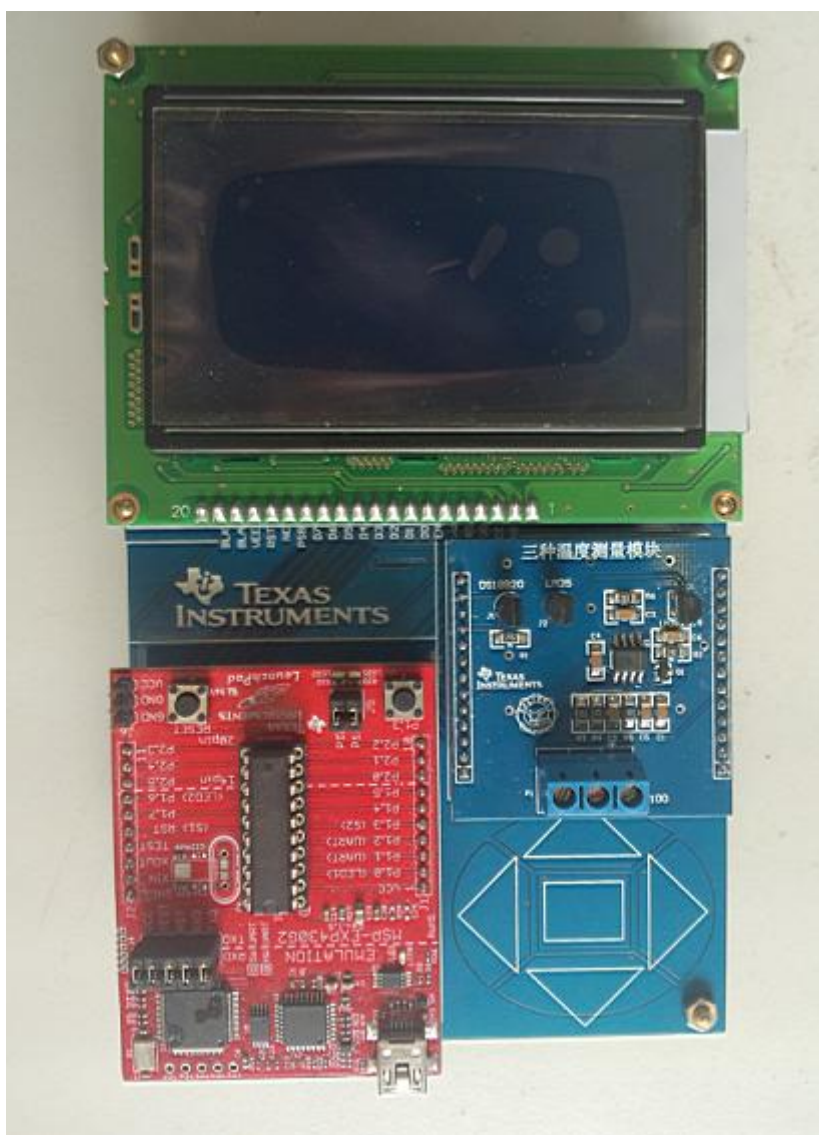


图 4.2.1 MSP430G2 系列单片机一体化实验系统简化板

图 4.2.1 是 MSP430G2 系列单片机一体化实验系统简化板装配以后的实物图。从实物图中我们可以看出,实验系统简化板相比于试验系统标准版来说大大简化了,省略了拨码开关、扩展口和扩展实验区等内容,并且简化板只留出了一个模块插槽。简化板在 LCD12864 液晶显示上也做了改动,标准版可通过跳线选着 LCD12864 的工作方式,而简化板的 LCD12864 只能用串行工作方式。

虽然简化板相比于标准板有以上所说的不足之处,但是其具有体积小、便于携带、成本低等优点。并且简化板保留了标准版的主要功能部分,如触摸按键。

简化板使用方法:将 LaunchPad 如图 4.2.1 所示 USB 口朝下,以 LaunchPad 插槽低端为对准插入;将 LCD12864 有插针的一段朝下,插入 LCD12864 的插槽中;将模块有丝层汉字字符一段朝上,插入模块插槽中;最后上电即可使用。

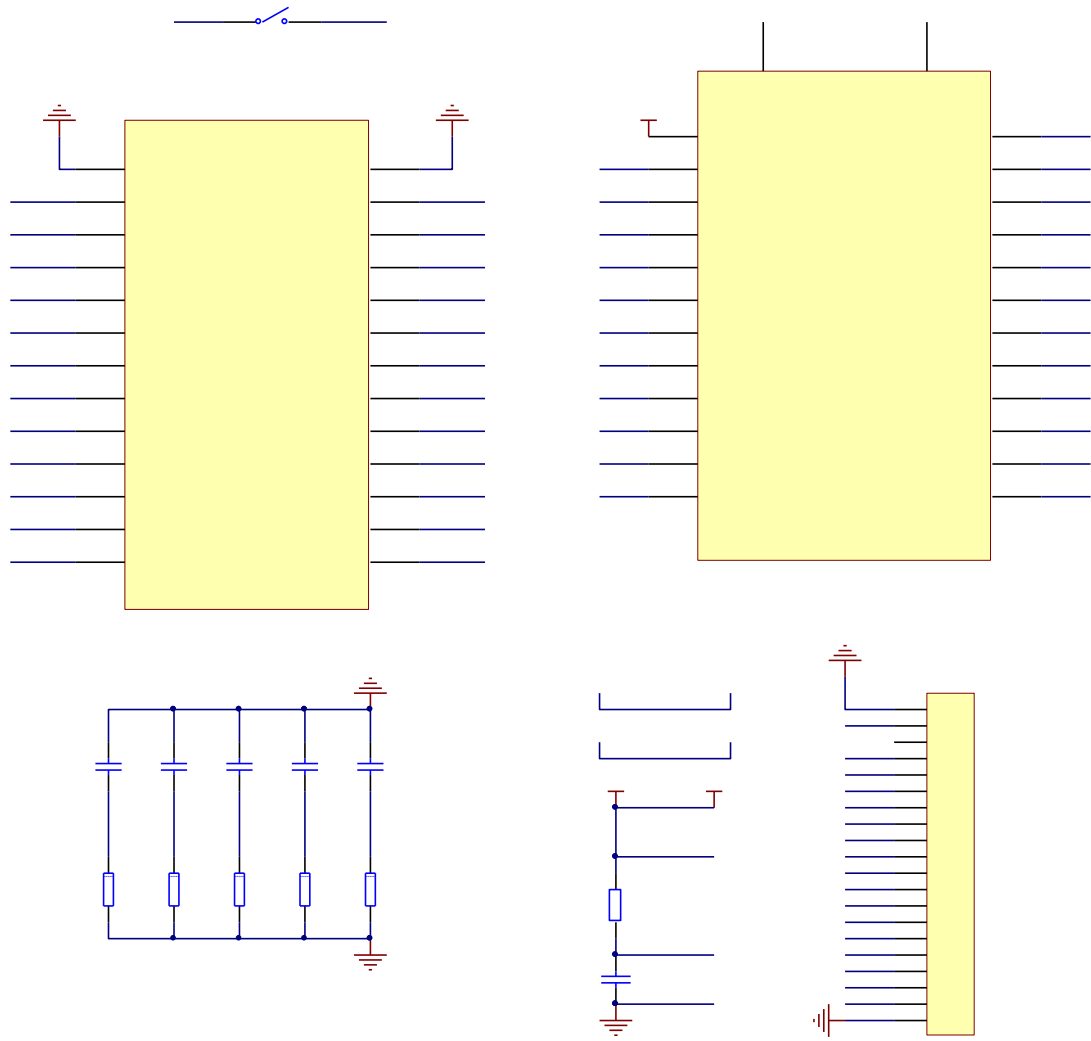


图 4.2.2 MSP430G2 系列单片机一体化实验系统简化板原理图

第三节 实验模块配置

4.3.1 模拟滤波器实验功能模块

4.3.1.1 实验功能模块功能说明

滤波器是一种选频装置，可以使信号中特定的频率成分通过，而极大地衰减其它频率成分。在测试装置中，利用滤波器的这种选频作用，可以滤除干扰噪声或进行频谱分析。

广义地讲，任何一种信息传输的通道（媒质）都可视为是一种滤波器。因为，任何装置的响应特性都是激励频率的函数，都可用频域函数描述其传输特性。因此，构成测试系统的任何一个环节，诸如机械系统、电气网络、仪器仪表甚至连接导线等等，都将在一定频率范围内，按其频域特性，对所通过的信号进行变换与处理。

该实验功能模块具有低通滤波和带通滤波的功能，可以完成对带有杂波分量的频率信号进行滤波和频率的测量。

4.3.1.2 实验功能模块组成及工作原理

图 4.3.1 是该模拟滤波器模块的整体组成框图：

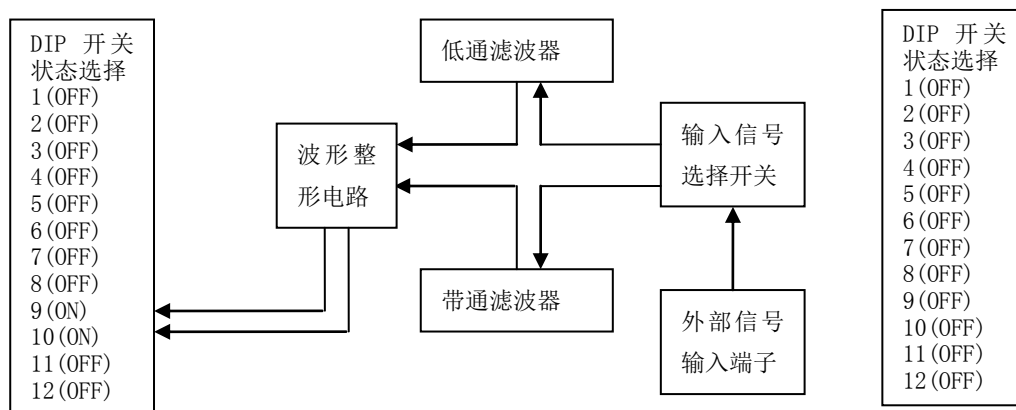


图 4.3.1 模拟滤波器功能模块组成框图

该实验功能模块通过开关的选择，外部带有高频分量的输入信号可以进入低通滤波器或者带通滤波器，从而可以分别实现带通滤波和低通滤波的功能。用示波器检测相应滤波器的输入和输出的波形，对比两波形的平滑程度或者两者幅度的衰减量等指标可以观察带通或者低通的效果。将对从滤波器的输出光滑的特定的频率信号进行整形，处理后形成的高低电平信号进入单片机 MSP430G2553 的捕获管脚，通过单片机一定的运算和数据处理得出频率值。然后单片机将结果送入到显示终端进行显示，方便用户观察当前信号的频率值。

4.3.1.3 二阶有源滤波器概述

滤波器是一种二端口网络，它的作用是允许某频率范围的信号通过，滤掉或抑制其他频率的信号。允许通过的信号频率范围称为通带，其余信号的频率范围称为阻带。许多通过电信号进行通信的设备，如电话、收音机、电视和卫星等都需要使用滤波器。严格的说，实际的滤波器并不能完全滤掉所选频率的信号，只能衰减信号。

无源滤波器通常由 RLC 元件组成，一般采取多节 T 型或 π 型结构，制造难，成本高，特别是电感元件的重量和体积都很大。用 RC 元件与运放集成块构成的有源滤波器，不用电感线圈，因此广泛用于工程电路。此外，运放的开环电压增益很大，输入阻抗高，输出阻抗低，

组成的滤波器有一定的放大、隔离和缓冲作用。

相比于无源滤波器，有源滤波器有许多优点：

- (1) 可以按要求灵活设置增益，并且无论输出端是否带载，滤波特性不变，
- (2) 有源滤波电路一般由 RC 网络和集成运放组成，因而必须在适合直流电源作用下才能正常工作；
- (3) 不用电感，故体积小，重量轻，不需加磁屏蔽；
- (4) 可加电压串联负反馈，使 R_i 高、 R_o 低，在输入与输出之间有良好的隔离性。
- (5) 除能滤波外，还能将信号放大，而且 A_u 易调节。
- (6) 不适宜高电压大电流的负载，只适用于信号处理。
- (7) 不适宜高频范围，频率范围 10^3 的负 3 次方到 10^6 的 6 次方 Hz。
- (8) 有源滤波器有着极高的输入阻抗和极低的输出阻抗，可直接进行级联，不需进行阻抗匹配。同时，有源滤波器电路还可进行增益调整，通过调节桥臂电阻，可补偿电路中的增益衰减。电路对直流信号及低频信号几乎无增益衰减。

二阶有源滤波器的设计原则：

- 1) 品质因数 Q 的选择（对有源二阶）
 - (1) $Q=0.707$, 构成巴特沃思滤波器（常用）
 - (2) $Q=0.577$, 构成贝塞尔滤波器
 - (3) $Q=1$, 切比雪夫滤波器

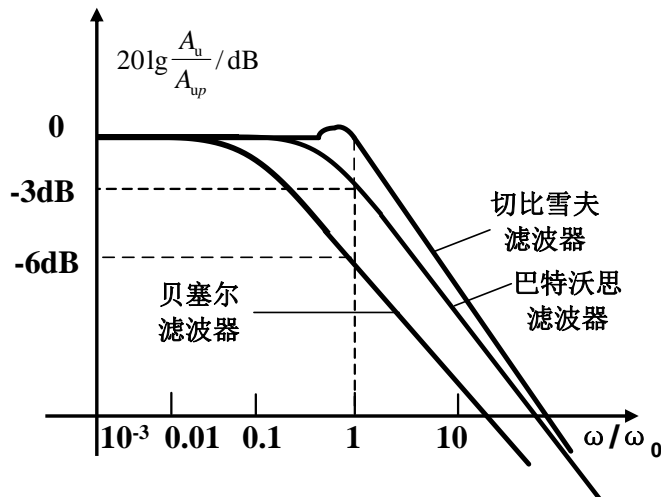


图 4.3.2 二阶有源滤波器频率特性

4.3.1.3 关键芯片及各单元电路介绍

1. 运算放大器 OPA2365

OPA2365 与普通放大器的不同在于它是轨对轨，高性能的。CMOS 运算放大器是非常适合低电压和单电源供电的应用。。下面介绍一下 OPA2365 的主要特性：

- 增益带宽：50MHz
- 轨对轨输入/输出
- 低噪声：12nV/ $\sqrt{\text{Hz}}$
- 输入失调电流：0.2pA
- 转换速率：25V/us
- 快速建立时间：0.3us 到达 0.01%
- 工作电压范围：2.2V 到 5.5V

(1) 二阶有源低通滤波器

本次实验设计采用压控电压源型设计二阶有源低通滤波器，压控电压型 (VCVS) ——同

相输入，输入阻抗很高，输出阻抗很低，如下图所示电路为常用二阶低通。由于 C14 接到集成运放的输出端，形成正反馈，使电压放大倍数在一定程度上受输出电压控制，且输出电压近似为恒压源，所以称之为二阶压控电压源低通滤波器。其优点是电路性能稳定、增益容易调节。

本实验采用的是 $Q=0.7$ 的伯特瓦兹低通滤波器，伯特瓦兹低通滤波器的特点

1)、伯特瓦兹低通滤波器在通频带内具有最大的平坦度，阶数越高，平坦度越好。在截止频率处，所有的伯特瓦兹低通滤波器都有 -3dB 的增益衰减。

2)、伯特瓦兹低通滤波器的阶数越高，在通频带内愈平坦，且对高频噪声的抑制能力也越强。

滤波器阶数不同对性能有着影响，下图为二阶有限增益的低通滤波器的原理图。

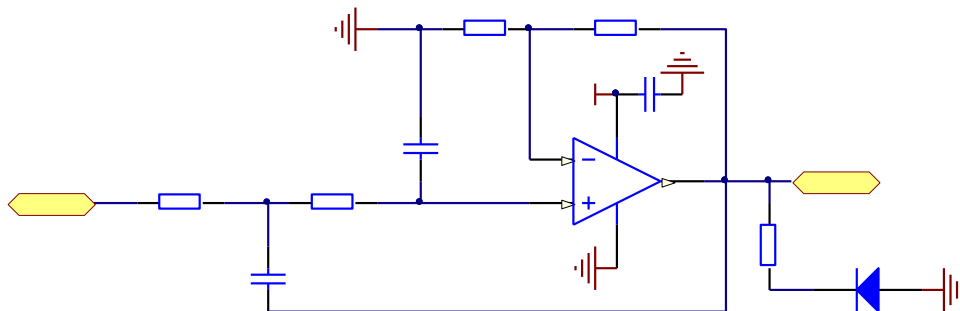


图 4.3.3 二阶有源低通滤波器连接电路图

一般的，电路中通常取： $C_8=C_{14}=C$ ， $R_{17}=R_{18}=R$

幅频特性：

$$\begin{cases} -\frac{1}{R}\dot{V}_i + \left(\frac{2}{R} + CS\right)\dot{V}_1 - SCV_o - \frac{1}{R}\frac{\dot{V}_o}{H_o} = 0 \\ -\frac{1}{R}\dot{V}_1 + \left(\frac{1}{R} + CS\right)\frac{\dot{V}_o}{H_o} = 0 \end{cases}$$

传递函数为：

$$H(S) = \frac{V_o(S)}{V_i(S)} = \frac{H_o}{1 + (3 - H_o)RCS + (RCS)^2}$$

增益为：

$$H_o = \frac{R_{15} + R_{16}}{R_{15}}$$

滤波器的低通截止频率为：

$$\omega_0 = \frac{1}{RC}$$

说明：

1)、这种二阶低通滤波器中，放大倍数 H_0 不能任意指定，当 $3 - H_o \leq 0$ 滤波器电路不稳定。

2)、电路中元件离散性少，电路参数调整方便。不过由于电路中通过引进了正反馈，所

以整个电路的增益大小受到一定的限制。

(二) 二阶有源带通滤波器

本次实验设计无限增益多路反馈型 (MFB) 设计二阶有源低通滤波器, MFB——运放为反相输入。其优点是电路有倒相作用, 使用元件较少, 但增益调节对其性能参数会有影响, 故应用范围比 VCVS 电路要小。

图 4.3.4 所示为一个无限增益多路反馈带通滤波器电路, 电路中一般去 $C_1=C_2=C$, 传递函数为:

$$H(s) = \frac{K_p Bs}{s^2 + Bs + \omega_0^2}$$

其中各系数为: $B = \frac{2}{R_3 C}$, $\omega_0^2 = \frac{1}{R_3 C^2} \left(\frac{1}{R_1} + \frac{1}{R_2} \right)$, $K_p = -\frac{R_3}{2R_1}$

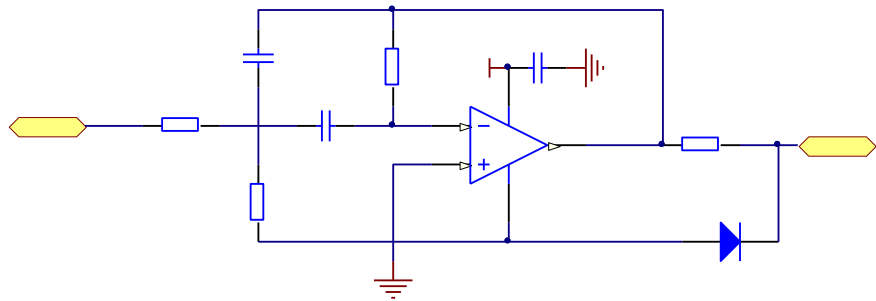


图 4.3.4 二阶有源带通滤波器连接电路图

表征带通滤波器性质的重要参数有三个, 分别是:

$\omega_0 = \frac{1}{\sqrt{RC}}$: 中心频率, 也即谐振频率, 带通滤波器在中心频率处转移函数的幅值最大;

$\beta = \omega_2 - \omega_1 = \frac{R}{L}$: 带宽, 定义为两个截止频率之差; 截止频率 ω_c 的定义为: 转移函数

的幅值由最大值下降为最大值的 $1/\sqrt{2}$ 时的频率, 即 $|H(j\omega_c)| = \frac{1}{\sqrt{2}} H_{\max}$ 。

$Q = \frac{\omega_0}{\beta} = \frac{\omega_0 L}{R} = \frac{1}{\omega_0 CR}$: 品质因数, 定义为与带宽之比。

比较器 TLC372:

TLC372 是采用 LinCMOSE 技术制造的, 它由两个独立的电压比较器组成, 每个比较器由单一的单电源供电, 从 2 V 至 18 V 的双电源供电也是可以的。每个比较器具有极高的输入阻抗 (通常大于 10^{12} 欧姆), 可直接接口与高阻抗信号源。输出是 N 沟道开漏型的。TLC372 内部有静电放电 (ESD) 保护电路。下面介绍一下 TLC372 的主要特性:

- ◆ 单或双电源供电
- ◆ 宽电源电压范围: 2 V 至 18 V
- ◆ 非常低的电源电流消耗, 在典型值 5V 时只有 150mA 的消耗
- ◆ 内置 ESD 保护
- ◆ 极低的输入偏置电流, 典型值 5 pA
- ◆ 超稳定的低输入失调电压

- ◆ 共模输入电压范围包括地
- ◆ 输出兼容 TTL, MOS, CMOS 电平

单一比较器内部原理图如图 4. XX 所示

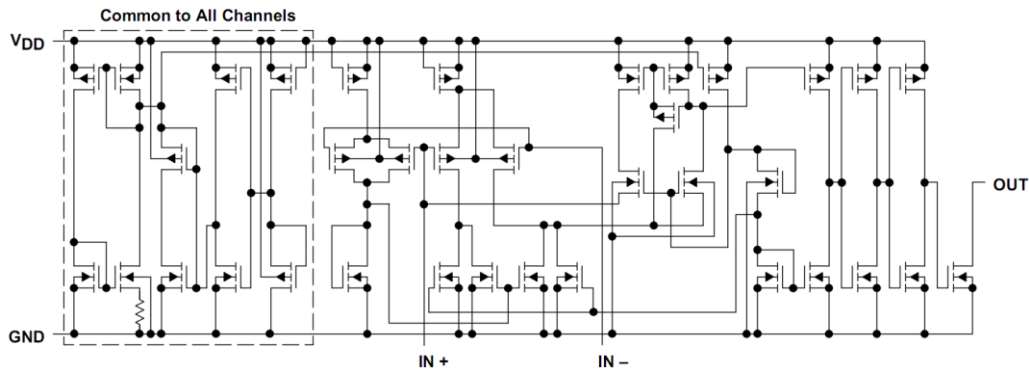


图 4.3.5 比较器内部等效原理图

从原理图可以看出比较器是集电极输出，所以在其输出端必须接上拉电阻，否则得不出输出信号。长期以来，比较器的应用一直受到运算放大器的冲击，直到目前随着比较器性能指标的不断改进，这一现状才得到改善，比较器的两路输入为模拟信号，输出则为二进制信号，当输入电压的差值增大或减小时，其输出保持恒定。因此，也可以将其当作一个 1 位模/数转换器（ADC）。运算放大器在不加负反馈时从原理上讲可以用作比较器，但由于运算放大器的开环增益非常高，它只能处理输入差分电压非常小的信号。而且，一般情况下，运算放大器的延迟时间较长，无法满足实际需求。比较器经过调节可以提供极小的时间延迟，但其频响特性会受到一定限制。为避免输出振荡，许多比较器还带有内部滞回电路。因此，比较器不能当作运算放大器使用。这也是运算放大器的应用范围相对比较广泛的主要原理之一。电压比较器(以下简称比较器)是一种常用的集成电路。比较器与运放的差别是运放可以做比较器电路，但性能较好的比较器比通用运放的开环增益更高，输入失调电压更小，共模输入电压范围更大，压摆率较高(使比较器响应速度更快)。另外，比较器的输出级常用集电极开路结构，如图 4. XX 所示，它外部需要接一个上拉电阻或者直接驱动不同电源电压的负载，应用上更加灵活。但也有些比较器为互补输出，无需上拉电阻。它可用于报警器电路、自动控制电路、测量技术，也可用于 V/F 变换电路、A/D 变换电路、高速采样电路、电源电压监测电路、振荡器及压控振荡器电路、过零检测电路等。本文主要介绍其基本概念、工作原理及典型工作电路，并介绍一些常用的电压比较器。比较器的功能是比较两个电压的大小。常用的幅度比较电路有电压幅度比较器、窗口比较器和具有滞回特性的施密特触发器。这些比较器的阈值是固定的，有的只有一个阈值，有的具有两个阈值。比较器的基本特点为：工作在开环或正反馈状态。开关特性，因开环增益很大，比较器的输出只有高电平和低电平两个稳定状态。非线性，因大幅度工作，输出和输入不成线性关系。

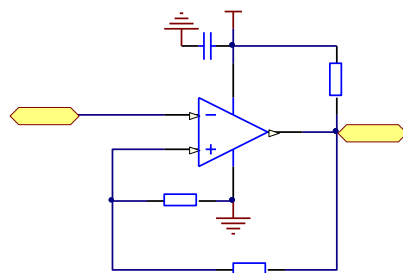


图 4.3.6 波形整形电路

4.3.1.4 实验功能模块信号引脚与 LaunchPad 系统连接

LaunchPad 开发板将 I/O 口及电源引脚引出来，方便与其它模块结合使用，其与程控放大模块的连接关系如图 4. XX 所示

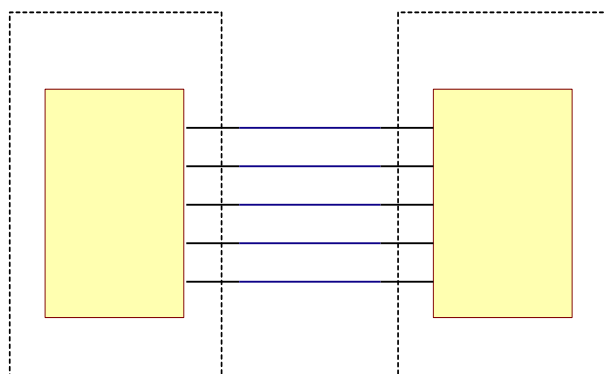


图 4.3.7 LaunchPad 开发板与模拟滤波器模块连接图

滤波器的频率信号经过整形后分别输入到 MSP430G2553 单片机的 P2.0、P2.1 两管脚，为了需要与 MSP430G2553 单片机的 3.3V 和 5V 电源连接；程控放大器模块通过引出两排 13 针的排针与 LaunchPad 开发板结合在一块，该模块的实物图及 PCB 图可参见图 4.3.8。

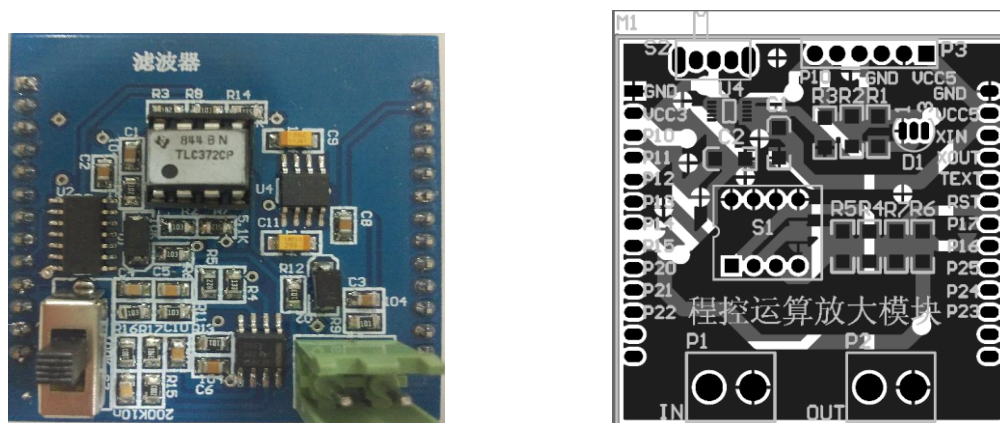


图 4.3.8 模拟滤波器模块实物图及 PCB 图

4.3.2 程控放大器实验功能模块

4.3.2.1 实验功能模块功能说明

信号放大是信号调理技术的重要组成部分，应用十分广泛。例如，在信号采集过程中，输入信号通常是微弱信号，就要求信号调理电路具有放大功能。在很多情况下，需要根据输入信号的变化情况对放大器的增益做相应的改变，这就需要采用程控放大器电路来进行处理。程控放大器实验功能模块在 MSP430G2553 单片机支持下，可实现对输入信号的增益进行选择与控制。该模块的核心芯片是 TI 公司的程控放大器 PGA112，通过 MSP430G2553 与 PGA112 之间的 SPI 接口控制程控放大器的增益。

该实验功能模块通过与 MSP430G2553 单片机的 SPI 接口及 I/O 口的连接，可以完成 I/O 口输入输出、SPI 接口输出、A/D 转换和点阵液晶显示器控制等基础性实验，并在此基础上

实现对输入信号的增益选择与控制的功能性实验，还可以结合按键功能模块及点阵液晶显示器实现对输入信号、输出信号及放大倍数的选择、控制与显示等较为复杂的应用实验。

4.3.2.2 实验功能模块组成及工作原理

图 4.3.9 是该程控放大器模块的整体组成框图：

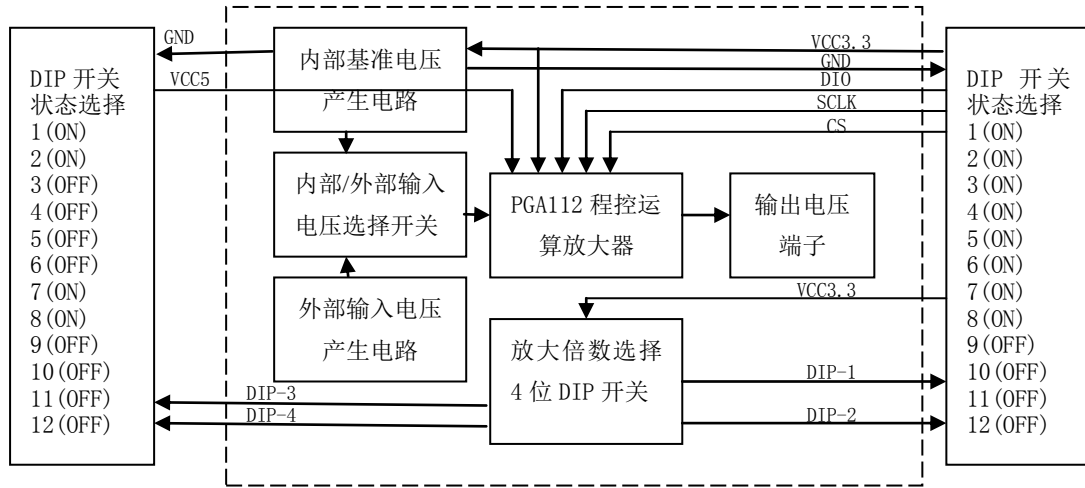


图 4.3.9 程控放大器功能模块组成框图

该实验功能模块的输入信号包括内部输入和外部输入，通过 S2 拨动开关选择输入信号，当 S2 拨动开关打在右方时选择内部输入状态，反之选择外部输入状态。内部输入信号由 LM385 芯片将 3.3V 电源电压稳压到 2.5V，然后经过电阻网络分压得到十几毫伏的直流电压信号，如图 4.XX 所示。外部输入信号由外部信号源提供，与 P1 端子连接（可选择一定频率的小信号）。程控放大器的增益选择通过单片机 MSP430G2553 与 PGA112 进行 SPI 通信控制，所选择的放大倍数由四位 DIP 拨码开关的状态确定，即通过读取四位拨码开关的不同状态，控制不同的增益。PGA112 的输出信号接到 MSP430G2553 单片机的 I/O 口 P1.0，进行 A/D 转换和采样，另一方面 PGA112 的输出信号通过 P2 端子引出来，可与示波器结合观察放大前后的信号大小和波形。

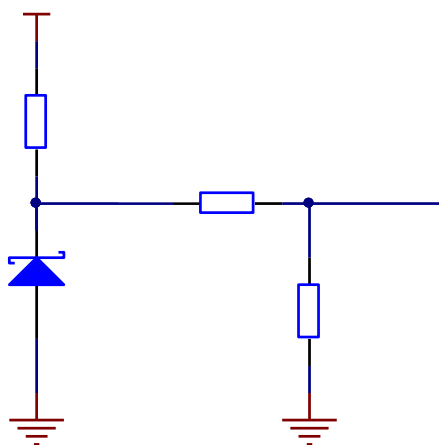


图 4.3.10 内部输入信号产生连接图

综上所述，该模块的工作原理可简单概括为：将输入信号接到 PGA112 的信号输入引脚，PGA112 通过与单片机 MSP430G2553 进行的 SPI 通信来改变其增益，单片机通过读取四位 DIP 拨码开关的状态，向程控放大器 PGA112 发送不同的 SPI 命令控制其增益变化，并将 PGA112 输出的信号接到单片机进行 A/D 转换和采样，为在点阵 LCD 显示器上显示相关参数提供必要

的数据。

4.3.2.3 关键芯片及各单元电路介绍

1. 程控运算放大器 PGA112

程控运算放大器与普通放大器的不同在于反馈电阻网络可变且受控于控制接口给出的控制信号。不同的控制信号，将产生不同的反馈系数，从而改变放大器的增益。

图 4.3.11 是 PGA112 芯片内部结构及与单片机的典型连接图

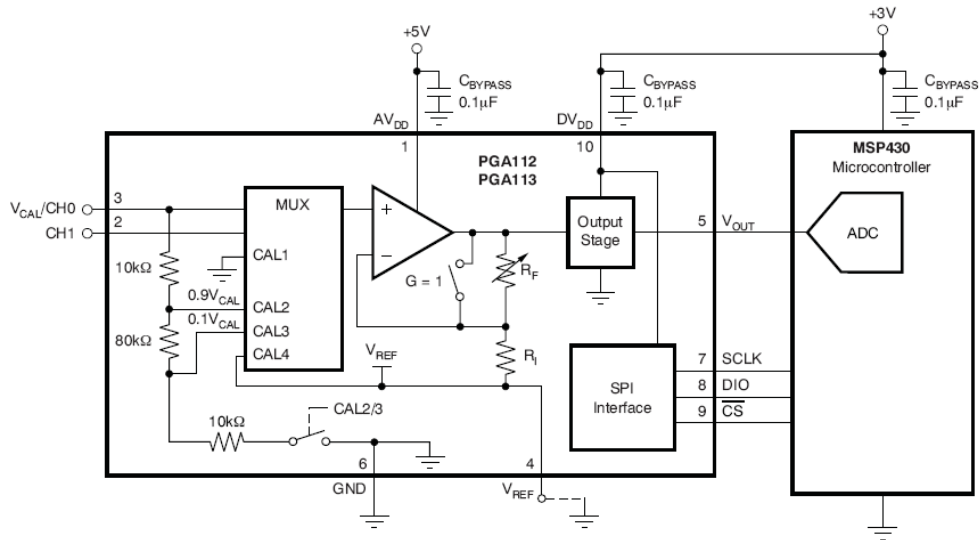


图 4.3.11 PGA112 芯片内部结构及与单片机的典型连接图

由图 4.3.11 可知，PGA112 芯片内部结构由校准部分、运放部分及 SPI 接口部分组成。当系统校准时，为了使用四个内部校准通道，VCAL/CHO 连接到低阻抗的外部参考电压，这四个内部通道分别连接到地、0.9V VCAL、0.1V VCAL、及 VREF。

PGA112 的 SPI 接口包括时钟线、数据线及片选线，分别与单片机对应的 SPI 口连接，完成对输入信号的增益控制。

对于运放部分，由其内部结构图可知，当 VREF 接地时，该程控放大器的增益为 $(1+R_F/R_I)$ ，改变 PGA112 的增益，实际上是改变了 R_F 的值。

PGA112 是具有单端输入、单电源供电、可编程增益特征的放大器，它可以通过标准 SPI 接口实现多路通道的选择和增益的选择，其中二进制增益包括 1、2、4、8、16、32、64、128 八种选择。下面介绍一下 PGA112 的主要特性：

- ◆ 轨对轨输入/输出
- ◆ 偏移：25uV(典型值)、100uV(最大值)
- ◆ 零漂移：0.35mV/° C(典型值)、1.2mV/° C(最大值)
- ◆ 低噪声：12nV/√Hz
- ◆ 输入失调电流：±5nA 最大 (+25° C)
- ◆ 增益误差：0.1% 最大($G \leq 32$)，0.3% 最大($G > 32$)
- ◆ 二进制增益：1, 2, 4, 8, 16, 32, 64, 128
- ◆ 增益开关时间：200ns
- ◆ 双通道选择
- ◆ 四个内部校准通道
- ◆ 输出摆幅：50mV 至供电轨
- ◆ AVDD 和 DVDD 混合电压系统
- ◆ $I_0 = 1.1\text{mA}$ (典型值)

- ◆ 温度范围：-40° C 到+125° C
- ◆ 标准 SPI 接口(10MHz)

PGA112 具有 10 个引脚，其引脚功能如表 4. xx 所示

表 4. 1 PGA112 引脚介绍

PIN#	名称	引脚功能描述
1	AVDD	模拟电源电压 (+2. 2V 到+5. 5V)
2	CH1	输入通道 1
3	VCAL/CH0	输入通道 0 及 VCAL 输入，可用于系统校准
4	VREF	参考输入引脚
5	VOUT	模拟电压输出
6	GND	接地引脚
7	SCLK	时钟输入 SPI 串行接口
8	DIO	数据输入/输出 SPI 串行接口
9	*CS	片选线 SPI 串行接口
10	DVDD	数字和运算放大器输出级的供电电压 (+2. 2V 多电源系统，以+5. 5V)

程控运算放大器 PGA112 顾名思义由程序控制其输入信号的放大倍数，具体来说是通过与单片机的 SPI 接口进行通信，单片机通过发送不同的指令来选择不同的增益。PGA112 具有标准 SPI 接口，通过 SCLK、DIO、*CS 与单片机 MSP430G2553 进行 SPI 通信。图 4. XX 是 PGA112 连接电路图

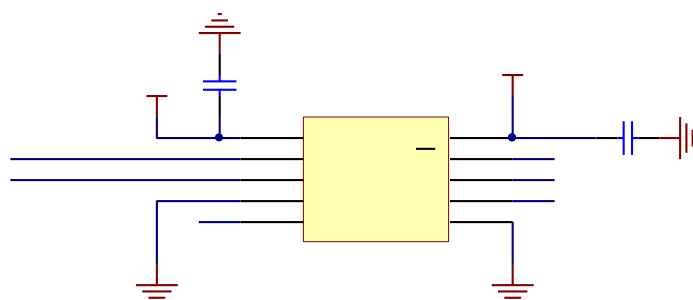


图 4. 3. 12 PGA112 连接电路图

VIN1 和 VIN0 是 PGA112 的两个不同的模拟输入通道，可通过不同的命令选择当前输入通道（该程控放大器模块默认选择通道 1）。PGA112 的 SPI 接口与单片机的 P1. 2 (SIMO)、P1. 4 (UCA0CLK)、P1. 5 (普通 I/O) 连接，通过 SPI 通信实现对 PGA112 的增益控制。下面介绍一下 SPI 通信：

SPI 是高速同步串行口，具有 3~4 线接口，收发独立，可同步进行。它通过主从方式工作，这种模式通常有一个主设备和一个或多个从设备，需要至少 4 根线，分别是 SDI（数据输入），SDO（数据输出），SCK（时钟），CS（片选），事实上 3 根也可以（单向传输时）。PGA112 工作于从模式，它只接收单片机发来的指令即可，属于单向传输，所以只需 3 根线，单片机 MSP430G2553 工作于主模式，通过配置单片机各寄存器完成 SPI 的初始化，在配置 SPI 接口时钟时注意一定要弄清楚从设备的时钟要求，因为主设备这边的时钟极性和相位都是以从设备为基准的，因此在时钟极性的配置上一定要明确从设备是在时钟的上升沿还是下降沿接收数据。另外要注意的是，由于主设备的 SDO 连接从设备的 SDI，从设备的 SDO 连接主设备的 SDI，从设备 SDI 接收的数据是主设备的 SDO 发送过来的，主设备 SDI 接收的数据是从设备 SDO 发送过来的，所以主设备这边 SPI 时钟极性的配置（即 SDO 的配置）跟从设备

的 SDI 接收数据的极性是相反的，跟从设备 SDO 发送数据的极性是相同的。时钟极性和相位配置正确后，数据才能够被正确的发送和接收，因此应该对照从设备 SPI 接口时序来正确配置主设备的时钟，该程控放大器模块的 SPI 寄存器配置可参考 PGA112 数据手册的 SPI 时序图。

单片机通过 SPI 通信发送不同的指令控制 PGA112 的增益，表 4. xx 给出了 PGA112 的 SPI 指令集

表 4.2 PGA112 SPI 指令集

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	三线 SPI 指令
0	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	READ
0	0	1	0	1	0	1	0	G3	G2	G1	G0	CH3	CH2	CH1	CH0	WRITE
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	NOP WRITE
1	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	SDN_DIS WRITE
1	1	1	0	0	0	0	1	1	1	1	1	0	0	0	1	SDN_EN WRITE

注：（1）SDN=关断模式，通过发出 SDN_EN 输入命令关断模式。由 SDN_DIS 命令或任何有效的写命令使关断模式被清除，返回到最后的有效写配置。

（2）POR (Power-on-Reset) 后的内部增益/频道选择寄存器均是 0s; 此时设定增益为 1, 通道选择为 VCAL/CH0。

由于 PGA112 通过接收单片机发送的 WRITE 指令控制其增益，而增益选择位由 G3、G2、G1、G0 决定，表 4.3 给出增益选择位

表 4.3 PGA112 增益选择位

G3	G2	G1	G0	二进制增益
0	0	0	0	1
0	0	0	1	2
0	0	1	0	4
0	0	1	1	8
0	1	0	0	16
0	1	0	1	32
0	1	1	0	64
0	1	1	1	128

因为该程控放大器模块默认选择输入通道 1，由 PGA112 数据手册知 CH3、CH2、CH1、CH0 为固定值 0001，结合表 4. xx 和表 4. xx 知，PGA112 的八种二进制增益 1、2、4、8、16、32、64、128 分别对应指令 {0x2A, 0x01}、{0x2A, 0x11}、{0x2A, 0x21}、{0x2A, 0x31}、{0x2A, 0x41}、{0x2A, 0x51}、{0x2A, 0x61}、{0x2A, 0x71}，所以 SPI 通信时发送上述八种指令之一即可选择 PGA112 不同的放大倍数。

2. 四位 DIP 拨码开关

该程控放大器模块上的四位 DIP 拨码开关是 PGA112 八种放大倍数的选择开关，即四位 DIP 拨码开关拨向不同的位置时，MSP430G2553 单片机通过读取 DIP 拨码开关的状态向 PGA112 发送相应指令进而控制 PGA112 的增益。图 4.3.13 是四位 DIP 拨码开关连接电路图。

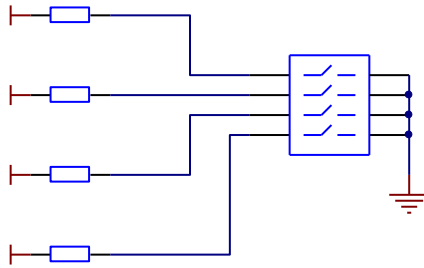


图 4.3.13 四位拨码开关连接电路图

由图 4.XX 可知，四位 DIP 拨码开关与 MSP430G2553 单片机的普通 I/O 口 P1.1、P1.3、P1.6、P1.7 连接，且为低电平有效，通过读取 I/O 口的高低电平即可判断四位 DIP 拨码开关的状态。该模块中，当 DIP 拨码开关的第 4 脚有效时，软件才扫描其它三个脚，且 3、2、1 脚构成的八种组合控制 PDA112 的八种增益，四位拨码开关的状态与放大倍数的选择关系如表 4.4 所示。

表 4.4 拨码开关状态与放大倍数选择的关系表

放大倍数	DIP-4 状态	DIP-3 状态	DIP-2 状态	DIP-1 状态
1	ON	OFF	OFF	OFF
2	ON	OFF	OFF	ON
4	ON	OFF	ON	OFF
8	ON	OFF	ON	ON
16	ON	ON	OFF	OFF
32	ON	ON	OFF	ON
64	ON	ON	ON	OFF
128	ON	ON	ON	ON

4.3.2.4 实验功能模块信号引脚与 LaunchPad 系统连接

LaunchPad 开发板将 I/O 口及电源引脚引出来，方便与其它模块结合使用，其与程控放大模块的连接关系如图 4.XX 所示

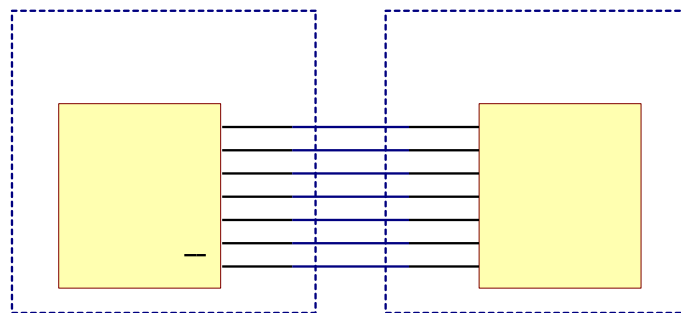


图 4.3.14 LaunchPad 开发板与程控放大模块连接图

PGA112 与单片机进行 SPI 通信需要与 MSP430G2553 单片机的 P1.2、P1.4、P1.5 连接，为了测量 PGA112 放大后的模拟电压值，则需要将 PGA112 的输出脚与 MSP430G2553 单片机的 P1.0 口连接，对其进行 A/D 转换和采样。另外给 PGA112 芯片供电，需要与 MSP430G2553 单片机的 3.3V 和 5V 电源连接；DIP 拨码开关需要与单片机 MSP430G2553 的 P1.1、P1.3、P1.6、P1.7 口连接。

程控放大器模块通过引出两排 13 针的排针与 LaunchPad 开发板结合在一块，该模块的实物图及 PCB 图和电路原理图可参见图 4.3.15。

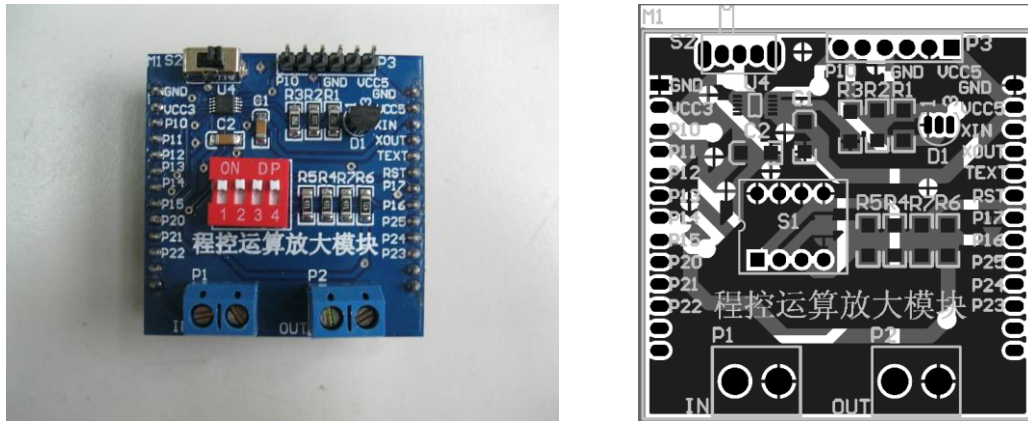


图 4.3.15 程控放大器模块实物图及 PCB 图

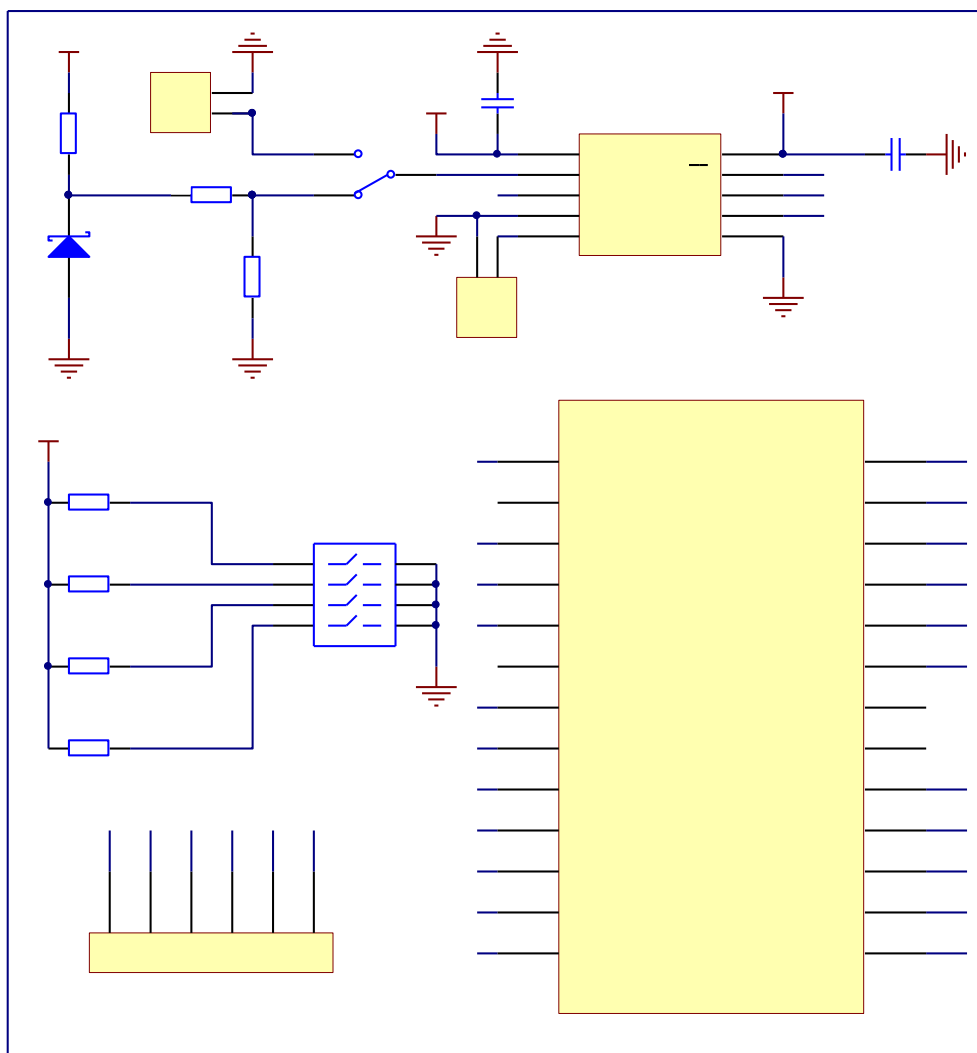


图 4.3.16 程控放大器模块电路原理图

4.3.3 晶体管特性测试模块

4.3.3.1 实验功能模块功能说明

晶体管又称“半导体三极管”或“晶体三极管”。它是由半导体锗或硅的单晶上制备两

个能相互影响的 PN 结，组成一个 PNP（或 NPN）结构。中间的 N 区（或 P 区）叫基区，两边的区域叫发射区和集电区，这三部分各有一条电极引线，分别叫基极 B（base）、发射极 E（emitter）和集电极 C（collector）。晶体管是能起到放大、振荡或开关等作用的半导体电子器件。由于晶体管广泛应用于信号的调理放大电路中，且管型多种多样，故在此设计了晶体管特性测试模块。该模块的实验功能是在 MSP430G2553 单片机的支持下完成的。可实现判断管型（是 NPN 还是 PNP 型）、判断引脚（辨别 e、b、c）和计算放大倍数 β 的功能。该模块由三个 74LS125 三态门组成，这样利于编程对晶体管进行测试。

该实验模块通过与 MSP430G2553 单片机的 I/O 口、ADC 口连接，可以完成 I/O 口输入、ADC 转换、FLASH 存储、比较器比较等基础性实验，并在此基础上实现对目标晶体管特性的测量实验，并通过液晶显示屏向用户显示所需的实验结果。

4.3.3.2 实验功能模块组成及工作原理

图 4.3.17 是该晶体管特性测试模块的整体组成框图。

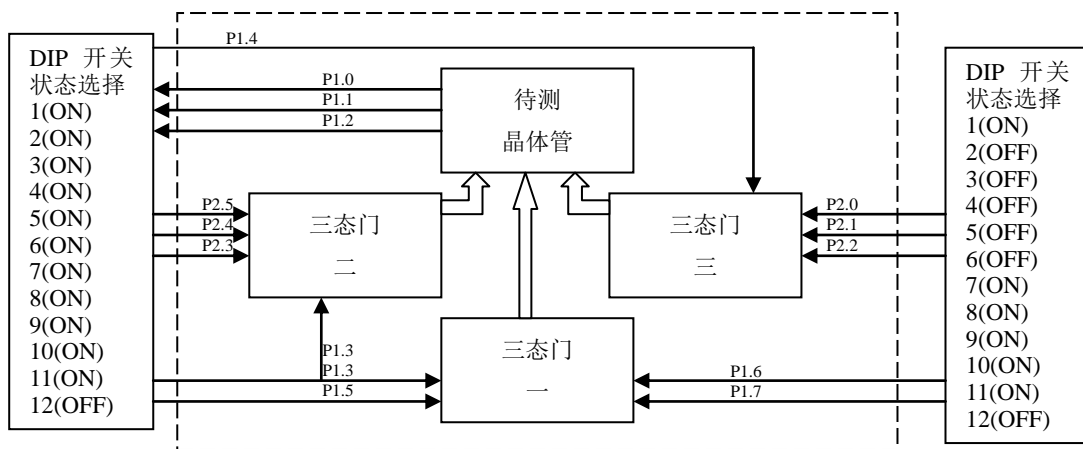


图 4.3.17 晶体管特性测试模块组成框图

该实验功能模块的输入信号包括外部输入 P1.3 和 P1.4 口，其功能是控制三态门的输入引脚的电平状态，其中 P1.5 至 P1.7 口分别用来控制三态门一上的三个控制端，P2.0 至 P2.5 分别用来控制三态门二和三态门三上的六个控制端。该模块的输出信号包括 A0 至 A2 三个 ADC 端口，分别测算待测晶体管的三个引脚电压。MSP430G2553 芯片的 FLASH 模块和比较器模块联合作用，将测得的数据进行存储、比较和计算，得到实验所需的结果并显示在液晶屏幕上。

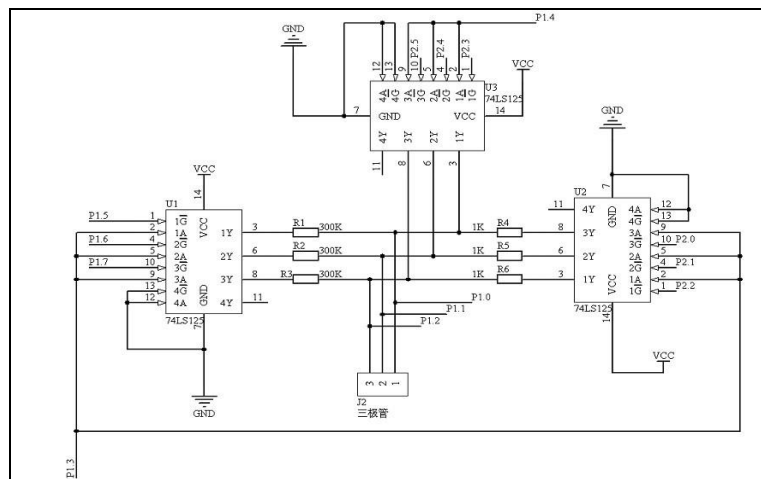


图 4.3.18 晶体管特性测试的电路原理图

用上述单元电路并配置 PNP 型晶体管测量后总结得到的数据如下表:

表 4.6 晶体管管型测量结果

引脚 1	引脚 2	引脚 3	结论
H	H	L*	管型为 PNP 型 引脚 1 为 B、2 为 E、3 为 C
H	L*	H	
H*	H	H	

引脚 1	引脚 2	引脚 3	结论
H	H	H*	管型为 NPN 型 引脚 1 为 E、2 为 B、 3 为 C
H	L*	H	
H*	H	H	

注: 星号标出的为 ADC 测量所得到的电压值, 未加星号的是通过 I/O 赋给晶体管的电压值。

H——高电平

L——低电平

得出晶体管的管型后就可以用该晶体管组成最简单的基本放大电路, 在放大电路中引入 300K 和 1K 电阻的意义在于将电压的压降与电阻阻值相比得到电流值, 通过集电极的电流值与基极的电流值相比就可以得到该晶体管的放大倍数, 同时也可以确定集电极 (C) 和射极 (E) 的位置。

若测得的晶体管是 PNP 型晶体管, 且求得基极 B 的位置, 则需要按照图 4. xx 进行连接。

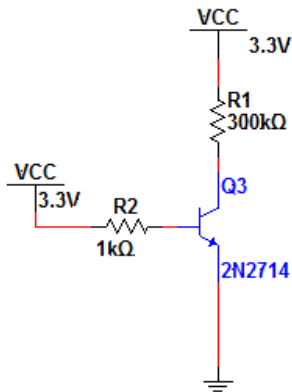


图 4. 3. 21 PNP 晶体管的基本放大电路

通过对三态门一、二、三进行编程对控制线置高或者置低, 将目标晶体管按照图 4. xx 连接, 用三个 ADC 口测量晶体管的引脚电压, 根据公式:

$$I_B = (3.3V - V_B) / 1K \Omega$$

$$I_C = (3.3V - V_C) / 300K \Omega$$

$$\beta = I_C / I_B$$

就可以求出晶体管的放大倍数 β 。

若测得的晶体管是 NPN 型晶体管，且知道基极 B 的位置，则需要按照图 4. xx 进行连接。

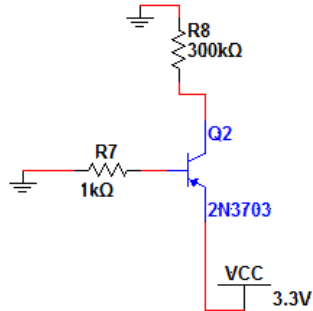


图 4. 3. 22 NPN 型晶体管的基本放大电路

通过对三态门一、二、三进行编程对控制线置高或者置低，将目标晶体管按照图 4. xx 连接，用三个 ADC 口测量晶体管的引脚电压，根据公式：

$$I_B = (V_B - 0V) / 1K \Omega$$

$$I_C = (V_C - 0V) / 300K \Omega$$

$$\beta = I_C / I_B$$

就可以求出晶体管的放大倍数 β 。

4. 3. 3. 4 实验功能模块信号引脚与 LaunchPad 系统连接

LaunchPad 开发板将 I/O 口及电源引脚引出来，方便与其它模块结合使用，其与程控放大模块的连接关系如图 4. 3. 23 所示。

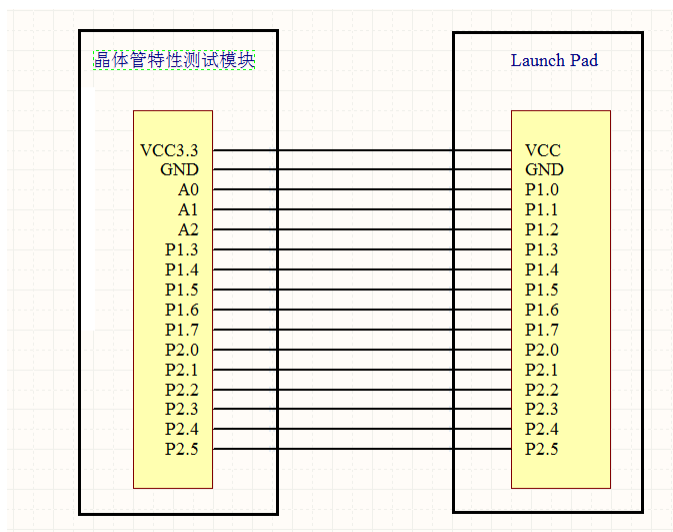


图 4. 3. 23 LaunchPad 开发板与晶体管特性测试模块连接图

为了测量晶体管引脚的电压，将晶体管与 MSP430G2553 单片机的 P1. 0、P1. 1、P1. 2 连

接，利用这几个接口的 ADC 功能。为了控制三态门的输入电压值，则需要将三态门的输入端与 MSP430G2553 单片机的 P1.3 和 P1.4 接口连接，实现单片机对三态门的输入控制。其次，为了控制三态门的控制端选通与中断，需要将三态门的控制端分别连接到 MSP430G2553 的 P1.5 至 P1.7 和 P2.0 至 P2.5。另外给 74LS125 芯片供电，需要与 MSP430G2553 单片机的 3.3V 电源连接。

晶体管特性测试模块通过引出两排 13 针的排针与 LaunchPad 开发板结合在一块，该模块的实物图及 PCB 图和电路原理图可参见图 4.3.24。

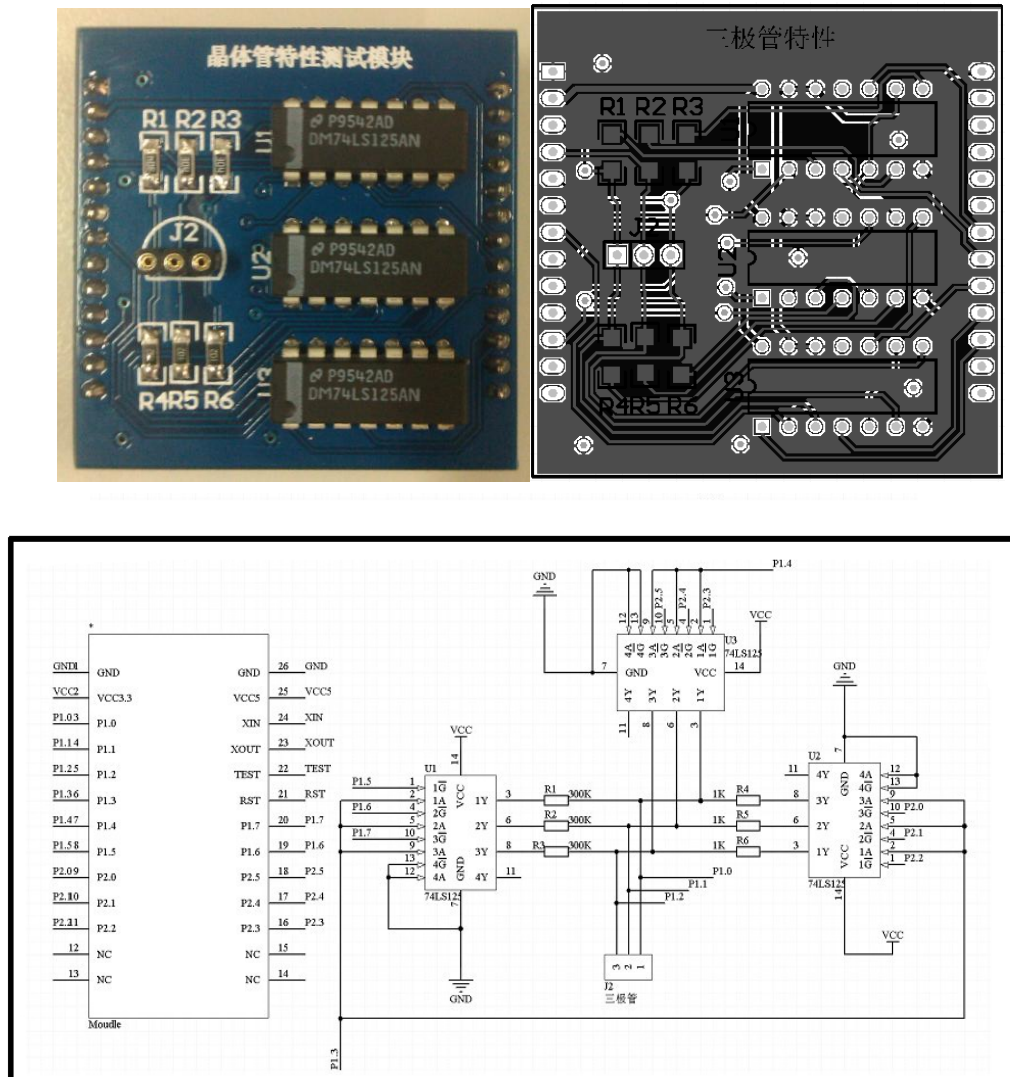


图 4.3.24 晶体管特性测试模块实物图、PCB 图和电路原理图

4.3.4 光照度检测实验功能模块

4.3.4.1 实验功能模块功能说明

在日常生活中，我们与光息息相关，可能偶尔一次光线的变化就直接影响我们的心情。例如：在平时居住的房间内，一般要求窗户的面积不小于地板面积的八分之一，这是为了房间内有较好的的采光条件，让人们觉得没有压抑感；又如我们看书时，光线较差会直接伤害我们的眼睛，如此等等，总之，我们的生活离不开光。光照强度的检测也显得尤为重要，本

模块即为基于这个理念来设计的。

该功能模块通过光电池加上一定的信号调理电路，最后与单片机 ADC 模块相连，可以完成所在对环境实时光强的检测，同时，将检测结果通过点阵式 LCD12864 显示出来。通过该模块的实验，学生可以学习本模块调理电路的设计方法、单片机 ADC 模块的应用以及 LCD12864 的使用方法。

4.3.4.2 实验功能模块组成及工作原理

图 4.3.25 是该光照度检测模块的整体组成框图：

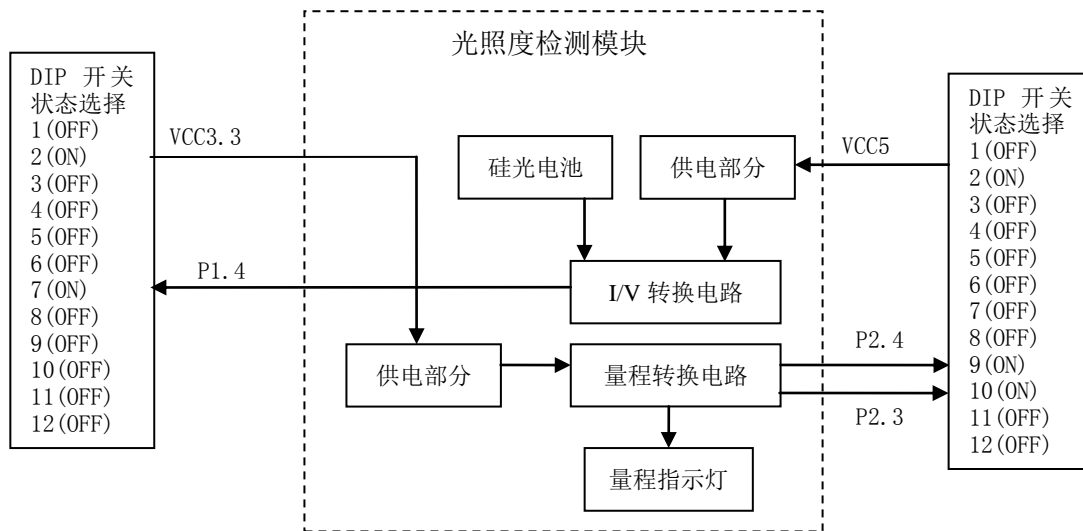


图 4.3.25 光照度检测模块组成框图

该实验功能模块的主体部分为信号调理电路即 I/V 转换电路，硅光电池输出部分要么是开路电压，要么是短路电流，本模块使用的是短路电流（具体见 4.3.6.3 小节），通过电流电压转换器，将短路电流信号变成电压信号，然后将此调理后的信号输入到单片机 ADC 模块，经过单片机的处理，将所得到的结果显示在 LCD12864 上。由于光照度量程范围较宽，故本模块采取两个量程切换的方法，通过用户手动切换量程来显示不同强度下的光照强度。

4.3.4.3 关键元器件及各单元电路介绍

1. 硅光电池

光电池简介：

要想测得环境中光照强度的值，必须要有一定的感光元件，硅光电池就是其中用的比较多的一种传感器。

像平时我们见到的各种干电池一样，光电池也会产生一定的电压和电流，只不过普通的干电池不需要外界条件的参与，而光电池需要由光照射才会产生电。光电池是一种无源元件，即不需要外界供电就能工作，有源元件必须外界供电才能工作。光电池是一种在光的照射下产生电动势的半导体元元件。光电池的种类很多，常用有硒光电池、硅光电池和硫化铊、硫化银光电池等。主要用于仪表，自动化遥测和遥控方面。有的光电池可以直接把太阳能转变为电能，这种光电池又叫太阳能电池，图 4.3.26 列举了几种太阳能电池。

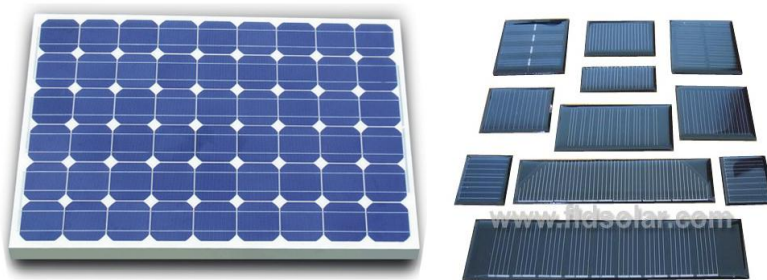


图 4.3.26 太阳能电池

光电池是一种特殊的半导体二极管，能将可见光转化为直流电。有的光电池还可以将红外光和紫外光转化为直流电。本次设计所使用到的光电池是型号为 SP-10A 的两脚直插式硅光电池，其外形如图 4.3.27 所示。

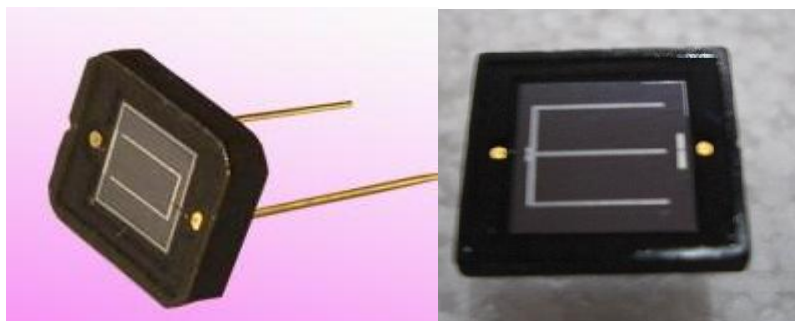


图 4.3.27 SP-10A 硅光电池

在图 4.3.27 所示的硅光电池中，顶部“E”型的方块区域即为受光区，改变这部分的光通量即会改变光电池的电动势。一般情况下“E”型右侧为正极，左侧为负极。读者可用万用表自行检测，通过不同程度的遮挡受光区，观察光电池的输出电压有何变化。

硅光电池的工作原理：

硅光电池是一种利用光生伏特效应制成的光电转换器件，通过将光信号转变为电信号来检测待测量。光电池工作原理，当光照射P—N结时，原子受激发而产生电子—空穴对，由于电子和空穴分别向两极移动而产生电动势，两极接入电路就能产生电流了。

硅光电池是一种直接把光能转换成电能的半导体器件。它的结构很简单，核心部分是一个大面积的PN结如图4.3.28所示。

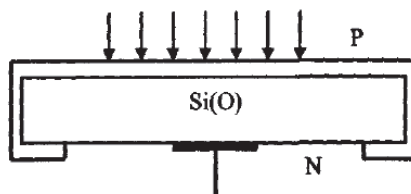


图4.3.28 硅光电池结构示意图

硅光电池响应时间短 ($10^{-10} \sim 10^{-6}$ s)，光电池转换效率高(目前转换效率高达27.5%的硅光电池已经研制成功)。若有 1m^2 的这种光电池，在足够的阳光照射下，可以产生100多瓦的电能。硅光电池主要有两个方面的应用，即作为电源和作为光电检测器件的应用。硅光电池作为测量元件使用时，应当作电流源，不宜作电压源。为什么呢？请读者带着这个疑惑继续

往下看。

硅光电池的基本特性：

硅光电池的基本特性包括：伏安特性、光照特性和光谱特性等。在本次设计中，主要使用了硅光电池的光照特性，因此在本小节主要介绍本特性，其他特性请读者自行查看硅光电池的数据手册。

光照特性指在不同光照强度下，硅光电池的短路电流和开路电压的大小。具体如图 4.3.29 所示。

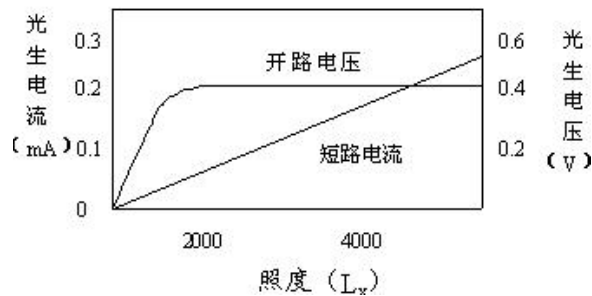
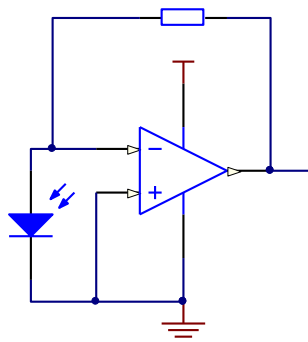


图4.3.29 硅光电池的光照特性曲线

在图4.3.29所示的光照特性曲线中，可以清楚的看到，当照度在2000Lx之前，硅光电池的开路电压随着光照度的增加迅速上升，而照度在2000Lx之后，开路电压基本上趋于平坦，即达到了饱和状态（这是由硅光电池的内部特性决定的），但是其短路电流与照度的关系却成正比关系，这也就是在硅光电池工作原理部分最后提到硅光电池作为测量元件使用时，应当作电流源，不宜作电压源的原因。

2. 光电转换与信号调理电路

一个线性度好、稳定度高的光电转换与信号放大电路对于整个测试系统是至关重要的。它直接影响整个系统的测量精度，灵敏度、稳定性及系统的测试速度等指标。前面已经多次提到硅光电池作为测量元件使用时，应当作电流源，但是单片机能够识别的是电压信号，因此我们需要将硅光电池的输出电流转换成电压，这就需要使用 I/V 转换器（即电流/电压转换器）。其电路如图 4.3.30 所示。



4.3.30 I/V 转换电路

在图4.3.30中，硅光电池电流的方向从上到下，由运放的特性“虚断”可得到该电流全部流经电阻R2，因此在R2上会产生一个压降，又由运放特性“虚短”可得到，运放的2端和3端电势相同，3端与地连接，故这两端电势均为零，从而1端输出的电势就等于R2上的压降，

注意电流的流动方向是从R2的右端流向左端，因此1端的输出电压为正电压。

根据硅光电池的光照特性曲线可知，硅光电池的短路电路与光照度成正比，带任何负载后形成的电流都不再是短路电流，也就谈不上与光照度成正比这个关系，下面来探讨为何图5所示的电压所使用的电流就能看作是短路电流。

还是从运放特性出发，运放“虚短”特性表明，2端和3端电压相同，即硅光电池负载电阻 $R_L=0$ ，相当于硅光电池工作时处于短路状态，因此硅光电池的电流就能看作短路电流；由于运放的输入电阻很大，因此可看成该短路 i 电流全部流经 R_2 ，所以运放输出端所得到的电压 V_o 为：

$$V_o=i \times R_2$$

从上式可知，运放输出电压 V_o 是硅光电池的短路电流放大 R_2 倍的结果，另外此电压也与光照度成正比。

根据以上推导过程可以得到：光照度与I/V转换后输出的电压成正比关系，因此光照度的计算公式可由下式表示。

$$L_x=V_o \times x$$

上式中 L_x 为索要显示的光照度， x 为比例系数，具体为多少请读者参考第五章第2节A/D转换基础实践中本模块软件设计部分。

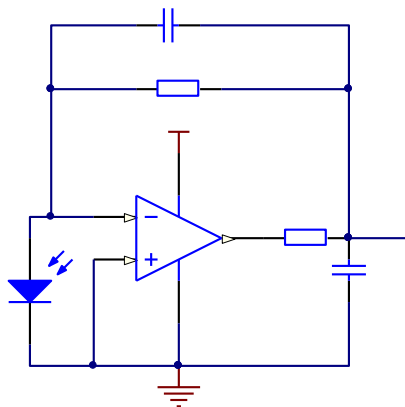


图 4.3.31 改进后的 I/V 转换电路

图 4.3.31 是改进后的 I/V 转换电路，也就是光电转换与信号前置放大电路。改进后的光电检测电路在放大器的输出和检测电路的输出之间加一个 RC 滤波电路，这样就限制了放大器输出信号的带宽，滤掉经过放大的噪声和放大器本身的噪声。该电路具有较高的信噪比；电阻 R_3 采用了温度系数小的精密电阻，从而保证了电流/电压转换放大电路的稳定性。

3. 运放选型

在仪器仪表的设计过程中，精度是设计时要考虑的最主要因素之一，运放作为最主要的元器件必须认真选择。

由于影响微电流测量灵敏度的首要因素是运放偏置电流，其次是噪声电压和零点漂移要实现微电流测量，运算放大器的输入须满足：输入阻抗 \gg 反馈电阻；偏置电流 $<$ 被测电流；失调电压及漂移小；增益与共模抑制比高；噪声小。

在本次设计中，运放选择TI公司的LM358，其具有以下优点：

- ◆ 内部频率补偿
- ◆ 低输入偏流
- ◆ 低输入失调电压和失调电流
- ◆ 共模输入电压范围宽，包括接地
- ◆ 差模输入电压范围宽，等于电源电压范围
- ◆ 直流电压增益高(约 100dB)
- ◆ 单位增益频带宽(约 1MHz)
- ◆ 电源电压范围宽：单电源(3—30V)
- ◆ 双电源(± 1.5 — ± 15 V)
- ◆ 低功耗电流，适合于电池供电 . 输出电压摆幅大(0 至 $V_{cc}-1.5$ V)

通过 Multisim 仿真，该芯片能够较好的工作，综合考虑多种因素，最终确定使用TI公司的LM358。

4. 多量程设计

在不同强度的光线照射下，硅光电池的短路电流范围较大，大概从0.1 μ A到10mA，从图5可以得知，假设R2为10K Ω ，I/V变换后输出电压Vo的范围为从1mV到100V，这显然是不可能的，因为运放OPA2356的工作电压不可能达到100V，因此，I/V变换后输出电压Vo的范围最多只能是从1mA到VCC（理想情况下）。所以，当光线较强势，运放输出饱和，所测得的光照度也就不是真实的光照度，这显然不是我们想要的结果。反之，假设R2为100 Ω ，I/V变换后输出电压Vo的范围为从10 μ V到1V，对于单片机来说10 μ V太小了。那怎么样才能做到即使大电流也不至于使运放输出饱和、小电流不至于使转换后的电压过小呢？这就是在本次设计加入多量程设计的原因所在。

本次多量程设计的原理是：当光线较弱时，硅光电池产生的短路电流较小，适当取较大的反馈电阻R2；当光线较强时，硅光电池产生的短路电流较大，适当取较小的反馈电阻R2。总之，要使经I/V转换后的电压在合适的范围内，以便于单片机进行处理。综合考虑成本、结构、实用等多方面因素，最终本次设计选用两个量程，一大一小供用户选择。具体电路原理如图4.3.32所示。

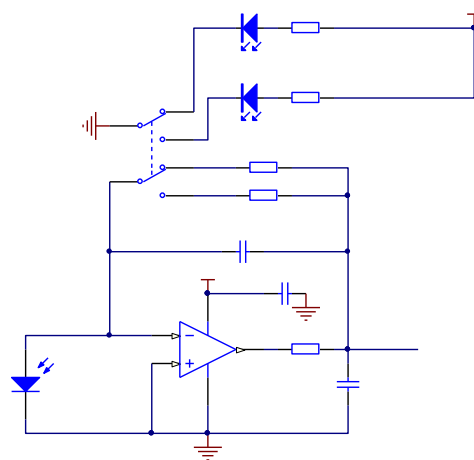


图 4.3.32 多量程设计的 I/V 转换器

图4.3.32中，SW DPDT是双刀双掷开关，即用于切换量程用；LED-red和LED-green是量

程指示灯，用户可通过指示灯的亮灭清楚的看到当前选用哪个量程；R4、R5是两个指示灯的限流电阻，用于保护指示灯；R2、R6即为I/V转换的反馈电阻，其精度越高越好，具体阻值可按测试环境的不同来选取。看到这里，可能有些仔细的人会有疑惑：为什么加入两个指示灯就使用双刀双掷开关呢？用单片机IO来控制指示灯的亮灭不是更简单吗？其中自有有一定的原因，在这里，笔者先不作介绍，读者可在第五章第2节A/D转换基础实践中本模块软件设计部分找到这个问题的答案。

4.3.4.4 实验功能模块信号引脚与 LaunchPad 系统连接

图4.3.33给出了光照度检测模块的外接引脚，在本模块中只用到了VCC3.3、VCC5、P1.4、P2.3、P2.4五个引脚。其中VCC3.3为量程转换指示灯供电，VCC5为LM358供电，P1.4是单片机ADC转换模块的输入通道4，与调理好之后的信号相连，用于AD转换，P2.3和P2.4是量程切换的信号端，用于“告诉”单片机当前用户具体选择哪个量程，以便采取不同的处理方式。

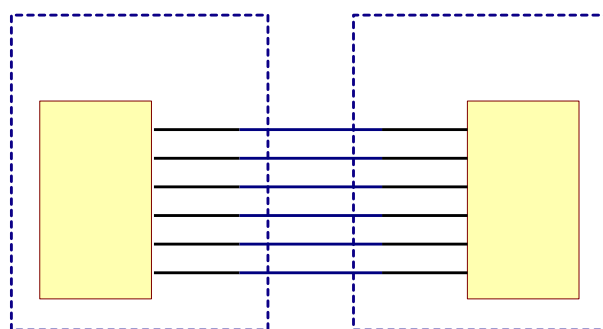


图 4.3.33 光照度检测模块的外接引脚

图4.3.33封装成两排13排针，对应实验底板的另两排13排针的排座，无需额外的连线即可实现实验过程中的即插即用，非常方便。该模块的电路原理图如图4.3.34所示，最终实物图及PCB可分别参考图4.3.35和图4.3.36。

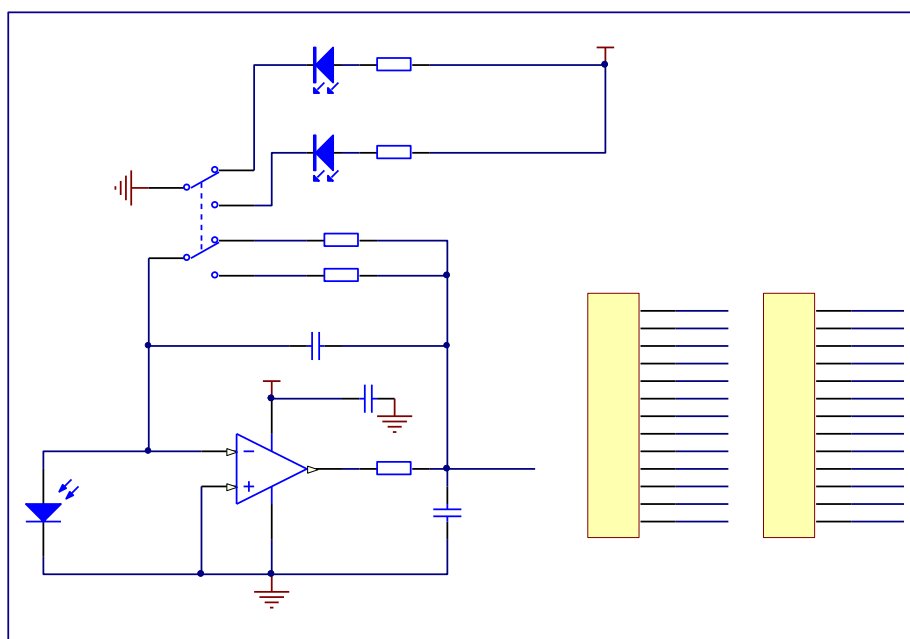


图4.3.34 光照度检测模块原理图



图 4.3.35 光照度检测模块实物

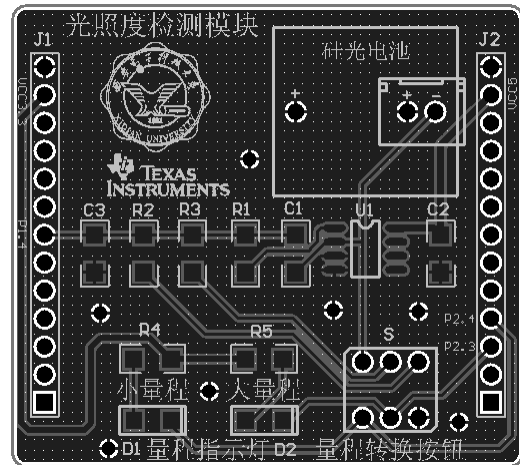


图 4.3.36 光照度检测模块 PCB

4.3.5 三种温度测量模块

4.3.5.1 实验功能模块功能说明

温度是一个重要的基本物理量，自然界中任何物理、化学过程都紧密地与温度相联系，温度与我们的生活和生产都紧密相关，因此温度测量具有重要的意义。

三种温度测量模块上有三种温度传感器，分别是 DS18B20、LM35 和 PT100。MSP430G2553 单片机可利用三种温度测量模块上的三种温度传感器和单片机的内部温度传感器实现温度的综合测量。该实验功能模块通过与 MSP430G2553 单片机的 I/O 口及 ADC10 的连接，可实现对每个温度传感器测量温度的实时监测，并可以设置与 ADC10 连接的传感器的采样次数和采样间隔，最后将测量结果加以比较，求其相对误差和绝对误差。

4.3.5.2 实验功能模块组成及工作原理

图 4.3.37 是该三种温度测量模块的整体组成系统框图：

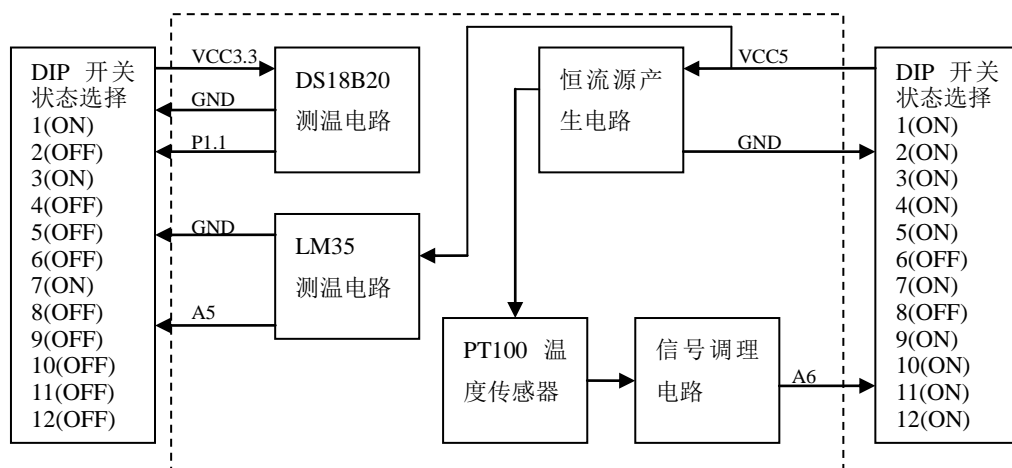


图 4.3.37 三种温度测量模块组成框图

模块上 DS18B20 温度传感器是一个数字温度传感器，它和单片机的一根 I/O 相连与单片机通信，将其测量的温度数据传送给单片机。LM35 温度传感器输出的电压值和温度成线性关系，其输出电压 $V_{out} = 10.0 \text{ mV}/^{\circ}\text{C} * T^{\circ}\text{C}$ ，其中 T 为当前温度，因此单片机 ADC10 对其输出电压进行采样，便可求的当前的温度。PT100 是一种使用非常广泛的热电阻传感器，其阻值和温度成函数关系。本模块设计了一个恒流产生电路，使 PT100 上流过一个恒定电流，这样 PT100 两端的电压值只与其阻值有关，在将 PT100 两端的电压经过信号调理，再送给单

片机 ADC10 采样, 便可计算出 PT100 的阻值, 从而求得温度值。

最后, 单片机将通过模块上的三种温度传感器测量的温度值和单片机内部温度传感器测得的温度值送给 LCD 进行显示。

4.3.2.3 关键芯片及各单元电路介绍

1. 温度传感器 DS18B20

DS18B20 是一种可编程分辨率的数字温度传感器。DS18B20 的核心功能是它的直接到数字温度传感器。温度传感器的分辨率可以配置成 9, 10, 11, 12 位, 相应的递增量分别为 0.5°C , 0.25°C , 0.125°C , 和 0.0625°C 。加电默认分辨率为 12 位。其特点如下:

- 独特的 1-Wire 接口, 只需一个端口引线来通信
- 每个器件都有一个独一无二的 64 位序列编码存储在板级 ROM 内
- 多点能力简化了分布式温度采集应用
- 无需外部器件
- 可从数据线供电, 供电电压范围从 3.0V 到 5.5V
- 测温范围从 -55°C 到 $+125^{\circ}\text{C}$ (-67°F 到 $+257^{\circ}\text{F}$)
- -10°C 到 $+85^{\circ}\text{C}$ 精确度 $\pm 0.5^{\circ}\text{C}$
- 用户可选择从 9 到 12 位测温分辨率
- 750ms(最大值)内温度转换成 12 位数位字
- 用户定义非易失性(NV)报警设置
- 报警搜索命令识别并寻址温度超出指定的极限(温度报警状态)的器件
- 可用封装有 8-PinSO(150 mils), 8-Pin μSOP , 和 3-PinTO-92
- 可应用于温度控制, 工业系统, 消费产品, 温度测量以及任何热敏感系统

DS18B20 的封装图如图 4.3.38 所示。

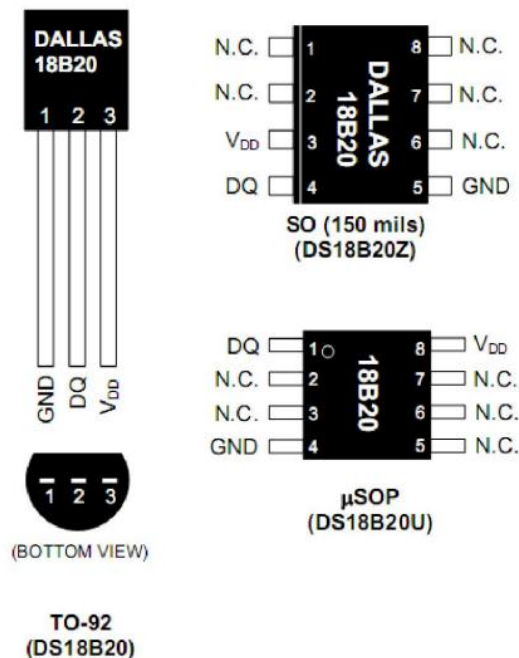


图 4.3.38 DS18B20 封装图

DS18B20 的管脚说明见表 4.6。

管脚			名称	功能
S0	μSOP	T0-92		
1, 2, 6, 7, 8	2, 3, 5, 6, 7	—	NC	未连接
3	8	3	VDD	可选的 VDD，寄生供电模式下须接地
4	4	2	DQ	数据输入输出，开漏一线接口管脚。寄生供电模式下也为器件提供电源
5	1	1	GND	接地

图 4.3.38 是 DS18B20 的内部组成框图，管脚描述在管脚描述表中给出。64 位 ROM 存储着器件独一无二的序列码。暂存器包括存储温度传感器的数字输出的 2 字节温度寄存器。另外寄存器提供对 1 字节高低警报触发寄存器 (TH 和 TL) 和 1 字节配置寄存器的读写。配置寄存器允许用户设置温度到数字转换的分辨率为 9, 10, 11, 或 12 位。TH, TL 和配置寄存器是非易失性的 (EEPROM)，所以器件掉电时数据会保持。

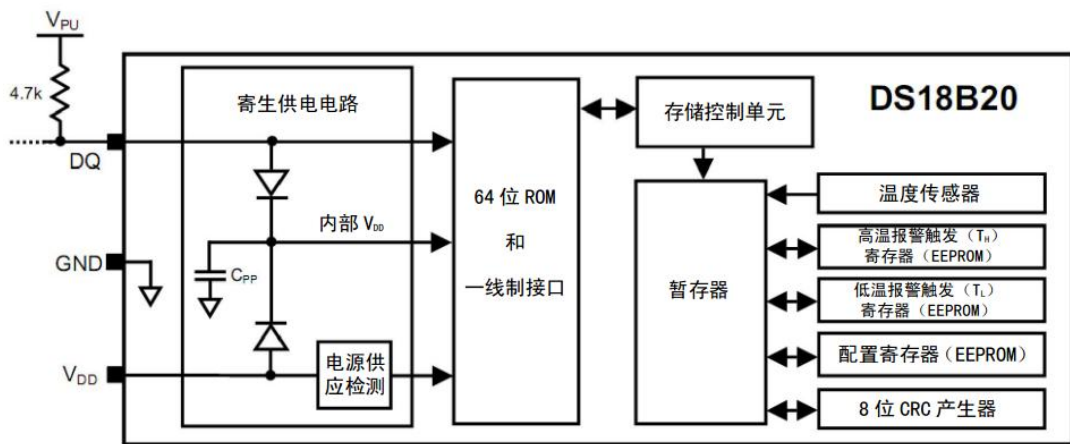


图 4.3.39 DS18B20 内部组成框图

DS18B20 仅靠数据线 DQ 一根线和单片机进行通信，其与单片机的电路连接如下图：

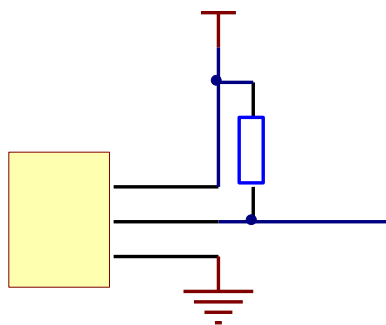


图 4.3.40 DS18B20 电路图连接

DS18B20 的数据线通过上拉电阻和单片机 P1.1 连接，可利用单片机的 I/O 口模拟 DS18B20 的读写时序，便可完成对 DS18B20 暂存器的访问，读出 DS18B20 测量的 12 位温度数据。具体程序实现详见 5.4.3 节。

2. 温度传感器 LM35

LM35 是 TI 公司推出的一款集成精密温度传感器，其输出电压正比于温度（单位℃）。其特点如下：

- 无需外部校准便能达到 $\pm 1/4^{\circ}\text{C}$ 的精度
- 输出电压为 $10\text{mV}/^{\circ}\text{C}$
- 在 25°C 时能保证 0.5°C 精度
- 测温范围 $-55^{\circ}\text{C}\sim 150^{\circ}\text{C}$
- 适合远程应用
- 成本低廉
- 供电电压范围从 4V 到 30V
- 漏电流小于 $60\mu\text{A}$
- 非线性误差只有 $\pm 1/4^{\circ}\text{C}$
- 低阻抗输出 (0.1Ω 的负载输出 1mA)

其管脚说明见下表:

名称	管脚号	功能
+VSS	1	供电电源 $4\text{V}\sim 30\text{V}$
Vout	2	输出端, 输出电压等于 $10\text{mV}/^{\circ}\text{C}$
GND	3	接地

LM35 工作电压 $4\sim 30\text{V}$, 在上述电压范围以内, 芯片从电源吸收的电流几乎是不变的 (约 $50\mu\text{A}$), 所以芯片自身几乎没有散热的问题。其输出电压和温度成良好的线性关系, 无需外部校准, $V_{\text{out}}=10.0\text{mV}/^{\circ}\text{C} * T^{\circ}\text{C}$, 其中 T 为当前温度。下图为 LM35 的电路连接图。LM35 输出端接单片机 P1.5, 即 ADC10 的通道 5。

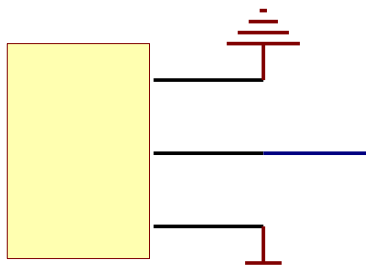


图 4.3.40 LM35 电路连接

3. 温度传感器 Pt100

Pt100 是铂热电阻, 其阻值会随着温度的变化而改变。其工作原理: 当 Pt100 在 0°C 的时候他的阻值为 100Ω , 在 100°C 时它的阻值约为 138.5Ω , Pt100 阻值会随着温度上升而成近似匀速的增长。但他们之间的关系并不是简单的正比的关系, 而更应该趋近于一条抛物线。

铂电阻的阻值随温度的变化而变化的计算公式:

$$-200 < t < 0^{\circ}\text{C} \quad R_t = R_0 [1 + At + Bt^2 + C(t-100)t] \quad (1)$$

$$0 < t < 850^{\circ}\text{C} \quad R_t = R_0 (1 + At + Bt^2) \quad (2)$$

R_t 为 $t^{\circ}\text{C}$ 时的电阻值, R_0 为 0°C 时的阻值。公式中的 A, B, C 系数为实验测定。这里给出标准的系数: $A=3.90802 \times 10^{-3}^{\circ}\text{C}$; $B=-5.802 \times 10^{-7}^{\circ}\text{C}$; $C=-4.27350 \times 10^{-12}^{\circ}\text{C}$ 。

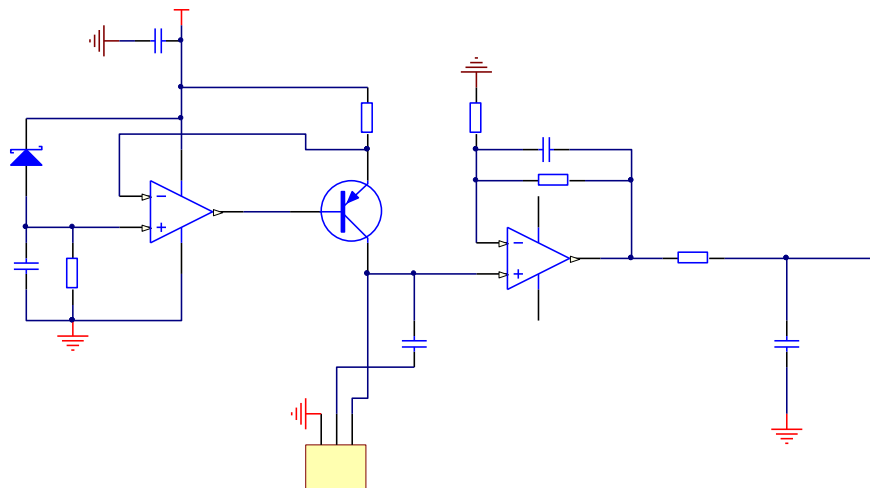


图 4.3.41 Pt100 部分电路

由上图可知本模块选用恒流源驱动 Pt100，从而使 Pt100 两端的电压变化直接反映出其电阻变化。但是根据公式 $P=I^2R$ 电流流过电阻，电阻会发热。因此流过 Pt100 上的电流不能太大，本模块上电路设计流过 Pt100 上的电流为 1mA，这样 Pt100 发热非常小，基本上不影响测温结果。

然后将 Pt100 两端电压经过放大电路接入到单片机 ADC10 的通道 6。最后，可根据采样的电压值计算出温度值。

4.3.5.4 实验功能模块信号引脚与 LaunchPad 系统连接

LaunchPad 开发板将 I/O 口及电源引脚引出来，方便与其它模块结合使用，其与三种温度测量模块的连接关系如图 4.3.42 所示。

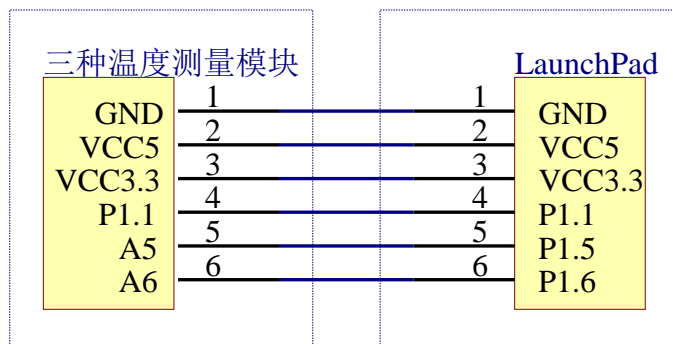


图 4.3.42 LaunchPad 开发板与三种温度测量模块连接图

模块上的三个温度传感器 DS18B20、LM35 和 Pt100 对应的三个部分电路分别接在 LaunchPad 上的 P1.1、P1.5 和 P1.6。P1.5 和 P1.6 对应 MSP430G2553 的 ADC10 的通道 5 和通道 6。

模块两端还引出两排 13 针的排阵，模块插在一体化实验板上后，排阵引脚就与 LaunchPad 开发板上的引脚对应，排阵引脚如图 4.3.43 所示。

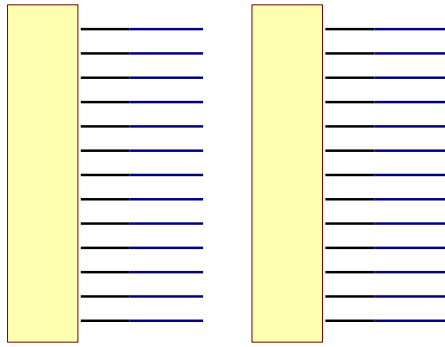


图 4.3.43 模块排阵引脚图

该模块的实物图及 PCB 图和该模块的原理图如图 4.3.44 和图 4.3.45 所示。

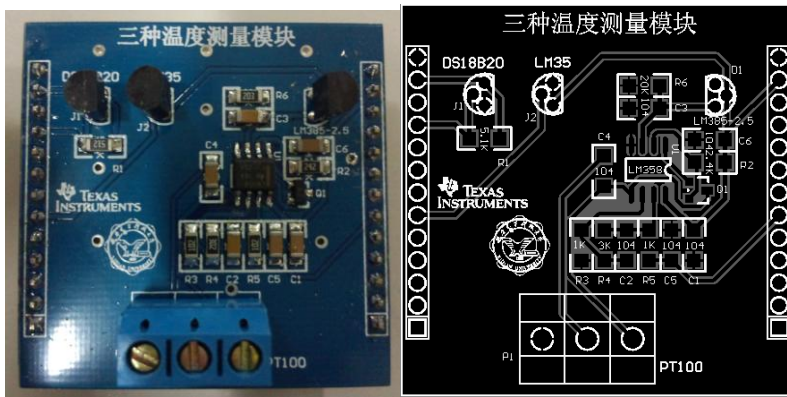


图 4.3.44 三种温度测量模块实物图及 PCB 图

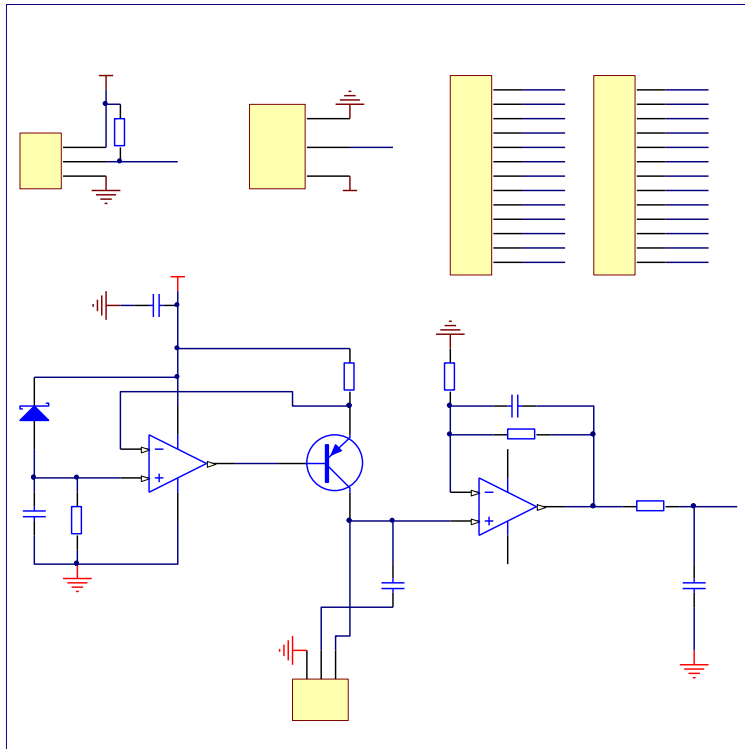


图 4.3.45 三种测温模块原理图

4.3.6 矩阵键盘及数码管实验模块

4.3.6.1 实验功能模块功能说明

该模块由 4 位数码管和 4×4 矩阵键盘组成，主要用于辅助实验板上其它模块，也可以用作简易计算器。

4.3.6.2 实验功能模块组成及工作原理

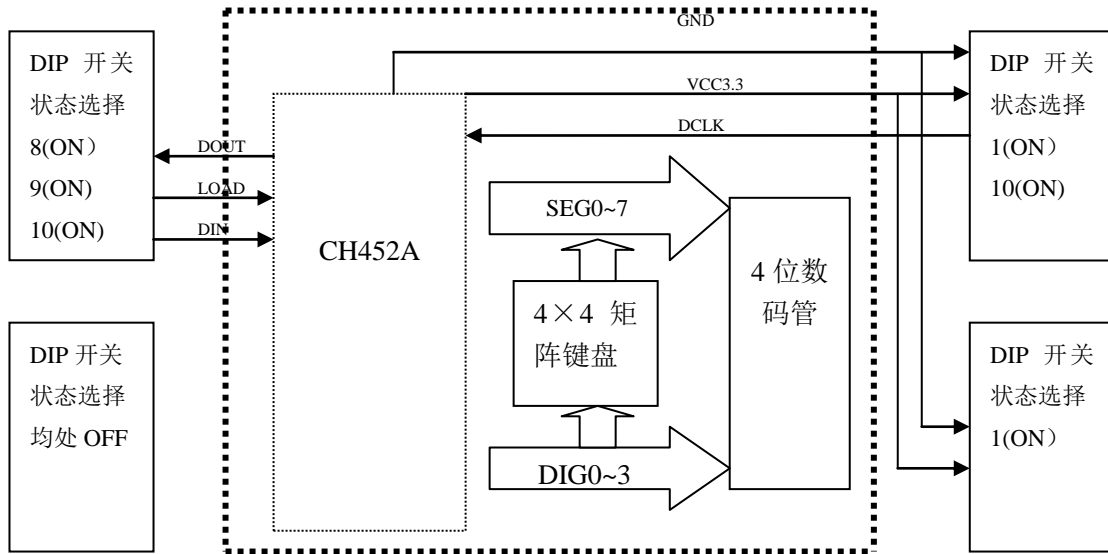


图 4.3.46 矩阵键盘及数码管功能模块组成框图

实验模块通过矩阵键盘进行输入，数码管可对输入输出进行显示。芯片 CH452A 可同时驱动矩阵键盘和数码管，由于该实验模块需用到实验底板上的两个模块，因此在上图 4. xx 中位于上面的两个“DIP 开关状态选择”同属于底板上的一个模块，而位于下面的两个“DIP 开关状态选择”同属于底板上的另外一个模块。此外，除在上图 4. xx 中标示出的位置的 DIP 开关处于拨通（ON）外，其它未标示出的位置的 DIP 开关均处于断开（OFF）状态。

4.3.6.3 关键芯片及各单元电路介绍

1. 驱动芯片 CH452A

CH452A 是数码管显示驱动和键盘扫描控制芯片。内置时钟振荡电路，可以动态驱动 8 位数码管或者 64 只 LED，具有 BCD 译码、闪烁、移位、段位寻址等功能；同时还可以进行 64 键的键盘扫描；CH452A 通过可以级联的 4 线串行接口或 2 线串行接口与单片机等交换数据，并且可以对单片机提供上电复位信号。

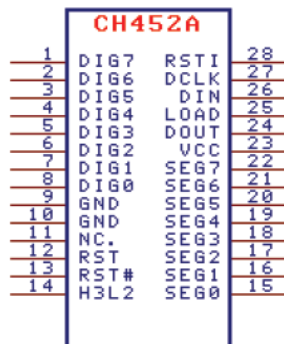


图 4.3.47 CH452A 管脚图

特点：

显示驱动

- ◆ 内置电流驱动级，段电流不小于 20mA，字电流不小于 100mA；
- ◆ 动态显示扫描控制，直接驱动 8 位数码管、64 支发光管 LED 或 64 级光柱；
- ◆ 可选数码管的段与数据位相对应的不译码方式或者 BCD 译码方式；
- ◆ BCD 译码支持一个自定义的 BCD 码，用于显示特殊字符；
- ◆ 数码管的字数据左移、右移、左循环、右循环；
- ◆ 任意段位寻址，独立控制各个 LED 或者各数码管的各个段的亮与灭；
- ◆ 扫描极限控制，支持 1 到 8 个数码管，只为有效数码管分配扫描时间；
- ◆ 可以选择字驱动输出极性，便于外部扩展驱动电压和电流。

键盘控制

- ◆ 内置 64 键盘控制器，基于 8×8 矩阵键盘扫描；
- ◆ 内置按键状态输入的下拉电阻，内置去抖动电路；
- ◆ 键盘中断，可选择低电平有效输出或者低电平脉冲输出；
- ◆ 提供按键释放标志位，可供查询按键按下或释放；
- ◆ 支持按键唤醒，处于低功耗节电状态中的 CH452 可以被部分按键唤醒。

外部接口

- ◆ 同一芯片，可选择高速的 4 线串行接口或经济的 2 线串行接口；
- ◆ 4 线串行接口：支持多个芯片级联，时钟速度从 0 到 2MHz，兼容 CH451 芯片；
- ◆ 4 线串行接口：DIN、DCLK 信号线可与其它接口电路公用，节约引脚；
- ◆ 2 线串行接口：支持两个 CH452 并联（由 ADDR 引脚电平设定各自地址）；
- ◆ 2 线串行接口：400kHz 时钟速度，兼容 I^2C 总线，节约引脚；
- ◆ 内置上电复位，可以为单片机提供高电平有效和低电平有效复位输出。

其它

- ◆ 内置时钟振荡电路，不需要外部提供时钟或者外接振荡元器件，更抗干扰；
- ◆ 支持低功耗睡眠，节约电能，可以被按键唤醒或者被命令操作唤醒；
- ◆ 支持 3V 到 5V 电源电压；
- ◆ 支持 SOP28 和 DIP24S 无铅封装。

2. 数码管电路

CH452 可以动态驱动 8 个共阴数码管，CH452 的 15 脚 SEG0 接数码管的 11 脚（a 段），CH452 的 16 脚 SEG1 接数码管的 7 脚（b 段），CH452 的 17 脚 SEG2 接数码管的 4 脚（c 段），CH452 的 18 脚 SEG3 接数码管的 2 脚（d 段），CH452 的 19 脚 SEG4 接数码管的 1 脚（e 段），CH452 的 20 脚 SEG5 接数码管的 10 脚（f 段），CH452 的 21 脚 SEG6 接数码管的 5 脚（g 段），CH452 的 22 脚 SEG7 接数码管的 3 脚（dp 段）。一定要为段驱动引脚（即 CH452 的 15~22 脚）串接电阻 $R_1 \sim R_8$ ，用以限制和均衡各个段的驱动电流。串接限流电阻的阻值越大，则驱动电流越小，数码管的显示亮度越低，其阻值一般在 60Ω 到 $1k\Omega$ 之间，在其他条件相同的情况下，应该优先选择较大的阻值。在 5V 电源电压下，串接 270Ω 电阻通常对应段电流 10mA。如果启用了 CH452 的段电流 LMTC 功能，那么限流电阻 $R_1 \sim R_8$ 可以省掉。

CH452 的 8 脚 DIG0 接数码管的 6 脚（COM4），CH452 的 7 脚 DIG1 接数码管的 8 脚（COM3），CH452 的 6 脚 DIG2 接数码管的 9 脚（COM2），CH452 的 5 脚 DIG3 接数码管的 12 脚（COM1）。由于某些数码管在较高工作电压时存在反向漏电现象，容易被 CH452 误认为是某个按键一直按下，所以应该实用二极管 $D_1 \sim D_4$ 防止反向漏电，确保按键扫描更可靠。

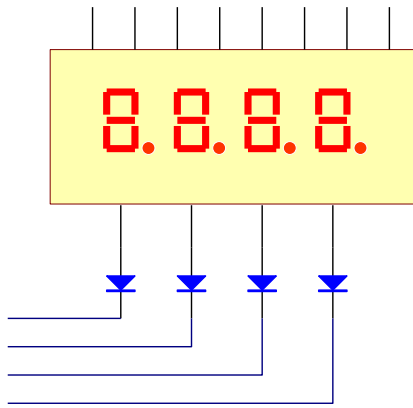


图 4.3.48 CH452A 与 4 位数码管连接原理图

3. 矩阵键盘电路

CH452 具有 64 键的键盘扫描功能，为了防止按键被按下后在 SEG 信号线和 DIG 信号线之间形成短路而影响显示，应该在 CH452 的 DIG0~DIG3 引脚与矩阵键盘之间串接限流电阻 R9~R12，其阻值可以从 1kΩ 到 5kΩ。

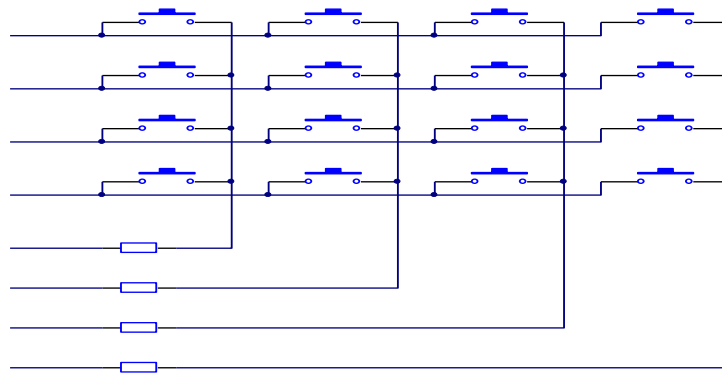


图 4.3.49 CH452A 与矩阵键盘连接原理图

4.3.6.4 实验功能模块信号引脚与 LanchPad 系统连接

LaunchPad 开发板将 I/O 口及电源引脚引出来，方便与其它模块结合使用，其与程控放大模块的连接关系如图 4.XX 所示。

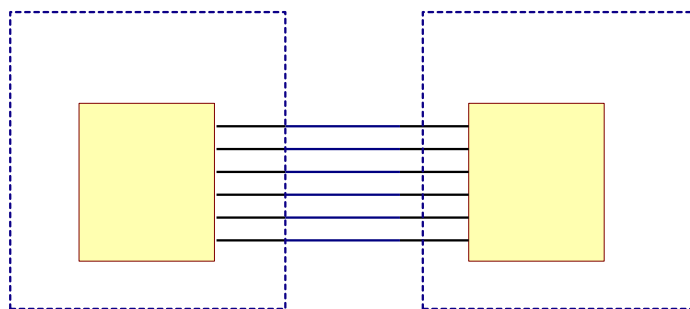


图 4.3.50 LaunchPad 开发板与矩阵键盘及数码管模块连接图

CH452 的 H3L2 脚接高电平（或者悬空）时，CH452 通过 4 线串行接口与单片机 G2433 进行通信。由于实验板主板已经加了电源退耦电容，所以该模块无需再加退耦电容，否则要加之，减少驱动大电流产生的干扰，其中 CH452 的 24 脚 DOUT 与单片机 G2334 的 P2.0 连接，CH452 的 25 脚 LOAD 与单片机 G2334 的 P2.1 连接，CH452 的 26 脚 DIN 与单片机 G2334 的 P2.2 连接，CH452 的 24 脚 DCLK 与单片机 G2334 的 P2.3 连接。在与 CH452 进行远距离连接的电

路中，建议对 DOUT、LOAD、DIND、CLK 加上上拉电阻减少干扰。该模块的实物图及 PCB 图和电路原理图可参见图 4.3.51 和图 4.3.52。

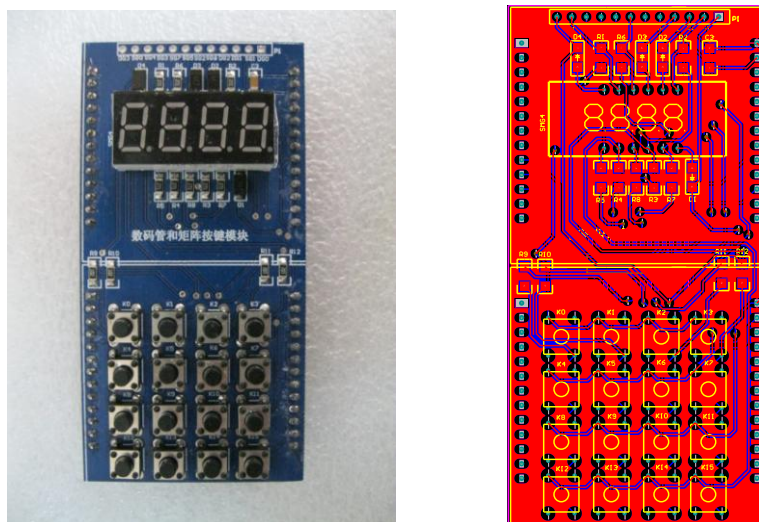


图 4.3.51 矩阵键盘及数码管模块实物图及 PCB 图

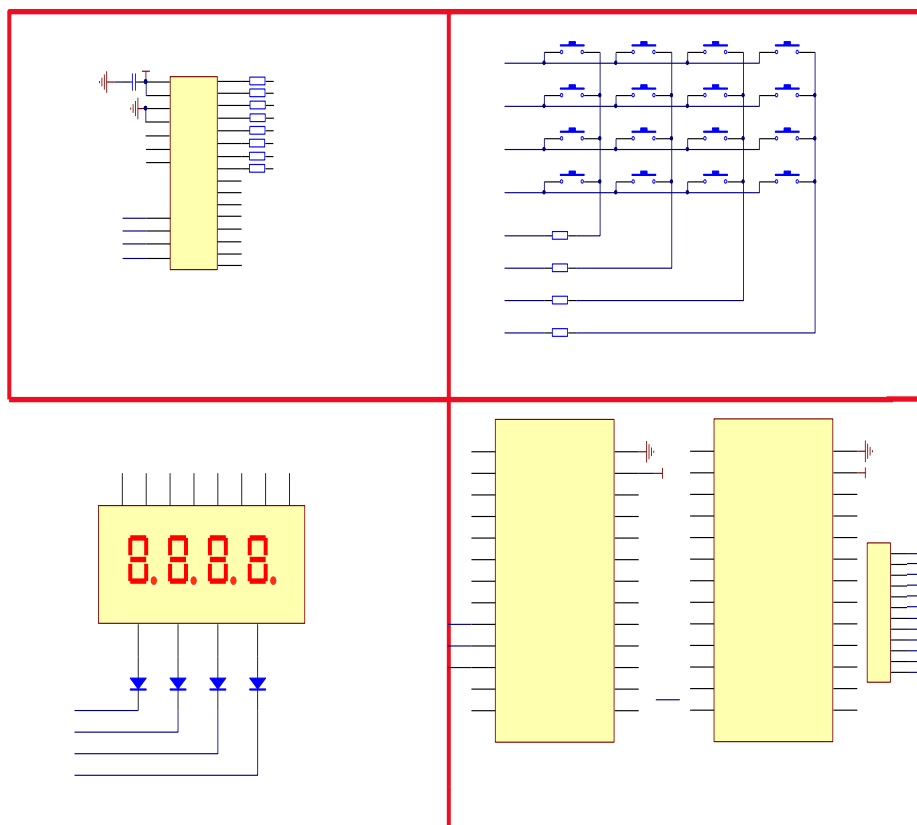


图 4.3.52 矩阵键盘及数码管电路原理图

4.3.7 触控 RGBLED 模块

4.3.7.1 实验功能模块功能说明

由于 RGBLED 灯的参数使得其在更多的显示领域得到了广泛的应用。该实验功能模块主要是使用单片机及触控按键，实现对 RGB LED 灯的控制。通过 MSP430G2553 与 RGBLED 模块

的连接，不仅可以完成简单的 I/O 口功能实验，而且可以完成 TA 模块的各种功能模式测试的实验，并且可以结合按键模块实现对 RGBLED 灯更为复杂的控制实验。

4.3.7.2 实验功能模块组成及工作原理

图 4.3.53 为该触控 RGBLED 模块的整体组成框图。

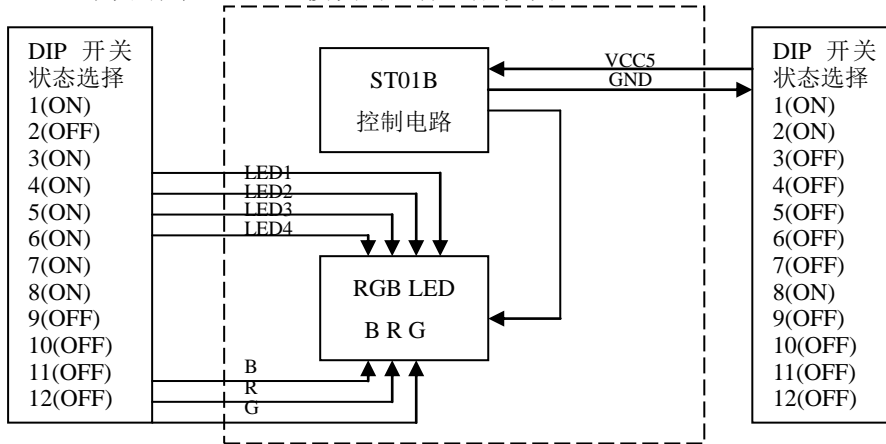


图 4.3.53 触控 RGBLED 模块组成框图

由框图可见实验模块通过两种方式来实现对 RGBLED 灯的控制：

(1) ST01B 是一个单通道带自校正功能的容性触摸感应器，它是对输入感应电容的变化经过芯片内部电路控制得到相应的输出，通过改变模式实现对 RGBLED 灯不同的控制。

(2) 通过 G2553 I/O 口控制 RGBLED 灯，即编程定时 I/O 控制或者使用 PWM 波控制 LED 灯的闪烁。

4.3.7.3 关键芯片及各单元电路介绍

1. RGB LED 5050

5050 LED 的封装尺寸：5mm×5mm×1.6mm（封装如图 4.xx），光通量可以达到 5500–6000MCD（纯白光，暖光的稍微小一点），它的工作电压和普通的 LED 一样，只需要 3.2–3.4V，电流也是一样 20MA。

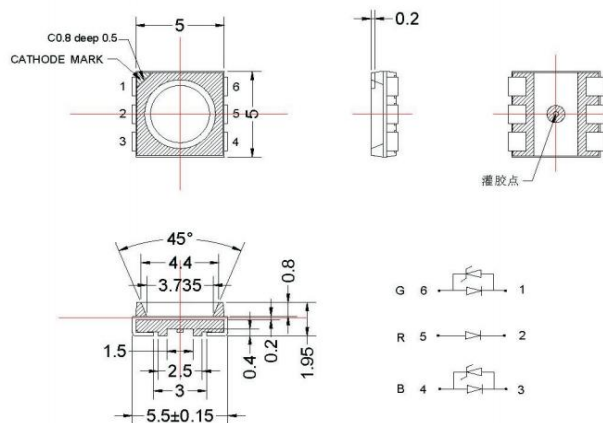


图 4.3.54 5050 封装

由于 5050 贴片 LED 的体积比较小，做工比较精细，所以手工焊接不大方便，但是不是不能焊接，其焊接温度要控制在 250 度内，否则会烫坏较贵的 5050 贴片 LED。

由于 5050 贴片 LED 参数决定了他必定是款受人喜爱的 LED，所以他常被用在高端民用节能灯，汽车仪表板，LED 灯带、手机背光及按键，仪器仪表背光及要求小体积 LED 之产品，LED 背光、开关及标志的平面背光，来电显示、闪光灯、汽车、音响、户内显示屏、照明

灯具市场。

(1) 光电特性参数:

电流越大, 电压越高, 温度越高, 电流越小, 发光角度: 125 度等; 正向使用电流是 60mA, 反向电压最大不能超过 5V; 工作温度在-35~+60°C 之间最好。如果不考虑光衰寿命问题, 也可在+85 度以下工作; 功率: 0.18W (与市面上号称 0.2W 的相同功率)。

(2) 5050LED 白光应用参数:

a、电流: 43-46mA (这是最理想的驱动电流); 最大使用电流不能超过 60mA;

b、电压: 2.8-3.6V (要求灯珠的瞬间电压不能超过 5V, 要不就对 LED 灯珠有损坏或严重损坏而无法修复);

c、环境温度: -20°C~+40°C (要求灯具的散热设计做得比较好, LED 灯引脚的温度不能超过 60 度);

d、如果电流在 14-20mA 的时候, 每降低 1mA 电流, 其亮度相应降低 4%;

e、功率在 43-46mA 的时候为 0.12W, 在 46-53mA 的时候为 0.15W, 在 54-60mA 的时候为 0.18W;

f、耗电量低

g、使用寿命长: 在恰当的电流和电压下, LED 的使用寿命可达 10 万小时;

h、高亮度、低热量: 比 HID 或白炽灯更少的热辐射;

i、环保: LED 是由无毒的材料作成, 不像荧光灯含水银会造成污染, 同时 LED 也可以回收再利用;

j、坚固耐用: LED 是被完全的封装在环氧树脂里面, 它比灯泡和荧光灯管都坚固。灯体内也没有松动的部分, 这些特点使得 LED 可以说是不易损坏的;

k、可控性强: 易于实现颜色的变化和亮暗变化, 而且反应迅速。

2. ST01B (单通道带自校正功能的容性触摸感应器)

ST01B 触摸感应器可以用平均电容值作为基准检测感应点的电容变化, 管脚如图 4. xx。它可以通过任何非导电介质来感应电容变化。这样感应模块就可以很好的跟水和灰尘隔离。ST01B 有更强的抗干扰性和更好的一致性。

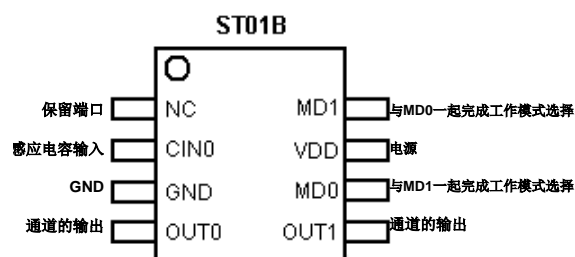


图 4.3.55 ST01B

(1) ST01B 特点

- 带自校正功能的单通道感应芯片
- 可以通过任何非导电介质感应“按键触摸”
- 通过外部电容调整灵敏度
- Open-Drain 的输出形式
- 工作电压范围: 2.5v~6.5v

(2) 额定值 (如表 4.8):

表 4.8

工作温度	存储温度	最大工作电压	管脚的容限电压	功率损耗	直流输出电流
-40°C--+85°C	-50°C--+150°C	6.5V	VDD+0.3v	80mW	10.0 mA

注意：超出上述额定值可能导致芯片工作不正常并且导致芯片的永久损坏。

(3) 工作模式

①普通按键无省电模式(MD0=VDD MD1=VDD)

时序	时段 1	时段 2	时段 3	时段 4	时段 5	时段 6
端口	芯片复位	无手指	手指触摸	无手指	手指触摸	无手指
OUT0	高阻	高阻	低电平	高阻	低电平	高阻
OUT1	低电平	低电平	高电平	低电平	高电平	低电平

②普通按键带省电模式(MD0=VDD MD1=GND)

时序	时段 1	时段 2	时段 3	时段 4	时段 5	时段 6
端口	芯片复位	无手指	手指触摸	无手指	手指触摸	无手指
OUT0	高阻	高阻	低电平	高阻	低电平	高阻
OUT1	低电平	低电平	高电平	低电平	高电平	低电平

③锁存按键无省电模式(MD0=GND MD1=VDD)

时序	时段 1	时段 2	时段 3	时段 4	时段 5	时段 6
端口	芯片复位	无手指	手指触摸	无手指	手指触摸	无手指
OUT0	低电平	低电平	高电平	高电平	低电平	低电平
OUT1	高电平	高电平	低电平	低电平	高电平	高电平

④锁存按键带省电模式(MD0=GND MD1=GND)

时序	时段 1	时段 2	时段 3	时段 4	时段 5	时段 6
端口	芯片复位	无手指	手指触摸	无手指	手指触摸	无手指
OUT0	低电平	低电平	高阻	高阻	低电平	低电平
OUT1	高电平	高电平	低电平	低电平	高电平	高电平

⑤锁存按键无省电模式(MD0=悬空 MD1=VDD)

时序	时段 1	时段 2	时段 3	时段 4	时段 5	时段 6
端口	芯片复位	无手指	手指触摸	无手指	手指触摸	无手指
OUT0	高阻	高阻	低电平	低电平	高阻	高阻
OUT1	低电平	低电平	高电平	高电平	低电平	低电平

⑥LED 调光模式(MD0=悬空 MD1=GND)

在 LED 调光模式下，OUT0 为 open-drain 结构，低电平有效（如果需要输出高电平，则外部必须加上拉电阻）。芯片复位后，OUT0 端口的 PWM 输出的预设的默认占空比为 100%，长时间按键 2 秒以上则进入调光模式，进入调光模式后每 0.5 秒左右占空比减小 6.25%，直到占空比减小到最小值 25%或手指离开。如果用户需要调高占空比，则需要先将占空比调至最小值 25%，然后再一次长时间按键超过 2 秒钟以上。OUT1 的输出是 push-pull，输出电平和 OUT0 相反。图 4.3.36 为 ST01B 的电路连接图：

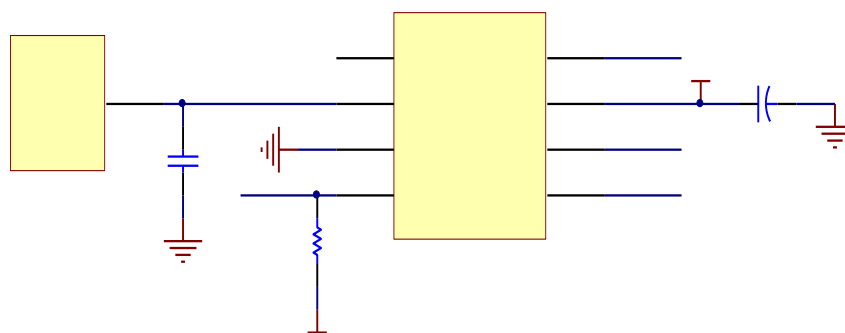


图 4.3.36 ST01B 的电路连接图

(4) 功能参数

2. R1 当 OUT0 的输出模式选择为强驱动时，上拉电阻是不必要的。
3. MD0, MD1 端口根据功能要求，直接连接到 VDD 或 GND 或悬空。

4.3.7.4 实验功能模块信号引脚与 LaunchPad 系统连接

LaunchPad 开发板将 I/O 口及电源引脚引出来，方便与其它模块结合使用，其与触控 RGBLED 模块的连接关系如图 4.3.59 所示：

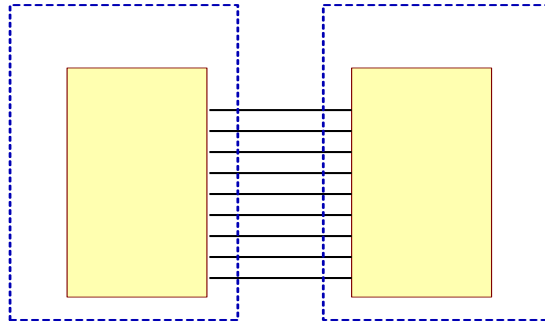


图 4.3.59 LaunchPad 开发板与触控 RGBLED 模块连接图

通过 MSP430G2553 的 P1.0、P1.1、P1.2、P1.3 控制 RGBLED 灯是否工作，该模块通过三个单刀双掷开关实现两种方式的选择；当使用 MSP430G2553 编程来控制 RGBLED 灯时则将灯的另一端与单片机的 I/O P1.4、P1.5、P1.6 连接，实现定时或者自行规定的 PWM 波的控制。

触控 RGBLED 模块通过引出两排排针与 LaunchPad 开发板结合在一块，该模块的 PCB 图及原理图如图 4.3.60 和 4.3.61 所示

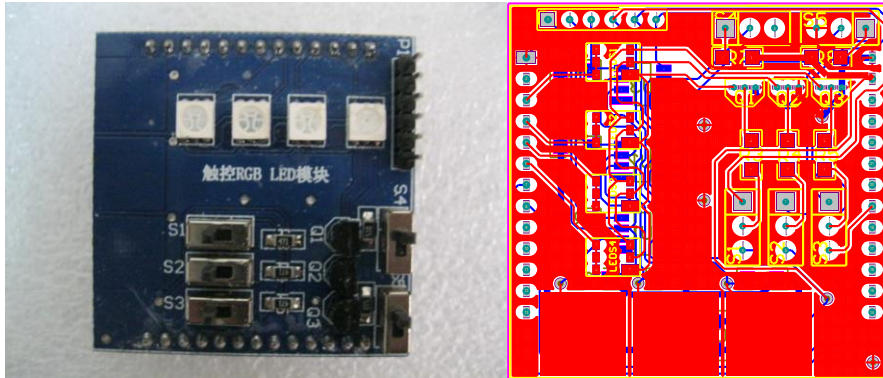


图 4.3.60 触控 RGBLED 模块 PCB 图

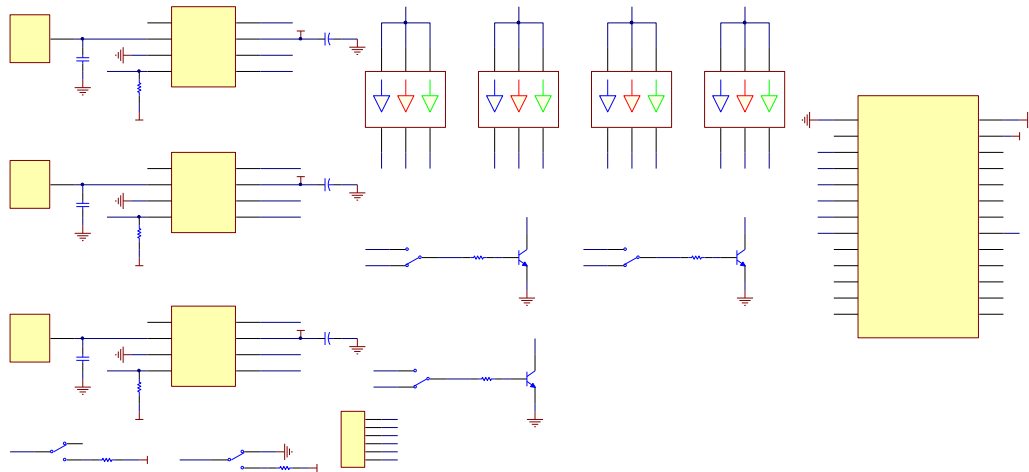


图 4.3.61 触控 RGBLED 模块原理图

4.3.8 频率计及 D/A 转换模块

4.3.8.1. 实验功能模块功能说明

频率信号测量是最常用测量任务之一，掌握频率测量技术也是单片机应用的重要内容。利用单片机和 D/A 转换器件实现模拟信号的输出控制技术，也是应熟练掌握的内容。在频率计及 D/A 转换功能模块的设计中，可提供三路独立的 0—5V 交直流电压输出和一路频率测量输入，能够更好的帮助学习和了解 MSP430G2 系列单片机定时/计数器的使用方法，掌握频率测量和模拟电压输出技术。

4.3.8.2. 实验功能模块组成及工作原理

图 4.3.62 是该频率计及 D/A 转换功能模块的整体组成框图：

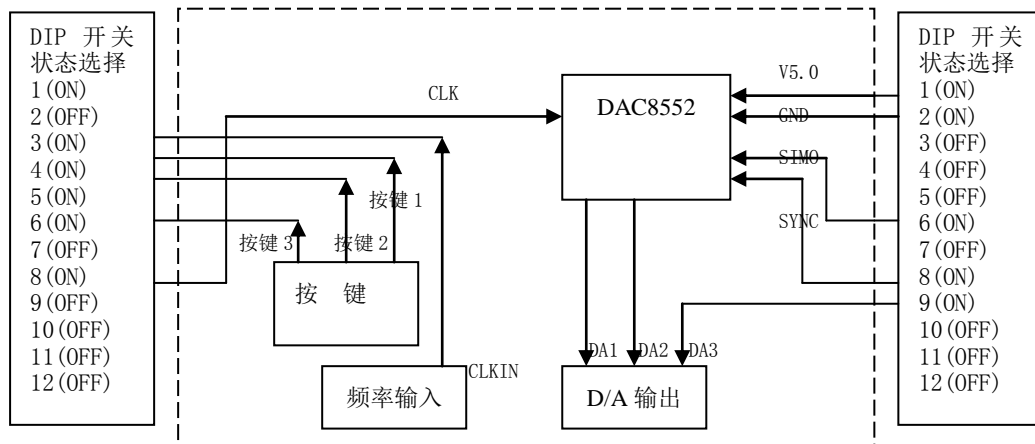


图 4.3.62 频率计及 D/A 转换功能模块组成框图

该实验模块的主要功能有两个，测量频率和 D/A 输出。

1. 测量频率的常用方法为：

方案一：时间闸门法。精确定时 1S，对 1S 以内到来的信号进行上升沿计数，计多少次频率就是多少。

方案二：计算周期法。用 TA 对输入的信号进行捕获，通过计算两次捕获 TAR 的差来计算输入信号的周期，最后算出频率。

方案三：单片机定时/计数器计数法。当 TimerA 选择外部时钟输入时，每当一个外部时钟 TACLK 到达，TAR 的值加 1，可在 1s 内统计 TAR 的值，算出脉冲频率。

方案四：单片机 A/D 转换器比较法。可以通过 ADC10 模块，对假设的高电平设置一个上门限，一旦采样值高于上门限，就将计数器的值加 1，在 1s 内统计计数器的值即可计算出信号的频率。

以上的频率测量方案各有特点：方案一容易实现，简单直观，但是由于使用软件计数所以会占用 CPU 周期，而且无法测量低频率信号，比如当测量 0.1Hz 的信号时，要定时 10s 才能产生捕获到一次上升沿，这显然不合理。方案二由于是测上升沿之间的时间间隔所以适合测量低频率信号。方案三适合于测量高频率信号，且由于是硬件技术所以所占用 CPU 周期很少。方案四适合于测量频率不高，没有一定规律且已知高电平上门限的信号。在硬件连接关系中，可将单片机的 P1.0 口连接至频率输入端，P1.0 可作为 TACLK 输入端，ADC 输入端和普通 IO 口的输入端，可以兼容以上所提到的方案一、方案三和方案四。

2. D/A 输出的常用方法为：

方案一：PWM+二阶低通滤波器。PWM 调制的本质是改变波形中高电平的比例。实际上是波形在一个周期内的“面积”。如果将 PWM 调制波进行低通滤波，取出其直流分量。滤波后的输出电压将线性正比于 PWM 占空比。根据这个原理可以利用 PWM 输出作为低速 DAC 使用

例如，当 MSP430 单片机 3V 供电，PWM 输出在 0—3V 之间切换。PWM 的输出高电平的时间比例越大，其平均值电压越高。经过简单的 RC 低通滤波取出平均电压，则最终的输出电压=占空比*3.0V。RC 滤波时常数越大，剩余的纹波越小，但是 DAC 稳定所需的时间越长。必要时可以使用较小的时间常数进行二次滤波，获得更低的纹波并兼顾速度。

该电路的缺点：首先，PWM 输出电压等于单片机的电源电压，因此该电路的稳定性依赖于电源电压的稳定性。实际中电源电压是很不稳定的。其次，该电路输出阻抗很高，如果后端所接的负载电阻较小，会与内阻分压造成误差。

解决的办法是为单片机提供稳压电源，或者在 PWM 输出部分使用基准源供电，并在输出前加一个运放跟随器，获得理想的输出特性。

PWM-DAC 的缺点是速度很慢只适合于低速运用，但优点是成本低，线性度高，而且数字信号很容易通过光耦隔离，在设计隔离变送系统时非常方便。

方案二：外接 DAC 输出。外接 DAC 输出是较为普遍的方法，在该模块中，我们选用了 TI 公司的 DAC8552 这一外界 DAC 芯片。它具有两路电压输出，SPI 接口，速度快，失调低，16 位精度。具体性能在后面详细介绍。

4.3.8.3 关键芯片及各单元电路介绍

1. D/A 转换器 DAC8552

DAC8552 是一个 16 位，双通道电压输出型 DA 转换器。为用户提供了 SPI 接口。供电电压在 2.7V~5.5V，能够实现轨对轨输出，在 VDD 位 5V 时输出速度可达 30MHz。由于 MSP430G2 系列单片机内存和主频的限制，在该模块中，可实现 1MHz 的输出速率，并得到了完美的输出波形。

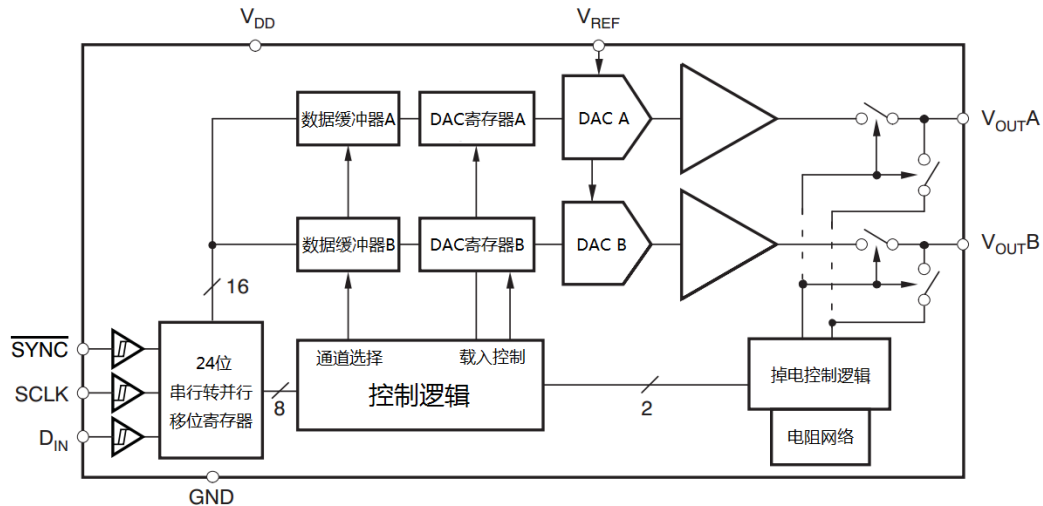


图 4. 3. 63 DAC8552 芯片内部结构及与单片机的典型连接图

由图 4. 3. 78 可知，DAC8552 芯片内部结构由输入施密特触发器，24 位串行转并行控制逻辑，数据输入缓冲，DAC 寄存器，DA 转换电阻网络，输出通道选择逻辑以及 SPI 接口组成。

芯片的原理是：将输入的串行 SPI 信号解析成数据和控制字两组，数据送往 DAC 转换网络，控制字送往通道模式选择逻辑，经过缓冲转换后进行两路模拟信号输出。

以下是 DAC8552 的技术指标

- 实际精度：4LSB
- 最低功耗：2.7V 时 155uA
- 上电后将输出复位至 0
- 电源电压：2.7V 至 5.5V
- 建立时间：10us 至 0.003%FSR
- 低功耗的串行接口外接施密特触发器
- 输出轨对轨
- 双缓冲的输入架构
- 双通道选择
- 温度范围：-40° C 到+125° C
- 标准 SPI 接口(10MHz)

DAC8552 具有 8 个引脚，其引脚功能如表 4. 12 所示

表 4. 9 DAC8552 引脚介绍

PIN#	名称	引脚功能描述
1	VDD	模拟电源电压 (+2.7V 到+5.5V)
2	Vref	基准源输入
3	VoutB	通道 B 输出
4	VoutA	通道 A 输出
5	~SYNC	SPI 帧同步信号
6	SCLK	SPI 时钟信号
7	DIN	SPI 数据输入
8	GND	地

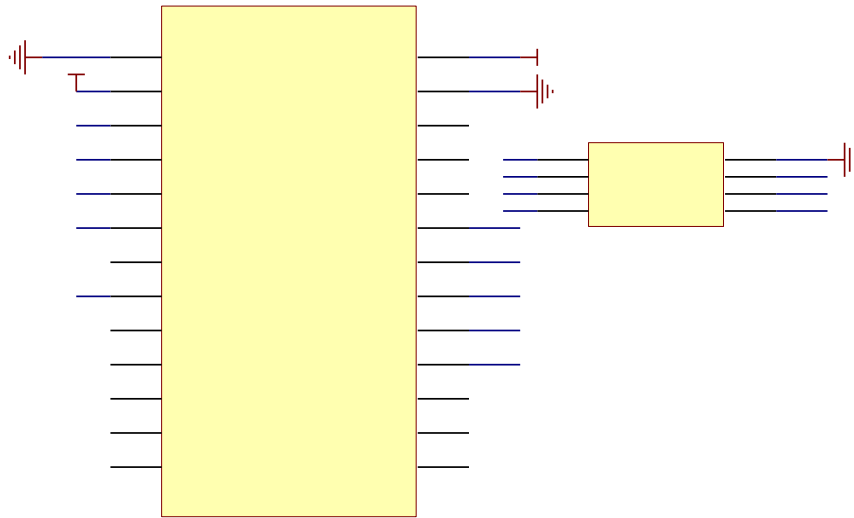


图 4.3.64 DAC8552 连接电路图

如图 4.3.79 所示，DAC8552 与 2553 的连接是通过 SPI 接口连接的，2553 提供标准的 SPI 接口，SPI 是一种同步通信接口，有自己的时钟信号，输入输出口。SYNC 是一个单独的帧同步信号，MSP430 不提供帧同步信号，用 IO 口模拟。

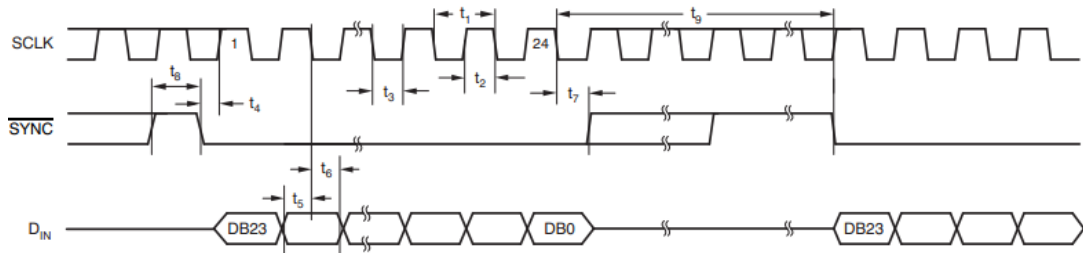


图 4.3.65 DAC8552 写时序图

如图 4.3.80 所示为 DAC8552 的写周期时序图，SYNC 下降沿后有效，DAC8552 在每个 SCLK 的下降沿读取 DIN 上的信号，每帧是 24 位数据，写入数据时 SYNC 必须处于低电平。

单片机通过 SPI 通信发送不同的指令控制 DAC8552 的数据输出，表 4.13 给出了的 SPI 指令集。

表 4.10 SPI 指令集

D23	D22	D21	D20	D19	D18	D17	D16	D15	...	D0	描述
保留	保留	装载 B	装载 A	无效	缓冲器选择	PD1	PD0	16 位转换数据		最高位在前	
0	0	0	0	0	0=A 1=B	0	0	16 位转换数据		最高位在前	写通道 A 或 B 的缓冲器(D18)
0	0	0	1	0	0=A 1=B	0	0	16 位转换数据		最高位在前	写通道 A 或 B 的缓冲器(D18)，并且装载 DAC A
0	0	1	0	0	0=A 1=B	0	0	16 位转换数据		最高位在前	写通道 A 或 B 的缓冲器，并且装载 DAC B
0	0	1	1	0	0=A	0	0	16 位转换数据			写通道 A 或 B 的缓冲器，并且

					1=B			最高位在前	装载 DAC A 和 B		
--	--	--	--	--	-----	--	--	-------	--------------	--	--

DB23						DB12					
0	0	LDB	LDA	X	Buffer Select	PD1	PD0	D15	D14	D13	D12
DB11						DB0					
D11	D10	D9	D8	D7	D6	D5	D5	D3	D2	D1	D0

图 4.3.66 帧格式图

如图 4.3.81 所示为 DAC8552 的帧格式图。

D0—D15 为数据，如在基准源是 5V 的时候，要想输出 5V 的电压，则应往 D15—D0 写入全 1 的数据，设数据值位 data，则写入的数据和电压值之间的关系为：

$$\text{输出电压 } V = 5 * (\text{data} / 65535)$$

PD1 和 PD0 用于选择输出阻抗，详见数据手册。

Buffer Select 位用于选择数据输入到哪一个缓冲区。

LDB 和 LDA 用于选择将数据输出到哪一个通道，如：LDB 位置 1，则将输出到 B 通道。

举例：将数据输入至缓冲区 A，并从从通道 A 输出 5V 电压，基准源是 5V

则应写入控制字：0001 0000 1111 1111 1111 1111

2. 二阶低通滤波器

用电阻电容网络搭建的二阶低通滤波器如图 4.3.82 所示，RC 常数要大于 PWM 周期 6~8 倍，因此 PWM 频率可设置为 66HZ 以上。

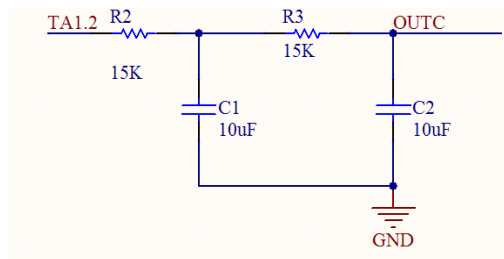


图 4.3.67 二阶低通滤波器

4.3.8.4 实验功能模块信号引脚与 LaunchPad 系统连接

LaunchPad 开发板将 I/O 口及电源引脚引出来，方便与其它模块结合使用，其与频率计 DA 模块的连接关系如图 4.3.83 所示

DAC8552 需要外接从底板上引出 5V 电源作为参考源和外接电源。SPI 接口的时钟口 CLK 接到 P1.5 上，数据口 SIMO 接到 P1.7 帧同步，SYNC 接到 P2.5，测频率信号输入 CLKIN 接到 P1.0。

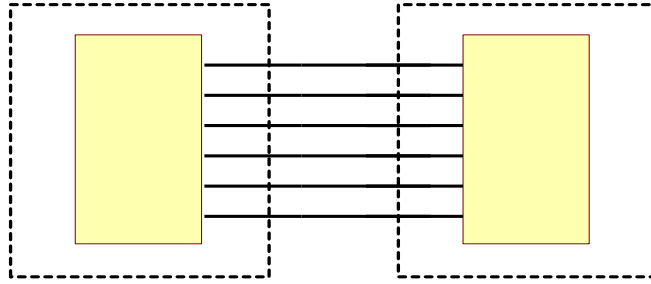


图 4.3.68 LaunchPad 开发板与频率计 DA 模块连接图

程控放大器模块通过引出两排 13 针的排针与 LaunchPad 开发板结合在一块，该模块的实物图及 PCB 图和电路原理图可参见图 4.3.84 和图 4.3.85。

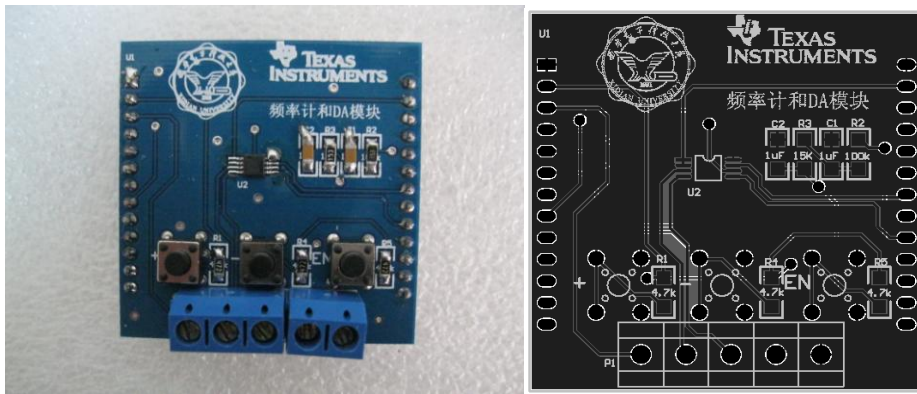


图 4.3.69 程控放大器模块实物图及 PCB 图

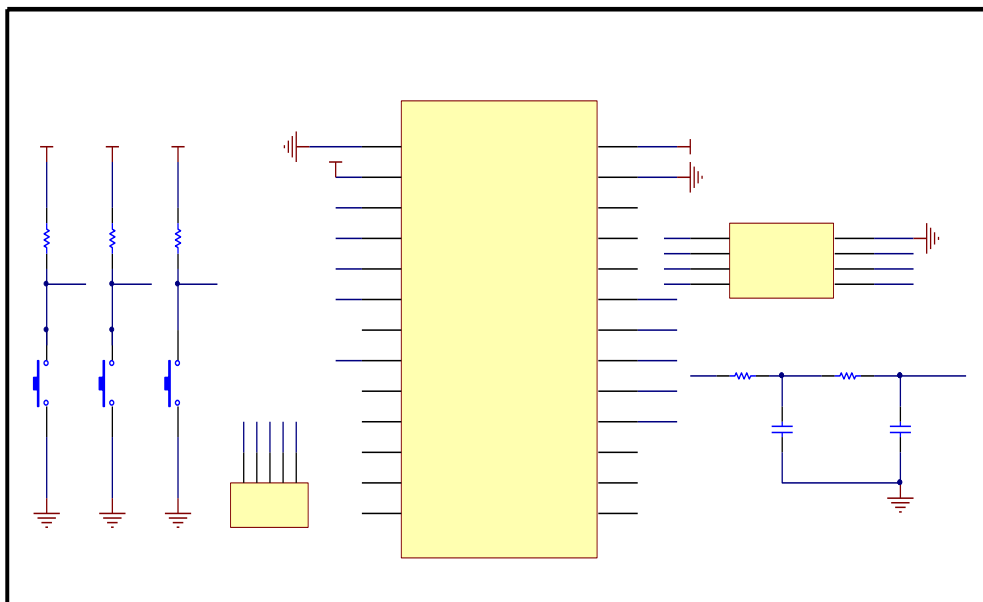


图 4.3.70 程控放大器模块电路原理图

4.3.9 三种通信接口模块

4.3.9.1 实验功能模块功能说明

本模块主要包括三种通信接口：RS232、RS485 及 PS2 接口。由于 MSP430G2553 只有一

个可配置为 UART 的 USCI 功能模块，而 RS232 和 RS485 都是异步串行通信接口，在此，两种接口共用一套单片机串口，用拨动开关进行切换。本模块可实现 PS2 键盘解析、RS232/RS485 通信，帮助读者学习工程中最常用的通信方式以及了解电脑必备外设—键盘的原理。本模块结构分布为：上面接口为 RS232 接头；下面左侧端子为 RS485 接口，右侧为 PS2 键盘接口，也可用于 PS2 鼠标。中间部分为两个拨动开关用于切换 232 和 485，拨到上端连接 232，下端则连接 485；两个芯片分别为 3232 和 3485，即 TTL 转 232/485 电平转换芯片。最上端两个 LED 用于指示数据的收发。

4.3.9.2 实验功能模块组成及工作原理

图 4.3.71 是该三种通信接口模块的整体组成框图：

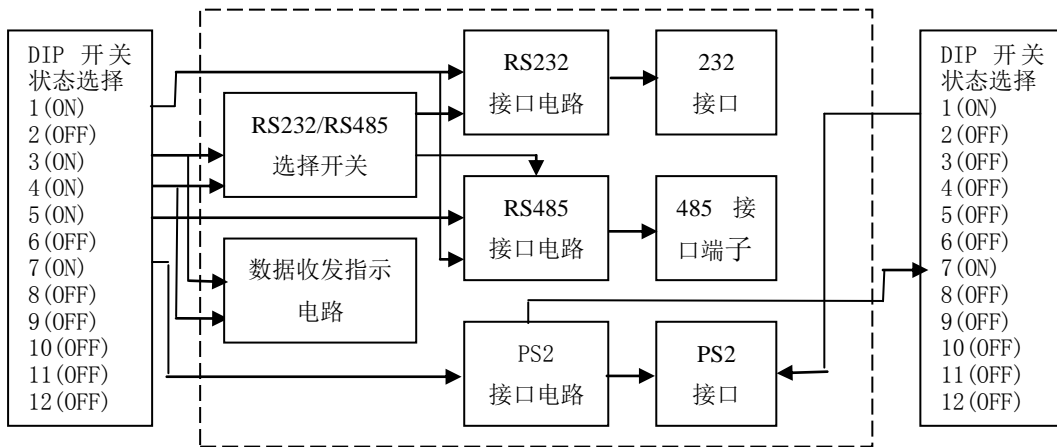


图 4.3.71 程控放大器功能模块组成框图

本模块的电路组成可细分为以下五部分：RS232、RS485、PS2 接口、模块与底板接口和 232/485 切换及数传指示部分，其中 RS232 和 RS485 部分均为 3.3V 供电，PS2 部分为 5V 供电，如果只使用串口通信部分，5V 电源开关可不打开。仅使用 PS2 接口也一样。RS232 和 RS485 采用标准接口转换芯片，将单片机输出的 TTL 电平转换为标准 232 电平或 485 电平，又将外部进来的 232 电平或 485 电平的信号转换为 TTL 电平，传给单片机串口，其大大提高了数据的传输距离，实际应用中可根据实际的通信距离需要选择合适的接口。而 5V 逻辑的 PS2 接口通过分压网络达到电平匹配的目的，同时根据协议要求进行上拉和下拉。将时钟线与数据线接与单片机 I/O 口，这样，当键盘有键按下时，单片机这两个 I/O 口将接收到一定的方波信号，在程序中进行解析，即可判断出键值，进而可以将其显示在液晶屏上。

4.3.9.3 关键芯片及各单元电路介绍

1. RS232 接口电路

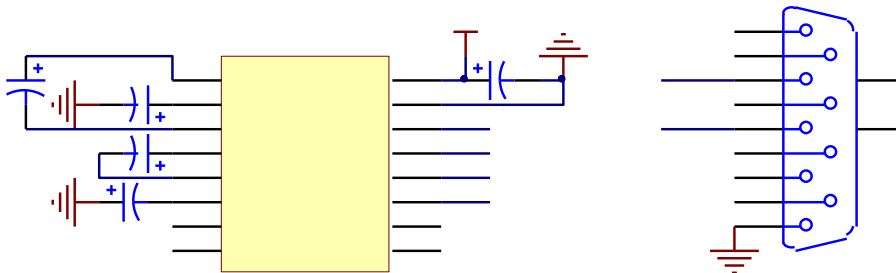


图 4.3.72 RS232 接口电路

使用芯片 TRS3232E 简介:

RS232 电路采用 TI 的 3.3V 供电的双通道 RS232 芯片 TRS3232E, 图中接法为该芯片数据手册中的典型接法。TRS3232E 芯片特点如下:

- ◆ 适用于自动化应用场合
- ◆ 对 RS-232 总线接口提供静电保护
 - $\pm 15\text{-kV}$ 人体静电模式 (HBM)
 - $\pm 8\text{ kV}$ (IEC61000-4-2, 接触放电)
 - $\pm 15\text{ kV}$ (IEC61000-4-2, 气隙放电)
- ◆ 满足或超越 TIA/EIA-232-F 和 ITU v. 28 标准的要求
- ◆ 工作电压范围: 3V 到 5.5V
- ◆ 运行速度可高达 250 kbit/s
- ◆ 两通道接收端和发送端
- ◆ 低供电电流: $300\ \mu\text{A}$ (Typ)
- ◆ 外部电容器: $4 \times 0.1\ \mu\text{F}$
- ◆ 在 3.3-V 供电电压下可接受 5-V 逻辑输入
- ◆ 引脚兼容可选的高速模式设备 (1 Mbit/s): TRSF3232E

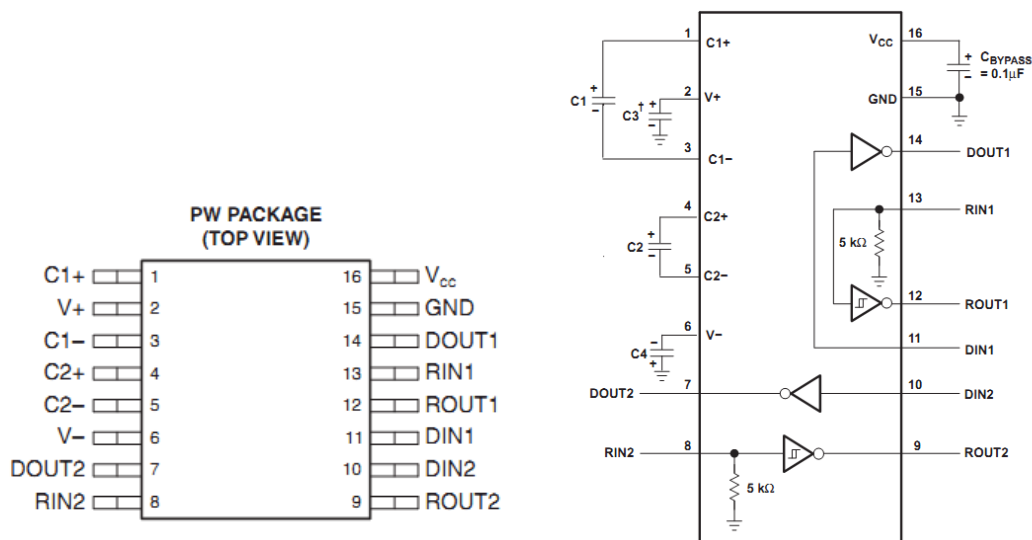


图 4.3.73 引脚分配图及典型接法

DB9 接口引脚定义:

(1) RS-232 端 (DB9 母头/孔型) 引脚定义

引脚序号	2	3	5	1、4、6	7、8
信号定义	TXD	RXD	地	内部相连	内部相连

注: 该口可直接插入计算机的 COM 口

(2) RS-232 端 (DB9 公头/针型) 引脚定义

引脚序号	2	3	5	1、4、6	7、8
信号定义	RXD	TXD	地	内部相连	内部相连

注: 该口可接与计算机通讯的设备

(3) DB9 母头/孔型与 DB9 母头/孔型连接方式: 2-2, 3-3, 5-5
串行口常用的三根线 (TXD RXD GND), 有这三根就可以读写数据了。



图 4.3.74 DB9 实物图

9 芯	信号方向来自	缩写	描述
1	调制解调器	CD	载波检测
2	调制解调器	RXD	接收数据
3	PC	TXD	发送数据
4	PC	DTR	数据终端准备好
5		GND	信号地
6	调制解调器	DSR	通讯设备准备好
7	PC	RTS	请求发送
8	调制解调器	CTS	允许发送
9	调制解调器	RI	响铃指示器

注：调制解调器（在这里是一个例子，它可以是其它的 RS232 终端设备）

2. RS485 接口电路

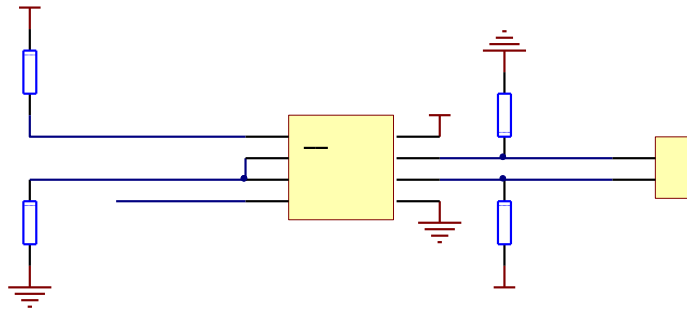
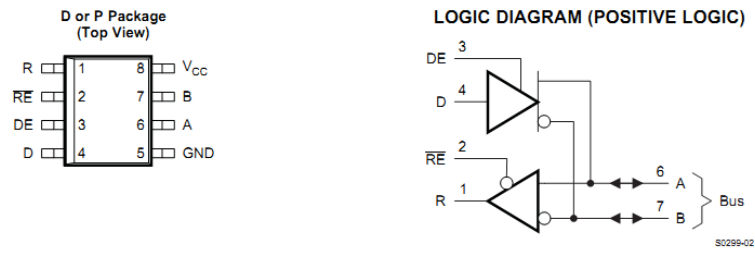


图 4.3.74 RS485 接口电路

RS485 电路采用 TI 的 3.3V 供电的单通道半双工 RS485 芯片 SN65HVD1782。该芯片特点如下：

- ◆ 总线接口故障保护：
 - > $\pm 70\text{ V}$ ('HVD1780, 81)
 - > $\pm 30\text{ V}$ ('HVD1782)
- ◆ 工作电压范围 3.3V 到 5V
- ◆ 总线引脚 $\pm 16\text{ kV}$ 人体静电保护
- ◆ 最多支持 320 个节点单元载荷
- ◆ 接收器开路、短路和空闲线路条件下的失效保护，
- ◆ 低功耗
 - 低待机供电电流，最大 1 mA
 - 工作期间不活动时 ICC 4 mA

- ◆ 引脚兼容工业标准 SN75176
- ◆ 信号变化范围从 115 kbps, 1 Mbps, 最高至 10 Mbps



DEVICE INFORMATION

DRIVER FUNCTION TABLE

Input	Enable	Outputs		Driver State
		A	B	
D	DE	A	B	
H	H	H	L	Actively drive bus High
L	H	L	H	Actively drive bus Low
X	L	Z	Z	Driver disabled ⁽¹⁾
X	OPEN	Z	Z	Driver disabled by default ⁽¹⁾
OPEN	H	H	L	Actively drive bus High by default

图 4. 3. 75 引脚分布及驱动功能表

3. PS2 接口电路

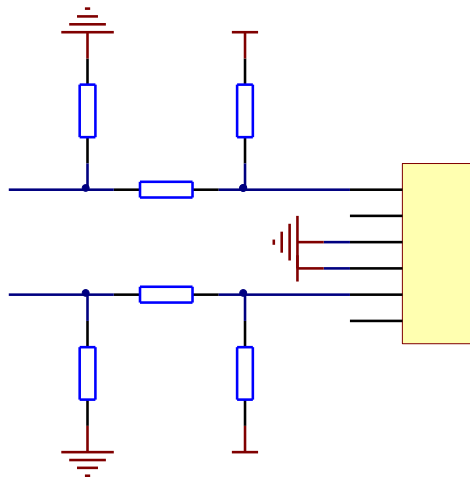


图 4. 3. 76 PS2 接口电路

PS2 连接器种类及各种连接器引脚定义如下图所示：

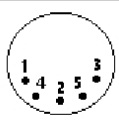
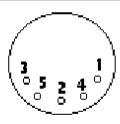

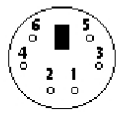


Male 公的	Female 母的	5-pin DIN (AT/XT):	5 脚 DIN(AT/XT)
		1 - Clock	1—时钟
		2 - Data	2—数据
		3 - Not Implemented	3—未实现, 保留
		4 - Ground	4—电源地
(Plug) 插头	(Socket) 插座	5 - +5v	5—电源+5V
Male 公的	Female 母的	6-pin Mini-DIN (PS/2):	6 脚 Mini-DIN(PS/2)
		1 - Data	1—数据
		2 - Not Implemented	2—未实现, 保留
		3 - Ground	3—电源地
		4 - +5v	4—电源+5V
(Plug) 插头	(Socket) 插座	5 - Clock	5—时钟
		6 - Not Implemented	6—未实现, 保留
		6-pin SDL:	6 脚 SDL
		A - Not Implemented	A—未实现, 保留
		B - Data	B—数据
		C - Ground	C—电源地
		D - Clock	D—时钟
		E - +5v	E—电源+5V
		F - Not Implemented	F—未实现, 保留

图 4.3.77 PS2 连接器种类及各种连接器引脚定义图

在此，我们使用的位圆形六脚插座母头，与常见 PS2 鼠标、键盘相对应。由于 PS2 位 5V 逻辑，为了使其与单片机 3.3V 逻辑匹配，采用了图 4.3.76 所示的电阻分压网络。当时钟线输出（PS2 键盘输出）高时，R4 与 R5 分压，单片机输入为 3.3V；当时钟线输入为低，输入单片机的电压也为低；数据线同理。另外，由于数据和时钟都是集电极开路的，这就意味着它们通常保持高电平而且很容易下拉到地（逻辑 0），任何连接到 PS/2 鼠标、键盘或 host 的设备在时钟和数据线上要有一个大的上拉电阻，“置 0”就把线拉低；“置 1”就让线上浮成高电平。所以图 4.3.76 中采用了上拉和下拉电阻。即键盘无输入时，时钟和数据线保持 5V 高电平，而单片机引脚输入为 2.5V 逻辑高点平，满足 PS2 通信协议要求。

4.232/485 切换及数传指示

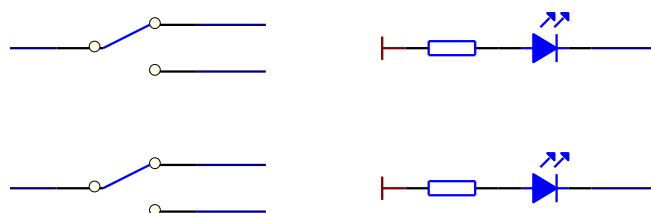


图 4.3.78 232/485 切换及数传指示

RS232/485 的切换采用两个拨动开关（单刀双掷），当然，使用一个双刀双掷开关更为简便。数传指示方面，将 LED 灯直接接于单片机串口数据输入输出端，另一端通过电阻上拉到 VCC3.3，由于串口不发数据时处于高电平状态，此时，等两端压差小于 0.7V，灯不亮。当有数据发送时，会有若干个 0 被发送，但由于间隔时间较短，人眼无法识别，看上去，每发送或接收一次数据，LED 会闪烁一次，以此来标示数据的成功传送。由于 LED 工作电流为 10mA，故选取 330 欧电阻作为上拉电阻。

4.3.9.4 实验功能模块信号引脚与 LaunchPad 系统连接

LaunchPad 开发板将 I/O 口及电源引脚引出来，方便与其它模块结合使用，其与三种通信接口模块的连接关系如图 4.3.79 所示

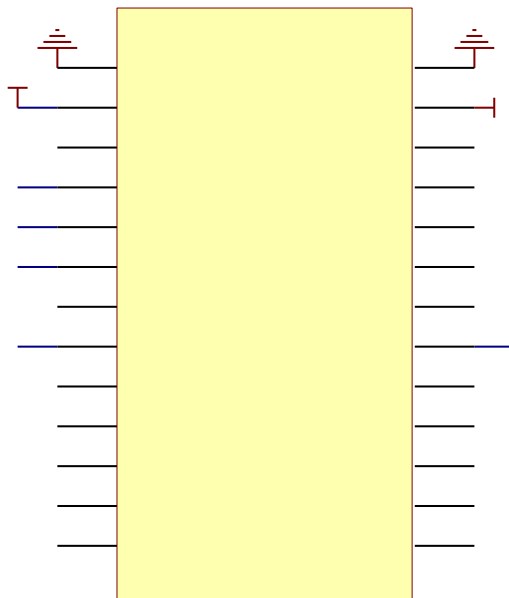


图 4.3.79 LaunchPad 开发板与三种通信模块连接图

本模块使用了 3.3V 及 5V 电源，另外，使用了 USCI 模块的 UART 引脚—P1.1 和 P1.2；485 还另加了方向切换引脚 P1.3；PS2 则使用了带 I/O 中断功能的两个引脚 P1.5、P1.6 作为时钟和数据线。模块与底板连接时，注意 DB9 即 RS232 接口向上，与底板插座一对一连接即可。

三种通信接口模块通过引出两排 13 针的排针与 LaunchPad 开发板结合在一块，该模块的 PCB 图和电路原理图可参见图 4.3.80 和图 4.3.81。

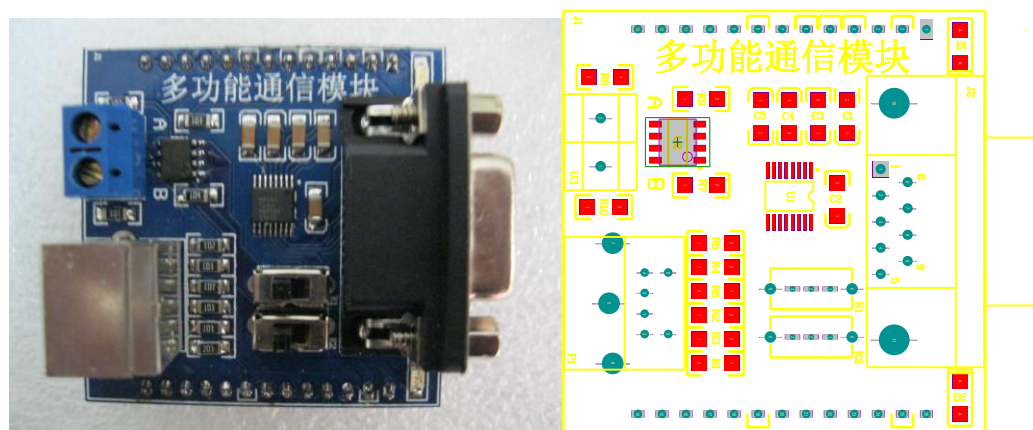


图 4.3.80 三种通信接口模块的 PCB 图

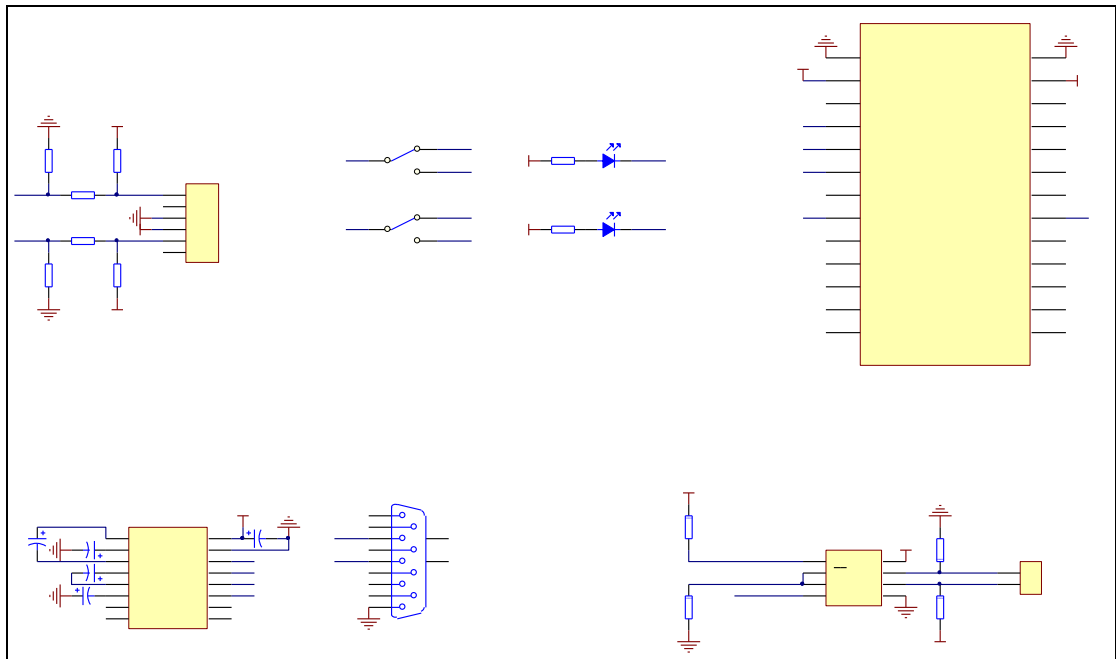


图 4.3.81 三种通信模块电路原理图

4.3.10 声音强度检测模块

4.3.10.1 实验功能模块功能说明

声音是通过物体振动产生的，通过介质（如空气或固体、液体）传播并能被人或动物听觉器官所感知的波动现象。人耳所能听到的声音的频率范围一般在 20Hz 到 20KHz 之间，而人主观上感觉到的声音的大小称为响度，单位为分贝（dB）。人耳刚能听见的声音大小大约为 1dB，这叫闻阈；当大于 100dB 的噪声就会使耳朵发胀，疼痛，这样的声响叫痛阈。人们总是处于不同的响度的噪声包围之中，而本功能模块为检测日常生活中的声音的大小带来了很大的方便。

该功能模块通过传声器采集声音信号，加上部分前置放大电路和峰值检波电路，最后与单片机的 ADC 模块相连，实现对周围声音强度的实时检测，再加上 LCD12864 点阵显示，可以准确的将检测结果显示出来。通过本实验的学习，学生可以熟练的掌握本模块的电路设计方法，单片机 ADC 模块的应用以及 LCD12864 点阵显示原理。

4.3.10.2 实验功能模块组成及工作原理

图 4.3.82 是该声音强度检测模块的整体组成框图：

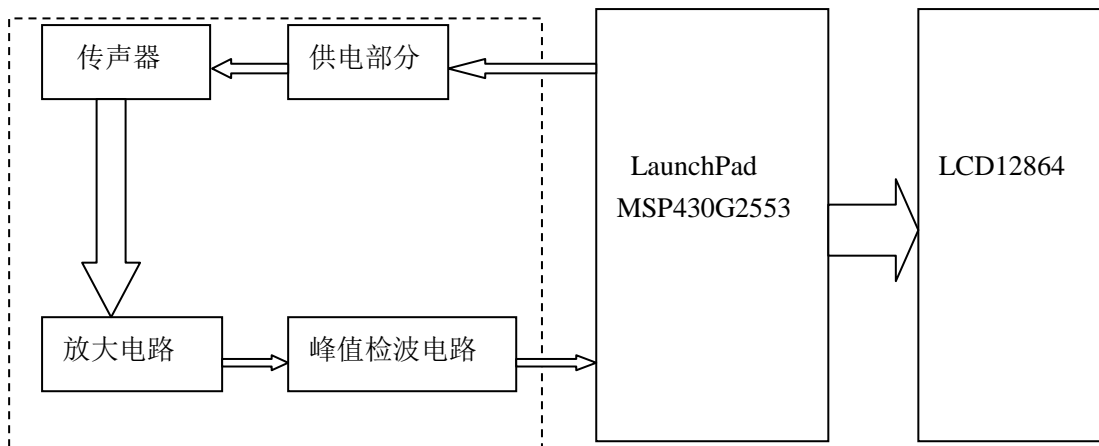


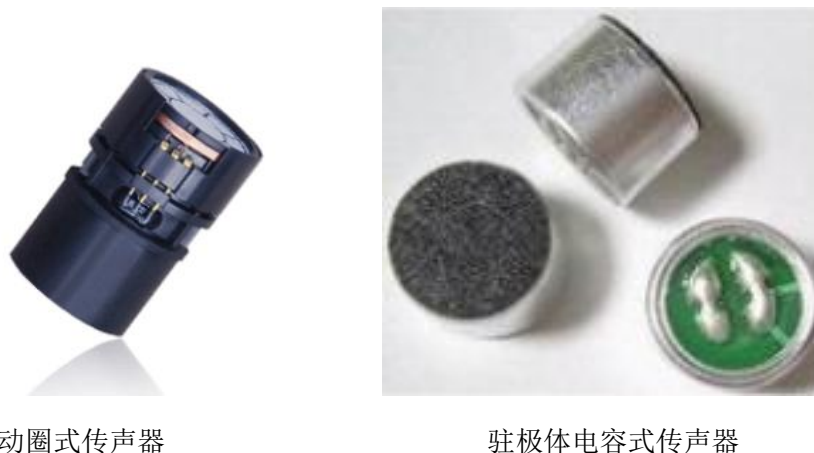
图 4.3.82 声音照度检测模块组成框图

该功能模块直接利用单片机的 V_{CC} 供给系统 3.3V 电压，省去了较为复杂的外围电源部分，传声器将声音信号转换为电压信号，由于其输出信号很微弱，不利于单片机 ADC 采集，故需要配合适当的前置放大电路使用，放大后的信号通过峰值检波电路将采集到的信号的峰值检出并输入单片机进行 A/D 转换和采样。经过单片机的处理后，将检测结果通过 LCD12864 显示出来。这就是该实验模块的工作原理。

4.3.10.3 关键芯片及各单元电路介绍

1. 传声器

传声器是将声音信号转换为电信号的能量转换器，俗称话筒（麦克风）。而麦克风主要分为动圈式和驻极体电容式，其中动圈式传声器音质很好，但体积较大，电容式传声器体积小，成本低廉，在手机、电话等设备中应用十分广泛。图 4.xx 为常见的两种传声器。



动圈式传声器

驻极体电容式传声器

图 4.3.83 传声器

基于驻极体电容式传声器的诸多优点和试验模块的本身特点，在本次试验模块设计中首选驻极体电容式传声器。

1) 驻极体电容式传声器简介

驻极体电容式传声器是用事先已注入电荷而被极化的驻极体代替极化电源的电容传声器。其有两种类型，一种是用驻体高分子薄膜材料做振膜（振模式），此时振膜同时担负着声波接收和极化电压双重任务；另一种是用驻极材料做后极板（背极式），这时它仅起着极化电压的作用。由于驻极体传声器具有体积小、价格低廉、电声性能好、结构简单等特点，被广泛应用于无线话筒、盒式录音机及声控等电路中，从而成为最常用的电容传声器。由于输入输出阻抗很高，所以要在这种话筒外壳内设置一个场效应管作为阻抗转换器，为此驻极体电容式话筒在工作时需要直流工作电压。图 4.3.84 为本次试验模块选用的直插电容式传声器 EM-9767P



图 4.3.84 电容式传声器 EM-9767P

2) 驻极体电容式传声器结构及工作原理

驻极体传声器的基本结构是由一片单面涂有金属的驻极体薄膜与一个上面有若干小孔的金属电极（背称为背电极）构成。驻极体面与背电极相对，中间有一个极小的空气隙，形成一个以空气隙和驻极体作绝缘介质，以背电极和驻极体上的金属层作为两个电极构成一个平板电容器。电容的两极之间有输出电极。图 4.3.85 为实际驻极体话筒内部结构。

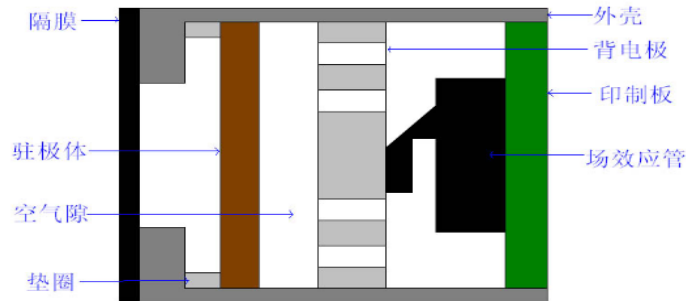


图 4.3.85 驻极体话筒内部结构

由于实际电容器的电容量很小，输出的电信号极为微弱，输出阻抗极高，可达数百兆欧以上。因此，它不能直接与放大电路相连接，必须连接阻抗变换器。通常用一个专用的场效应管和一个二极管复合组成阻抗变换器。内部电气原理如图 4.3.86 所示。

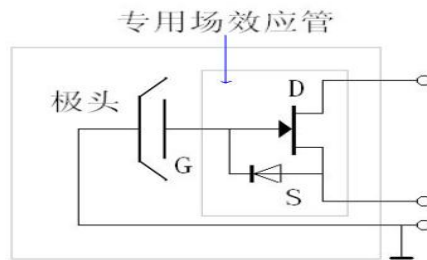


图 4.3.86 驻极体话筒内部电气原理

由于高分子极化膜上生产时就注入了一定的永久电荷(Q)，而没有放电回路，所以这个电荷量是不变的，在声波的作用下，极化膜随着声音震动，因此和背极的距离也跟着变化，也就是说极化膜和背极间的电容是随声波变化的。根据公式： $Q = CU$ ，则 $U = Q/C$ 可知，驻极体总的电荷 Q 是不变的，所以当极板在声波压力下后退时，两极板间距变大，导致电容 C 减小，故电压 U 随电容 C 的变小而变大，反之当电容量增加时电容两极之间的电压就会成反比的下降。最后通过阻抗非常高的场效应管将电容两端的电压取出来，同时进行适当的放大，我们就可以得到和声音对应的电压了。由于场效应管是有源器件，需要一定的偏置和电流才可以工作在放大状态，因此，驻极体话筒都要加一个直流偏置才能工作。这就是驻极体话筒的工作原理。

3) EM-9767P 传感器的性能参数

传声器的类型很多，本设计选用 EM-9767P 作为传声器，其相关电性能参数为：

基准工作电压 (Standard Operation Voltage): 3VDC

阻抗 (Impedance): 2.2k Ω (maximum)

灵敏度 (Sensitivity): -52dB \pm 2dB (0dB=1V/ μ bar, 1kHz)

信噪比 (S/N Ratio): >60dB

消耗电流 (Current Consumption): 0.5mA (maximum)

频率响应曲线 (Frequency Response): 如图 4.3.87 所示

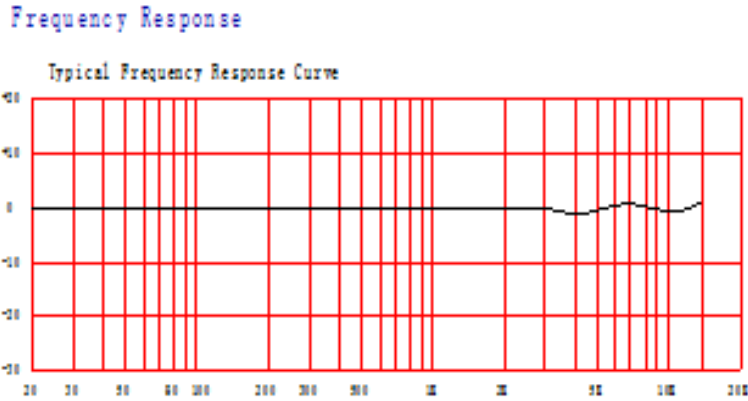


图 4.3.87 EM-9767P 传声器的频响特性曲线

2. 前置放大电路

前面已经提过，由于传声器的输出与声音的大小有关，小可至微伏级别，如果直接输入单片机的 ADC 信号将很难被采集到，所以适当的放大信号，对整个系统的灵敏度，测试的稳定性，ADC 的采集与转换等都有很大的影响。基于此，在本实验模块中，前置放大电路如图 4. xx 所示。

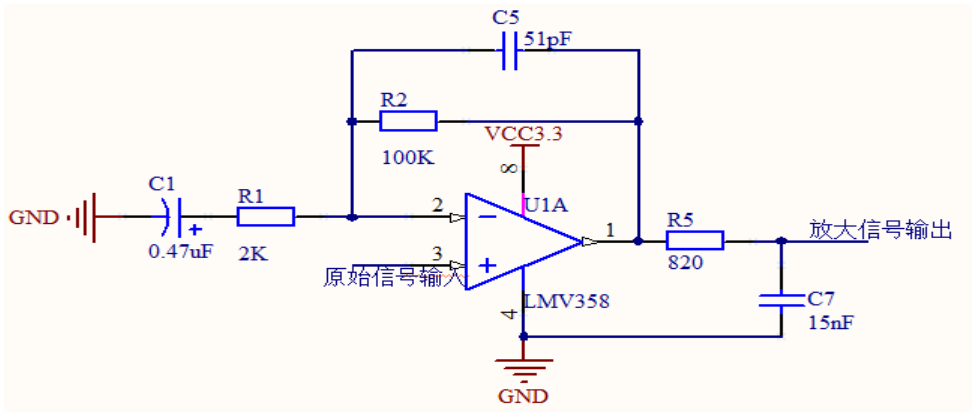


图 4.3.88 前置放大电路

本设计选用的运放是 TI 公司的 LMV358 低功耗运算放大器，其优点为：

- ◆双通道低电压（2.7-5V）版本
- ◆rail-to-rail 输出摆幅能力
- ◆共模输入电压范围宽，包括接地
- ◆单位增益频带宽（约 1MHz）
- ◆双极性输入与输出，提高了抗噪声性能的阶段和更高的输出电流驱动
- ◆低的供应电流（约 145uA）
- ◆无交越失真，节省空间包装，适用于任何一种电池供电和便携式电子设备

由上图可知，经传声器转换后的声音信号从 LMV358 的 3 端输入，即为同向比例运算放大电路的接法，而同向比例运算放大器由于引入了深度电压负反馈，所以其输出电阻很小 (R_o 约为 0)，而根据“虚断”的概念，该电路的输入电流为 0，所以输入电阻很大 (R_i 为无穷)。由同向比例运算放大器的增益计算方法可知，其放大倍数为：

$$A_u = R_2/R_1 + 1 = 100K/2K + 1 = 51$$

而放大后的信号由 LMV358 的 1 端，经过负载 R5 输出。而负载电容 C7 与 R5 组成一个 RC 滤波电路，滤除因输入端引入的干扰噪声，平滑电压信号。

3. 峰值检波电路

峰值检波电路是一种能记忆信号峰值的电路，其输出电压的大小，一直追随输入信号的峰值，而且保持在输入信号的最大峰值。本设计加入峰值检波电路，为单片机的采集和处理带来了很大的方便。具体的峰值检波电路如图 4.3.89 所示。

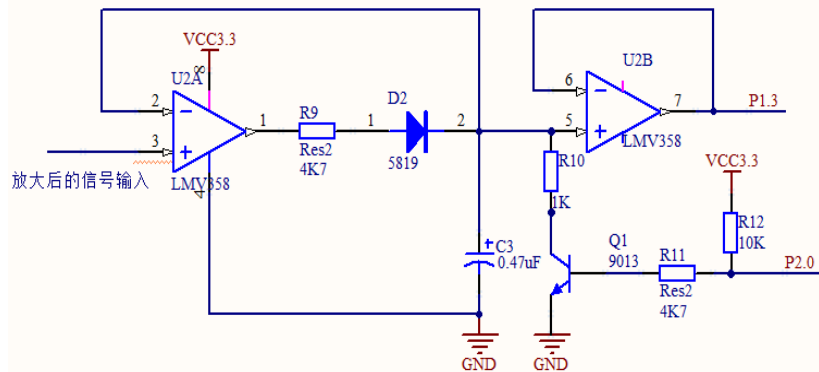


图 4.3.89 峰值检波电路

这是一个典型的峰值检波电路，放大后的信号从 LMV358 的 3 端输入，我们可以控制电容 C3 的放电过程，即当 P2.0 置高时，三极管 Q1 导通，保持器电容放电；当 P2.0 置低时，三极管 Q1 截止，峰值信号可以被锁存在存储器中。同时，该电路还加入一定的保护措施以及阻抗匹配电路（如跟随器等），使得该电路可以重复的工作。检波后的信号由 LMV358 的 7 端输出，通过 I/O 口 P1.3 与单片机相连。

总之，该系统就是先将声音信号转换为电信号，再将电信号转换为声强显示出来，这就涉及到声学计算的问题，与整个系统的灵敏度（包括传声器灵敏度，放大电路的增益以及各种滤波电路的衰减等）有着密切的关系，其具体的转换方式将在软件中进行说明。

4.3.10.4 实验功能模块信号引脚与 LaunchPad 系统连接

为了将试验模块与实验板方便连接，故对应实验板上的两排 13 排针底座，设计了两排 13 排针，如图 4.3.90 所示。而本模块中只用到 P1.3、P2.0 和 VCC3.3，其中，VCC3.3 为整个系统提供电源电压，P2.0 通过软件改变其高低电平来控制保持器电容的放电过程，而 P1.3 则是单片机 ADC 转换的输入通道 A3，与调理好的信号相连，实现 A/D 转换。

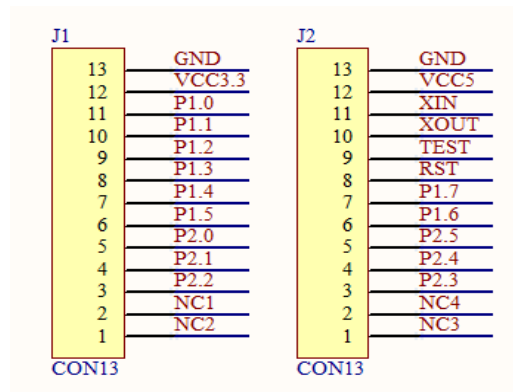


图 4.3.90 声音强度检测模块的外接引脚

该模块的最终原理图，PCB 图以及模块实物图如图 4.3.91、4.3.92 和 4.3.93 所示。

第五章 MSP430G2 系列单片机基础实验

第一节 I/O 基础实验

5.1.1 矩阵键盘按键扫描实验

1. 实验准备

lanchpad 核心实验板一套，按键开关 16 个，1K 直插电阻 4 个，万用表一部，导线若干。

2. 实验目的

掌握矩阵键盘的扫描的原理以及 IO 口输入扫描的方法。

3. 实验步骤

(1) 用万用表的电阻档检测电阻是否正常，用二极管档检测所需导线是否完好，并检查按键开关是否有损坏

(2) 按照下图所示在面包板上搭建电路

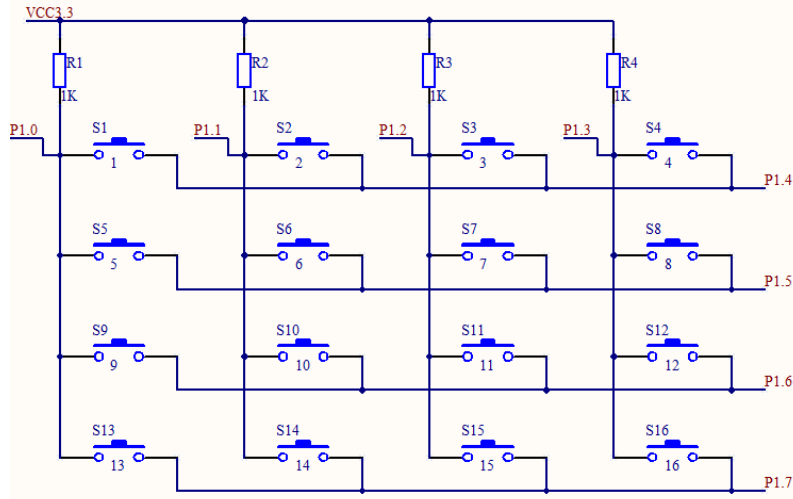


图 5.1.1 矩阵键盘原理图

(3) 对照上图，将 VCC3.3、P1.0~P1.7 分别与 lanchpad 的 VCC3.3、P1.0~P1.7 相连，并仔细检查电路是否连接正确无误

(4) 打开电源，并打开 CCS 对单片机仿真，编写一段 IO 读取程序 IO 中高电平个数的总和

(5) 打开“watch”“register”，在按下其中任意一个按键开关，观察开关在按下时“P1IN”寄存器的数值，并与原理图相比较，看是否相符。

4. 程序示例

```
#include <msp430g2553.h>
typedef unsigned char uchar;
typedef unsigned int uint;

/*****全局变量*****/
uchar key_Pressed; //按键是否被按下:1--是, 0--否
uchar key_val; //存放键值
uchar key_Flag; //按键是否已放开: 1--是, 0--否
//设置键盘逻辑键值与程序计算键值的映射
uchar key_Map[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16};
```

```

/*****
函数名称: Init_Keypad
功    能: 初始化扫描键盘的 IO 端口
参    数: 无
返回值 : 无
*****/
void Init_Keypad(void)
{
    P1DIR = 0xf0;          //P1.0~P1.3 设置为输入状态, P1.4~P1.7 设置为输出状态
    P1OUT |= 0xf0;        // P1.4~P1.7 输出高电平
    key_Flag = 0;
    key_Pressed = 0;
    key_val = 0;
}
/*****
* Check_Key(), 检查按键, 确认键值
*****/
/*****
函数名称: Check_Key
功    能: 扫描键盘的 IO 端口, 获得键值
参    数: 无
返回值 : 无
*****/
void Check_Key(void)
{
    uchar row , col, tmp1, tmp2;

    tmp1 = 0x80;
    for(row = 0; row < 4; row++)          //行扫描
    {
        P1OUT = 0xf0;                    //P1.4~P1.7 输出全 1
        P1OUT -= tmp1;                   //P1.4~p1.7 输出四位中有一个为 0
        tmp1 >>=1;
        if ((P1IN & 0x0f) < 0x0f)        //是否 P1IN 的 P1.0~P1.3 中有一位为 0
        {
            tmp2 = 0x01;                  // tmp2 用于检测出那一位为 0
            for(col = 0; col < 4; col++)  // 列检测
            {
                if((P1IN & tmp2) == 0x00) // 是否是该列, 等于 0 为是
                {
                    key_val = key_Map[row * 4 + col]; // 获取键值
                    return;                          // 退出循环
                }
                tmp2 <<= 1;                // tmp2 左移 1 位
            }
        }
    }
}
/*****
函数名称: delay

```

功 能：延时约 15ms，完成消抖功能

参 数：无

返回值：无

```
*****/
```

```
void delay()
```

```
{
```

```
    uint tmp;
```

```
    for(tmp = 12000;tmp > 0;tmp--);
```

```
}
```

```
*****
```

函数名称：Key_Event

功 能：检测按键，并获取键值

参 数：无

返回值：无

```
*****/
```

```
void Key_Event(void)
```

```
{
```

```
    uchar tmp;
```

```
    P1OUT &= 0x00;          // 设置 P1OUT 全为 0，等待按键输入
```

```
    tmp = P1IN;            // 获取 p1IN
```

```
    if ((key_Pressed == 0x00)&&((tmp & 0x0f) < 0x0f)) //如果有键按下
```

```
    {
```

```
        key_Pressed = 1;      // 如果有按键按下，设置 key_Pressed 标识
```

```
        delay();             //消除抖动
```

```
        Check_Key();         // 调用 check_Key(), 获取键值
```

```
    }
```

```
    else if ((key_Pressed == 1)&&((tmp & 0x0f) == 0x0f)) //如果按键已经释放
```

```
    {
```

```
        key_Pressed = 0;      // 清除 key_Pressed 标识
```

```
        key_Flag = 1;         // 设置 key_Flag 标识
```

```
    }
```

```
    else
```

```
    {
```

```
        _NOP();
```

```
    }
```

```
}
```

5.1.2 控制数码管显示数字实验

1. 实验准备

LaunchPad 一个，共阴极数码管一个，200K 电阻 8 个，导线若干。

2. 实验目的

掌握动态扫描、静态扫描数码管的原理和方法，以及 IO 口输出的原理和方法。

3. 实验步骤

(1) 将 LaunchPad 插到实验系统标准板上。

(2) 按下图在实验系统标准板上的面包板上搭好实验电路，将导线接到 LaunchPad 的相应引脚上。

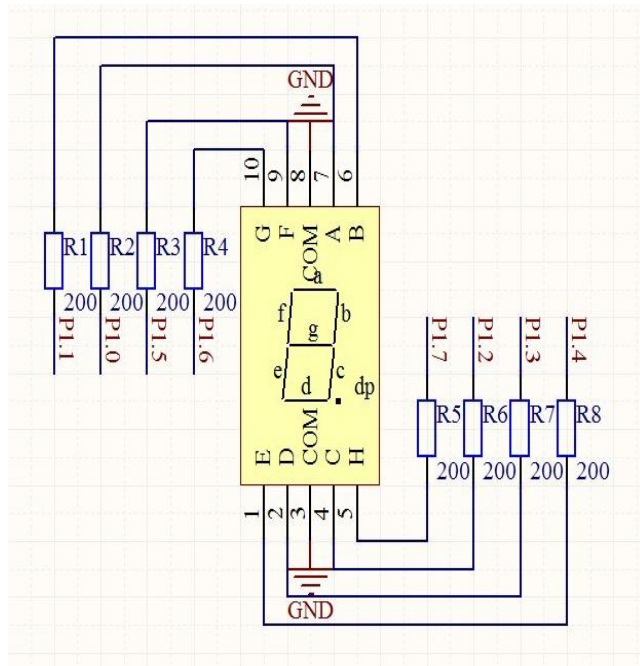


图 5.1.2 数码管原理图

- (3) 打开 CCS 新建工程，将示例程序写入 main.c 文件。
- (4) 在工具栏上如下图所示点击鼠标，开始硬件仿真。

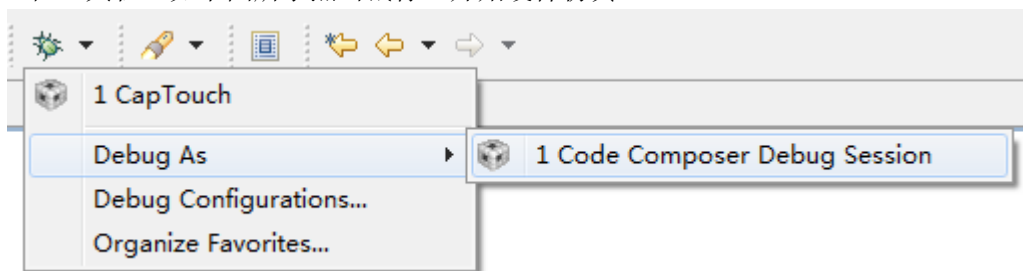


图 5.1.3 仿真步骤

- (5) 不断点击工具栏上的“Step Over”按键，观察数码管上显示的字符。

4. 程序示例

```
#include <MSP430g2553.h>

#define Dis00    0x3F    //数码管显示字符"0"
#define Dis01    0x06    //数码管显示字符"1"
#define Dis02    0x5B    //数码管显示字符"2"
#define Dis03    0x4F    //数码管显示字符"3"
#define Dis04    0x66    //数码管显示字符"4"
#define Dis05    0x6D    //数码管显示字符"5"
#define Dis06    0x7D    //数码管显示字符"6"
#define Dis07    0x07    //数码管显示字符"7"
```

```

#define Dis08      0x7F      //数码管显示字符"8"
#define Dis09      0x6f      //数码管显示字符"9"
#define Dis0A      0X77      //数码管显示字符"A"
#define Dis0B      0x7C      //数码管显示字符"B"
#define Dis0C      0x39      //数码管显示字符"C"
#define Dis0D      0x5E      //数码管显示字符"D"
#define Dis0E      0x79      //数码管显示字符"E"
#define Dis0F      0X71      //数码管显示字符"F"

const unsigned char seg_[16] = {0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07,
                                0x7f, 0x6f, 0x77, 0x7c, 0x39, 0x5e, 0x79, 0x71};

void delay_(int m)          //延时子程序
{
    int i;
    int j;
    for (i=m;i>0;i--)
        for(j=100;j>0;j--);
}

void Display( signed char i ) //数码管显示子程序
{
    // unsigned char result();
    if(0<=i<=15)
        P1OUT = seg_[i];      // 数码管显示数值

    // delay_(100);
    // while(1);
}

void main(void)
{

```



```

WDTCTL = WDTPW + WDTHOLD; // 关闭看门狗

P1SEL = 0; //设置 P1 口全为普通 IO 口

P1DIR = 0xff; //将 P1 所有端口设为输出

P1OUT = 0X00; //p1 口先全部输出低电平

unsigned char i;

for(i=0;i<16;i++) //依稀显示从“0”到“F”

    Display(i);

}

```

5.1.3 按键扫描并控制数码管显示键值实验

1. 实验准备

lanchpad 实验核心板一套，CH452 驱动芯片一块，数码管 4 个，二极管 4 个，270 欧电阻 8 个，1K 电阻 4 个，按键开关 16 个，0.1uF 电容一个，万用表一部，导线若干。

2. 实验目的

掌握 CH452 芯片同时驱动键盘扫描和数码管显示的功能和应用。

3. 实验步骤

- (1) 用万用表分别检查电阻值是否准确，二极管、导线是否完好。
- (2) 按照下图所示在面包板上搭建电路。

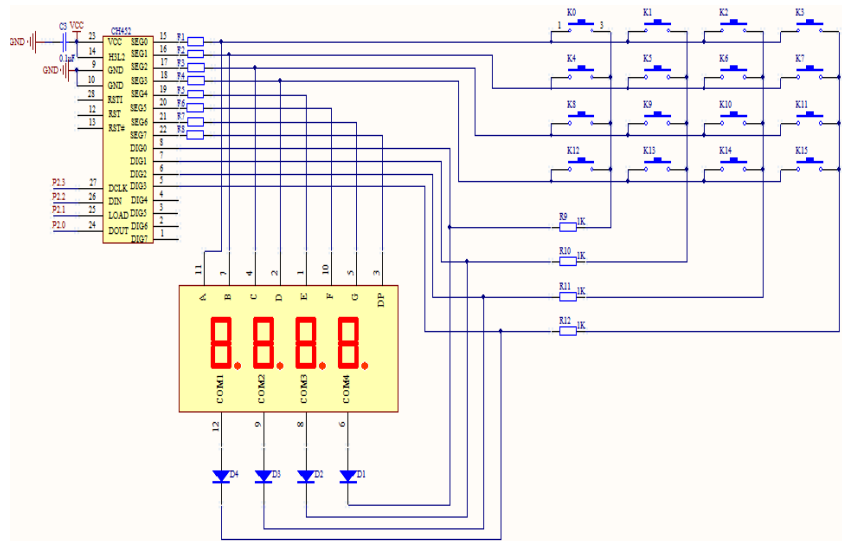


图 5.1.4 键盘与数码管原理图

- (3) 将 CH452 的 DOUT, LOAD, DIN, DCLK, VCC 分别与 lanchpad 的 P2.0, P2.1, P2.2, P2.3, VCC 相连，并用 USB 为系统供电
- (4) 打开 CCS 编译环境，编写程序代码并对单片机进行仿真
- (5) 打开“watch”“register”窗口，按下键盘中任一按键，看键值与数码管的显示值是否对应

4. 程序示例

```

#include <msp430g2553.h>
#include "Key&Display.h" //自定义头文件

```

```

void main( void )
{
    WDTCTL = WDTPW + WDTHOLD; // 停止开门狗
    Init_4lines_Mode(); // 初始化 4 线工作模式
    Send_Command(CH452_RESET); // CH452 芯片内部复位
    Send_Command(KeyDisplay_ON); // 允许显示驱动并启动键盘扫描
    // 开中断, P2.0 接 CH452 的 DOUT 引脚, 当有键按下时, DOUT 上产生由高到低的脉冲
    P2IE|=BIT0;
    P2IES|=BIT0;
    P2IFG&=~BIT0;
    _EINT();
    while(1)
    {
    }
}
// 中断处理函数
#pragma vector = PORT2_VECTOR // 中断处理程序, 接收到 DOUT 脉冲信号时, 运行之
__interrupt void Port2(void)
{
    unsigned char Keyvalue;
    Send_Command(CH452_GET_KEY); // 单片机向 CH452 发送读取按键代码命令
    Keyvalue=Key_Read();
    switch(Keyvalue)
    {
        case 0x40:// 按键 K0 按下
        {
            Send_Command( NDis1); // 第 1 位数码管不显示
            Send_Command(Dis00); // 第 0 位数码管显示 0
            break;
        }
        case 0x41:// 按键 K1 按下
        {
            Send_Command( NDis1); // 第 1 位数码管不显示
            Send_Command(Dis01); // 第 0 位数码管显示 1
            break;
        }
        case 0x42:// 按键 K2 按下
        {
            Send_Command( NDis1); // 第 1 位数码管不显示
            Send_Command(Dis02); // 第 0 位数码管显示 2
            break;
        }
        case 0x43:// 按键 K3 按下
        {
            Send_Command( NDis1); // 第 1 位数码管不显示
            Send_Command(Dis03); // 第 0 位数码管显示 3
            break;
        }
        case 0x48:// 按键 K4 按下
        {
            Send_Command( NDis1); // 第 1 位数码管不显示

```

```

        Send_Command(Dis04); //第 0 位数码管显示 4
        break;
    }
case 0x49://按键 K5 按下
    {
        Send_Command( NDis1); //第 1 位数码管不显示
        Send_Command(Dis05); //第 0 位数码管显示 5
        break;
    }
case 0x4A://按键 K6 按下
    {
        Send_Command( NDis1); //第 1 位数码管不显示
        Send_Command(Dis06); //第 0 位数码管显示 6
        break;
    }
case 0x4B://按键 K7 按下
    {
        Send_Command( NDis1); //第 1 位数码管不显示
        Send_Command(Dis07); //第 0 位数码管显示 7
        break;
    }
case 0x50://按键 K8 按下
    {
        Send_Command( NDis1); //第 1 位数码管不显示
        Send_Command(Dis08); //第 0 位数码管显示 8
        break;
    }
case 0x51://按键 K9 按下
    {
        Send_Command( NDis1); //第 1 位数码管不显示
        Send_Command(Dis09); //第 0 位数码管显示 9
        break;
    }
case 0x52://按键 K10 按下
    {
        Send_Command(Dis00); //第 0 个数码管显示字符"0"
        Send_Command(Dis11); //第 1 个数码管显示字符"1"
        break;
    }
case 0x53://按键 K11 按下
    {
        Send_Command(Dis01); //第 0 个数码管显示字符"1"
        Send_Command(Dis11); //第 1 个数码管显示字符"1"
        break;
    }
case 0x58://按键 K12 按下
    {
        Send_Command(Dis02); //第 0 个数码管显示字符"2"
        Send_Command(Dis11); //第 1 个数码管显示字符"1"
        break;
    }
}

```

```

case 0x59://按键 K13 按下
{
    Send_Command(Dis03);//第 0 个数码管显示字符"3"
    Send_Command(Dis11);//第 1 个数码管显示字符"1"
    break;
}
case 0x5A://按键 K14 按下
{
    Send_Command(Dis04);//第 0 个数码管显示字符"4"
    Send_Command(Dis11);//第 1 个数码管显示字符"1"
    break;
}
case 0x5B://按键 K15 按下
{
    Send_Command(Dis05);//第 0 个数码管显示字符"5"
    Send_Command(Dis11);//第 1 个数码管显示字符"1"
    break;
}
default:break;
}
P2IFG&=~BIT0;
}

```

拓展：上述实验只需用到 2 段数码管，学者可自己拓展，编写一个简易计算器的程序，实现简单的“+”、“-”、“×”、“÷”的功能

5.1.4 点阵 LCD 显示器控制实验

1. 实验准备

实验系统标准板一块，LaunchPad 一块。

2. 实验目的

了解 LCD 的工作原理和操作方法，掌握一种单片机 I/O 模拟时序控制的外设方法。

3. 实验步骤

(1) 将 LaunchPad 和液晶模块插到标准版上，用跳线帽将液晶模块下方的 J1 和 J2 上的下方的 2 根插针连接，J3 上的靠上方 2 根插针连接。再它们右边的 XIN 和 XOUT 连接。

(2) 用 miniUSB 线将 LaunchPad 和电脑连接起来。

(3) 打开 CCS 中新建工程，将示例程序写入到 main.c 文件中。

(4) 在工具栏上如下图所示点击鼠标，开始硬件仿真。

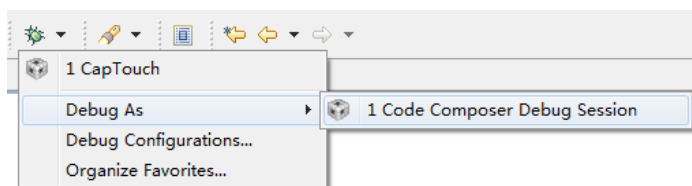


图 5.1.5 仿真步骤

(5) 点击“View”菜单下的“Expressions”，在出现的“Expression”标签中加入变量 display，查看 display 中数组各元素的值。

(6) 在点击下图按钮进入 main 函数中 while(1) 超循环, 不断的在 LCD 上显示 display 中的内容

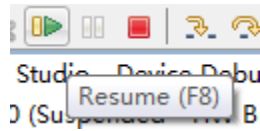


图 5.1.6 运行步骤

4. 程序示例

```
#include "msp430g2553.h"
#include "LCD12864.h"

unsigned char display[]={
    "====欢迎使用===="
    "西安电子科技大学"
    "实验系统标准板  "
    "TexasInstruments"
};

void main()
{
    WDTCTL = WDTPW + WDTHOLD; //关闭看门狗
    P2SEL &= ~(BIT6+BIT7);    //关闭 P2.6 和 P2.7 的第二功能
    P2DIR |= BIT6+BIT7;      //P2.6 和 P2.7 设置为输出
    lcd_init();              //初始化 LCD

    while(1)
    {
        chn_displ (display); //显示数组 display 中的内容
    }
}
```

5.1.5 触摸按键实验

1. 实验准备

实验系统标准板一块, LaunchPad 一块。

2. 实验目的

掌握一种触摸按键的工作原理, 和 IO 定时扫描的方法。

3. 实验步骤

- (1) 将 LaunchPad 插到标准版上, 用 miniUSB 线连接到电脑。
- (2) 打开 CCS 新建工程, 将示例程序写入到 main.c 文件中。
- (3) 在工具栏上如下图所示点击鼠标, 开始硬件仿真。

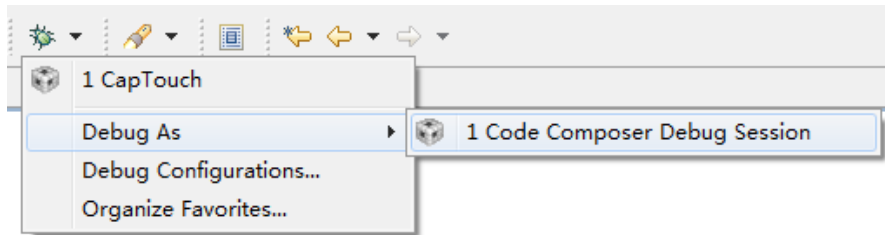


图 5.1.7 仿真步骤

(4) 点击“View”菜单下的“Expressions”，在出现的“Expression”标签中依次加入系统全集变量 base_cnt, meas_cnt, delta_cnt, key_pressed。

(5) 在 main 函数 while(1)死循环中的 delta_cnt = base_cnt - meas_cnt;语句处设置断点。

(6) 点击工具栏上如下图所示的“Resume”按钮，观察④中变量的变化。再将手指按在触摸键上(这里是向下的方向键)，重复观察。

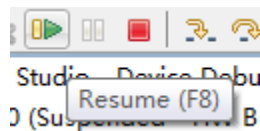


图 5.1.8 运行步骤

4. 程序示例

```
#include "msp430g2553.h"
/* 定义用户配置(根据具体的硬件平台可能要在相应修改) */
#define WDT_meas_setting (DIV_SMCLK_512)//定义看门狗定时器的测量时间间隔(时间较短)
#define WDT_delay_setting (DIV_ACLK_512)//定义看门狗定时器的测量间的延时(时间较长)
#define KEY_LVL      50                // 定义触摸板是否按下的阈值

/* 使用看门狗定时器可以使用的宏定义 */
#define DIV_ACLK_32768 (WDT_ADLY_1000)    // ACLK/32768
#define DIV_ACLK_8192  (WDT_ADLY_250)     // ACLK/8192
#define DIV_ACLK_512   (WDT_ADLY_16)      // ACLK/512
#define DIV_ACLK_64    (WDT_ADLY_1_9)     // ACLK/64
#define DIV_SMCLK_32768 (WDT_MDLY_32)      // SMCLK/32768
#define DIV_SMCLK_8192  (WDT_MDLY_8)       // SMCLK/8192
#define DIV_SMCLK_512   (WDT_MDLY_0_5)    // SMCLK/512
#define DIV_SMCLK_64    (WDT_MDLY_0_064)  // SMCLK/64

/* LED 相关定义 */
#define LED_1 (0x01)                    // P1.0 = LED1 输出
#define LED_2 (0x40)                    // P1.6 = LED2 输出

/* 用来测量的全局变量 */
unsigned int base_cnt, meas_cnt; //电容(实际是频率)测量的基准值和测量值
int delta_cnt;                  //电容(实际是频率)测量的基准值和测量值的差值
```

```

char key_pressed;           //状态量: 1 表示有按键按下; 0 表示没有按键按下
int cycles;
/* 系统功能函数 */
void measure_count(void);   // Measures each capacitive sensor
void pulse_LED(void);      // LED gradient routine
/* 主函数从这里开始 */
void main(void)
{
    unsigned int i, j;
    WDTCTL = WDTPW + WDTHOLD; // 停止看门狗定时器
    BCSCTL1 = CALBC1_1MHZ;    // 设置 DCO 为 1MHz
    DCOCTL = CALDCO_1MHZ;    // 设置 DCO 为 1MHz
    BCSCTL3 |= LFXT1S_2;     // LFXT1 为 VLO
    IE1 |= WDTIE;           // 使能看门狗定时器中断
    P2SEL = 0x00;           // 禁止外部晶振
    P1DIR = LED_1 + LED_2;  // P1.0 和 P1.6 设置为输出
    P1OUT = 0x00;
    __bis_SR_register(GIE); // 使能全局中断
    measure_count();        // 建立测量基准
    base_cnt = meas_cnt;
    for(i=15; i>0; i--)     // 重复测量来建立平均的基准值
    {
        measure_count();
        base_cnt = (meas_cnt+base_cnt)/2;
    }
    /* 主循环从这里开始 */
    while (1)
    {
        j = KEY_LVL;
        key_pressed = 0;    // 假设现在没有按键按下
        measure_count();    // 测量一次电容的大小
        delta_cnt = base_cnt - meas_cnt; // 计算差值: 指示电容的改变
        /* 为基本的电容下降 处理基准测量值 */
        if (delta_cnt < 0) // 如果为负说明结果超出了基准值, 例如, 电容下降
        {
            base_cnt = (base_cnt+meas_cnt) >> 1; // 快速的得到新的平均值
            delta_cnt = 0; // 差值清零
        }
        if (delta_cnt > j) // 根据预定的阈值检查是否有键按下
        {
            j = delta_cnt;
            key_pressed = 1; // 有按键按下
        }
    }
    else

```

```

    key_pressed = 0;
// 延时到下一次采样, 如果没有按键按下就采的更慢
if (key_pressed)
{
    BCSCTL1 = (BCSCTL1 & 0xCF) + DIVA_0;    // ACLK/(0:1, 1:2, 2:4, 3:8)
    cycles = 20;
}
else
{
    cycles--;
    if (cycles > 0)
        BCSCTL1 = (BCSCTL1 & 0xCF) + DIVA_0; // ACLK/(0:1, 1:2, 2:4, 3:8)
    else
    {
        BCSCTL1 = (BCSCTL1 & 0xCF) + DIVA_3; // ACLK/(0:1, 1:2, 2:4, 3:8)
        cycles = 0;
    }
}
}
WDTCTL = WDT_delay_setting; //采样间隔(较 WDT_meas_setting 长)
/* 为基本的电容上升 处理基准测量值*/
if (!key_pressed)          // 只当没有键按下时才缓慢的向下校准基准值
{
    base_cnt = base_cnt - 1; // 应该缓慢的减少基准来适应真实的电容变化
}
pulse_LED();              //根据按键是否按下调整 LED (有按键按下就点亮)
__bis_SR_register(LPM3_bits);
}
}                          // 主循环结束
/* 测量每一个按键的计数值(电容值) */
void measure_count(void)
{
    TAOCTL = TASSEL_3 + MC_2;    // TA 时钟源选择 IO 振荡器, 连续模式
    TAOCTL1 = CM_3+CCIS_2+CAP; // 上升沿和下降沿都捕获,
    /* 为弛张振荡器设置端口 */
    /* P2SEL2 寄存器允许 TA 从 GPIO 获取时钟 */
    P2DIR &= ~ BIT1;    // P2.1 设置为输入
    P2SEL &= ~ BIT1;    // P2.1 第二功能设置为电容感测
    P2SEL2 |= BIT1;     // P2.1 第二功能设置为电容感测
    /*Setup Gate Timer*/
    WDTCTL = WDT_meas_setting;    // 测量时间(较 WDT_delay_setting 短)
    TAOCTL |= TACLRL;             // 清除 TA 定时器
    __bis_SR_register(LPM0_bits+GIE); // 等待看门狗中断
    TAOCTL1 ^= CCIS0;            // 产生 CCR1 通道的软件捕获操作
    meas_cnt = TACCR1;          // 保存结果
}

```



```

    WDTCTL = WDTPW + WDTHOLD;           // 停止看门狗定时器
    P2SEL2 &= ~BIT1;
}
void pulse_LED(void)//LED 处理函数(按下则亮灯)
{
    if(key_pressed)
    {
        P1OUT ^= LED_1 + LED_2;
    }else{
        P1OUT = 0;
    }
}

/* 看门狗定时中断 */
#pragma vector=WDT_VECTOR
__interrupt void watchdog_timer(void)
{
    TAOCCTL1 ^= CCIS0;                 // 产生 CCR1 通道的软件捕获操作
    __bic_SR_register_on_exit(LPM3_bits); // 中断后推出睡眠
}

```

5.1.6 RGBLED 触控实验

1. 实验准备

实验系统标准板一块，LaunchPad，触摸 RGB_LED 模块。

2. 实验目的

掌握一种颜色可变的 LED 原理和控制方法，了解相关的光学知识。

3. 实验步骤

(1) 将 LaunchPad 和触摸 RGB_LED 模块插到标准版上，将模块上的开关 S1, S2, S3 都拨到远离触摸键的方向。

(2) 用 miniUSB 线连接 LaunchPad 和电脑。用手分别触摸 3 个触摸按键，观察 LED 灯的颜色。

(3) 打开 CCS 新建工程，将示例程序写入到 main.c 文件中。

(4) 在工具栏上如下图所示点击鼠标，开始硬件仿真。

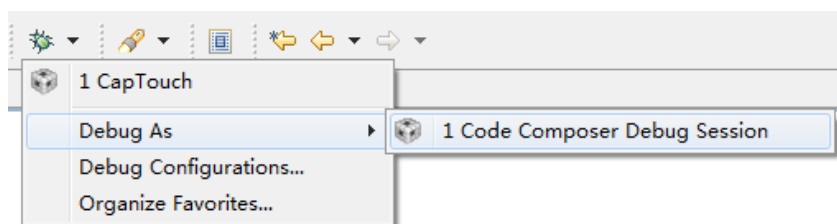


图 5.1.9 仿真步骤

(5) 点击工具栏上如下图所示的“Resume”按钮，将开关 S1, S2, S3 中的某一个拨到反方向，观察 LED 灯闪烁；接着触摸其他快关对应位置的触摸键，观察现象。

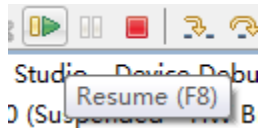


图 5.1.10 运行步骤

4. 程序示例

```

#include "msp430g2553.h"
void main( void )
{
    WDTCTL=WDTPW+WDTHOLD;          //关闭看门狗
    P1DIR |=BIT0+BIT1+BIT2+BIT3;    //GPIO 设置
    P1DIR |=BIT4+BIT5+BIT6;
    P1OUT&=~(BIT4+BIT5+BIT6);
    P1OUT &=~(BIT0+BIT1+BIT2+BIT3);
    P1OUT |=BIT0+BIT1+BIT2+BIT3;
    TAOCCR0=65500;                  //定时器 A 的设置
    TAOCCR1=13107;
    TAOCCR2=45875;
    TAOCTL |=MC_1+TASSEL_2+ID_3;
    TAOCTL |=TAIE;
    TAOCTL1 |=CCIE;
    TAOCTL2 |=CCIE;
    _EINT();
    while(1)
    {
    }
}
//定时器 A 中断函数
#pragma vector=TIMER0_A1_VECTOR
__interrupt void TA1_ISR(void)
{
    switch(TA0IV)
    {
    case 2: P1OUT&=~(BIT4+BIT5+BIT6);
            P1OUT |=BIT4;
            break;
    case 4:P1OUT&=~(BIT4+BIT5+BIT6);
            P1OUT |=BIT5;
            break;
    case 10:
            P1OUT&=~(BIT4+BIT5+BIT6);
            P1OUT |=BIT6;
            break;
    }
}

```

第二节 AD 转换基础实验

5.2.1 输入电压检测实验

1. 实验准备

3. 3V 直流电源一台，LaunchPad 实验系统一套，导线若干。

2. 实验目的

了解 AD 的工作原理和操作方法，熟悉模拟量输入测量的步骤。

3. 实验步骤

(1) 将 LaunchPad 正确插在实验底板上，并将直流电源的电压输出端接到单片机任一模拟输入通道 (A_0 — A_7)，本实验就选择 A_4 ；

(2) 理论估算：在选择 2.5V 为参考电压、10 位 ADC 的情况下，下表显示了模拟电压与 AD 转换后数字量的对照表（只给出 6 个典型值，用于参照）；

模拟电压/V	AD 转后对应数字量
0	0
0.5	205
1	409
1.5	614
2	818
2.5	1023

(3) 将该系统连接至电脑，并将下述程序实例通过 CCS 下载到单片机里并全速运行；

(4) 打开“watch”“register”，一边调节直流电源输出电压，一边观察寄存器窗口中 ADC10 转换后的结果 ADCMEM，并记录在下表，比较与第 2 步结果是否相符。

模拟电压/V	ADCMEN
0	
0.5	
1	
1.5	
2	
2.5	

4. 程序示例

```
#include "msp430g2553.h"
void ADC10_Init(void)
{
    ADC10CTL1=CONSEQ_2;//单通道多次转换，这句应当写在最前面
    ADC10CTL0 = REFON+SREF_1+REF2_5V; //打开 2.5V 正参考,地为负参考
    ADC10CTL0 |= ADC100N+ADC10SHT_3+ADC10IE;//打开 ADC10 内核,设定采样保持时间为 64 个
```

```

ADC10CLK, 使能 ADC10 中断
ADC10CTL1 = INCH_4+SHS_0+ADC10SSEL_2;           // input A4, 采样保持
ADC10AEO |= 0x10;                               // P1.4 DC option select。A4 模
拟信号输入使能
}
void main()
{
    WDTCTL = WDTPW + WDTHOLD;                    // Stop watchdog timer

    ADC10_Init();
    _BIS_SR(GIE); //开中断
    while(1);
}

```

5.2.2 光照度检测实验

1. 实验准备

LaunchPad 实验系统一套，光照度检测模块，LCD12864。

2. 实验目的

掌握光照度传感器的原理和使用方法。

3. 实验步骤

(1) 将 LaunchPad、LCD12864 和光照度检测模块正确插在实验底板上，光照度检测模块的电路如下图所示；

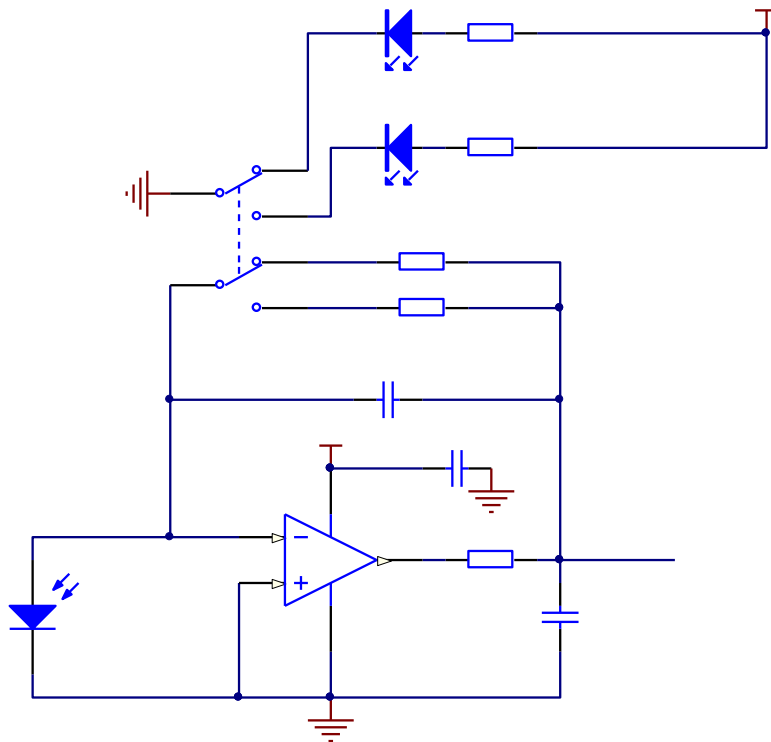


图 5.2.1 光超度检测原理图

- (2) 将下述程序通过 CCS 下载到单片机里并全速运行；
- (3) 通过不同程度的遮挡硅光电池，观察 LCD12864 示数的变化，通过光照度检测模块上自锁按键可以切换所采用的量程；
- (4) 将此模块测得的数据与标准照度计对比（没有条件的可以省略此步）。

4. 程序示例

```

//*****
*
//程序名称：光照度检测模块
//功能描述：光电池将光信号转化成电信号，然后输入到单片机模拟量输入端 A4(即 P1.4)，
//          经 AD 转换后，由单片机处理，将该信号转化成光照度，最后显示在 LCD12864
//          上。
//          MCLK 采用 1M 的内部 DC0，SMCLK 采用 125K 的时钟，参考电压选择内部 2.5V
//          正参考，
//          地为负参考。
//
//          MSP430G2553
//          +-----+
//          |                                     |
//          |                                     XOUT|--p2.7 (EN)
//          |                                     |
//          |                                     XIN|--p2.6 (R/W)
//          |                                     |
//          |                                     |
//          | 模拟输入--|P1.4
//          |                                     |
//          |                                     P2.4|--小量程
//          |                                     P2.3|--大量程
//          |                                     |
//          +-----+
//*****
*
#include "msp430g2553.h"
#include "LCD12864.h"
unsigned char const tab1[]={
    "
    "    LaunchPad
    "    Welcome
    "
};
unsigned char const tab2[]={
    "====欢迎使用===="
    "西安电子科技大学"
    " 光强检测模块  "

```

```

        "TexasInstruments"
};
unsigned char const tab3[]={
        " 您正在使用:  "
        "当前光照强度为: "
        "      量程      "
        "                Lx "
};
unsigned char CHN1[]="大",CHN2[]="小";
void IO_init()
{
    P2DIR&=~(BIT3+BIT4);
    P2REN|=BIT3+BIT4;
    P2OUT|=BIT3+BIT4;//内部上拉
    //P1REN|=BIT4;
    //P1OUT|=~BIT4;//内部下拉
}
void ADC10_Init(void)
{
    ADC10CTL1=CONSEQ_2;//单通道多次转换, 这句应当写在最前面
    ADC10CTL0 = REFON+SREF_1+REF2_5V; //打开 2.5V 正参考, 地为负参考, 默认 1.5V
    ADC10CTL0 |= ADC10ON+ADC10SHT_3+ADC10IE;//打开 ADC10 内核, 设定采样保持时间为 64
    个 ADC10CLK, 使能 ADC10 中断
    ADC10CTL1 = INCH_4+SHS_0+ADC10SSEL_2; // input A4, 采样
    保持
    ADC10AEO |= 0x10; // P1.4 DC option select. A4 模拟信
    号输入使能
}
int m,ave,n=0;
unsigned long AD_Result,Lx;
void main()
{
    WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer
    //BCSCTL1 = CALBC1_1MHZ;
    //DCOCTL = CALDCO_1MHZ;//上面两句将内部 DCO 校准至 1MHz
    //BCSCTL2 |= SELM_0;//MCLK 采用 1M 的内部 DCO
    //BCSCTL2 |= DIVS_0;//SMCLK 采用 125K 的时钟
    P2SEL &= ~(BIT6+BIT7);
    P2DIR |= BIT6+BIT7;
    led_init();
    ADC10_Init();
    IO_init();
    _BIS_SR(GIE);//开中断
}

```

```

chn_displ (tab1);
delay_ms(3000);
chn_displ (tab2);
delay_ms(3000);
chn_displ (tab3);
while(1)
{
    {
        Write_Num(0x98, Lx, 0);
        delay_ms(100);
        ADC10CTL0 |= ENC + ADC10SC;
    }
}
}
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR (void)
{
    ADC10CTL0 &= ~ENC;           // Disable ADC conversion
    m++;                          //计数变量加一，记录进入AD中断的次数
    AD_Result+=ADC10MEM;         //累加每次采样值
    if(m>=8)                      //判断采样的次数，若采样次数等于8，作下面处理
    {
        m=0;
        ave=AD_Result>>3;        //对累加和求平均
        AD_Result=0;
        if((P2IN&BIT3)==0x00) //切换量程
        {
            Lx=(long)((ave*2.5/1023*1000)/1.4); //大量程，将电压值换算成光强度
            write_oneChinese(2, 2, CHN1);
        }
        else
        {
            Lx=(long)((ave*2.5/1023*100)/1); //小量程
            write_oneChinese(2, 2, CHN2);
        }
    }
}
}

```

5.2.3 Pt100 温度测量实验

1. 实验准备

LaunchPad 实验系统一套，Pt100，三种温度传感器模块、LCD12864、LM385 一只、9012 一只、LM358 一个；电阻、电容、导线若干。

2. 实验目的

掌握 PT100 温度传感器的原理和使用方法。

3. 实验步骤

(1) 按照下述电路在面包板上搭建电路，本电路在三种温度传感器模块中有相应电路，读者也可直接使用该模块，下面各步骤以该模块为例来说明；

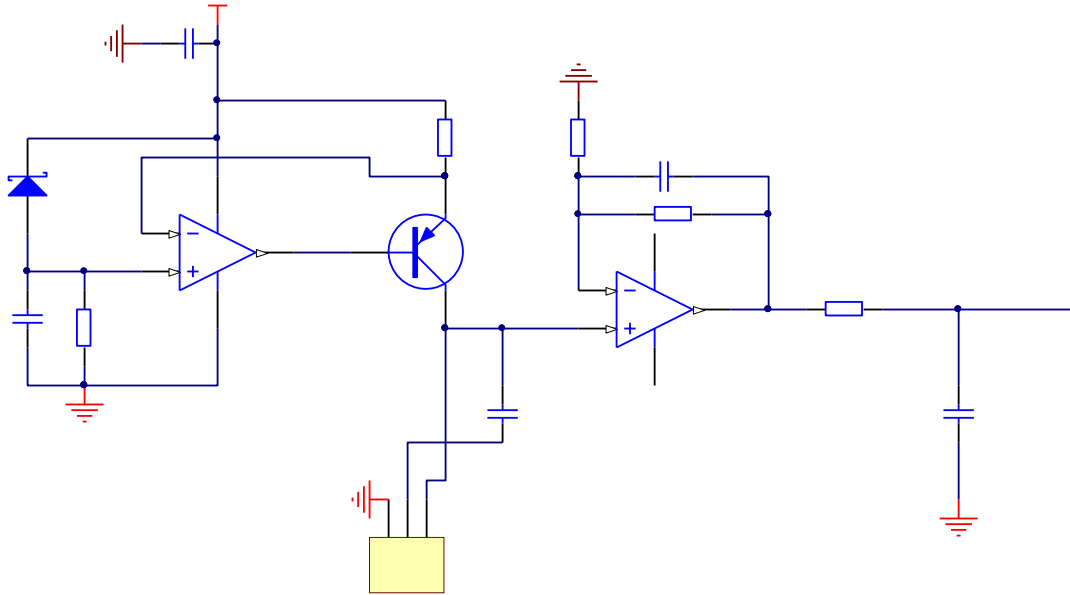


图 5.2.2 PT100 测温电路

- (2) 将 LaunchPad、LCD12864 和温度传感器综合试验模块正确插在实验底板上；
- (3) 将下述程序通过 CCS 下载到单片机里并全速运行；
- (4) 通过不同程度的加热 Pt100 探头，观察 LCD12864 示数的变化；
- (5) 将此模块测得的数据与标准温度计对比（没有条件的可以省略此步）。

4. 程序示例

```
#include "msp430g2553.h"
#include "LCD12864.h"
#define ADC_0 0
#define ADC_F 679
#define VCAL 1 //ADC 测电压校准参数, 1v

unsigned char m=0;
int ave, R_PT100, T_PT100, n=0;
unsigned long AD_Result;

unsigned char const tab1[]={
    "
    "
```



```

        "          °C"
        " Pt100 所测温度: "
        "          "
    };

/*****
* 名 称: Calculate(int ADC_Result)
* 功 能: 将计算采样值对应的电压
* 入口参数: ADC_Result, 采样值
* 出口参数: 无
*****/
float Calculate(int ADC_Result)
{
    float Vol;
    Vol=(float) (ADC_Result-ADC_0)*VCAL/(ADC_F-ADC_0);
    return(Vol);
}

void ADC10_Init(void)
{
    ADC10CTL1=CONSEQ_2;//单通道多次转换, 这句应当写在最前面
    ADC10CTL0 = REFON+SREF_1+REF2_5V; //打开 2.5V 正参考, 地为负参考, 默认 1.5V
    ADC10CTL0 |= ADC10ON+ADC10SHT_3+ADC10IE;//打开 ADC10 内核, 设定采样保持时间为 64
    个 ADC10CLK, 使能 ADC10 中断
    ADC10CTL1 = INCH_6+SHS_0+ADC10SSEL_2; // input A6, 采样保持
    ADC10AE0 |= 0x10; // P1.4 DC option select. A6 模拟信号输入
    使能
}

void main()
{
    WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer
    P2SEL &= ~(BIT6+BIT7);
    P2DIR |= BIT6+BIT7;
    lcd_init();
    ADC10_Init();
    _BIS_SR(GIE);//开中断
    chn_displ(tab1);
    while(1)
    {
        Write_Num(0x93, T_PT100, 0);
        delay_ms(100);
        ADC10CTL0 |= ENC + ADC10SC;
    }
}

#pragma vector=ADC10_VECTOR

```

```

__interrupt void ADC10_ISR (void)
{
    ADC10CTL0 &= ~ENC;           // Disable ADC conversion
    m++;                          //计数变量加一，记录进入 AD 中断的次数
    AD_Result+=ADC10MEM;         //累加每次采样值
    if(m>=8)                      //判断采样的次数，若采样次数等于 8 ，作下面处理
    {
        m=0;
        ave=AD_Result>>3;       //对累加和求平均
        AD_Result=0;
        R_PT100=Calculate(ave);
        R_PT100=R_PT100*1000/4.0;
        T_PT100=(int)((R_PT100/1.055-100)/0.03851);           //计算 PT100 所测温度
    }
}

```

5.2.4 声音强度检测实验

1. 实验准备

LaunchPad 实验系统一套，声音强度检测模块、LCD12864。

2. 实验目的

熟悉和掌握正负强度检测的方法和原理。

3. 实验步骤

(1) 将 LaunchPad、LCD12864 和声音强度检测模块正确插在实验底板上，声音强度检测模块的电路如下图所示；

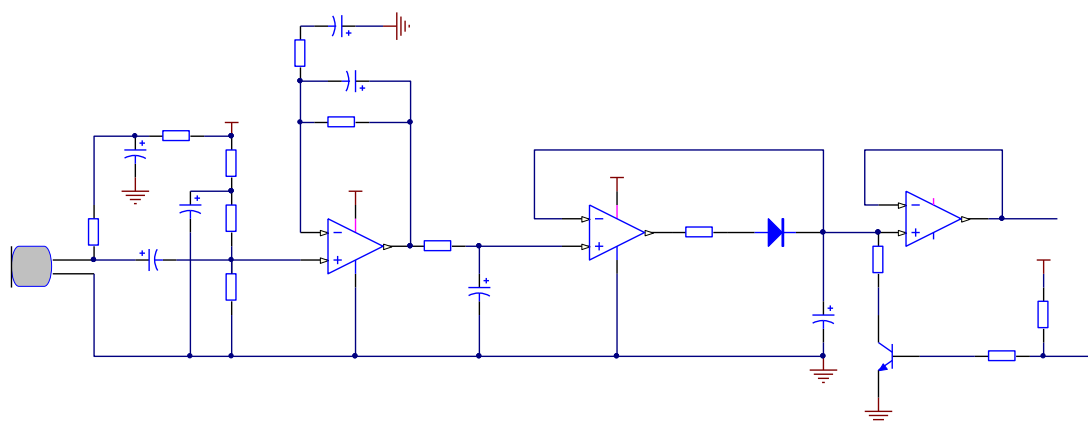


图 5.2.3 声音强度检测原理图

- (2) 将下述程序通过 CCS 下载到单片机里并全速运行；
- (3) 改变检测环境的声音强度，观察 LCD12864 示数的变化；
- (4) 将此模块测得的数据与标准声音强度测试计对比（没有条件的可以省略此步）。

4. 程序示例

```
#include "msp430g2553.h"
```

```

#include "math.h"
#include "LCD12864.h"

unsigned char const tab1[]={
    "
    "
    "    LaunchPad "
    "    Welcome  "
    "
};

unsigned char const tab2[]={
    "====欢迎使用===="
    "西安电子科技大学"
    " 声强检测模块  "
    "TexasInstruments"
};

unsigned char const tab3[]={
    "
    "
    "                dB "
    " 当前声强为:  "
    "
};

void IO_init()
{
    P2DIR|=BIT0;
}

void ADC10_Init(void)
{
    ADC10CTL1=CONSEQ_2;//单通道多次转换，这句应当写在最前面
    ADC10CTL0 = REFON+SREF_1+REF2_5V; //打开 2.5V 正参考,地为负参考,默认 1.5V
    ADC10CTL0 |= ADC100N+ADC10SHT_3+ADC10IE;//打开 ADC10 内核,设定采样保持时间为 64
    个 ADC10CLK,使能 ADC10 中断
    ADC10CTL1 = INCH_3+SHS_0+ADC10SSEL_2;    // input A3, 采样保持
    ADC10AEO |= 0x02;                        // PA.3 DC option select。A3 模拟信
    号输入使能
}

unsigned int m,ave;
unsigned long  AD_Result,sum=0;
double dB1;
long dB;
void main()
{
    WDTCTL = WDTPW + WDTHOLD;                // Stop watchdog timer
    BCSC1L1 = CALBC1_1MHZ;
    DCOCTL = CALDCO_1MHZ;//上面两句将内部 DCO 校准至 1MHz
}

```

```

BCSCTL2 |= SELM_0;//MCLK 采用 1M 的内部 DCO
BCSCTL2 |= DIVS_0;//SMCLK 采用 125K 的时钟
P2SEL &= ~(BIT6+BIT7);
P2DIR |= BIT6+BIT7;
lcd_init();
ADC10_Init();
IO_init();
_BIS_SR(GIE);//开中断
chn_displ (tab1);
delay_ms(3000);
chn_displ (tab2);
delay_ms(3000);
chn_displ (tab3);
while(1)
{
    Write_Num(0x88, dB, 0);
    delay_ms(50);
    ADC10CTL0 |= ENC + ADC10SC;
}
}
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR (void)
{
    P2OUT&=~BIT0;
    ADC10CTL0 &= ~ENC;           // Disable ADC conversion
    m++;                          //计数变量加一，记录进入 AD 中断的次数
    AD_Result+=ADC10MEM;         //累加每次采样值
    if(m>=8)                      //判断采样的次数，若采样次数等于 8 ，作下面处理
    {
        m=0;
        ave=AD_Result>>3;        //对累加和求平均
        AD_Result=0;
        dB1=ave*2.5/1023;
        dB=(long) (70+120*log10(dB1));
    }
    P2OUT|=BIT0;
}
}

```

第三节 D/A 转换基础实验

5.3.1 D/A 转换器件实现电压输出实验

1. 实验准备

LaunchPad 系统板，D/A 模块，万用表一部。

2. 实验目的

了解 D/A 转换器的工作原理和操作方法。

3. 实验步骤

(1) USB 线连接 PC 机和 LaunchPad 系统板，取出与系统配套的 D/A 模块，将其固定在系统板的模块扩展接口上；

(2) 用万用表检测 D/A 供电是否为 5V，开启液晶并检测其是否正常；

(3) 打开 CCS 添加 DAC8552 的初始化程序，并编写一段功能程序，让单片机分别输出 5 个 16 位数据到 DAC8552 的一个通道并进行 DA 转换；

(4) 根据公式 $V_{OUTA, B} = V_{REF} \times D / 65536$ ，计算出 DAC8552 的输出端的电压值 U_1 ，用万用表测得的输出端电压值 U_2 ，填入下表：

设定数据 D					
计算电压 U_1					
输出电压 U_2					
相对误差					

(5) 添加按键程序和 LCD 显示程序，设置开机输出为 0V，最大输出为 5V，切换到通道 A (B) 输出，按键 1 为幅值加 0.1，按键 2 为幅值减 0.1。每按键一次记录下 LCD 上的幅度值 U_1 ，以及万用表测得的 DAC8552 的输出端的电压值 U_2 ，填入下表：

LCD 上显示电压 U_1					
万用表测得电压 U_2					
相对误差					

4. DAC8552 程序示例

```
void dac_Init(void)
{
    P1OUT = 0x00;
    P1DIR |= BIT5 + BIT7;
    P2DIR |= BIT5; //SYNC 置为输出
    SYNC_H; //P2.5 做 SYNC
    P1SEL = BIT5 + BIT7; //P1.5、7 分别作 SCLK 和 SIMO
    P1SEL2 = BIT5 + BIT7;
    UCBOCTL1 |= UCSWRST; //禁止 USCI 状态机

    /* ~UCCKPH -- 数据在 UCLK 的第一个边沿变换，在下一个边沿捕获。
    * ~UCCKPL -- 空闲状态为低——这两个就决定数据在下降沿写出
    * UCMSB -- 最高位先出
    * ~UC7BIT -- 8 位数据
```

```

    * UCMST -- 主机模式
    * UCMODE_0 -- 3 线制 SPI
    * UCSYNC -- 同步模式*/
UCBOCTL0 = UCMSB + UCMST + UCMODE_0 + UCSYNC;//
UCBOCTL1 = UCSSEL_2 + UCSWRST;
UCBOBR0 = 1;
UCBOCTL1 &= ~UCSWRST;           //启动 USCI 状态机
}

```

5.3.2 PWM 实现电压输出实验

1. 实验准备

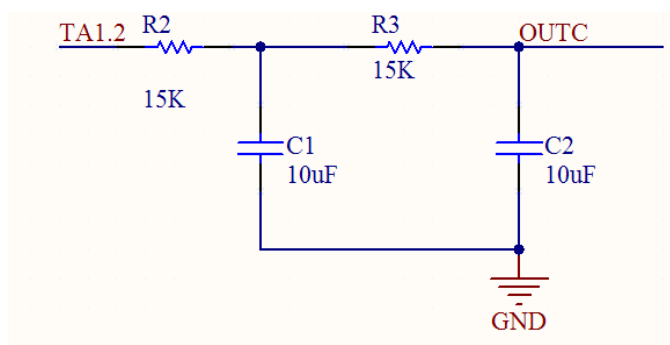
LaunchPad 系统板，15K 电阻，10uF 电容，万用表一部，导线若干。

2. 实验目的

了解 PWM 模拟 DA 的原理和实现方法。

3. 实验步骤

- (1) 用万用表电阻档检查电阻电容是否正常；
- (2) 按照下图搭建低通滤波器电路，并计算截止频率 f_0 ：



5.3.1 PWM 调理电路

- (3) 将 PWM 输出 P2.4 口接到 RC 网络的输入端；
- (4) 打开 CCS 对单片机进行仿真，先初始化时钟配置和 PWM 输出初始化配置，使得时钟输出频率为 f_1 ；
- (5) 将寄存器 TA1CCR0 写入数据，由公式 $f_2 = f_1 / TACCR0$ ，需要满足 f_2 大于 6~8 倍的 f_0 ；
- (6) 将寄存器 TA1CCR2 写入数据，由公式 $V = 3.3 * (TACCR2 / TACCR0)$ 计算输出电压值 U_1 ，万用表测得输出电压 U_2 ；
- (7) 改变 f_1 、TACCR0、TACCR2 的值，重复测量，填写下表：

f1				
TACCR0				
TACCR2				
U1				
U2				
相对误差				

4. 程序示例

```
void sys_Clock_Init()                //时钟初始化为 1MHz;
{
    BCSCTL1 = CALBC1_1MHZ;           // Set range
    DCOCTL = CALDCO_1MHZ;           // Set DCO step + modulation */
}

void PMW_OUT _Init ()                //PWM 输出初始化
{
    P2DIR |= BIT4;
    P2SEL |= BIT4;
    TA1CTL |= TASSEL_2+ID_0+MC_1;
    TA1CCTL2 |= OUTMOD_7;            //选择高电平模式的 PWM 输出
    TA1CCR0=2000;                    // TA1CCR0 写入 2000 (1MHZ/2000) 输出达到了 500HZ
}
```

第四节 定时/计数器基础实验

5.4.1 定时信号产生实验

1. 实验准备

示波器一台、LaunchPad 开发板一套（焊接 32.768KHz 的外部晶振）。

2. 实验目的

练习 MSP430G2553 单片机定时比较模式以及中断功能。

3. 实验步骤

- (1) 用短接帽将 LaunchPad 开发板上的 J5 上的 P1.0 短接；
 - (2) 将实验程序编辑在仿真软件上，编译并下载程序。
 - (3) 观察 LaunchPad 上 LED 的闪烁情况，并且用示波器观察 LED 闪烁的频率，观察 LED 的闪烁频率是不是 0.5Hz。
 - (4) 用示波器观察 P1.1 口方波频率，检查频率是不是 8.192KHz。
 - (5) 观察 P1.2 口的方波频率，P1.2 口的方波频率应该为 2.048KHz。
- （注意：在本程序中如果选用时钟源 SMCLK，由于内部时钟不稳定，所以测出来的方波频率可能有误差。）

4. 程序示例

```
/*
  在本程序中，使用了定时器的三种中断，事件 1、2 和 3。事件 2 是在 P1.1 口产生 8.192KHZ
  的方波，事件 1 是在
  P1.0 口产生 5HZ 的方波，事件 3 是在 P1.2 口产生 2.048KHZ 的方波。
  在本程序中，采用比较模式 4，这种模式是当计数器计至 CCRX 时，TAX 管脚取反。
  特别注意：在本程序中千万不能将两种中断混淆。
*/
#include <MSP430G2553.H>

int main( void )
{
    WDTCTL = WDTPW + WDTHOLD;           // 停止看门狗
    P1SEL |= 0x06;                       // P1.1 - P1.2 的功能选择
    P1DIR |= 0x07;                       // P1.0 - P1.2 方向
    CCTLO = OUTMOD_4 + CCIE;            // CCR0 选择比较模式 4，使能中断
    CCTL1 = OUTMOD_4 + CCIE;            // CCR1 选择比较模式 4，使能中断
    TACTL = TASSEL_1 + MC_2 + TAIE;     // 时钟源选择 ACLK, 计数模式选择连续
    计数模式，主计数器计满中断允许

    _BIS_SR(LPM3_bits + GIE);           // 进入低功耗模式 3，打开全局中断

    // Timer A0 interrupt service routine
    #pragma vector=TIMER0_A0_VECTOR
```



```

__interrupt void Timer_A0 (void)
{
    CCRO += 4;                //给 CCRO 赋初始值
}

// Timer_A2 Interrupt Vector (TA0IV) handler
#pragma vector=TIMER0_A1_VECTOR
__interrupt void Timer_A1(void)
{
    switch( TA0IV )
    {
        case 2: CCR1 += 16;    // 给 CCR1 赋初始值
            break;
        case 10: P1OUT ^= 0x01; // 主计数器计满溢出
            break;
    }
}
}

```

5.4.2 信号频率测量实验

1. 实验准备

信号源、示波器、LaunchPad 开发板一套。

2. 实验目的

- (1) 学习频率测量的两种方法,了解两种方法的差别以及各自的优点;
- (2) 加深对单片机定时器的了解及应用;
- (3) 学会使用定时器的中断。

3. 实验原理

本实验中,对信号频率的测量我们有两种方法:直接测频率和通过测周期计算频率。其中直接测量频率的方法适用于测量较高频率的信号;通过测周期计算频率适用于频率较低的信号。下面分别介绍两种测量频率方法的原理。

(1) 通过测周期计算频率

通过测量周期计算频率适用于测量频率较低的信号。具体原理是运用 TA 模块的捕获功能,在第一个上升沿到来的时候,开始计数,到下一次上升沿到来的时候停止计数,通过计数个数计算出高电平的时间。这个时间就是测量周期,取测量周期的倒数就是待测频率。通过测量周期进行低频率信号的测量具有得天独厚的优势,试想如果想要通过计数的方法测量 0.1HZ 的方波频率,那就需要等待至少 10s 才能测量出带测信号的频率,这种灵敏度是不可容忍的。

(2) 直接测频法

直接测频法有硬件测量和软件测量两种方式,但是无论是软件测量还是硬件测量都需要精准的 1S 钟定时中断,1S 的定时中断常用 TA 模块产生。思路是:将 DCO 校准到 1MHZ,DCO 是 1MHZ 的实质含义是一个脉冲花费 1us,如果有 100 万个脉冲到来则可认为到了 1s。以下是 1s 定时的产生方法:将 SMCLK (1MHZ) 输入 TA 模块,对输入时钟进行 4 分频,那么输入的时钟就等效于 250KHZ,将 TA 设置为增计数模式,TACCR0 设置为 10000,在 TA 溢出中断


```

    DCOCTL = 0x00;
    switch(FRQ)
    {
    case 1:    //1Mhz
    {
        BCSCTL1 = CALBC1_1MHZ;    // Set range
        DCOCTL = CALDCO_1MHZ;    // Set DCO step + modulation */
        break;
    }
    case 8:    //8Mhz
    {
        BCSCTL1 = CALBC1_8MHZ;    // Set range
        DCOCTL = CALDCO_8MHZ;    // Set DCO step + modulation */
        break;
    }
    case 12:   //12Mhz
    {
        BCSCTL1 = CALBC1_12MHZ;   // Set range
        DCOCTL = CALDCO_12MHZ;    // Set DCO step + modulation*/
        break;
    }
    case 16:   //16Mhz
    {
        BCSCTL1 = CALBC1_16MHZ;   // Set range
        DCOCTL = CALDCO_16MHZ;    // Set DCO step + modulation*/
        break;
    }
    default:break;
    }
}
BCSCTL1 |= XT2OFF + DIVA_0;
BCSCTL3 = XT2S_0 + LFXT1S_2 + XCAP_1;
}

```

```

void delayus(unsigned int us)
{
    unsigned int i;
    for(i=0;i<us;i++)
        __delay_cycles(16);
}

```

(3) D/A 程序

```

#include "DAC8552.h"
#include "sin_Tab.h"
#include "CLK.h"

```

```

#include "msp430g2553.h"

unsigned int frequency;
unsigned int time_us;
/*DAC8552——SPI 初始化*/
void dac_Init(void)
{
    P1OUT = 0x00;
    P1DIR |= BIT5 +BIT7;           //P1.4 做 SYNC
    SYNC_H;
    P1SEL = BIT5 + BIT7;           //P1.5、7 分别作 SCLK 和 SIMO
    P1SEL2 = BIT5 + BIT7;
    P2DIR |= BIT5;

    UCB0CTL1 |= UCSWRST;           //禁止 USCI 状态机

    /* ~UCCKPH -- 数据在 UCLK 的第一个边沿变换，在下一个边沿捕获。
     * ~UCCKPL -- 空闲状态为低——这两个就决定数据在下降沿写出
     * UCMSB -- 最高位先出
     * ~UC7BIT -- 8 位数据
     * UCMST -- 主机模式
     * UCMODE_0 -- 3 线制 SPI
     * UCSYNC -- 同步模式*/
    UCB0CTL0 = UCMSB + UCMST + UCMODE_0 + UCSYNC;//

    UCB0CTL1 = UCSSEL_2 + UCSWRST;

    UCB0BRO = 1;

    /* Enable USCI */
    UCB0CTL1 &= ~UCSWRST;         //启动 USCI 状态机
}

void out_v(char channl,float voltage)
{
    unsigned int buf;
    char high,low;
    buf=(voltage/4.95)*65535;
    low=buf&0xFF;
    high=buf>>8;
    switch(channl)
    {
        case 1:
    {

```

```

    SYNC_L;          //选通拉低
    UCB0TXBUF = 0X10; //向 DA 写入命令——通道 A, 立即转换
    while (!(IFG2 & UCB0TXIFG));
    UCB0TXBUF = high;    //写入高 8 位
    while (!(IFG2 & UCB0TXIFG));
    UCB0TXBUF = low;     //写入低 8 位
    while (!(IFG2 & UCB0TXIFG));
    SYNC_H;          //选通拉高
}break;
case 2:
{
    SYNC_L;          //选通拉低
    UCB0TXBUF = 0X24; //向 DA 写入命令——通道 B, 立即转换
    while (!(IFG2 & UCB0TXIFG));
    UCB0TXBUF = high;    //写入高 8 位
    while (!(IFG2 & UCB0TXIFG));
    UCB0TXBUF = low;     //写入低 8 位
    while (!(IFG2 & UCB0TXIFG));
    SYNC_H;          //选通拉高
}break;
case 3:
{
    TA1CCR2=(voltage/3.54)*2000;
}
}
}

```

5.4.3 模拟滤波器实验

1. 实验准备

示波器、信号源、LaunchPad 开发板一套，模拟滤波器模块。

2. 实验目的

- (1) 了解滤波器的原理并学会设计简单的滤波器；
- (2) 利用滤波器的选频特性，完成对输入信号的整形和频率的测量。

3. 实验原理：

滤波器是一种选频装置，可以使信号中特定的频率成分通过，而极大地衰减其它频率成分。在测试装置中，利用滤波器的这种选频作用，可以滤除干扰噪声或进行频谱分析。

在本实验装置中有两种滤波器：带通滤波器和低通滤波器。其原理分别如下所述。

(1) 带通滤波器

在本实验模块中，带通滤波器的设计采用的是无限增益多路反馈型（MFB）。其具体的电路如下所示：

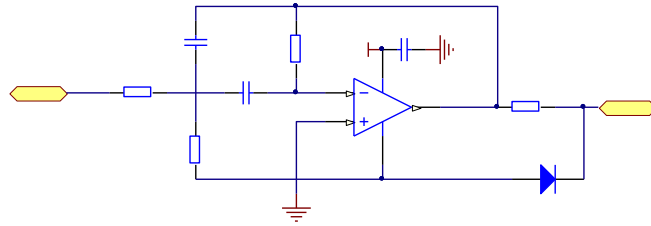


图 5.4.2 二阶有源带通滤波器连接电路图

在图中 $C_1=C_2=C$ ，其传递函数为 $H(s) = \frac{K_p Bs}{s^2 + Bs + \omega_0^2}$ (其中 $B = \frac{2}{R_3 C}$, $\omega_0^2 = \frac{1}{R_3 C^2} (\frac{1}{R_1} + \frac{1}{R_2})$, $K_p = -\frac{R_3}{2R}$)

表征带通滤波器性质的参数主要有三个，它们分别是：

中心频率：也叫谐振频率，带通滤波器在中心频率处转移函数的幅值最大，计算公式为 $\omega_0 = \frac{1}{\sqrt{RC}}$ ；

带宽：是两个截止频率之差，截止频率是转移函数的幅值由最大值下降为最大值的 $1/\sqrt{2}$ 时的频率，即 $|H(j\omega_c)| = \frac{1}{\sqrt{2}} H_{max}$ 。带宽的计算公式是 $\beta = \omega_2 - \omega_1 = \frac{R}{L}$ 。

品质因数：中心频率与带宽之比。计算公式为 $Q = \frac{\omega_0}{\beta} = \frac{\omega_0 L}{R} = \frac{1}{\omega_0 CR}$ 。

(2) 低通滤波器

在本实验中低通滤波器的设计采用压控电压源型 (VCVS)。所谓压控电源型是指同相输入，输入阻抗很高，输出阻抗很低。其中低通滤波器的电路图如图 5.XX 所示。在图中由于 C14 接到集成运放的输出端，形成正反馈，使电压放大倍数在一定程度上受输出电压控制，且输出电压近似为恒压源，所以称之为二阶压控电压源低通滤波器。其优点是电路性能稳定、增益容易调节。

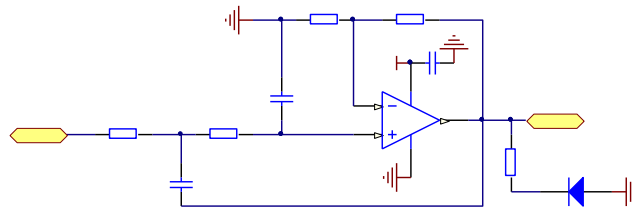


图 5.4.3 二阶有源低通滤波器连接电路图

在电路中通常取： $C_8=C_{14}=C$, $R_{17}=R_{18}=R$

幅频特性：

$$\begin{cases} -\frac{1}{R} \dot{V}_i + (\frac{2}{R} + CS) \dot{V}_1 - SCV_o - \frac{1}{R} \frac{\dot{V}_o}{H_0} = 0 \\ -\frac{1}{R} \dot{V}_1 + (\frac{1}{R} + CS) \frac{\dot{V}_o}{H_0} = 0 \end{cases}$$

传递函数:

$$H(S) = \frac{V_o(S)}{V_i(S)} = \frac{H_o}{1 + (3 - H_o)RCS + (RCS)^2}$$

增益为:

$$H_o = \frac{R_{15} + R_{16}}{R_{15}}$$

滤波器的低通截止频率为:

$$\omega_0 = \frac{1}{RC}$$

(3) 滤波器实验的整体电路图

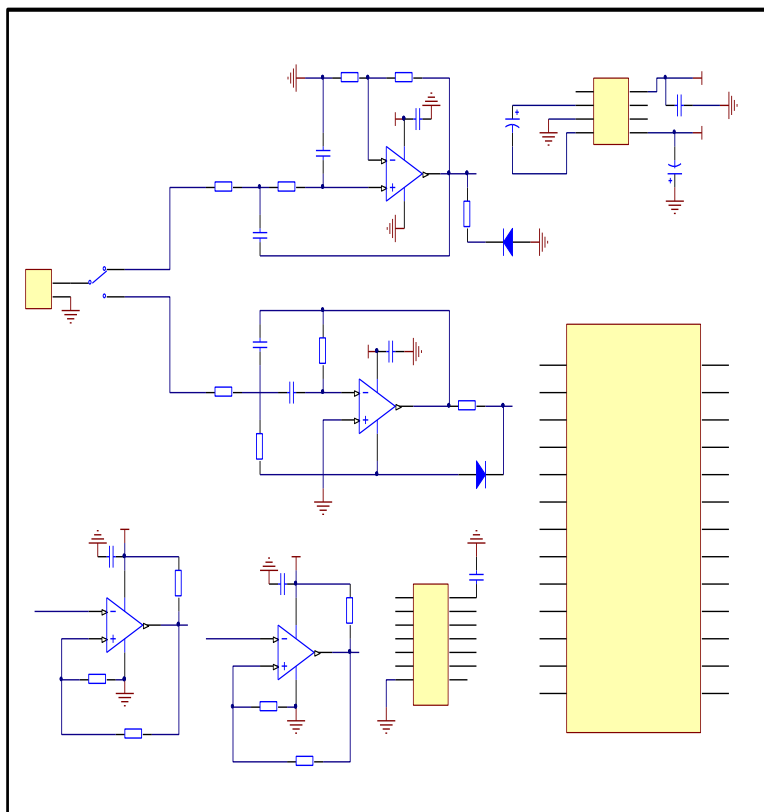


图 5.4.4 模拟滤波器模块电路原理图

4. 实验步骤

- (1) 根据实验原理连接电路并检查电路是否正确。
- (2) 将下列例程编辑在仿真软件中，编译并下载程序。
- (3) 先将实验电路调到低通滤波器的状态，并用信号源产生高频信号（频率大于低通滤波器的最高频率），观察输出情况，可以观察到输出信号基本没有，即使有信号的幅度也很低。
- (4) 将实验电路调节到带通滤波器的状态，然后用信号源产生方波输入到电路中，观察输出情况。
- (5) 同样是在带通滤波器的状态，用信号源产生正弦波输入到电路中，观察输出情况。
- (6) 总结实验结果。

4. 程序示例

```
#include "msp430g2433.h"  
#include "LCD12864.h"
```

```

unsigned int TA_OverflowCntdi;    // TA 溢出次数存放变量（低通）
unsigned int TA_OverflowCntdai;  // TA 溢出次数存放变量（带通）
unsigned long int Perioddi;      // 周期测量结果存放变量（低通）
unsigned long int Perioddai;    // 周期测量结果存放变量（带通）
unsigned int PervCapValdi;      // 前一次捕获值存放变量（低通）
unsigned int PervCapValdai;    // 前一次捕获值存放变量（带通）
unsigned int Pulse_Cntdi=0;     // 脉冲个数（低通）
unsigned int Pulse_Cntdai=0;   // 脉冲个数（带通）
unsigned int Flagdi=0;          // 低通标志
unsigned int Flagdai=0;        // 带通标志
unsigned int timers;            // 液晶显示汉字的频率设置
unsigned char const tab0[]={
    " 低通滤波器    " //第一行
    "          Hz  " //第三行
    "   频率值     " //第二行
    "              " //第四行
};

unsigned char const tab1[]={
    " 带通滤波器    " //第一行
    "          Hz  " //第三行
    "   频率值     " //第二行
    "              " //第四行
};

unsigned char const tab4[]={
    " 欢迎使用      " //第一行
    " 滤波器模块    " //第三行
    "              " //第二行
    "              " //第四行
};

//函数声明
/*****
* 名    称: FrequenceSet ()
* 功    能: 频率校准为 1MHZ
* 入口参数: 无
* 出口参数: 无
*****/
void FrequenceSet(void)
{
    if (CALBC1_1MHZ ==0xFF || CALDCO_1MHZ == 0xFF)
        FaultRoutine();           // If calibration data is erased
                                   // run FaultRoutine()
    BCCTL1 = CALBC1_1MHZ;         // Set range

```



```

    DCOCTL = CALDCO_1MHZ;           // Set DCO step + modulation
    BCSCTL3 |= LFXT1S_2;           // LFXT1 = VLO
    IFG1 &= ~OFIFG;               // Clear OSCFault flag
    BCSCTL2 |= SELM_0 ;//+ DIVM_3 + DIVS_3; // MCLK = SMCLK = 1M
}
/*****
* 名 称: TA1_Init()
* 功 能: TIMER_A 初始化, 捕获模块 1 启动, 选择 TA1 (P2. 1) 引脚作为捕获源, 上升沿捕获, 同步模式, 开启捕获中断
捕获模块 0 启动, 选择 TA1 (P2. 0) 引脚作为捕获源, 上升沿捕获, 同步模式, 开启捕获中断
* 入口参数: 无
* 出口参数: 无
*****/

void TA1_Init(void)
{
    TA1CTL = TASSEL_2 + MC_2 + TAIE + TACLR; // TA 连续计数, 开始计时, SMCLK, 开中断
    TA1CCTL1 = CAP + CM_1 + CCIS_0 + SCS + CCIE;//

    //捕获模块 1 启动, 选择 TA1 (P2. 1) 引脚作为捕获源, 上升沿捕获, 同步模式, 开启捕获中断
    TA1CCTL0 = CAP + CM_1 + CCIS_0 + SCS + CCIE;
    //捕获模块 0 启动, 选择 TA1 (P2. 0) 引脚作为捕获源, 上升沿捕获, 同步模式, 开启捕获中断
    TAOCCTL0|=CCIE;
    TAOCTL|=TASSEL_2+MC_2;//时钟源 SMCLK; 连续奇数模式
    TAOCCR0=65535;
}
/*****
* 名 称: IO_Init()
* 功 能: IO 口初始化, 选择 P2. 0, P2. 1 的第二功能; P2. 6, P2. 7 为输入
* 入口参数: 无
* 出口参数: 无
*****/

void IO_Init(void)
{
    P2DIR&=~(BIT1+BIT0); //TA1.1 CCI1A P2.1 TA1.0 CCI0A P2.0
    P2SEL |=BIT1+BIT0;
    P2SEL &=~(BIT6+BIT7);
    P2DIR |= BIT6+BIT7;
    P1DIR |= BIT2+BIT1;
}

void main( void )
{

```

```

WDICTL = WDTPW + WDTLHOLD;
FrequenceSet();
TA1_Init();
IO_Init();
lcd_init ();
chn_displ(tab4);           //开机界面显示
_EINT();                  // 总中断允许
while(1)
{
    if(Flagdi>=100)
    {
        TA1CTL|=TACLR;           //TA 计数清零
        TA1CCR1=0;
        chn_displ(tab0);
        Write_Num(0x88, Pulse_Cntdi, 2);    //显示低通频率值
        Flagdi=0;
    }
    if(Flagdai>=100)
    {
        TA1CTL|=TACLR;
        TA1CCR0=0;
        chn_displ(tab1);

        Write_Num(0x88, Pulse_Cntdai, 2);    //显示带通频率值
        Flagdai=0;
    }

}
}
/*****
功    能： TA 中断，被测信号的两次相邻上升沿都到达时，TA0 中断一次，并计算频率值
*****/

#pragma vector=TIMER1_A1_VECTOR
__interrupt void TA_ISR(void)           //带通中断源
{
    switch(TA1IV)
    {

        case 2:
            Flagdai++;
            Perioddai=TA_OverflowCntdai*65536 + TA1CCR1 - PervCapValdai;//计
算周期
            PervCapValdai=TA1CCR1;           //保存捕获值，供下一次使用

```

```

        TA_OverflowCntdai=0;           //溢出次数清零
        Pulse_Cntdai=1000000/Perioddai+4;
        break;
case 10:  TA_OverflowCntdai++;         //TA 每次溢出，溢出次数变量+1
        break;

    }
}
#pragma vector=TIMER1_A0_VECTOR
__interrupt void TA1_ISR(void)        //低通中断源
{
    Flagdi++;
    Perioddi=TA_OverflowCntdi*65536 + TA1CCR0 - PervCapValdi;//计算周期
    PervCapValdi=TA1CCR0;             //保存捕获值，供下一次使用
    TA_OverflowCntdi=0;               //溢出次数清零
    Pulse_Cntdi=1000000/Perioddi+3;   //+3 是频率补偿
}

```

该实例程序的性能和指标为（假定系统时钟没有误差 =1MHz）:

- (1) 频率测量绝对误差：±1Hz。
- (2) 被测最高频率值：2.5KHz。
- (3) 测量频度：1次/秒。
- (4) 使用资源：两个定时器，两个中断。

第五节 通信接口基础实验

5.5.1 SPI 接口基础实验

1. 实验准备

LaunchPad 实验系统，导线若干。

2. 实验目的

了解 SPI 通信的含义和基本实现方法。

3. 实验步骤

(1) 使用面包板的导线将两个实验系统的 P1.2、P1.4、P1.5、GND 相连。即系统左边插槽的左边圆针插排从上向下数第 1、5、7、8 的圆孔，同时将这些 I/O 口的开关打开。

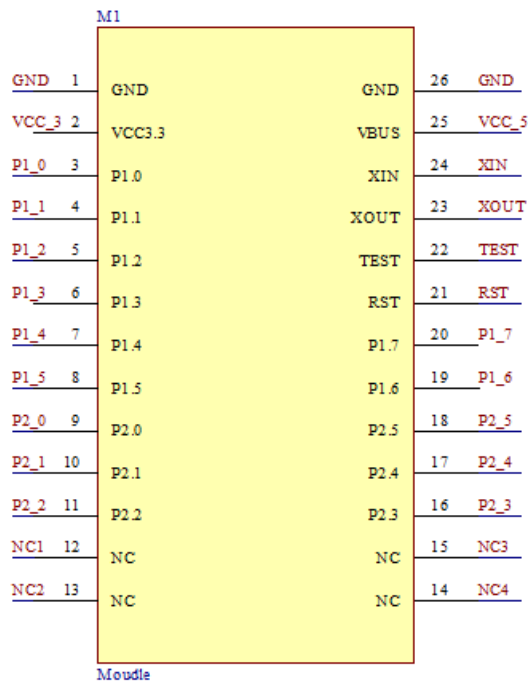


图 5.5.1 模块接口一览

- (2) 将两个试验系统都连接到电脑。
- (3) 打开 CCS 编写两个实验系统中 G2 单片机的程序。
- (4) 对单片机仿真，实现主机发送程序，从机显示主机发送程序的数据。

4. 程序示例

主机程序：

```
#include "msp430g2433.h"
#include "SPI.h" //串口通讯程序库头文件

unsigned char TXBuff[5]={0x50, 0x40, 0x30, 0x20, 0x10};
#define TXBUF_SIZE 32
#define FRAME_SIZE 5 /*定长帧的长度*/
```

```

unsigned char TX_BUFF[TXBUF_SIZE];
unsigned char RX_BUFF[FRAME_SIZE];
unsigned char SPI_RcvCnt=0; //接收计数

void SPI_M_Init(void)
{

    UCAOCTL1 |= UCSWRST;
    // UCA0STAT |= UCLISTEN;
    UCAOCTL1 |= UCSSEL1; //ACLK
    UCAOCTL0 |= UCMST + UCSYNC+UCMSB; // 8-bit SPI Master **SWRST**
MSB 起始
    UCAOBR0 = 0x02;
    UCAOBR1 = 0x00;
    UCAOMCTL = 0x00; // no modulation
    UCAOCTL0 |= UCMODE_0; // 3-pin mode
    UCAOCTL0 |= UCCKPH;
    UCAOCTL0 &=~ UCCKPL; //下降沿发送, 上升沿接收
    UCAOCTL1 &=~ UCSWRST; // Initalize USART state machine

    P1SEL |= BIT4;
    P1SEL2|= BIT4;
    P1DIR |= BIT4; //UCA0CLK
    P1SEL |= BIT2;
    P1SEL2|= BIT2;
    P1DIR |= BIT2; //SIMO
    P1SEL |= BIT1;
    P1SEL2|= BIT1;
    P1DIR &=~ BIT1; //SOMI

    IE2 |=UCAORXIE;
}

//串口发送函数
void SpiWriteDat(unsigned char *Ptr,unsigned int Lenth)
{
    int i;

    for(i=0;i<Lenth;i++)
    {
        UCA0TXBUF=Ptr[i];
        while ((IFG2&UCA0TXIFG)==0); // 等待上一字节发完
        IFG2&=~UCA0TXIFG;
    }
}

```

```

}

//串口接收中断函数
#pragma vector=USCIABORX_VECTOR
__interrupt void UartR0()

{

    RX_BUFF[SPI_RcvCnt]=UCA0RXBUF; //接收该字节数据
    SPI_RcvCnt++;                //指向下一单元
    if(SPI_RcvCnt>=FRAME_SIZE)
    {
        SPI_RcvCnt=0;          //接收数组清零
    }

}

void main(void)
{
    WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer to prevent time out reset
    BCSCTL1 = CALBC1_1MHZ;    // Set range
    DCOCTL = CALDCO_1MHZ;
    BCSCTL2 &= ~(DIVS_3);    // SMCLK = DCO / 8 = 1MHz
    SPI_M_Init();
    _EINT();

    while(1)                //串口测试
    {

        SpiWriteDat(TXBuff, 5);
        // __delay_cycles(1000000);

    }
}
}

从机程序:
#include "msp430g2433.h"
#include "SPI.h"           //串口通讯程序库头文件

#define TXBUF_SIZE 32
#define FRAME_SIZE 5     /*定长帧的长度*/

unsigned char TX_BUFF[TXBUF_SIZE];
unsigned char RX_BUFF[FRAME_SIZE];
unsigned char SPI_RcvCnt=0; //接收计数

```

```

void SPI_S_Init(void)
{

    UCAOCTL1 |= UCSWRST;
    UCAOCTL0 |= UCMSB + UCSYNC;           // 8-bit SPI Slave **SWRST**
    UCAOCTL0 |= UCMODE_0;                 // 3-pin mode
    UCAOCTL0 |= UCCKPH;
    UCAOCTL0 &=~ UCCKPL;                  //下降沿发送，上升沿接收
    UCAOCTL1 &=~ UCSWRST;                 // Initalize USART state machine

    P1SEL |= BIT4;
    P1SEL2|= BIT4;
    P1DIR &=~ BIT4;                       //UCA0CLK
    P1SEL |= BIT2;
    P1SEL2|= BIT2;
    P1DIR &=~ BIT2;                       //SIMO
    P1SEL |= BIT1;
    P1SEL2|= BIT1;
    P1DIR |= BIT1;                       //SOMI

    IE2   |=UCAORXIE;
}

//串口发送函数
void SpiWriteDat(unsigned char *Ptr,unsigned int Lenth)
{
    int i;

    for(i=0;i<Lenth;i++)
    {
        UCA0TXBUF=Ptr[i];
        while ((IFG2&UCA0TXIFG)==0); // 等待上一字节发完
        IFG2&=~UCA0TXIFG;
    }
}

//串口接收中断函数
#pragma vector=USCIAB0RX_VECTOR
__interrupt void UartR0()

{

    RX_BUFF[SPI_RcvCnt]=UCAORXBUF; //接收该字节数据
    SPI_RcvCnt++;                 //指向下一单元
}

```

```

        if(SPI_RcvCnt>=FRAME_SIZE)
        {
            SPI_RcvCnt=0;                //接收数组清零
        }
        // __delay_cycles(1000);
    }
    unsigned char TXBuff[5]={0xFF, 0x20, 0x30, 0x40, 0x50};

void main( void )
{
    // Stop watchdog timer to prevent time out reset
    WDTCTL = WDTPW + WDTHOLD;
    SPI_S_Init();
    _EINT();

    while(1)                            //串口测试
    {
        SpiWriteDat(TXBuff, 5);
        // __delay_cycles(1000000);
    }
}

```

5.5.2 程控放大器实验

1. 实验准备

LaunchPad 系统板，程控放大器模块，20M 示波器一台。

2. 实验目的

了解程控放大器的工作原理，单片机以 SPI 方式控制外设的步骤。

3. 实验步骤

- (1) 将模块插入试验系统左侧插槽中，同时打开旁边的 DIP 开关
- (2) 将单片机连接到电脑，并打开 CCS 对单片机仿真
- (3) 用示波器查看信号的放大倍数，一通道接到信号的输入端，二通道接到信号的输出端。
- (4) 调节拨码开关实现对放大倍数的调节。

拨码开关状态与放大倍数选择的关系表

放大倍数	DIP-4 状态	DIP-3 状态	DIP-2 状态	DIP-1 状态
1	ON	OFF	OFF	OFF
2	ON	OFF	OFF	ON
4	ON	OFF	ON	OFF
8	ON	OFF	ON	ON
16	ON	ON	OFF	OFF
32	ON	ON	OFF	ON
64	ON	ON	ON	OFF
128	ON	ON	ON	ON

(5) 再次用示波器查看原始波形和放大之后的波形。

4. 程序示例

```
#include "msp430g2433.h"
#include "spi.h"
#include "LCD.h"
#include "AD.h"
#include "key.h"
#include "TA.h"

//unsigned int ADC10_Result;
unsigned long int Voltage0;
unsigned long int Voltage1;
unsigned char flag=0;

unsigned char W0Buff[2]={0x2A, 0x01}; //1, 通道 1
unsigned char W1Buff[2]={0x2A, 0x11}; //2, 通道 1
unsigned char W2Buff[2]={0x2A, 0x21}; //4, 通道 1
unsigned char W3Buff[2]={0x2A, 0x31}; //8, 通道 1
unsigned char W4Buff[2]={0x2A, 0x41}; //16, 通道 1
unsigned char W5Buff[2]={0x2A, 0x51}; //32, 通道 1
unsigned char W6Buff[2]={0x2A, 0x61}; //64, 通道 1
unsigned char W7Buff[2]={0x2A, 0x71}; //128, 通道 1

volatile unsigned int num=1;
volatile unsigned int val0;
volatile unsigned int val1;
#define TXBUF_SIZE 32
#define FRAME_SIZE 5 /*定长帧的长度*/

unsigned char TX_BUFF[TXBUF_SIZE];

unsigned char RX_BUFF[FRAME_SIZE];
unsigned char SPI_RcvCnt=0; //接收计数
unsigned char FrameRcvStatus=0; /*帧接收状态*/

void SPI_M_Init(void)
{
    UCAOCTL1 |= UCSWRST;
    // UOCTL |= LISTEN;
    UCAOCTL1 |= UCSSEL1; //ACLK
```

```

        UCAOCTL0 |= UCMST + UCSYNC+UCMSB;           // 8-bit SPI Master
**SWRST** MSB 起始
        UCAOBRO = 0x02;
        UCAOBR1 = 0x00;
        UCAOMCTL = 0x00;                           // no modulation
        UCAOCTL0 |= UCMODE0;                         // 3-pin mode
        UCAOCTL0 |= UCCKPH;
        UCAOCTL0 &= ~ UCCKPL;                       //下降沿发送, 上升沿接收
//    ME2 |= USPIE0;                                // Enable USART0 SPI mode
        UCAOCTL1 &= ~UCSWRST;                       // Initialize USART state
machine

```

```

        P1SEL |= BIT4;
        P1SEL2 |= BIT4;
        P1DIR |= BIT4;                              //UCAOCLK
        P1SEL |= BIT2;
        P1SEL2 |= BIT2;
        P1DIR |= BIT2;                              //SIMO
//    P1SEL |= BIT5;
//    P1SEL2 |= BIT5;
//    P1DIR |= BIT5;

```

```

//    IFG1 &= ~URXIFG0;
    IE2 |=UCAORXIE;
}

```

//串口发送函数

```

void SpiWriteDat(unsigned char *Ptr,unsigned int Lenth)
{
    int i;

    for(i=0;i<Lenth;i++)
    {
        UCA0TXBUF=Ptr[i];
        while ((IFG2&UCA0TXIFG)==0); // 等待上一字节发完
        IFG2&=~UCA0TXIFG;
    }
}

```

//串口接收中断函数

```

#pragma vector=USCIABORX_VECTOR
__interrupt void UartR0()

{

```

```

        RX_BUFF[SPI_RcvCnt]=UCAORXBUF; //接收该字节数据
        SPI_RcvCnt++; //指向下一单元
        if(SPI_RcvCnt>=FRAME_SIZE)
        {
            SPI_RcvCnt=0; //接收数组清零
        }
    }

    unsigned int ADC10_Result0;
    unsigned int ADC10_Result1;
    volatile unsigned long int ADC_Result[1];
    volatile unsigned long int ADC_Result0;
    volatile unsigned long int ADC_Result1;
    //unsigned long int Voltage0;
    unsigned char ADC10_Flag=0;

    unsigned char i=1;
    unsigned int j=0;

    void Init_ADC10(void)
    {
        ADC10CTL0 |= ADC10SR+REFON+REF2_5V+ADC10ON+MSC;
        ADC10CTL0 |= ADC10SHT_3+SREF_1;
        ADC10CTL1 |= ADC10SSEL_3+ADC10DIV_1+INCH_1+SHS_0+CONSEQ_3;
//OSC, 无分频
        ADC10AEO |= BIT0+BIT1;
    }

    void ADC10_Sample(unsigned int AverageNum)
    {
        unsigned long int ADC10_Sum0=0;
        unsigned long int ADC10_Sum1=0;
        unsigned int i;
        ADC10CTL0 |= ADC10IE;
        ADC10CTL0 |= ADC10SC+ENC;
        for(i=0;i<AverageNum;i++)
        {
            while((ADC10_Flag)==0);
            ADC10_Flag=0;
            ADC10_Sum0+=(int)ADC_Result0;
            ADC10_Sum1+=(int)ADC_Result1;
        }
    }

```

```

        ADC10CTL0 &=~ ENC;
        ADC10CTL0 &=~ ADC10IE;
        ADC10_Result0=ADC10_Sum0/AverageNum;
        ADC10_Result1=ADC10_Sum1/AverageNum;
    }

#pragma vector=ADC10_VECTOR
__interrupt void ADC10ISR(void)
{
    ADC_Result[i]=ADC10MEM;
    i--;
    if(i>2)
    {
        i=1;
        ADC_Result1=ADC_Result[1];
        ADC_Result0=ADC_Result[0];
        ADC10_Flag=1;
    }
}

//延时函数
void Delay(int n)//延时，时间为 n
{
    int i;

    for(i=0;i<n;i++);//循环 n 次
}

//初始化 4 线工作模式函数
void Init_4lines_Mode(void)//初始化 4 线工作模式
{
    //初始化 P5 端口, P5. 5 接 CH452 的 DCLK 引脚, P5. 6 接 CH452 的 DIN 引脚, P5. 7 接
    CH452 的 LOAD 引脚
    P2DIR|=BIT1+BIT2+BIT3;
    P2DIR&=~BIT0;

    //初始化 DCLK、DIN 和 LOAD
    DCLK_1;
    DIN_1;
    LOAD_1;
}

//发送命令

```

```
void Send_Command(unsigned cmd)//MSP430 单片机向 CH452 发送命令，Command 为 12
位数据
```

```
{
    unsigned char i;
    LOAD_0;
    for(i=0;i!=12;i++)          //送入 12 位数据，低位在前
    {
        if (cmd&1) {DIN_1;}
        else {DIN_0;} // 输出位数据
        DCLK_0 ;
        cmd>>=1;
        DCLK_1 ;          //上升沿有效
    }
    LOAD_1;
    Delay(1000);
}
```

```
//输入按键代码子程序，从 CH452 读取
```

```
unsigned char Key_Read( void )
```

```
{
    unsigned char i;
    unsigned char keycode; //定义命令字，和数据存储器
    Send_Command(CH452_GET_KEY);
    keycode=0; //清除 keycode
    for(i=0;i!=7;i++)
    {
        keycode<<=1; //数据移入 keycode, 高位在前, 低位在后
        if (P2IN&BIT0) keycode++; //从高到低读入 451 的数据
        DCLK_0; //产生时钟下降沿通知 CH451 输出下一位
        DCLK_1;
    }
    return(keycode); //返回键值
}
```

```
void main( void )
```

```
{
    // Stop watchdog timer to prevent time out reset
    WDTCTL = WDTPW + WDTHOLD;
    Init_4lines_Mode(); //初始化 4 线工作模式
    Send_Command(CH452_RESET);
    Send_Command(KeyDisplay_ON);
```

```
    P2IE|=BIT0; //开中断，P1.2 接 CH452 的 DOUT 引脚，当有键
按下时，DOUT 上产生由高到低的脉冲
```

```
    P2IES|=BIT0;
```

```

P2IFG&=~BIT0;

LCD_Port_Init();
LCD_Init();
Init_ADC10();
Init_TA();
SPI_M_Init();
P1DIR|=BIT5;
CHN_Dis(Start_Topic_0);
__delay_cycles(2000000);
Clr_ram();
_EINT();
while(1)
{
    if(Flag_Count0==1)
    {
        ADC10_Sample(100);
        Voltage0=ADC10_Result0*250000/10230;
        Voltage1=ADC10_Result1*250000/10230;
        val0=Voltage0/num;
        Flag_Count0=0;
    }

    if(Flag_Count1==1)
    {
        if(flag==0)
        {
            CHN_Dis(Start_Topic);
            Set_position_Dis(2,3);
            Write_Num(Voltage0,1);
            Set_position_Dis(3,4);
            Write_Num(num,0);
            Set_position_Dis(1,3);
            Write_Num(val0,1);
        }
        else
        {
            CHN_Dis(Start_Topic_1);
            Set_position_Dis(2,3);
            Write_Num(Voltage1,1);
            Set_position_Dis(1,3);
            Write_Num(178,1);
        }
    }
    Flag_Count1=0;
}

```

```

    }
}

//中断处理函数
#pragma vector = PORT2_VECTOR //中断处理程序，接收到 DOUT 脉冲信号时，运行之
__interrupt void Port2(void)
{
    unsigned char Keyvalue;
    Send_Command(CH452_GET_KEY);
    Keyvalue=Key_Read();
    switch(Keyvalue)
    {
        case Key0:
            {
                P1OUT&=~BIT5;
                SpiWriteDat(W0Buff, 2);
                P1OUT|=BIT5;
                Send_Command(NDis1);
                Send_Command(NDis2);
                // Send_Command(NDis3);
                Send_Command(Dis01);

                num=1;
                break;
            }

        case Key1:
            {
                P1OUT&=~BIT5;
                SpiWriteDat(W1Buff, 2);
                P1OUT|=BIT5;
                Send_Command(NDis1);
                Send_Command(NDis2);
                // Send_Command(NDis3);
                Send_Command(Dis02);

                num=2;
                break;
            }

        case Key2:
            {
                P1OUT&=~BIT5;
                SpiWriteDat(W2Buff, 2);
            }
    }
}

```

```

        P1OUT|=BIT5;
        Send_Command( NDis1);
        Send_Command( NDis2);
        // Send_Command( NDis3);
        Send_Command(Dis04);

        num=4;
        break;
    }
case Key3:
    {
        P1OUT&=~BIT5;
        SpiWriteDat(W3Buff, 2);
        P1OUT|=BIT5;
        Send_Command( NDis1);
        Send_Command( NDis2);
        // Send_Command( NDis3);
        Send_Command(Dis08);

        num=8;
        break;
    }
case Key4:
    {
        P1OUT&=~BIT5;
        SpiWriteDat(W4Buff, 2);
        P1OUT|=BIT5;
        Send_Command( NDis2);
        // Send_Command( NDis3);
        Send_Command(Dis06);
        Send_Command(Dis11);

        num=16;
        break;
    }
case Key5:
    {
        P1OUT&=~BIT5;
        SpiWriteDat(W5Buff, 2);
        P1OUT|=BIT5;
        Send_Command( NDis2);
        // Send_Command( NDis3);
        Send_Command(Dis02);
        Send_Command(Dis13);
    }

```



```

        num=32;
        break;
    }
case Key6:
    {
        P1OUT&=~BIT5;
        SpiWriteDat(W6Buff, 2);
        P1OUT|=BIT5;
        Send_Command( NDis2);
        // Send_Command( NDis3);
        Send_Command(Dis04);
        Send_Command(Dis16);

        num=64;
        break;
    }
case Key7:
    {
        P1OUT&=~BIT5;
        SpiWriteDat(W7Buff, 2);
        P1OUT|=BIT5;
        // Send_Command( NDis3);
        Send_Command( NDis1);
        Send_Command(Dis08);
        Send_Command(Dis12);
        Send_Command(Dis21);

        num=128;
        break;
    }
case 0x50:
    {

        // Send_Command(Dis08);
        break;
    }
case 0x51:
    {

        // Send_Command(Dis09);
        break;
    }
case 0x52:

```

```

        {
            //    Send_Command(Dis00);
            //    Send_Command(Dis11);
            break;
        }
    case 0x53:
        {
            //    Send_Command(Dis01);
            //    Send_Command(Dis11);
            break;
        }
    case 0x58:
        {
            //    Send_Command(Dis02);
            //    Send_Command(Dis11);
            break;
        }
    case 0x59:
        {
            //    Send_Command(Dis03);
            //    Send_Command(Dis11);
            break;
        }
    case 0x5A:
        {
            Clr_ram();
            flag=1;
            break;
        }
    case 0x5B:
        {
            Clr_ram();
            flag=0;
            break;
        }
    default:break;
}
P2IFG&=~BIT0;
}

```

5.5.3 RS-232 接口通信实验 (UART 接口)

1. 实验准备

LaunchPad实验系统标准板，三种通信接口模块。

2. 实验目的

了解RS-232通信方式的原理和简单实现方式。

3. 实验步骤

(1) 用USB线连接LaunchPad和PC机，设置LaunchPad的跳线J3部分的TXD、RXD按照J3左方的SW|UART示意图连接跳线帽，其余三个跳线帽默认。

(2) 打开CCS，编写程序（可参考程序示例）并将程序写入单片机。

(3) 设置LaunchPad的跳线J3部分的TXD、RXD按照J3左方的HW|UART示意图连接跳线帽，这样才能与PC机进行串口通信

(4) 打开PC机上的串口调试助手程序（64位Win7系统下可能会出现程序不兼容的情况，选择一款支持64位Win7的软件即可），设置通信格式为9600bps，8数据位，1停止位，无校验位，如下图所示。注意图中端口号要设置为自己所用端口号，不一定是COM3。



图5.5.2 初始化串口调试助手

(5) 点击打开串口，成功后在串口调试助手的发送区写入字符，点击发送，观察串口调试助手接收区的显示情况，如图5.5.3所示。



图5.5.3 PC机发送字符后由单片机接收并回发

4. 程序示例

程序功能为接收PC机发送来的数据后自动回发给PC机。

```
#include "msp430g2553.h"
void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;           //停止看门狗
    UCAOCTL1 |= UCSWRST;                // USCI_A0 进入软件复位状态
    UCAOCTL1 |= UCSSEL_2;                //时钟源选择 SMCLK
    BCSCCTL1 = CALBC1_1MHZ;             //设置 DCO 频率为 1MHz
    DCOCTL = CALDCO_1MHZ;
    P1SEL = BIT1 + BIT2 ;                // P1.1 = RXD, P1.2=TXD
    P1SEL2 = BIT1 + BIT2 ;              // P1.1 = RXD, P1.2=TXD
    P1DIR |= BIT0;
    UCAOBR0 = 0x68;                      //时钟源 1MHz 时波特率为9600
    UCAOBR1 = 0x00;                      //时钟源 1MHz 时波特率为9600
    UCAOMCTL = UCBSR0;                  //小数分频器
    UCAOCTL1 &= ~UCSWRST;               //初始化 USCI_A0 状态机
    IE2 |= UCAORXIE;                   //使能 USCI_A0 接收中断
    _EINT();                             //开总中断
}
```

```

while(1)
{
}
}

#pragma vector = USCIAB0RX_VECTOR           //接收中断
__interrupt void USCI0RX_ISR(void)
{
while ( !(IFG2&UCA0TXIFG) );              //确保发送缓冲区准备好
P1OUT ^= BIT0;                             //接收指示灯状态改变
UCA0TXBUF = UCA0RXBUF;                     //发送接收到的数据
}

```

5.5.4 RS-485 接口通信实验 (UART 接口)

1. 实验准备

LaunchPad实验系统标准板,三种接口通信模块,RS232/485转换器(如图1所示),MAX3485芯片,面包板线若干。



图5.5.4 某型号的RS232/485转换器

2. 实验目的

了解和掌握RS-232的通信原理和基本实现方式。

3. 实验步骤

(1) 按照图5.5.5在面包板上搭建电路,把RS232/485转换器的串口端连接至PC机串口,RS232/485转换器的485端按图5.5.5接线。

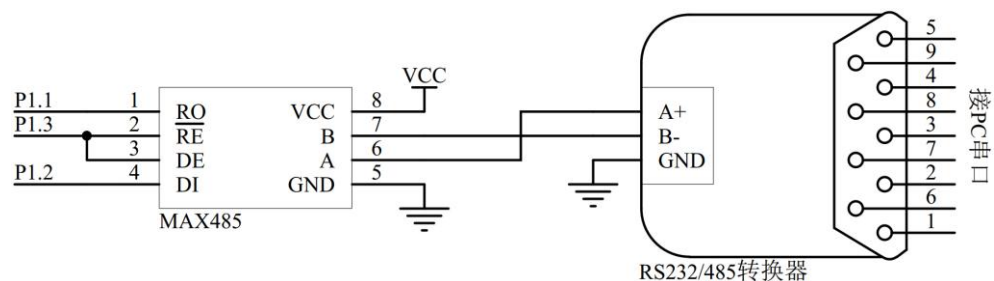


图5.5.5 RS-485接口通信实验电路图

(2) 设置LaunchPad的跳线J3部分的TXD、RXD按照J3左方的SW|UART示意图连接跳线帽,其余三个跳线帽默认。

(3) 打开CCS,编写程序(可参考程序示例),并将程序写入单片机。

(4) 设置LaunchPad的跳线J3部分的TXD、RXD按照J3左方的HW|UART示意图连接跳线帽

或者直接拔掉，否则485通信可能会受LaunchPad板上程序下载电路的影响。

(5) 打开PC机上的串口调试助手程序(64位Win7系统下可能会出现程序不兼容的情况，选择一款支持64位Win7的软件即可)，设置通信格式为9600bps，8数据位，1停止位，无校验位，如实验三中所示。注意图中端口号要设置为自己所用端口号，不一定为COM3。

(6) 点击打开串口，成功后在串口调试助手的发送区写入字符，点击发送，观察串口调试助手接收区的显示情况，如实验二中所示。

4. 程序示例

本范例由MSP430单片机作为RS485总线上的从机，PC机作为主机。功能是由PC机发送字符到从机，从机接收到后再发回主机。此实验的程序部分之比上一小节的RS232实验中多了一个IO的方向控制，用来控制485芯片的接收与发送。

```
#include "msp430g2553.h"
void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;           //停止看门狗
    UCAOCTL1 |= UCSWRST;                 // USCI_A0 进入软件复位状态
    UCAOCTL1 |= UCSSEL_2;                 //时钟源选择 SMCLK
    BCSCCTL1 = CALBC1_1MHZ;              //设置 DCO 频率为 1MHz
    DCOCTL = CALDCO_1MHZ;
    P1SEL = BIT1 + BIT2 ;                // P1.1 = RXD, P1.2=TXD
    P1SEL2 = BIT1 + BIT2 ;               // P1.1 = RXD, P1.2=TXD
    P1DIR |= BIT0+BIT3;                  //P1.0用于控制接收指示灯，P1.3用于控制485收发状态
    UCAOBR0 = 0x68;                       //时钟源 1MHz 时波特率为9600
    UCAOBR1 = 0x00;                       //时钟源 1MHz 时波特率为9600
    UCAOMCTL = UCBR50;                    //小数分频器
    UCAOCTL1 &= ~UCSWRST;                 //初始化 USCI_A0 状态机
    IE2 |= UCAORXIE;                      //使能 USCI_A0 接收中断
    _EINT();                               //开总中断

    while(1)
    {
    }
}

#pragma vector = USCIABORX_VECTOR        //接收中断
__interrupt void USCIORX_ISR(void)
{
    while ( !(IFG2&UCA0TXIFG) );          //确保发送缓冲区准备好
    P1OUT ^= BIT0;                         //接收指示灯状态改变
    P1OUT|=BIT3;                           //使485为发送状态
    UCA0TXBUF = UCA0RXBUF;                 //发送接收到的数据
    delay_ms(10);
    P1OUT&=~BIT3;                          //使485为接收状态
```

}

5.5.5 : 18B20 温度测量 (单总线)

1. 实验准备

LaunchPad 实验系统标准板, DS18B20, 面包板线若干

2. 实验目的

了解总线式通信的原理和基本实现方式。

3. 实验步骤

(1) 按照如下电路图在面包板上连接实物, 其中 DS18B20 的 2 号管脚接 LaunchPad 的 P1.1 脚

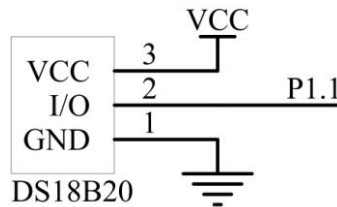


图 5.5.6 DS18B20 测温电路

(2) 打开 CCS, 参考程序示例编写 DS18B20 测温程序。

(3) 用 USB 线连接 LaunchPad 和计算机, 通过 CCS 将程序下载到 MSP430G2 单片机里并进入仿真模式。

(4) 在仿真界面内设置测得的温度值为观察值, 并在计算温度结束处设置断点, 改变 DS18B20 的温度 (如用手捏住 DS18B20), 每次运行至断点时观察温度值的变化。

4. 程序示例

```
#include "msp430g2553.h"
#define CPU_F ((double)1000000)
#define delay_us(x) __delay_cycles((long)(CPU_F*(double)x/1000000.0)) //延时 x
微秒
#define DQ_H P1DIR&=~BIT1 //P1.1 设为输入, 上拉电阻把总线拉高, 即电阻上拉
#define DQ_L P1DIR|=BIT1
#define DQ_0 P1OUT&=~BIT1 //P1.1 设为输入, 输出低电平, 把总线拉低, 即总线
主器件拉低
unsigned char temp1,temp2;
/*****
* 名 称: reset()
* 功 能: 检测 DS18B20 初始化序列
* 入口参数: 无
* 出口参数: int t。t=1, DS18B20 初始化; t=0, DS18B20 没有初始化
*****/
int reset()
```

```

{
    int t;
    DQ_H;
    delay_us(50);
    _NOP();
    _NOP();
    DQ_L;
    DQ_0;          //主器件拉低
    delay_us(500); //最少 480us
    DQ_H;          //电阻上拉
    delay_us(60);  //等待 15~60us
    if((P2IN&BIT3)==0) t=1; //判断存在脉冲, DS18B20 是否拉低
    else t=0;
    delay_us(480); //最小 480us
    return(t);     //若 DS18B20 复位, 返回值 t=1
}
/*****
* 名 称: writebit()
* 功 能: 单片机对 DS18B20 写时隙
* 入口参数: unsigned char bit。bit=1, 写 1 时隙; bit=0, 写 0 时隙。
* 出口参数: 无
*****/
void writebit(unsigned char bit)
{
    DQ_L;          //主器件拉低开始
    DQ_0;
    delay_us(3);   //最少延时 1us
    if(bit) DQ_H;  //电阻上拉, 写 1
    else
    {
        DQ_L;      //主器件拉低, 写 0
        DQ_0;
    }
    delay_us(80);  //写时隙必须至少持续 60 μs
    DQ_H;
    delay_us(60);  //大于 1us 延时即可
}
/*****
* 名 称: readbit()
* 功 能: 单片机对 DS18B20 读时隙
* 入口参数: 无
* 出口参数: char bit。bit=1, 读 1 时隙; bit=0, 读 0 时隙。
*****/
unsigned char readbit()

```



```

{
    char bit;
    DQ_L;          //主器件拉低开始
    DQ_0;
    delay_us(3);   //延时大于 1us
    DQ_H;          //IO 口设为输出, DS18B20 拉低, 则读出 0; 电阻上拉拉高,
    则读出 1
    delay_us(3);   //延时小于 15us
    if(P1IN&BIT1) bit=1; //判断 IO 输入状态
    else bit=0;
    delay_us(80);  //延时大于 45us
    DQ_H;
    delay_us(60);
    return(bit);
}
/*****
* 名 称: write_byte ()
* 功 能: 单片机向 DS18B20 写入一个字节
* 入口参数: unsigned char byte。byte 为单片机写入的数据
* 出口参数: 无
*****/
void write_byte(unsigned char byte)
{
    int i;
    for(i=0;i<8;i++)
    {
        writebit(byte&0x01); //byte 最后一位为 1, 这写 1 时隙; 若为 0, 则写 0 时
    隙
        byte=byte>>1; //byte 右移一位, byte 从低位到高位, 逐位写入
        _NOP();
    }
}
/*****
* 名 称: read_byte ()
* 功 能: 单片机从 DS18B20 读出一个字节
* 入口参数: 无
* 出口参数: unsigned char ans。ans 为单片机从 DS18B20 读出的数据
*****/
unsigned char read_byte()
{
    int t;
    unsigned char ans=0;
    for(t=0;t<8;t++)
    {

```

```

    ans>>=1;                //ans 右移一位
    if(readbit())
        ans|=0x80;         //若读出 1 则, ans 最高位写入 1, 写满 8 位
        _NOP();
    }
    return(ans);           //返回读出的数据
}

void main()
{
    WDTCTL=WDTPW+WDTHOLD;
    float t;
    while(1)
    {
        while(!reset())    //访问 DS18B20 以一个初始化序列未开始
        {}
        write_byte(0xCC);  //总线上只有一个 DS18B20, 因此可直接跳过寻址
        write_byte(0x44);  //开始温度转换
        delay_us(5000);    //等待转换
        while(!reset())    //等待 DS18B20 复位
        {}
        write_byte(0xCC);
        write_byte(0xBE);  //读暂存器
        temp1=read_byte(); //读出 LS
        temp2=read_byte(); //读出 MS
        reset();
        t=(float)temp1*0.0625+(float)temp2*16; //计算温度值
        t=t*10+0.5;       //放大 10 倍, 四舍五入
    }
}

```

第六章 MSP430G2 系列单片机应用实践

第一节 用三种温度传感器实现的温度巡检/控制器

6.1.1 目的与要求

本节运用三种温度测量模块和频率计及 D/A 转换模块与实验系统标准板配合,实现温度巡检,并根据温度值发出相应的控制信号,具体实现功能如下:显示三种温度传感器测量的温度值、设置温度上下限报警、根据一路温度大小输出对应的 0 到 5V 电压。

本应用实践可用于恒温装置,输出的 0 到 5V 信号用于控制加热装置,结合 PID 算法,温度传感器将温度值反馈给单片机,单片机根据测量的温度和设定要达到的恒定温度的差值,单片机输出对应的电压控制加热装置的工作状态,差值越大输出的电压越大。

6.1.2 电路设计和系统连接

本应用选择三种温度测量模块和频率计及 D/A 转换模块,三种温度测量模块将测量的温度值反馈给单片机,单片机控制频率计及 D/A 转换模块输出 0 到 5V 的电压信号。其系统组成框图如下:

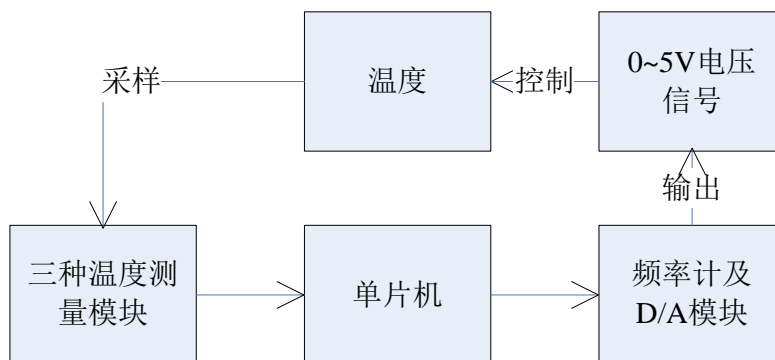


图 6.1.1 用三种温度传感器实现温度巡检/控制器系统框图

将两个模块插在实验系统标准版的插槽上,其电路连接如下图所示。

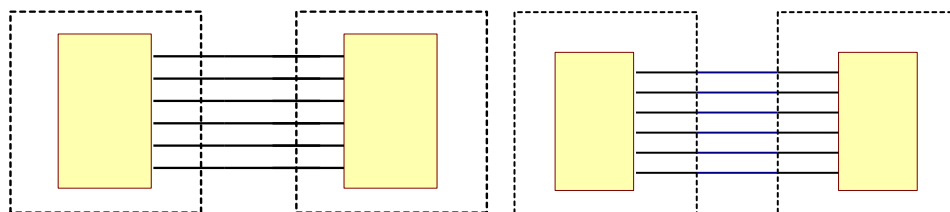


图 6.1.2 LaunchPad 开发板与频率计 DA 模块和三种温度测量模块连接图

6.1.3 信号与数据处理

从本应用的要求可知,需要设计一种控制算法根据设定温度与实际温度的差异来实时调整加热装置控制信号大小。频率计及 D/A 模块输出 0~5V 电压信号用来控制加热装置的功率大小,因此只需实时调整 D/A 输出电压信号大小即可调节加热装置的功率。在这里仅大致介绍最常用的 PID 算法。

PID 算法包括 P 比例算法、I 积分算法和 D 微分算法,针对本应用,可设计的一种算法如下: P 比例算法, D/A 输出电压 $V_p = \text{比例系数} P * \text{温度差 } e$, 其中 P 为比例系数; I 积分算法,

D/A 输出电压 V_I =积分系数 I *温度差 e 的累加值, I 是控制积分时间常数; D 微分算法, D/A 输出电压 V_D =微分系数*温度变化速率 Δe , 其中 D 是微分系数。把上面三种运算结合起来, 即 D/A 输出电压值 $V=V_P+V_I+V_D$, 就成为控制中最常用的 PID 算法。

在使用 PID 算法是, 无需对被控对象进行建模, 只需用实验的方法不断尝试调节 P 、 I 、 D 三个系数的大小, 即可接近完美的控制效果。

下面列出 PID 算法的关键程序。

```
//=====
//调整 PID 的三个系数, 达到最佳的控制效果
#define P_Coefficient 40
#define D_Coefficient 480
#define I_Coefficient 0.08
//=====
#define U_MAX 5000 //D/A 能输出的最大电压值 5V
unsigned int Integral; //积分累计
int Prev_Error; //记录前一次误差
int P, I, D; //PID 分量
float Ek, E; //误差临时变量
unsigned char FirstFlag=1; //第一次执行表示
/*****
*名 称: PID_Caculate()
*功 能: PID 控制算法
*入口参数: Error: 实际温度与设定温度的误差, 10 代表 1℃
*出口参数: OutPut: 单片机应控制 D/A 输出的电压值, 1000=1000mV
*****/
int PID_Caculate(int Error)
{
    int i;
    float OutPut;
    if(FirstFlag) //判断第一次上电
    {
        FirstFlag=0;
        Prev_Error=Error;
    }//上电第一次时, 没有前一次测量, 赋当前值, 以免微分计算错误
    Ek=Error-Prev_Error; //相邻两侧测量值去差分
    Prev_Error=Error; //保存当前采样值, 下一次使用
    E=0.8*E+Ek*0.2; //微分通过 IIR 滤波器, 降噪
    P=P_Coefficient*Error; //计算比例分量
    I=I_Coefficient*Integral; //计算积分分量
    D=D_Coefficient*E; //计算微分分量
    OutPut=P+D+I; //PID 合成输出
    if((OutPut>10000) || (Error>100) || (Error<-100)) //输出饱和或偏差大时不积
分
    {
        //if((Integral>0)&&(Error<0)) Integral+=Error;
```

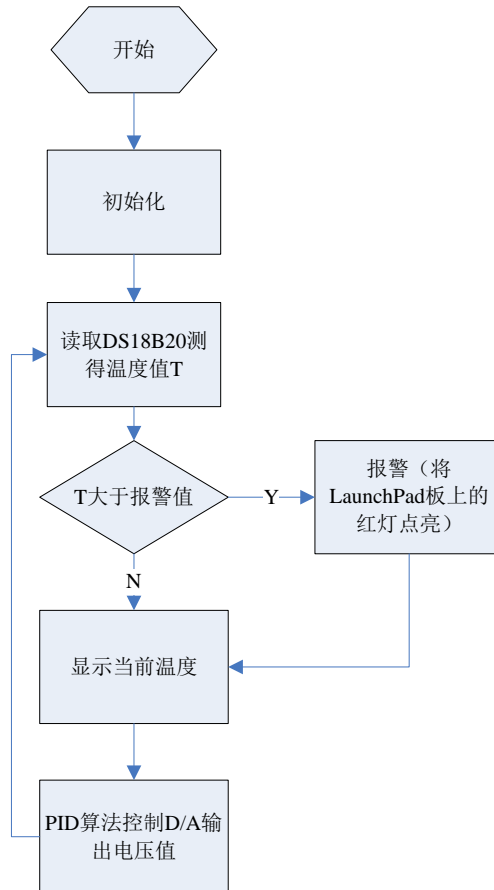
```

    //if((Integral<0)&&(Error>0)) Integral+=Error;
}
Else //偏差较小时启动积分
{
    Integral+=Error;
    if(Integral<0) Integral=0;
}
if(OutPut>1000) OutPut=1000;
else if(OutPut<0) OutPut=0;
OutPut= U_MAX*OutPut/10000; //计算 D/A 应输出的电压值
return(OutPut);
}

```

6.1.4 软件设计与调试

根据本应用的要求，其程序流程图大致如图 6.1.3 所示。



6.1.3 温度巡检/控制器程序流程图

本应用选用三种温度测量模块上的 DS18B20 测当前温度，若单片机从 DS18B20 读取的温度值 T 大于设定的报警值，则单片机发出报警信息；然后显示温度值 T，并根据 T 值运用 PID 算法，控制 D/A 模块输出的电压值。

DS18B20 测温的原理和程序示例已经在第五章第三节中提到，这里不做叙述。下面给出单片机通过 SPI 控制 D/A 输出的程序示例。

```

#define DAC_PORT P2OUT
#define SYNC BIT5

```

```

#define SCLK BIT1
#define DIN BIT2
#define SYNC_L DAC_PORT &=~ SYNC
#define SYNC_H DAC_PORT |= SYNC
#define SCLK_L DAC_PORT &=~ SCLK
#define SCLK_H DAC_PORT |=SCLK
#define DIN_L DAC_PORT &=~ DIN
#define DIN_H DAC_PORT |= DIN
unsigned int frequency;
unsigned int time_us;
/*DAC8552——SPI 初始化程序*/
void dac_Init(void)
{
    P1OUT = 0x00;
    P1DIR |= BIT5 +BIT7; //P1.4 做 SYNC
    SYNC_H;
    P1SEL = BIT5 + BIT7; //P1.5、7 分别作 SCLK 和 SIMO
    P1SEL2 = BIT5 + BIT7;
    P2DIR |= BIT5;
    UCBOCTL1 |= UCSWRST; //禁止 USCI 状态机
    UCBOCTL0 = UCMSB + UCMST + UCMODE_0 + UCSYNC;//
    UCBOCTL1 = UCSSEL_2 + UCSWRST;
    UCBOBRO = 1;
    UCBOCTL1 &= ~UCSWRST; //启动 USCI 状态机
}
/*****
*名 称: out_v()
*功 能: D/A 电压输出
*入口参数: channl: D/A 输出通道选择, 共有三个通道; voltage: 要输出的电压值 (mV)
*出口参数: 无
*****/
void out_v(char channl,float voltage)
{
    unsigned int buf;
    char high,low;
    buf=(voltage/4950)*65535;
    low=buf&0xFF;
    high=buf>>8;
    switch(channl)
    {
        case 1:
        {
            SYNC_L; //选通拉低
            UCBOTXBUF = 0X10; //向 DA 写入命令——通道 A, 立即转换
        }
    }
}

```

```

    while (!(IFG2 & UCB0TXIFG));
    UCB0TXBUF = high;    //写入高 8 位
    while (!(IFG2 & UCB0TXIFG));
    UCB0TXBUF = low;    //写入低 8 位
    while (!(IFG2 & UCB0TXIFG));
    SYNC_H;            //选通拉高
}break;
    case 2:
    {
        SYNC_L;        //选通拉低
        UCB0TXBUF = 0x24; //向 DA 写入命令——通道 B, 立即转换
        while (!(IFG2 & UCB0TXIFG));
        UCB0TXBUF = high;    //写入高 8 位
        while (!(IFG2 & UCB0TXIFG));
        UCB0TXBUF = low;    //写入低 8 位
        while (!(IFG2 & UCB0TXIFG));
        SYNC_H;            //选通拉高
    }break;
    case 3:
    {
        TA1CCR2=(voltage/3540)*2000;
    }
}
}
}

```

根据上面的程序流程图, 结合 DS18B20 测温程序和 D/A 输出程序, 本应用的程序如下:

```

#define T_Aralm  320            //报警温度值 32.0℃
#define T_Set    300            //设定恒定温度值 30.0℃
void main()
{
    WDTCTL = WDTPW + WDTHOLD; // 停止看门狗
    BCSCTL1 = CALBC1_1MHZ;    // Set range
    DCOCTL = CALDCO_1MHZ;
    BCSCTL2 |= SELM_0;        //MCLK 采用 1M 的内部 DCO
    BCSCTL2 |= DIVS_0;        //配置时钟
    int T, Error;
    float V_OUT;
    dac_Init();                //D/A 模块初始化
    lcdreset();                //LCD12864 液晶初始化
    while(1)
    {
        T=ReadTemp();          //从 DS18B20 中读取温度, 10=1℃
        wr_lcd(comm, 0x80);
        printstring("当前温度值: ");
        Write_Num(0x92, T, 1); //LCD12864 显示温度值 T
    }
}

```

```

if(T>T_Aralm)          //T 大于报警值
{
    P1DIR|=BIT0;
    P1OUT|=BIT0;      //点亮红灯
}
else
{
    P1DIR|=BIT0;
    P1OUT&=~BIT0;
}
if(T>T_Set)           //T 大于设定恒温值
    Error=T-T_Set;
else
    Error=T_Set-T;
V_OUT=PID_Caculate(Error); //PID 算法
out_v(2, V_OUT);      //D/A 输出电压
}
}

```

6.1.5 总结与扩展

本应用通过三种测温模块和频率计及 D/A 模块与实验板配合,大致给出了一个恒温控制范例,通过此应用相信读者会对这两种实验模块有更好的理解。同时本应用也大概介绍了一种工程控制算法。

然而本应用 D/A 只是输出的电压信号,没有外接用于控制加热装置,因此无法完成恒温控制调试。读者可在实验系统标准版的扩展区域搭建加热电路,去完成恒温控制调试。

第二节 键盘输入控制的程控放大器

6.2.1 目的与要求

本节主要介绍结合按键功能模块及点阵液晶显示器的程控放大器综合应用实验。通过按键选择程控放大器的放大倍数，并在液晶上实时显示输入值、实际输出值、理论输出值及放大倍数，可以根据显示值计算程控放大器的误差。

通过该应用实践课题可以使读者熟练掌握程控运算放大器模块和按键功能模块的原理，学会应用 SPI 通信、A/D 转换，点阵液晶显示器控制及 I/O 口中断等技术方法，将理论与实践相结合，从而锻炼了读者的动手能力和独立思考问题及解决问题的能力。

在该应用实践课题过程中，A/D 转换的精度可以再提高一些，这样可以减小程控放大器的误差，另外，按键功能模块也可由底板上的触摸按键代替来控制程控运算放大器的放大倍数，读者可根据实际情况，解决相关问题，顺利完成该应用实践课题。

6.2.2 电路设计与系统连接

该应用实践课题系统电路由矩阵键盘及数码管模块+程控放大器模块+点阵 LCD 显示器组成，可通过按键选择放大倍数，点阵 LCD 显示器显示相关信息。其设计思想整体可简单概括为将输入信号接到 PGA112 的信号输入引脚，PGA112 通过与单片机 MSP430G2553 进行的 SPI 通信来改变其增益，单片机通过 I/O 中断读取按键状态，向程控放大器 PGA112 发送不同的 SPI 命令控制其增益变化，并将 PGA112 输出的信号接到单片机进行 A/D 转换和采样，最后在点阵 LCD 显示器上显示输入值、理论输出值、实际输出值及放大倍数。该按键功能模块实际上代替了程控运算放大模块上的四位拨码开关，起到了选择放大倍数的作用。

下面介绍一下按键功能模块，由第五章的按键功能模块介绍可知，当某一个按键按下时，会引起 I/O 口中断，并在中断内根据不同键值执行相应的代码，其中键值与放大倍数选择的关系如表 1 所示

表 1 键值与放大倍数选择的关系表

键值	Key0	Key1	Key2	Key3	Key4	Key5	Key6	Key7
放大倍数	1	2	4	8	16	32	64	128
数码管显示	1	2	4	8	16	32	64	128

按键功能模块与一体化实验系统的连接如图 6.2.1 所示

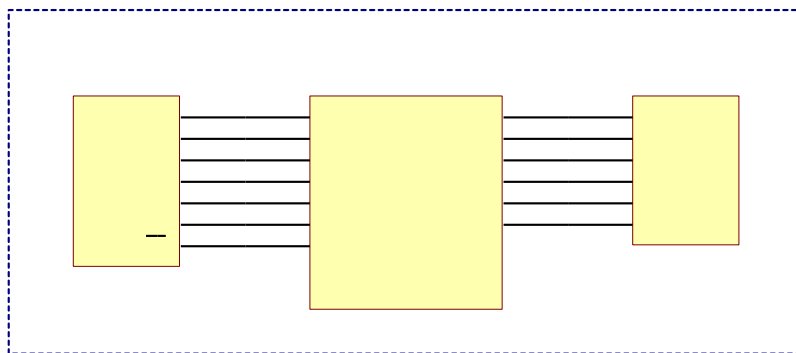


图 6.2.1 按键功能模块与一体化实验系统的连接图

将矩阵键盘及数码管模块、程控放大器模块及点阵 LCD 显示器安装在一体化实验系统上

时，注意模块的放置方向及排针的对齐方式，以免造成错误，并将 DIP 开关拨到对应连接位置上。

6.2.3 软件设计与调试

这部分主要介绍该应用实践课题的软件设计与调试，软件设计主要包括对 SPI 通信、A/D 采样、I/O 中断及液晶显示的设计。下面对关键子程序及中断处理程序进行说明。

首先介绍一下 SPI 通信程序设计，其流程图如图 6.2.2 所示

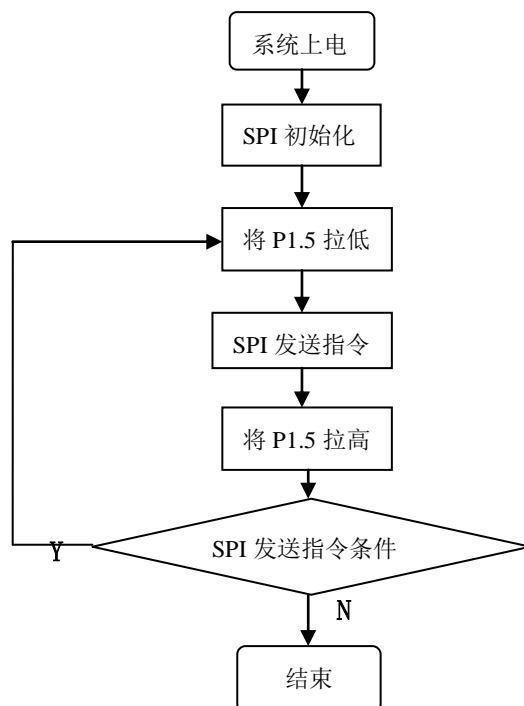


图 6.2.2 SPI 通信流程图

软件实现部分可参考第五章。

接下来介绍 A/D 采样程序设计，该 A/D 采样的软件设计通过多次采样求均值获得采样值，这样可以减小采样误差，其流程图如图 6.2.3 所示

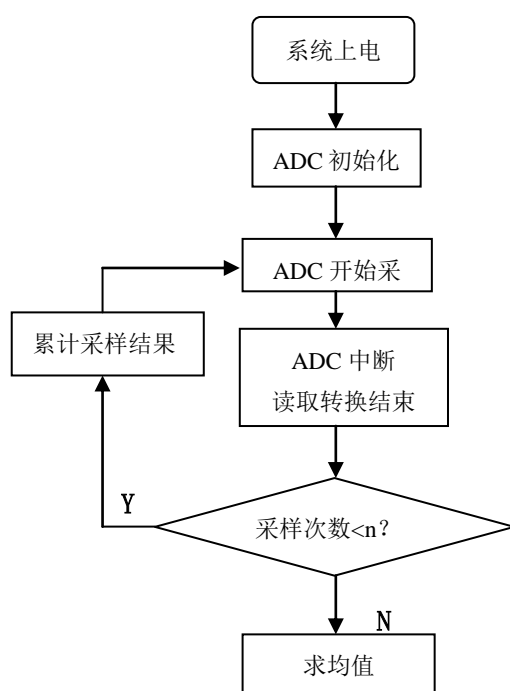




图 6.2.3 A/D 采样流程图

其软件实现如下：

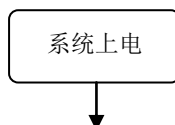
```

void Init_ADC10(void)                                     //ADC10 初始化
{
    ADC10CTL0 |= REFON+REF2_5V+ADC10ON+ADC10IE+MSC; //参考电压 2.5V 多
    次采样模式 打开 ADC10 模块 中断 ADC10 允许
    ADC10CTL0 |= ADC10SHT_3+SREF_1;
    ADC10CTL1|=ADC10SSEL_3+ADC10DIV_0+INCH_0+SHS_0+CONSEQ_2;
    //SMCLK, 无分频, 通道 0 单通道多次转换
    ADC10AEO |= 0x01;
}

void ADC10_Sample(unsigned int AverageNum)              //ADC10 采样函数
{
    unsigned long int ADC10_Sum0=0;
    unsigned int i;
    ADC10CTL0 |= ADC10IE;
    ADC10CTL0 |= ADC10SC+ENC;                            //开始采样
    for(i=0;i<AverageNum;i++)
    {
        while((ADC10_Flag)==0);
        ADC10_Flag=0;
        ADC10_Sum0+=(int)ADC_Result0;
    }
    ADC10CTL0 &=~ ENC;                                    //禁止采样
    ADC10CTL0 &=~ ADC10IE;
    ADC10_Result0=ADC10_Sum0/(AverageNum);              //多次采样求平均
}

#pragma vector=ADC10_VECTOR                               //采样中断内读取采样
值
__interrupt void ADC10ISR(void)
{
    ADC_Result0=ADC10MEM;
    ADC10_Flag=1;
}
  
```

最后介绍 I/O 中断程序设计，其流程图如图 6.2.4 所示



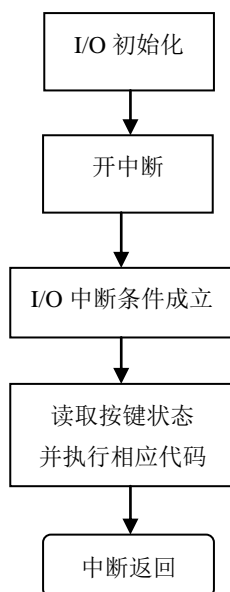


图 6.2.4 I/O 中断流程图

结合表 1，按键控制程控放大倍数的软件实现如下

```

#pragma vector = PORT2_VECTOR // P2 口中断函数
__interrupt void Port2(void)
{
    unsigned char Keyvalue;
    Send_Command(CH452_GET_KEY); //向 CH452A 发送获取按键,
    返回按键代码命令
    Keyvalue=Key_Read(); // 读取按键键值
    switch(Keyvalue)
    {
        case Key0: // 按下 K0 键
            {
                P1OUT&=~BIT5;
                SpiWriteDat(W0Buff, 2); // 放大 1 倍
                P1OUT|=BIT5;
                Send_Command( NDis1); // 数码管显示 1
                Send_Command( NDis2);
                Send_Command(Dis01);
                num=1; // 放大倍数=1
                vall=val0*1; // 理论输出值=输入值*放
                大倍数
            }
            break;
        case Key1: // 按下 K1 键
            {
                P1OUT&=~BIT5;
                SpiWriteDat(W1Buff, 2); // 放大 2 倍
            }
    }
}
  
```

```

        P1OUT|=BIT5;
        Send_Command( NDis1);           // 数码管显示 2
        Send_Command( NDis2);
        Send_Command(Dis02);
        num=2;                           // 放大倍数=2
        val1= val0*2;                     // 理论输出值
        break;
    }
case Key2:                               // 按下 K2 键
    {
        P1OUT&=~BIT5;
        SpiWriteDat(W2Buff, 2);         // 放大 4 倍
        P1OUT|=BIT5;
        Send_Command( NDis1);           // 数码管显示 4
        Send_Command( NDis2);
        Send_Command(Dis04);
        num=4;                           // 放大倍数=4
        val1= val0*4;                     // 理论输出值
        break;
    }
case Key3:                               // 按下 K3 键
    {
        P1OUT&=~BIT5;
        SpiWriteDat(W3Buff, 2);         // 放大 8 倍
        P1OUT|=BIT5;
        Send_Command( NDis1);           // 数码管显示 8
        Send_Command( NDis2);
        Send_Command(Dis08);
        num=8;                           // 放大倍数=8
        val1= val0*8;                     // 理论输出值
        break;
    }
case Key4:                               // 按下 K4 键
    {
        P1OUT&=~BIT5;
        SpiWriteDat(W4Buff, 2);         // 放大 16 倍
        P1OUT|=BIT5;
        Send_Command( NDis2);           // 数码管显示 16
        Send_Command(Dis06);
        Send_Command(Dis11);
        num=16;                           // 放大倍数=16
        val1= val0*16;                     // 理论输出值
        break;
    }

```

```

case Key5: // 按下 K5 键
{
    P1OUT&=~BIT5;
    SpiWriteDat(W5Buff, 2); // 放大 32 倍
    P1OUT|=BIT5;
    Send_Command( NDis2); // 数码管显示 32
    Send_Command(Dis02);
    Send_Command(Dis13);
    num=32; // 放大倍数=32
    val1= val0*32; // 理论输出值
    break;
}
case Key6: // 按下 K6 键
{
    P1OUT&=~BIT5;
    SpiWriteDat(W6Buff, 2); // 放大 64 倍
    P1OUT|=BIT5;
    Send_Command( NDis2); // 数码管显示 64
    Send_Command(Dis04);
    Send_Command(Dis16);
    num=64; // 放大倍数=64
    val1= val0*64; // 理论输出值
    break;
}
case Key7: // 按下 K7 键
{
    P1OUT&=~BIT5;
    SpiWriteDat(W7Buff, 2); // 放大 128 倍
    P1OUT|=BIT5;
    Send_Command( NDis1); // 数码管显示 128
    Send_Command(Dis08);
    Send_Command(Dis12);
    Send_Command(Dis21);
    num=128; // 放大倍数=128
    val1= val0*128; // 理论输出值
    break;
}
default:break;
}
P2IFG&=~BIT0;
}

```

以上完成了按键控制程控放大器的放大倍数选择,并将放大倍数通过数码管实时显示的功能。对于液晶显示部分,该应用实践选择 12864 液晶,其驱动程序很常见,读者可根据数据手册自己动手写一下,也可参照网上的一些驱动程序,这里就不在重复了。

关于主程序的结构，这里简单介绍一下，该应用实践课题的主程序结构流程图如图 6.2.5 所示

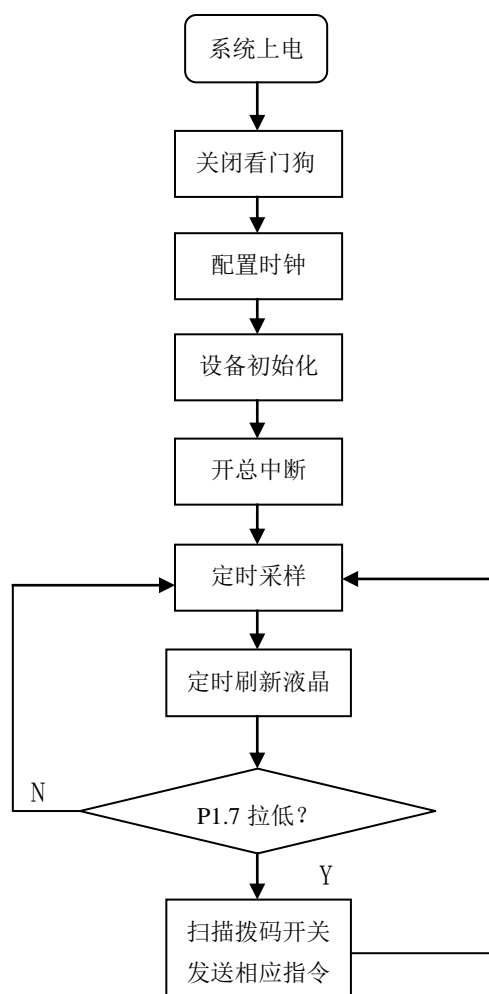


图 6.2.5 主程序流程图

在主程序软件设计中，注意定时刷新液晶，如果一直扫描液晶，可能由于液晶显示频率太快导致液晶显示界面闪烁的厉害，同时也会增大功耗。

下面简单介绍一下该应用实践课题的软件调试，软件开发环境选的 CCS，第三章已经对开发环境 CCS 进行了详细介绍，简单概括为首先建一个工程，选择相应器件类型，然后编写各部分程序，并将编写的程序添加到该工程，如图 6.2.6 所示

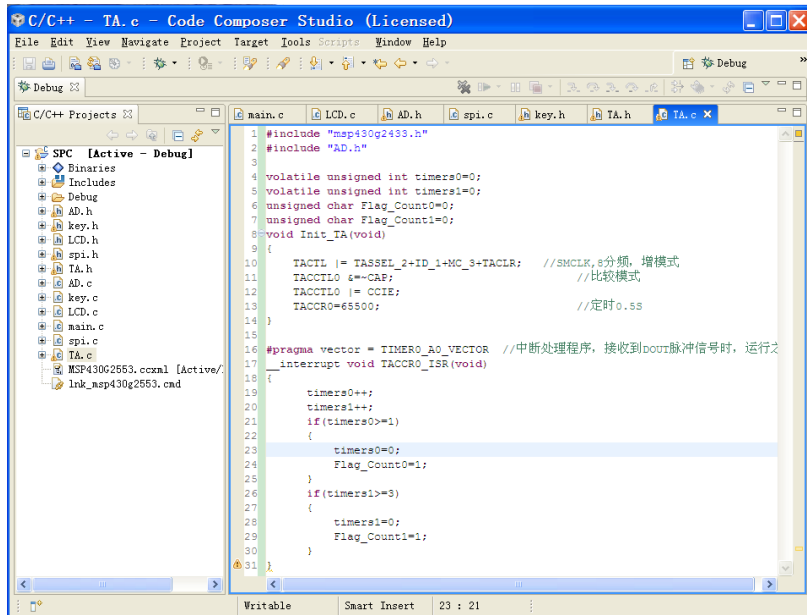


图 6.2.6 CCS 工程界面

接下来开始对写好的程序进行编译，可以选择菜单栏中 Project 选项里的 Rebuild All 进行编译，也可以选中工程点击右键选择 Rebuild Project 进行编译，编译时的界面如图 6.2.7 所示

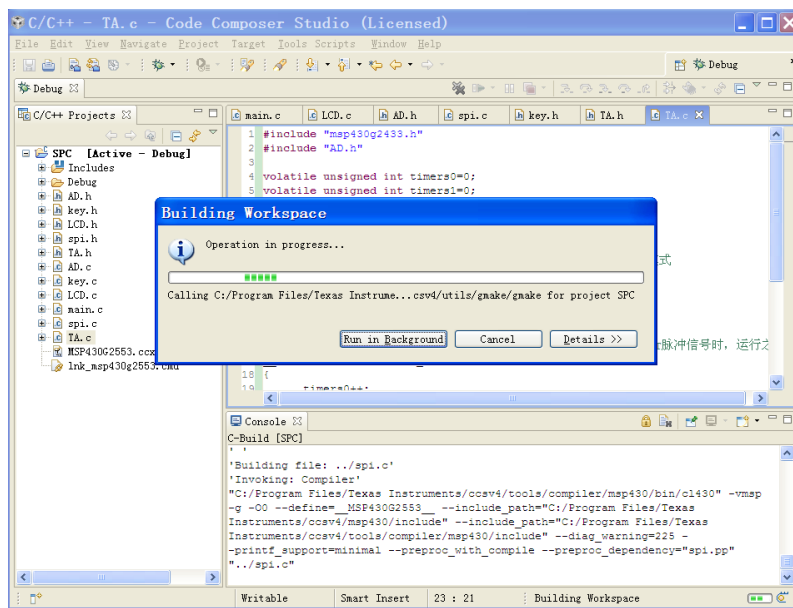


图 6.2.7 CCS 编译界面

编译完成后的界面如图 6.2.8 所示

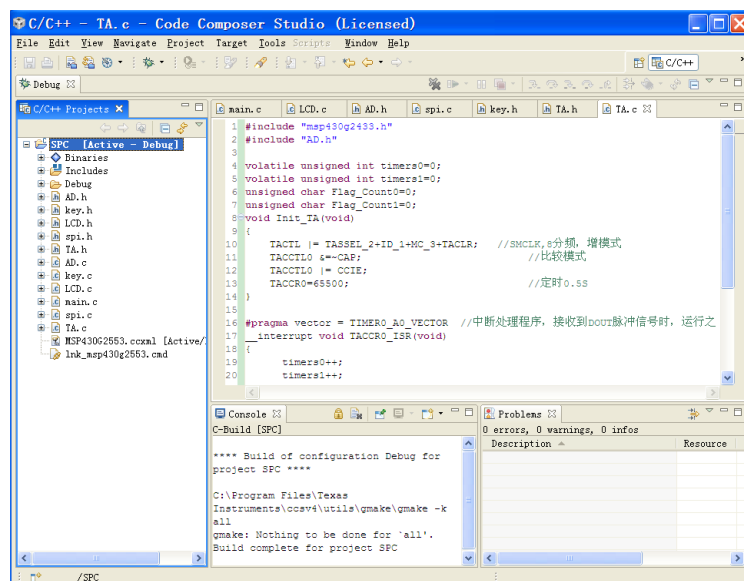



图 6.2.8 编译结束界面

如果有错误或警告会在界面上显示，如上图右下角部分所示，此时必须根据错误提示修改程序，直到没有错误。接下来将程序下载到硬件上进行调试，只需点击工具栏中的选项即可进行下载。调试过程中要充分利用 Debug 栏中的运行、单步、运行到光标处、设置断点等查找错误原因，逐步缩小错误范围，另外在调试过程中要有耐心，一步步找出错误，最终完成整个软件的调试。

6.2.4 总结与扩展

通过对该应用实践课题的软硬件设计及调试，实现了用按键选择程控放大器放大倍数并将相关值显示在液晶上的功能，基本达到了预期的功能指标。在这个实践课题中，可以使读者对 SPI 通信、A/D 采样、I/O 中断及液晶显示等有更深层的认识与理解，希望读者通过该应用实践可以熟练掌握其原理并学会应用，从而达到提升自己能力的目的。

众所周知，信号放大是信号调理技术的重要组成部分，应用十分广泛。在数据采集系统、自动增益控制、动态范围扩展、远程仪表测试等方面尤为适宜。例如在信号采集过程中，输入信号通常是微弱信号，这就要求信号调理电路具有放大功能，另外在很多情况下，需要根据输入信号的变化情况对放大器的增益做相应的改变，这时就需要采用程控放大器电路来进行处理。因此，该应用实践课题在实际应用中有重要意义。

该应用实践课题在实现方法仍存在一些不足，例如 A/D 采样的精度不够高，针对这一问题，可以在程控输出信号与单片机之间加一些外围滤波电路，这样会大大提高 A/D 采样的精度，进而减小程控放大器的误差。读者可根据自身需要，适当完善该应用实践课题，使其能更好的发挥控制放大器增益的功能。

第三节 声音强度检测仪

6.3.1 声音强度采样方法

通过第四章第三节声音强度检测模块的介绍,对该模块的硬件有了一定的了解。如果在此基础上,认为只需将此模块的输出电压经单片机 A/D 转换并作相应处理,即可得到所需声音强度,其实则不然,因为声音强度是一个变化非常快的物理量,将声音强度检测模块的输出连接至示波器,就可发现它的波形并不是随着声音强度的增大而增大,减小而减小,而是其峰峰值(或是振幅)随着声音强度的增大而增大,减小而减小。换句话说,该模块初始输出的是交流电压信号,并非直流,所以直接采样肯定会出现不准的地方,因此,作者在光照度检测模块和单片机 A/D 模块之间加了一个峰值检波电路,其作用就是抽取该模块的输出峰值(即振幅)进行 A/D 转换。

6.3.2 声音强度数据处理

在该模块的软件设计中,主要分为以下几个模块:显示模块、IO 初始化模块、ADC 初始化模块以及中断处理模块。根据以上模块的划分,程序总体框图如下图 6.3.1 所示:

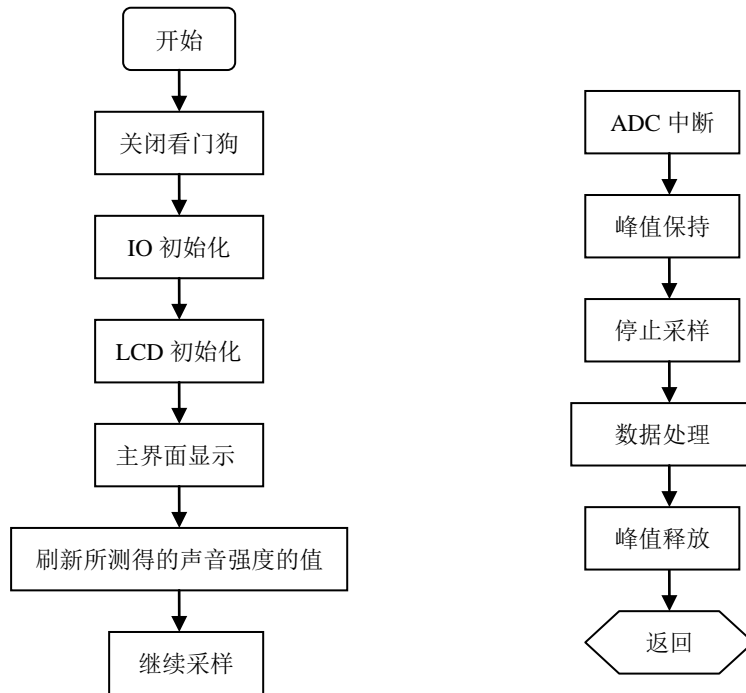


图6.3.1 光照度检测程序框图

图 6.3.1 中,左边是主函数程序框图,右边是中断程序框图。显示部分通过每 50ms 刷新一次的频率来实现实时声音强度的显示。为了能够准确、实时的显示光照度的大小,在数据处理时采取采样 8 次求平均值的方法,这样可有效避免某些干扰导致的不正确现象。具体实现步骤详见程序设计部分。

需要说明的是中断程序中的数据处理部分。经查阅大量资料,声音强度(dB)与该模块的输出电压成对数关系,但具体系数尚不可知,经过大量对比、matlab 曲线拟合,最终得到声音强度的换算公式如下:

$$\text{dB}=70+\lg(\text{ave}\times 2.5/1023)$$

上式中 ave 为采样电压的平均值，2.5/1023 为 A/D 的分辨率。该曲线关系由 matlab 显示如下图 6.3.2 所示。

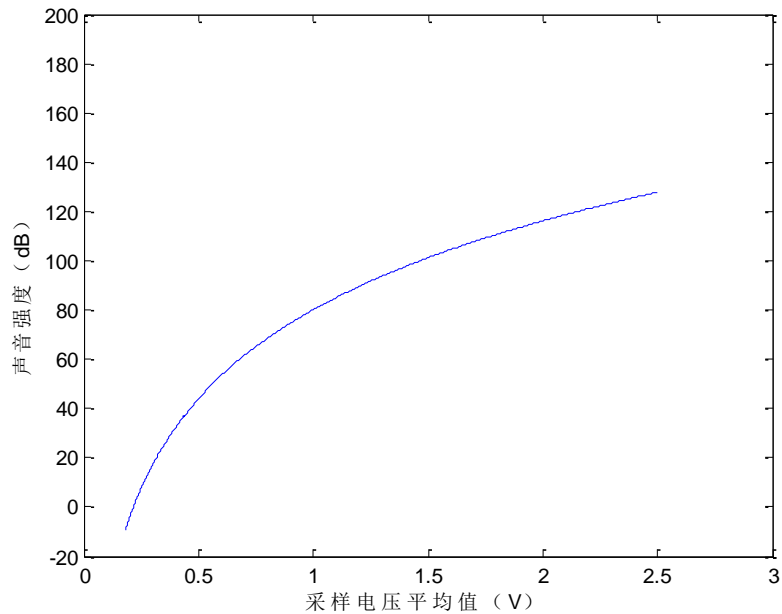


图 6.3.2 采样电压与声音强度换算关系曲线

6.3.3 光照度检测模块程序设计

```
#include "msp430g2553.h"
#include "math.h"
#include "LCD12864.h"

unsigned char const tab1[]={
    "          "
    "   LaunchPad   "
    "   Welcome    "
    "          "
};

unsigned char const tab2[]={
    "====欢迎使用===="
    "西安电子科技大学"
    "  声强检测模块  "
    "TexasInstruments"
};

unsigned char const tab3[]={
    "          "
    "          dB   "
    "  当前声强为:  "
    "          "
};

void IO_init()
```

```

{
    P2DIR|=BIT0;        //初始化 P2.0, 用于峰值检波
}
void ADC10_Init(void)
{
    ADC10CTL1=CONSEQ_2;//单通道多次转换, 这句应当写在最前面
    ADC10CTL0 = REFON+SREF_1+REF2_5V; //打开 2.5V 正参考, 地为负参考
    ADC10CTL0 |= ADC10ON+ADC10SHT_3+ADC10IE;//打开 ADC10 内核, 设定采样保持//持时
        间为 64 个 ADC10CLK, 使能 ADC10 中断
    ADC10CTL1 = INCH_3+SHS_0+ADC10SSEL_2;        // input A3, 采样保持
    ADC10AEO |= 0x02;        // PA.3 DC option select。A3 模拟信号输入使
能
}
unsigned int m, ave;
unsigned long  AD_Result, sum=0;
double dB1;
long dB;
void main()
{
    WDCTL = WDTPW + WDHOLD;        //关看门狗 r
    BCSCTL1 = CALBC1_1MHZ;
    DCOCTL = CALDCO_1MHZ;//上面两句将内部 DCO 校准至 1MHz
    BCSCTL2 |= SELM_0;//MCLK 采用 1M 的内部 DCO
    BCSCTL2 |= DIVS_0;//SMCLK 采用 125K 的时钟
    P2SEL &= ~(BIT6+BIT7);
    P2DIR |= BIT6+BIT7;
    lcd_init();//包含在头文件“LCD12864.h”中
    ADC10_Init();
    IO_init();
    _BIS_SR(GIE);//开中断
    chn_displ (tab1);
    delay_ms(3000);        //延时 3 秒
    chn_displ (tab2);
    delay_ms(3000);
    chn_displ (tab3);        //tab1、tab2、tab3 用于主界面显示
    while(1)
    {
        Write_Num(0x88, dB, 0);        //刷新显示的声音强度
        delay_ms(50);
        ADC10CTL0 |= ENC + ADC10SC;
    }
}
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR (void)

```

```

{
P2OUT&=~BIT0;           //峰值保持
ADC10CTL0 &= ~ENC;      // Disable ADC conversion
m++;                    //计数变量加一，记录进入 AD 中断的次数
AD_Result+=ADC10MEM;    //累加每次采样值
if(m>=8)                //判断采样的次数，若采样次数等于 8 ，作下面处理
{
    m=0;
    ave=AD_Result>>3;    //对累加和求平均
    AD_Result=0;
    dB1=ave*2.5/1023;
    dB=(long)(70+120*log10(dB1)); //声音强度换算公式
}
P2OUT|=BIT0;           //峰值释放
}

```

第四节 RS-232 接口与 PC 机通信

6.4.1 目的与要求

学习 MSP430G2553 单片机内部 UART 和 I/O 中断的使用方法，掌握用数码管显示数值或者某些字符的基本方法，熟悉通过 RS-232 接口与 PC 机通信的方法，掌握不同功能模块组合使用的系统软硬件构建方法。通过矩阵键盘及数码管模块输入按键值，并在 LED 数码管上显示出来，同时通过 RS-232 接口传送给 PC 机，在 PC 机显示器串口通信界面上给予显示。

6.4.2 电路设计与系统连接

由于本实践课题要求通过矩阵键盘输入键值，并在 LED 数码管上显示出来，同时通过 RS-232 接口传送给 PC 机，故系统由三种通信接口模块+矩阵键盘及数码管模块+PC 机组成。信号线连接方式为第四章三种通信接口模块与矩阵键盘及数码管模块连接方式的和，拨动开关拨到 RS-232 一端，用串口延长线将模块与 PC 机连接，连接好的实物图如图 6.4.1 所示。

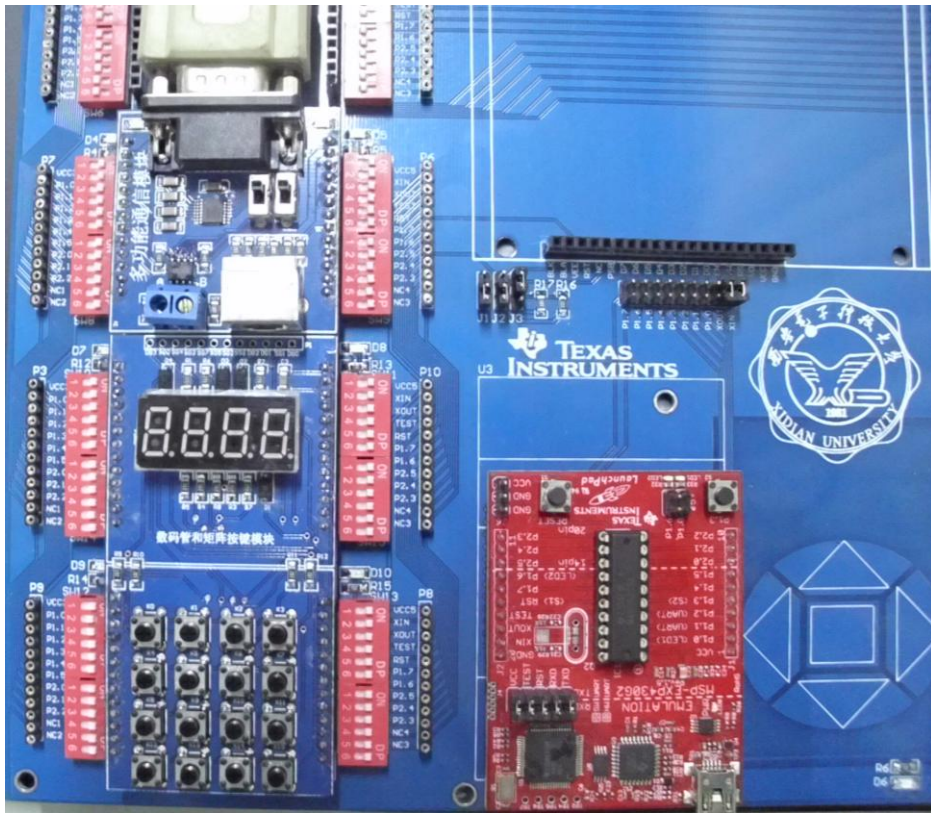


图 6.4.1 电路连接实物图

6.4.3 软件设计与调试

本实验的主要信号和数据处理过程即为矩阵键盘和数码管显示模块异步串行通信过程相关处理方法，在前面章节已给出，在此不再赘述，以下仅给出系统软件设计及调试方法。

对软件设计要求进行论述，对关键子程序或中断服务处理程序进行说明，给出流程图和例程。对主程序结构进行说明，给出主程序流程图和例程。对程序设计中的要点和技巧进行介绍，并注意在所有的例程中给出标注和注释。并注意介绍程序调试过程中应注意的问题和调试方法，给出最终结果显示画面等相关内容。

1. 系统程序流程图

软件设计要求体现出 MSP430 系列单片机的低功耗特性，且具有良好的实时性。流程图如图 6.4.1 所示。

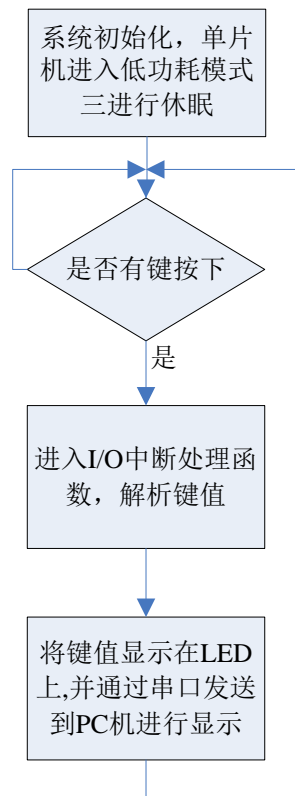


图 6.4.1 系统程序流程图

2. 例程

软件实现功能功能：按下某个按键，则数码管会显示该按键对应的键值，即若按下按键 K_i (i 取值为 $0\sim 15$)，在数码管上显示其键值 V_i ，如若 K_0 按下，则在数码管上显示数字 0，若 K_{15} 按下，则在数码管上显示数字 9999 等。各个按键的具体定义见下表 1，键值对应表见表 1，其定义是为了显示范围更广，用户可根据自己的喜好自行更改，程序中只需要修改相应宏定义即可。

K0	K1	K2	K3
K4	K5	K6	K7
K8	K9	K10	K11
K12	K13	K14	K15

图 6.4.2 检测程序中 4×4 矩阵键盘盘面

表一 键值对照表

K0	K1	K2	K3	K4	K5	K6	K7	K8	K9	K10	K11	K12	K13	K14	K15
0	1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	9999

1) 主函数

主函数中仅实现各功能部件初始化，然后就进入低功耗模式，等待事件触发。

```
void main( void )
```

```

{
    ConfigWDT();    //配置看门狗
    ConfigClocks(); //配置时钟
    UART_init();   //串口初始化
    Init_4lines_Mode(); //初始化4线工作模式
    Send_Command(CH452_RESET); //CH452芯片内部复位
    Send_Command(KeyDisplay_ON); //允许显示驱动并启动键盘扫描
    //开中断, P2.0接CH452的DOUT引脚, 当有键按下时, DOUT上产生由高到低的脉冲
    P2IE|=BIT0;
    P2IES|=BIT0;
    P2IFG&=~BIT0;
    _EINT();
    LPM3;
    while(1)
    {
    }
}

```

2) 按键中断处理函数

#pragma vector = PORT2_VECTOR //中断处理程序, 接收到DOUT脉冲信号时, 运行之

```

__interrupt void Port2(void)
{
    unsigned char Keyvalue;
    unsigned int U_Tem, TValue;
    Send_Command(CH452_GET_KEY); //单片机向CH452发送读取按键代码命令
    Keyvalue=Key_Read();
    switch(Keyvalue)
    {
        case 0x40://按键K0按下
        {
            Send_Command( NDis3); //第3位数码管不显示
            Send_Command( NDis2); //第2位数码管不显示
            Send_Command( NDis1); //第1位数码管不显示
            Send_Command( Dis00); //第0位数码管显示0
            U_Tem=0;
            TValue=U_Tem>>8;
            TX_BUFF[0]=(unsigned char)TValue;
            TX_BUFF[1]=(unsigned char)U_Tem;
            UART0_PutFrame(TX_BUFF, 2);
            break;
        }
        case 0x41://按键K1按下
        {
            Send_Command( NDis3); //第3位数码管不显示
            Send_Command( NDis2); //第2位数码管不显示

```



```

Send_Command( NDis1); //第1位数码管不显示
Send_Command( Dis01); //第0位数码管显示1
U_Tem=1;
TValue=U_Tem>>8;
TX_BUFFER[0]=(unsigned char)TValue;
TX_BUFFER[1]=(unsigned char)U_Tem;
UART0_PutFrame(TX_BUFFER, 2);
break;
}
case 0x42://按键K2按下
{
Send_Command( NDis3); //第3位数码管不显示
Send_Command( NDis2); //第2位数码管不显示
Send_Command( NDis1); //第1位数码管不显示
Send_Command( Dis02); //第0位数码管显示2
U_Tem=2;
TValue=U_Tem>>8;
TX_BUFFER[0]=(unsigned char)TValue;
TX_BUFFER[1]=(unsigned char)U_Tem;
UART0_PutFrame(TX_BUFFER, 2);
break;
}
case 0x43://按键K3按下
{
Send_Command( NDis3); //第3位数码管不显示
Send_Command( NDis2); //第2位数码管不显示
Send_Command( NDis1); //第1位数码管不显示
Send_Command( Dis04); //第0位数码管显示4
U_Tem=4;
TValue=U_Tem>>8;
TX_BUFFER[0]=(unsigned char)TValue;
TX_BUFFER[1]=(unsigned char)U_Tem;
UART0_PutFrame(TX_BUFFER, 2);
break;
}
case 0x48://按键K4按下
{
Send_Command( NDis3); //第3位数码管不显示
Send_Command( NDis2); //第2位数码管不显示
Send_Command( NDis1); //第1位数码管不显示
Send_Command( Dis08); //第0位数码管显示8
U_Tem=8;
TValue=U_Tem>>8;
TX_BUFFER[0]=(unsigned char)TValue;

```

```

        TX_BUFFER[1]=(unsigned char)U_Tem;
        UART0_PutFrame(TX_BUFFER, 2);
        break;
    }
case 0x49://按键K5按下
    {
        Send_Command( NDis3);//第3位数码管不显示
        Send_Command( NDis2);//第2位数码管不显示
        Send_Command( Dis11);//第1位数码管显示1
        Send_Command( Dis06);//第0位数码管显示6
        U_Tem=16;
        TValue=U_Tem>>8;
        TX_BUFFER[0]=(unsigned char)TValue;
        TX_BUFFER[1]=(unsigned char)U_Tem;
        UART0_PutFrame(TX_BUFFER, 2);
        break;
    }
case 0x4A://按键K6按下
    {
        Send_Command( NDis3);//第3位数码管不显示
        Send_Command( NDis2);//第2位数码管不显示
        Send_Command( Dis13);//第1位数码管显示3
        Send_Command( Dis02);//第0位数码管显示2
        U_Tem=32;
        TValue=U_Tem>>8;
        TX_BUFFER[0]=(unsigned char)TValue;
        TX_BUFFER[1]=(unsigned char)U_Tem;
        UART0_PutFrame(TX_BUFFER, 2);
        break;
    }
case 0x4B://按键K7按下
    {
        Send_Command( NDis3);//第3位数码管不显示
        Send_Command( NDis2);//第2位数码管不显示
        Send_Command( Dis16);//第1位数码管显示6
        Send_Command( Dis04);//第0位数码管显示4
        U_Tem=64;
        TValue=U_Tem>>8;
        TX_BUFFER[0]=(unsigned char)TValue;
        TX_BUFFER[1]=(unsigned char)U_Tem;
        UART0_PutFrame(TX_BUFFER, 2);
        break;
    }
case 0x50://按键K8按下

```

```

    {
        Send_Command( NDis3); //第3位数码管不显示
        Send_Command( Dis21); //第2位数码管显示1
        Send_Command( Dis12); //第1位数码管显示2
        Send_Command( Dis08); //第0位数码管显示8
        U_Tem=128;
        TValue=U_Tem>>8;
        TX_BUFFER[0]=(unsigned char)TValue;
        TX_BUFFER[1]=(unsigned char)U_Tem;
        UART0_PutFrame(TX_BUFFER, 2);
        break;
    }
case 0x51://按键K9按下
    {
        Send_Command( NDis3); //第3位数码管不显示
        Send_Command( Dis22); //第2位数码管显示2
        Send_Command( Dis15); //第1位数码管显示5
        Send_Command( Dis06); //第0位数码管显示6
        U_Tem=256;
        TValue=U_Tem>>8;
        TX_BUFFER[0]=(unsigned char)TValue;
        TX_BUFFER[1]=(unsigned char)U_Tem;
        UART0_PutFrame(TX_BUFFER, 2);
        break;
    }
case 0x52://按键K10按下
    {
        Send_Command( NDis3); //第3位数码管不显示
        Send_Command( Dis25); //第2位数码管显示5
        Send_Command( Dis11); //第1位数码管显示1
        Send_Command( Dis02); //第0位数码管显示2
        U_Tem=512;
        TValue=U_Tem>>8;
        TX_BUFFER[0]=(unsigned char)TValue;
        TX_BUFFER[1]=(unsigned char)U_Tem;
        UART0_PutFrame(TX_BUFFER, 2);
        break;
    }
case 0x53://按键K11按下
    {
        Send_Command( Dis31); //第3位数码管显示1
        Send_Command( Dis20); //第2位数码管显示0
        Send_Command( Dis12); //第1位数码管显示2
        Send_Command( Dis04); //第0位数码管显示4
    }

```

```

    U_Tem=1024;
    TValue=U_Tem>>8;
    TX_BUFFER[0]=(unsigned char)TValue;
    TX_BUFFER[1]=(unsigned char)U_Tem;
    UART0_PutFrame(TX_BUFFER, 2);
    break;
}
case 0x58://按键K12按下
{
    Send_Command( Dis32);//第3位数码管显示2
    Send_Command( Dis20);//第2位数码管显示0
    Send_Command( Dis14);//第1位数码管显示4
    Send_Command( Dis08);//第0位数码管显示8
    U_Tem=2048;
    TValue=U_Tem>>8;
    TX_BUFFER[0]=(unsigned char)TValue;
    TX_BUFFER[1]=(unsigned char)U_Tem;
    UART0_PutFrame(TX_BUFFER, 2);
    break;
}
case 0x59://按键K13按下
{
    Send_Command( Dis34);//第3位数码管显示4
    Send_Command( Dis20);//第2位数码管显示0
    Send_Command( Dis19);//第1位数码管显示9
    Send_Command( Dis06);//第0位数码管显示6
    U_Tem=4096;
    TValue=U_Tem>>8;
    TX_BUFFER[0]=(unsigned char)TValue;
    TX_BUFFER[1]=(unsigned char)U_Tem;
    UART0_PutFrame(TX_BUFFER, 2);
    break;
}
case 0x5A://按键K14按下
{
    Send_Command( Dis38);//第3位数码管显示8
    Send_Command( Dis21);//第2位数码管显示1
    Send_Command( Dis19);//第1位数码管显示9
    Send_Command( Dis02);//第0位数码管显示2
    U_Tem=8192;
    TValue=U_Tem>>8;
    TX_BUFFER[0]=(unsigned char)TValue;
    TX_BUFFER[1]=(unsigned char)U_Tem;
    UART0_PutFrame(TX_BUFFER, 2);
}

```

```

        break;
    }
    case 0x5B://按键K15按下
    {
        Send_Command( Dis39);//第3位数码管显示9
        Send_Command( Dis29);//第2位数码管显示9
        Send_Command( Dis19);//第1位数码管显示9
        Send_Command( Dis09);//第0位数码管显示9
        U_Tem=9999;
        TValue=U_Tem>>8;
        TX_BUFFER[0]=(unsigned char)TValue;
        TX_BUFFER[1]=(unsigned char)U_Tem;
        UART0_PutFrame(TX_BUFFER, 2);
        break;
    }
    default:
        break;
}
P2IFG&=~BIT0;
}

```

3. 注意事项及调试方法

(1) 实验时请注意以下问题:

在退出下载环境, 对该模块进行操作时, 一定要先按一下复位键; 在将功能模块及msp430LaunchPad 插到主板上时, 一定要确保主板电源断开; 在将模块旁边的拨码开关打开时确保主板电源处于断开状态; 三种通信接口模块拨动开关一定要拨到RS232端。

(2) 调试方法

调试模式下, 分别在按键中断处理函数各按键对应代码处设置断点, 全速执行, 按下相应按键, 观察程序是否会停在断点处, 若可以, 表明按键解析正确。

在观察窗口中加入串口发送缓冲数组, 在调用串口发送函数处设置断点, 观察数组中的数据是否为我们要发送的。

在调用显示语句处设置断点, 将存放显示值的变量加入观察窗口, 观察变量值是否正确; 在观察窗口修改变量值, 全速执行, 观察显示结果是否正确。

4. 实验结果

数码管显示及PC机界面显示效果图如图6.4.3和图6.4.4所示。



图 6.4.3 实际显示效果（一）



图 6.4.4 实际显示效果（二）

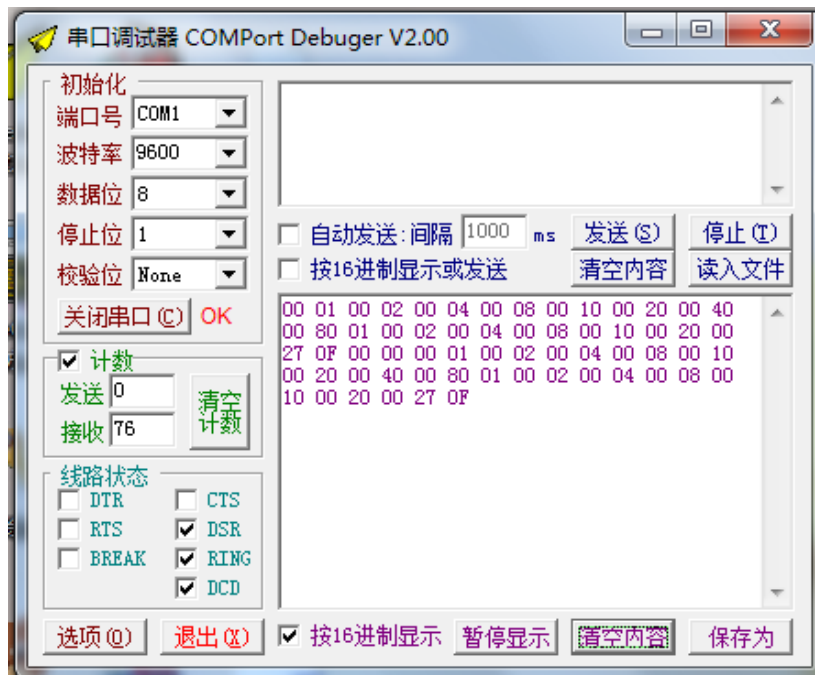


图 6.4.5 PC 机接收数据显示界面

6.4.4 总结与扩展

该实践项目结合了工程中最常用的输入方式—矩阵键盘,显示方式—数码管显示以及通信方式—串口通信,通过该项目的学习,我们熟练掌握了以上三项内容的软硬件设计和调试方法,了解了系统各部分的软件构架方式,为实际应用打下良好基础。由于数码管位数的限制和矩阵键盘大小的限制,本实验仅能显示 16 种四位数以下的键值,无法满足实际应用需求,可通过增加驱动芯片增加数码管位数及按键个数,构建功能更为全面的输入输出及通信外设。

最后给出两个练习题:

1. 本实验中是直接在数码管上显示键值,并通过串口将键值发送到 PC 机,请尝试将矩阵键盘设计成小键盘,分字符键和功能键,在数码管上显示最后输入的四个字符(仅限数字和字母),按发送键将先前输入的数据发送到 PC 机上(数据长度一定,旧的被新的覆盖)。
2. 可尝试反向显示,即将从 PC 机收来的数据显示在数码管上。

第五节 RS-485 接口多机通信

利用两套或多套实验系统构成多单片机 RS-485 接口通信系统,可在一体化实验标准板上插接三种温度测量模块+三种通信接口模块构建,通过 RS-485 接口的互连与通信,在点阵 LCD 显示器上实现测量数据的互换显示。

6.5.1 目的与要求

现在一般的单片机构成的应用都需要拥有一定的通信功能(包括与 PC 机或多个模块之间的通信),而在 PC 机上一般都拥有 RS-232 标准串行口,而单片机的串行口一般都是 TTL 电平。但是普通的 TTL 电路都有以下缺点驱动能力差,输入电阻小,灵敏度不高以及抗干扰性能差,因此它传输信号的距离相对较短。另有一个问题是信号地,RS-232 电气连接方法在许多场合是能正常工作的,但却埋下了很大的隐患,这有二个原因:(1)共模干扰问题:RS-485 接口采用差分方式传输信号,并不需要相对于某个参照点来检测信号,系统只需检测两线之间的电位差就可以了。但人们往往忽视了收发器有一定的共模电压范围,RS-485 收发器共模电压范围为 $-7\sim+12\text{V}$,只有满足上述条件,整个网络才能正常工作。当网络线路中共模电压超出此范围时就会影响通信的稳定可靠,甚至损坏接口。(2)EMI(电磁兼容性)问题:发送驱动器输出信号中的共模部分需要一个返回通路,如没有一个低阻的返回通道(信号地),信号中的共模部分就会以辐射的形式返回源端,整个总线就会像一个巨大的天线向外辐射电磁波。同时 RS-485 通讯标准接口电路会在发送端驱动器就会将 TTL 电平信号转化成差分信号输出;在接收端再将差分信号转变回 TTL 电平,因此具备了抗共模干扰的能力,弥补了 RS-232 的先天不足,使之传输效率大大提高,传输距离也因此提升。

下面简单的介绍一下 RS-485 接口。RS485 接口组成的半双工网络,一般是两线制(以前有四线制接法,只能实现点对点的通信方式,现很少采用),多采用屏蔽双绞线传输。这种接线方式为总线式拓扑结构在同一总线上最多可以挂接 32 个结点。在 RS485 通信网络中一般采用的是主从通信方式,即一个主机带多个从机。很多情况下,连接 RS-485 通信链路时只是简单地用一对双绞线将各个接口的“A”、“B”端连接起来。RS485 接口连接器采用 DB-9 的 9 芯插头座,与智能终端 RS485 接口采用 DB-9(孔),与键盘连接的键盘接口 RS485 采用 DB-9(针)。

由于现在 PC 机默认的只可能带有 RS232 接口,有两种方法可以得到 PC 上位机的 RS485 电路:(1)通过 RS232/RS485 转换电路将 PC 机串口 RS232 信号转换成 RS485 信号,对于情况比较复杂的工业环境最好是选用防浪涌带隔离珊的产品。(2)通过 PCI 多串口卡,可以直接选用输出信号为 RS485 类型的扩展卡。

6.5.2 电路设计与系统连接

本节主要介绍的是利用两套或多套实验系统构成多单片机 RS-485 接口通信系统,可在一体化实验标准板上插接三种温度测量模块+三种通信接口模块构建,通过 RS-485 接口的互连与通信,在点阵 LCD 显示器上实现测量数据的互换显示。在由单片机构成的多机串行通信系统中,一般采用两种通信结构,一种是星状结构,另一种则是网状结构。星状结构也称主从式,即从机不主动发送命令或数据,一切都由主机控制。并且在一个多机通信系统中,只有一台单机作为主机,各台从机之间不能相互通讯,即使有信息交换也必须通过主机转发。网状结构又称多主式。其又分为全连接网状和不完全连接网状两种形式。全连接网状中,每一个节点和网中其它节点均有链路连接。不完全连接网中,两节点之间不一定有直接链路连接,它们之间的通信,依靠其它节点转接。这种网络的优点是节点间路径多,碰撞和阻塞可大大减少,局部的故障不会影响整个网络的正常工作,可靠性高;网络扩充和主机入网比较

灵活、简单。但这种网络关系复杂，建网不易，网络控制机制复杂。广域网中一般用不完全连接网状结构。第一种模式通信是完全由主机同步，实现起来较为简单可靠，对实时性要求不高节点数不是太多的情况比较适合，而后一种则是每台主机都可以主动发送数据，不过实施起来对协议要求相对较高在大系统且实时性要求相对严格的情况下适合使用

在本节中我们主要讲一下星状结构和网络结构中的全连接网状结构。而相对于这两种结构在本实践课题的系统电路和系统与系统直接的电路连接方法是完全相同的且非常简单，这主要是因为 RS-485 本身的特性所决定的。即在总线末端接一个匹配电阻，吸收总线上的反射信号，保证正常传输信号干净、无毛刺。匹配电阻的取值应该与总线的特性阻抗相当。具体连接方式图如下。

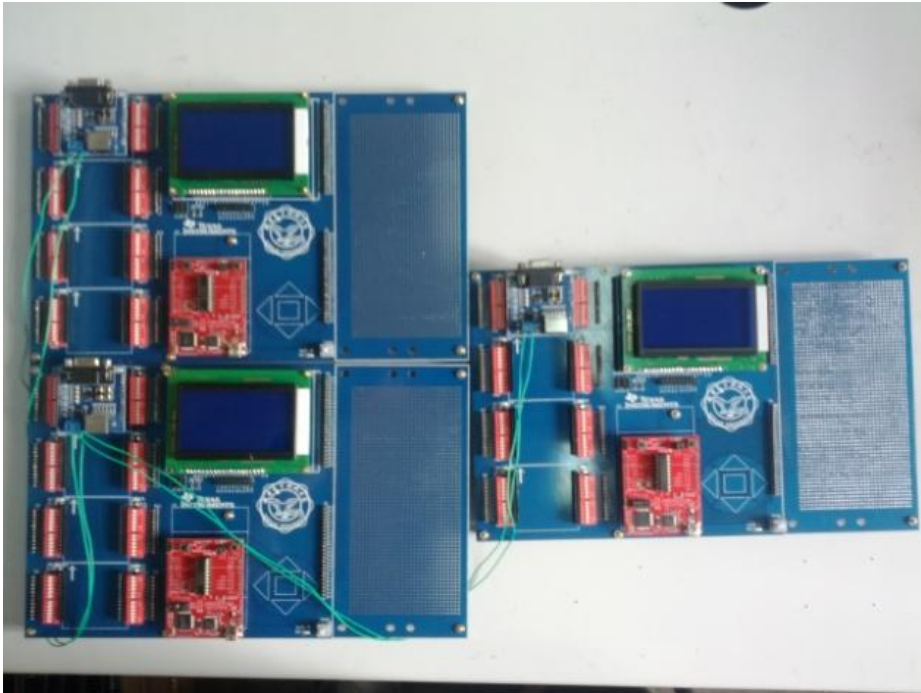


图 6.5.1 采用 RS-485 构成的多机通讯

当总线上没有信号传输时，总线处于悬浮状态，容易受干扰信号的影响。将总线上差分信号的正端 A+和+5V 电源间加一个上拉电阻；正端 A+和负端 B-间接一个电阻；负端 B-和地间接一个下拉电阻，三个电阻一样，形成一个电阻网络。当总线上没有信号传输时，A+和 B-的电压相对稳定，即使有干扰信号，却很难产生串行通信的起始信号 0，这样可以增加了总线抗干扰的能力。

6.5.3 软件设计与调试

在实际应用过程中我们要分情况选用星状结构或者是网状结构，第一种模式通信是完全由主机同步，实现起来较为简单可靠，对实时性要求不高节点数不是太多的情况比较适合，而后一种则是每台主机都可以主动发送数据，不过实施起来对协议要求相对较高在大系统且实时性要求相对严格的情况下适合使用。

下面我们从相对功能比较全面的网状网络结构开始讲起，该种结构实现了每个模块都可以接收到另一个模块的信息，同时根据自己的需要决定是否接受信息。首先我们第一步就是要对我们的单片机进行初始化：

```
void uartInit(void)
{
    UCAOCTL1 |= UCSWRST;    // USCI_A0 进入软件复位状态
```

```

UCAOCTL1 |= UCSSEL_2; // 时钟源选择 SMCLK
UCAOBRO = 104; // 时钟源 1MHz 时波特率为 9600
UCAOBR1 = 0; // 时钟源 1MHz 时波特率为 9600
UCAOMCTL = UCBSR0; // 小数分频器
UCAOCTL1 &= ~UCSWRST; // 初始化 USCI_A0 状态机
IE2 |= UCAORXIE; // 使能 USCI_A0 接收中断
}
/*时钟系统设置*/
BCSCTL1 = CALBC1_1MHZ; //设置 DCO 频率为 1MHz
DCOCTL = CALDCO_1MHZ; //设置 DCO 频率为 1MHz

/*UCSI (UART 模式)管脚初始化*/
P1SEL = BIT1 + BIT2 ; // P1.1 = RXD, P1.2=TXD
P1SEL2 = BIT1 + BIT2 ; // P1.1 = RXD, P1.2=TXD
P1DIR |= BIT0; // P1.0 做接收指示

uartInit(); // UCSI (UART 模式) 寄存器初始化

status = START; // 状态机初始化
uartCount = 0; // 计数器清零
flag = 0; // 帧标志清零

```

之后就是编写一段简单地协议，实现我们需要的功能，而为了实现主机在需要从机数据时向从机发出命令这个功能，我们主要要对串口发送和接收程序进行编写，先在发送程序中加入本模块的地址，再在接收程序中加入对地址的识别而接收程序我们直接在串口中断中执行。

发送程序:

```

/* 帧发送函数 */
void sendFrame(void)
{
    uartBuf[0] = 0x23; //假设目的地址为 0x23
    uartBuf[1] = LOCAL_ADDR; //本机地址

    unsigned char i;

    for(i=2;i<LENGTH;i++) //填充数据(作为示例,随意填写)
    {
        uartBuf[i] = i;
    }

    /* 循环 LENGTH 次, 将一个帧完整的发送出去 */
    for(i=0;i<LENGTH;i++)
    {
        UCA0TXBUF = uartBuf[i]; //将待发送的字节装入 UCA0TXBUF
        while (!(IFG2&UCA0TXIFG)); //等待这个字节发送完毕
    }
}

```

```

}
}

```

接收程序:

```

/*串口接收中断*/
#pragma vector=USCIABORX_VECTOR
__interrupt void USCI0RX_ISR(void)
{
    switch(status)
    {
        case START: //如果是“开始状态”
        {
            uartBuf[0] = UCA0RXBUF; //这是帧的第一个字节（目的地址）
            if(uartBuf[0]== LOCAL_ADDR) //如果
            {
                status = RECEIVE; //转换为“接收状态”
                uartCount++; //计数器加 1
            }
            else //如果目的地址与本机地址不相符
                status = START; //状态不变
            break;
        }
        case RECEIVE: //如果是“接收状态”
        {
            uartBuf[uartCount] = UCA0RXBUF; //将收到的字节装入缓存
            uartCount++; //计数器加 1
            if(uartCount == LENGTH) //如果已经达到最大帧长度
            {
                status = START; //状态转换为“开始状态”
                flag = 0xff; //设置“收到一帧”标志
                uartCount = 0; //计数器清零
            }
            break;
        }
        default: break;
    }
}

```

以上就是一个简单的星状网络程序的示例，该示例主要写了一个模块只接受另一个模块数据的功能，在发射的数据中第一个字节为目的地址（即需要接受信息的模块的物理地址），第二个字节则表示自己的物理地址，之后的字节则是相关的数据（本程序为固定字节长度，如字节长度不固定，则需在第三字节加入接收数据字节长度，方便接收模块接收数据），如果需要接收多个程序的信息则只需在接收中断中加入对其他模块地址接收信息的程序就可以了。在该程序中每个模块都一直在被动接收状态，只有在需要发送数据时才更改状态，发送结束后回到接收状态。

而主从模式就是主机则只需发送目的地址加上数据就可以了，而从模块则只是被动接收

数据，或者在接收到数据后进行一个数据回馈即可。

如果系统需要的功能比较复杂则需要将程序改为状态机形式进行编写。本节在此就不做细表了。

第六节 PS2 接口键盘输入与 LCD 显示

6.6.1 目的与要求

学习 MSP430G2553 单片机内部 I/O 中断的使用方法，了解 PS2 协议，掌握利用 I/O 中断解析 PS2 键盘等设备的原理和方法。在这一节中，要求用三种通信接口模块+点阵 LCD 显示器+标准键盘实现键盘输入值的显示，通过标准键盘输入键值，并在点阵 LCD 显示器上显示出对应的键值。

6.6.2 电路设计与系统连接

由于本实践课题要求通过标准键盘输入键值，并在点阵 LCD 显示器上显示出对应的键值，故系统由三种通信接口模块+点阵 LCD 显示器+标准键盘组成。信号线连接图与第四章给出的三种通信接口模块连接方式相同，PS2 接口直接按正确方式插好即可（接头有防插错设计）连接好的实物图如图 6.6.1 所示。

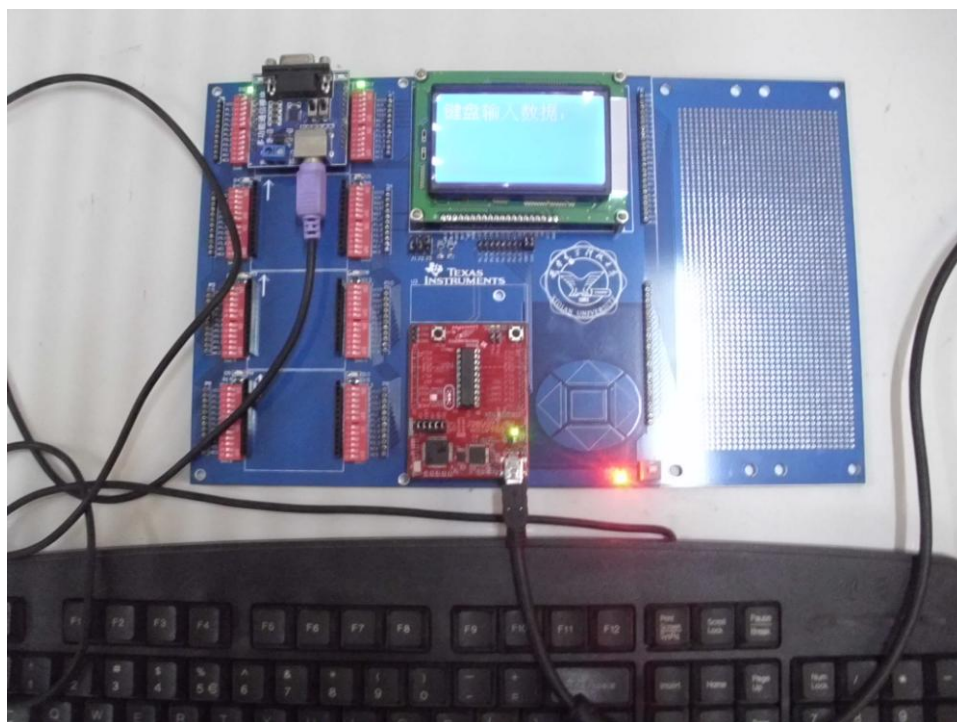


图 6.6.1 连接实物图

6.6.3 信号与数据处理

本实验的关键信号为 PS2 键盘按键按下产生的时钟和数据信号。在此，先介绍一下 PS2 通信协议及 PS2 鼠标、键盘原理。

1. PS2 鼠标和接口协议简介

(1) 接口的物理特性

PS/2 接口用于许多现代的鼠标和键盘，由 IBM 最初开发和使用。物理上的 PS/2 接口有两种类型的连接器：5 脚的 DIN 和 6 脚的 mini-DIN。图 6.6.2 是两种连接器的引脚定义。使用中，主机提供+5V 电源给鼠标，鼠标的地连接到主机电源地上。

	插座(孔)	插头(针)		5脚的DIN	6脚的mini-DIN
5脚的DIN			1	时钟 (CLOCK)	数据 (DATA)
			2	数据 (DATA)	未实现、保留
			3	未实现、保留	电源地 (GND)
6脚的mini-DIN			4	电源地 (GND)	电源+5V (VCC)
			5	电源+5V (VCC)	时钟 (CLOCK)
			6		未实现、保留

图 6.6.2 PS2 接口连接器引脚定义

(2) 接口协议原理

PS/2 鼠标接口采用一种双向同步串行协议。即每在时钟线上发一个脉冲,就在数据线上发送一位数据。在相互传输中,主机拥有总线控制权,即它可以在任何时候抑制鼠标的发送。方法是把时钟线一直拉低,鼠标就不能产生时钟信号和发送数据。在两个方向的传输中,时钟信号都是由鼠标产生,即主机不产生通信时钟信号。

如果主机要发送数据,它必须控制鼠标产生时钟信号。方法如下:主机首先下拉时钟线至少 100 μ s 抑制通信,然后再下拉数据线,最后释放时钟线。通过这一时序控制鼠标产生时钟信号。当鼠标检测到这个时序状态,会在 10ms 内产生时钟信号。如图 3 中 A 时序段。主机和鼠标之间,传输数据帧的时序如图 2、图 3 所示。2.2 数据包结构在主机程序中,利用每个数据位的时钟脉冲触发中断,在中断例程中实现数据位的判断和接收。在实验过程中,通过合适的编程,能够正确控制并接收鼠标数据。但该方案有一点不足,由于每个 CLOCK 都要产生一次中断,中断频繁,需要耗用大量的主机资源。

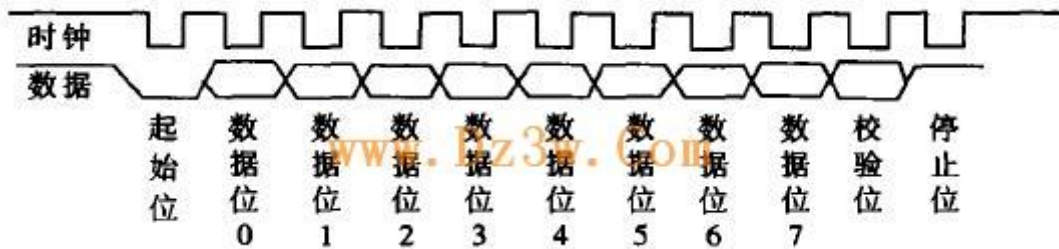


图 6.6.3 鼠标到主机的传输时序

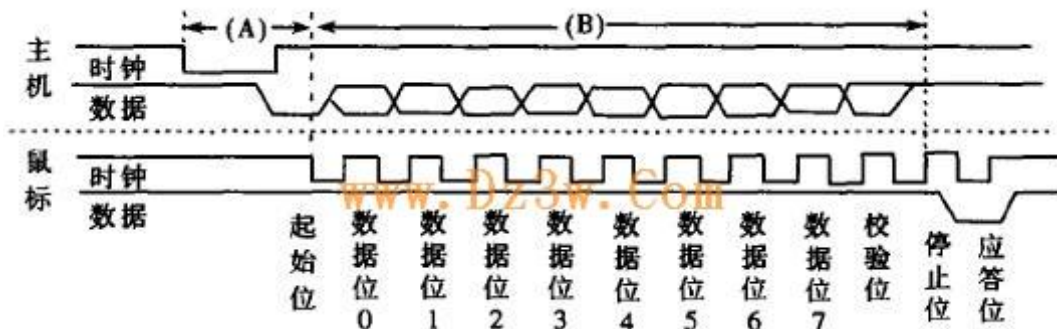


图 6.6.4 主机到鼠标的传输时序

2. PS2 键盘简介

键盘上包含了一个大型的按键矩阵，它们是由安装在电路板上的处理器（叫做键盘编码器）来监视的。具体的处理器在键盘与键盘之间是多样化的，但好似它们基本上都做着同样的事情：监视哪些按键被按下或释放了，并在适当的时候传送到主机，如果有必要，处理器处理所有的去抖动并在它的 16 字节缓冲区里缓冲数据。你的主板包含了一个键盘控制器，负责解码所有来自键盘的数据，并告诉你的软件什么事件发生了，在主机和键盘之间的通讯使用 IBM 的协议。PS/2 键盘使用了与 PS/2 鼠标一样的协议，具体请参考上一节内容。

键盘的处理器花费很多的时间来扫描或监视按键矩阵 如果它发现有键被按下、释放或按住，键盘将发送“扫描码”的信息包到计算机，扫描码有两种不同的类型：“通码”和“断码”，当一个键被按下或按住就发送通码，当一个键被释放就发送断码。每个按键被分配了唯一的通码和断码，这样主机通过查找唯一的扫描码就可以测定是哪个按键，每个键一整套的通断码组成了“扫描码集”。有三套标准的扫描码集，分别是第一套、第二套和第三套。所有现代的键盘默认使用第二套扫描码。感兴趣的读者可查看相关资料（第一套第二套键盘扫描码），这里不多做介绍。

正如键按下通码就被发往计算机一样，只要键释放，断码就会被发送。每个键都有它自己唯一的通码，它们也都有唯一的断码。幸运的是，你不用总是通过查表来找出按键的断码，通码和断码之间存在着必然的联系：多数第二套断码有两字节长，它们的第一个字节是 F0h 第二个字节是这个键的通码。

简单地说，每按下一个按键，会产生三组波形：时钟线上是每组长为 11 位方波；数据线上为每组 11 位（包括 1 位起始位（为 0），8 位数据位，1 个校验位，1 个停止位（为高））的数据，第一组数据部分为通码，第二组和第三组为断码，其中，第二组数据部分为 F0h，第三组数据部分也为通码。

本实验例程中，为简化键盘解析，仅解析第三组波形（取全部 11 位）作为每个键的标识，这就是代码中的 3 和 11 的由来。读者若感兴趣，可对键盘做出更精确的解析（可更换一款内存更大的单片机）。

3. 信号处理方法——使用普通 I/O 解析键盘波形

PS2 键盘是自行产生 CLK 时钟信号的 PS2 设备，所以只需给键盘供电，单片机只需简单地读取波形即可。本模块中，键盘的时钟和数据线分别接于单片机 P1.5、P1.6。这两个 I/O 口设置为普通 I/O，输入模式；开启 P1.5（即时钟信号）的中断，设为下降沿触发。代码见 PS2 初始化函数。在中断处理函数中读取数据线上的数据，11 位移位存储在变量中。然后通过该变量值，判断按下的是哪个键。因为开启的是时钟线的中断，时钟线上是规则的方波信号，所以每按下一个按键，会产生 33 次 I/O 中断。每 11 次作为一个键值，忽略前两个键值，进去第三个键值有效。具体做法可参见代码。

4. 数据处理流程图及例程

（1）数据处理流程

数据处理流程如图 6.6.5 所示，单片机监测键盘波形输入，一旦键盘有键被按下，则会触发 I/O 中断，在 I/O 中断处理函数中对数据线上的波形进行移位存储，得到键值。最后再在主循环中将键值转换为相应字母、数字或功能键在 LCD 上给出相应显示。

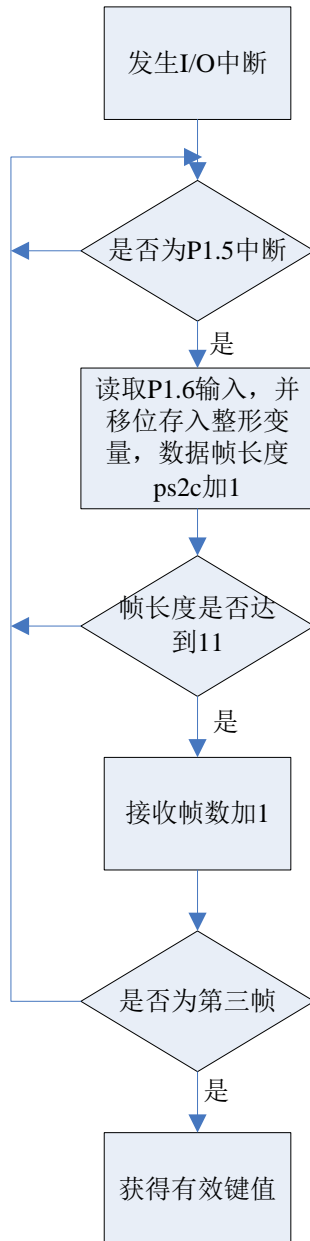


图 6.6.5 数据处理流程

(2) 相关例程

1) PS2 接口初始化函数:

```

void ConfigPS2(void)
{
    P1SEL&=~(BIT5+BIT6); //P1.5、P1.6配置为普通I/O
    P1SEL2&=~(BIT5+BIT6); //P1.5、P1.6配置为普通I/O
    P1DIR&=~(BIT5+BIT6); //P1.5、P1.6配置为输入
    P1IE|=BIT5;           //开启P1.5中断
    P1IES|=BIT5;         //P1.5配置为下降沿中断
}
  
```

2) I/O 中断处理函数


```

#pragma vector=PORT1_VECTOR
__interrupt void PORT1 (void)
{
    if(P1IFG&BIT5)
    {
        P1IFG&=~BIT5;
        if((P1IN&BIT6) !=0)
            PS2DATA|=(0x0001<<ps2c);

        ps2c++;//每帧数据长度为11
        if(ps2c>=11)
        {
            ps2c=0;
            j++;//取第三组为有用数据
            if(j>=3)
            {
                PS2Temp=PS2DATA;
                i++;//数组位置计数
                j=0;
                stat=0;
                if(i==(ShiftFlag+1))
                    Shift=0;
            }
            PS2DATA=0;

            if(i>=17)
            {
                if(PS2Temp!=1740)
                {
                    Txflag=0;
                    i=1;
                    crow++;
                    if(crow>=3)
                    {
                        crow=0;
                        FirstIn=0;
                    }
                    for(k=0;k<16;k++)
                    {
                        if(crow==0)
                        {
                            TxData[k]=PS2Data3[k];
                            PS2Data1[k]=' ';
                        }
                    }
                }
            }
        }
    }
}

```



```

void main(void)
{
    PS2DATA=0;
    ps2c=0;
    PS2Temp=0;
    stat=0;
    i=0, j=0;
    Shift=0, Caps=0, ShiftFlag=20;
    ConfigWDT();
    ConfigClocks();
    ConfigPS2();
    ConfigTimerA2();
    ConfigLCD();
    clear();
    __enable_interrupt();           // Enable interrupts.

    while(1)
    {
        switch(PS2Temp)
        {
            case 1080://A
                if(((Shift==0)&&(Caps==0)) || ((Shift!=0)&&(Caps!=0)))
                {
                    if(crow==0)
                        PS2Data1[i-1]='a';
                    else if(crow==1)
                        PS2Data2[i-1]='a';
                    else if(crow==2)
                        PS2Data3[i-1]='a';
                }
                else
                {
                    if(crow==0)
                        PS2Data1[i-1]='A';
                    else if(crow==1)
                        PS2Data2[i-1]='A';
                    else if(crow==2)
                        PS2Data3[i-1]='A';
                }
                break;

            case 1124://B
                if(((Shift==0)&&(Caps==0)) || ((Shift!=0)&&(Caps!=0)))
                {

```

```

        if(crow==0)
            PS2Data1[i-1]=' b' ;
        else if(crow==1)
            PS2Data2[i-1]=' b' ;
        else if(crow==2)
            PS2Data3[i-1]=' b' ;
    }
    else
    {
        if(crow==0)
            PS2Data1[i-1]=' B' ;
        else if(crow==1)
            PS2Data2[i-1]=' B' ;
        else if(crow==2)
            PS2Data3[i-1]=' B' ;
    }
    break;

case 1602://C
    if(((Shift==0)&&(Caps==0)) || ((Shift!=0)&&(Caps!=0)))
    {
        if(crow==0)
            PS2Data1[i-1]=' c' ;
        else if(crow==1)
            PS2Data2[i-1]=' c' ;
        else if(crow==2)
            PS2Data3[i-1]=' c' ;
    }
    else
    {
        if(crow==0)
            PS2Data1[i-1]=' C' ;
        else if(crow==1)
            PS2Data2[i-1]=' C' ;
        else if(crow==2)
            PS2Data3[i-1]=' C' ;
    }
    break;

case 1094://D
    if(((Shift==0)&&(Caps==0)) || ((Shift!=0)&&(Caps!=0)))
    {
        if(crow==0)
            PS2Data1[i-1]=' d' ;

```

```

        else if(crow==1)
            PS2Data2[i-1]=' d' ;
        else if(crow==2)
            PS2Data3[i-1]=' d' ;
    }
    else
    {
        if(crow==0)
            PS2Data1[i-1]=' D' ;
        else if(crow==1)
            PS2Data2[i-1]=' D' ;
        else if(crow==2)
            PS2Data3[i-1]=' D' ;
    }
    break;

case 1608://E
    if(((Shift==0)&&(Caps==0)) || ((Shift!=0)&&(Caps!=0)))
    {
        if(crow==0)
            PS2Data1[i-1]=' e' ;
        else if(crow==1)
            PS2Data2[i-1]=' e' ;
        else if(crow==2)
            PS2Data3[i-1]=' e' ;
    }
    else
    {
        if(crow==0)
            PS2Data1[i-1]=' E' ;
        else if(crow==1)
            PS2Data2[i-1]=' E' ;
        else if(crow==2)
            PS2Data3[i-1]=' E' ;
    }
    break;

case 1622://F
    if(((Shift==0)&&(Caps==0)) || ((Shift!=0)&&(Caps!=0)))
    {
        if(crow==0)
            PS2Data1[i-1]=' f' ;
        else if(crow==1)
            PS2Data2[i-1]=' f' ;
    }

```

```

        else if(crow==2)
            PS2Data3[i-1]=' f' ;
    }
    else
    {
        if(crow==0)
            PS2Data1[i-1]=' F' ;
        else if(crow==1)
            PS2Data2[i-1]=' F' ;
        else if(crow==2)
            PS2Data3[i-1]=' F' ;
    }
break;

case 1128://G
    if(((Shift==0)&&(Caps==0)) || ((Shift!=0)&&(Caps!=0)))
    {
        if(crow==0)
            PS2Data1[i-1]=' g' ;
        else if(crow==1)
            PS2Data2[i-1]=' g' ;
        else if(crow==2)
            PS2Data3[i-1]=' g' ;
    }
    else
    {
        if(crow==0)
            PS2Data1[i-1]=' G' ;
        else if(crow==1)
            PS2Data2[i-1]=' G' ;
        else if(crow==2)
            PS2Data3[i-1]=' G' ;
    }
break;

case 1638://H
    if(((Shift==0)&&(Caps==0)) || ((Shift!=0)&&(Caps!=0)))
    {
        if(crow==0)
            PS2Data1[i-1]=' h' ;
        else if(crow==1)
            PS2Data2[i-1]=' h' ;
        else if(crow==2)
            PS2Data3[i-1]=' h' ;
    }

```

```

    }
    else
    {
        if(crow==0)
            PS2Data1[i-1]=' H' ;
        else if(crow==1)
            PS2Data2[i-1]=' H' ;
        else if(crow==2)
            PS2Data3[i-1]=' H' ;
    }
break;

case 1158://I
    if(((Shift==0)&&(Caps==0)) || ((Shift!=0)&&(Caps!=0)))
    {
        if(crow==0)
            PS2Data1[i-1]=' i' ;
        else if(crow==1)
            PS2Data2[i-1]=' i' ;
        else if(crow==2)
            PS2Data3[i-1]=' i' ;
    }
    else
    {
        if(crow==0)
            PS2Data1[i-1]=' I' ;
        else if(crow==1)
            PS2Data2[i-1]=' I' ;
        else if(crow==2)
            PS2Data3[i-1]=' I' ;
    }
break;

case 1142://J
    if(((Shift==0)&&(Caps==0)) || ((Shift!=0)&&(Caps!=0)))
    {
        if(crow==0)
            PS2Data1[i-1]=' j' ;
        else if(crow==1)
            PS2Data2[i-1]=' j' ;
        else if(crow==2)
            PS2Data3[i-1]=' j' ;
    }
    else

```

```

    {
        if(crow==0)
            PS2Data1[i-1]='J' ;
        else if(crow==1)
            PS2Data2[i-1]='J' ;
        else if(crow==2)
            PS2Data3[i-1]='J' ;
    }
break;

case 1668://K
    if(((Shift==0)&&(Caps==0)) || ((Shift!=0)&&(Caps!=0)))
    {
        if(crow==0)
            PS2Data1[i-1]='k' ;
        else if(crow==1)
            PS2Data2[i-1]='k' ;
        else if(crow==2)
            PS2Data3[i-1]='k' ;
    }
    else
    {
        if(crow==0)
            PS2Data1[i-1]='K' ;
        else if(crow==1)
            PS2Data2[i-1]='K' ;
        else if(crow==2)
            PS2Data3[i-1]='K' ;
    }
break;

case 1686://L
    if(((Shift==0)&&(Caps==0)) || ((Shift!=0)&&(Caps!=0)))
    {
        if(crow==0)
            PS2Data1[i-1]='l' ;
        else if(crow==1)
            PS2Data2[i-1]='l' ;
        else if(crow==2)
            PS2Data3[i-1]='l' ;
    }
    else
    {
        if(crow==0)

```



```

        PS2Data1[i-1]=' L' ;
    else if(crow==1)
        PS2Data2[i-1]=' L' ;
    else if(crow==2)
        PS2Data3[i-1]=' L' ;
}
break;

case 1652://M
    if(((Shift==0)&&(Caps==0)) || ((Shift!=0)&&(Caps!=0)))
    {
        if(crow==0)
            PS2Data1[i-1]=' m' ;
        else if(crow==1)
            PS2Data2[i-1]=' m' ;
        else if(crow==2)
            PS2Data3[i-1]=' m' ;
    }
    else
    {
        if(crow==0)
            PS2Data1[i-1]=' M' ;
        else if(crow==1)
            PS2Data2[i-1]=' M' ;
        else if(crow==2)
            PS2Data3[i-1]=' M' ;
    }
break;

case 1122://N
    if(((Shift==0)&&(Caps==0)) || ((Shift!=0)&&(Caps!=0)))
    {
        if(crow==0)
            PS2Data1[i-1]=' n' ;
        else if(crow==1)
            PS2Data2[i-1]=' n' ;
        else if(crow==2)
            PS2Data3[i-1]=' n' ;
    }
    else
    {
        if(crow==0)
            PS2Data1[i-1]=' N' ;
        else if(crow==1)

```

```

        PS2Data2[i-1]=' N' ;
    else if(crow==2)
        PS2Data3[i-1]=' N' ;
    }
break;

case 1672://0
    if(((Shift==0)&&(Caps==0)) || ((Shift!=0)&&(Caps!=0)))
    {
        if(crow==0)
            PS2Data1[i-1]=' o' ;
        else if(crow==1)
            PS2Data2[i-1]=' o' ;
        else if(crow==2)
            PS2Data3[i-1]=' o' ;
    }
    else
    {
        if(crow==0)
            PS2Data1[i-1]=' 0' ;
        else if(crow==1)
            PS2Data2[i-1]=' 0' ;
        else if(crow==2)
            PS2Data3[i-1]=' 0' ;
    }
break;

case 1690://P
    if(((Shift==0)&&(Caps==0)) || ((Shift!=0)&&(Caps!=0)))
    {
        if(crow==0)
            PS2Data1[i-1]=' p' ;
        else if(crow==1)
            PS2Data2[i-1]=' p' ;
        else if(crow==2)
            PS2Data3[i-1]=' p' ;
    }
    else
    {
        if(crow==0)
            PS2Data1[i-1]=' P' ;
        else if(crow==1)
            PS2Data2[i-1]=' P' ;
        else if(crow==2)

```

```

        PS2Data3[i-1]=' P' ;
    }
break;

case 1066://Q
    if(((Shift==0)&&(Caps==0)) || ((Shift!=0)&&(Caps!=0)))
    {
        if(crow==0)
            PS2Data1[i-1]=' q' ;
        else if(crow==1)
            PS2Data2[i-1]=' q' ;
        else if(crow==2)
            PS2Data3[i-1]=' q' ;
    }
    else
    {
        if(crow==0)
            PS2Data1[i-1]=' Q' ;
        else if(crow==1)
            PS2Data2[i-1]=' Q' ;
        else if(crow==2)
            PS2Data3[i-1]=' Q' ;
    }
break;

case 1626://R
    if(((Shift==0)&&(Caps==0)) || ((Shift!=0)&&(Caps!=0)))
    {
        if(crow==0)
            PS2Data1[i-1]=' r' ;
        else if(crow==1)
            PS2Data2[i-1]=' r' ;
        else if(crow==2)
            PS2Data3[i-1]=' r' ;
    }
    else
    {
        if(crow==0)
            PS2Data1[i-1]=' R' ;
        else if(crow==1)
            PS2Data2[i-1]=' R' ;
        else if(crow==2)
            PS2Data3[i-1]=' R' ;
    }
}

```

```

break;

case 1590://S
    if(((Shift==0)&&(Caps==0)) || ((Shift!=0)&&(Caps!=0)))
    {
        if(crow==0)
            PS2Data1[i-1]=' s' ;
        else if(crow==1)
            PS2Data2[i-1]=' s' ;
        else if(crow==2)
            PS2Data3[i-1]=' s' ;
    }
    else
    {
        if(crow==0)
            PS2Data1[i-1]=' S' ;
        else if(crow==1)
            PS2Data2[i-1]=' S' ;
        else if(crow==2)
            PS2Data3[i-1]=' S' ;
    }
break;

case 1112://T
    if(((Shift==0)&&(Caps==0)) || ((Shift!=0)&&(Caps!=0)))
    {
        if(crow==0)
            PS2Data1[i-1]=' t' ;
        else if(crow==1)
            PS2Data2[i-1]=' t' ;
        else if(crow==2)
            PS2Data3[i-1]=' t' ;
    }
    else
    {
        if(crow==0)
            PS2Data1[i-1]=' T' ;
        else if(crow==1)
            PS2Data2[i-1]=' T' ;
        else if(crow==2)
            PS2Data3[i-1]=' T' ;
    }
break;

```

```

case 1656://U
    if((Shift==0)&&(Caps==0) || ((Shift!=0)&&(Caps!=0)))
    {
        if(crow==0)
            PS2Data1[i-1]=' u' ;
        else if(crow==1)
            PS2Data2[i-1]=' u' ;
        else if(crow==2)
            PS2Data3[i-1]=' u' ;
    }
    else
    {
        if(crow==0)
            PS2Data1[i-1]=' U' ;
        else if(crow==1)
            PS2Data2[i-1]=' U' ;
        else if(crow==2)
            PS2Data3[i-1]=' U' ;
    }
break;

```

```

case 1108://V
    if((Shift==0)&&(Caps==0) || ((Shift!=0)&&(Caps!=0)))
    {
        if(crow==0)
            PS2Data1[i-1]=' v' ;
        else if(crow==1)
            PS2Data2[i-1]=' v' ;
        else if(crow==2)
            PS2Data3[i-1]=' v' ;
    }
    else
    {
        if(crow==0)
            PS2Data1[i-1]=' V' ;
        else if(crow==1)
            PS2Data2[i-1]=' V' ;
        else if(crow==2)
            PS2Data3[i-1]=' V' ;
    }
break;

```

```

case 1594://W
    if((Shift==0)&&(Caps==0) || ((Shift!=0)&&(Caps!=0)))

```

```

    {
        if(crow==0)
            PS2Data1[i-1]=' w' ;
        else if(crow==1)
            PS2Data2[i-1]=' w' ;
        else if(crow==2)
            PS2Data3[i-1]=' w' ;
    }
    else
    {
        if(crow==0)
            PS2Data1[i-1]=' W' ;
        else if(crow==1)
            PS2Data2[i-1]=' W' ;
        else if(crow==2)
            PS2Data3[i-1]=' W' ;
    }
    break;

case 1604://X
    if(((Shift==0)&&(Caps==0)) || ((Shift!=0)&&(Caps!=0)))
    {
        if(crow==0)
            PS2Data1[i-1]=' x' ;
        else if(crow==1)
            PS2Data2[i-1]=' x' ;
        else if(crow==2)
            PS2Data3[i-1]=' x' ;
    }
    else
    {
        if(crow==0)
            PS2Data1[i-1]=' X' ;
        else if(crow==1)
            PS2Data2[i-1]=' X' ;
        else if(crow==2)
            PS2Data3[i-1]=' X' ;
    }
    break;

case 1642://Y
    if(((Shift==0)&&(Caps==0)) || ((Shift!=0)&&(Caps!=0)))
    {
        if(crow==0)

```

```

        PS2Data1[i-1]=' y' ;
    else if(crow==1)
        PS2Data2[i-1]=' y' ;
    else if(crow==2)
        PS2Data3[i-1]=' y' ;
}
else
{
    if(crow==0)
        PS2Data1[i-1]=' Y' ;
    else if(crow==1)
        PS2Data2[i-1]=' Y' ;
    else if(crow==2)
        PS2Data3[i-1]=' Y' ;
}
break;

case 1076://Z
    if(((Shift==0)&&(Caps==0)) || ((Shift!=0)&&(Caps!=0)))
    {
        if(crow==0)
            PS2Data1[i-1]=' z' ;
        else if(crow==1)
            PS2Data2[i-1]=' z' ;
        else if(crow==2)
            PS2Data3[i-1]=' z' ;
    }
    else
    {
        if(crow==0)
            PS2Data1[i-1]=' Z' ;
        else if(crow==1)
            PS2Data2[i-1]=' Z' ;
        else if(crow==2)
            PS2Data3[i-1]=' Z' ;
    }
break;

case 1248:
    if(crow==0)
        PS2Data1[i-1]=' 0' ;
    else if(crow==1)
        PS2Data2[i-1]=' 0' ;
    else if(crow==2)

```

```
        PS2Data3[i-1]=' 0' ;  
break;
```

```
case 1746:  
    if(crow==0)  
        PS2Data1[i-1]=' 1' ;  
    else if(crow==1)  
        PS2Data2[i-1]=' 1' ;  
    else if(crow==2)  
        PS2Data3[i-1]=' 1' ;  
break;
```

```
case 1764:  
    if(crow==0)  
        PS2Data1[i-1]=' 2' ;  
    else if(crow==1)  
        PS2Data2[i-1]=' 2' ;  
    else if(crow==2)  
        PS2Data3[i-1]=' 2' ;  
break;
```

```
case 1268:  
    if(crow==0)  
        PS2Data1[i-1]=' 3' ;  
    else if(crow==1)  
        PS2Data2[i-1]=' 3' ;  
    else if(crow==2)  
        PS2Data3[i-1]=' 3' ;  
break;
```

```
case 1238:  
    if(crow==0)  
        PS2Data1[i-1]=' 4' ;  
    else if(crow==1)  
        PS2Data2[i-1]=' 4' ;  
    else if(crow==2)  
        PS2Data3[i-1]=' 4' ;  
break;
```

```
case 1254:  
    if(crow==0)  
        PS2Data1[i-1]=' 5' ;  
    else if(crow==1)  
        PS2Data2[i-1]=' 5' ;
```



```

        else if(crow==2)
            PS2Data3[i-1]=' 5' ;
break;

case 1768:
    if(crow==0)
        PS2Data1[i-1]=' 6' ;
    else if(crow==1)
        PS2Data2[i-1]=' 6' ;
    else if(crow==2)
        PS2Data3[i-1]=' 6' ;
break;

case 1752:
    if(crow==0)
        PS2Data1[i-1]=' 7' ;
    else if(crow==1)
        PS2Data2[i-1]=' 7' ;
    else if(crow==2)
        PS2Data3[i-1]=' 7' ;
break;

case 1258:
    if(crow==0)
        PS2Data1[i-1]=' 8' ;
    else if(crow==1)
        PS2Data2[i-1]=' 8' ;
    else if(crow==2)
        PS2Data3[i-1]=' 8' ;
break;

case 1658:
    if(crow==0)
        PS2Data1[i-1]=' 9' ;
    else if(crow==1)
        PS2Data2[i-1]=' 9' ;
    else if(crow==2)
        PS2Data3[i-1]=' 9' ;
break;

case 1266:
    if(crow==0)
        PS2Data1[i-1]=' +' ;
    else if(crow==1)

```

```

        PS2Data2[i-1]='+' ;
    else if(crow==2)
        PS2Data3[i-1]='+' ;
break;

case 2038:
    if(crow==0)
        PS2Data1[i-1]='-' ;
    else if(crow==1)
        PS2Data2[i-1]='-' ;
    else if(crow==2)
        PS2Data3[i-1]='-' ;
break;

case 1272:
    if(crow==0)
        PS2Data1[i-1]='*' ;
    else if(crow==1)
        PS2Data2[i-1]='*' ;
    else if(crow==2)
        PS2Data3[i-1]='*' ;
break;

case 1472:
    if(crow==0)
        PS2Data1[i-1]='/' ;
    else if(crow==1)
        PS2Data2[i-1]='/' ;
    else if(crow==2)
        PS2Data3[i-1]='/' ;
break;

case 1706:
    if(crow==0)
        PS2Data1[i-1]='=' ;
    else if(crow==1)
        PS2Data2[i-1]='=' ;
    else if(crow==2)
        PS2Data3[i-1]='=' ;
break;

case 1106://空格键
    if(crow==0)
        PS2Data1[i-1]=' ' ;

```

```

        else if(crow==1)
            PS2Data2[i-1]=' ';
        else if(crow==2)
            PS2Data3[i-1]=' ';
break;

case 1740://退格键
{
    if(stat==0)
    {
        i--;
        if(crow==0)
            PS2Data1[i-1]=' ';
        else if(crow==1)
            PS2Data2[i-1]=' ';
        else if(crow==2)
            PS2Data3[i-1]=' ';
        i--;
        stat=1;
    }
}
break;

case 1170:
    if(crow==0)
        PS2Data1[i-1]='.';
    else if(crow==1)
        PS2Data2[i-1]='.';
    else if(crow==2)
        PS2Data3[i-1]='.';
break;

case 1666:
    if(crow==0)
        PS2Data1[i-1]=',';
    else if(crow==1)
        PS2Data2[i-1]=',';
    else if(crow==2)
        PS2Data3[i-1]=',';
break;

case 1068:
    if(crow==0)
        PS2Data1[i-1]='!';

```

```

        else if(crow==1)
            PS2Data2[i-1]='!';
        else if(crow==2)
            PS2Data3[i-1]='!';
break;

case 1176:
    if(Shift==0)
    {
        if(crow==0)
            PS2Data1[i-1]='';
        else if(crow==1)
            PS2Data2[i-1]='';
        else if(crow==2)
            PS2Data3[i-1]='';
    }
    else
    {
        if(crow==0)
            PS2Data1[i-1]=': ';
        else if(crow==1)
            PS2Data2[i-1]=': ';
        else if(crow==2)
            PS2Data3[i-1]=': ';
    }
break;

case 1172:
    if(crow==0)
        PS2Data1[i-1]='?';
    else if(crow==1)
        PS2Data2[i-1]='?';
    else if(crow==2)
        PS2Data3[i-1]='?';
break;

case 1572://上档键
{
    if(stat==0)
    {
        Shift=1;
        ShiftFlag=i;
        i--;
        stat=1;
    }
}

```

```

    }
}
break;

case 1200://大写锁定
{
    if(stat==0)
    {
        Caps^=1;
        i--;
        stat=1;
    }
}
break;

case 1716://Enter, 界面切换
{
    if(stat==0)
    {
        menu^=1;
        FirstIn=0;
        i--;
        stat=1;
    }
}
break;

case 1576://Ctrl, 485方向切换, P1. 3, 0->输入;1->输出.
{
    if(stat==0)
    {
        P1OUT^=BIT3;
        i--;
        stat=1;
    }
}
break;

default:
break;
}

if(FirstIn==0)
{

```

```

        clear ();
        set_position(0,0);
send_str(dis_sbf0);

if(crow==0)
{
    set_position(1,0);
    send_str(PS2Data1);
}
else if(crow==1)
{
    set_position(1,0);
    send_str(PS2Data1);
    set_position(2,0);
    send_str(PS2Data2);
}
else if(crow==2)
{
    set_position(1,0);
    send_str(PS2Data1);
    set_position(2,0);
    send_str(PS2Data2);
    set_position(3,0);
    send_str(PS2Data3);
}
    FirstIn=1;
}
switch (crow)
{
    case 0:
        set_position(1,0);
        send_str(PS2Data1);
    break;

    case 1:
        set_position(2,0);
        send_str(PS2Data2);
    break;

    case 2:
        set_position(3,0);
        send_str(PS2Data3);
    break;
}

```

}

3. 注意问题及调试方法

(1) 实验时请注意以下问题:

将硬件各个部分正确插入实验底板上;程序调试过程中不要按下 LaunchPad 上的复位按钮。如果处于非在线调试模式,上电后若无显示,按下 LaunchPad 上的复位按钮进行复位;数据区域使用的是键盘右端的小键盘数字区;另外,不要同时按下多个按钮,否则键盘输入波形会相互干扰,无法正确解析,键值出错,液晶上出现若干个空白。

PS2 使用时钟和数据线分别使用 P1.5、P1.6, LaunchPad 上将 P1.6 用跳线帽连接到 LED 上,实验时必须将跳线帽拔下,否则键盘将无法正确响应。

(2) 调试方法:

在中断函数中设置一变量,使该变量每次进入中断自加,可以发现每按下键盘上一个按钮(特殊功能键除外),会产生 33 次 I/O 中断。

进行键值和字符转换时,可在 I/O 中断处理函数中 `if(j>=3)` 语段中设置断点,在观察窗口中读取 PS2Temp 的值,即为所按键对应键值。

检验键值转换是否成功,可在主函数相应字母处设置断点,按下该键,程序停在断点处,表明解析成功。

还可以使用示波器观察波形,可采用单触发方式,此时得到的波形为第一组通码波形,可与 PS2 协议及扫描码集进行对比,得到按下的按钮代表的字符。也可将两种方法对比,看是否一致。

4. 最终效果

键盘输入显示效果图如图 6.6.7 所示。

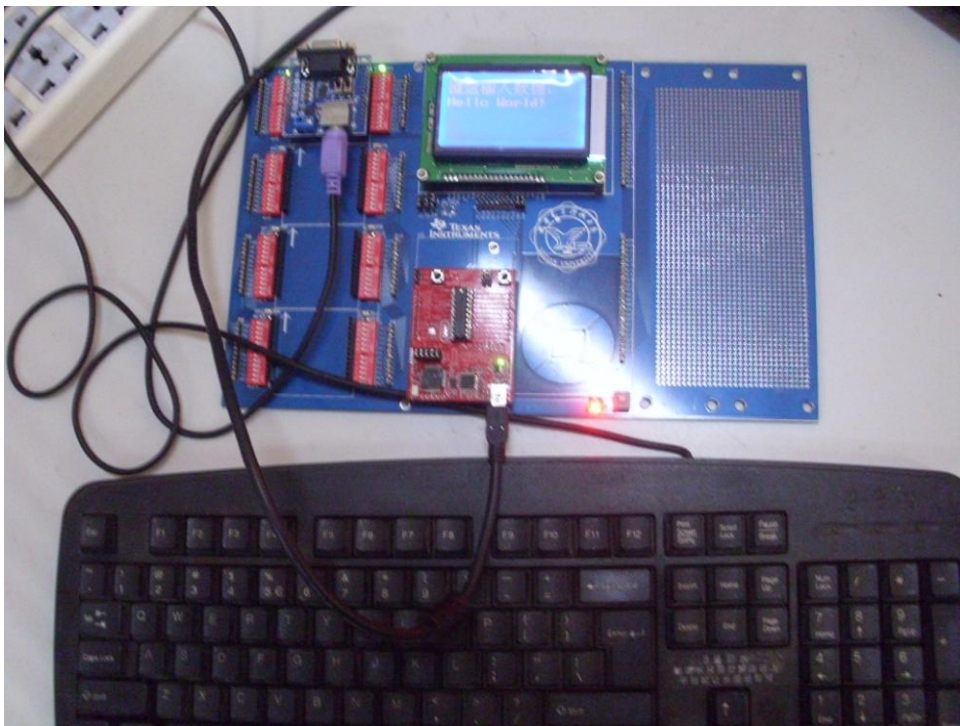


图 6.6.7 实际显示效果图

6.6.5 总结与扩展

该实践项目对 PS2 键盘进行了解析,帮不我们跟好的了解了常用计算机外设的工作原

理及设备识别原理，可以说掀开了计算机世界的神秘面纱的一角，在此基础上我们可以进一步学习 PS2 鼠标的解析、USB 鼠标、键盘的解析等，为嵌入式底层驱动的编写打下基础。并且，一个小小的接口，加上几百行代码，只要身边有计算机，我们就可省去在板子上设计矩阵键盘需要驱动及占用空间的麻烦，一个接口即可获得几十个按键，长短键？按键复用？众多的标志位？通通不再需要。当然，本项目也存在不足之处，对一些复杂的功能键，我们未给出完整解析，多个按键同时按下由于处理器速度的限制也无法给出正确解析，按键扫描码也只解析了一部分，对差错控制还不够，感兴趣的读者可以尝试予以完善。

最后给出两个思考题：

1. PS2 鼠标原理类似，请根据本实验给出方法解析出 PS2 鼠标动作，并在液晶上显示对应文字，或者做一个有鼠标操作的菜单，结合其他功能模块（如测温模块和 RGBLED 灯模块）实现复杂的菜单控制功能。

2. 结合 RS485 通信功能实现两机远程通信，即用键盘输入，显示在也将上，输入完成后按键发送到彼方，实现聊天工具的功能。