TI MSP430F6638 教学开发系统
实验指导书

Rev 1.3

清华大学电机工程与应用电子技术系
华清科仪（北京）科技有限公司
Huatsing Instruments

目　　录

第 1 章 MSP430F6638 教学开发系统

MSP430F6638 教学开发系统由主板及电池板、键盘板、电机板 3 个子板构成。系统核心 CPU 采用 TI 公司 MSP430 系列 16 位超低功耗 MCU 中最新的 F6638 芯片，系统拥有丰富的外设资源与接口，不但可以作为单片机实验教学平台，还可以作为工业控制、无线射频应用、物联网系统等开发与评估平台使用。

1.1 MSP430F6638 微控制器介绍

1.1.1 MSP430F6638 特性

- 低电源电压范围：1.8V 至 3.6V
- 超低功耗：
  - 激活模式(AM)：
    所有系统时钟激活：
    在 8MHz，3.0V，闪存程序执行时为 270μA/MHz（典型值）
  - 待机模式 (LPM3)：
    带有晶振的安全装置、且电源监控器可用、完全 RAM 保持、快速唤醒；
    2.2V 时为 1.8μA，3.0V 时为 2.1μA（典型值）
  - 关断 RTC 模式 (LPM 3.5)：
    关断模式，带有晶振的有源实时时钟：
    3.0V 时为 1.1μA（典型值）
  - 关断模式 (LPM4.5)：
    3.0V 时为 0.3μA（典型值）
- 在 3μs 内从待机模式唤醒（典型值）
- 16 位精简指令集 (RISC) 架构、扩展内存、高达 20MHz 的系统时钟
- 灵活的电源管理系统
  - 具有可编程经稳压内核电源电压的完全集成低压降稳压器 (LDO)
  - 电源电压监控、监视、和临时限电
- 统一时钟系统
  - 针对频率稳定的锁频环路 (FLL) 控制环路
  - 低功耗低频内部时钟源 (VLO)
  - 低频修整内部参照源 (REFO)
  - 32kHz 晶体 (XT1)
  - 高达 32MHz 的高频晶振 (XT2)
- 四个 配有 3，5，或者 7 个捕捉/比较寄存器的 16 位寄存器
- 2 个通用串行通信接口
- 全速通用串行总线 (USB)
- 具有内部共用基准、采样保持、和自动扫面功能的 12 位模数 (A/D) 转换器
- 具有同步功能的双通道 12 位数模 (D/A) 转换器

- 电压比较器
- 具有高达 160 段对比度控制的集成 LCD 驱动器
- 支持 32 位运算的硬件乘法器
- 串行板上编程，无需外部编程电压
- 6 通道内部直接内存访问 (DMA)
- 具有电源电压后备开关的实时时钟模块

1.1.2 MSP430F6638 引脚图及结构框图

MSP430F6638 的引脚图如图 1.1.1 所示，结构框图如图 1.1.2 所示。

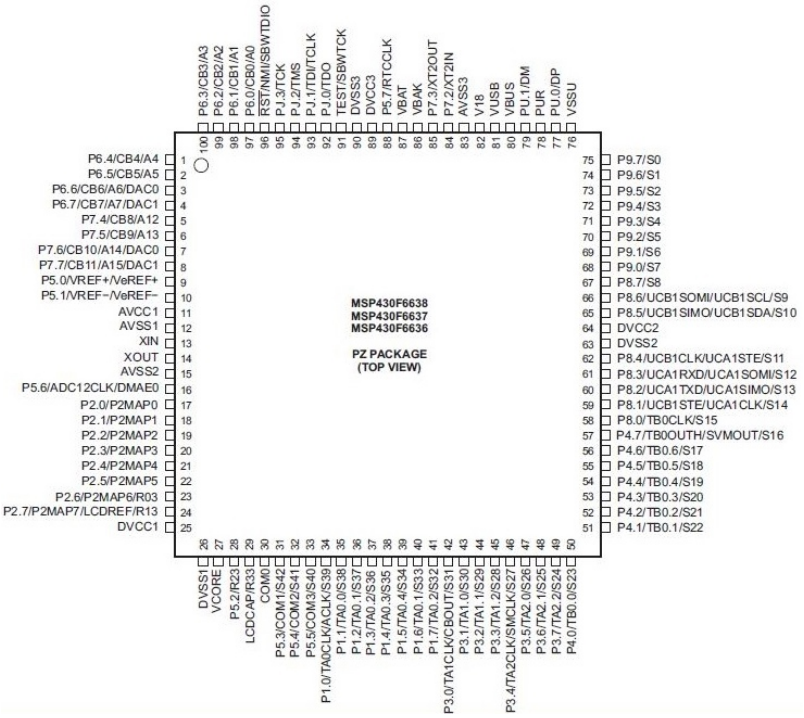

图 1.1.1　MSP430F6638 引脚图

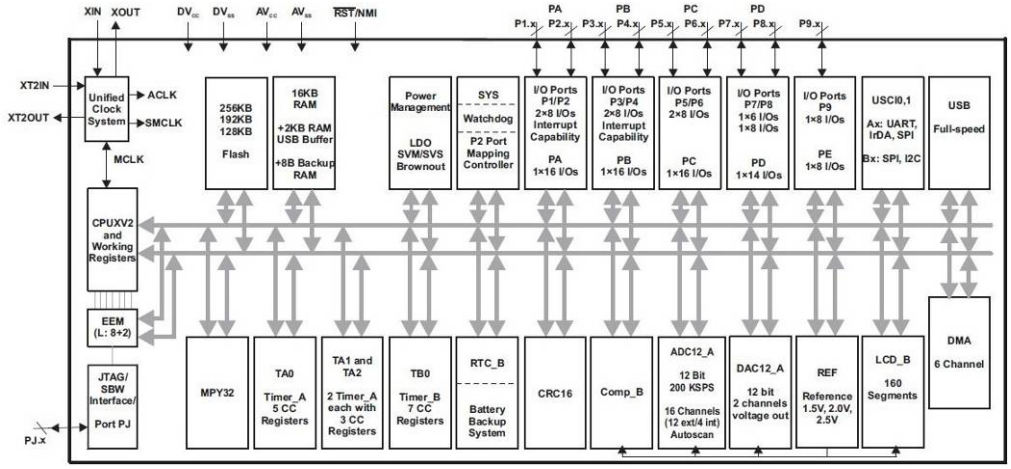

图 1.1.2　MSP430F6638 结构框图

1.2 MSP430F6638 教学开发系统硬件资源介绍

1.2.1 实验主板

主板是 MSP430F6638 教学开发系统的核心，板上不但有 TI 公司高性能低功耗的 16 位 MCU 芯片 MSP430F6638，还集成了板载仿真器以及丰富的外设资源，主要包括有：

- 电源模块：集成了电源选择开关（3 种供电模式选择）
- 无线射频模块接口：1 个 CC1101 模块接口、1 个 RF 射频模块接口
- eZ430-FET 板载仿真器模块
- 串行通信接口：1 个 RS232 接口、1 个 RS485 接口
- 音频模块：Line in 接口、Line out 接口、麦克风、扬声器
- 拨盘电位器与跳线帽收纳区
- 频率计信号输入端子
- 人机交互模块：用户指示灯 LED1~LED5、用户按键 S3~S7、触摸按键 Pad1~Pad
- 外部仿真器 JTAG 接口
- USB 接口：与 PC 机数据通信、电池充电
- I^2C 接口：2 组—P11、P14
- 段式液晶模块：单色，6 位 8 段
- TFT 液晶模块：彩色，320×240
- 温度传感器模块：可以测量本地（Local）温度与远程（Remote）温度
- MicroSD 卡模块
- Booster Pack 扩展模块：2 组—BP1（P17-P18）、BP2（P15-P16）

主板硬件资源图请看图 1.2.1。

注意：主板有 3 个硬件版本，分别是 v1.93、v1.93a 与 v1.95，这 3 个硬件版本区别不大，只是在个别芯片、模块与接口上有少量改动，在后面的介绍中会作出说明。其中 v1.93a 与 v1.95 版本主板的程序完全通用，与 v1.93 版本的程序大部分通用，只是涉及 TFT 屏幕显示的程序略有不同，这些也会在程序说明文档中加以说明。

主板版本号在主板背面中间偏下的位置有标识，v1.93a 与 v1.95 版本在实验箱外箱贴纸以及主板正面中间的丝印型号“MSP430F6638 EVM”旁边也都有注明。

电池子板、电机子板、键盘子板这 3 个附件板都只有 v1.93 一个硬件版本。

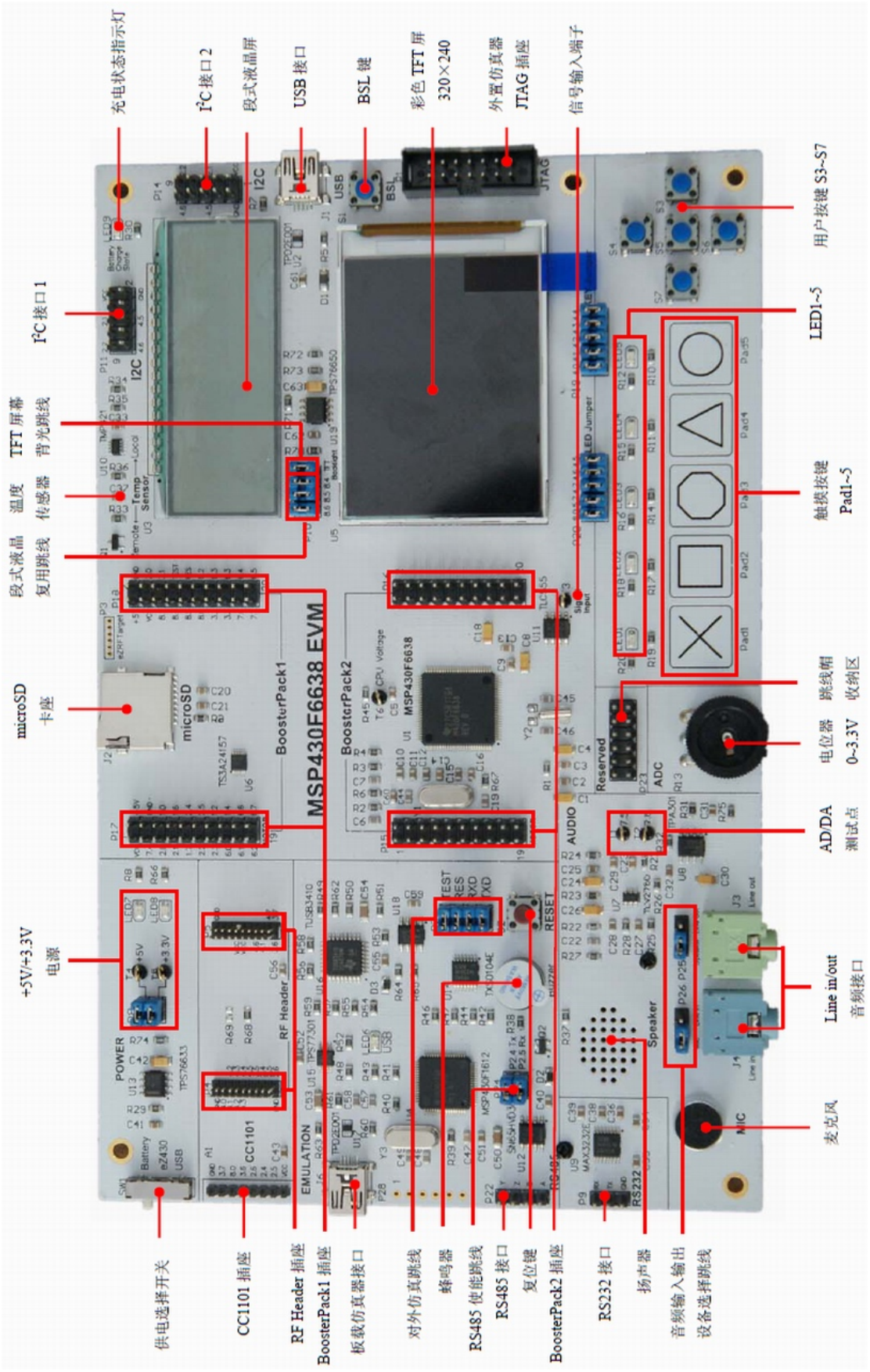

图 1.2.1　MSP430F6638 教学开发系统主板硬件资源图

1.2.1.1 电源模块

主板左上角是电源模块，v1.93 与 v1.93a 版本主板采用 TI TPS76633 LDO 芯片为系统提供 3.3V 供电（v1.95 版本主板采用 TI TPS73533 芯片），SW1 电源选择开关为用户提供 3 种供电选择。

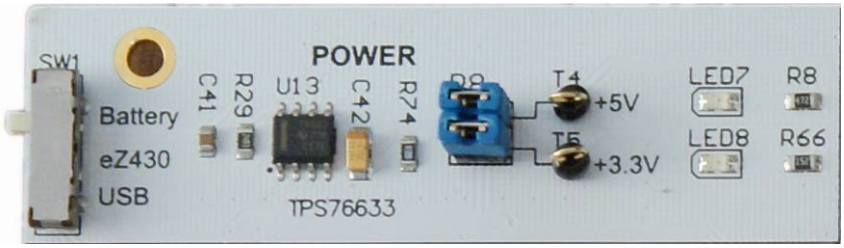

图 1.2.2　电源模块外观图

系统有 3 种供电选择：

- SW1拨至Battery 位置，系统由主板背面的锂电池子板供电（供电电压 5V，最高可提供 1A 电流）；
- SW1拨至eZ430 位置，系统由主板左侧的板载仿真器 eZ430 USB 接口（J6）供电；
- SW1拨至USB 位置，系统由主板右侧的USB 接口（J1）供电；

电源模块电源模块部分相关电路原理图如下：

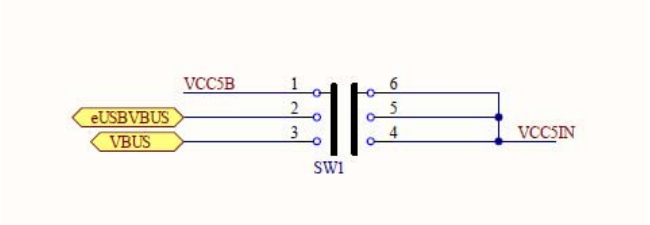

图 1.2.3.　电源模块部分原理图 1

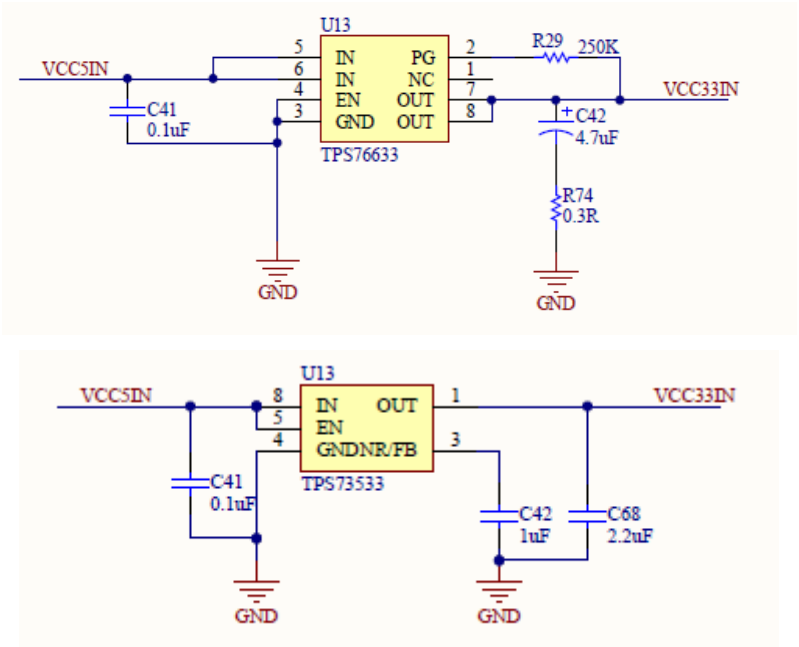

图 1.2.4　电源模块部分原理图 2（上图 1.93&1.93a，下图 1.95）

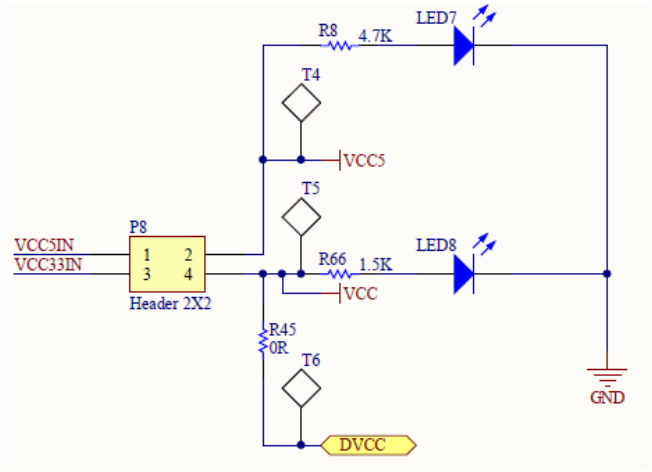

图 1.2.5　电源模块部分原理图 3

电源模块上的 TPS76633（或 TPS73533）芯片为 3.3V 的 LDO，可提供 250mA 的电流；P8 为系统 5V、3.3V 电源的供电跳线帽，断开后串接电流表可用于测量系统功耗；LED7、LED8 指示灯指示 5V、3.3V 两路电源供电状态。

1.2.1.2 无线射频模块接口

通过无线射频接口可以扩展两种无线模块，左侧的为 CC1101 型模块接口，右侧为 RF Header 接口，外观与电路原理见下图。

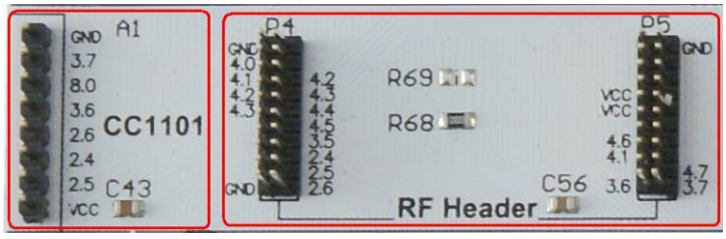

图 1.2.6　无线射频模块接口外观图

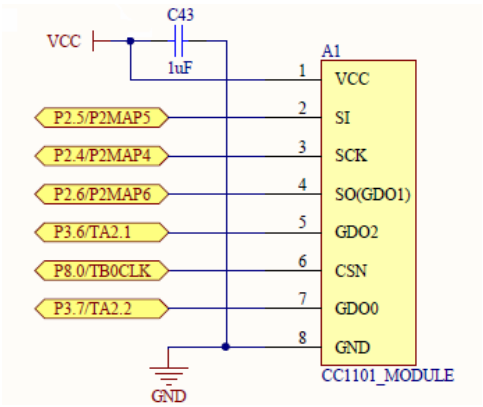

图 1.2.7　无线射频模块接口原理图 1

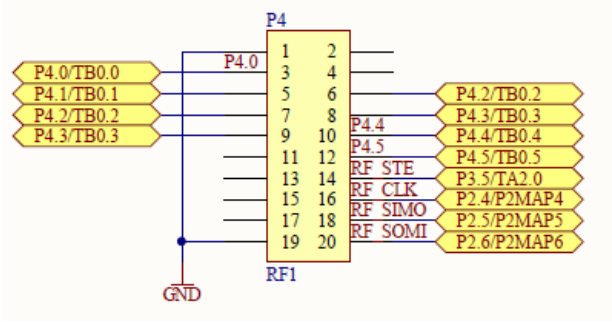

图 1.2.8　无线射频模块接口原理图 2

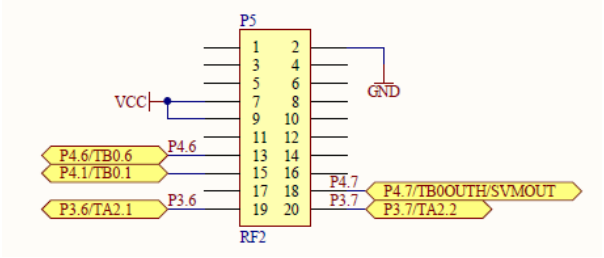

图 1.2.9　无线射频模块接口原理图 3

CC1101 模块接口（A1）引脚定义见下：

位置	序号	定义	说明	序号	定义	说明
A1（CC1101）	1	GND		*2	P3.7	TA2.2
	3	P8.0		*4	P3.6	TA2.1
	*5	P2.6	GPIO	*6	P2.4	UCA0SIMO
	7	P2.5	UCA0SOMI	8	+3.3V	

注：带*的引脚与其他功能模块有复用关系，使用时需注意

表 1.2.1　CC1101 模块接口引脚定义

RF Header 模块接口（P4、P5）引脚定义见下：

位置	序号	定义	说明	序号	定义	说明
P4（RF Header 左侧）	1	GND		2	NC	
	*3	P4.0	TB0.0	4	NC	
	*5	P4.1	TB0.1	*6	P4.2	TB0.2
	*7	P4.2	TB0.2	*8	P4.3	TB0.3
	*9	P4.3	TB0.3	*10	P4.4	TB0.4
	11	NC		*12	P4.5	TB0.5
	13	NC		14	P3.5	TA2.0
	15	NC		*16	P2.4	UCA0SIMO
	17	NC		*18	P2.5	UCA0SOMI
	19	NC		*20	P2.6	GPIO

位置	序号	定义	说明	序号	定义	说明
P5（RF Header 右侧）	1	NC		2	GND	
	3	NC		4	NC	
	5	NC		6	NC	
	7	+3.3V		8	NC	
	9	+3.3V		10	NC	
	11	NC		12	NC	
	*13	P4.6	TB0.6	14	NC	
	*15	P4.1	TB0.1	16	NC	
	17	NC		18	*P4.7	TB0.7
	*19	P3.6	TA2.1	20	*P3.7	TA2.2

注：带*的引脚与其他功能模块有复用关系，使用时需注意；

表 1.2.2　RF Header 模块接口引脚定义

1.2.1.3 eZ430-FET板载仿真器模块

eZ430-FET 板载仿真器模块具有 3 个主要功能：

- 为系统供电，为锂电池充电；
- 可通过左侧 USB 口（J6）连接 PC 机，进行程序下载、仿真和调试；
- 在 PC 端虚拟一个串口设备，实现与 PC 机串行通信；

模块上拥有独立的电源转换芯片 TPS77301，该芯片向 eZ430-FET 模块提供 3.6V，最大 250mA 的电流，该电平仅提供给 eZ430-FET 模块，不向其它任何位置供电，供电状态由 LED6 显示。

模块中右下角的 P2 跳线就是 eZ430-FET 模块与 MSP430F6638 的通信接口，连接“TEST”和“RES”两个跳线即可对系统提供下载和调试功能，“RXD”和“TXD”跳线为 MSP430F6638 的 UART1 的串行通讯连接功能。

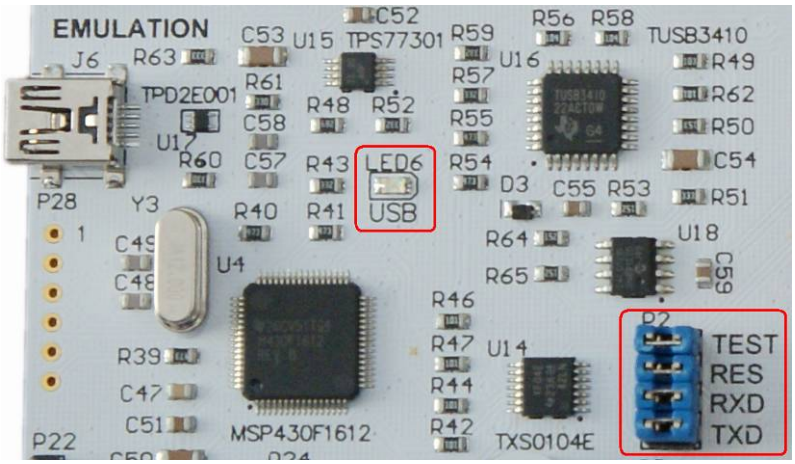

图 1.2.10　板载仿真器模块外观图

需要特别注意的是：当左侧 USB 接口（J6）连接 PC 机时，无论电源选择开关 SW1 拨到何种供电方式，如需主板上的复位按键（红色 RESET 键）有效，须将 P2 跳线中的“TEST”和“RES”跳线帽移除，否则可能导致复位失效。

位置	名称	跳线短接	跳线断开
P2	TEST	使能板载仿真 （J6 连接 PC 时系统 RESET 失效）	断开板载仿真 （系统 RESET 生效）
	RES	使能板载仿真 （J6 连接 PC 时系统 RESET 失效）	断开板载仿真 （系统 RESET 生效）
	RXD	连接J6与MSP430F6638的UART口	断开J6与MSP430F6638的UART口
	TXD	连接J6与MSP430F6638的UART口	断开J6与MSP430F6638的UART口

表 1.2.3　P2 处跳线定义

1.2.1.4 串行通信模块

串行通讯模块中包含 RS232 和 RS-485 两种通信接口。其中 RS485 模块采用 TI 公司的 3.3V 全双工驱动器 SN65HVD30，P24 处两个跳线连接 MSP430F6638 芯片上 P2.4 与 P2.5 端口，P24 的跳线移除时，断开 MSP430F6638 与 RS485 串口的连接。

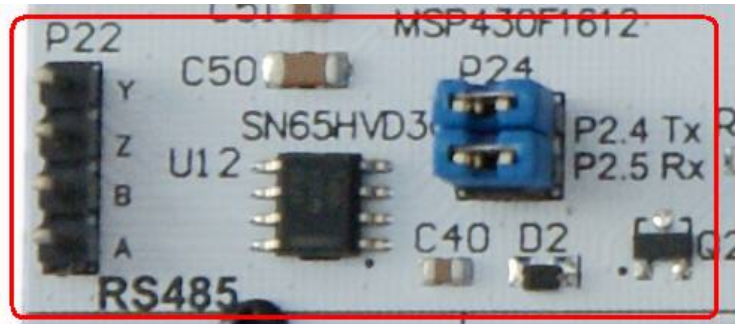

图 1.2.11　RS485 串口模块外观图

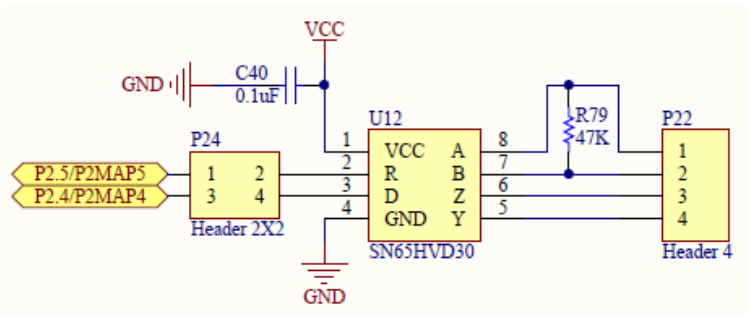

图 1.2.12　RS485 串口模块原理图

RS232 通信模块采用的是 TI 公司的 MAX3232E 芯片（U9），RS232 接口与 MicroSD 接口通过 TI 公司的 TS3A24157 芯片（U6）高速开关芯片构成端口复用形式。

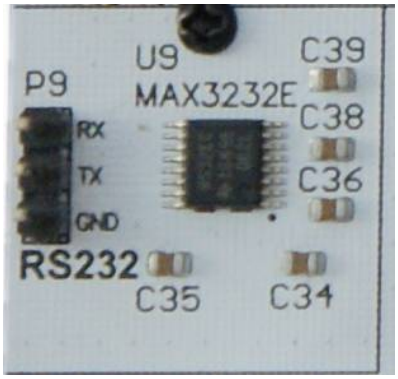

图 1.2.13　RS232 串口模块外观图

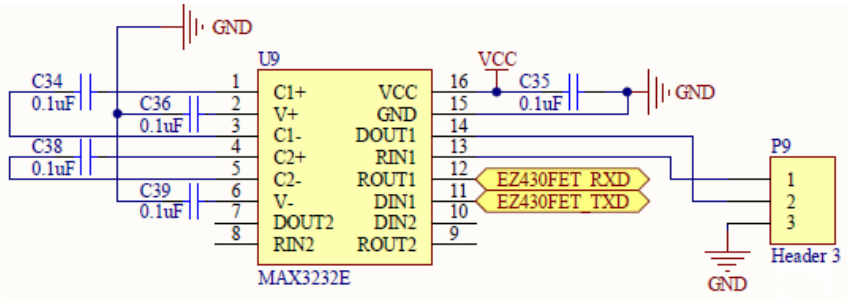

图 1.2.14　RS232 串口模块原理图 1

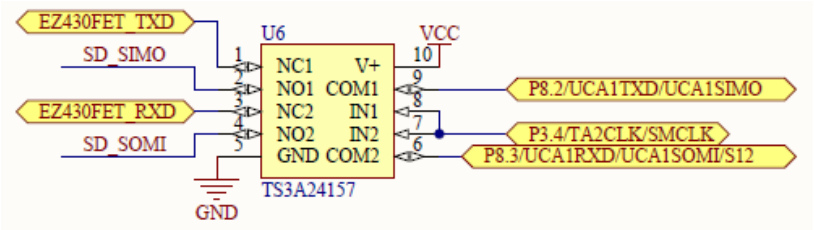

图 1.2.15　RS232 串口模块原理图 2

1.2.1.5 音频模块

音频模块由音频输入与输出两部分组成。

音频输入部分由板载麦克风（MIC）、Line in 接口（J4）以及 TI 微功耗单电源放大器 TLV2760 芯片（U7）组成。P26 处跳线可用来选择音频信号的输入方式，短接左侧两个插针时，音频信号由麦克风（MIC）输入；短接右侧两个插针时，音频信号由 Line in 接口（J4）输入。

音频输出部分由扬声器（Speaker、安装在主板背面）、Line out 接口（J3）以及 TI 音频功率放大器 TPA301 芯片（U8）组成。P25 处跳线可用来选择音频信号的输出方式，短接左侧两个插针时，音频信号由扬声器（Speaker）输出；短接右侧两个插针时，音频信号由 Line out 接口（J3）输出。

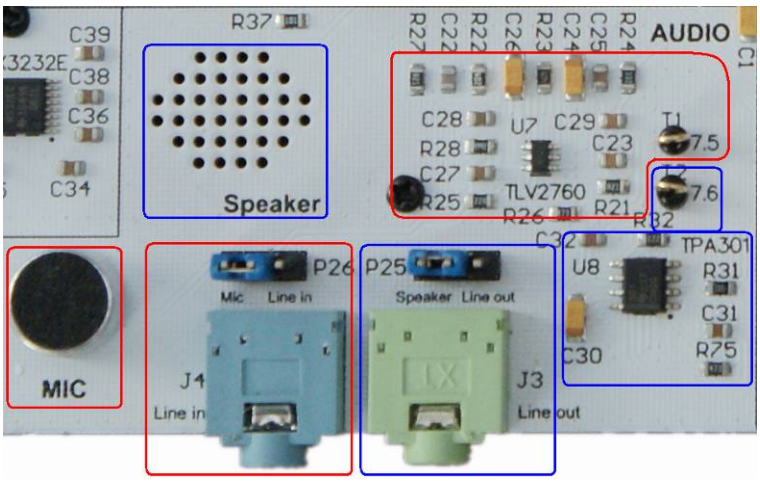

图 1.2.16　音频模块外观图

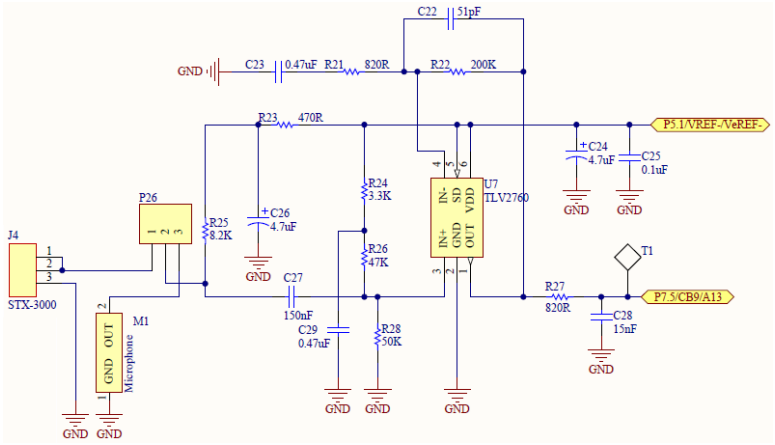

图 1.2.17　音频模块原理图（音频输入部分）

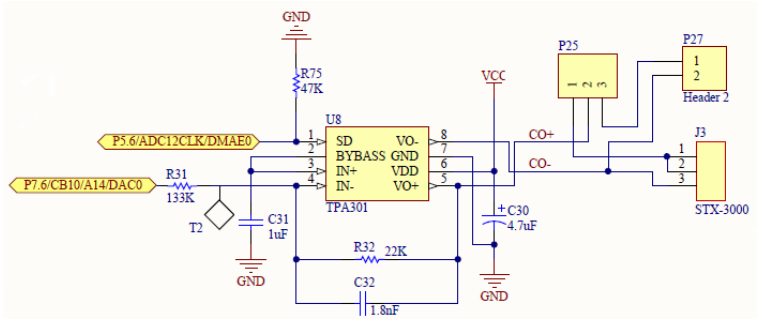

图 1.2.18　音频模块原理图（音频输出部分）

1.2.1.6 拨盘电位器与跳线帽收纳区

R13 是 50KΩ的拨盘电位器，电位器中心抽头连接到 F6638 芯片的 P7.7/CB11/A15 引脚，通过旋转旋钮可在 ADC 端得到 0~3.3V 的输入电压。电位器上方的双排针为跳线帽收纳区（Reserved，P23），此处没有任何电气连接，用户可将暂时用不到的跳线帽插在此处，以免

丢失。

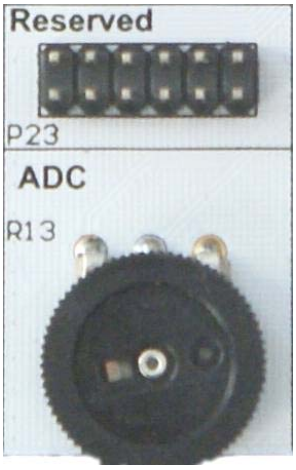

图 1.2.19　拨盘电位器与跳线帽收纳区外观

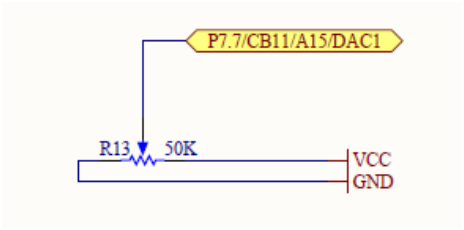

图 1.2.20　拨盘电位器模块原理图

1.2.1.7 人机交互模块

人机交互模块包含 5 个 LED 指示灯（LED1~LED5）、5 个电容触摸按键（Pad1~Pad5）、5 个机械按键（S3~S7）。

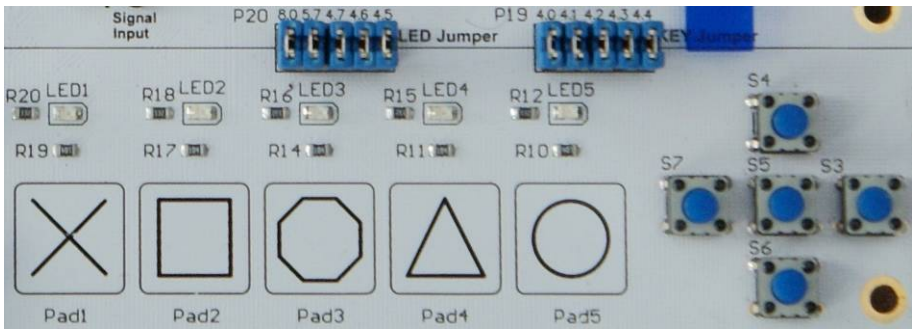

图 1.2.21　人机交互模块外观图

人机交互模块有 KEY Jumper（P19）、LED Jumper（P20）两组跳线，使用 S3~S7 按键时必须将 KEY Jumper（P19）5 个跳线短接上；使用 LED1~LED5 按键时必须将 LED Jumper（P20）5 个跳线短接上。

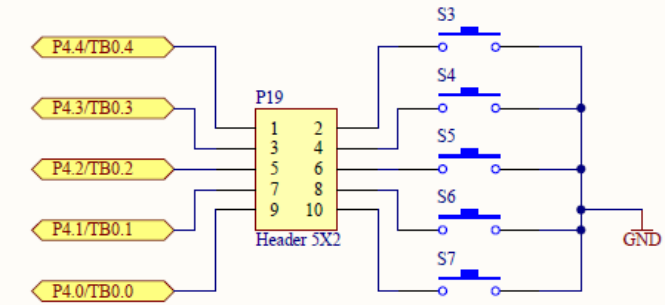

图 1.2.22　人机交互模块原理图 1

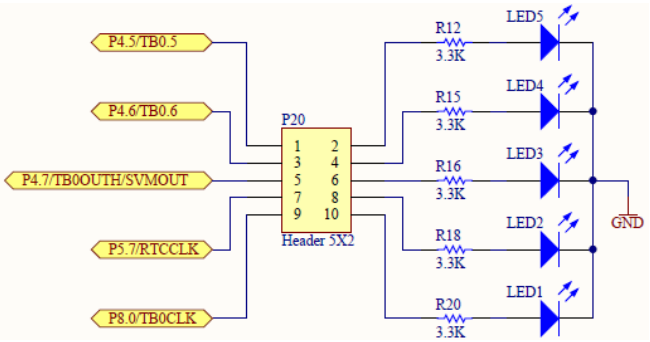

图 1.2.23　人机交互模块原理图 2

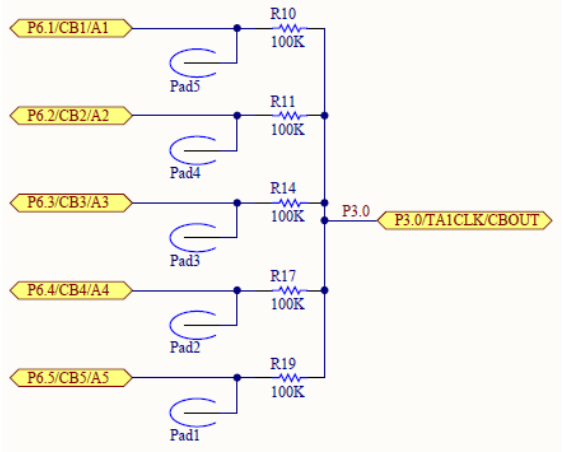

图 1.2.4　人机交互模块原理图 3

1.2.1.8 TFT屏幕模块

TFT 屏幕采用是一块高亮度、低功耗的 2.2 英寸 LCD 屏幕，屏幕分辨率为 320×240、具有 18 位色彩深度。系统通过 SPI 接口控制屏幕显示，在无信号情况下为常黑状态。v1.93 与 v1.93a 版本主板屏幕的 LED 背光由 TI 公司的 TPS76650 芯片进行控制（v1.95 版本主板采用 TPS75105），同时可通过 P10 的最右侧跳线来手动关闭背光。

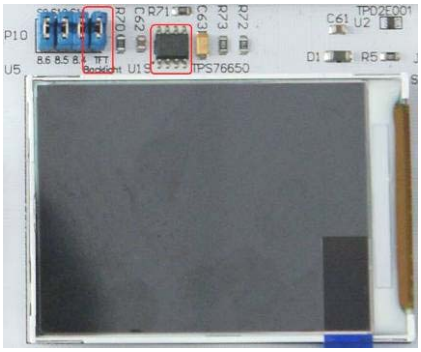

图 1.2.25　TFT 屏幕模块外观图

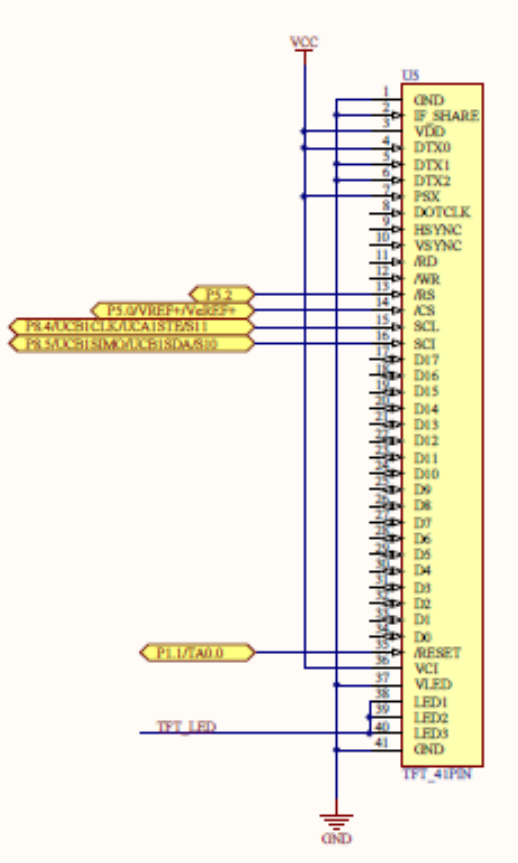

图 1.2.26　TFT 屏幕接口原理图

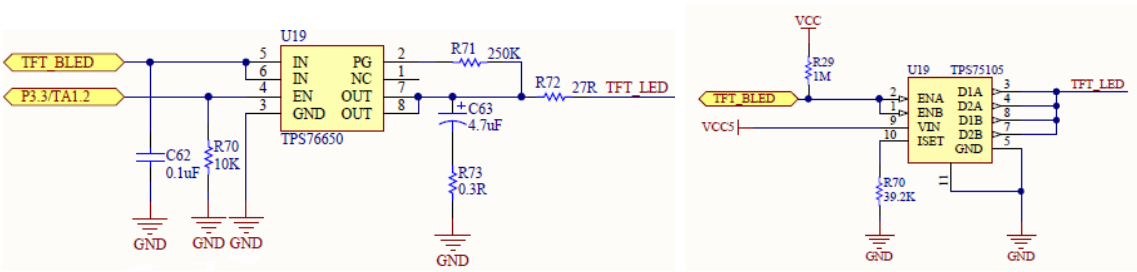

图 1.2.27　TFT 屏幕背光控制原理图

（左图为 v1.93 及 v1.93a 版本，右图为 v1.95 版本，TFT_BLED 信号定义在两套原理图中定义不同，请看对应原理图）

1.2.1.9 段式液晶模块

段式液晶采用是一块 48 段、1/3 偏压驱动液晶模块，能够显示 6 位字符（其中后 3 位字符带小数点），3 段电量符号。

图 1.2.28　段式液晶外观图

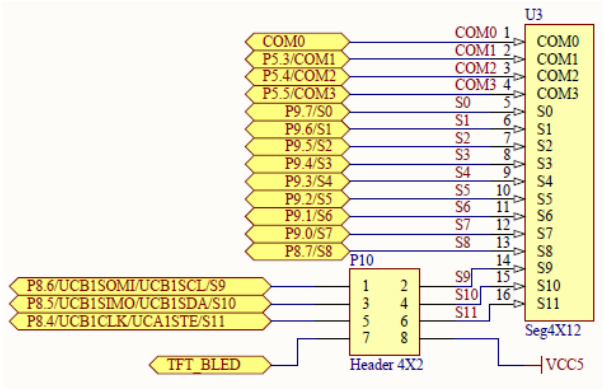

图 1.2.29　段式液晶模块电路原理图

注意：由于端口复用的原因，当不使用段式液晶模块时请将模块左下角 P10 处的 S9~S11 3 个跳线断开，否则系统工作时段式液晶模块上有可能出现不规则的显示。

1.2.1.10 I²C接口

主板上有 P11、P14 两组 I²C 接口，可以同时连接两组 I²C 设备，接口定义如下：

位置	序号	定义	说明	序号	定义	说明
I²C 接口（P11&P14）	1	+3.3V		2	GND	
	3	NC		4	NC	
	*5	P2.1	UCB0DAT	*6	P4.5	GPIO
	7	NC		8	NC	
	*9	P2.2	UCB0CLK	*10	P4.6	GPIO

注：带*的引脚与其他功能模块有复用关系，使用时需注意；

表 1.2.4　I²C 接口引脚定义

1.2.1.11 Booster Pack接口

系统提供 2 组 Booster Pack 接口（以下简称为 BP1、BP2，参考本书第 4 页图 1.2.1），可以连接多种外设模块，v1.93 及 v1.93a 版本主板 BP 接口定义如下：

位置	序号	定义	说明	序号	定义	说明
BP1/BP2 左侧 (P17/P15)	1	3.3V		2	+5V	
	3	P7.4	CB8/A12	4	GND	
	5	P2.0	UCB0STE/UCA0CLK	6	P1.0	TA0CLK/ACLK
	*7	P2.1	UCB0SIMO/UCB0SDA	8	P1.6	TA0.1
	9	P1.7	TA0.2	*10	P2.4	UCA0TXD/UCA0SIMO
	*11	P2.2	UCB0SOMI/UCB0SCL	*12	P2.5	UCA0RXD/UCA0SOMI
	13	P2.3	UCB0CLK/UCA0STE	14	P1.2	TA0.1
	15	P6.0	CB0/AI0	16	P1.4	TA0.4
	*17	P6.1	CB1/AI1	18	P6.6	CB6/AI6/DAC0
	*19	P6.2	CB2.AI2	20	P6.7	CB7/AI7/DAC1
BP1/BP2 右侧 (P18/P16)	1	+5V		2	GND	
	3	+3.3V		*4	P3.0	TA1CLK/CBOUT
	*5	P8.1	UCB1STE/UCA1CLK	*6	P3.1	TA1.0
	*7	P8.4	UCB1CLK/UCA1STE	8	TEST	
	*9	P8.5	UCB1SIMO/UCB1SDA	10	RES	
	*11	P8.6	UCB1SOMI/UCB1SCL	*12	P8.2	UCA1TXD/UCA1SIMO
	*13	P3.2	TA1.1	*14	P8.3	UCA1RXD/UCA1SOMI
	*15	P3.3	TA1.2	*16	P6.3	CB3/AI3
	*17	P7.6	CB10/AI14	*18	P6.4	CB4/AI4
	*19	P7.7	CB11/AI15	*20	P6.5	CB5/AI5

注：带*的引脚与其他功能模块有复用关系，使用时需注意；

表 1.2.5　Booster Pack 接口引脚定义（v1.93 及 v1.93a 版本）

以下为 v1.95 版本主板 Booster Pack 接口定义，其中左侧接口（**P17/P15**）没有变化，右侧接口（**P18/P16**）有部分调整：

位置	序号	定义	说明	序号	定义	说明
BP1/BP2 左侧 (P17/P15)	1	3.3V		2	+5V	
	3	P7.4	CB8/A12	4	GND	
	5	P2.0	UCB0STE/UCA0CLK	6	P1.0	TA0CLK/ACLK
	*7	P2.1	UCB0SIMO/UCB0SDA	8	P1.6	TA0.1
	9	P1.7	TA0.2	*10	P2.4	UCA0TXD/UCA0SIMO
	*11	P2.2	UCB0SOMI/UCB0SCL	*12	P2.5	UCA0RXD/UCA0SOMI
	13	P2.3	UCB0CLK/UCA0STE	14	P1.2	TA0.1
	15	P6.0	CB0/AI0	16	P1.4	TA0.4
	*17	P6.1	CB1/AI1	18	P6.6	CB6/AI6/DAC0
	*19	P6.2	CB2.AI2	20	P6.7	CB7/AI7/DAC1
BP1/BP2 右侧 (P18/P16)	*1	5.7	RTCCLK	2	GND	
	*3	6.3	CB3/AI3	*4	3.1	TA1.0
	*5	6.4	CB4/AI4	*6	3.0	TA1CLK/CBOUT
	*7	6.5	CB5/AI5	8	TEST	
	*9	8.5	UCB1SIMO/UCB1SDA	10	RST	
	*11	3.2	TA1.1	*12	8.2	UCA1TXD/UCA1SIMO
	*13	3.3	TA1.2	*14	8.3	UCA1RXD/UCA1SOMI
	*15	3.5	TA2.0	*16	8.4	UCB1CLK/UCA1STE
	*17	3.6	TA2.1	*18	8.6	UCB1SOMI/UCB1SCL
	*19	3.7	TA2.2	*20	8.1	UCB1STE/UCA1CLK

注：带*的引脚与其他功能模块有复用关系，使用时需注意；

表 1.2.6　Booster Pack 接口引脚定义（v1.95 版本）

1.2.2 电池子板

电池子板上包含一块 3.7V，800mAH 的聚合物锂离子电池，可以为系统提供长达 10 小时以上的供电。板上还有 TI 公司先进的电源管理芯片，能够实现智能充放电管理、电池状态监测、DC-DC 稳压输出等功能。

图 1.2.30　电池子板外观图

1.2.2.1 Li-ion电池规格特性及使用注意事项

电池子板上使用的的是聚合物锂离子电池，规格如下：

- 电压 3.7V，容量 800mAH；
- 充电电压限制：4.2V；
- 持续放电电流：1.0C（800mA）；
- 最大充电电流：1.0C（800mA）；
- 工作温度：0~40°C；
- 存储温度：0~30°C；

这个电池子板只可以配合 MSP430F6638 开发平台使用，不可作为其他设备的供电电源。电池需要充电时，只可以将电池子板插到主板背面，然后通过主板左侧 USB 口（J6）给电池充电（主板右侧 USB 口 J1 没有充电功能），不支持其他方式为电池充电。充电时，将主板左侧 USB 口（J6）连接到电脑，并将供电选择开关 SW1 拨至 eZ430 档位，此时将对锂电池进行充电，主板右上角的充电状态指示灯 LED9 点亮，电池完全充满后 LED9 熄灭（300mA 充电电流约需要 4.5 小时）。根据负载不同，环境温度不同，电池供电时间会发生变化。

特别需要指出的是：电池不可过充、过放、短路、解体、反向充电、在高温高湿环境中使用等，否则**有可能造成设备损坏及人身伤害**；

1.2.2.2 电池充放电管理

电池子板上采用 TI 公司 BQ24230 锂离子电池充电管理芯片做充放电控制，最大充电电流 500mA。

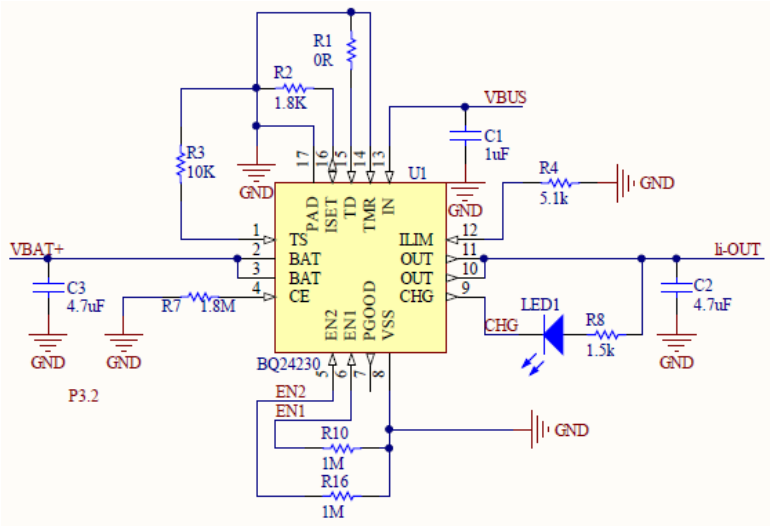

图 1.2.31　充放电管理电路原理图

充电时，主板右上角处的 LED9 点亮。当充电结束时，这两处的 LED 灯会同时熄灭。电池的最大充电电流我们可以通过电池子板上 P3 处的两个跳线来限定，跳线与最大充电电流的关系可以看电池子板背面的丝印。

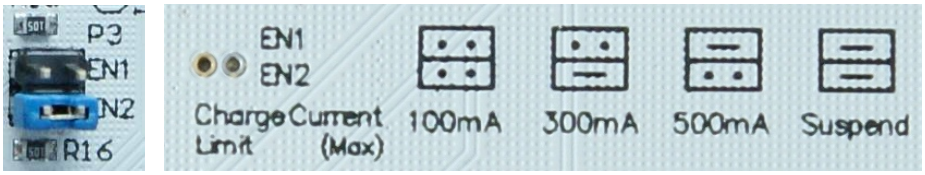

图 1.2.32　充电电流选择跳线 P3 及对应关系

位置	EN1	EN2	最大充电电流
P3	断开	断开	100mA
	断开	短接	300mA
	短接	断开	500mA
	短接	短接	挂起（不充电）

表 1.2.7　最大充电电流跳线设定

1.2.2.3 电池电量状态监测

电池子板上通过 TI 公司是 BQ27410 电量监测芯片对电池电量做实时监测，BQ27410 采用获专利的 Impedance Track™ 算法支持电量监测，可提供剩余电池容量(mAH)、充电状态(%)、电池电压(mV)等信息。

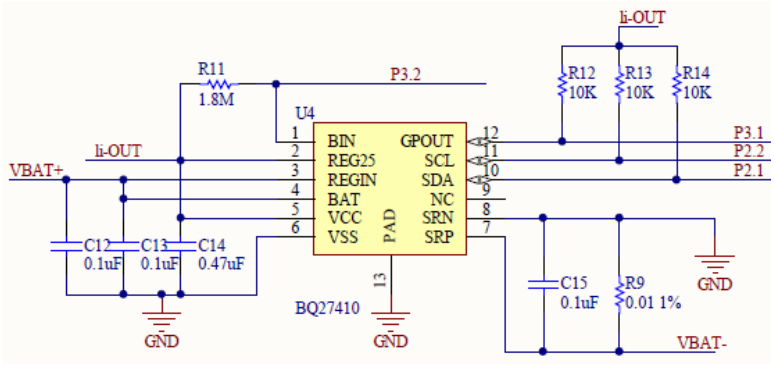

图 1.2.33　电池电量状态监测电路原理图

1.2.2.4 电池升压管理

锂离子电池输出电压为 3.7V 且不稳定，电池子板上使用 TI 公司的 TPS63061DC-DC 转换芯片将电池电压提升到稳定的 5V 电压对外输出。TPS63061 专门为电池供电的产品提供了一套电源解决方案，可以升压或者降压，效率高达 93%。

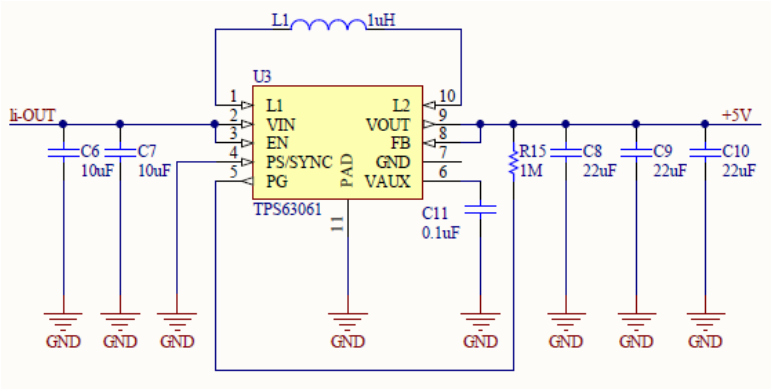

图 1.2.34　电池升压管理电路原理图

1.2.3 键盘与数码管子板

键盘与数码管子板通过 I²C 总线与主板连接，子板上包含 1 个 4× 4 矩阵键盘和 8 位带小数点的七段数码管。通过子板左侧 T1 与 T2 测试点可以测试 I²C 总线中数据（SDA）与时钟（SCL）信号。

图 1.2.35　键盘与数码管子板外观图

1.2.3.1 矩阵键盘

4× 4 矩阵键盘通过子板上的 TI 公司 TCA6408A 8 位 I²C 扩展芯片进行行列扫描，获取扫描结果后通过 I²C 总线将结果发送给主板。

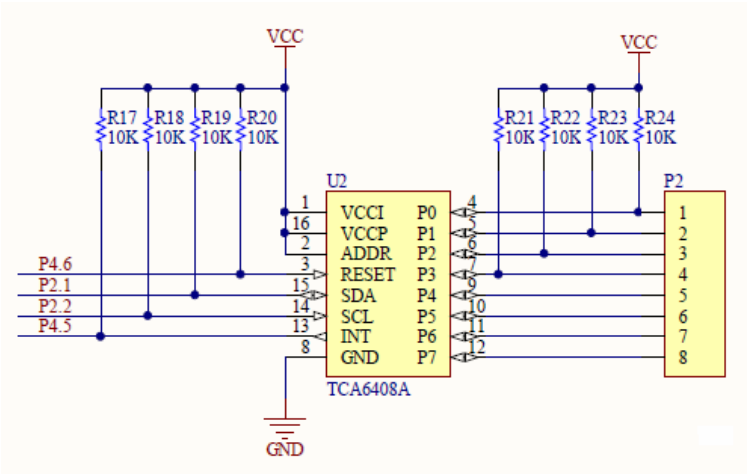

图 1.2.36　矩阵键盘电路原理图

1.2.3.2 数码管

子板上的 8 个七段数码管通过 TI 公司的 TCA6416A 16 位 I^2C 扩展芯片控制显示，芯片的 P0～P7 口产生八位数码管的位选信号；P10～P17 口用来控制七段数码管的 8 个段（含小数点）。如七段数码管显示异常，可通过模块上的 RESET 按键来随时重置数码管模块。

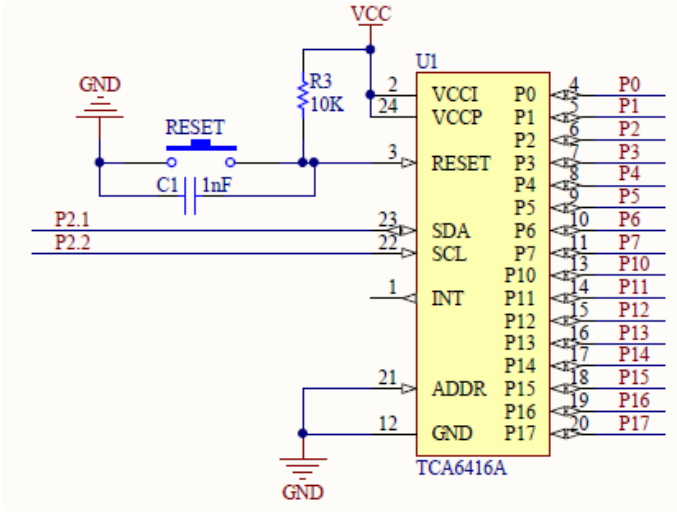

图 1.2.37　数码管电路原理图 1

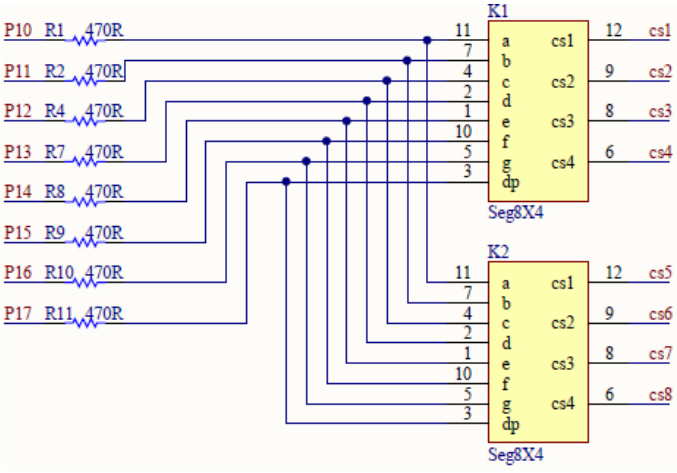

图 1.2.38　数码管电路原理图 2

1.2.4 电机子板

电机子板上包含有直流电机（DC Motor）与步进电机（Step Motor），这两个电机都采用 TI 公司 DRV8833 双路 H 桥接驱动芯片控制，且都带有光耦测速模块。其中 T7、T8 测试点为光耦模块输出信号，T1~T6 测试点信号内容请看直流电机与步进电机原理图。

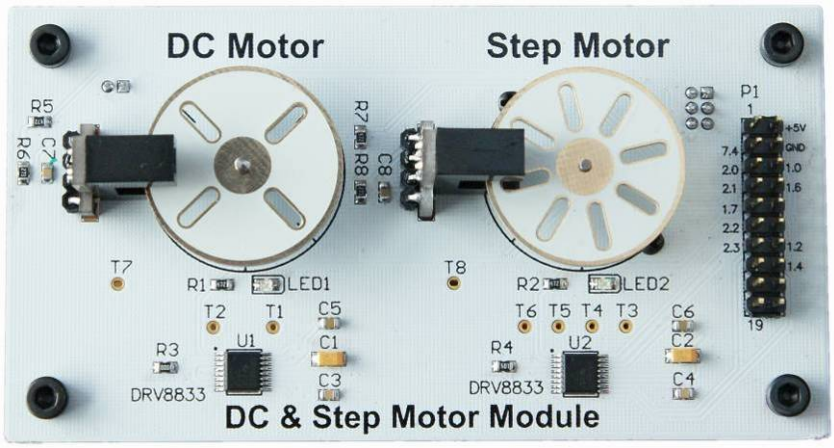

图 1.2.39　电机子板外观图

1.2.4.1 直流电机

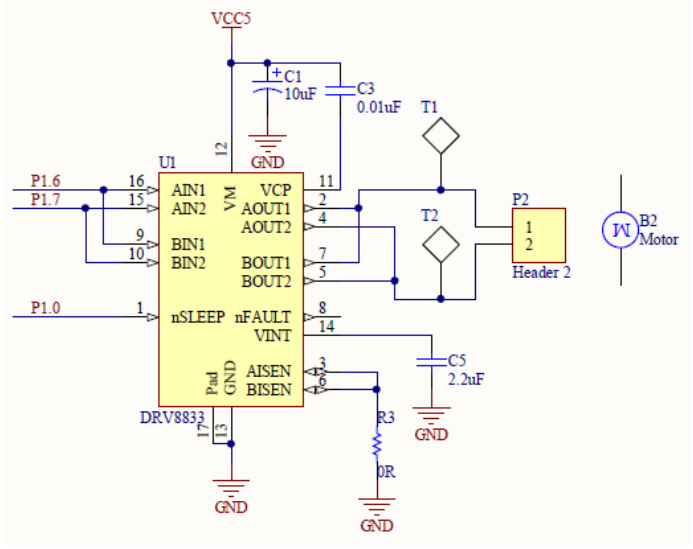

图 1.2.40　直流电机电路原理图

1.2.4.2 步进电机

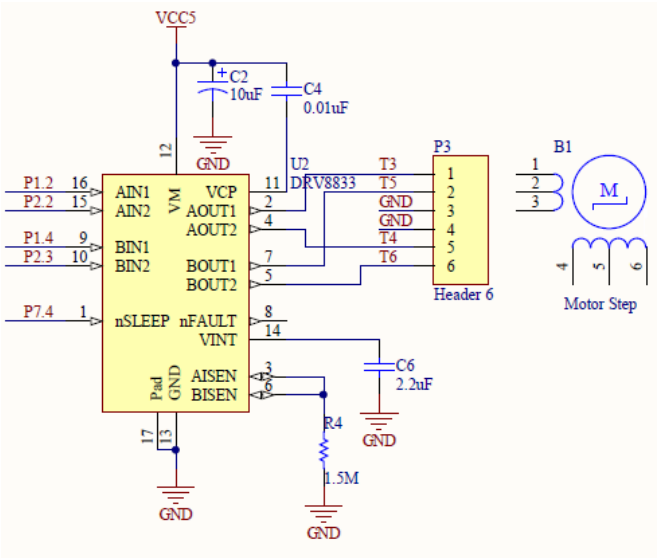

图 1.2.41　步进电机电路原理图

1.2.4.3 光耦测速模块

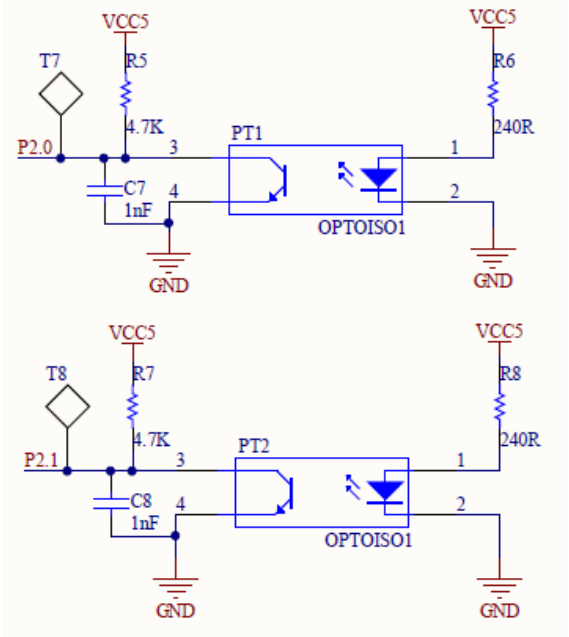

图 1.2.42　光耦测速电路原理图

第 2 章 集成开发环境CCS的安装与使用

CCS（Code Composer Studio）是 TI 公司研发的一款具有环境配置、源文件编辑、程序调试、跟踪和分析等功能的集成开发环境，能够帮助用户在一个软件环境下完成编辑、编译、链接、调试和数据分析等工作。CCSv5.3 为 CCS 软件的最新版本，功能更强大、性能更稳定、可用性更高，是 MSP430 软件开发的理想工具。

2.1 CCSv5.3 的安装

（1）运行下载的安装程序 ccs_setup_5.3.0.00090.exe，当运行到如图 2.1.1 处时，选择 Custom 选项，进入手动选择安装通道。

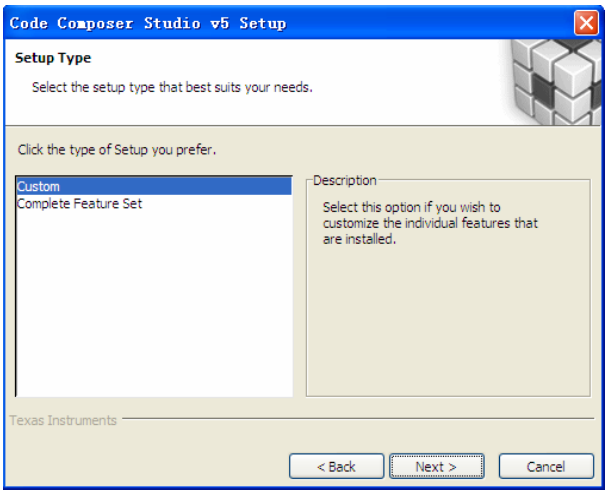

图 2.1.1 安装过程 1

（2）单击 Next 得到如图 2.1.2 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。

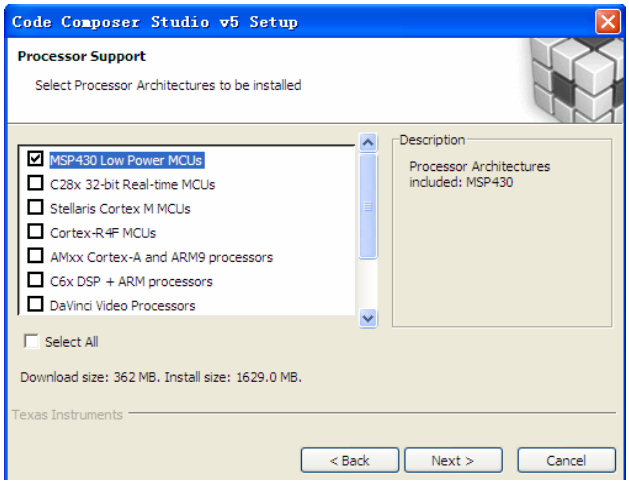

图 2. 1. 2 安装过程 2

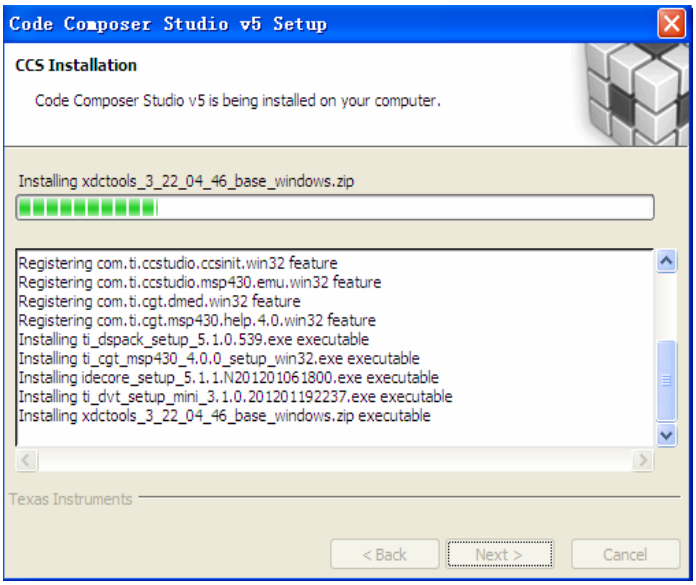

图 2.1.3　软件安装中

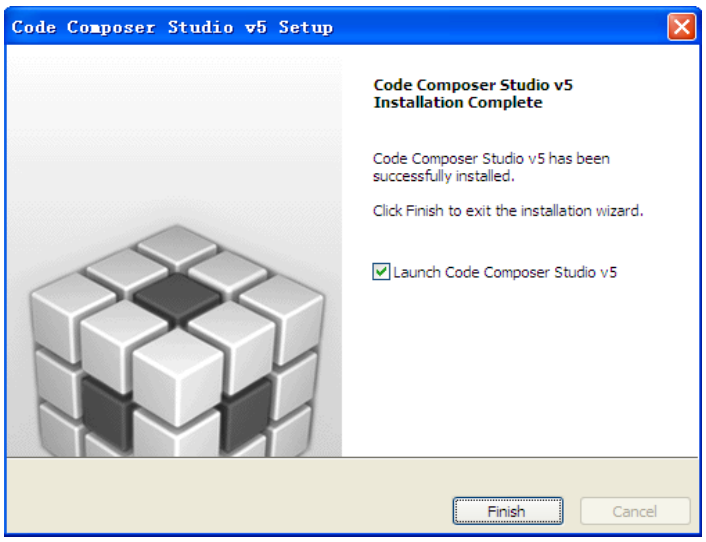

图 2.1.4　软件安装完成

（3）单击 Finish，将运行 CCS，弹出如图 2.1.5 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：-\MSP430F6638\Workspace，单击 Browse，将工作区间链接到所建文件夹，不勾选"Use this as the default and do not ask again"。

图 2.1.5　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 2.1.6 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CCSv5.3 软件开发集成环境，如图 2.7 所示。

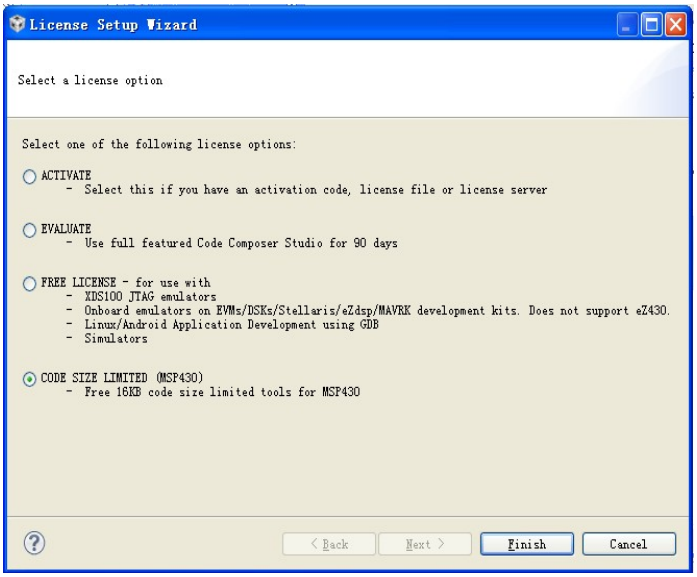

图 2.1.6　软件许可选择窗口

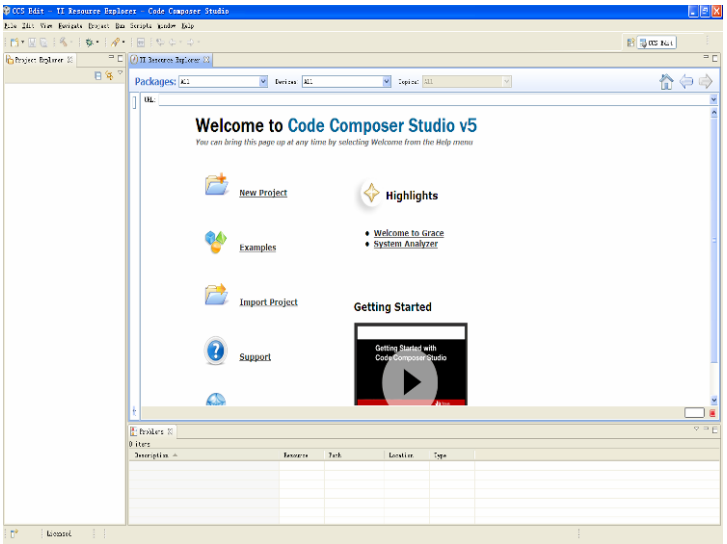

图 2.1.7　CCSv5 软件开发集成环境界面

2.2 利用CCSv5.3 导入已有工程

（1）在此以实验一的工程为例进行讲解，首先打开 CCSv5.3 并确定工作区间：F:\MSP430F6638\Workspace，选择 File-->Import 弹出图 2.2.1 对话框，展开 Code Composer Studio 选择 Existing CCS/CCE Eclipse Projects。

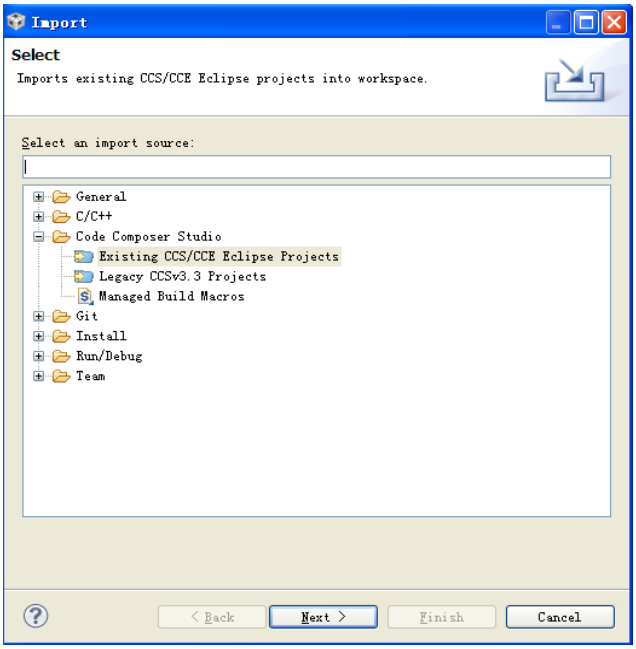

图 2.2.1　导入新的 CCSv5 工程文件

（2）单击 Next 得到图 2.2.2 对话框。

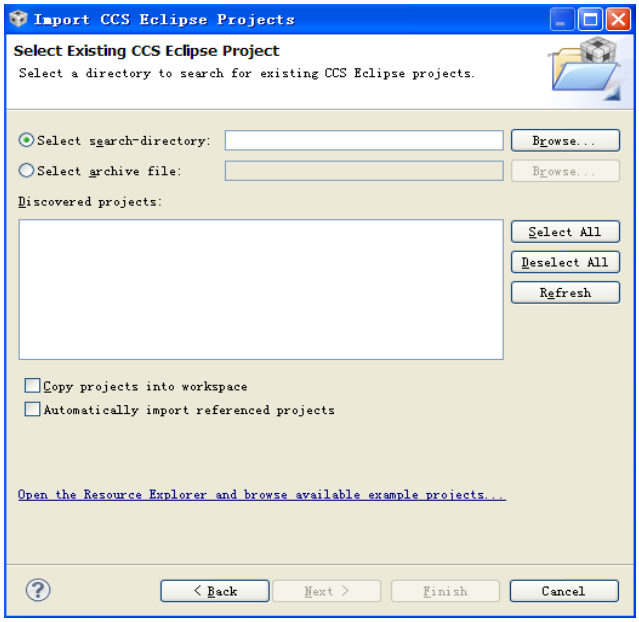

图 2.2.2　选择导入工程目录

（3）单击 Browse 选择需导入的工程所在目录，在此，我们选择 F:\MSP430F6638\Workspace\MSP430F6638EVM\LAB01（需在此之前，将实验代码复制到工作

区间下），得到图 2.2.3。

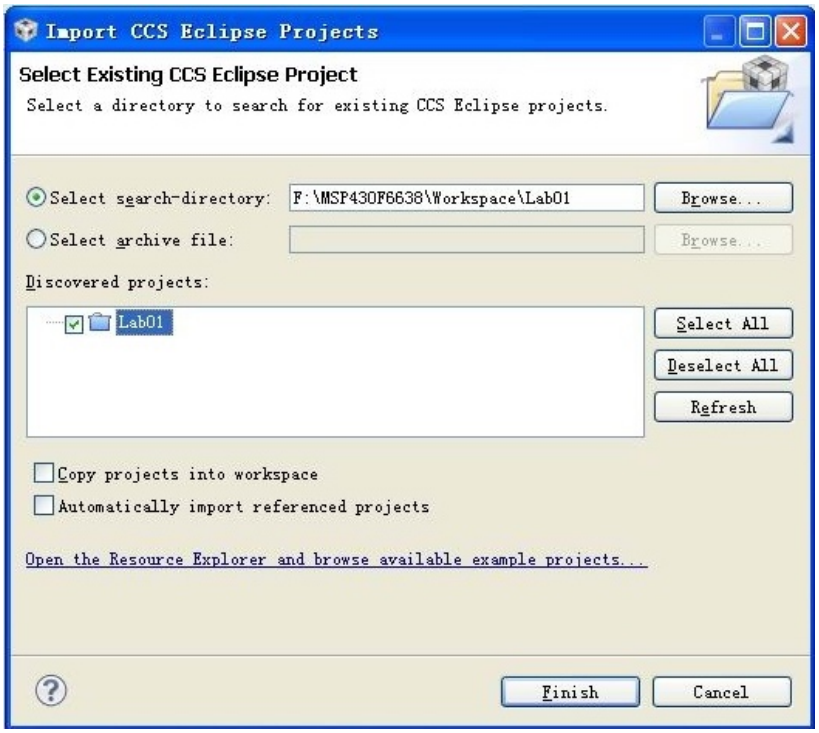

图 2.2.3　选择导入工程

（4）单击 Finish，即可完成既有工程的导入。

2.3 利用CCSv5.3 新建工程

（1)首先打开 CCSv5.3 并确定工作区间，然后选择 File-->New-->CCS Project 弹出图 2.3.1 对话框。

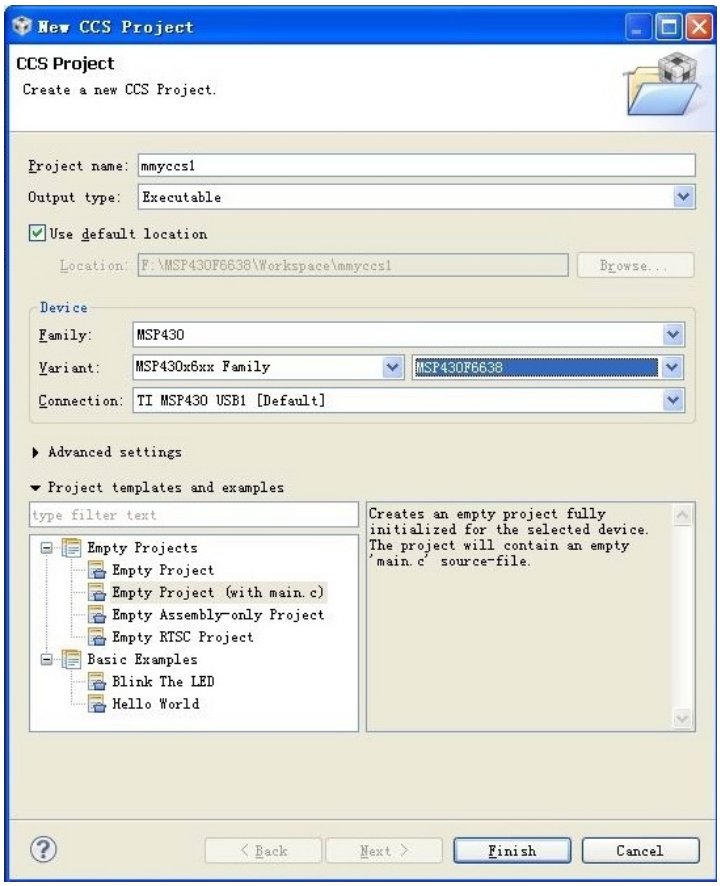

图 2.3.1　新建 CCS 工程对话框

（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 MSP430X6XX family，芯片选择 MSP430F6638；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 2.3.2 所示。

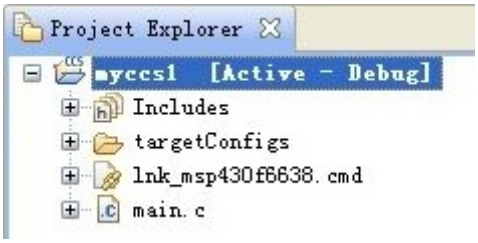

图 2.3.2　初步创建的新工程

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 2.3.3 对话框。

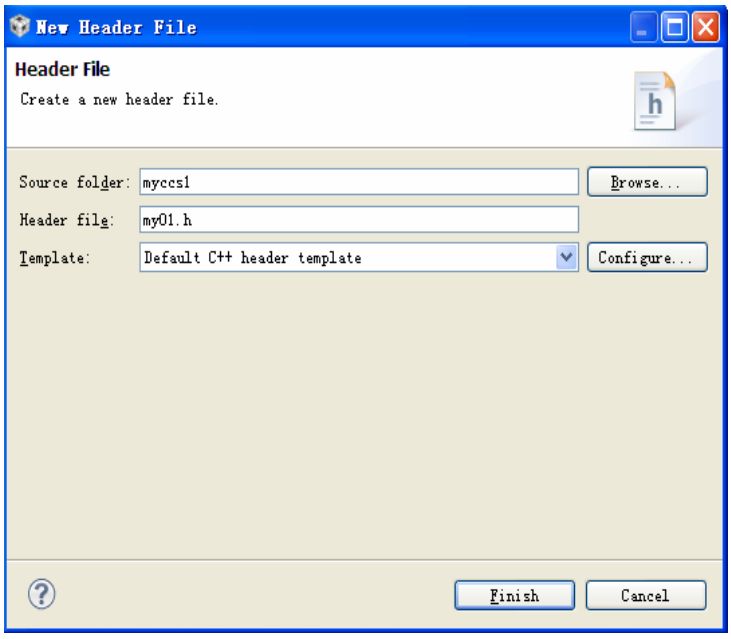

图 2.3.3　新建.h 文件对话框

在 Header file 中输入头文件的名称，注意必须以.h 结尾，在此输入 myo1.h。

（8）新建.c 文件：在工程名上右键单击，选择 New-->source file 得到图 2.3.4 对话框。

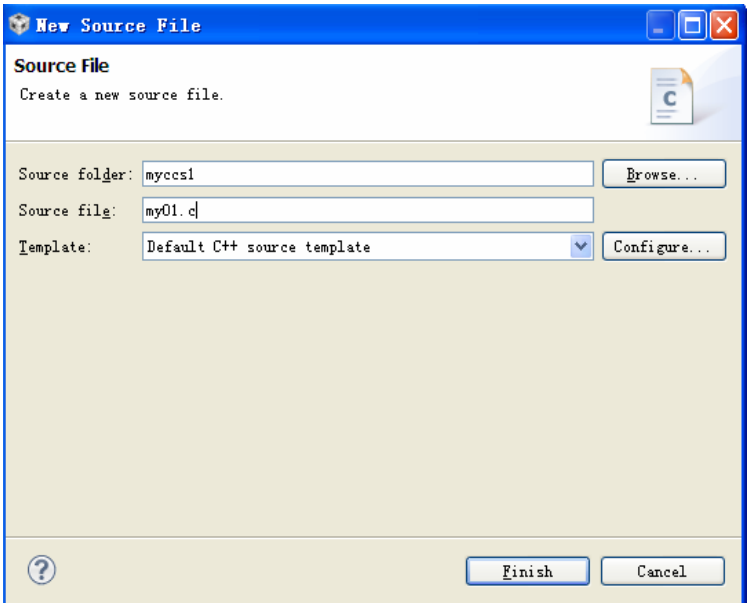

图 2.3.4　新建.c 文件对话框

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 2.3.5 对话框。

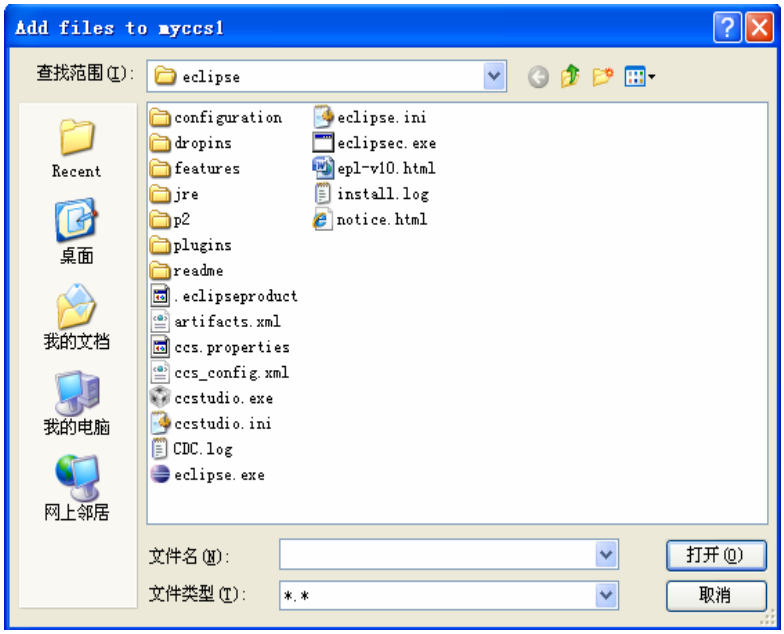

图 2.3.5 导入已有文件对话框

找到所需导入的文件位置，单击打开，得到图 2.3.6 对话框。

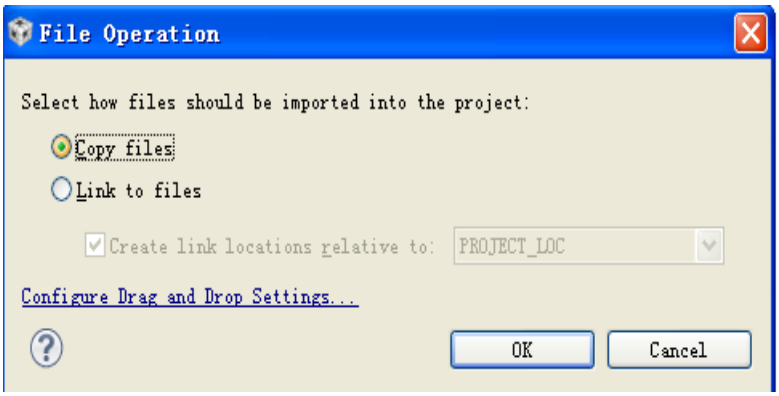

图 2.3.6 添加或连接现有文件

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。

2.4 利用CCSv5.3 调试工程

2.4.1 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：首先导入实验一的工程，导入步骤请参考 2.2 节，如图 2.4.1 所示，**其中 MSP430F6638.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 myccs1 的工程再次创建目标配置文件。

图 2.4.1 LAB1 工程浏览器

（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。

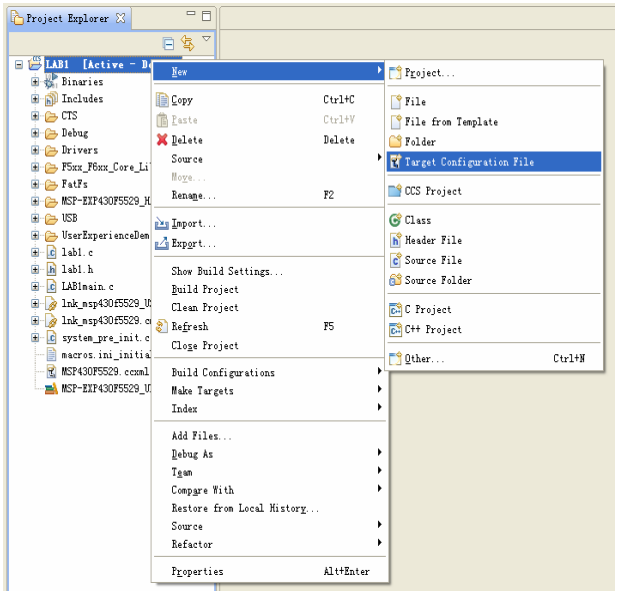

图 2.4.2 创建新的目标

（3）在 File name 中键入后缀为.ccxml 的配置文件名，由于创建 F6638 开发板的目标配置文件，因此，将配置文件命名为 MSP430F6638.ccxml，如图 2.4.3 所示。

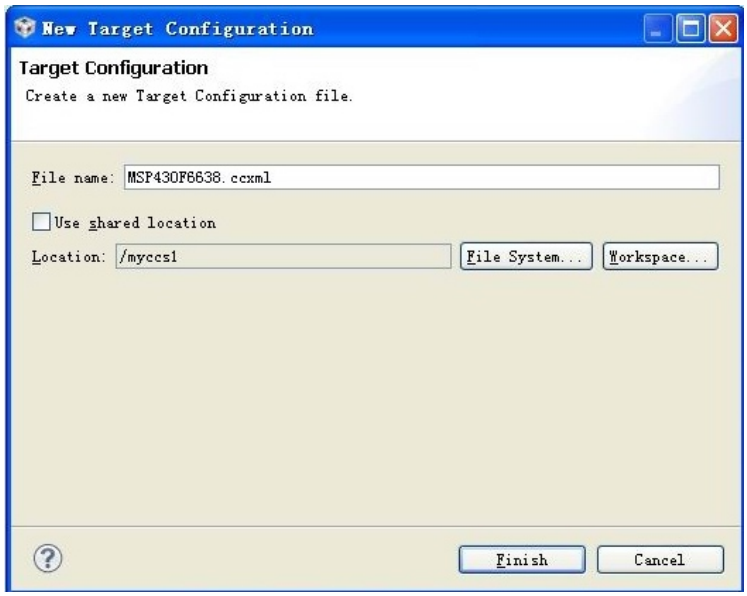

图 2.4.3　目标配置文件名

（4）单击 Finish，将打开目标配置编辑器，如图 2.4.4 所示。

图 2.4.4　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 MSP430F6638。配置完成之后，单击 Save，配置将自动设为活动模式。如图 2.4.5 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。

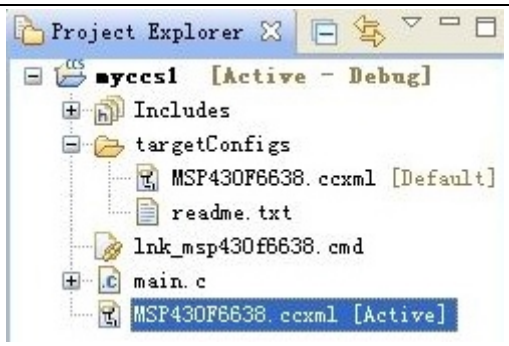

图 2.4.5 项目与配置后的目标文件

2.4.2 启动调试器

（1)如 2.2 节所述,将一个已经建好的工程导入并进行编译: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 rts430xl.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 rts430xl.lib 库文件复制到 CCSV5.3 的库资源文件夹内，其复制路径为：----\tools\compiler\msp430\lib（----为 CCSv5.3 的安装路径)。

编译结果，如图 2.4.6 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。

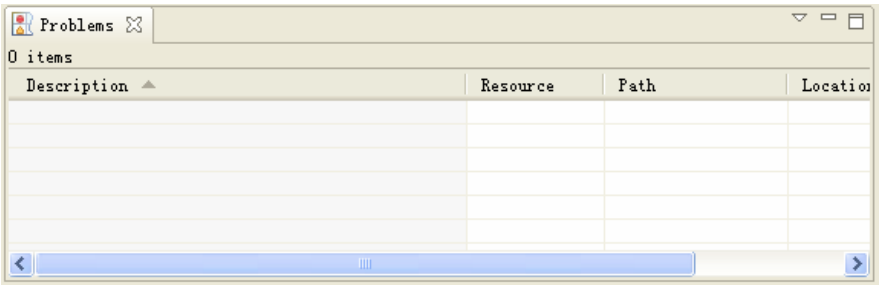

图 2.4.6 工程调试结果

（2）单击绿色的 Debug 按钮  进行下载调试，得到图 2.4.7 所示的界面。

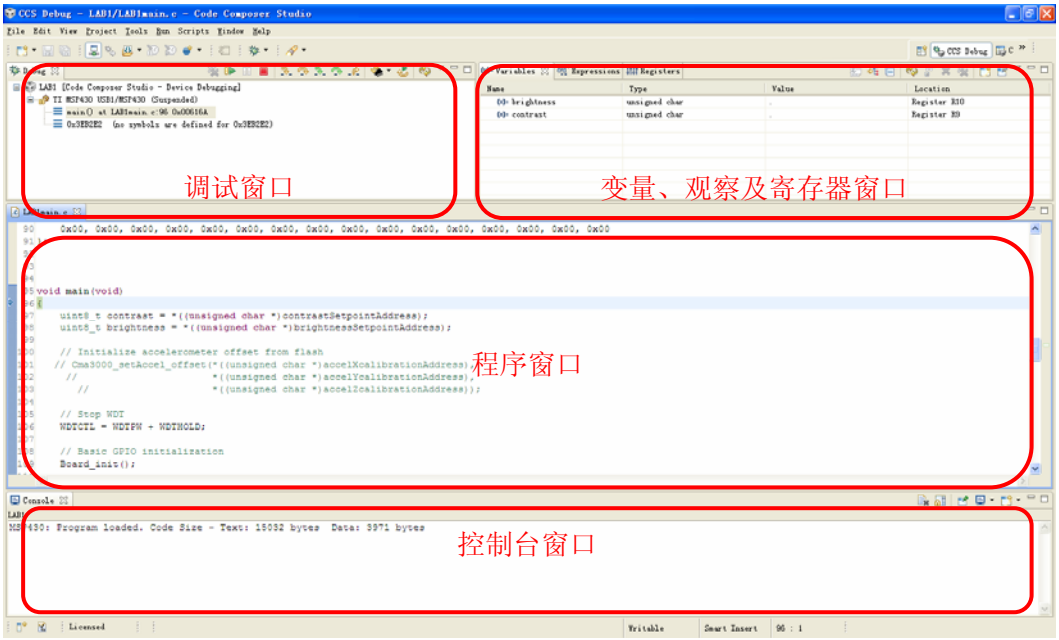

图 2.4.7　调试窗口界面

（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 CCSV5.3 查看变量、寄存器、汇编程序或者是 Memory 等的信息 显示出程序运行的结果，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 2.4.8 所示。

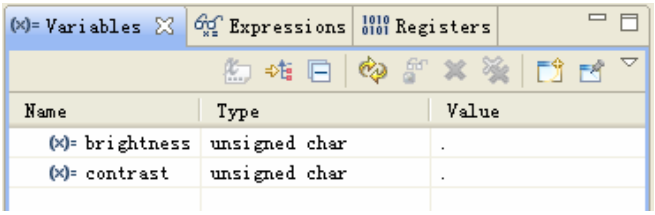

图 2.4.8　变量查看窗口

（5）点击菜单 View→Registers，可以查看到寄存器的值，如图 2.4.9 所示。

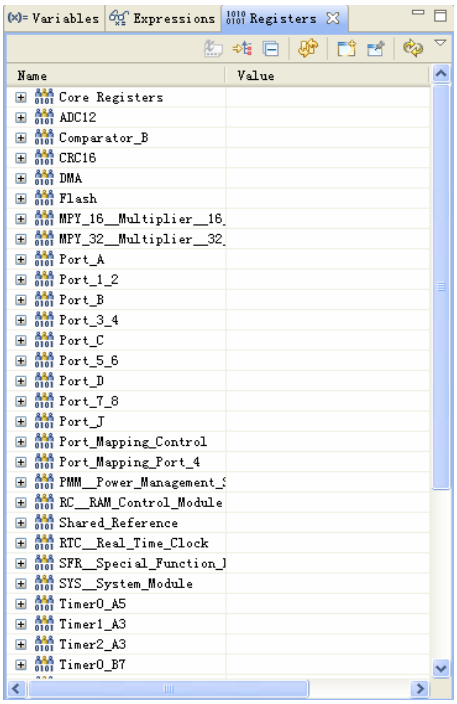

图 2.4.9　寄存器查看窗口

（6）点击菜单 View→Expressions，可以得到观察窗口，如图 2.4.10 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。

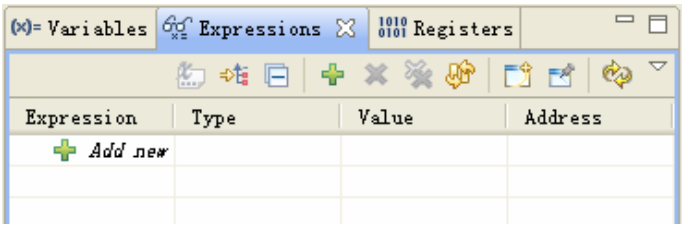

图 2.4.10 观察窗口

（7）点击菜单 View→Disassembly，可以得到汇编程序观察窗口，如图 2.4.11 所示。

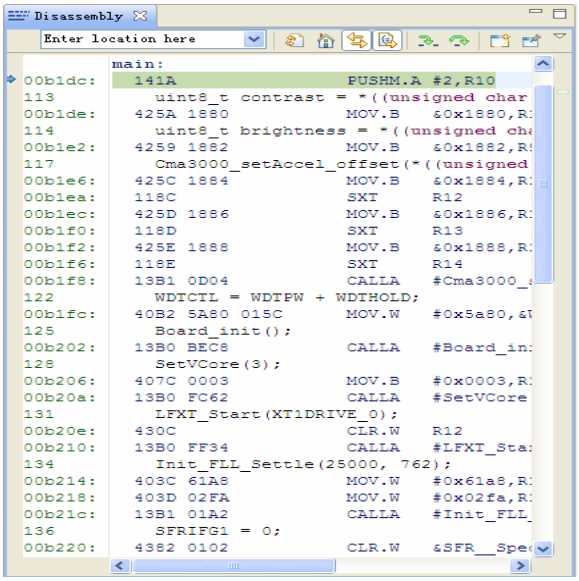

图 2.4.11 汇编程序观察窗口

（8）点击菜单 View→Memory Browser，可以得到内存查看窗口，如图 2.4.12 所示。

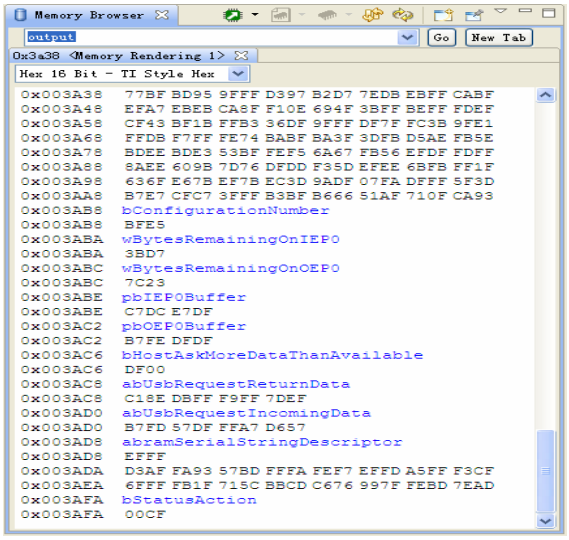

图 2.4.12 内存查看窗口

（9）点击菜单 View→Break points，可以得到断点查看窗口，如图 2.4.13 所示。

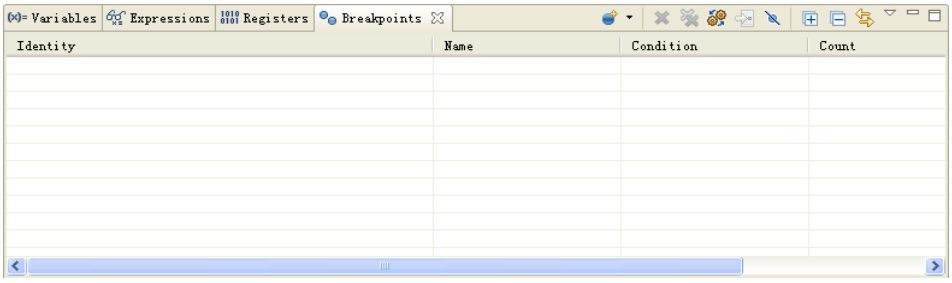

图 2.4.13 断点查看窗口

2.5 CCSv5.3 资源管理器介绍及应用

2.5.1 CC5.3 资源管理器介绍

（1）CCSv5.3 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 CCSv5.3 资源管理器的应用。如图 2.5.1 所示，通过 Help-->Welcome to CCS 打开 CCSv5.3 的欢迎界面。

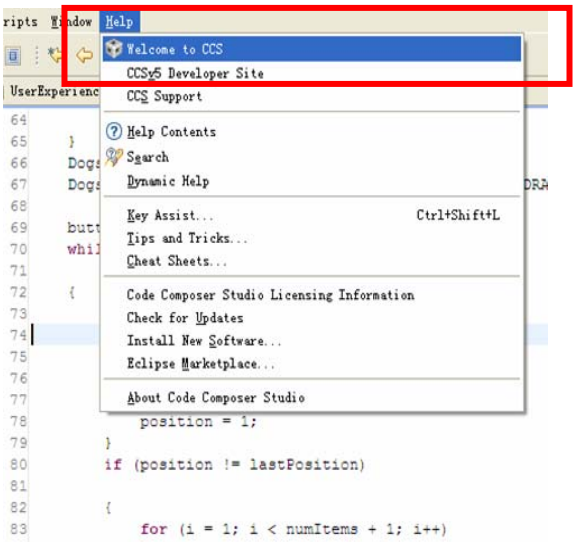

图 2.5.1　欢迎界面打开途径

（2）具体 TI 欢迎界面如图 2.5.2 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 节；利用 CCSv5.3 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 2.2 节；利用 CCSv5.3 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 CCSv5.3 网络教程，学习 CCSv5.3 有关知识。

图 2.5.2　TI 欢迎窗口界面

（3）在“Packages”下拉菜单下选择 ALL，进入 CCSv5.3 资源管理器，如图 2.5.3 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。

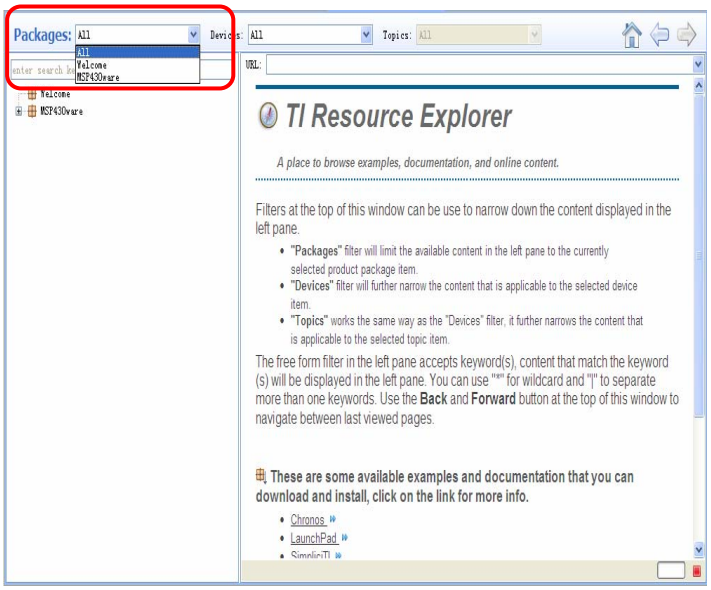

图 2.5.3　CCSv5.3 资源管理器窗口

（4）如图 2.5.4 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。

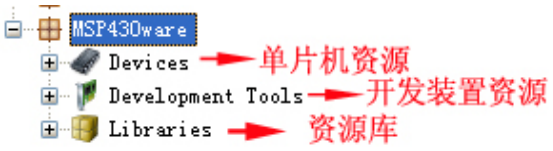

图 2.5.4　MSP430ware 界面

（5）展开 MSP430 单片机资源，得到如图 2.5.5 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。

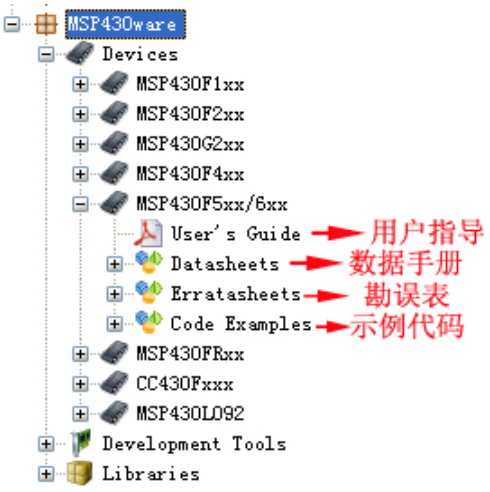

图 2.5.5　单片机资源管理图

（6）展开 Code Examples，在下拉选项上选择 MSP430F663x，在右面窗口中，将得到 MSP430F663x 有关各内部外设的应用程序资源，如图 2.5.6 所示。

MSP430F663x One-Click Project Files

This will automatically create a new project in CCS workspace with the selected example. It prompts you which device to select.

Project Name	Description
MSP430F66xx_1	LED Toggle code example
msp430f66xx_adc_01	ADC12, Sample A0, Set P1.0 if A0 > 0.5*AVcc
msp430f66xx_adc_02	ADC12, Using the Internal Reference
msp430f66xx_adc_05	ADC12, Using an External Reference
msp430f66xx_adc_06	ADC12, Repeated Sequence of Conversions
msp430f66xx_adc_07	ADC12, Repeated Single Channel Conversions
msp430f66xx_adc_08	ADC12, Using A8 and A9 Ext Channels for Conversion
msp430f66xx_adc_09	ADC12, Sequence of Conversions (non-repeated)
msp430f66xx_adc_10	ADC12, Sample A10 Temp and Convert to oC and oF
msp430f66xx_bakmem	LPM4.5, Backup RAM
msp430f66xx_compB_01	COMPB output Toggle in LPM4;
msp430f66xx_compB_02	COMPB output Toggle from LPM4; input channel CB1;
msp430f66xx_compB_03	COMPB interrupt capability;
msp430f66xx_compB_04	COMPB Toggle from LPM4; Ultra low power mode;
msp430f66xx_compB_05	COMPB Hysteresis, CBOUT Toggle in LPM4; High speed mode
msp430f66xx_crc16_01	CRC16, Compare CRC output with software-based algorithm
msp430f66xx_crc16_02	CRC16, fed by DMA, compare w/ software algorithm
msp430f66xx_dac12_1	DAC12_0, Output 1.5V on DAC0
msp430f66xx_dac12_2	DAC12_1, Output 0.75V on DAC1

图 2.5.6　MSP430F663x 应用程序资源

（7）展开 Libraries 资源库，得到如图 2.5.7 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 2.5.7　资源库管理图

2.5.2 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 系列还提供了驱动库文件，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 2.5.8 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。

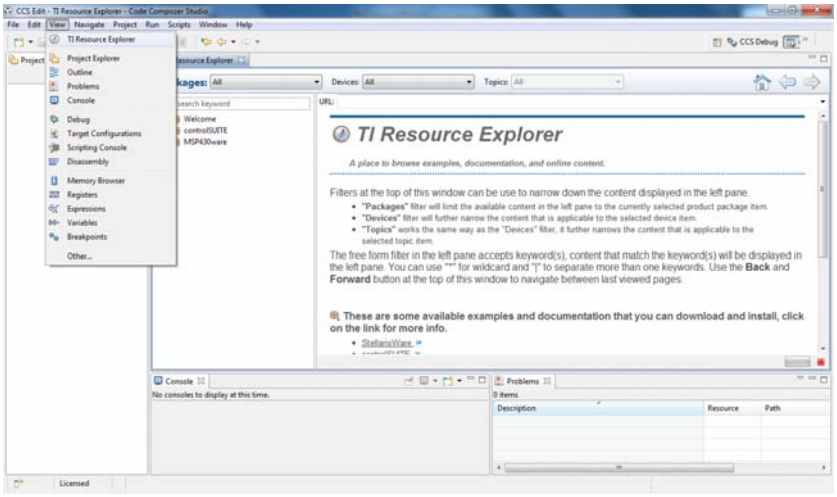

图 2.5.8 TI Resource Explorer 界面

(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 Libaries，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。

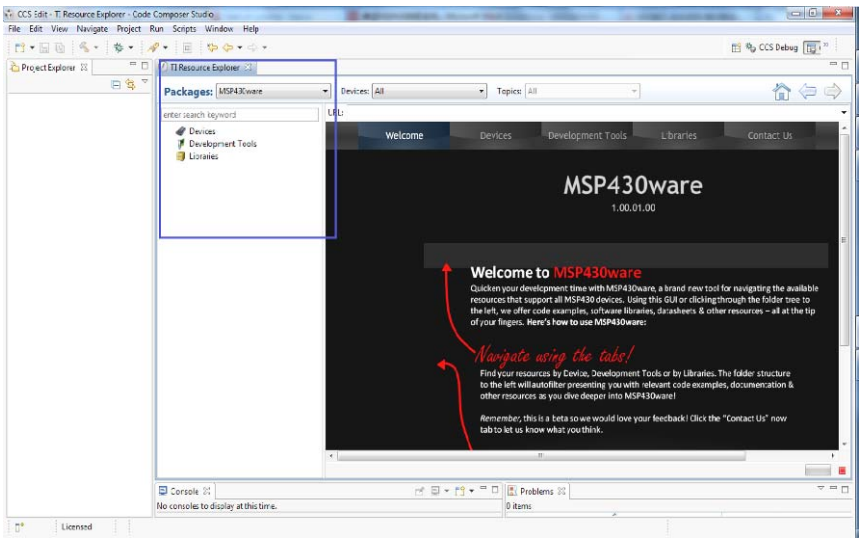

图 2.5.9 TI Resource Explorer 界面

(4) 单击图 2.5.10 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。

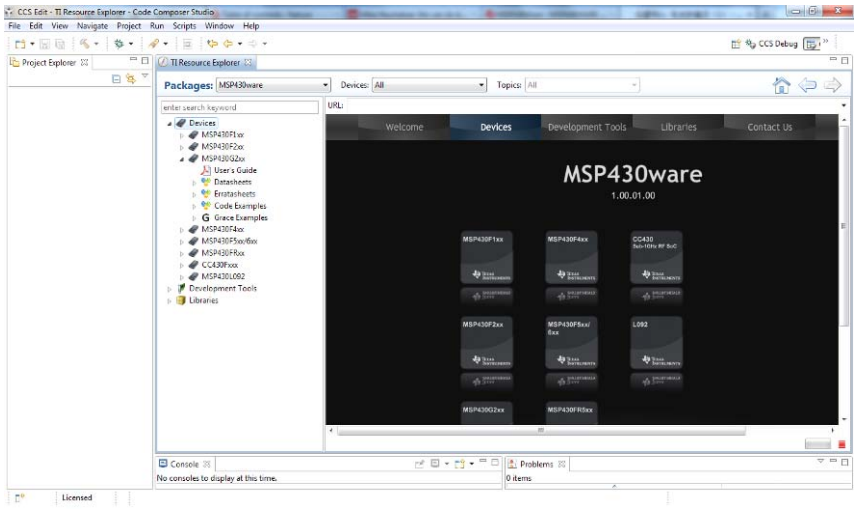

图 2.5.10　MSP430ware 界面

(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 2.5.11 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 2.5.12 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。

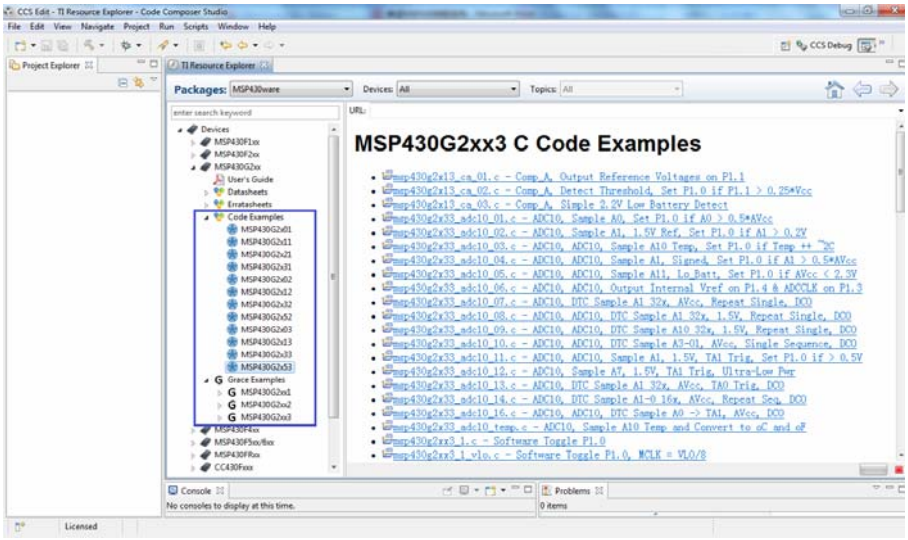

图 2.5.11　MSP430ware 界面

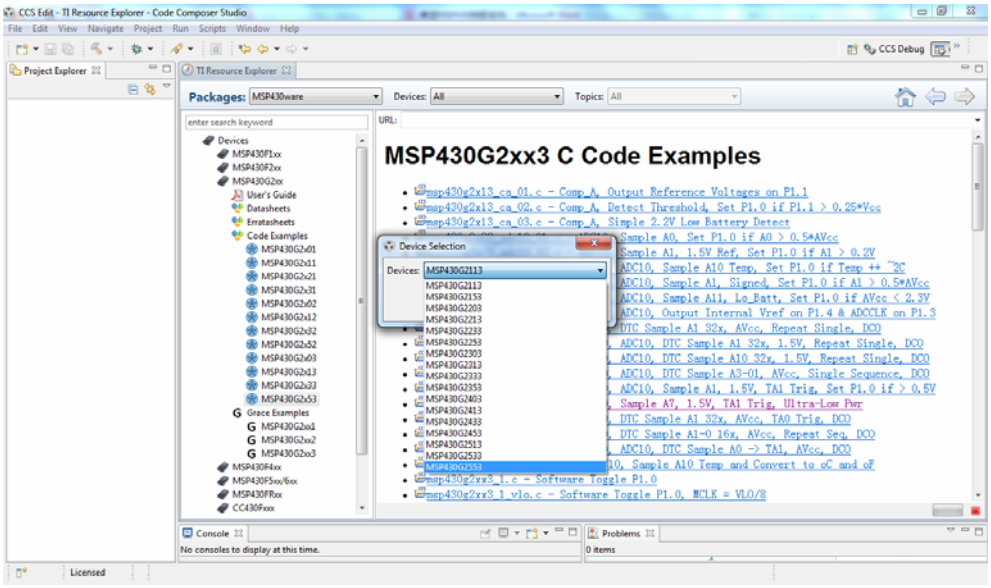

图 2.5.12　MSP430ware 界面

(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。

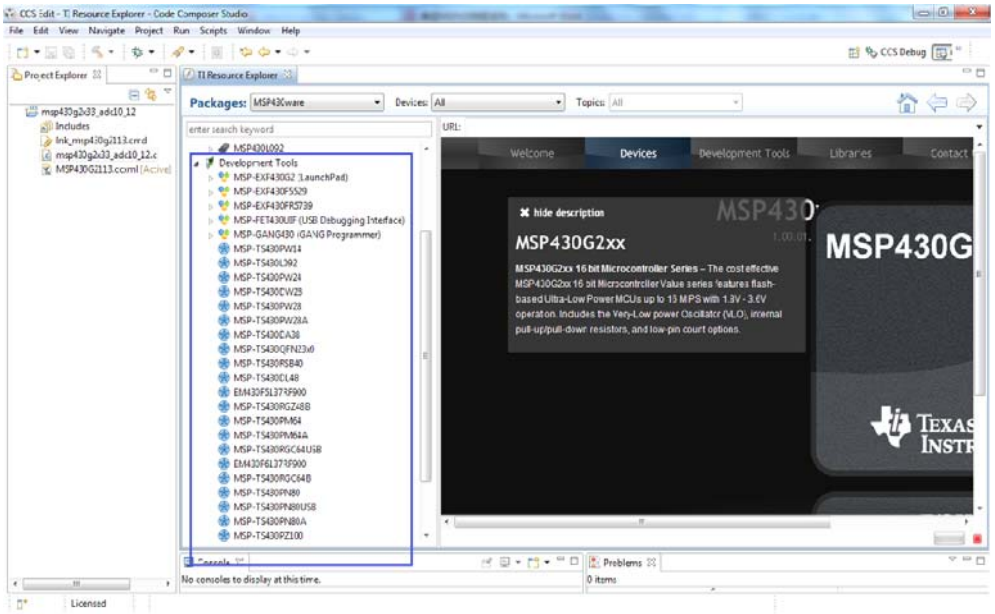

图 2.5.13　MSP430ware 界面

(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，这样用户可以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。

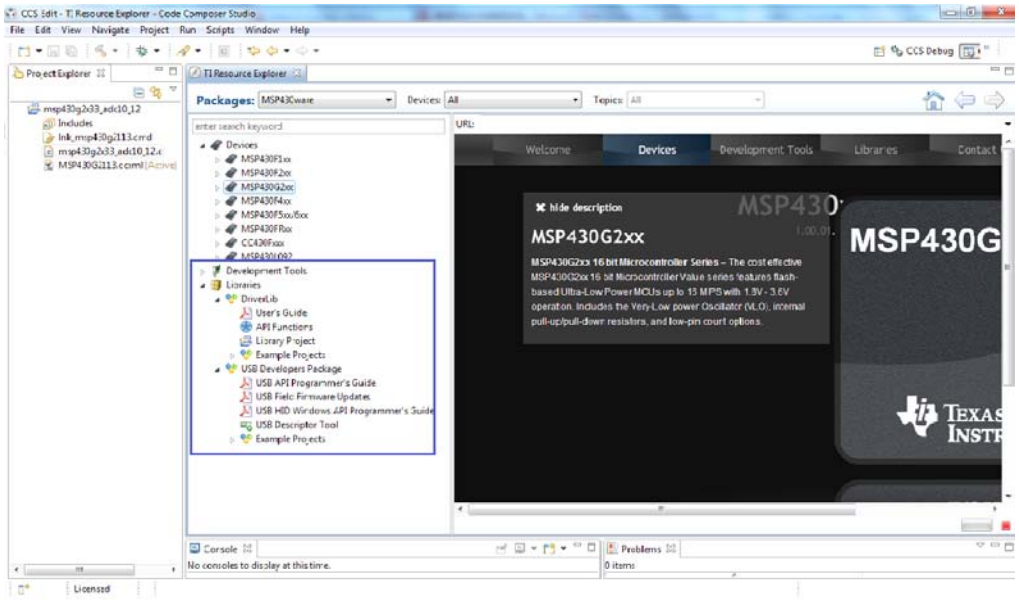

图 2.5.14　MSP430ware 界面驱动库函数

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。

第 3 章 基础实验

MSP430F6638 教学开发系统拥有众多的外设资源，可以完成丰富的实验项目，这一章我们根据 MSP430F6638 芯片中的重要功能模块来设计一系列基础实验项目，通过这些实验项目能够对 MSP430F6638 芯片有一个全面的了解与掌握。

3.1 GPIO模块

这一节作为学习 MSP430F6638 的入门实验，介绍了如何操作 MSP430 的 GPIO 接口，我们设计了两个简单的实验让大家了解 GPIO 的用法，这为后面更深层次的 IO 操作做了铺垫，因此必须熟练掌握。

GPIO 是 MCU 中最基本、最常用的功能单元，信号、数据的输入输出都要依靠 GPIO 接口，GPIO 管脚也是 MCU 芯片上数量最多的管脚，现在 MCU 上的 GPIO 一般都具有多种复用功能，可以通过软件进行配置。

GPIO是MCU与外界交互的重要途径，它具有如下的特性：

- 可以独立控制每个GPIO口的方向（输入/输出模式）
- 可以独立设置每个GPIO的输出状态（高/低电平）
- 所有GPIO口在复位后都有个默认方向（输入/输出）

MSP430F6638 上 GPIO 接口类型丰富，P1～P4 具有输入输出、中断和外部功能模块，这些功能可以通过他们各自 9 个控制寄存器的设置来实现。P1～P4 大多数端口包含 8 个 I/O 引脚，但一些端口可能含有较少引脚。每个 I/O 引脚单独配置输入或输出方向,每个引脚可以单独读或写。P1～P4 端口具有中断功能，每一个 P1～P4 I/O 引脚的中断可以单独启用和配置，在输入信号处于上升或下降沿时提供中断。在一些设备中还会附加具有中断能力的端口，并且包含了它们各自的中断向量。所有端口寄存器（除中断向量寄存器）是用命名的方式来操作的，例如：PAIV 不存在 P1IV 和 P2IV。通常 IO 的配置方式通过软件来实现。

3.1.1 Lab1-1 按键对LED灯的控制实验（查询方式）

3.1.1.1 实验介绍

通过按键对 IO 口 P4.0 的操作，查询实现高低电平检测，从而对与 LED 相连的 P4.5 的电平控制，实现 LED 灯的亮与灭。

3.1.1.2 实验目的

- 掌握对 IO 口的查询操作和 IO 基本操作的流程
- 对部分引脚功能有一个初步了解

3.1.1.3 实验原理

MSP430各种端口有大量的控制寄存器供用户操作，这最大限度提供了输入/输出的灵活性。下表是端口的各个寄存器的使用方式：

名称	缩写	BIT=1	BIT=0
方向寄存器	PxDIR	输出模式	输入模式
输入寄存器	PxIN	输入高电平	输入低电平
输出寄存器	PxOUT	输出高电平	输出低电平
上下拉电阻使能寄存器	PxREN	使能	禁用
功能选择寄存器	PxSEL	外设功能	IO端口
驱动强度寄存器	PxDS	高强度	低强度
中断使能寄存器	PxIE	允许中断	禁止中断
中断触发沿寄存器	PxIES	下降沿置位	上升沿置位
中断标志寄存器	PxIFG	有中断请求	无中断请求

表3.1.1　端口寄存器说明

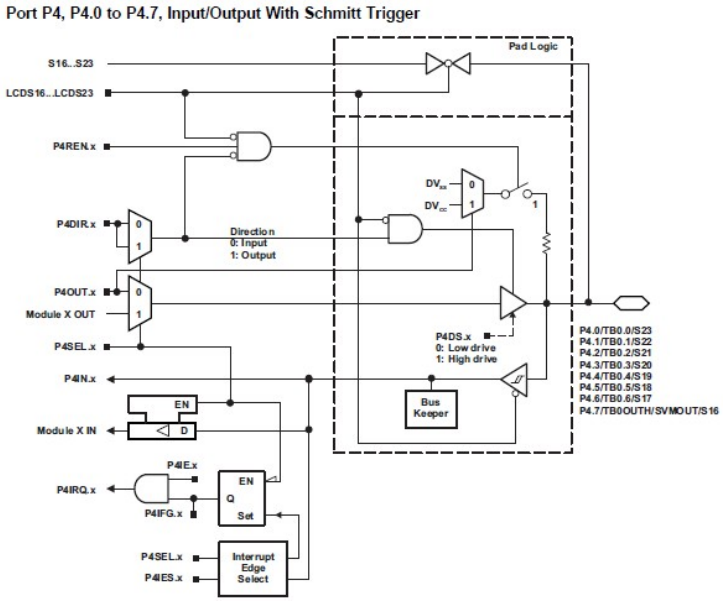

图3.1.1 MSP430F6638 P4 口功能框图

主板上右下角S3~S7按键与MSP430F6638 P4.0~P4.4口连接：

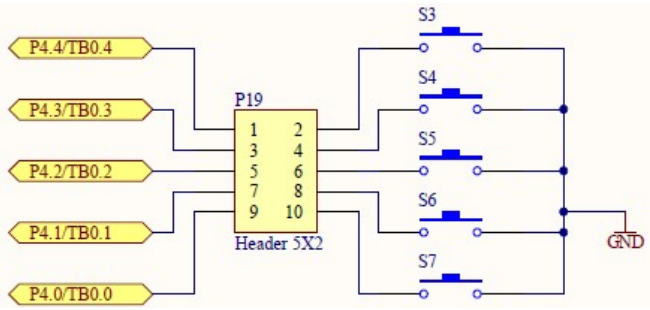

图3.1.2 按键模块原理图

与P4口相关的寄存器地址信息见下表：

REGISTER DESCRIPTION	REGISTER	OFFSET
Port P3 input	P3IN	00h
Port P3 output	P3OUT	02h
Port P3 direction	P3DIR	04h
Port P3 pullup/pulldown enable	P3REN	06h
Port P3 drive strength	P3DS	08h
Port P3 selection	P3SEL	0Ah
Port P3 interrupt vector word	P3IV	0Eh
Port P3 interrupt edge select	P3IES	18h
Port P3 interrupt enable	P3IE	1Ah
Port P3 interrupt flag	P3IFG	1Ch
Port P4 input	P4IN	01h
Port P4 output	P4OUT	03h
Port P4 direction	P4DIR	05h
Port P4 pullup/pulldown enable	P4REN	07h
Port P4 drive strength	P4DS	09h
Port P4 selection	P4SEL	0Bh
Port P4 interrupt vector word	P4IV	1Eh
Port P4 interrupt edge select	P4IES	19h
Port P4 interrupt enable	P4IE	1Bh
Port P4 interrupt flag	P4IFG	1Dh

图3.1.3 寄存器描述

如上图所示，P4端口的基址为0x0220，我们需要设置两个相关的寄存器：P4OUT和P4DIR。其中P4DIR为方向寄存器，P4OUT为数据输出寄存器。

主板上右下角LED1~LED5指示灯与MSP430F6638 P4.5~P4.7、P5.7、P8.0连接：

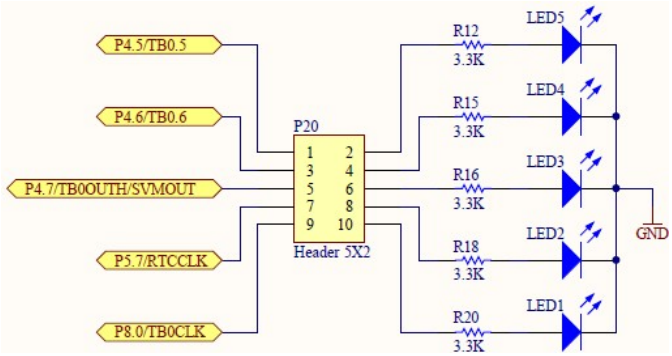

图3.1.4 LED指示灯模块原理图

P4IN和P4OUT分别是输入数据和输出数据寄存器，PDIR为方向寄存器，P4REN为使能寄存器：

```
#define P4IN        (PBIN_H)     /* Port 4 Input */
#define P4OUT       (PBOUT_H)    /* Port 4 Output */
#define P4DIR       (PBDIR_H)    /* Port 4 Direction */
#define P4REN       (PBREN_H)    /* Port 4 Resistor Enable */
```

3.1.1.4 程序分析

- 编程思路：

关闭看门狗定时器后，对 P4.0 的输出方式、输出模式和使能方式初始化，然后进行查询判断，最后对 P4.0 的电平高低分别作处理来控制 LED 灯。

- 程序流程图：

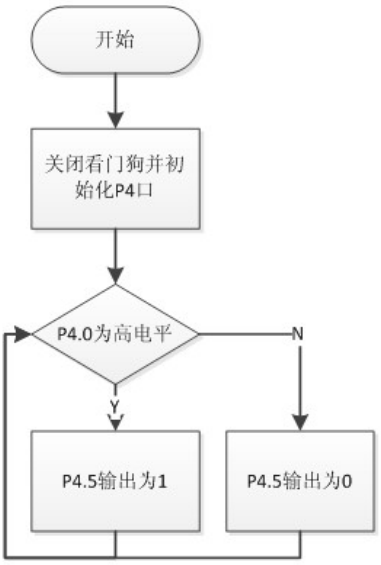

图3.1.5 程序流程图

- 关键代码分析

P4 口初始化函数

```
#include <msp430f6638.h>
void main(void)
{
 WDTCTL = WDTPW + WDTHOLD;          //关闭看门狗定时器
  P4DIR |= BIT5;                    //设置4.5口为输出模式
  P4REN |= BIT0;                    //使能P4.0上下拉电阻功能
  P4OUT |= BIT0;                    //置P4.0为上拉电阻方式
```

循环判断部分

```
 while (1)
  {
    if (P4IN & BIT0)                //判断是否按下键，S7按键按下P4.0=0，抬起P4.0=1
     P4OUT |= BIT5;                 //S7按键抬起，P4.5输出高（LED5点亮）
    else
     P4OUT &= ~BIT5;
  }
}
```

3.1.1.5 实验步骤与现象

- **实验步骤**

1. 将 PC 和板载仿真器通过 USB 线相连；
2. 打开CCS 集成开发工具，选择Project->Import Existing CCS Eclipse Project,导入所建文件夹中相应的工程；
3. 选择  对该工程进行编译链接，生成.out 文件。然后选择  ，将程序序下载到实验板中。程序下载完毕之后，可以选择  全速运行程序，也可以选择  单步调试程序，选择 F3 查看具体函数。也可以程序下载之后，按下  终止调试，软件界面恢复到原编辑程序的画面。再按下实验板的复位键，运行程序。（调试方式下的全速运行和直接上电运行程序在时序有少许差别，建议上电运行程序）。

- **实验现象**

按下 S7 后，LED5 熄灭，松开后恢复点亮。

3.1.1.6 实验思考

1. MSP430 的 IO 口都具备哪些功能？
2. 与 IO 口相关的寄存器有哪些？
3. 为何要关闭看门狗定时器？

3.1.2 Lab1-2 多个按键对LED灯的控制实验（查询方式）

3.1.2.1 实验介绍

该实验结合了各个按键的触发方式以及基本的定时器操作，将各个按键对应的 IO 口添加到一个结构体中操作 LED 灯，这使得代码简单易读，比上一个实验的简单查询方式更加规范，操作也更加方便。

3.1.1.2 实验目的

- 深入了解 IO 口的原理和操作方式
- 学习时钟定时使用
- 学习结构体定义各个指针变量的方法
- 掌握循环调用各个结构体内变量的方法

3.1.2.3 实验原理

按键与 LED 模块的电路连接在上一个实验中做过介绍，这里不再赘述，请阅读 3.1.1.3 章节。

由于 LED 灯不在同一个端口上，为了可以直接使用 int 进行索引操作，程序中将 IO 口的相关寄存器的地址声明为结构体并保存在数组中。即首先声明了保存寄存器地址的结构：

```
typedef     struct
{ const     volatile    uint8_t* PxIN;
volatile    uint8_t* PxOUT;
volatile    uint8_t* PxDIR;
volatile    uint8_t* PxREN;
volatile    uint8_t* PxSEL; } GPIO_TypeDef;
//之后声明所需操作的 IO 端口的结构体 GPIO4、GPIO5、GPIO8，如：
const  GPIO_TypeDef GPIO4 =    { &P4IN, &P4OUT, &P4DIR, &P4REN, &P4SEL };
const  GPIO_TypeDef GPIO5 =    { &P5IN, &P5OUT, &P5DIR, &P5REN, &P5SEL };
const  GPIO_TypeDef GPIO8 =    { &P8IN, &P8OUT, &P8DIR, &P8REN, &P8SEL };
```

3.1.2.4 程序分析

- 编程思路

设置各个引脚变量并且初始化，开启定时器 XT1，然后循环检查按键是否按下，如果按下，就把 IO 电平取反，并且延时 100ms（此处用延时来确定了扫描频率）。

- 程序流程图

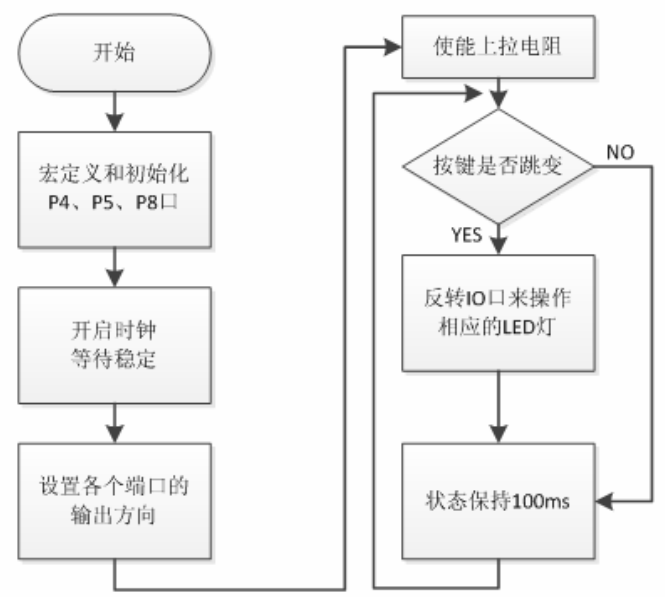

图3.1.6 程序流程图

- 关键代码分析

宏定义和参数定义

```
Typedefstruct                          //以指针形式定义P8口的各个寄存器
{
constvolatile uint8_t* PxIN;           //定义一个不会被编译的无符号字符型变量
volatile uint8_t* PxOUT;
volatile uint8_t* PxDIR;
volatile uint8_t* PxREN;
volatile uint8_t* PxSEL;
} GPIO_TypeDef;

constGPIO_TypeDef GPIO4 ={ &P4IN, &P4OUT, &P4DIR, &P4REN, &P4SEL};
constGPIO_TypeDef GPIO5 ={&P5IN, &P5OUT, &P5DIR, &P5REN, &P5SEL};
constGPIO_TypeDef GPIO8 ={&P8IN, &P8OUT, &P8DIR, &P8REN, &P8SEL};
constGPIO_TypeDef* LED_GPIO[5] = {&GPIO4, &GPIO4, &GPIO4, &GPIO5, GPIO8};
const uint8_t LED_PORT[5] = {BIT5, BIT6, BIT7, BIT7, BIT0};
```

主函数

```
while(BAKCTL & LOCKIO) // Unlock XT1 pins for operation
     BAKCTL &= ~(LOCKIO);
UCSCTL6 &= ~XT1OFF;//启动XT1
while (UCSCTL7 & XT1LFOFFG)                //等待XT1稳定
    UCSCTL7 &= ~(XT1LFOFFG);
UCSCTL4 = SELA__XT1CLK + SELS__REFOCLK + SELM__REFOCLK;
                                           //时钟设为 XT1，频率较低，方便软件延时
int i;
for(i=0;i<5;++i)
  *LED_GPIO[i]->PxDIR |= LED_PORT[i];     //设置各LED灯所在端口为输出方向
P4REN |= 0x1F;                             //使能按键端口上的上下拉电阻
P4OUT |= 0x1F;                             //上拉状态

uint8_t last_btn = 0x1F, cur_btn, temp;
while(1)
```

```
{
cur_btn = P4IN & 0x1F;
  temp = (cur_btn ^ last_btn) &last_btn;            //找出刚向下跳变的按键
last_btn = cur_btn;
int i;
for(i=0;i<5;++i)
if(temp & (1 << i))
      *LED_GPIO[i]->PxOUT ^= LED_PORT[i];           //翻转对应的LED
  __delay_cycles(3276);                             //延时大约100ms
}
```

3.1.2.5 实验步骤与实验现象

- **实验步骤**

按照上次实验内容将生成的相应文件烧入板子内，依次按下各个键S3、S4、S5、S6、S7观察LED灯的变化。

- **实验现象**

依次按下 S3、S4、S5、S6、S7 按键后，LED1、LED2、LED3、LED4、LED5 依次点亮，再一次按下时，对应的 LED 指示灯熄灭。

3.1.2.6 实验思考

1. 如何处理按键产生的毛刺、抖动现象？

3.2 段式液晶模块

段式液晶 LCD 因为具有功耗低、结构简单、性能稳定等优点，大量的在电子表、计算器、万用表等各种设备上使用。MSP430F6638 芯片上有 2 组专用的段式液晶接口，能够方便的连接段式液晶屏幕。

MSP430F6638 内部集成了 LCD_B 与 LCD_C 两个段式液晶控制器，本节实验使用的是 LCD_B 控制器，这是一个能对 160 段进行对比度控制的 LCD 驱动器，有 LCD 段码寄存器 LCDMEM1—LCDMEM20，其地址从 0x91 到 0xA4，每一个地址可以存放一个八位，每一位对应液晶上面的一段，即一个地址对应液晶上面就是一个字。将 LCD 的段码整理好之后，将这 8 位的段码直接写入芯片预留的 LCD 寄存器写入端口数组即可在段式液晶屏上显示想要的数据。这些端口就是 MSP430F6638 里面的 LCDMEM[x]数组，x 表示要显示的位，该位就是段式液晶屏上面的位置，本节实验所用的位 6 位段式液晶，则 x 的取值范围应为 0—5。

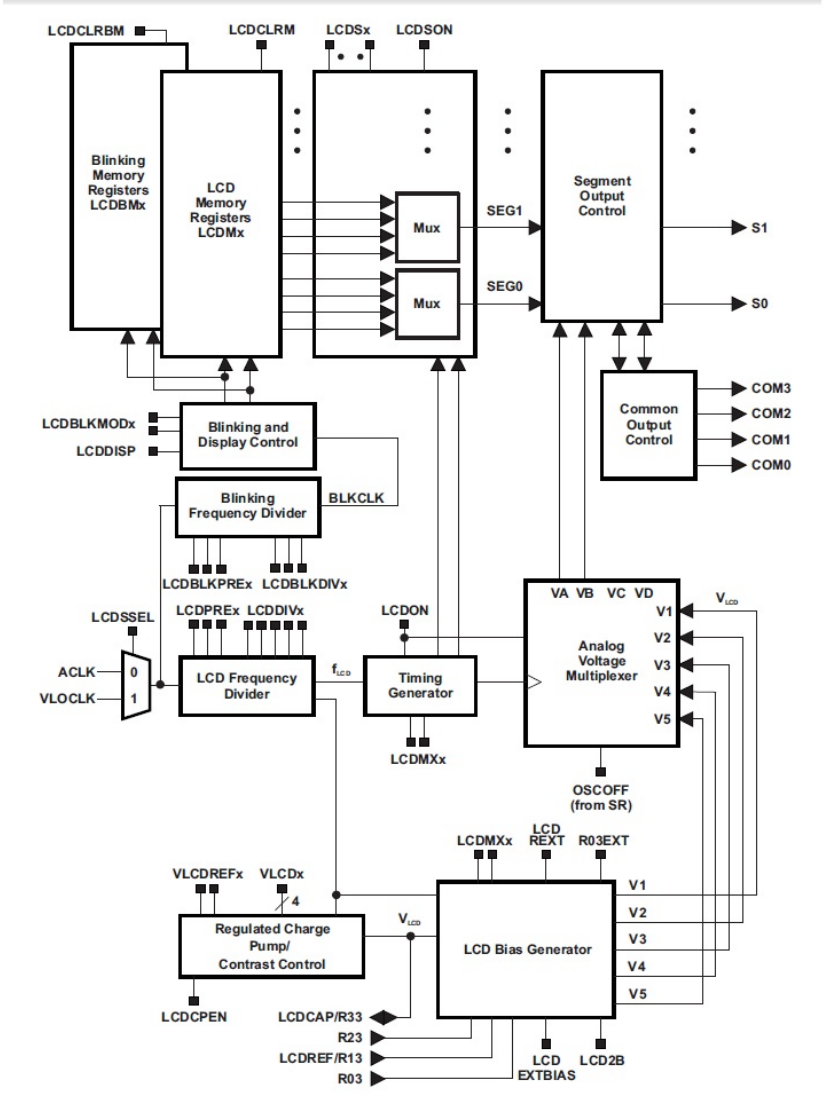

Figure 32-1. LCD_B Controller Block Diagram

图3.2.1 LCD_B控制器框图

Associated Common Pins	3	2	1	0	3	2	1	0		
Register	7							0	n	Associated Segment Pins
LCDM20	--	--	--	--	--	--	--	--	38	39, 38
LCDM19	--	--	--	--	--	--	--	--	36	37, 36
LCDM18	--	--	--	--	--	--	--	--	34	35, 34
LCDM17	--	--	--	--	--	--	--	--	32	33, 32
LCDM16	--	--	--	--	--	--	--	--	30	31, 30
LCDM15	--	--	--	--	--	--	--	--	28	29, 28
LCDM14	--	--	--	--	--	--	--	--	26	27, 26
LCDM13	--	--	--	--	--	--	--	--	24	25, 24
LCDM12	--	--	--	--	--	--	--	--	22	23, 22
LCDM11	--	--	--	--	--	--	--	--	20	21, 20
LCDM10	--	--	--	--	--	--	--	--	18	19, 18
LCDM9	--	--	--	--	--	--	--	--	16	17, 16
LCDM8	--	--	--	--	--	--	--	--	14	15, 14
LCDM7	--	--	--	--	--	--	--	--	12	13, 12
LCDM6	--	--	--	--	--	--	--	--	10	1, 10
LCDM5	--	--	--	--	--	--	--	--	8	9, 8
LCDM4	--	--	--	--	--	--	--	--	6	7, 6
LCDM3	--	--	--	--	--	--	--	--	4	5, 4
LCDM2	--	--	--	--	--	--	--	--	2	3, 2
LCDM1	--	--	--	--	--	--	--	--	0	1, 0
	Sn+1				Sn					

Figure 32-2. LCD Memory - Example for 160 Segments Maximum

图3.2.2 LCD_B段码寄存器

3.2.1 Lab2-1 段式液晶实验

3.2.1.1 实验介绍

字段式液晶，就是常用于计算器、电子表、数字万用表等显示的 LCD，显示类型与数码管类似；本实验通过 MSP430F6638 控制段式液晶模块，使在段式液晶上循环显示数字。

3.2.1.2 实验目的

- 了解 MSP430F6638 的 LCD 驱动模块
- 了解段式液晶显示的原理

3.2.1.3 实验原理

段式液晶的驱动信号由两个部分组成，第一部分是公共端偏压信号（COM），主板上的段式液晶模块是 1/3 偏压（bias）的，也就是加在液晶模块每一位上的电压分为 VCC—2/3VCC—1/3VCC—GND 四个等级，这个偏压信号可以通过软件设置让 MSP430F6638 内部的 LCD 驱动模块自动生成，其引脚 COM0—COM3 就是这些电压的输出引脚，直接与段式液晶屏的 COM0—COM3 相连即可。第二部分信号是驱动信号（S0—S11），与液晶屏上的 S0—S11 连接。从下图中可以看出，S9—S12 具有复用关系，当不使用段式液晶时可以将 P10 处 S9~S11 三个跳线拔掉，以免屏幕上出现干扰。

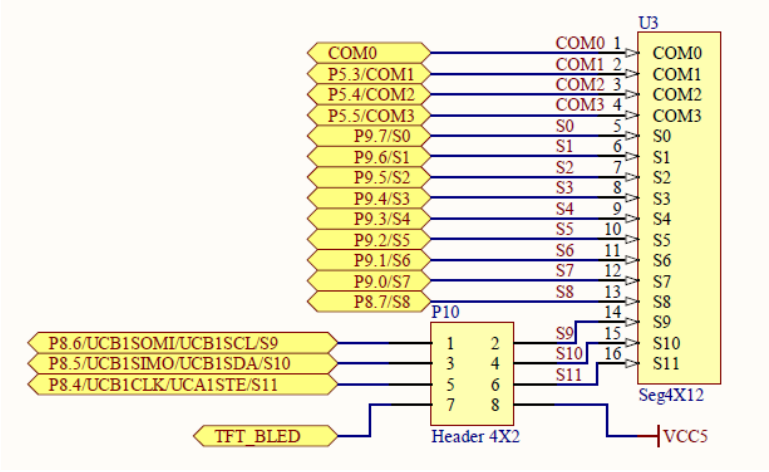

图3.2.3 段式液晶模块电路原理图

主板上的段式液晶显示采用 4MUX 模式，其段式液晶显示参数对应关系如下图所示，即 4 个公共端（COM0—COM3）及 12 个驱动端（S0—S11），为了显示出我们想要的数字，当分别给公共端与驱动端合适液晶信号时，就会显示对应的数码（其中 X1、X2、X3 是三个电池符号，4P、5P、6P 是后面三位字符的三个小数点）。

PIN	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
C1	C1				1D	X2	2D	X3	3D	X1	4D	4P	5D	5P	6D	6P
C2		C2			1E	1C	2E	2C	3E	3C	4E	4C	5E	5C	6E	6C
C3			C3		1G	1B	2G	2B	3G	3B	4G	4B	5G	5B	6G	6B
C4				C4	1F	1A	2F	2A	3F	3A	4F	4A	5F	5A	6F	6A

图3.2.4　段式液晶显示参数对应关系

3.2.1.4 程序分析

- **编程思路**

在写程序操作段式液晶之前首先就是要先配置好系统时钟，然后要配置好与段式液晶相连的 IO 口，即确定相应 IO 口的工作模式，再通过操作 IO 口对段式液晶进行初始化，最后再通过控制 IO 口使段式液晶显示出需要的信息来。配置操作可直接读写 MSP430F6638 内部 LCD 驱动器相关寄存器来完成，需设置 P5.3、P5.4、P5.5 作为 LCD 的 COM 口，S0—S11 为 LCD 的段选，清空 LCD 寄存器，启动 LCD 模块，配置相关的寄存器包括 P5SEL、LCDBPCTL0、LCDBCTL0、LCDBMEMCTL、LCDBCTL0，寄存器详细说明可阅读相关文档；显示过程首先确定位及其相应位的段码数据，即确定 LCDMEM[x]的值，x 表示相应的位，这一过程可直接调用已有的液晶驱动函数来完成。

- **程序流程图**

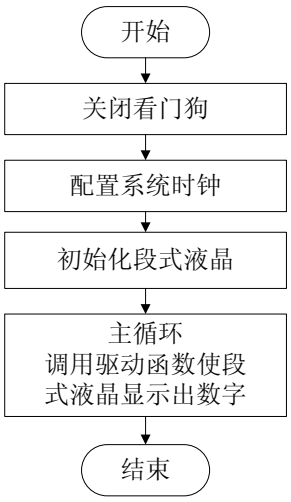

图3.2.5　程序流程图

- **关键代码分析**

首先需要了解几个驱动函数的用法：

- voidLCDSEG_SetDigit(intpos, int value);

  该函数的作用为在段式液晶的第pos位上显示数字value，其中1<=pos<=6,0<=value<=F,当value为-1时表示清除该位上数字的显示。

- voidLCDSEG_DisplayNumber(int32_t num, intdppos);

  该函数的作用为在段式液晶上显示一个num的整数，dppos为小数点要显示的位置，0<=num<=999999,0<=dppos<=3,当dppos=0表示不显示小数点。

main.c

```
#include<msp430f6638.h>
```

```
#include<stdint.h>
#include<stdio.h>
#include<string.h>
#include "dr_lcdseg.h"                          //调用段式液晶驱动头文件
#define XT2_FREQ      4000000
#define MCLK_FREQ 16000000
#define SMCLK_FREQ 4000000
voidinitClock()                                 //系统时钟初始化函数
{
while(BAKCTL & LOCKIO)                          //解锁XT1引脚操作
      BAKCTL &= ~(LOCKIO);
      UCSCTL6 &= ~XT1OFF;                       //启动XT1，选择内部时钟源
P7SEL |= BIT2 + BIT3;                           //XT2引脚功能选择
      UCSCTL6 &= ~XT2OFF;                       //启动XT2
      while (SFRIFG1 & OFIFG)                   //等待XT1、XT2与DCO稳定
      {
     UCSCTL7 &= ~(DCOFFG+XT1LFOFFG+XT2OFFG);
     SFRIFG1 &= ~OFIFG;
}
  UCSCTL4 = SELA__XT1CLK + SELS__XT2CLK + SELM__XT2CLK;     //避免DCO调整中跑飞
  UCSCTL1 = DCORSEL_5; //6000kHz~23.7MHz
  UCSCTL2 = MCLK_FREQ / (XT2_FREQ / 16);    //XT2频率较高，分频后作为基准可获得更高的精度
  UCSCTL3 = SELREF__XT2CLK + FLLREFDIV__16;                 //XT2进行16分频后作为基准
  while (SFRIFG1 & OFIFG)                                   //等待XT1、XT2与DCO稳定
  {
    UCSCTL7 &= ~(DCOFFG+XT1LFOFFG+XT2OFFG);
    SFRIFG1 &= ~OFIFG;
  }
  UCSCTL5 = DIVA__1 + DIVS__1 + DIVM__1;                    //设定几个CLK的分频
  UCSCTL4 = SELA__XT1CLK + SELS__XT2CLK + SELM__DCOCLK;     //设定几个CLK的时钟源
}
voidmain(void)
{
      unsignedchari,num1;
      int32_t num2;
     WDTCTL = WDTPW | WDTHOLD;                  //停止看门狗
initClock();                                    //配置系统时钟
initLcdSeg();                                   //初始化段式液晶
while(1)                                        //进入程序主循环，循环显示数字
     {
      for(i=0;i<6;i++)
      {
            for(num1=0;num1<10;num1++)
            {
                  LCDSEG_SetDigit(i,num1);    __delay_cycles(MCLK_FREQ/5);      //延时200ms
                LCDSEG_SetDigit(i,-1);
                  }
      }
for(num2=111111;num2<1000000;num2=num2+111111)
      {
LCDSEG_DisplayNumber(num2,0);         __delay_cycles(MCLK_FREQ/2);              //延时500ms
      }
      for(i=0;i<6;i++)
            LCDSEG_SetDigit(i,-1);                                              //段式液晶清屏
      __delay_cycles(MCLK_FREQ);                                                //延时1000ms
     }
}
```

3.2.1.5 实验步骤与现象

- **实验步骤**
  1. 将跳线帽接到 MSP430F6638 试验箱 P10 处引脚 S9、S10、S11 上
  2. 打开 CCS5 开发软件，创建 MSP430F6638 的一个空工程
  3. 将段式液晶驱动文件 dr_lcdseg.c 及 dr_lcdseg.h 添加到工程下
  4. 在主函数中编写以上的代码，连接开发板与计算机，编译运行将程序下载到实验板中运行

- **实验现象**

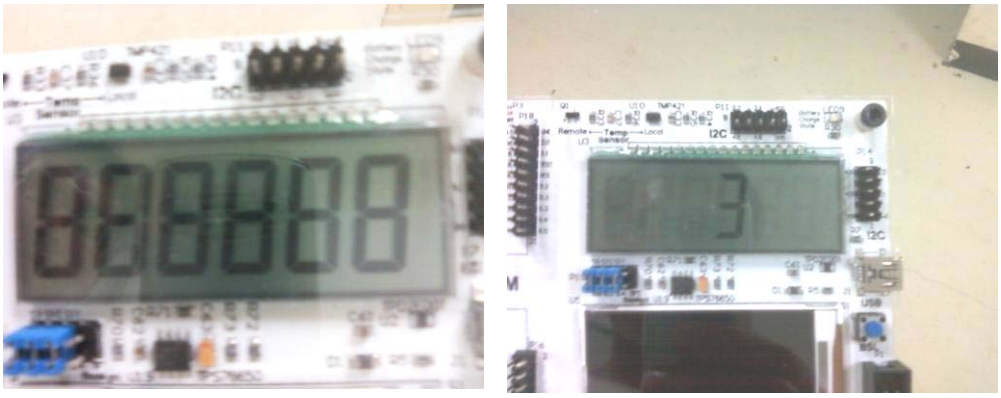

图 3.2.6　多个字符与单个字符显示

3.2.1.6 实验思考

1. 如何在段式液晶上显示出小数来，编写一个能显示小数的函数。
2. 段式液晶上还包含了电池电量显示的 3 段，将这 3 段也显示出来，用来表示电池所剩的电量。
3. 进行其他数据的显示，如将 ADC 转换所得的数据在段式液晶上显示出来。

3.3 中断与低功耗工作模式

中断与低功耗工作模式是 MCU 两个重要功能，二者间存在在紧密联系，比如当我们从一种功耗模式进入到另外一种功耗模式往往都是通过中断实现的，因此我们将这两个功能放在一起介绍。

3.3.1 中断

中断装置和中断处理程序统称为中断系统，中断系统是单片机中重要的组成部分。中断是 CPU 对系统发生的某个事件作出的反应，暂停正在运行中的程序，转去执行处理相应的中断事件，完毕后返回被中断的程序继续运行。中断系统的使用大大提高了 CPU 的效率，中断的实现由软件和硬件综合完成。

MSP430F6638 的中断装置比较多，其主要的有 IO 口中断、定时器中断、串口中断、外部中断、ADC 转换中断、看门狗中断、捕获比较中断等，这些装置都能引发中断事件。中断分为不可屏蔽（nonmaskable）中断与可屏蔽（maskable）中断两大类，各种中断的优先级也是不同的，MSP430F6638 的一些中断源、中断标志及优先级等如下图所示。

Interrupt Source	Interrupt Flag	System Interrupt	Word Address	Priority
Reset: power up, external reset watchdog, flash password	... WDTIFG KEYV	... Reset	... 0FFFEh	... Highest
System NMI: PMM		(Non)maskable	0FFFCh	...
User NMI: NMI, oscillator fault, flash memory access violation	... NMIIFG OFIFG ACCVIFG	... (Non)maskable (Non)maskable (Non)maskable	... 0FFFAh	... ...
Device specific			0FFF8h	...
...			...	...
Watchdog timer	WDTIFG	Maskable	...	...
...			...	...
Device specific			...	...
Reserved		Maskable	...	Lowest

图 3.3.1　中断资源表

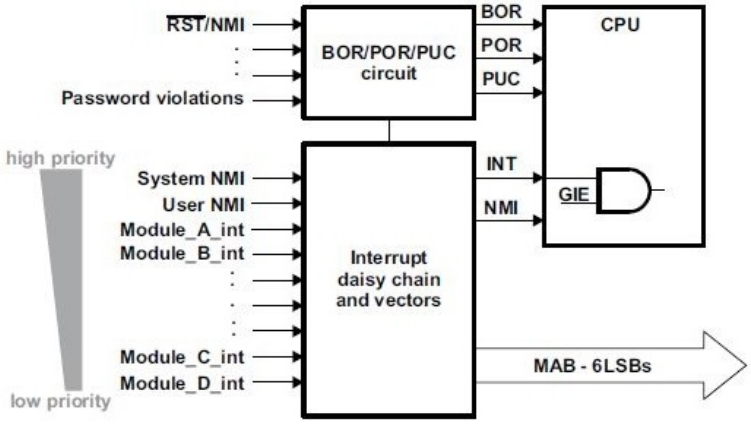

图3.3.2　中断资源示意图

3.3.2 低功耗工作模式

低功耗即是将能量消耗尽可能的降低，TI 的 MSP430 系列单片机最突出的特点之一就是超低功耗，它采用 RISC 内核结构，非常适用于应用电池的场合或手持设备，在超低功耗方面 MSP430F6638 能够实现在 1.8V—3.6V 电压和 1MHz 的时钟条件下运行，耗电电流（0.1—400uA 之间），其中 MSP430F6638 提供功耗模式，且不同功耗模式间可以方便切换是实现系统超低功能的重要原因之一。

实现低功耗方面主要有三个方法：一是降低工作电压，MSP430 位控制器在 1.8V 低电压下仍可以工作；二是把暂时不用的模块功能给关闭掉实现节能，MSP430 把一个芯片分成了 N 个不同模块的部分，不用的部分功能模块关闭掉使电流近似为零；第三是根据实际情况适当降低工作频率，MSP430 可以有三个时钟源并产生更多的内部工作频率，内部各个模块根据实际需要工作在不同的频率，不用的时钟还可以关闭。

开发板上的主芯片 MSP430F6638 的工作模式有 6 种，包括活动模式（AM）和 5 种低功耗模式（LPM0—LPM5），图 3.3.4 表示了各个低功耗模式之间的关系。活动模式和低功耗模式的实现是由状态寄存器（SR）中的几个标志位（SCG1、SCG0，OSCOOFF、CPUOFF）共同来控制，其组合控制如下图 3.3.3 所示。在所有低功耗模式下 CPU 的 MCLK 都是禁止的，要在低功耗模式下转到活动模式必须由中断来唤醒，所以一般情况下进入低功耗模式前，应保持开启中断状态即确保 GIE 为置位状态。

SCG1 (1)	SCG0	OSCOFF (1)	CPUOFF (1)	Mode	CPU and Clocks Status (2)
0	0	0	0	Active	CPU, MCLK are active. ACLK is active. SMCLK optionally active (SMCLKOFF = 0). DCO is enabled if sources ACLK, MCLK, or SMCLK (SMCLKOFF = 0). DCO bias is enabled if DCO is enabled or DCO sources MCLK or SMCLK (SMCLKOFF = 0). FLL is enabled if DCO is enabled.
0	0	0	1	LPM0	CPU, MCLK are disabled. ACLK is active. SMCLK optionally active (SMCLKOFF = 0). DCO is enabled if sources ACLK or SMCLK (SMCLKOFF = 0). DCO bias is enabled if DCO is enabled or DCO sources MCLK or SMCLK (SMCLKOFF = 0). FLL is enabled if DCO is enabled.
0	1	0	1	LPM1	CPU, MCLK are disabled. ACLK is active. SMCLK optionally active (SMCLKOFF = 0). DCO is enabled if sources ACLK or SMCLK (SMCLKOFF = 0). DCO bias is enabled if DCO is enabled or DCO sources MCLK or SMCLK (SMCLKOFF = 0). FLL is disabled.
1	0	0	1	LPM2	CPU, MCLK are disabled. ACLK is active. SMCLK is disabled. DCO is enabled if sources ACLK. FLL is disabled.
1	1	0	1	LPM3	CPU, MCLK are disabled. ACLK is active. SMCLK is disabled. DCO is enabled if sources ACLK. FLL is disabled.
1	1	1	1	LPM4	CPU and all clocks are disabled.
1	1	1	1	LPM3.5(3)	When PMMREGOFF = 1, regulator is disabled. No memory retention. In this mode, RTC operation is possible when configured properly. See the *RTC* module for further details.
1	1	1	1	LPM4.5(3)	When PMMREGOFF = 1, regulator is disabled. No memory retention. In this mode, all clock sources are disabled; that is, no RTC operation is possible.

(1) This bit is automatically reset when exiting low power modes. Please refer to Section 1.4.1 for details.
(2) The low-power modes and, hence, the system clocks can be affected by the clock request system. See the UCS chapter for details.
(3) LPM3.5 and LPM4.5 modes are not available on all devices. See the device-specific data sheet for availability.

图3.3.3 低功耗模式控制寄存器（SR）组合控制操作模式

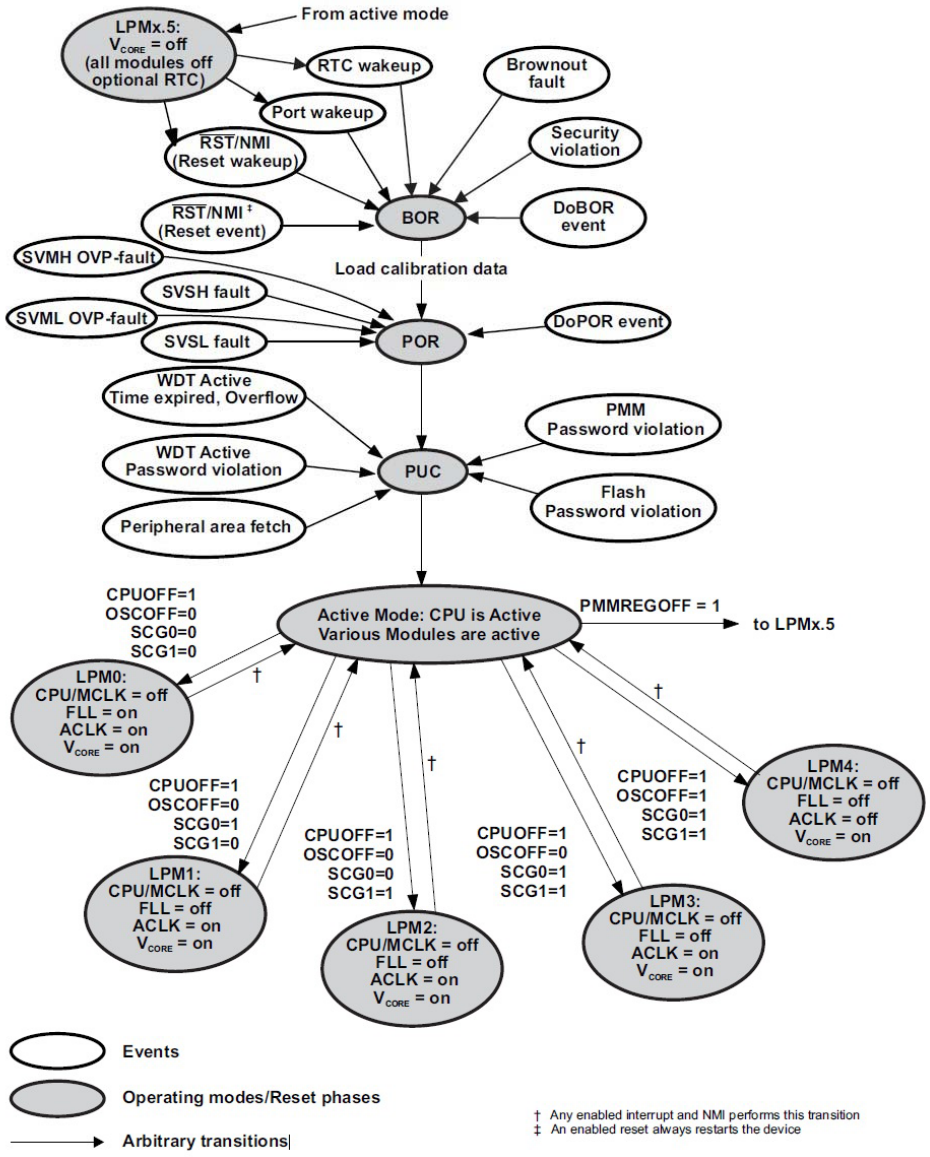

图3.3.4　不同功耗工作模式关系图

3.3.3 Lab3-1按键对LED灯的控制实验（中断方式）

3.3.3.1 实验介绍

本次实验应用了 MSP430F6638 中断事件中的 IO 口中断，通过使用开发板上的 S7 按键来触发 P4.0 口的中断事件，在中断事件处理函数中改变 LED5 灯的状态。

3.3.3.2 实验目的

- 了解 MSP430F6638 中断系统
- 掌握 IO 口中断的使用和编程

3.3.3.3 实验原理

本实验应用的是 MSP430F6638 的 IO 口中断，主板 S7 上按键连接到 P4.0 口，当 S7 按键被按下时，P4.0 口电平由高变成低触发一个中断事件，然后在 P4 口的中断函数中填写代码改变 LED5 灯的状态。按键与 LED 灯部分的电路原理图请看 Lab1-1 中实验原理章节，这里不再赘述。

3.3.3.4 程序分析

- **编程思路**

整个编程过程比较简单，只需要配置好相应的 IO 口就可以了，需要设置 P4.5 口为输出状态以控制 LED5 灯状态，还需要使能 P4.0 口的上拉电阻，选择 P4.0 口中断沿，使能中断，清中断标志位，然后进入等待状态，最后再写一个 P4 口的中断服务函数，在函数中改变 LED5 灯的点亮状态。当有按键被按下既产生了中断事件，程序转向中断服务函数并改变 LED5 灯的状态。用到的寄存器有 P4DIR、P4REN、P4OUT、P4IES、P4IFG、P4IE，具体功能可查看文档。写中断函数不同于写普通函数，中断函数不需要和普通函数那样声明，在 MSP430 中用拓展关键字“__interrupt”来表明，具体用法参考下面的代码。

- **程序流程图**

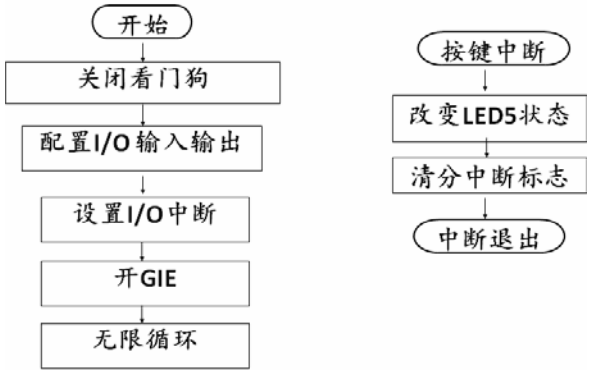

图 3.3.5　程序流程图

- 关键代码分析

main.c

```
#include <msp430f6638.h>
void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;              // 关闭看门狗
  P4DIR |= BIT5;                         // 设置 P4.5 口方向为输出
  P4DIR &= ~BIT0;                        // P4.0 置为输入
  P4REN |= BIT0;                         // 使能 P4.0 上拉电阻
  P4OUT |= BIT0;                         // P4.0 口置高电平
  P4IES |= BIT0;                         // 中断沿设置（下降沿触发）
  P4IFG &= ~BIT0;                        // 清 P4.0 中断标志
  P4IE |= BIT0;                          // 使能 P4.0 口中断

  __bis_SR_register(GIE);               // 开中断

While(1){ };                            //主循环
}

// P4 中断函数
#pragma vector=PORT4_VECTOR
__interrupt void Port_4(void)
{
  P4OUT ^= BIT5;                        // 改变 LED5 灯状态
  P4IFG &= ~BIT0;                       // 清 P4.0 中断标志位
}
```

3.3.3.5 实验步骤与现象

- 实验步骤

1. 将跳线帽接到 MSP430F6638 实验箱的 P20 引脚处 4.5 上，P19 引脚处 4.0 上；
2. 打开开发软件 CCS5，创建一个 MSP430F6638 的空工程；
3. 完成以上代码的编写；
4. 将代码编译下载到开发板中并全速运行，按下开发饭上的 S7 按键，观察 LED5 灯亮灭状态；

- 实验现象

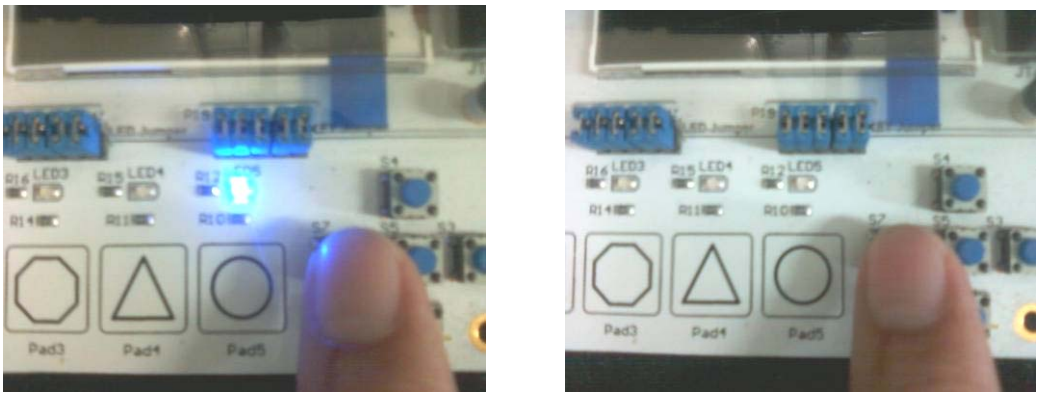

图 3.3.6　中断实验现象

3.3.3.6 实验思考

1. 在实验现象中可能会出现按一次按键却会出现 LED5 灯闪一次或者是多次的情况，这是为什么？如何解决？
2. 主函数中，没有调用中断子程序，中断子程序为什么能被执行？何时被执行?

3.3.3 Lab3-2 低功耗工作模式实验

3.3.3.1 实验介绍

TI 的 MSP430 是特别强调低功耗的单片机系列，具有多种低功耗模式，尤其是适用于采用电池供电的长时间场合，实验演示了 MSP430F6638 低功耗的编程方法，用过软件设置使 MSP430F6638 进入低功耗模式 1。

3.3.3.2 实验目的

1. 熟悉 MSP430F6638 实验板
2. 学习使用 TI 官方提供的函数库
3. 了解 MSP430F6638 的几种低功耗模式，掌握 MSP430F6638 低功耗设置的编程方法

3.3.3.3 实验原理

请参考 3.3.2 低功耗工作模式

3.3.3.4 程序分析

- **编程思路**

要使 MSP430F6638 进入低功耗模式主要就是对其控制寄存器（SR）的配置了，可直接调用库函数“__bis_SR_register(LPMx_bits)”使系统进入相应的低功耗模式（x 为模式编号），但是在进入低功耗之前要进行一些其他的配置，如关闭振荡器、配置驱动电平模式、配置未使用的 IO 口、设置禁止访问 USB 配置寄存器、禁用高低压侧 SVS 等，最后调用库函数进入低功耗模式 1。需要进行设置的寄存器包括 P7SEL、UCSCTL6、UCSCTL7、SFRIFG1、PxOUT、PxDIR、USBKEYPID、USBPWRCTL、PMMCTL0_H、SVSMHCTL、SVSMLCTL，具体寄存器功能可查阅文档。

- **程序流程图**

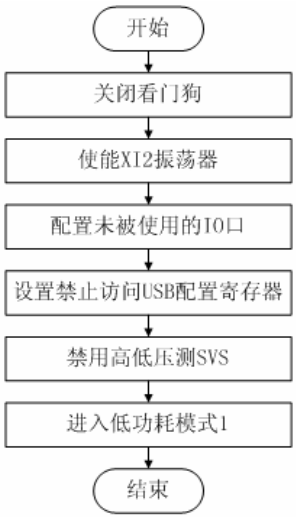

图3.3.7　程序流程图

- 关键代码分析

main.c

```
#include<msp430f6638.h>
voidmain(void)
{
  WDTCTL = WDTPW + WDTHOLD;                    //关闭看门狗
  P7SEL |= BIT2+BIT3;                          //端口XT2的选择
  UCSCTL6 &= ~(XT2OFF);                        //打开XT2振荡器
  UCSCTL6 |= XCAP_3;
do                                             //循环等待XT1和DOC振荡器稳定
  {
    UCSCTL7 &= ~(XT2OFFG + XT1LFOFFG + DCOFFG);
                                               // 清除XT1、XT2、DOC故障标志位
    SFRIFG1 &= ~OFIFG;                         // 清除OSC故障标志位
  }while (SFRIFG1&OFIFG);                      // 等待振荡器稳定
  UCSCTL6 &= ~(XT1DRIVE_3);                    // 减少驱动器
                                               //进一步降低低功耗而配置未使用的引脚
  P1OUT = 0x00;
  P2OUT = 0x00;
  P3OUT = 0x00;
  P4OUT = 0x00;
  P5OUT = 0x00;
  P6OUT = 0x00;
  P8OUT = 0x00;
  P1DIR = 0xFF;
  P2DIR = 0xFF;
  P3DIR = 0xFF;
  P4DIR = 0xFF;
  P5DIR = 0xFF;
  P6DIR = 0xFF;
  P8DIR = 0xFF;
  USBKEYPID = 0x9628;                  //置KEYandPID为0x9628（允许访问USB配置寄存器）
 USBPWRCTL &= ~(SLDOEN+VUSBEN);                          //禁用VUSB、LDO和SLDO
  USBKEYPID = 0x9600;                  //置KEYandPID为0x9600（禁止访问USB配置寄存器）
  //禁用SVS
  PMMCTL0_H = PMMPW_H;                       //PMM密码
  SVSMHCTL &= ~(SVMHE+SVSHE);                //禁用高压测SVS
  SVSMLCTL &= ~(SVMLE+SVSLE);                //禁用低压测SVS
```

```
    __bis_SR_register(LPM1_bits);                //进入低功耗模式1
    __no_operation();                            //延时一个机器周期
}
```

3.3.3.5 实验步骤与现象

- **实验步骤**

1. 打开开发软件 CCS5，创建一个 MSP430F6638 的空工程
2. 完成以上代码的编写
3. 将开发板与计算机连接，把程序编译并下载到开发板中，运行程序

- **实验现象**

无明显实验现象，请查看相应寄存器，并阅读实验思考。

3.3.3.6 实验思考

1. 低功耗模式 LPM1 的 ACLK、SMCLK 分别处于什么状态？MSP430F6638 微控制器的 ACLK、SMCLK 可以通过相应的引脚 P1.0、P3.4 输出，利用示波器探测观察，方法是将相应的引脚的 SEL 功能选择寄存器位设置为 1，并把 DIR 方向寄存器位设置为输出。请编程实现完成，并用示波器观察。
2. 给出的例程可否通过按键唤醒低功耗模式？为什么？如果想通过实验板上的按键 S7 来唤醒低功耗模式，如何修改程序？
3. 请将该实验改写为一个有明显现象的实验

3.4 定时器

定时器是 MCU 中重要的功能单元之一，有着多种用法，能够实现多种功能。下面主要介绍 MSP430F6638 中的定时器 A（TIMER_A）、看门狗定时器（WDT）、实时时钟（RTC_B）。

定时器是根据时钟脉冲累积计时的，根据时钟源和分频系数来获取不同的时钟脉冲（定时器的工作过程实际上是对时钟脉冲计数）。每个定时器均有一个设定值寄存器和一个当前值寄存器。设定值寄存器存储编程时赋值的计时时间设定值，当前值寄存器记录计时当前值，这两个寄存器可以是 8 位、16 位或者 32 位二进制存储器。当前值寄存器的最大值乘以定时器的计时单位值即是定时器的最大计时范围值。定时器满足计时条件开始计时，当前值寄存器则开始计数，当当前值与设定值相等时定时器动作，产生相应的响应信号。

MSP430 单片机的定时器相当丰富。有定时器 A（TA）、定时器 B(TB)、实时时钟 RTC、看门狗定时器 WDT 等。其中看门狗主要用于出现软件故障后自动复位系统；TA 提供多路捕捉/比较模块，除了基本的定时功能，还可以用于 PWM 和脉冲测量；TB 与 TA 基本相同，具有提供计数器长度可编程、双缓冲式设定值寄存器、输出可设定为高阻态等高级特性，适用于一些有特殊要求的 PWM 操作；RTC 主要用来计时，可以提供日历模式的输出（可从分离的寄存器中直接读到二进制或 BCD 编码的年月日时分秒），也可用来计数。

3.4.1 Lab4-1 TIMER_A实验

3.4.1.1 实验介绍

本实验使用了 MSP430F6638 中的 Timer_A 定时器模块，运用定时中断功能实现对 LED 指示灯的定时亮灭控制，并且控制蜂鸣器的发声频率。

3.4.1.2 实验目的

- 学习 Timer_A 定时器原理
- 熟练掌握 Timer_A 的定时中断
- 掌握蜂鸣器的使用方法
- 熟练应用 GPIO 控制 LED 亮灭

3.4.1.3 实验原理

◆ **TIMER_A 介绍**

定时器 A 是一个十六位的定时/计数器，MSP430F6638 中包含有 3 个定时器 A 的子模块：TA0、TA1、TA2，它们分别具有 5 个、3 个、3 个捕捉/比较模块。定时器 A 支持多重捕获/比较，PWM 输出和内部定时。定时器还有扩展中断功能，中断可以由定时器溢出产生或由捕获/比较寄存器产生。定时器 A 的特性包括：

- 四种运行模式的异步 16 位定时/计数器
- 可选择配置的时钟源
- 可配置的 PWM 输出
- 异步输入和输出锁存
- 对所有 TA 中断快速响应的中断向量寄存器

Timer_A 主要有 TAxCTL，TAxR，TAxCTLn，TAxCCRn，TAxIV、TAxEX0 几个寄存器（其中 x 指定时器的实例号，对 MSP430F6638 可以是 0~2，即 TA0~TA2；n 指捕捉/比较模块号）。其中最主要的是 TAxCTL 寄存器，用于设定 Timer_A 的输入时钟信号源、工作模式，复位、中断使能等。定时器 A 大致可分计数器和若干个捕捉比较模块。计数器在工作时由选定的时钟信号驱动计数，并且在计数归零时触发对应的定时器的 TAIFG；计数器配合几个比较/捕获寄存器便可实现定时、PWM、脉冲测量等功能。

TIMER_A 有 3 种定时/计数方式：

- 增计数模式：计数周期 TAxR 从 0 增加到 TAxCCR0

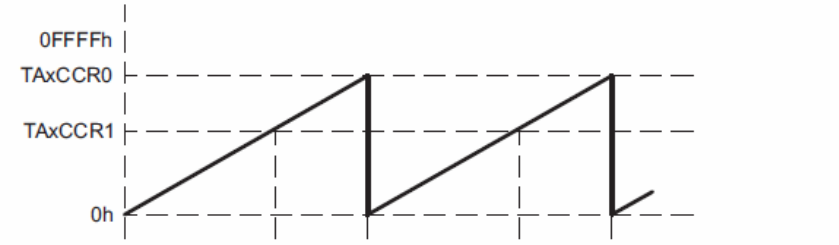

图3.4.1　增计数模式

- 连续计数模式：计数周期TAxR从0增加到0xffff

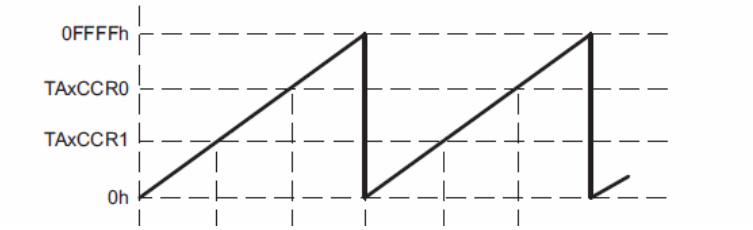

图3.4.2　连续计数模式

- 增减计数模式：TAxR从0增加到TAxCCR0然后再从TAxCCR0减到0

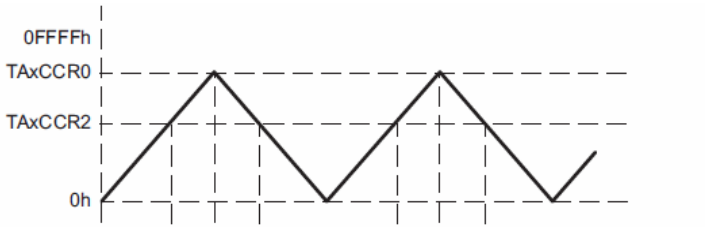

图3.4.3　增减计数模式

16 位定时/计数器寄存器 TAxR，随着分频后的时钟信号的上升到来沿增加/减少（由模式所决定）。TAxR 可以由软件读写。除此之外，定时器溢出时可以产生中断。TAxR 可以通过设置 TACLR 位来归零；在 UP/DOWN 模式下，设置 TACLR 也可以清除时钟分频器和计数方向。需要注意的是，操作定时器的相关寄存器前应当先停止定时器（中断使能、中断标志、TACLR 例外），以避免产生错误的运行结果。另外若定时器时钟与 CPU 时钟不同步，则在定时器运行中读取 TAxR 将产生不可预知的结果。一种可行的解决方案是在定时器运行时多读几次，通过软件表决的方式来确定正确的读数。对 TAxR 的写操作将会立即生效。

定时器的时钟源可以是内部时钟源 ACLK，SMCLK 或者或者外部源 TACLK。时钟源由 TAxCTL 寄存器的 TASSEL 域来选择，所选的时钟可以通过 ID 域选择进行 1，2，4，8 分频，之后时钟可以使用 TAxEX0 寄存器的 IDEX 域控制进一步进行 1，2，3，4，5，6，7，8 分频。

设置 TACLR 位将清除 TAxR 和分频器的状态。当 TACLR 位被清除时，TAxR 和分频器中的计数都将重新从 0 开始。

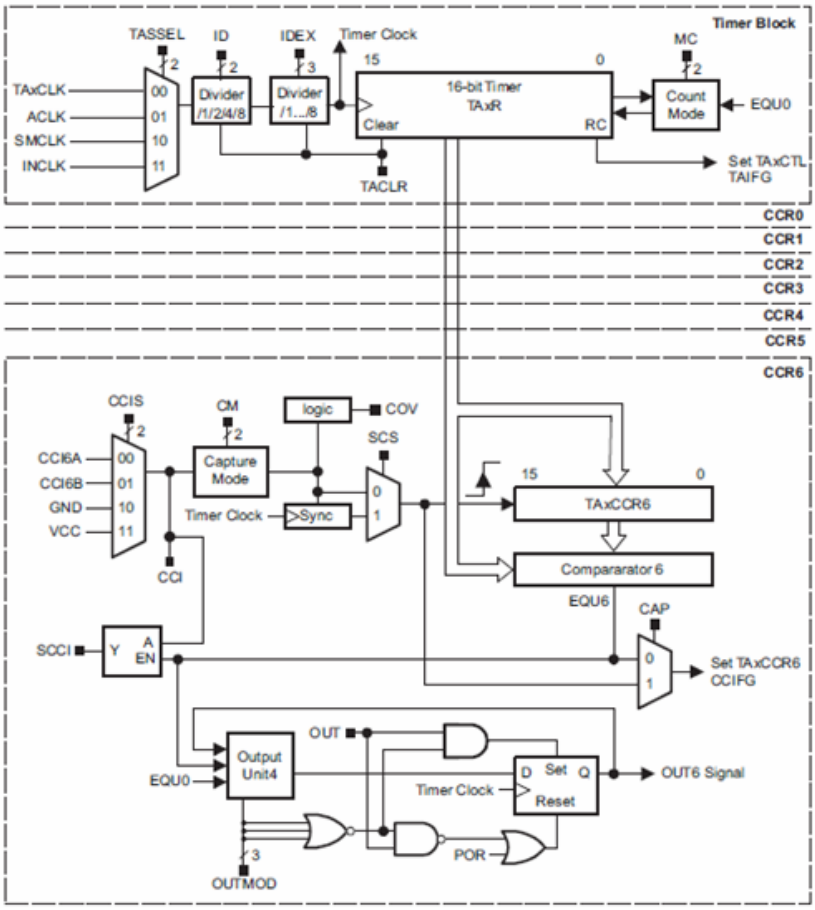

图3.4.4　TIMER_A 原理框图

◆ 蜂鸣器介绍

蜂鸣器按其是否带有信号源又分为有源和无源两种类型。有源蜂鸣器只需要在其供电端加上额定直流电压，其内部的震荡器就可以产生固定频率的信号，驱动蜂鸣器发出声音。无源蜂鸣器可以理解成与喇叭一样，需要在其供电端上加上高低不断变化的电信号才可以驱动发出声音。用定时器产生时序高低电平的信号送入 P1.5 引脚便实现了对蜂鸣器的控制。

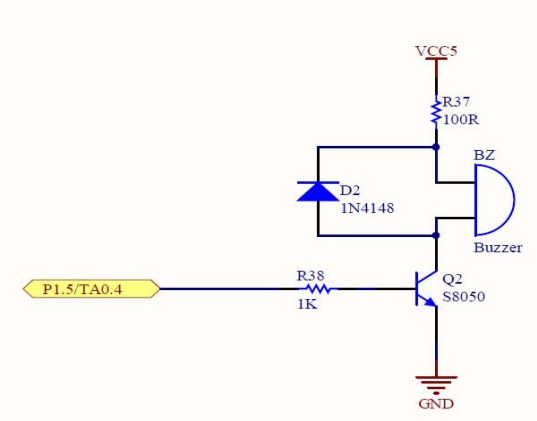

图3.4.5　蜂鸣器模块电路原理图

3.4.1.4 程序分析

- **编程思路**

实验开始必须先关闭看门狗定时器，然后配置好连接蜂鸣器和 LED 灯的 GPIO 寄存器，重点设置 TA0CTL 寄存器，使 Timer_A 工作在增计数模式，并选择 SMCLK 为其时钟源。使能比较器中断后，设定比较值为 50000，最后进入低功耗并打开总中断。

- **程序流程图**

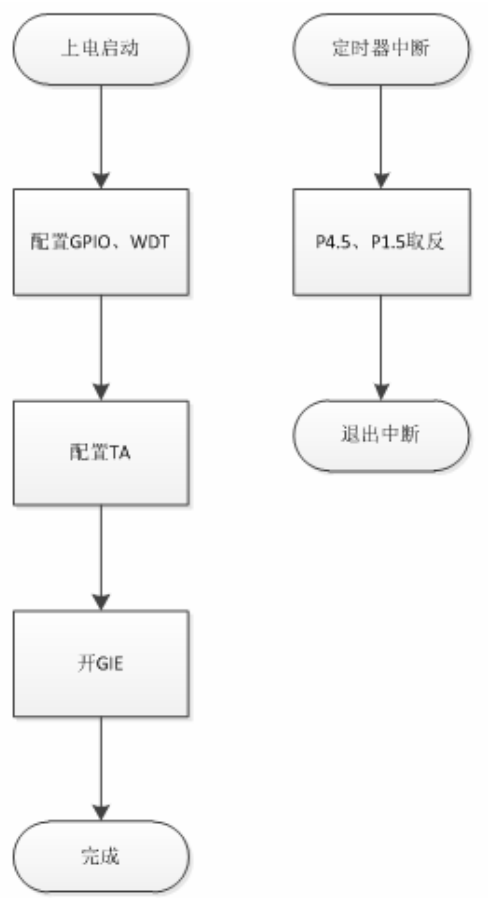

图3.4.6 程序流程图

- **关键代码分析**

主函数

```
void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;                  //关闭看门狗
    P1DIR |= BIT5;                             //控制蜂鸣器输出
    P4DIR |= BIT5;                             //控制 LED 输出
    TA0CTL |= MC_1 + TASSEL_2 + TACLR;
                                               //时钟为 SMCLK,比较模式，开始时清零计数器
    TA0CCTL0 = CCIE;                           //比较器中断使能
    TA0CCR0   = 50000;                         //比较值设为 50000，相当于 50ms 的时间间隔
    __bis_SR_register(LPM0_bits + GIE);   //进入低功耗并开启总中断
}
```

Timer_A 定时器中断服务函数

```
#pragma vector = TIMER0_A0_VECTOR
__interrupt void Timer_A (void)
{
    P1OUT ^= BIT5;                  //形成鸣叫效果
    P4OUT ^= BIT5;                  //形成闪灯效果
}
```

3.4.1.5 实验步骤与现象

- **实验步骤**

1、 实现控制代码的设计，画出流程图；
2、 完成编码，并将代码烧录到开发板中；
3、 观察 LED 的状态变化，验证结果；
4、 听蜂鸣器的声音频率，验证效果。

- **实验现象**

主板右下角 LED5 指示灯每隔50ms闪烁一次，同时蜂鸣器发出响声。

3.4.1.6 实验思考

1、 Timer_A 定时器都有哪些工作模式，如何配置？
2、 如何通过定时器设定蜂鸣器发声的音调？

3.4.2 Lab4-2 看门狗定时器（WDT）实验

3.4.2.1 实验介绍

本实验使用了 MSP430F6638 中的看门狗定时器模块，运用看门狗定时中断功能实现了对 LED 的定时亮灭控制。

3.4.2.2 实验目的

- 学习看门狗定时器原理。
- 熟练掌握看门狗定时中断。
- 熟练应用 GPIO 控制 LED 亮灭。

3.4.2.3 实验原理

看门狗定时器（WDT）实际上是一个计数器，初始计数值为 0，处理器上电复位(PUC)后看门狗开始计数。如果程序运行正常，过一段时间 CPU 应发出指令让看门狗复位，重新开始计数；如果看门狗溢出就认为程序没有正常工作，强制整个系统复位。

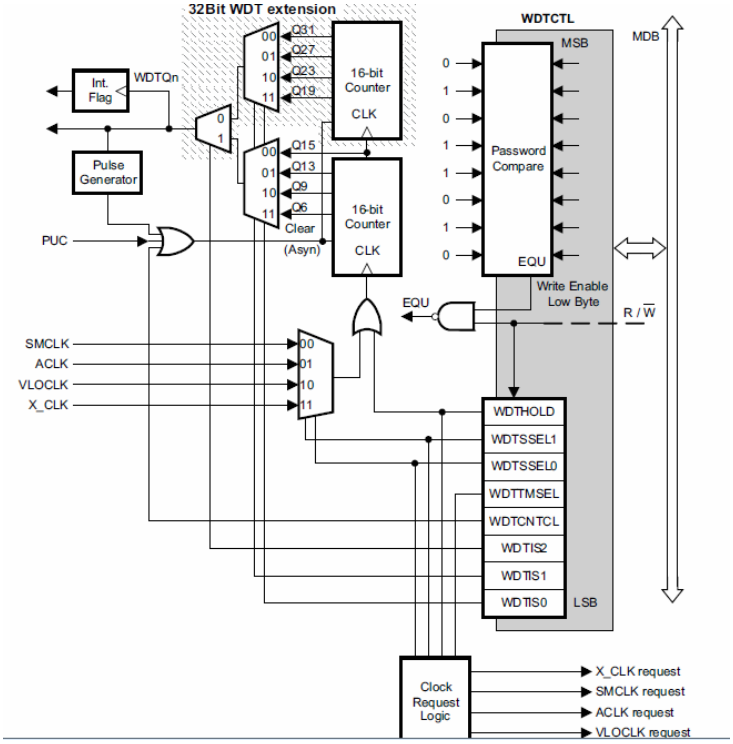

图3.4.7 WDT 定时器结构框图

WDT 由控制寄存器 WDTCTL 控制其计数器不可由程序访问。控制寄存器 WDTCTL 为 16 位寄存器，其高八位用作口令，低八位用作对 WDT 的实际控制，下面为控制寄存器各域的说明：

- WDTPW： 需要注意每次操作 WDTCTL 时必须写入本段，即对 WDTCTL 的操作均应当为 WDTCTL= WDTPW+....这种格式，否则将触发器件复位。
- WDTHOLD： 看门狗使能控制，向该位写入 1 时关闭看门狗。
- WDTTMSEL：模式选择，‘0’ 看门狗模式，‘1’ 定时器模式
- WDTCNCTL：计数器清除控制,当被置’1’后会自动复位为’0’，’0’ 无动作，’1’计数器清零
- WDTSSEL： 选择 WDTCN 的时钟源，’0’ 选择 SMCLK，’1’选择 ACLK。
- WDTISx： 看门狗的定时间隙选择。

Watchdog Timer Control Register (WDTCTL)

15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0
Read as 069h WDTPW, Must be written as 05Ah							
7	6	5	4	3	2	1	0
WDTHOLD	WDTSSEL		WDTTMSEL	WDTCNTCL	WDTIS		
rw-0	rw-0	rw-0	rw-0	r0(w)	rw-1	rw-0	rw-0

WDTPW	Bits 15-8	Watchdog timer password. Always read as 069h. Must be written as 05Ah, or a PUC is generated.
WDTHOLD	Bit 7	Watchdog timer hold. This bit stops the watchdog timer. Setting WDTHOLD = 1 when the WDT is not in use conserves power.
	0	Watchdog timer is not stopped.
	1	Watchdog timer is stopped.
WDTSSEL	Bits 6-5	Watchdog timer clock source select
	00	SMCLK
	01	ACLK
	10	VLOCLK
	11	X_CLK; VLOCLK in devices that do not support X_CLK
WDTTMSEL	Bit 4	Watchdog timer mode select
	0	Watchdog mode
	1	Interval timer mode
WDTCNTCL	Bit 3	Watchdog timer counter clear. Setting WDTCNTCL = 1 clears the count value to 0000h. WDTCNTCL is automatically reset.
	0	No action
	1	WDTCNT = 0000h
WDTIS	Bits 2-0	Watchdog timer interval select. These bits select the watchdog timer interval to set the WDTIFG flag and/or generate a PUC.
	000	Watchdog clock source /2G (18:12:16 at 32 kHz)
	001	Watchdog clock source /128M (01:08:16 at 32 kHz
	010	Watchdog clock source /8192k (00:04:16 at 32 kHz)
	011	Watchdog clock source /512k (00:00:16 at 32 kHz)
	100	Watchdog clock source /32k (1 s at 32 kHz)
	101	Watchdog clock source /8192 (250 ms at 32 kHz)
	110	Watchdog clock source /512 (15,6 ms at 32 kHz)
	111	Watchdog clock source /64 (1.95 ms at 32 kHz)

图3.4.8 WDT 定时器寄存器

3.4.2.4 程序分析

- **编程思路**

本实验的关键在于 WDT 的定时功能应用，首先得配置好看门狗的工作模式和时钟源，然后通过中断的方式实现定时地对 LED 的亮灭进行控制。

- **程序流程图**

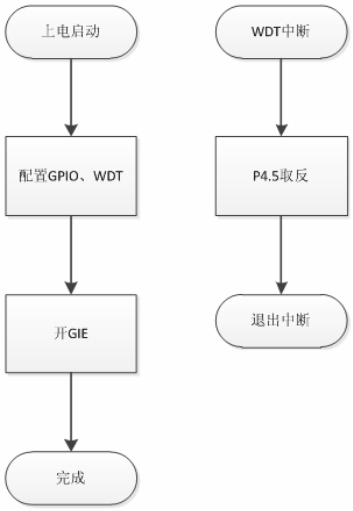

图3.4.9　程序流程图

- 关键代码分析

WDT 实验代码

主函数

```
void main(void)
{
    WDTCTL= WDT_ADLY_1000;                  // 时钟为 ACLK，模式为内部时钟
    SFRIE1 |= WDTIE;                        // 开启看门狗中断
    P4DIR |= BIT5;
    __bis_SR_register(LPM0_bits + GIE);     // 进入低功耗，开启总中断
}
```

看门狗定时器中断服务函数

```
#pragma vector=WDT_VECTOR
__interrupt void WDT_ISR(void)
{
    P4OUT ^= BIT5;                          // P4.5 口取反
}
```

3.4.2.5 实验步骤与现象

- 实验步骤

1. 完成代码，并将代码烧录到开发板中；
2. 观察 LED5 的状态变化，验证结果。

- 实验现象

主板右下角 LED5 指示灯以 1 秒位间隔，反复熄灭点亮

3.4.2.6 实验思考

1. 如何灵活运用 WDT 实现负责任务的调度与切换？
2. 能否自定义 WDT 的中断时间间隔？

3.4.3 Lab4-3 实时时钟（RTC）实验

3.4.3.1 实验介绍

该实验使用了 MSP430F6638 的 RTC_B 模块和外围的按键、段式液晶设备。RTC_B 用来计时，按键来控制计时的开始、暂停、清零；液晶模块用来显示计时的时间。

3.4.3.2 实验目的

- 熟练使用 RTC_B 的初始化和 RTC_B 中断
- 熟练应用按键中断和液晶显示

3.4.3.3 实验原理

◆ RTC 特点

实时时钟模块提供了具有日历模式、可编程闹钟和晶振频率校准功能的时钟。MSP430F6638 单片机有 RTC_A、RTC_B、RTC_C 三种实时时钟模块，RTC_A 模块可以配置为日历模式，也可以作为通用计数器使用；RTC_B 模块只能配置成日历模式，并且时钟源只能是 32768HZ 的外设时钟，能够在低功耗 LPMx.5 下工作；RTC_C 模块相比 RTC_B 模块额外提供了偏移校正和温度补偿，能在低功耗 LPM3.5 下工作。本实验用到的是 RTC_B 模块，它具有以下几个特点：

- 在日历模式中提供了秒钟、分钟、小时、星期、日期、月份和年份；
- 具有中断功能；
- 可以选择 BCD 码或者二进制格式；
- 具有可编程闹钟；
- 具有时间偏差的校正逻辑。

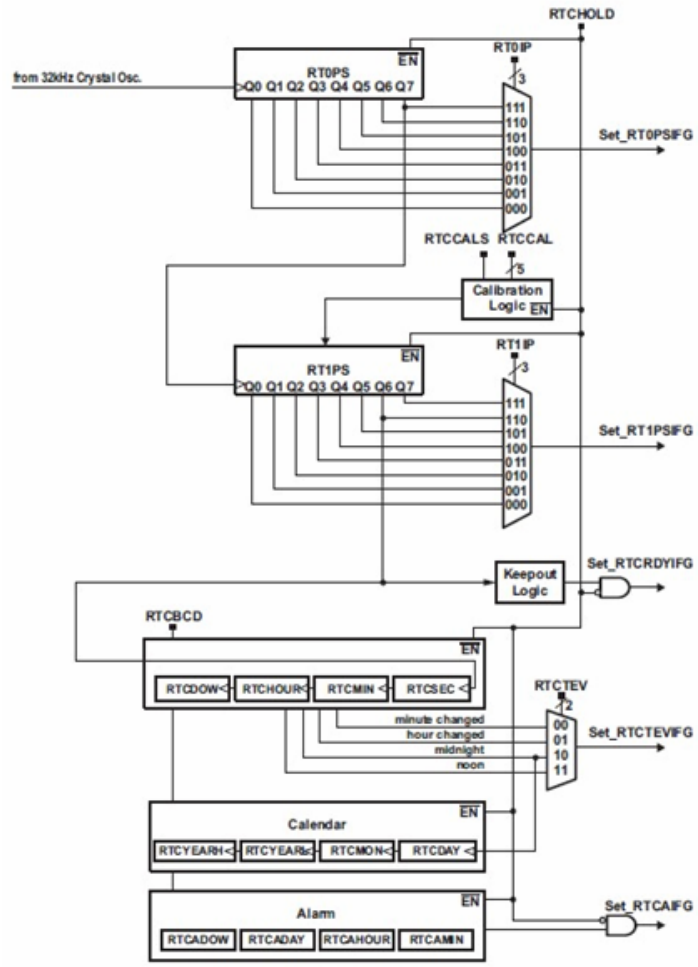

图3.4.13　RTC_B 结构框图

◆ **RTC_B 的初始化**

实时时钟模块的大多数寄存器没有初始化。在使用这个模块之前，用户必须通过软件进行配置。通过配置 RTCCTL01 寄存器的 HOLD 位来选择启动 RTC_B，配置 RTCBCD 位来选择数据格式。实时时钟将通过自动配置 RT0PS 和 RT1PS 为实时时钟提供一秒间隔的时钟。

◆ **RTC_B 的数据读取**

在日历模式下，实时时钟寄存器每秒更新一次。因为系统时钟实际上和实时时钟的时钟源不是同步的，为了防止在更新的时候读取实时数据而造成错误数据的读取，将会有一个禁止进入的窗口。RTCRDY 标明了可以进行读取的时段：仅有当 RTCRDY 被置位时可以安全的读取实时时钟寄存器。

◆ **RTC_B 的闹钟功能**

实时时钟模块提供了一个灵活的闹钟系统。通过设置闹钟的秒、分、小时、星期、月份、年份寄存器和对应的 AE 位，可以组合得到所需的闹钟功能。

实时时钟模块提供了 5 个中断源，每个中断源都有独立的使能位和标志位，分别是 RT0PSIFG、RT1PSIFG、RTCRDYIFG、RTCTEVIFG、RTCAIFG。这些中断标志共用同一个中断向量且具有优先次序，可由 RTCIV 判断当前请求中断的最高优先级的中断标志。

3.4.3.4 程序分析

- **编程思路**

首先初始化时钟、LCD，再初始化 RTC_B，将 RTC 的数据格式设为 BCD 格式、置高 HOLD 位，RTC 的小时、分钟、秒设为零，打开 RTCRDY 中断；连接按键的 IO 设置为输入上拉模式，使能 P4IO 中断，打开全局中断，然后进入低功耗 LPM3 模式；等待 RTC 中断和按键中断。在按键中断中，S7 键拉低 HOLD 位，RTC 开始计时，S5 键置高 HOLD 位 RTC 计时暂停，S3 键拉低 HOLD 位并清零 RTC 计时寄存器。在 RTCRDY 中断读取 RTC 的计时值，并显示在 LCD 屏上。

- **程序流程图**

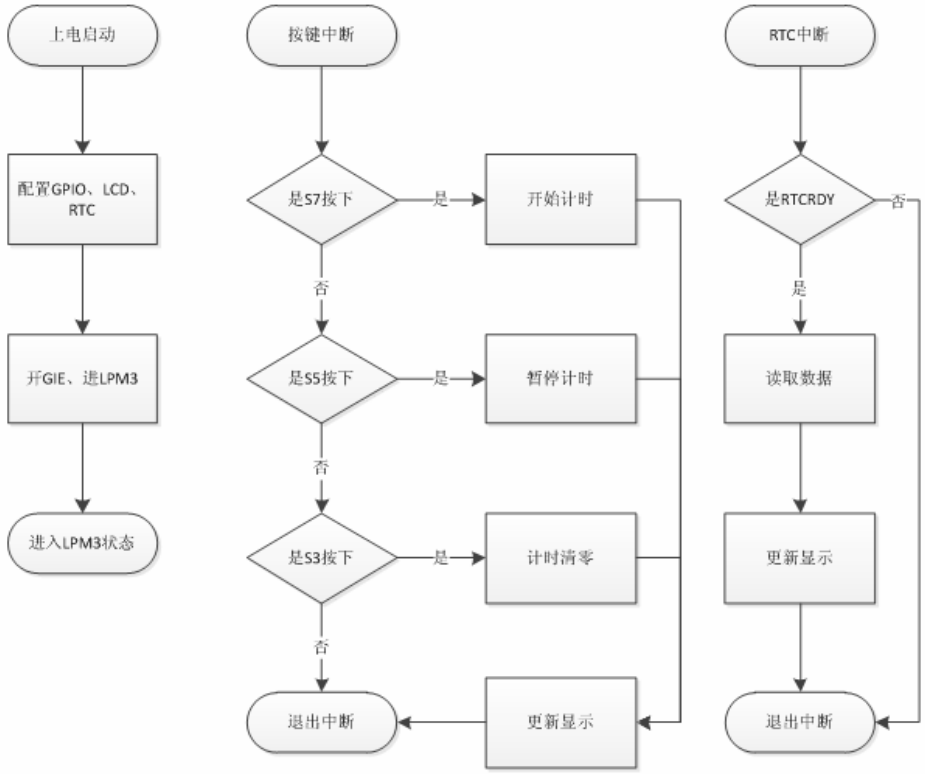

图3.4.14　程序流程图

- **关键代码分析**

实验 RTC 代码

实时时钟初始化

```
void SetupRTC(void)
{    RTCCTL01 |= RTCBCD + RTCHOLD;          //数据格式为 BCD 码
     RTCHOUR = 0x00;
     RTCMIN = 0x00;
     RTCSEC = 0x00;
     RTCCTL0 |= RTCRDYIE;                    // RTCRDY 中断使能
}
```

RTC 中断服务函数

```
#pragma vector=RTC_VECTOR
__interrupt void RTC_ISR(void)
{
     switch (__even_in_range(RTCIV, RTC_RT1PSIFG))
```

```
    {   case RTC_NONE: break;
        case RTC_RTCRDYIFG:
            hourBCD =   (RTCHOUR>>4)*10+(RTCHOUR&0x0f);        //获取小时数据
              minuteBCD = (RTCMIN>>4)*10+(RTCMIN&0x0f);        //获取分钟数据
              secondBCD = (RTCSEC>>4)*10+(RTCSEC&0x0f);        //获取秒数据
              show=hourBCD*10000+minuteBCD*100+secondBCD;      //变换为显示数据
              if(minuteBCD>0)
                  LCDSEG_DisplayNumber(show, 2);         //显示（有分钟时显示小数点）
              else
                  LCDSEG_DisplayNumber(show, 0);                 //显示（无小数点）
          break;
        case RTC_RTCTEVIFG:break;
        case RTC_RTCAIFG: break;
        case RTC_RT0PSIFG:break;                                 //分频器 0
        case RTC_RT1PSIFG: break;                                //分频器 1
        default: break;
    }
    __no_operation();
}
```

按键中断服务函数（略）

3.4.3.5 实验步骤与现象

- **实验步骤**

1. 将跳线帽接到 TFT 屏幕上方 P10 处引脚 S9--S11 上以及 P19 处 4.0--4.4 上；
2. 完成编码，并将代码烧录到开发板中；
3. 按下 S7 键，开始计时，然后按下 S5 键暂停计时，按下 S3 键计时清零。

- **实验现象**

a.计时器初始状态

b.不计数状态

c.暂停状态

d.清零状态

3.4.3.6 实验思考

1. RTC_A、RTC_B、RTC_C 三种实时时钟有何异同？
2. 如何使用 RTC_B 的闹钟功能？

3.4.4 Lab4-4 频率计实验

3.4.4.1 实验介绍

该实验算是对前几节所学知识的一个综合应用，实验中用到了 MSP430F6638 系统主板上多个外围设备，其中 555 定时器连接成的施密特触发器来对输入波形整形，GPIO 中断用来计数，按键输入来选择需要的闸门，TIMER_A 用来给闸门定时，段式液晶显示当前闸门信息和频率计数结果。

3.4.4.2 实验目的

- 学习使用施密特触发器
- 熟练应用 GPIO 中断
- 熟练应用 TimerA 定时器的计数模式，掌握定时器定时原理
- 熟练应用段式液晶显示数字
- 学习综合使用多个外围设备实现需要的功能

3.4.4.3 实验原理

◆ **施密特触发器**

施密特触发器也有两个稳定状态，但与一般触发器不同的是，施密特触发器采用电位触发方式，其状态由输入信号电位维持；对于负向递减和正向递增两种不同变化方向的输入信号，施密特触发器有不同的阈值电压。因此利用施密特触发器能将边沿变化缓慢的信号波形整形为边沿陡峭的矩形波。主板通过一个 555 定时器实现施密特触发器，其中正向阈值为 VT_+=2/3Vcc，反向阈值为 VT_-=1/3Vcc。

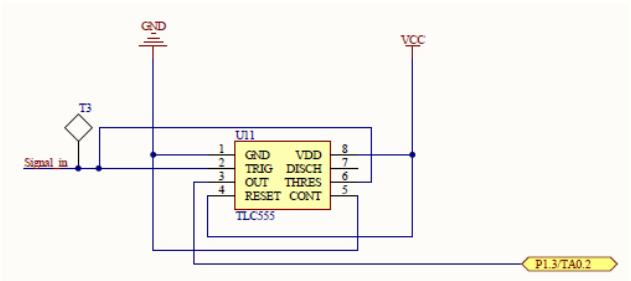

图3.4.15 555施密特触发器电路原理图

◆ **计数频率计**

本实验使用计数法计算输入波形频率，直接计数单位时间内被测信号的脉冲数，然后以数字形式显示频率值。该方法可以通过选择不同的闸门时间以满足不同的精度要求，常见的闸门有 0.01s、0.1s、1s、10s 等，得到的信号频率就是计数结果再乘以相应闸门的倍率。比如使用 10s 闸门完成计数后要对计数结果乘以 1/10。

◆ **按键输入**

本实验使用试验箱右下角 5 个按键，其电路图如下图所示。按键按下引脚接地，按键松开引脚悬空，所以需要配置引脚的内置上拉电阻，将引脚悬空时的状态上拉到高电平。

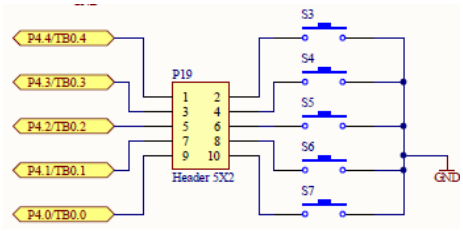

图3.4.16　按键模块电路原理图

3.4.4.4 程序分析

● **编程思路**

本实验的难点在于要设计多个定时时间以实现多个闸门选择。但是其定时时间差距较大，需要对子系统时钟信号进行多种分频。然而在设计 1s、10s 的闸门时对子系统时钟信号进行最大的 8 分频后，16 位的计数寄存器依然是不能满足定时需求，所以要在定时中断服务函数中使用一个辅助变量来拓展定时器可定时时间。

选择定时闸门时间的功能将通过一个定时器初始化函数来实现，采用不同的参数来区分不同的 Timer_A 控制寄存器的配置要求。

按键 S7 用来控制启动定时器，当定时结束时，定时器自动停止计数，并显示输入信号的计数结果。

● **程序流程图**

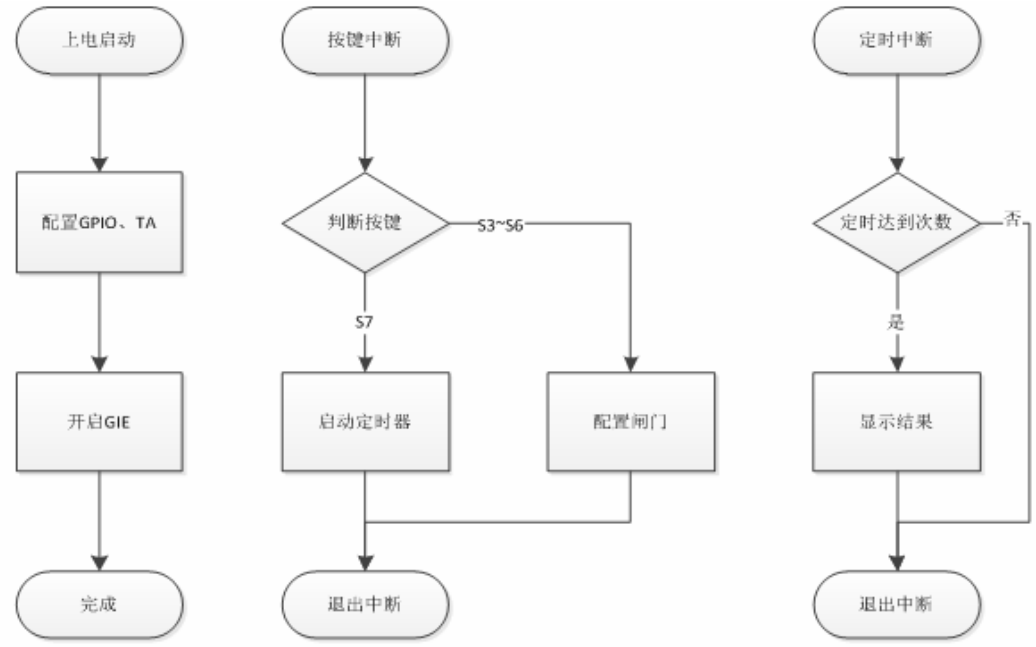

图3.4.17　程序流程图

- 关键代码分析

实验 频率计 代码

定时器初始化函数

```
void TimerA0_init(int ms)
{
    switch(ms)
    {
    case 10:                                        //10ms 闸门
        TA0CTL = TASSEL_2+MC_0+TACLR;
        TA0CCR0 = 10450;
        TA0CCTL0|=CCIE;
        gGate=10;
        break;
    case 100:                                       //100ms 闸门
        TA0CTL = TASSEL_2+MC_0+ID_1+TACLR;
        TA0CCR0 = 52250;
        TA0CCTL0|=CCIE;
        gGate=100;
        break;
    case 1000:                                      //1sec 闸门（略）
    ……
    case 10000:                                     //10sec 闸门（略）
    ……..
    default: break;                                 //暂不支持其他闸门初始化
    }
    ShowCymometerGate(gGate);
}
```

定时器中断服务函数

```
#pragma vector = TIMER0_A0_VECTOR
__interrupt void TimerA0_ISR(void)
{
    if(gTimer_count!=0)
    {
        gTimer_count--;
    }
    if(gTimer_count==0)
    {
        TA0CTL&=~MC_3;                              //计数器停止
        P1IE&=~BIT3;                                //关闭 P1 中断
        CymometerComplete(gFCount,gGate);
    }
}
```

按键中断服务函数

```
#pragma vector=PORT4_VECTOR
__interrupt void Port4Interrupt(void)
{                                                   //按键响应
    if(P4IFG&0X1F)
    {
        switch(P4IFG)
        {
        case SW3:TimerA0_init(10);          break;
        case SW4:TimerA0_init(100);         break;
        case SW5:TimerA0_init(1000);    break;
        case SW6:TimerA0_init(10000); break;
        case SW7:                                   //开始计频
                gFCount=0;
                TA0CTL|=TACLR;
```

```
                    TA0CTL |=MC_1;
                    P1IE=BIT3;
                default:break;                          //多个按键同时触发不做处理
                }
        }
        P4IFG=0;
}
```

3.4.4.5 实验步骤与现象

- **实验步骤**

1. 将跳线帽接到 TFT 屏幕上方 P10 处引脚 S9--S11 上以及 P19 处 4.0--4.4 上；
2. 完成编码，并将代码烧录到开发板中；
3. 调节波形发生器，使输出电压为 4.5V，将波形信号发生器 GND 接到开发板上 GND 端，将信号端接到开发板 Signal Input 端（T3）；
4. 选择合适的闸门（按键 S3-S6），按下 S7 开始计数，计数完毕后显示结果。

- **实验现象**
  - 选择闸门

10ms

100ms

1sec

10sec

图3.4.18 闸门选择

  - 显示频率计数结果

图 3.4.19 频率计结果显示（单位 Hz）

3.4.4.6 实验思考

1. 在测量频率时，闸门是越大越好吗？
2. 如果系统主频默认使用 1.05MHz，那么可以测得的频率最高是多少？

3.5 模拟电压比较器B模块

本节主要是介绍 MSP430F6638 中模拟电压比较器 B 模块的使用，这是一个重要的模块，主板上的电容触摸按键（Pad1~Pad5）的功能实现就是依靠电压比较器 B。

- 比较器 B 特点

比较器 B 模块可用于精确的斜率式模数转换，电压监控和外部模拟信号的监控。比较器 B 有如下特点：同相端和反相端均有输入复用器；比较器输出端具有软件选择的 RC 滤波器；比较器的输出可以作为定时器 A 的捕获输入；端口输入缓冲由软件控制；具有中断功能，并支持在低功耗模式响应；可选择的参考电压发生器,电压滞回发生器；可使用共享参考源作为参考电压；超低功耗比较模式。

- 比较器的结构

由下图可以看出比较器 B 模块大概可以分为 5 个部分：模拟输入部分、核心部分、低通滤波部分、基准电压部分和比较器输出部分。

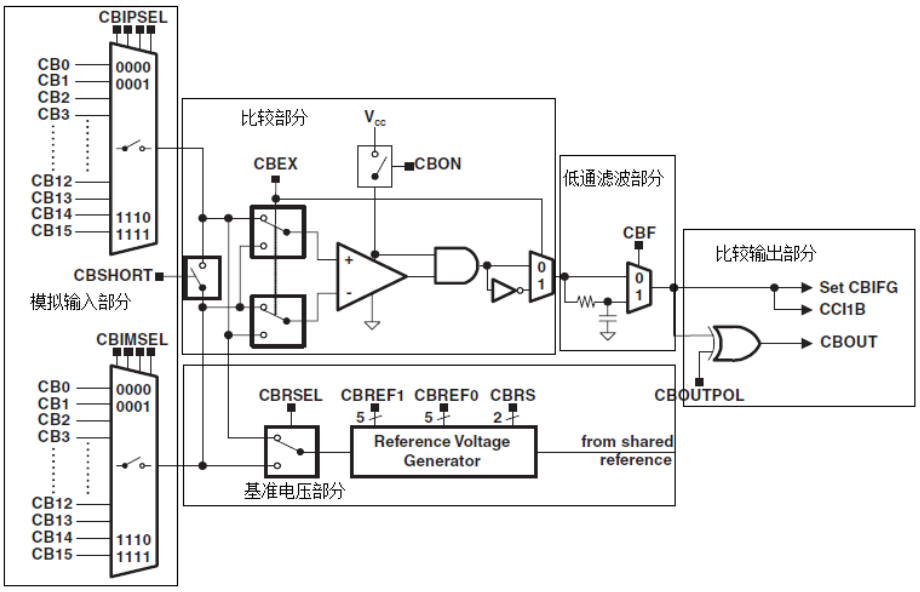

图 3.5.1 电压比较器 B 结构框图

- **模拟输入部分：**比较器 B 的正负输入端各自有 16 个通道选择，也都可以选择参考电压。CBCTL0 控制寄存器中的 CBIMEN 和 CBIPEN 位分别用于使能反相端和同相端的模拟信号输入，CBIMSEL、CBIPSEL 位分别用于选择反相端和同相端的输入通道。同时我们可以通过操作控制寄存器 CBCTL3 来禁止或者使能比较器相应的输入通道的输入缓存。

- **比较部分：**整个比较器 B 工作需将 CBCTL1 控制寄存器的 CBON 位置为 1，单片机上电复位时，此位为 0；CBCTL1 中的 CBEX 位控制输入方向。

- **基准电压部分：**主要由控制寄存器 CBCTL2 中的 CBRS、CBRSEL 、CBREF1、CBREF0 CBREFL 等位控制。CBRS 选择基准电压源，CBRSEL 位选择基准电压

输出到同相端或反相端，CBREF1、CBREF0 控制基准电压生成器的电阻网络以对输入电压进行分压。

- **低通滤波部分：**使用 CBCTL 控制寄存器中的 CBF 位来控制此滤波器的功能开与关。此滤波器功能是用于消除比较器输出信号的毛刺，以保证信号的质量和中断请求的可靠性。

比较器 B 的输出既可以输出到内部模块也可以输出到外部引脚，也可以用于产生中断。比较器 B 模块可以在比较器输出的上升沿或者下降沿产生中断，所用的边沿由 CBCTL1 的 CBIES 选择。比较器 B 模块具有独立的中断向量，中断被响应后硬件会自动清除中断标志位 CBIFG。

Register	Short Form	Register Type	Address Offset	Initial State
Comp_B control register 0	CBCTL0	Read/write	0x0000	Reset with PUC
Comp_B control register 1	CBCTL1	Read/write	0x0002	Reset with PUC
Comp_B control register 2	CBCTL2	Read/write	0x0004	Reset with PUC
Comp_B control register 3	CBCTL3	Read/write	0x0006	Reset with POR
Comp_B interrupt register	CBINT	Read/write	0x000C	Reset with PUC
Comp_B interrupt vector word	CBIV	Read	0x000E	Reset with PUC

图 3.5.2　电压比较器 B 相关寄存器

3.5.1 Lab5-1 电压比较器实验

3.5.1.1 实验介绍

该实验使用了 MSP430F6638 的比较器 B，一路使用 P6.0 管脚，一路使用内部的 1.5V 参考电压，将两路信号进行比较；当 P6.0 管脚大于 1.5V 时触发中断，点亮 LED5。

3.5.1.2 实验目的

- 熟悉比较器 B
- 学会使用低功耗工作模式

3.5.1.3 实验原理

比较器 B 的主要功能是指出两个输入电压 CPIP 和 CPIM 的大小关系，然后设置输出信号，比较器的两个输入电压各自有 16 个通道选择。在本次实验中 CPIP 选择的是 CB0 即 P6.0 管脚；CPIM 选择的是内部的 1.5V 参考电压；当 P6.0 管脚的电压大于 1.5V 时，触发中断，在中断里翻转触发的边缘，翻转 LED5 的状态。

3.5.1.4 程序分析

- **编程思路**

本次实验只使用比较器 B 和 LED 电路，首先将连接 LED5 的 P4.5 管脚设为输出模式，再初始化比较器 B，先选择 CNIP 的输入管脚为 CB0，关闭 P6.0 的输入缓冲。选择 CPIM 的管脚为内部参考电压 1（即 1.5V ）,使能比较器中断，打开比较器，进入低功耗，等待中断唤醒。

- **程序流程图**

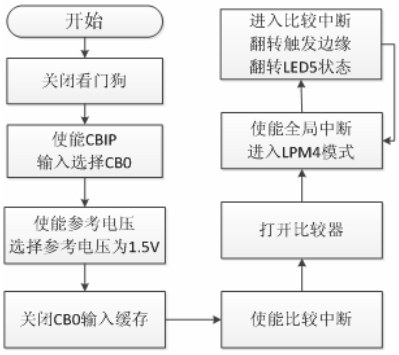

图 3.5.3 程序流程图

- 关键代码分析

实验 比较器 代码

主函数

```
void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // 关闭看门狗
  P4DIR |= BIT5;                            // P4.5/LED 设为输出
  CBCTL0 |= CBIPEN + CBIPSEL_0;             // 使能正端，输入通道选择 CB0
  CBCTL1 |= CBPWRMD_1;                      // 正常的电源供应模式
  CBCTL2 |= CBRSEL;                         // 使能内部参考电压选择
  CBCTL2 |= CBRS_3+CBREFL_1;                // 参考电压选择为 1.5V
  CBCTL3 |= BIT0;                           // 关闭 CB0 输入缓存
  __delay_cycles(75);
  CBINT &= ~(CBIFG + CBIIFG);               // 清除中断标志位
  CBINT  |= CBIE;                           // 使能比较中断，边缘选择为上升沿
  CBCTL1 |= CBON;                           // 打开比较器
  __bis_SR_register(LPM4_bits+GIE);         // 打开总中断，进入 LPM4 模式
  __no_operation();
}
```

比较器中断服务函数

```
#pragma vector=COMP_B_VECTOR
__interrupt void Comp_B_ISR (void)
{
  CBCTL1 ^= CBIES;                          // 翻转触发边缘
  CBINT &= ~CBIFG;                          // 清除中断标志
  P4OUT ^= BIT5;                            // 翻转LED5状态
}
```

3.5.1.5 实验步骤与现象

- 实验步骤

1. 完成编码，并将代码烧录到开发板中；
2. 将峰值是 2V、频率为 1Hz 的方波接入 P17 插槽中的 P6.0 管脚，将波形信号发生器 GND 接到开发板上 GND 端；
3. 观察 LED5 指示灯的状态。

- 实验现象

LED5 指示灯每一秒闪一次

3.5.1.6 实验思考

1. 为什么要关闭 P6.0 的缓存？
2. 比较器 A 和比较器 B 有什么区别和共同点？

3.5.2 Lab5-2 触摸按键实验

3.5.2.1 实验介绍

本实验使用 MSP430F6638 实验箱的触摸按键模块、LED 模块。利用 MSP430F6638 中的比较器 B 来判断触摸按键的充放电，通过 CPU 时钟算出电容的振荡频率，再判断是否有按键被触摸，如果有按键触摸则通过相应的 LED 显示当前的状态。

3.5.2.2 实验目的

- 学习电容触摸按键硬件电路原理；
- 学习触摸按键应用试验操作及编程思想；
- 学习定时器实现频率计的方法；
- 学习触摸电容程序资源。

3.5.2.3 实验原理

人体是具有一定电容的。当我们把 PCB 板上的覆铜画成如下形式的时候，就完成了一个最基本的触摸感应按键。

MSP430F6638 的比较器可以为比较结果 0 和 1 时设定两个阈值，这样便构成一个 RC 振荡器(电容电压低于较低阈值时充电、高于较高阈值时放电)，振荡频率与电容容值有关。当手指触摸到电容触摸按键以后，电容会由 C1 变化至 C2，振荡器的输出频率会发生变化，因此只需在固定时间内，计算出振荡器的输出频率。由于振荡频率较高(未触摸时在几百 KHz 量级)，可以直接用 CPU 时钟进行测量(即用变量自增来测量若干次振荡的耗时)。但为了防止测量中途出现中断影响测量，测量过程中应当关闭中断。如果在某一时刻输出频率有较大的变化的话，那就说明电容值已经被改变，即对应按键被按下了。

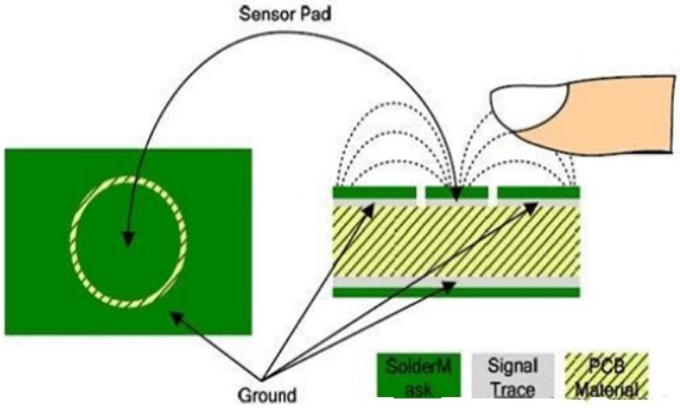

图 3.5.4 触摸按键工作原理图

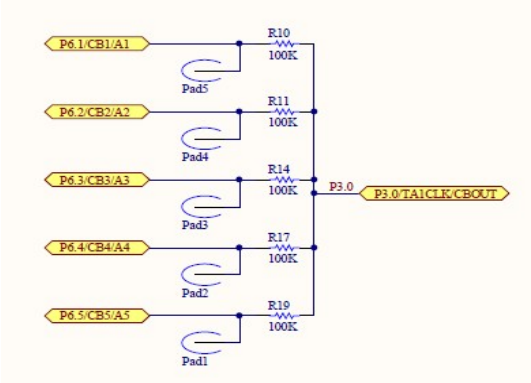

图 3.5.5　触摸按键模块电路原理图

3.5.2.4 程序分析

- **编程思路**

先初始化触摸按键和控制 LED 灯的 IO 口，然后反复的测量各个触摸按键的频率值，当它们的频率的值大于某个门限值时，操作对应的 LED 灯。在每次测量中要加入适当的延时，防止反复测量导致 LED 误操作。

- **程序流程图**

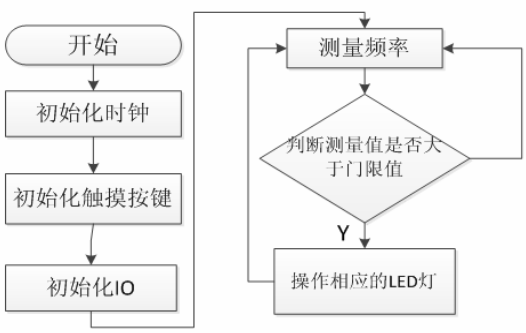

图 3.5.6　程序流程图

- **关键代码分析**

实验 触摸按键 代码

主函数

```
int main( void )
{
   int i;
   WDTCTL = WDTPW + WDTHOLD;
   _DINT();
   initClock();                                    //初始化时钟
   initCapTouch();                                 //初始化触摸按键
   for(i=0;i<5;++i)
*LED_GPIO[i]->PxDIR |= LED_PORT[i];
//设置各 LED 灯所在端口为输出方向
   _EINT();
   while(1)
```

```
    {
      uint32_t temp[5] ={0,0,0,0,0};
      for(i=5;i>=1;i--)
      {
        temp[i-1]= CapTouch_ReadChannel(i);                    //测量触摸按键的频率值
        if(temp[i-1]>captouch_max)
          *LED_GPIO[i-1]->PxOUT ^= LED_PORT[i-1]; //当测量值大于门限值时翻转
     }                                                          对应的 LED 指示灯
      __delay_cycles(8000000);
    }
    return 0;
}
```

比触摸按键测量函数

```
uint16_t CapTouch_ReadChannel(int i)
{
    uint16_t cpu_cnt = 0;
    uint16_t osc_cnt = 0;
    CBCTL0 = CBIMEN + (i << 8);                                //外部信号加到负极
    P6OUT &= ~ALL_PORT;
    P6DIR |= ALL_PORT & ~(1 << i);
    CBCTL3 = 1 << i;
    uint16_t gie;
    _ECRIT(gie);
    while(1)
    {
      if(CBCTL1 & CBOUT)                                       //控制电容的充放电
        P6OUT |= ALL_PORT;
      else
        P6OUT &= ~ALL_PORT;
      if(CBINT & CBIFG)
      {
        CBINT &= ~CBIFG;
        osc_cnt++;
      }
      if(osc_cnt >= OSC_CYCLES)
        break;
      cpu_cnt++;                                               //计算频率
    }
    _LCRIT(gie);
    CBCTL3 = ALL_PORT;
    P6DIR &= ~ALL_PORT;
    return cpu_cnt;
}
```

3.5.2.5 实验步骤与现象

触摸相应的按键 Pad1~Pad5（触摸时间不宜过长），可以观察到对应的 LED 指示灯的亮灭。

3.5.2.6 实验思考

请考虑用定时器来设计频率计，来计算触摸按键的电容。

3.6 电机模块实验

实验系统的电机模块包含直流电机与步进电机各一个，MSP430F6638 芯片 GPIO 口输出 PWM 波形对这两个电机的转速、转向、步长等进行控制。

PWM 是英文“Pulse Width Modulation”的缩写，即：脉冲宽度调制（简称脉宽调制），是利用微处理器的数字输出对模拟电路进行控制的一种非常有效的技术。

脉宽调制（PWM）基本原理：控制方式就是对电路开关器件的通断进行控制，使输出端得到一系列幅值相等的脉冲，使各脉冲的等值电压为所需要的波形。

例如下图显示了三种不同的 PWM 信号。下图 a 是一个占空比为 10%的 PWM 输出，即在信号周期中，10％的时间通，其余 90％的时间断。图 b 和图 c 显示的分别占空比为 50%和 90%的 PWM 输出。这三种 PWM 输出编码的分别是强度为满度值的 10%、50%和 90%的三种不同模拟信号值。例如，假设供电电源为 9V，占空比为 10%，则对应的是一个幅度为 0.9V 的模拟信号。如果调制频率不够高则不能得到相应的模拟信号，而是断裂的最大值与最小值。

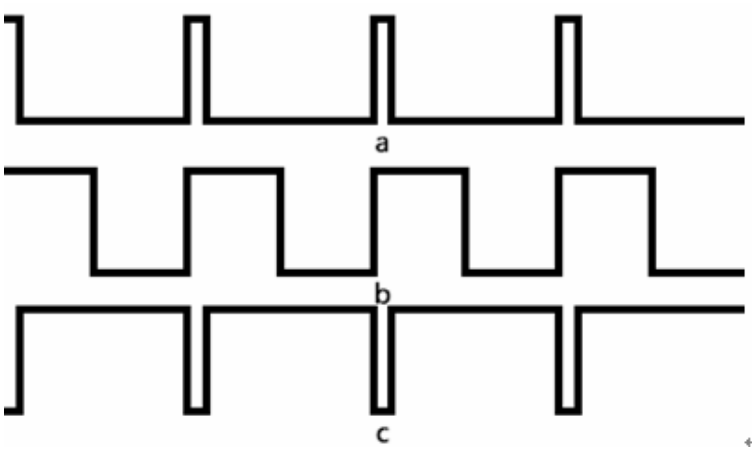

图3.6.1 PWM 波形

再例如，把正弦半波波形分成 N 等份，就可把正弦半波看成由 N 个彼此相连的脉冲所组成的波形。这些脉冲宽度相等，都等于 π/n，但幅值不等，即脉冲顶部不是水平直线，而是曲线，各脉冲的幅值按正弦规律变化。如果把上述脉冲序列用同样数量的等幅而不等宽的矩形脉冲序列代替，使矩形脉冲的中点和相应正弦等分的中点重合，且使矩形脉冲和相应正弦部分面积（即冲量）相等，就得到一组脉冲序列，这就是 PWM 波形。可以看出，各脉冲宽度是按正弦规律变化的。根据冲量相等效果相同的原理，PWM 波形和正弦半波是等效的。对于正弦的负半周，也可以用同样的方法得到 PWM 波形。

在 PWM 波形中，各脉冲的幅值是相等的，要改变等效输出波形的幅值时，只要按同一比例系数改变各脉冲的宽度即可。根据上述原理，在给出了所需产生的波形各时刻的瞬时值后，PWM 波形各脉冲的宽度和间隔就可以准确计算出来。按照计算结果控制电路中各开关器件的通断，就可以得到所需要的 PWM 波形。

3.6.1 Lab6-1 直流电机简单控制实验

3.6.1.1 实验介绍

该实验使用了电机子板上的直流电机。通过软件设置直流电机的转动方向或者停止电机转动。

3.6.1.2 实验目的

- 熟练使用 GPIO
- 学习直流电机控制

3.6.1.3 实验原理

- **直流电机介绍**

直流电机（Direct Current Motor）是指能将直流电能转换成机械能的旋转电机。它是能实现直流电能和机械能互相转换的电机。直流电机的结构由定子和转子两大部分组成，电机运行时静止不动的部分称为定子，定子的主要作用是产生磁场，由机座、主磁极、换向极、端盖、轴承和电刷装置等组成；运行时转动的部分称为转子，其主要作用是产生电磁转矩和感应电动势，是直流电机进行能量转换的枢纽，所以通常又称为电枢，由转轴、电枢铁心、电枢绕组、换向器等组成。

- **直流电机控制电路**

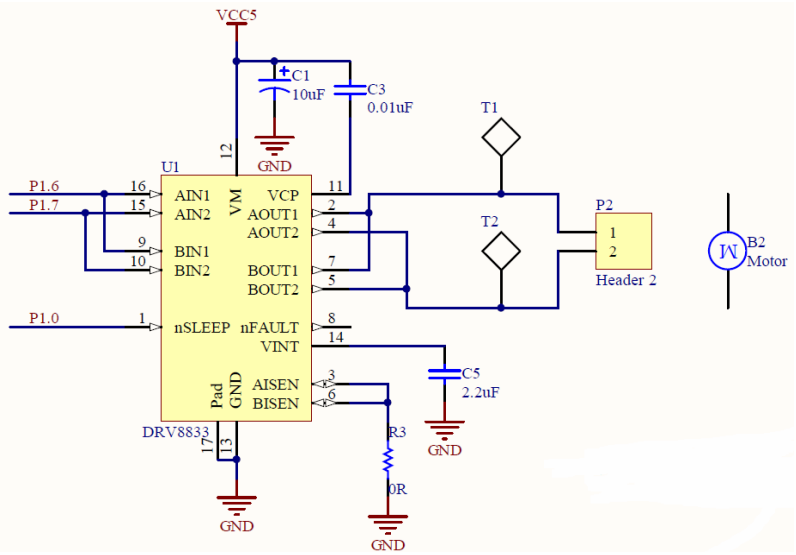

图3.6.2 直流电机模块电路原理图

这个控制电路主要使用了 TI DRV8833 低电压电机驱动芯片，该芯片为玩具、打印机及其他机电一体化应用提供了一款双通道桥式电机驱动器解决方案。它能驱动两个直流电机或一个步进电机，每个 H 桥能输出 1.5A 电流。

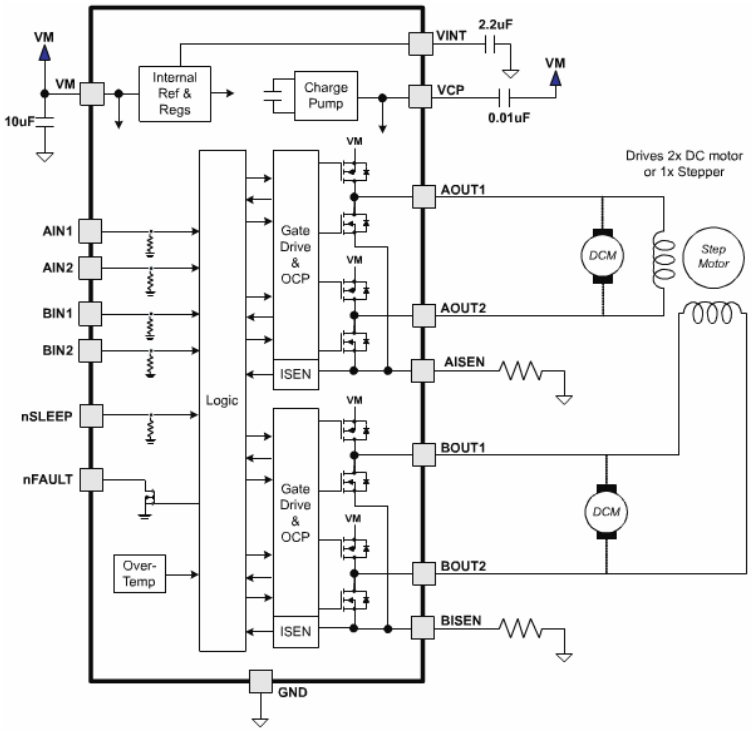

图3.6.3 TI DRV8833 芯片内部结构图

其中输入端控制 H 桥逻辑如下表所示

xIN1	xIN2	xOUT1	xOUT2	功能
0	0	Z	Z	惯性旋转
0	1	L	H	反转
1	0	H	L	正转
1	1	L	L	制动

表3.6.1 DRV8833 H桥逻辑表

3.6.1.4 程序分析

- **编程思路**

编程思路非常简单，在弄懂 DRV8833 控制逻辑以后，只需要简单的在对应引脚上输出想要的电平即可。

- **程序流程图**

图3.6.4　程序流程图

- 关键代码分析

实验 直流电机简单控制 代码

DC Motor 控制函数

```
void DCmotor(int p)
{
  switch(p)
  {
  case 0:                                  //停转
    {
      P1OUT &=~ BIT0;
      P1OUT &=~ BIT6;
      P1OUT &=~ BIT7;
      break;
    }
  case 1:                                  //正转
    {
      P1OUT |= BIT0;
      P1OUT |= BIT6;
      P1OUT &=~ BIT7;
      break;
    }
  case 2:                                  //反转
    {
      P1OUT |= BIT0;
      P1OUT &=~ BIT6;
      P1OUT |= BIT7;
      break;
    }
  }
}
```

3.6.1.5 实验步骤与现象

- **实验步骤**

1. 打开开发软件 CCS，新建工程，完成以上代码的编写；
2. 用 USB 数据线把开发板左端的 USB 口 J1 与电脑连接，点击开发软件上的 Debug 按钮将程序下载到开发板中；
3. 用 20-pin 连接线将电机板连接到主板 P15 或 P17 端口（注意连接线方向：连接线红线对准端口上“1”管脚）；
4. 点击开发软件上的 Resume 按钮全速运行程序，观察电机转动情况。

- **实验现象**

直流电机按照程序设定转动。

1. 实验思考

1. 可否直接使用 GPIO 控制电机，而不用 DRV8833？
2. 如何调整电机转速？

3.6.2 Lab6-2 步进电机简单控制实验

3.6.2.1 实验介绍

该实验使用了 MSP430F6638 系统电机子板上的步进电机。通过软件设置步进电机的转动方向或转动角度。

3.6.2.2 实验目的

- 熟练使用 GPIO
- 学习步进电机控制

3.6.2.3 实验原理

- **步进电机介绍**

步进电机是一种感应电机，它的工作原理是利用电子电路，将直流电变成分时输出的多相时序控制电流，用这种电流为步进电机供电，步进电机才能正常工作。

虽然步进电机已被广泛地应用，但步进电机并不能像普通的直流电机、交流电机在常规条件下使用。它必须由双环形脉冲信号、功率驱动电路等组成控制系统方可使用。因此用好步进电机却非易事，它涉及到机械、电机、电子及计算机等许多专业知识。

步进电机是一种将电脉冲转化为角位移的执行机构。通俗一点讲：当步进驱动器接收到一个脉冲信号，它就驱动步进电机按设定的方向转动一个固定的角度（即步进角）。可以通过控制脉冲个数来控制角位移量，从而达到准确定位的目的；同时可以通过控制脉冲频率来控制电机转动的速度和加速度，从而达到调速的目的。

人们早在 20 世纪 20 年代就开始使用这种电机，随着嵌入式系统(例如打印机、磁盘驱动器、玩具、雨刷、震动寻呼机、机械手臂和录像机等)的日益流行，步进电机的使用也开始暴增。不论在工业、军事、医疗、汽车还是娱乐业中，只要需要把某件物体从一个位置移动到另一个位置，步进电机就一定能派上用场。步进电机有许多种形状和尺寸，但不论形状和尺寸如何，它们都可以归为两类：可变磁阻步进电机和永磁步进电机。

步进电机是由一组缠绕在电机固定部件——定子齿槽上的线圈驱动的。通常情况下，一根绕成圈状的金属丝叫做螺线管，而在电机中，绕在齿上的金属丝则叫做绕组、线圈或相。

- **步进电机控制电路**

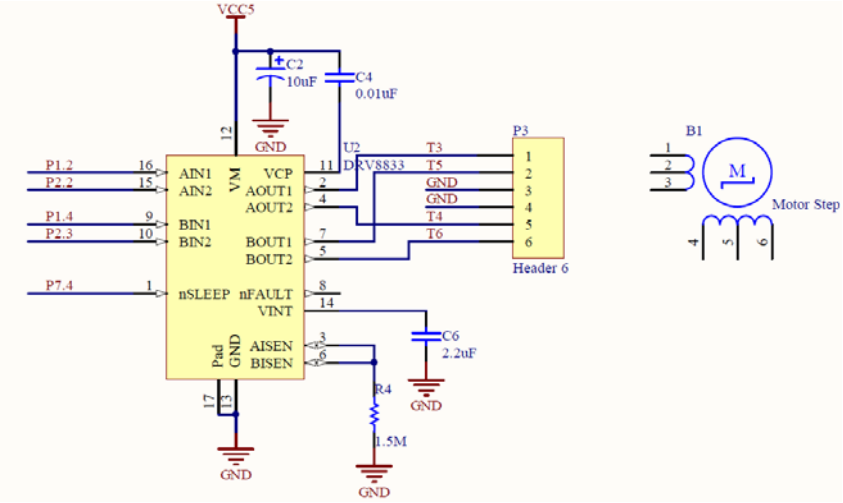

图3.6.5　步进电机模块电路原理图

步进电机控制电路同样使用了一片 TI DRV8833 低电压电机驱动芯片，关于这个芯片的介绍请看直流电机实验部分。

3.6.2.4 程序分析

- **编程思路**

编程思路非常简单，在明白 DRV8833 控制逻辑和步进电机驱动电流以后，只需要简单的在对应引脚上输出想要的电平就行。

- **程序流程图**

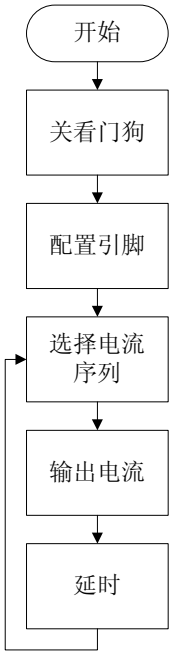

图3.6.6　程序流程图

- 关键代码分析

验 步进电机简单控制 代码

步进控制函数

```
void step(int p)
{
  switch(p)
  {
  case 0:
    {
      P7OUT &=~ BIT4;
      P1OUT &=~ BIT2;
      P2OUT &=~ BIT2;
      P1OUT &=~ BIT4;
      P2OUT &=~ BIT3;
      break;
    }
  case 1:
    {
      P7OUT |= BIT4;
      P1OUT |= BIT2;
      P2OUT &=~ BIT2;
      P1OUT &=~ BIT4;
      P2OUT &=~ BIT3;
      break;
    }
  case 2:                                  略
  case 3:                                  略
  case 4:                                  略
  default:
      break;
  }
}
```

主函数

```
int main(void) {
    WDTCTL = WDTPW | WDTHOLD;          // Stop watchdog timer
     P1DIR |= BIT2 + BIT4;
     P2DIR |= BIT2 + BIT3;
     P7DIR |= BIT4;
     P7OUT |= BIT4;
     int a=0;
     int b[8]={1,3,2,4,1,4,2,3};          //b[0]~b[3]正转，b[4]~b[7]翻转

     while(1)
     {
          step(b[a++]);                   //正转
          if (a>3)
             a=0;
          __delay_cycles(160000);
     }
}
```

3.6.2.5 实验步骤与现象

- 实验步骤

1. 打开 CCS，新建工程，完成以上代码的编写；
2. 用 USB 数据线把开发板左端的 USB 口 J1 与电脑连接，点击开发软件上的 Debug 按钮将程序下载到开发板中；
3. 用 20-pin 连接线将电机板连接到主板 P15 或 P17 端口（注意连接线方向：连接线红线对准端口上“1”管脚）；
4. 点击开发软件上的 Resume 按钮全速运行程序，观察步进电机转动情况。

- **实验现象**

  步进电机按照程序设定转动。

3.6.2.6 实验思考

1. 步进电机转动方式与直流电机有何区别？
2. 步进电机相比直流电机有何优势？

3.7 通用串行通信接口----SPI模式

通用串行通信接口（USCI）模块是 MSP430F6638 芯片上的重要通信接口，很多外设模块使用这一接口。6638 中 USCI 接口有 2 组：USCI_Ax 与 USCI_Bx，工作模式主要有：UART、SPI、I^2C 等等，不同的 USCI 模块所具有的功能也不尽相同，其中 USCI_Ax 模块支持 UART 模式、SPI 模式，该模块还可用于 IrDA 通信的脉冲整形以及 LIN 通信的自动波特率检测等；USCI_Bx 模块支持的特性包括 I^2C 模式和 SPI 模式。本节首先介绍 SPI 模式。

- **SPI 介绍**

在同步模式下，USCI 通过 3 个或者 4 个引脚把 MSP430 连接到一个外部系统中，这些引脚分别是：UCxSMIO，UCxSOMI，UCxCLK 和 UCxSTE。同步位 UCSYNC 被置位且模式选择位 UCMODE 位选择 SPI 时 USCI 模块工作于 SPI 模式。

- **SPI 的特点**
  - 具有 7 位或 8 位的数据位选择
  - 支持 3 线或 4 线 SPI
  - 支持主从模式
  - 独立的发送和接受寄存器
  - 分离的发送和接受缓冲寄存器
  - 支持低位在前或高位在前的收发
  - 独立的接收中断和发送中断
  - 主模式下时钟频率可编程
  - 从模式可工作在 LPM4 下

- **SPI 的结构**

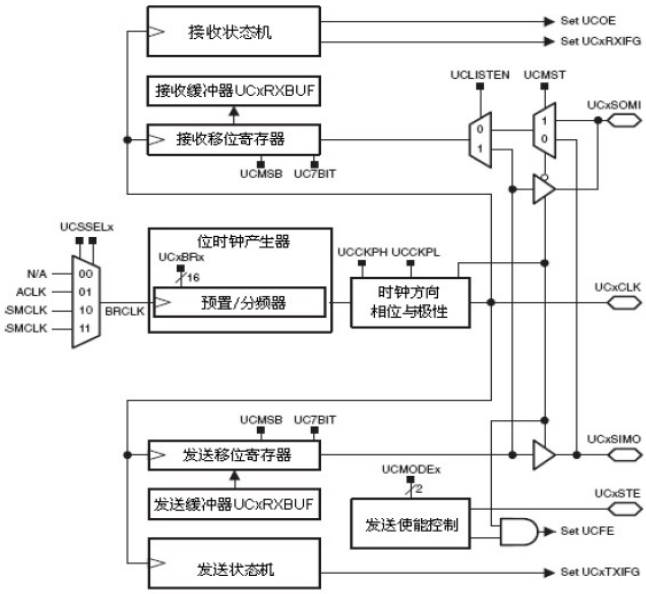

图3.7.1　SPI 接口结构框图

如上图所示，在 SPI 模式下，数据的发送和接收是由多个器件共享一个由时钟运行的，

该时钟是由一个主机提供的。UCxSIMO 是从机输入、主机输出，UCxSOMI 是从机输出、主机输入；UCxCLK 是 USCI 的 SPI 模式的时钟；UCxSTE 是从模式下的发送使能端，由主机控制，用于 4 线模式中选择一个从机接收和发送数据。

- **USCI 初始化和复位**

USCI 可以被 PUC 或者 UCSWRST 位复位。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 UCRXIE、UCTXIE、UCRXIFG、UCOE、UCFE 位并置 UCAXTXIFG 位。清除 UCSWRST 位可以是 USCI 进入工作状态。相应的 USCI 初始化/重配置的过程如下：

1. 设置 UCSWRST
2. UCSWRST=1 时初始化所有的 UCSI 寄存器（包括 UCxCTL1）
3. 配置端口
4. 软件复位 UCSWRST
5. 通过 UCRXIE 或 UCTXIE 使能中断（可选）

- **字符格式**

在 SPI 模式下的 USCI 模块支持由 UC7BIT 位来选择 7~8 位字符长度。在 7 位数据模式下，只能选择 LSB；UCMSB 位控制着数据发送的方向且选择低位在前还是高位在前，缺省情况下是低位在前。

- **主模式**

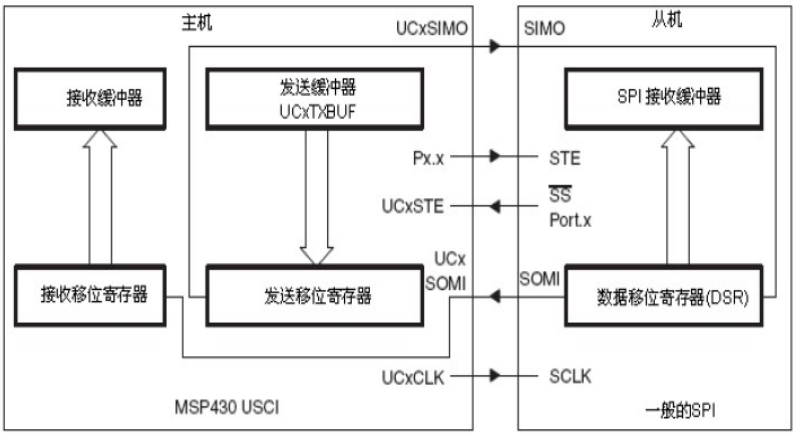

图3.7.2　主机与外部从机结构

如上图所示 MSP430 的 USCI 作为主机，当数据移动到数据发送缓冲区时，USCI 开始数据发送。当 TX 移位寄存器为空的时候，缓冲区的数据会移动到 TX 移位寄存器，开始在 UCxSIMO 口进行数据发送，并且由 UCMSB 位的设置决定高位在前还是低位在前。在相反的时钟边沿，在 UCxSOMI 端的数据被移位到数据接收寄存器。当一个字节的数据接收完成，被接收的数据就会从 RX 寄存器被移动到数据接收缓冲区 UCxRXBUF，而且接收中断标志位 UCRXIFG 被置 1。

发送中断标志位 UCTXIFG=1 意味着数据已经从发送缓冲寄存器被完整移动到发送移位寄存器，此时发送缓冲区已经准备好了发送一组新的数据，但并不意味着数据的发送和接受工作已经完成了。在主模式下，由于数据的接收和发送是并行工作的，如需要接收数据到 USCI，必须向发送缓冲区写入数据。

- 从模式

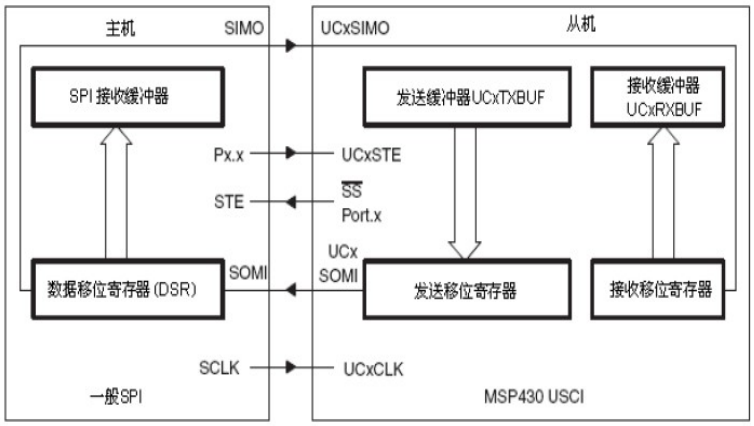

图3.7.3 从机与外部主机结构

如上图所示 MSP430 的 USCI 作为从机。UCxCLK 用于 SPI 的时钟输入而且必须有外部主机供电，数据传输的速率由这一时钟决定。待发数据应当在 UCxCLK 的起始边沿之前被写入并送至发送移位寄存器，并且会从 UCxSOMI 端口上发送出去。在时钟的另一边沿，UCxSIMO 端的数据会被移动到接收移位寄存器里，并且在预设位数的数据被接收完成后送入接收缓冲区 UCxRXBUF。当数据由接收移位寄存器移动到接收缓冲区时，接收的中断标志位被置 1，表明了数据已被接收。当之前传送的数据在新数据到来之前没有被接收缓冲区读到时会出现溢出错误，同时溢出错误标志位被置 1。

- **串行时钟控制**

UCxCLK 由 SPI 总线上的主机提供，当 UCMST=1 时，位时钟由 UCxCLK 引脚上的 USCI 位时钟产生器提供，并且由 UCSSELx 位来进行时钟选择；当 UXMST=0 时，USCI 时钟由主机的 UCxCLK 引脚提供，此时不使用位时钟产生器，并且 UCSSELx 位并不起作用。SPI 的数据接收器和发送器并行工作，使用同一个时钟源。位于位速率控制寄存器 UCxxBR1 和 UCxxBR0 中的 16 位值的 UCBRx 是 USCI 时钟源的分频数。主模式下可生成的最高频率的位时钟是 BRCLK。在 SPI 模式中不使用调制，同时 SPI 模式时的 USCI-A 模块下 UCAxMCTL 应该被清零。

- **SPI 中断**

每个 USCI 模块只有一个由发送和接收共享的中断向量，但 USCI-Ax 和 USCI-Bx 不共享同一个中断向量。发送中断标志位 UCTXIFG 是由发送器置 1 的，以便用来指示发送缓冲区 UCxTXBUF 做好接收下个字符的准备；此时如果 UCTXIE 和 GIE 也被置”1“，那么一个中断到来的时候就会产生一个中断请求。当一个字符被写入发送数据缓冲区时 UCTXIFG 会被自动复位。当 PUC 或者 UCSWRST=1 时会置位 UCTXIFG 并复位 UCTXIE。每次当接收一个字符并把字符装载到数据接收缓冲区时接收数据的中断标志位 UCRXIFG 就会被置 1，同时若 UCRXIE 和 GIE 被置 1 则也会有一个中断请求产生。UCRXIFG 和 UCRXIE 也会由 PUC 或者 UCSWRST=1 复位，当读取接收数据缓冲区时接收中断标志位也会自动复位。中断向量寄存器 UCxIV 可以被用来判断当前产生的是接收中断还是发送中断。

3.7.1 Lab7-1 SD卡读写实验

3.7.1.1 实验介绍

本实验中需用到以下电路模块：USB 接口模块、SD 卡接口模块。单片机通过 USB 与主机通信，并通过 SPI 控制 SD 卡的读写。

3.7.1.2 实验目的

- 学习 SD 卡接口硬件电路原理。
- 学习 SD 卡读写程序资源。
- 学习单片机读取 SD 卡信息的操作及编程思想。
- 学习 USB 与 PC 机的通信操作及编程思想。

3.7.1.3 实验原理

SD 读卡器模块连接在 USCI_A1 接口上。Micro SD 卡（亦称 TF 卡）是一种极细小的快闪存储器卡，其格式源自 SanDisk 公司创造，主要应用于移动电话，但因它的体积微小和储存容量的不断提高，已经使用于 GPS 设备、便携式音乐播放器和一些快闪存储器盘中。它的体积为 15×11×1mm，差不多相等于手指尖的大小，是目前最细小的记忆卡。它也能通过 SD 转接卡来接驳于 SD 卡插槽中使用。目前 Micro SD 卡提供 128MB、256MB、512MB、1G、2G、4G、8G、16G、32G 和 64G 的容量。

程序中的 USB MSC(USB mass storage device class)模块为 PC 机提供了一个 U 盘的操作接口，使得 PC 机可以像操作 U 盘一样对 SD 卡进行操作。这一接口对 SD 卡的操作是通过程序中的 SD 卡模块的盘级接口部分实现的。

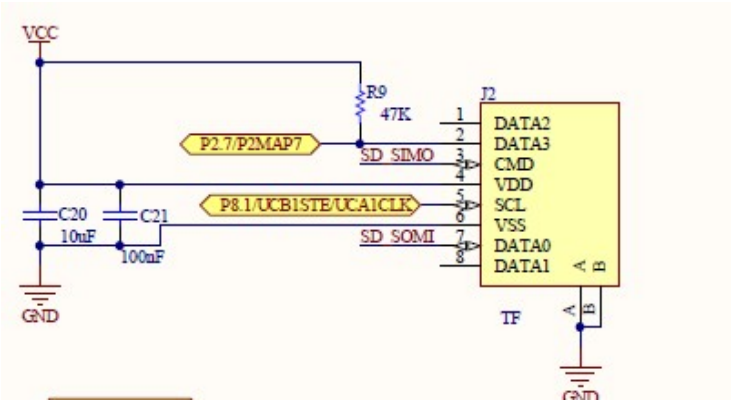

图3.7.4 SD 卡模块电路原理图

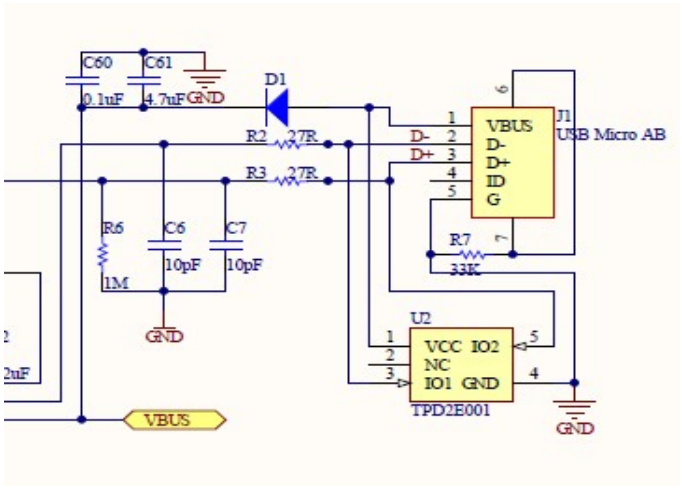

图3.7.5 USB 接口（J1）电路原理图

3.7.1.4 程序分析

- **编程思路**

SD 卡工作有两种模式，既 SD 模式和 SPI 模式，通过拉低 CS 线并发送 REST 命令进入 SPI 模式；然后再初始化 SD 卡和 USB 接口。通过 FAT 文件系统对 SD 进行访问。

- **程序流程图**

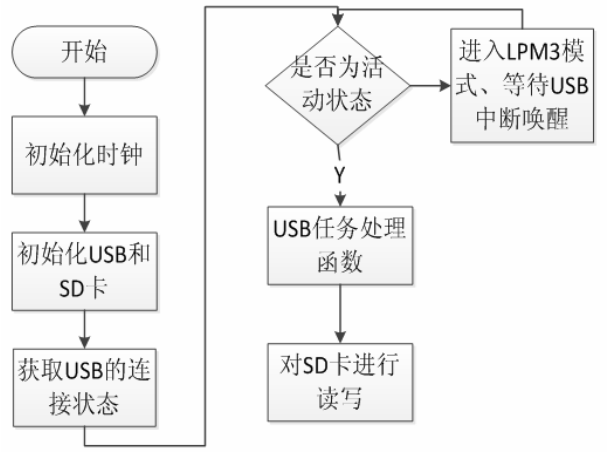

图3.7.6 程序流程图

- **关键代码分析**

实验 SD 卡读写 代码

SD 卡读写主函数

```
int main( void )
{
  WDTCTL = WDTPW + WDTHOLD;
  _DINT();
  initClock();                                    //初始化时钟
  initUSBMSC();                                   //初始化 USB 和 SD 卡
  _EINT();
```

```
    while (1)
    {
        switch (USB_connectionState())
        {
            case ST_USB_DISCONNECTED:
                __bis_SR_register(LPM3_bits + GIE);
                                        // 打开中中断，进入低功耗，等待 USB 激活唤醒
                _NOP();
                break;
            case ST_USB_CONNECTED_NO_ENUM:
                break;
            case ST_ENUM_ACTIVE:
              usbmsc_IdleTask();
                                        // USB 的任务处理函数，来实现对 SD 卡的读写
                break;
            case ST_ENUM_SUSPENDED:
                __bis_SR_register(LPM3_bits+GIE);
                break;
            case ST_ENUM_IN_PROGRESS:
                break;
            case ST_ERROR:
                break;
            default:;
        }
    }
    return 0;
}
```

3.7.1.5 实验步骤与现象

- **实验步骤**

1. 插入 SD 卡（4G 以下，FAT16 格式；4G 以上、FAT32 及 NTFS 格式的卡片无法识别）。
2. 将主板右侧的 USB 接口（J1）与 PC 机相连。
3. 在 PC 机上将发现 PC 机已连接到一个 U 盘，这样我们就可以像 U 盘一样操作 TF 卡。

- **实验现象**

可以读写主板上的 TF 卡

3.7.1.6 实验思考

FAT32 或 NTFS 分区格式的 SD 卡为何无法识别？

3.7.2 Lab7-2 TFT屏幕显示实验

3.7.2.1 实验介绍

TFT-LCD液晶显示屏是薄膜晶体管型液晶显示屏，也就是“真彩”(TFT)。TFT液晶为每个像素都设有一个半导体开关，每个像素都可以通过点脉冲直接控制，因而每个节点都相对独立，并可以连续控制，不仅提高了显示屏的反应速度，同时可以精确控制显示色阶，所以TFT液晶的色彩更真。TFT液晶显示屏的特点是亮度好、对比度高、层次感强、颜色鲜艳，但也存在着比较耗电和成本较高的不足。TFT液晶技术加快了手机彩屏的发展。彩屏手机中基本上都支持 65536 色，还有 26 万色、130 万色显示，有的甚至支持 1600 万色显示，这时TFT的高对比度，色彩丰富的优势就非常重要了。

3.7.2.2 实验目的

- 了解 MSP430F6638 的 SPI 时序对 TFT 屏幕控制。
- 了解 TFT 屏幕信息及控制显示原理。

3.7.2.3 实验原理

- **TFT 屏幕介绍**

TFT 屏幕液晶显示器上每一个液晶像素点都是由集成在其后的薄膜晶体管来驱动，从而可以做到高速度高亮度高对比度显示屏幕信息，TFT 能够显示出彩色图片是因为显示屏由许多可以发出意颜色的光线的像素组成，只要控制各个像素显示相应的颜色就能达到目的了。为了能精确地控制每一个像素的颜色和亮度就需要在每一个像素之后安装一个类似百叶窗的开关，当“百叶窗”打开时光线可以透过来，而“百叶窗”关上后光线就无法透过来。

主板上使用的是 2.2 英寸 TFT-LCD 模块，其分辨率为 240×320 像素，具有 18 位色彩深度。v1.93 版本主板使用的是一块 Toshiba 屏幕，有 41 个外部引脚，可串行或并行对控制（本系统使用串行控制），其驱动 IC 为 uPD161704（NEC）。v1.93a 与 v1.95 版本主板使用的是一块 Hitachi 屏幕，具有相同的尺寸、分辨率与色彩深度，这是一片 IPS 屏幕，具有更好的视角与亮度，使用的驱动 IC 是 BD663474（Hitachi）。Hitachi 屏是一块 39-pin 的并口屏幕，我们通过一片外置的 CPLD 芯片（在屏幕下方）做串-并转换，因此对于 MSP430F6638 而言，依然需要用串行信号控制屏幕。

由于屏幕控制器不同，这两款屏幕的驱动函数略有不同，因此涉及屏幕的实验例程我们都提供了两个版本，其中不带尾缀的适用于 v1.93 版本主板；带有尾缀 a 的程序适用于 v1.93a 与 v1.95 版本主板。这点请特别注意，否则屏幕会出现不正常的显示。

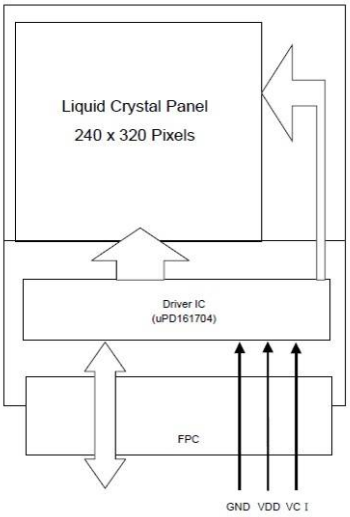

图3.7.7 TFT 屏幕模块结构图

Pin No	Symbol	I/O	Function
1	GND	—	Ground pin
2	IF_SHARE	I	Selects the interface mode
3	VDD	—	I/O Power Supply (System I/F)
4	DTX0	I	Selects the interface mode
5	DTX1	I	Selects the interface mode
6	DTX2	I	Selects the interface mode
7	PSX	I	Selects the interface mode
8	DOTCLK	I	(“L”)
9	HSYNC	I	(“L”)
10	VSYNC	I	(“L”)
11	/RD	I	Read enable signal
12	/WR	I	Write enable signal
13	/RS	I	Selects data or command
14	/CS	I	Chip selection pin
15	SCL	I	Serial clock input
16	SCI	I	Serial data input
17	D17	I/O	18-bit bi-directional data bus
18	D16	I/O	18-bit bi-directional data bus
19	D15	I/O	18-bit bi-directional data bus
20	D14	I/O	18-bit bi-directional data bus
21	D13	I/O	18-bit bi-directional data bus
22	D12	I/O	18-bit bi-directional data bus
23	D11	I/O	18-bit bi-directional data bus
24	D10	I/O	18-bit bi-directional data bus
25	D9	I/O	18-bit bi-directional data bus
26	D8	I/O	18-bit bi-directional data bus
27	D7	I/O	18-bit bi-directional data bus
28	D6	I/O	18-bit bi-directional data bus
29	D5	I/O	18-bit bi-directional data bus
30	D4	I/O	18-bit bi-directional data bus
31	D3	I/O	18-bit bi-directional data bus
32	D2	I/O	18-bit bi-directional data bus
33	D1	I/O	18-bit bi-directional data bus
34	D0	I/O	18-bit bi-directional data bus
35	/RESET	I	Initializes internally (“L”)
36	VCI	—	Power Supply for gate IC
37	VLED	—	LED Cathode
38	LED1	—	LED Anode1
39	LED2	—	LED Anode2
40	LED3	—	LED Anode3
41	GND	—	Ground pin

表 3.7.1 TFT 屏幕（Toshiba）管脚定义

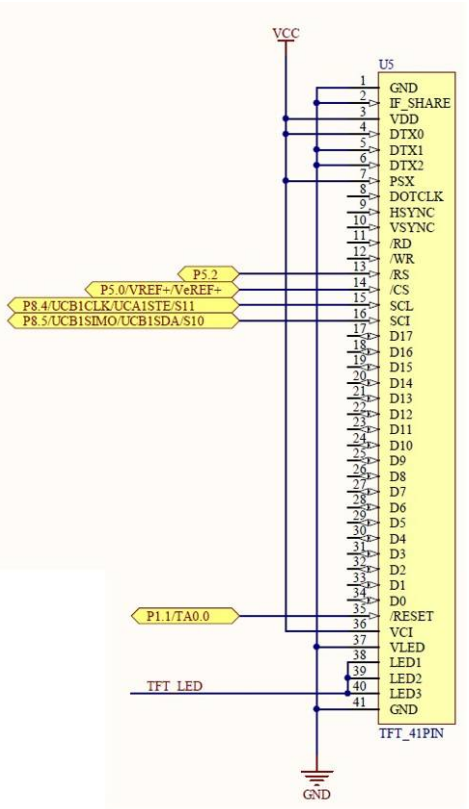

图3.7.8 TFT 屏幕接口电路原理图

3.7.2.4 程序分析

- **编程思路**

在写程序操作 TFT 屏幕显示之前首先要配置好系统时钟，即为系统时钟配置合适的晶振，然后初始化 TFT 屏幕，初始化 TFT 过程中包括了配置 MSP430F6638 的 SPI 接口以及通过 SPI 接口对 TFT 屏幕进行读写操作初始化 TFT 屏幕。初始化 MSP430F6638 的 SPI 时序接口需要配置的寄存器有 UCB1CTL1、UCB1CTL0、UCB1BRW、P8SEL 等，寄存器详细功能可查阅文档。配置完 SPI 时序后还要通过 SPI 对 TFT 屏幕的读写操作初始化 TFT 屏幕后才算完成整个初始化过程；初始化完成后在通过 SPI 对 TFT 屏幕的读写使 TFT 屏幕上显示出图片和文字来。初始化和显示 TFT 屏幕的操作可通过调用驱动来完成，相关的驱动函数及函数功能如下：

- void etft_AreaSet(uint16_t startX, uint16_t startY, uint16_t endX, uint16_t endY, uint16_t color)； 将特定的区域置为相应的颜色 color。
- void etft_DisplayString(const char* str, uint16_t sx, uint16_t sy, uint16_t fRGB, uint16_t bRGB)； 在指定位置显示一个字符串，其字符颜色为 fRGB，背景颜色为 bRGB。
- void etft_DisplayImage(const uint8_t* image, uint16_t sx, uint16_t sy, uint16_t width, uint16_t height)； 在指定位置显示一幅图片 image，图片数据以 24 位图数据表示，像素顺序从左到右，从下到上，每三个字节一个像素，顺序为 R、G、B、每行字节数用 0 补齐至 4 的整倍数，对常见的 24 位位图，从 0x36 复制到文件末尾即可。

说明：函数的形参“color、fRBG、bRGB”是 16 位无符号表示 RGB 颜色的整数，其中低五位表示蓝色光强度，高五位表示红光强度，中间六位表示绿光强度。

● 程序流程图

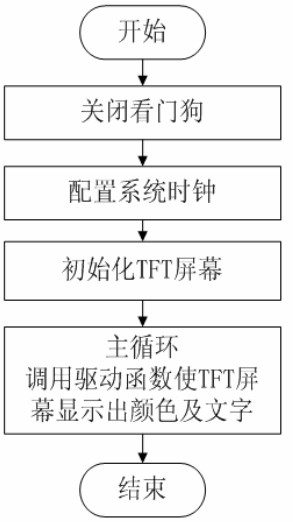

图3. 7. 9 程序流程图

● 关键代码分析

实验 TFT 屏幕显示 代码

main.c

```
#include<msp430.h>
#include<stdint.h>
#include<stdio.h>
#include "dr_tft.h"                    //调用TFT屏幕驱动头文件
voidinitClock()                        //系统时钟初始化函数
{
```

```
while(BAKCTL & LOCKIO)                                  //解锁XT1引脚操作
    BAKCTL &= ~(LOCKIO);
UCSCTL6 &= ~XT1OFF;                                     //启动XT1
 P7SEL |= BIT2 + BIT3;                                  //XT2引脚功能选择
 UCSCTL6 &= ~XT2OFF;                                    //启动XT2
   while (SFRIFG1 & OFIFG)                              //等待XT1、XT2与DCO稳定
{
UCSCTL7 &= ~(DCOFFG+XT1LFOFFG+XT2OFFG);
SFRIFG1 &= ~OFIFG;
}
UCSCTL4 = SELA__XT1CLK + SELS__XT2CLK + SELM__XT2CLK;    //避免DCO调整中跑飞
UCSCTL1 = DCORSEL_5;                                     //6000kHz~23.7MHz
UCSCTL2 = 20000000 / (4000000 / 16);          //XT2频率较高，分频后作为基准可获得更高的精度
  UCSCTL3 = SELREF__XT2CLK + FLLREFDIV__16;              //XT2进行16分频后作为基准
while (SFRIFG1 & OFIFG)                                  //等待XT1、XT2与DCO稳定
{
UCSCTL7 &= ~(DCOFFG+XT1LFOFFG+XT2OFFG);
SFRIFG1 &= ~OFIFG;
  }
  UCSCTL5 = DIVA__1 + DIVS__1 + DIVM__1;                 //设定几个CLK的分频
  UCSCTL4 = SELA__XT1CLK + SELS__DCOCLK + SELM__DCOCLK; //设定几个CLK的时钟源
}
intmain( void )
{
  WDTCTL = WDTPW + WDTHOLD;                 //停止看门狗
  _DINT();                                  //关闭总中断
initClock();                                //初始化系统时钟
initTFT();                                  //初始化TFT屏幕
  _EINT();                                  //开总中断
etft_AreaSet(0,0,319,239,0);                //清屏，把屏幕填充为黑色（color为0）
while(1)                                    //主循环
  {
      etft_AreaSet(0,0,39,239,0);           //指定区域填充黑色
      etft_AreaSet(40,0,79,239,31);         //指定区域填充蓝色
      etft_AreaSet(80,0,119,239,2016);      //指定区域填充绿色
      etft_AreaSet(120,0,159,239,63488);    //指定区域填充红色
      etft_AreaSet(160,0,199,239,2047);     //指定区域填充青色
      etft_AreaSet(200,0,239,239,63519);    //指定区域填充紫色
      etft_AreaSet(240,0,279,239,65504);    //指定区域填充黄色
      etft_AreaSet(280,0,319,239,65535);    //指定区域填充黑色
        __delay_cycles(MCLK_FREQ*3);        //延时3秒
      etft_AreaSet(0,0,319,239,0);//清屏
        __delay_cycles(MCLK_FREQ);          //延时1秒
                                            //指定位置显示指定颜色及背景颜色字符串
      etft_DisplayString("TI MSP430F6638 EVM",100,80,65535,0);
                                            //指定位置显示指定颜色及背景颜色字符串
      etft_DisplayString("TI UNIVERSITY PROGRAM",0,150,63488,0);
                                            //指定位置显示指定颜色及背景颜色字符串
      etft_DisplayString("- TSINGHUA UNIVERSITY",100,180,65504,0);
__delay_cycles(MCLK_FREQ*3);                //延时3秒
  }
}
```

3.7.2.5 实验步骤与现象

- **实验步骤**

1. 将跳线帽插到开发板上 P10 引脚的 TFT 处位置。
2. 打开 CCS5 开发软件，创建 MSP430F6638 的一个空工程。
3. 将 TFT 屏幕驱动文件 dr_tft.c、dr_tft.h、dr_tft2.c、dr_tft_ascii.h 添加到工程下。
4. 在主函数中编写以上的代码，编译，链接开发板与计算机，将程序下载到实验板上并运行，观察显示屏显示。
5. 如果无显示或显示不全请复位主板(将 P2 处 TEST、RES 跳线拔掉后按红色 RESET 键)

- **实验现象**

图 3.7.10 图形显示

图3.7.11 字符显示

3.7.2.6 实验思考

1. 如何使用驱动函数 voidetft_DisplayImage(const uint8_t* image, uint16_t sx, uint16_t sy, uint16_t width, uint16_t height)使 TFT 屏幕上显示出图片来
2. 结合开发板上的按键，欲使相应方向上的按键被按下时图片向相应的方向移动，该如何编写程序。

3.8 通用串行通信接口----I²C模式

这一节继续介绍通用串行通信接口的另一种工作模式：I²C 模式，因为只有 USCI_Bx 模块才支持 I²C 模式，因此主板上两个 I²C 接口都与 MSP430F6638 中的 USCI_Bx 连接。

- **I²C 总线介绍**

I²C 总线是由 PHILIPS 公司开发的两线式串行通信总线，一条串行数据线 SDA 和一条串行时钟线 SCL，用于连接微控制器及其外围设备。它是同步通信的一种特殊形式，具有接口线少，控制方式简单，器件封装形式小，通信速率较高等优点。

I²C 有两条总线线路，一条串行数据线 SDA 和一条串行时钟线 SCL；每一个连接到总线的器件都有唯一的地址；串行的 8 位双向数据传输位速率在标准模式下可达 100kbits/s，快速模式下可达 400kbits/s，高速模式下可达 3.4Mbits/s；支持多主控模块，但同一时刻只允许有一个主控。

- **I²C 总线协议简介**

总线只有在不忙时才可以启动数据传输，其“启动”和“停止”数据传输原理图如下图所示，当 SCL 线为高电平，SDA 线由高电平变为低电平会产生“启动”条件，所有数据传输前必须有“启动”条件；当 SCL 线为高电平，SDA 线由低电平变为高电平会产生“停止”条件，所有数据传输必须以“停止”条件结束。在“停止”条件后“启动”条件前，SCL 和 SDA 线在这期间保持高电平表示空闲状态。

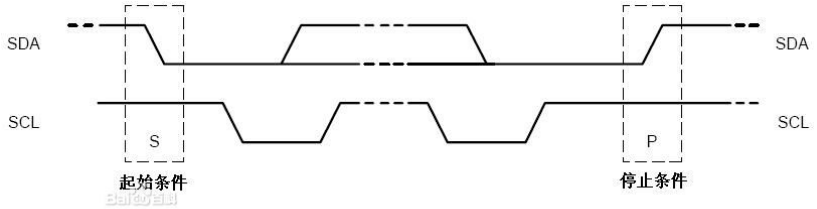

图 3.8.1 I²C 协议起始与终止条件

I²C 总线数据数据传输为字节格式，即发送到 SDA 线上的每个字节必须为 8 位，先传输最高位(MSB)，每一个被传输的字节后面都必须跟一位应答位（由从机发出）。所有主机在 SCL 线上产生它们自己的时钟来传输 I²C 总线上的报文。数据只在时钟的高电平周期有效，既在传输数据的时候，SDA 线必须在 SCL 线高电平周期保持稳定，SDA 的高或低电平状只有在 SCL 线低电平时才能改变，因此需要一个确定的时钟进行逐位仲裁。I²C 总线寻址方式有明确规定，采用 7 位寻址方式，发送的第一个字节高 7 位表示从机地址，最低位（第 8 位 LSB）决定数据的传输方向，如果是‘0’表示主机向从机写数据，如果是‘1’表示主机由从机读数据，当发送一个地址后，系统中的每个器件都在起始条件后将头 7 位与它自己的地址比较，如果一样，器件会判定它被主机寻址。

3.8.1 Lab8-1 数码管显示实验

3.8.1.1 实验介绍

该实验应用了 MSP430F6638 的 I²C 总线与外围器件 TCA6416A 连接，TCA6416A 与 8 位 7 段数码管连接，从而控制了 8 位七段数码管显示。

3.8.1.2 实验目的

- 学习 I²C 总线的工作原理
- 了解 TCA6416A 芯片的工作原理
- 掌握 7 段数码管扫描方式显示的过程

3.8.1.3 实验原理

- **MSP430F6638 的 I²C 总线介绍**

MSP430F6638 集成了 I²C 模块的接口，其中 P8.5 口作为 I²C 的 SDA 数据线，P8.6 口作为 I²C 的 SCL 时钟线，其数据传输起始位、停止位和数据位在 SDA 及 SCL 总线上的关系如图 3.8.2，7 位寻址格式如图 3.8.3 所示，先由主机发送起始（S）信号，再发送一个地址字节（高 7 位地址码，最低位 R/W），被控器件检测到主机发送地址与自己的地址相同发送一应答信号 ACK，主机接收到 ACK 后开始传输数据字节，每传输完一个字节数据后都需要被控器件反馈应答信号，直至主机发送停止位结束整个通讯，其他的 I²C 总线的工作原理在前面已经介绍过，这里不再做具体说明；MSP430F6638 的 I²C 总线控制器框图如图 3.8.4 所示。

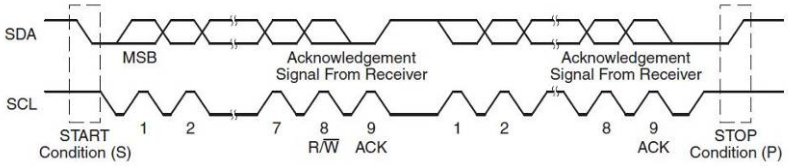

图 3.8.2 I²C 协议数据传输模式示意图

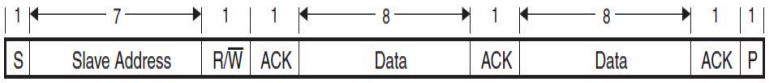

图 3.8.3 I²C 协议中 7 位寻址模式示意图

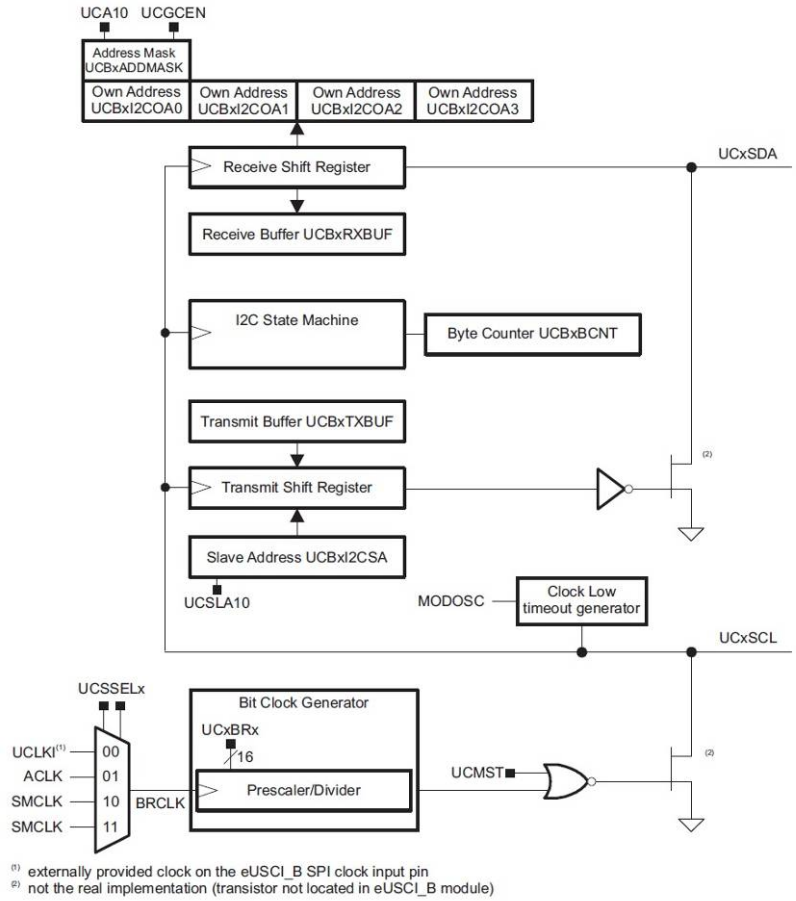

图 3.8.4　I²C 控制器示意图

- **TCA6416A 芯片介绍**

TCA6416A 是低电压 16 位 I²C 和 SMBus I/O 口拓展器，具有中断输出，复位和配置寄存器。通过 I²C 总线对芯片进行读写，可控制器件上 I/O 口，即可对这些 I/O 口进行读写操作。TCA6416A 通过 VCCI 提供电压电平转换，RESET 引脚为复位引脚，INT 可连接到微控制器的中断输入，芯片的 P 口具有输出高电流能力，可直接驱动 LED 灯的显示；TCA6416A 的终端功能如表 3.8.1 所示，TCA6416A 与 I²C 总线连接图以及与 8 位 7 段数码管连接如下图所示。

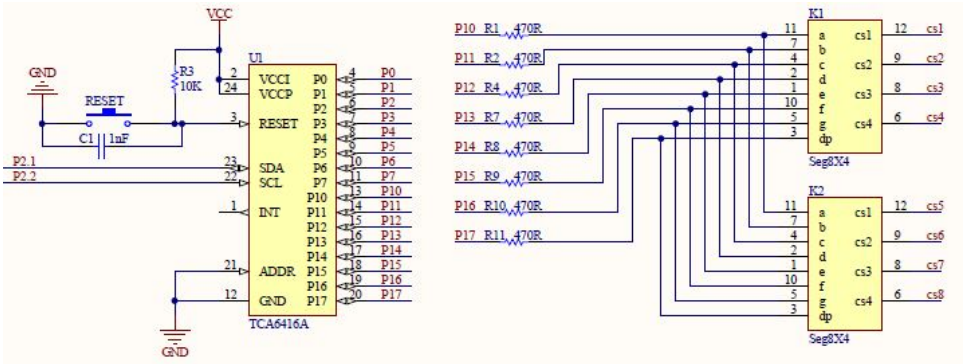

图 3.8.5　I²C 总线与 TCA6416A 芯片以及 TCA6416A 与 7 段数码管连接电路原理图

TERMINAL				DESCRIPTION
NO.			NAME	
TSSOP (PW)	QFN (RTW)	BGA (ZQS)		
1	22	A3	$\overline{INT}$	Interrupt output. Connect to V_{CCI} or V_{CCP} through a pullup resistor.
2	23	B3	V_{CCI}	Supply voltage of I^2C bus. Connect directly to the V_{CC} of the external I^2C master. Provides voltage-level translation.
3	24	A2	$\overline{RESET}$	Active-low reset input. Connect to V_{CCI} through a pullup resistor, if no active connection is used.
4	1	A1	P00	P-port input/output (push-pull design structure). At power on, P00 is configured as an input.
5	2	C3	P01	P-port input/output (push-pull design structure). At power on, P01 is configured as an input.
6	3	B1	P02	P-port input/output (push-pull design structure). At power on, P02 is configured as an input.
7	4	C1	P03	P-port input/output (push-pull design structure). At power on, P03 is configured as an input.
8	5	C2	P04	P-port input/output (push-pull design structure). At power on, P04 is configured as an input.
9	6	D1	P05	P-port input/output (push-pull design structure). At power on, P05 is configured as an input.
10	7	E1	P06	P-port input/output (push-pull design structure). At power on, P06 is configured as an input.
11	8	D2	P07	P-port input/output (push-pull design structure). At power on, P07 is configured as an input.
12	9	E2	GND	Ground
13	10	E3	P10	P-port input/output (push-pull design structure). At power on, P10 is configured as an input.
14	11	E4	P11	P-port input/output (push-pull design structure). At power on, P11 is configured as an input.
15	12	D3	P12	P-port input/output (push-pull design structure). At power on, P12 is configured as an input.
16	13	E5	P13	P-port input/output (push-pull design structure). At power on, P13 is configured as an input.
17	14	D4	P14	P-port input/output (push-pull design structure). At power on, P14 is configured as an input.
18	15	D5	P15	P-port input/output (push-pull design structure). At power on, P15 is configured as an input.
19	16	C5	P16	P-port input/output (push-pull design structure). At power on, P16 is configured as an input.
20	17	C4	P17	P-port input/output (push-pull design structure). At power on, P17 is configured as an input.
21	18	B5	ADDR	Address input. Connect directly to V_{CCP} or ground.
22	19	A5	SCL	Serial clock bus. Connect to V_{CCI} through a pullup resistor.
23	20	A4	SDA	Serial data bus. Connect to V_{CCI} through a pullup resistor.
24	21	B4	V_{CCP}	Supply voltage of TCA6416A for P port

表 3.8.1　TCA6416A 芯片终端功能

- **8 位 7 段数码管扫描显示原理**

该 8 位 7 段数码管为共阴数码管，电路连接如图 3.8.5 所示，CS1—CS8 是它们的共阴接地端，P10—P17 同时控制着每一位数码管显示的信息，若想数码管上显示出想要的数字来，如需显示 12345678，就必须进行不断的扫描处理，即采用视觉停留效果，先使第 7 位显示“1”，其他位关闭不显示，过一小段时间后（ms 级）使第 6 位显示“2”，其他位关闭不显示，如此进行重复循环着即是数码管扫描显示原理；只要扫描时间控制适当，实际上我们看到显示的就是“12345678”。

3.8.1.4 程序分析

- **编程思路**

本次试验主要是通过 I^2C 总线对器件 TCA6416A 进行读写操作，从而对 8 位 7 段数码管显示数字来，除了开始对系统的一些初始化操作外，要先对 I^2C 总线的初始配置，包括对设置 7 位地址、MSP430F6638 位主机、I^2C 总线设置同步、I^2C 时钟源、设定 P2.1 和 P2,2 分别为 I^2C 的 SDA 线及 SCL 线、开启 NACK 中断等，需要用到的相关寄存器有 UCB0CTL1、UCB0CTL0、UCB0BR0、UCB0BR1、P2SEL、P2MAP1、P2MAP2、UCB0IE，寄存器功能可查看文档。I^2C 初始化配置完成之后，主机通过 Timer2 定时中断对 TCA6416A 进行寻址、写数据操作，每次写数据使改变数码管显示，经过定时又快速的写数据操作使数码管上循环显示出数据来。对 I^2C 总线的初始化以及 TCA6416A 写数据操作可调用驱动函数来完成，相关的驱动函数介绍如下：

- int I2C_RequestSend(uint8_t slave, uint8_t reg, uint8_t value);
  该函数为请求向设备地址为 slave 的设备、设备寄存器地址为 reg 写入数据 value，成功则返回 1，错误则返回 0。

- void initI2C();
  初始化主机 I2C 总线。
- int I2C_AddRegQuery(uint8_t slave, uint8_t reg);
  该函数为添加一个I^2C读请求，失败则返回-1，成功则返回查询用代号。
- uint16_t I2C_CheckQuery(int index);
  该函数要求返回一个I^2C读请求的值，同时标记原始值为旧，准备进行下一次查询。

- **程序流程图**

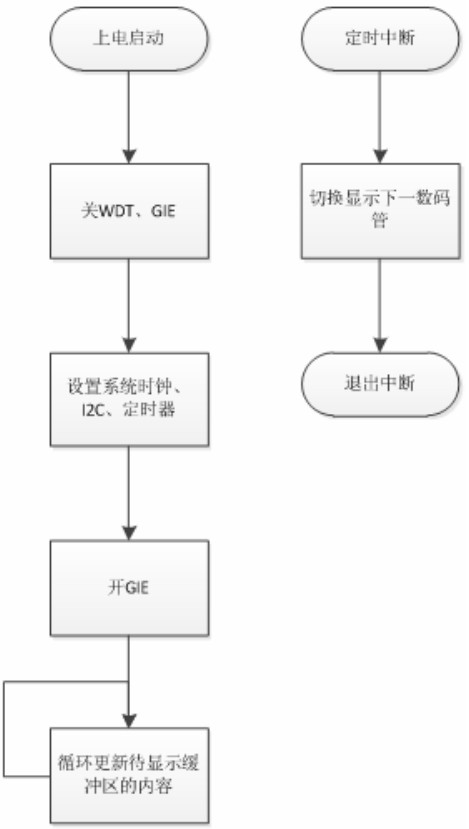

图 3.8.6　程序流程图

- **关键代码分析**

实验 数码管显示 代码

主函数

```
#include <msp430.h>
#include <stdint.h>
#include "dr_i2c.h"
#define SEG_ADDR 0x20
#define SEG_CS 2
#define SEG_SS 3
#define SEG_DIR0 6
#define SEG_DIR1 7
#define KB_ADDR 0x21
```

```
#define KB_IN 0
#define KB_OUT 1
#define KB_PIR 2
#define KB_DIR 3
const uint8_t SEG_CTRL_BIN[17] =
{
   0x3F,    //display 0
   0x06,    //display 1
   ……                                                    （其余字符略）
   0x79, //display E
   0x71, //display F
   0x80, //display .
};
constint KEYBOARD_VALUE[16] =                          //定义判别16个按键的数组
{
   15, 14, 13, 12, 11, 10, 0, 9,
   8, 7, 6, 5, 4, 3, 2, 1
};

uint8_t seg_display = 0;                                 //定义其他一些标识数
uint8_t seg_buffer[8] =
{
   0x3F,    0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07
};
int main( void )
{
   WDTCTL = WDTPW + WDTHOLD;                    //关闭看门狗
   _DINT();                                      //关闭总中断
initClock();                                     //配置系统时钟
   initI2C();                                    //初始化I2C
   _EINT();                                      //开启总中断
   I2C_RequestSend(SEG_ADDR, SEG_DIR0, 0);      //设置数码管的IO扩展器端口方向
   I2C_RequestSend(SEG_ADDR, SEG_DIR1, 0);
   while(1)
   {
int i, j;                                  //改变数码管的标识位使得数码管显示的数字变化显示
      for(i=seg_display, j=0;j<8;++j)
      {
seg_buffer[j] = SEG_CTRL_BIN[i];
         i++;
         if(i >= 16)
            i = 0;
      }
   }
}
```

定时器中断服务函数

```
#pragma vector=TIMER2_A0_VECTOR
__interrupt void softTimerInterrupt()
{
   static intcs = 0;
   I2C_RequestSend(SEG_ADDR, SEG_SS, 0);
   I2C_RequestSend(SEG_ADDR, SEG_CS, 1 <<cs);
   I2C_RequestSend(SEG_ADDR, SEG_SS, seg_buffer[cs]);          //点亮第cs位数码管
cs++;
   if(cs>= 8)                                          //如果达到第八位则重新回到第一位
cs = 0;
   static int count = 0;
```

```
    count++;
    if(count > 1000)
    {
        count = 0;
seg_display++;
        if(seg_display>= 16)
seg_display = 0;
    }
}
```

3.8.1.5 实验步骤与现象

- **实验步骤**

1. 用 10-pin 排线把键盘模块的 P1 接口与主板右上角 P14 位置的 I^2C 接口连接(注意：排线两边必须是红线对准 1 脚才是正确的接法)。
2. 新建工程，先在工程中添加驱动程序文件“dr_i2c.c”、“dr_i2c.h”及“fw_public.h”。
3. 完成以上代码的编写，将程序编译烧录到开发板中，观察 8 位 7 段数码管的显示。

- **实验现象**

图 3.8.7　数码管字符循环显示

3.8.1.6 实验思考

1. 若想要 8 位 7 段数码管上显示出任意想要的数字该如何编写代码，自写一个可调用并可显示出任意数字的函数。
2. 若想要 8 位 7 段数码管上显示出小数及负数又该如何编写代码。

3.8.2 Lab8-2 矩阵键盘实验

3.8.2.1 实验介绍

该实验应用了 MSP430F6638 的 I^2C 总线与外围器件 TCA6408A 连接，TCA6408A 与矩阵键盘连接，通过 I^2C 总线控制 TCA6408A 实时扫描矩阵键盘，如果矩阵键盘有按键被按下则点亮开发板相应的 LED1 与 LED2 灯。

3.8.2.2 实验目的

- 掌握 MSP430F6638 得 I^2C 总线工作原理
- 了解 TAC6408A 芯片的功能
- 掌握行列扫描识别矩阵键盘输入原理

3.8.2.3 实验原理

关于 I^2C 总线的介绍请看数码管显示实验中相关介绍。

- **TCA6408A 芯片介绍**

在数码管电路中使用的是 TCA6416A 芯片，矩阵键盘使用的是 TCA6408A，它与 TCA6416A 的功能及使用方法相同，但 TCA6408A 的 P 口只有 8 位，TCA6416A 的 P 口却有 16 位。矩阵键盘模块电路原理图如下所示。

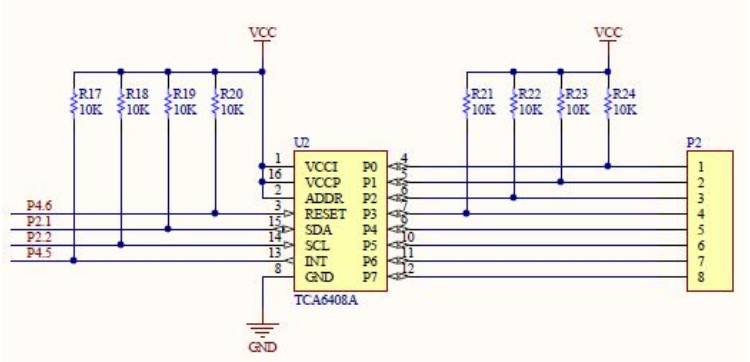

图 3.8.8 矩阵键盘模块电路原理图

- **行列扫描矩阵键盘输入原理介绍**

如上图所示，该矩阵键盘为 4X4 的输入，其中 TCA6408A 的 P0—P3 口分别与矩阵键盘的四“行”连接，P0—P3 设置为输入状态并分别接上拉电阻，P4—P7 分别与矩阵键盘的四“列”连接，P4—P7 设置为输出状态，当有按键被按下时，按键相应的“列”线会与“行”线短接。扫描开始，将第一“列”线电平置低，其他“列”线电平置高，若此时该“列”上有按键被按下则会把相应的“行”电平拉低，此时检测“行”线上是否有低电平即可知道哪

个按键被按下，若检测到没有，则将第二“列”线电平置低，其他“列”线电平拉高，再次重复检测“行”线是否有被拉低……如此重复进行下去，这就是矩阵键盘的行列扫描原理。

3.8.2.4 程序分析

- **编程思路**

实验通过 I^2C 总线对 TCA6408A 芯片进行读写操作，从而监测矩阵键盘是否有输入，如果有则进行点亮开发板上相应的 LED1 与 LED2 灯（因控制 LED3 与 LED4 灯的 IO 口已经被占用，所以只用 LED1 与 LED2 显示结果），两 LED 灯的四种状态分别表示矩阵键盘的四行按键中哪一行的键被按下。程序开始应初始化系统，在对 I^2C 总线控制器进行初始化，这部分在上一个实验里以有介绍，然后是通过 I^2C 总线控制 TCA6408A 初始化 TCA6408A 的 P0—P3 为输入状态，P4—P7 为输出状态，开启总中断，在每次进入 Timer2 中断函数中对 TCA6408A 进行读写操作来扫描矩阵键盘是否有按键被按下，若有则进行相应的处理。

- **程序流程图**

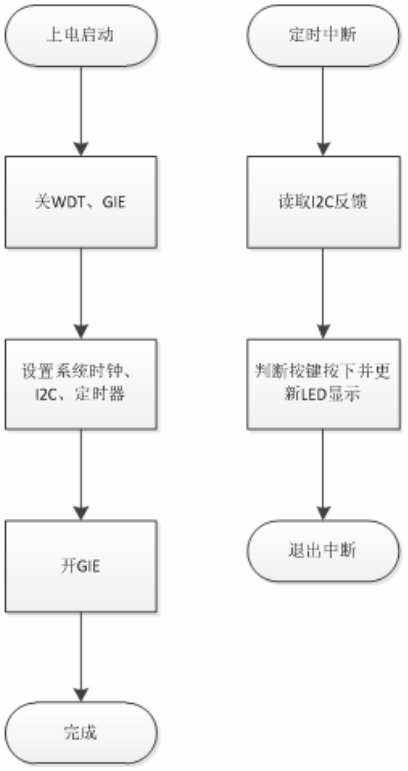

图 3.8.9　程序流程图

- **关键代码分析**

实验 矩阵键盘实验 代码

主函数

```
#include <msp430.h>
#include <stdint.h>
#include "dr_i2c.h"
#define SEG_ADDR 0x20
#define SEG_CS 2
```

```
#define SEG_SS 3
#define SEG_DIR0 6
#define SEG_DIR1 7
#define KB_ADDR 0x21
#define KB_IN 0
#define KB_OUT 1
#define KB_PIR 2
#define KB_DIR 3
uint8_t kb_line = 0;
uint16_t cur_kb_input = 0xFFFF, last_kb_input = 0xFFFF; //定义一些标识数
intkb_in;                                               //保存检查键盘输入用的 I2C 查询代号
int main( void )
{
    // Stop watchdog timer to prevent time out reset
     WDTCTL = WDTPW + WDTHOLD;
     _DINT();                                           //关闭总中断
     initClock();                                       //配置系统时钟
     initI2C();                                         //初始化 I2C 总线
     _EINT();                                           //开启总中断
     P5SEL&=~BIT7;                                      //初始化 LED 灯
     P5DIR|=BIT7;
     P5OUT&-~BIT7;
     P8SEL&=~BIT0;
     P8DIR|=BIT0;
     P8OUT&-~BIT0;
     I2C_RequestSend(KB_ADDR, KB_DIR, 0x0F);            //设置矩阵键盘的 IO 扩展器端口方向
     I2C_RequestSend(KB_ADDR, KB_OUT, ~(1 << (kb_line + 4)) ); //设置矩阵键盘的初始扫描线
     kb_in = I2C_AddRegQuery(KB_ADDR, KB_IN);                  //要求自动查询键盘输入
     while(1);                                                 //循环等待
}
```

定时器中断服务函数

```
pragma vector=TIMER2_A0_VECTOR
__interrupt void softTimerInterrupt()
{
static intkbcount = 0;
   kbcount++;
    I2C_CheckQuery(kb_in);                                  //空读取，迫使 I2C 模块更新数据
    if(kbcount> 10)
    {
     kbcount = 0;
     cur_kb_input&= ~(0xF << (kb_line * 4) );
       cur_kb_input |= ((I2C_CheckQuery(kb_in) & 0xF) << (kb_line * 4));

     uint16_t temp = (cur_kb_input ^ last_kb_input) &last_kb_inpu      //找出向下跳变的按键
     int i;
     for(i=0;i<16;++i)
     {
         if(temp & (1<<i))
         {
            if(i>=12&&i<=15) {P5OUT&=~BIT7;P8OUT&=~BIT0;}      //识别为第一行的键盘输入
            if(i>=8&&i<=11) {P5OUT|=BIT7;P8OUT&=~BIT0;}        //识别为第二行的键盘输入
            if(i>=4&&i<=7) {P5OUT&=~BIT7;P8OUT|=BIT0;}         //识别为第三行的键盘输入
            if(i>=0&&i<=3) {P5OUT|=BIT7;P8OUT|=BIT0;}          //识别为第四行的键盘输入
            break;
         }
     }
     last_kb_input = cur_kb_input;
     kb_line++;
```

```
        if(kb_line>= 4)
        kb_line = 0;
        I2C_RequestSend(KB_ADDR, KB_OUT, ~(1 << (kb_line + 4)) );
        }
}
```

3.8.2.5 实验步骤与现象

- **实验步骤**

1. 用 10-pin 排线把键盘模块的 P1 接口与主板右上角 P14 位置的 I^2C 接口连接（注意：排线两端必须是红线对准 1 脚才是正确的接法）。
2. 新建工程，先在工程中添加驱动程序文件“dr_i2c.c”、“dr_i2c.h”及“fw_public.h”。
3. 完成以上代码的编写，将程序编译烧录到开发板中，按下矩阵键盘上的按键，观察开发板上 LED1 与 LED2 灯的变化。

- **实验现象**

按下矩阵键盘上的按键，可以看到 LED1 与 LED2 的亮灭变化。

3.8.2.6 实验思考

结合前面的实验，通过扫描矩阵键盘的输入，把相应的按键值在 8 位 7 段数码管上显示出来。

3.8.3 Lab8-3 温度检测与电量检测实验

3.8.3.1 实验介绍

该实验涉及到了 MSP430F6638 试验箱多个外围设备，其中 BQ24230 充电芯片用来为锂电池充电，TPS63061 电压降压—升压转换芯片将电源电压升至 5V，电源监视芯片 BQ27410 用来监视电池的电流、电压、充电状态等。TMP421 远程温度传感器用来采集温度信息。MSP430F6638 通过 I^2C 接口与 TMP421 和 BQ27410 芯片进行通信。将采集到的信息显示在 TFT 屏上。

3.8.3.2 实验目的

- 了解 BQ24230 充电芯片和 TPS63061 芯片的使用
- 熟悉 I^2C 通信原理、TFT 屏显示原理。
- 掌握通过 MSP430F6638 的 I^2C 中断来收发数据。
- 掌握 BQ27410 电源监视芯片和 TMP421 温度传感器的使用

3.8.3.3 实验原理

I^2C 通信原理参考前面的 I^2C 实验，BQ24230 充电芯片和 TPS63061 芯片的电路连接请参考电源板的电路原理图，其中 BQ24230 芯片的 CE 管脚是用来选择是否充电，EN1、EN2 管脚来选择充电的方式。在这里着重介绍温度传感器 TMP421 和电源监控芯片 BQ27410 的使用。

- **TMP421 温度采集电路**

TMP421 是一款远程温度传感器控制芯片，内置一本地温度传感器，与 I^2C 和 SMBus 串行总线兼容，实现远程控制。如下图所示，通过 DXP、DXN 连接一个三极管或二极管组成一个远程温度传感器，远程温度传感器最大精度为±1℃；本地温度传感器最大精度为±1.5℃。本次实验是通过 I^2C 来控制 TMP421 的温度采集的，将格式设为扩展二进制，温度检测范围为-64 ~ 191℃。通过配置 TMP421 的配置寄存器来初始化，然后通过读取它的本地温度寄存器和远程温度寄存器来获取温度数据。

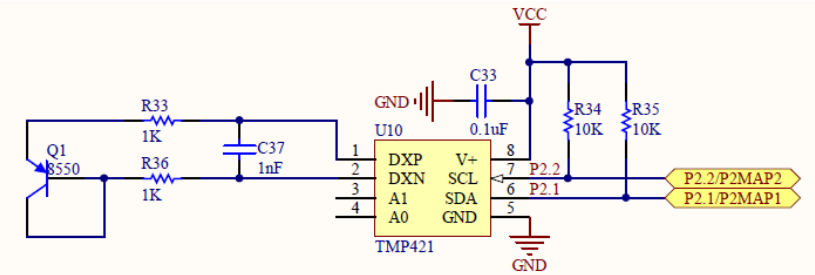

图 3.8.10 TMP421 温度采集电路

- **BQ27410 电源监视电路**

BQ27410 系统端 LiCoO2 电池电量监测计是一款易于配置的微处理器外设，可为单体 LiCoO2 电池组提供电量监测。采用获专利的 Impedance Track™算法支持电量监测，可提供剩余电池容量(mAh)、充电状态(%)、电池电压(mV) 等信息。通过 BQ24710 进行电池电量监测只需将 PACK+ (P+)与 PACK– (P–) 连接至可拆卸电池组或嵌入式电池电路。微处理器通过 BQ27410 提供的 I^2C 接口以及命令来控制 BQ27410 的数据采集和数据传送。

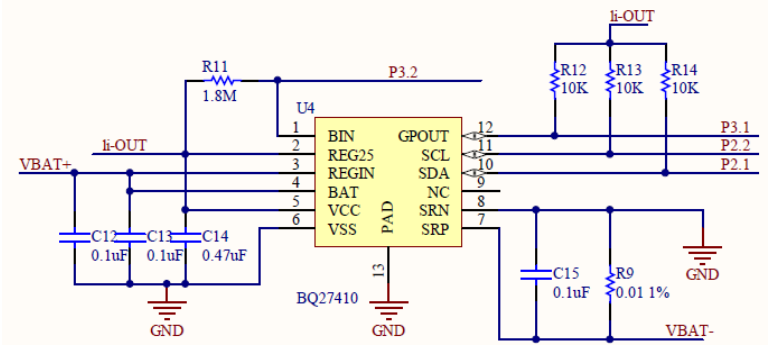

图 3.8.11 BQ27410 电源监视电路

3.8.3.4 程序分析

- **编程思路**

本实验的设计难点在于在中断里要处理 I^2C 的数据的接收和发送，中断里的接收和发送采用状态机的方式进行；每次中断，中断程序首先判断中断源再判断目前的状态，然后执行相关的程序，然后在跳转到下一个状态，等待下一个中断的到来。在主函数中首先初始化时钟、I^2C、TFT；然后配置 TMP421，获取要查询的数据的索引以及开始第一次查询；发送电池插入和报告状态命令，这时 BQ27410 在采集数据和更新数据以及等待主机的查询。主程序在 while 循环里不断的发送查询命令和更新要显示的数据。

- **程序流程图**

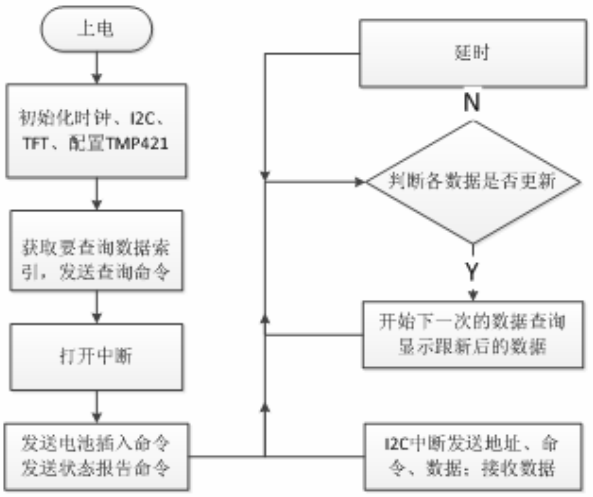

图 3.8.12 程序流程图

- 关键代码分析

实验 温度与电量检测 代码

I2C 中断函数

```
#pragma vector=USCI_B0_VECTOR
__interrupt void i2cInterrupt()
{
  if(UCB0IFG & UCNACKIFG)                          //从机没有响应
  {
    switch(curI2CStatus)
    {
    case SS_WAIT_FOR_SEND_REG:
    case SS_WAIT_FOR_SEND_DATA:
    case SS_WAIT_FOR_END:
      OVERFLOW_ADD(curSendingNo, TXREQ_BUFFER_SIZE);
      break;
    case RS_WAIT_FOR_SEND_REG:
    case RS_WAIT_FOR_SEND_READ_START:
    case RS_WAIT_FOR_RECV:
    case RS_WAIT_FOR_RECV2:
      OVERFLOW_ADD(curRecvingNo, rxreg_count);
      break;
    default:
      break;
    }
    curI2CStatus = I2C_NONE;                       //检查发送与接收均需要此状态
    UCB0IFG = 0;
    if(!I2C_Send())
      if(!I2C_StartQuery())                        //重新查询
        UCB0CTL1 |= UCTXSTP;
  }
  else
  {
    switch(curI2CStatus)                           //判断当前状态
    {
    case SS_WAIT_FOR_SEND_REG:                     //以下三个状态发送数据
      UCB0TXBUF = tx_requests[curSendingNo].reg;
      curI2CStatus = SS_WAIT_FOR_SEND_DATA;        //跳转到下个状态
      break;
    case SS_WAIT_FOR_SEND_DATA:
      UCB0TXBUF = tx_requests[curSendingNo].value;
      curI2CStatus = SS_WAIT_FOR_END;
      break;
    case SS_WAIT_FOR_END:
      curI2CStatus = I2C_NONE;                     //检查发送与接收均需要此状态
      UCB0IE &= ~UCTXIE;
      OVERFLOW_ADD(curSendingNo, TXREQ_BUFFER_SIZE);
      UCB0CTL1 |= UCTXSTP;
      while(UCB0CTL1 & UCTXSTP);                   //等待发送 STOP 结束
      __delay_cycles(100);                         //延迟 5us
      if(!I2C_Send())                              //查询
        I2C_StartQuery();
      break;
    case RS_WAIT_FOR_SEND_REG:                     //以下状态是接收数据
      UCB0TXBUF = rx_regs[curRecvingNo].reg;
      curI2CStatus = RS_WAIT_FOR_SEND_READ_START;
      break;
    case RS_WAIT_FOR_SEND_READ_START:
```

```
            UCB0I2CSA = rx_regs[curRecvingNo].slave;          //设定从机地址
            UCB0CTL1 &= ~UCTR;                               //设置接收模式
            UCB0CTL1 |= UCTXSTT;                              //发送 START
            UCB0IE &= ~UCTXIE;                               //关闭发送中断
            UCB0IE |= UCRXIE;                                //启动接收中断
            curI2CStatus = RS_WAIT_FOR_RECV;
            break;
        case RS_WAIT_FOR_RECV:
            curI2CStatus = RS_WAIT_FOR_RECV2;
  rx_regs[curRecvingNo].value = UCB0RXBUF;
            UCB0CTL1 |= UCTXSTP;                             //告知将在下一字节后停止
            break;
        case RS_WAIT_FOR_RECV2:
            curI2CStatus = I2C_NONE;                         //检查发送与接收均需要此状态
            UCB0IE &= ~UCRXIE;
  rx_regs[curRecvingNo].value |= (UCB0RXBUF << 8) & 0xFF00;
  rx_regs[curRecvingNo].hasnew = 1;
            OVERFLOW_ADD(curRecvingNo, rxreg_count);     //继续查询需要下一地址
            while(UCB0CTL1 & UCTXSTP);                        //等待发送 STOP 结束
            __delay_cycles(100);                             //延迟 5us
            if(!I2C_Send())                                   //重新查询
              I2C_StartQuery();
            break;
        default:
            break;
        }
    }
}
```

3.8.3.5 实验步骤与现象

- **实验步骤**

1. 将锂电池模块插头插入电池板上 BT1 插座中，电池板上跳线帽接到 EN1 上；
2. 将电池板插入主板背后 P6、P7 插槽中；
3. 用 USB 线将电脑与主板左侧的 USB 接口（J6）连接，供电开关 SW1 拨到 eZ430 位置；
4. 下载程序，观察 TFT 屏幕显示和主板右上角 LED9 指示灯。

- **实验现象**

当电池处于充电状态时 LED9 为点亮状态，当电池充满之后 LED9 熄灭。TFT 屏幕上显示温度及电池信息，由于未对电池监视芯片的电流测量进行校正，所以显示在 TFT 上的电流测量值实际上是不正确的（在现有采样电阻下约为实际值的两倍），由此也会引起监视芯片内部相关的功率与能量计算都有相应的误差。

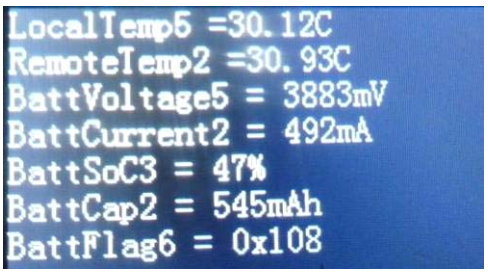

图 3.8.13 屏幕显示

3.8.3.6 实验思考

1. 如何对电流测量进行校正？
2. EN1、EN2 跳线的连接，对电池的充电有何影响？

3.9 通用串行通信接口----UART模式

本节介绍通用串行通信接口（USCI）模块的另一种工作模式 UART 模式，只有 USCI_Ax 模块支持 UART 模式，因此这里介绍 USCI_Ax 模块。在 UART 模式中，USCI_Ax 模块是通过两个外部引脚连接 MSP430 到外部系统，分别是 UCAXRXD 和 UCAXTXD。当 UCSYNC 位被清零时 UART 模式被选择。

- **USCI_Ax 模块特点**

该模块支持多种串行通信的模式，其中包括 UART 模式、带有脉冲整形的 IrDA 通信、带有自动波特率检测的 LIN 通信、SPI 模式。多种模式能满足多种不同接口的微机通信，这里主要介绍其中的最简单一种 UART 通信模式。

- **UART 模式下的 USCI_Ax 模块结构**

其结构框图如下图所示，可分为接收、发送、红外接口和波特率发生器四个主要部分。

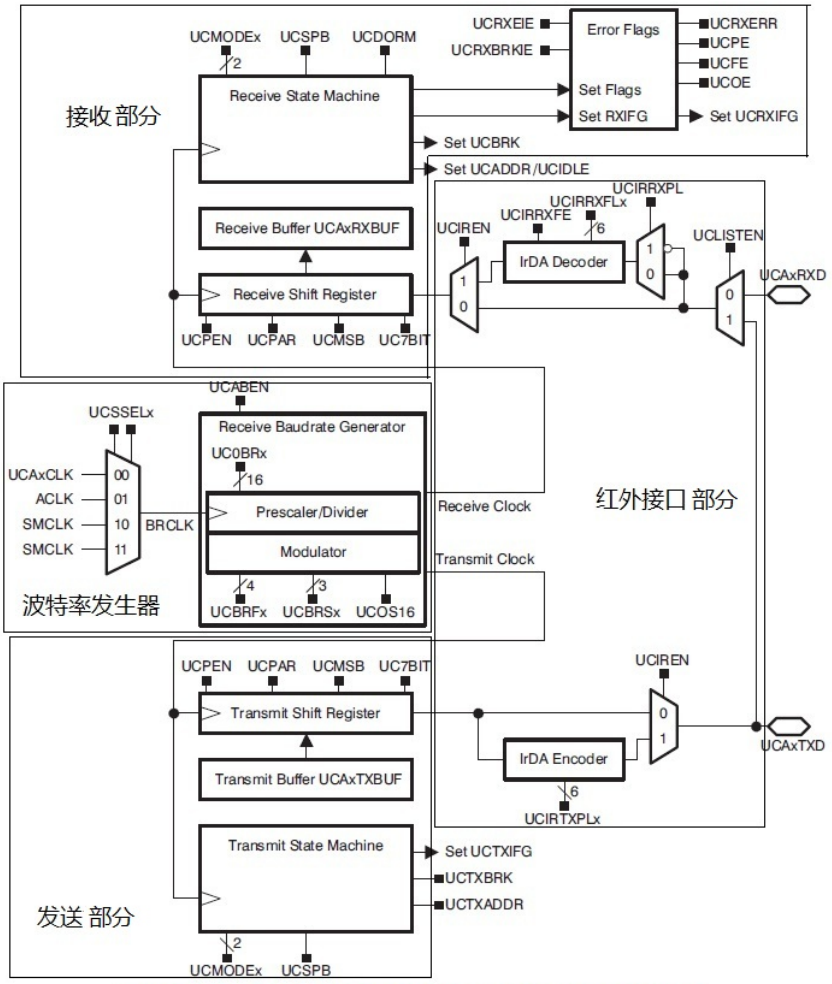

Figure 34-1. USCI_Ax Block Diagram – UART Mode (UCSYNC = 0)

图3.9.1 UART模式下USCI_Ax结构框图

- **接收部分：**接收部分将从UART外部接收串行数据，将所接收的数据放到UCAxRXBUF寄存器中存放起来，以便处理器读取。接收的格式由寄存器

UCAxCTL0各个位决定，接收过程由接收状态机控制。同时接收器状态机还要对数据位的溢出出错，奇偶检验出错，帧出错,BREAK等进行检验，并根据检验结果设置状态寄存器UCAxSTAT中相应的状态位。

- **发送部分：**发送部分将从处理器接收到的数据（存放在UCAxTXBUF寄存器中），按规定的格式加上起始位，奇偶检验位和停止位后串行输出。发送数据格式由寄存器UCAxCTL0各个位来决定，发送过程由发送状态机来控制。

- **波特率发生器：**波特率发生器部分提供UART进行通信时所需时钟，还需对外部接收时钟进行同步处理,通过控制UCAxCTL1寄存器中的UCSSELx来选择输入时钟。UCAxBR0和UCAxBR1分别为控制波特率低八位和高八位数据的寄存器，还可以通过UCAxABCTL寄存器中的UCABDEN来使能波特率自动设置功能。

- **红外接口部分：**主要由编码部分和解码部分完成，编码部分完成RS232到IRDA之间的转换，解码部分完成IRDA到RS232之间的转换。UART 在使用中，通过控制寄存器UCAxIRTCTL的UCIREN来决定采用哪种通信协议，UCIREN置1时，使能编码与解码器。

USCI_Ax 模块在 UART 模式工作方式配置相关寄存器如下图所示

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	UCAxCTL1	USCI_Ax Control 1	Read/write	Byte	01h	Section 34.4.2
01h	UCAxCTL0	USCI_Ax Control 0	Read/write	Byte	00h	Section 34.4.1
06h	UCAxBRW	USCI_Ax Baud Rate Control Word	Read/write	Word	0000h	
06h	UCAxBR0	USCI_Ax Baud Rate Control 0	Read/write	Byte	00h	Section 34.4.3
07h	UCAxBR1	USCI_Ax Baud Rate Control 1	Read/write	Byte	00h	Section 34.4.4
08h	UCAxMCTL	USCI_Ax Modulation Control	Read/write	Byte	00h	Section 34.4.5
09h		Reserved - reads zero	Read	Byte	00h	
0Ah	UCAxSTAT	USCI_Ax Status	Read/write	Byte	00h	Section 34.4.6
0Bh		Reserved - reads zero	Read	Byte	00h	
0Ch	UCAxRXBUF	USCI_Ax Receive Buffer	Read/write	Byte	00h	Section 34.4.7
0Dh		Reserved - reads zero	Read	Byte	00h	
0Eh	UCAxTXBUF	USCI_Ax Transmit Buffer	Read/write	Byte	00h	Section 34.4.8
0Fh		Reserved - reads zero	Read	Byte	00h	
10h	UCAxABCTL	USCI_Ax Auto Baud Rate Control	Read/write	Byte	00h	Section 34.4.11

图3.9.2　相关配置寄存器

3.9.1 Lab7-1 RS232 串口实验

3.9.1.1 实验介绍

RS232 接口标准是在串口通信中最基础最简单的通信协议，现已在微机通信接口中被广泛采用。本实验将利用 MSP430F6638 的 USCI_Ax 模块进行 RS232 串口通信，只需将 RS232 接口的 RXD 和 TXD 进行短接即可实现本机的发送以及接收，并把发送及接收的内容在 TFT 屏幕上显示出来。

3.9.1.2 实验目的

- 了解 RS232 接口标准概况
- 了解 MSP430F6638 的 USCI_Ax 模块的 UART 模式的使用
- 实现 MSP430F6638 的串口通信

3.9.1.3 实验原理

- **RS232 简介**

RS232 的通信标准中在早期是以一个 25 针的接口来定义的，并在 PC 机或 XT 机型上广泛使用，但是在 AT 机以后的机型上，实际采用了 9 针的简化版本应用，现在的 RS232 通信均默认为 9 针接口，图 3.9.3 显示了 9 针通信接口的管脚定义。

在实际的应用中 RS-232 接口在 9 针的基础上可以再进行一步简化，只用其中的 2、3、5 三个管脚进行通信，在我们的开发板上的 P9 接口就是简化后的 RS-232 接口，这三个接口分别是接收线、发送线、地线，在一般情况下只需这三根即可满足通信要求，P9 接口的电路图如图 3.9.4 所示，其中针口 1 为 RX 线，针口 2 为 TX 线。

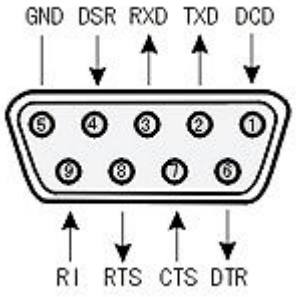

图3.9.3 RS232接口9针定义

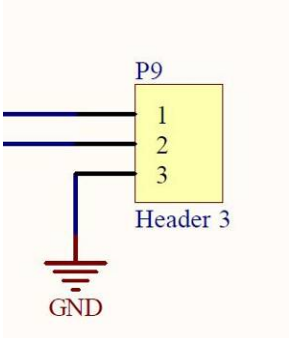

图3.9.4 开发板简化三针RS232接口

- **MSP430F6638 串口模块的使用**

MSP430F6638 自带的 USCI_Ax 模块在软件上的通信协议由使用者来确定，章节介绍中以介绍过 USCI_Ax 模块支持的四种通信方式。在此处用到的是 UART 通信模式，即通用异

步接收/发送模式，异步通讯的特性包括：

- 7 位或 8 位数据位，支持奇偶校验
- 独立的发送和接收移位寄存器
- 独立的发送接收缓存
- 可选择优先发送（接收）MSB 还是 LSB
- 空闲位多机模式和地址位多机模式
- 通过有效的起始位检测将 MSP430F6638 从低功耗中唤醒
- 状态标志检测错误或地址位
- 独立的接收和发送中断
- 可编程实现波特率调整

本实验中所用的 UART 配置为波特率 9600，8 位数据位，1 位停止位，无奇偶校验。

3.9.1.4 程序分析

- **编程思路**

本次实验使用了 USCI_Ax 模块的 UART 通信模式和定时器模块 TimerA 模块以及外围设备 TFT 屏幕，先将 UART 模式进行相应的配置，再进行初始化 TFT 屏幕，然后打开定时器，通过定时器中断方式来定时，定时 1s 通过 UART 发送出一个字符，在中断中接收数据，最后将发送和接收的数据在 TFT 屏幕上显示出来，因为这里要用到 TFT 屏幕，因此在编写代码之前先将 TFT 屏幕的驱动文件 dr_tft.c、dr_tft.h、dr_tft2.c、dr_tft_ascii.h 添加到代码工程中去。

- **程序流程图**

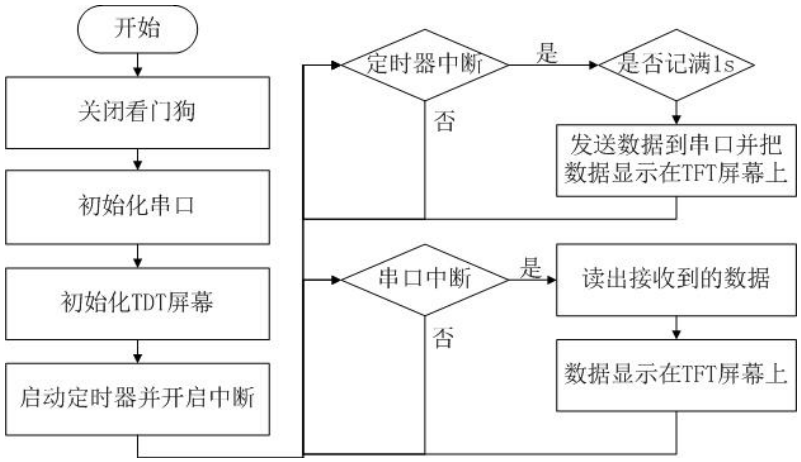

图 3.9.5 程序流程图

- **关键代码分析**

实验 RS232 通信 代码

主函数

```
#include <msp430f6638.h>
#include <stdint.h>
#include <stdio.h>
#include "dr_tft.h"
unsigned char flag0=0,flag1=0;                    //定义定时和接收数据的标识数据
unsigned char send_data[]={'0','\0'};             //发送的数据
unsigned char recv_data[]={'0','\0'};             //接收到的数据
```

```
void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;                          //关闭看门狗
    UART_RS232_Init();                                 //初始化 RS232 接口
    initTFT();                                         //初始化 TFT 屏幕
    etft_AreaSet(0,0,319,239,0);                       //TFT 清屏
    TimerA_Init();                                     //初始化定时器
    _EINT();                                           //开启中断
    etft_DisplayString("Send Data: ",80,100,65535,0);  //TFT 屏显示发送数据的标识
    etft_DisplayString("Recv Data: ",80,140,65535,0);  //TFT 屏显示接收数据的标识
    while(1)
    {
        if(flag0)                          //flag0 在定时器计时到达 1s 时被赋值为 1
        {
            flag0=0;
            UCA1TXBUF=send_data[0];                //写一个字符到发送缓存发送数据
            etft_DisplayString(send_data,170,100,65535,0);//TFT 屏上显示发送字符
            send_data[0]++;                        //字符加 1
            if(send_data[0]>'9')                   //字符超过'9'则重新置'0'
                send_data[0]='0';
        }
        if(flag1)                                  //flag1 在接收中断中被赋值为 1
        {
            flag1=0;
            etft_DisplayString(recv_data,170,140,65535,0);//屏幕上显示接收到字符
        }
    }
}
```

USCI 中断服务函数

```
#pragma vector=USCI_A1_VECTOR
__interrupt void USCI_A1_ISR(void)
{
    switch(__even_in_range(UCA1IV,4))
    {
    case 0:break;                                  //无中断
    case 2:                                        //接收中断处理
        while(!(UCA1IFG&UCTXIFG));                 //等待完成接收
        recv_data[0]=UCA1RXBUF;                    //数据读出
        flag1=1;                                   //置标识数据的值
        break;
    case 4:break;                                  //发送中断不处理
    default:break;                                 //其他情况无操作
    }
}
```

定时器 TA 中断服务函数

```
#pragma vector = TIMER0_A0_VECTOR
__interrupt void Timer_A (void)
{
    static unsigned char i=0;
    i++;
    if(i>=20)                                      //记满二十次为 1s
    {
        i=0;
        flag0=1;                                   //改变标识数据的值
    }
}
```

RS232 接口初始化函数

```
void UART_RS232_Init(void)
{
    /*通过对P3.4，P3.5，P4.4，P4.5的配置实现通道选择 使USCI切换到UART模式*/
    P3DIR|=(1<<4)|(1<<5);
    P4DIR|=(1<<4)|(1<<5);
    P4OUT|=(1<<4);
    P4OUT&=~(1<<5);
    P3OUT|=(1<<5);
    P3OUT&=~(1<<4);
    P8SEL|=0x0c;          //模块功能接口设置，即P8.2与P8.3作为USCI的接收口与发射口
    UCA1CTL1|=UCSWRST;                                          //复位USCI
    UCA1CTL1|=UCSSEL_1;                          //设置辅助时钟，用于发生特定波特率
    UCA1BR0=0x03;                                               //设置波特率
    UCA1BR1=0x00;
    UCA1MCTL=UCBRS_3+UCBRF_0;
    UCA1CTL1&=~UCSWRST;                                          //结束复位
    UCA1IE|=UCRXIE;                                            //使能接收中断
}
```

定时器 TA 初始化函数

```
void TimerA_Init(void)
{
    TA0CTL |= MC_1 + TASSEL_2 + TACLR;   //时钟为 SMCLK,比较模式，开始时清零计数器
    TA0CCTL0 = CCIE;                     //比较器中断使能
    TA0CCR0  = 50000;                    //比较值设为 50000，相当于 50ms 的时间间隔
}
```

3.9.1.5 实验步骤与现象

- **实验步骤**

1、 完成代码的编写，并将代码烧录到开发板中
2、 用跳线帽将 P9 接口的 RX 和 TX 接口短接
3、 观察 TFT 屏幕上显示的信息

- **实验现象**

TFT 屏幕上显示的 RS232 端口发送与接收的数据在变化，如下图所示。

图 3.9.6 实验现象

3.9.1.6 实验思考

能否将主板左侧 USB 接口（J6）虚拟为一个串口与 PC 进行通信？如何实现？

3.10 ADC与DAC模块

模拟量--数字量转换模块（ADC）与数字量--模拟量转换模块（DAC）是非常有用的功能模块，很多单片机中只有 ADC 模块、没有 DAC 模块，但是在 MSP430F6638 中两个功能模块都有。下面我们首先简要介绍一下模拟量与数字量的概念以及他们之间的关系。

- **模拟信号**

模拟信号是指用连续变化的物理量表示的信息，其信号的幅度、频率或相位随时间作连续变化，如目前广播的声音信号，或图像信号等。当模拟信号采用连续变化的电磁波来表示时，电磁波本身既是信号载体，同时作为传输介质；而当模拟信号采用连续变化的信号电压来表示时，它一般通过传统的模拟信号传输线路（例如电话网、有线电视网）来传输。

- **数字信号**

数字信号指幅度的取值是离散的，幅值表示被限制在有限个数值之内。二进制码就是一种数字信号。它具有抗干扰能力强、无噪声积累、便于加密处理、便于存储、处理和交换等特点。

- **模拟信号与数字信号之间的相互转换**

模拟信号一般通过 PCM 脉冲编码调制(Pulse Code Modulation)方法量化为数字信号，即让模拟信号的不同幅度分别对应不同的二进制值，例如采用 8 位编码可将模拟信号量化为 2^8=256 个量级，实用中常采取 24 位或 30 位编码；数字信号一般通过对载波进行移相(Phase Shift)的方法转换为模拟信号。计算机、计算机局域网与城域网中均使用二进制数字信号，目前在计算机广域网中实际传送的则既有二进制数字信号，也有由数字信号转换而得的模拟信号。但是更具应用发展前景的是数字信号。

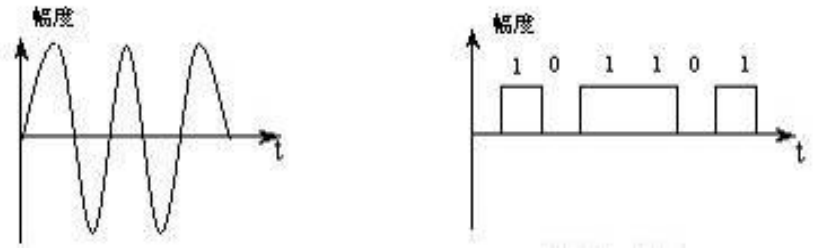

图 3.10.1　模拟信号（左）与数字信号（右）

AD 转换：AD 转换器根据其原理主要可以分为积分型、逐次逼近型和并行比较型。在获取相同的转换精度时，积分型 AD 转换器的电路最为简单、成本低、同时转换耗时最长，并行比较型则拥有最高的转换速度和最复杂的电路，逐次逼近型的性能介于另两种之间。MSP430 的 AD 模块为逐次逼近型，其转换器核心由一个比较器、一个 DA 转换器和逐次比较逻辑组成。工作时转换器会从最高位到最低位依次生成试探电压的数字值，将其送入 DA 转换器生成试探电压信号，并将这一信号与输入信号进行比较来决定每一位的取值，对于 n 位的转换器，每次转换过程需要比较 n 次。

DA 转换：DA 转换器由电阻阵列和开关组成，根据数字量输入选择导通的开关来产生所需的模拟量。

3.10.1 Lab10-1 ADC实验

3.10.1.1 实验介绍

本实验演示了 MSP430F6638 中 ADC12 模块的使用。实验结合了 ADC12 与电位器的应用，通过获取电位器的模拟电压值，实现了用电位器对 5 个不同 LED 灯的亮灭控制。

3.10.1.2 实验目的

- 了解 ADC 转换原理
- 学习 MSP430F6638 中 ADC12 的配置使用方法
- 掌握 LED 灯的控制方法
- 结合电位器与 ADC12 模块实现对 LED 灯的控制

3.10.1.3 实验原理

模数转换器（ADC）是指将连续的模拟信号转换为离散的数字信号的器件。真实世界的模拟信号，例如温度、压力、声音或者图像等，需要转换成更容易储存、处理和发射的数字形式。在 A/D 转换中，因为输入的模拟信号在时间上是连续的，而输出的数字信号是离散量，所以进行转换时只能按一定的时间间隔对输入的模拟信号进行采样，然后再把采样值转换为输出的数字量。通常 A/D 转换需要经过采样、保持量化、编码几个步骤。MSP430F6638 中的 ADC12 模块结构如图 3.10.2 所示。ADC12 模块主要组成组成部分有：

- 输入端 16 路模拟开关；
- 内部电压参考源；
- ADC12 内核；
- 时钟源部分；
- 采集与保持部分；
- 数据输出部分；
- 控制寄存器。

ADC12 的模块内核是共用的，通过前端的模拟开关来分别完成采集输入。ADC12 是一个精度为 12 位的 ADC 内核，1 位非线性微分误差，1 位非线性积分误差。内核在转换时会参用到两个参考基准电压，一个是参考相对的最大输入最大值，当模拟开关输出的模拟量大于或等于最大值时 ADC 内核的输出数字量为满量程，也就是 0xfff；另一个则是最小值，当模拟开关输出的模拟变量小于或等于最大值时，ADC 内核输出的数字量为最低值，也就是 0x000。转换后的数据将存贮在 ADC12MEMx 中，Nadc=4095*（Vin-Vr-）/（Vref-Vr-），Vr-在本实验中取值为 0，因此 Nadc=4095*Vin/Vref，Vin=Nadc*Vref/4095。

ADC12 模块的主要寄存器有 ADC12CTL0（转换控制寄存器）、ADC12CTL1（终端控制寄存器）、ADC12IFG（中断标志寄存器）)、 ADC12IE（中断使能寄存器）、ADC12IV（中断向量寄存器）。

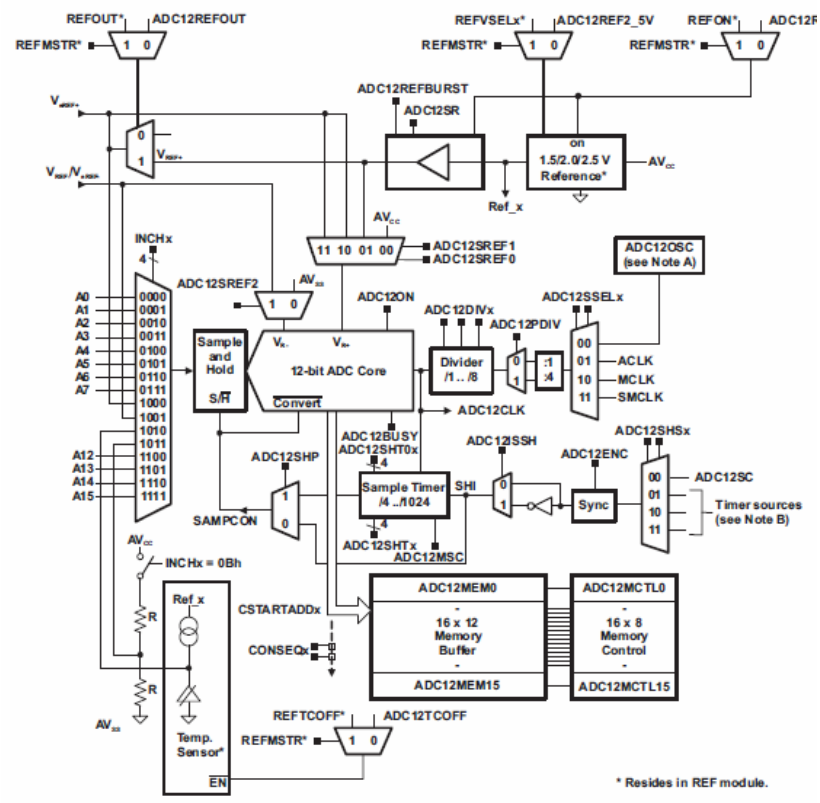

图 3.10.2　ADC12 模块结构框图

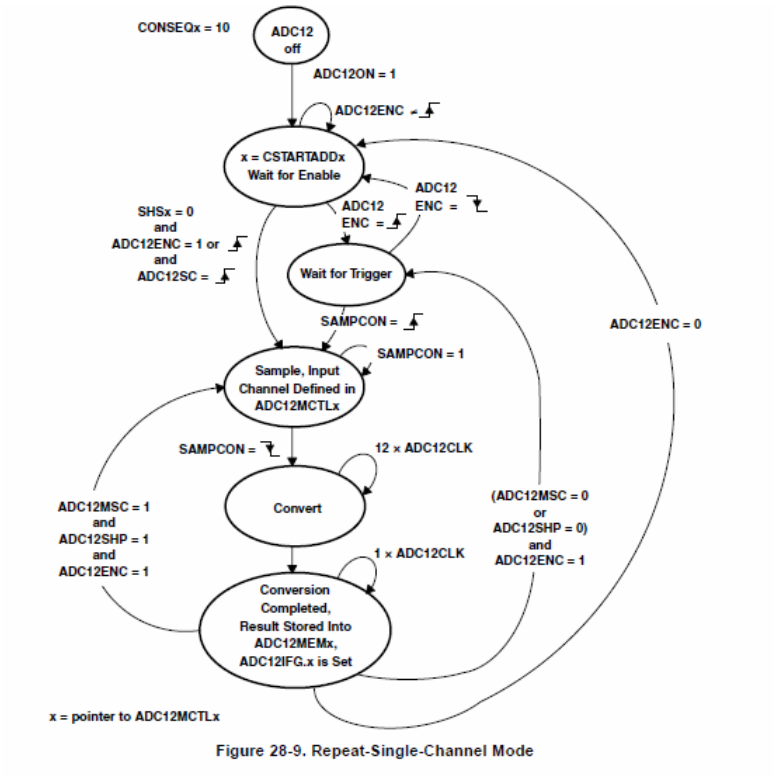

Figure 28-9. Repeat-Single-Channel Mode

图 3.10.3　ADC12 模块状态转移图

采样方式分为单通道单次采样、多通道单次采样、单通道循环采样、多通道循环采样四

种方式，我们本次实验采取的是单通道循环采样方式（上图所示）。当 ADC12ON 为高电平是，ADC 转换器启动并等待转换开始的信号；此时当 ADC12ENC 位为 1 且 ADC12SC 出现上升沿时开始转换过程，并把每次转换的数据保存在 ADC12MEN0 中。在单通道循环采样方式下转换过程会循环进行，直到将 ADC12ENC 复位。

实验中通过旋转拨盘电位器（R13）改变 ADC 端输入电压，然后依据电压高低分为几档通过 LED1~LED5 显示出来。

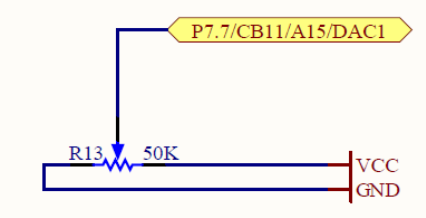

图 3.10.4　拨盘电位器模块电路原理图

3.10.1.4 程序分析

- **编程思路**

熟悉了 MSP430F6638 中的 ADC12 模块原理之后便可对其控制寄存器进行配置，设计采样模式，时刻得到电位器的输出端电压值。并通过其大小，设定范围从而来控制 LED 灯的亮灭。

- **程序流程图**

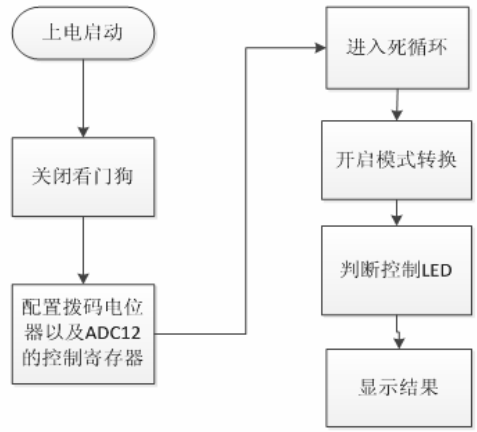

图 3.10.5　程序流程图

- **关键代码分析**

实验 ADC 代码

主函数

```
void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;              //关闭看门狗
    P4DIR |= BIT5 + BIT6 + BIT7;           //配置与LED有关的GPIO引脚
    P5DIR |= BIT7;
    P8DIR |= BIT0;
    ADC12CTL0 |= ADC12MSC;                 //自动循环采样转换
```

```
        ADC12CTL0 |= ADC12ON;                          //启动 ADC12 模块
        ADC12CTL1 |= ADC12CONSEQ1 ;                    //选择单通道循环采样转换
        ADC12CTL1 |= ADC12SHP;                         //采样保持模式
        ADC12MCTL0 |= ADC12INCH_15;                    //选择通道 15，连接电位器
        ADC12CTL0 |= ADC12ENC;
        volatile unsigned int value = 0;               //设置判断变量
        while(1)
        {
            ADC12CTL0 |= ADC12SC;                      //开始采样转换
            value = ADC12MEM0;                         //把结果赋给变量
                                /*判断结果范围，点亮 LED 灯*/
            if(value > 5)
                P4OUT |= BIT5;
            else
                P4OUT &= ~BIT5;
            if(value >= 800)
                P4OUT |= BIT6;
            else
                P4OUT &= ~BIT6;
            if(value >= 1600)
                P4OUT |= BIT7;
            else
                P4OUT &= ~BIT7;
            if(value >= 2400)
                P5OUT |= BIT7;
            else
                P5OUT &= ~BIT7;
            if(value >= 3200)
                P8OUT |= BIT0;
            else
                P8OUT &= ~BIT0;
        }
    }
```

3.10.1.5 实验步骤与现象

- **实验步骤**

1. 根据编程思路设计结构与实现方法。
2. 按照流程图实现代码编写，并在编译器上进行编译改错。
3. 将程序烧入开发板中进行调试与检测。
4. 通过调节电位器查看 LED 灯的变化是否符合设计要求。

- **实验现象**

旋转拨盘电位器，LED 灯指示出电位器输出电压的变化。

3.10.1.6 实验思考

1. 如何采集负电压信号？
2. 如何对正弦波或三角波进行模数转换实现计数功能？
3. 请尝试编写程序将采集的信号波形在 TFT 屏幕上显示。

3.10.2 Lab10-2 DAC实验

3.10.2.1 实验介绍

本实验演示了 MSP430F6638 中 DAC12 模块的使用。实验通过配置相关的寄存器，实现了在 P7.6 引脚中产生了一个 0-3.3V 之间的固定模拟电压。

3.10.2.2 实验目的

- 了解 DAC 转换原理
- 学习 MSP430F6638 中 DAC12 的配置使用方法
- 能够用 DAC12 输出 0-3.3V 之间的固定模拟电压

3.10.2.3 实验原理

- **数模转换模块（DAC）**

数模转换是将数字量转换为模拟电量（电流或电压），使输出的模拟电量与输入的数字量成正比。实现这种转换功能的电路叫数模转换器（DAC）。D/A 转换器基本上由 4 个部分组成，即权电阻网络、运算放大器、基准电源和模拟开关。数字量以串行或并行方式输入、存储于数字寄存器中，数字寄存器输出的各位数码，分别控制对应位的模拟电子开关，使数码为 1 的位在位权网络上产生与其权值成正比的电流值，再由求和电路将各种权值相加，即得到数字量对应的模拟量。

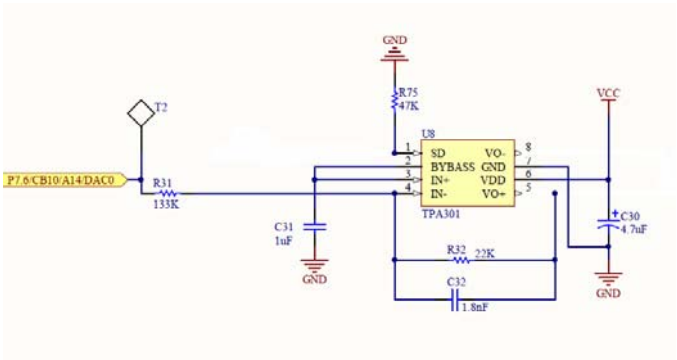

图 3.10.6　DAC 外部配套运放电路原理图

如上图所示，T2为DAC的外接输出端口，对应于P7.6，输出电压最大值为VCC，最小值为GND。

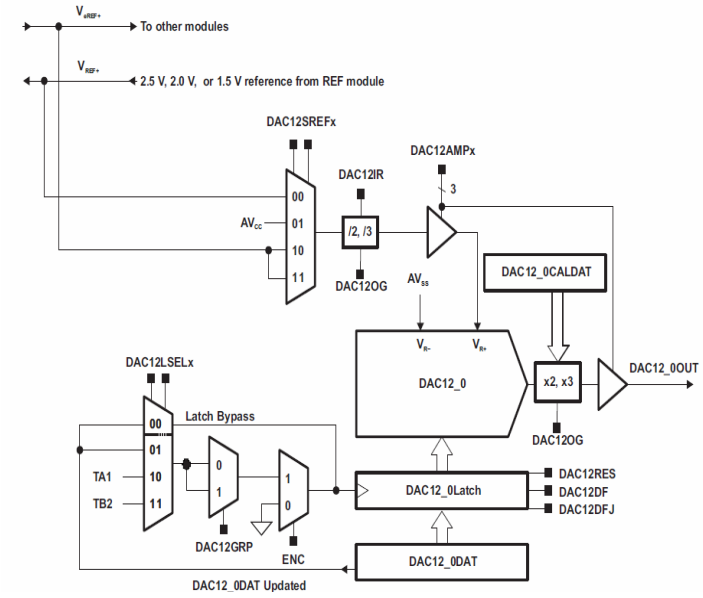

图 3.10.7　DAC12 模块结构框图

DAC12 内部结构如上图所示。DAC 参考电压可有五种选择，分别是：2.5V、2.0V、1.5V、外部电压，AVCC。DAC12_0DAT 中保存的是需要转换的数字电压值，它与 0x0FFF 的的比值应该等于实际输出的模拟电压值与参考电压的比值。另外配置 DAC 转换器的关键寄存器 DAC12_xCTL0 Register。主要的控制位有：

- DAC12OPS：选择输出通道
- DAC12SREFx：选择参考电压
- DAC12ENC：转换使能
- DAC12CALON：启动校验功能

3.10.2.4 程序分析

- **编程思路**

首先通过配置基础的控制寄存器，并按照 DAC 原理，有顺序的对各个位进行赋值，然后再启动 DAC12 转换，并赋予数字电压值。最后让芯片处于低功耗状态。

- **程序流程图**

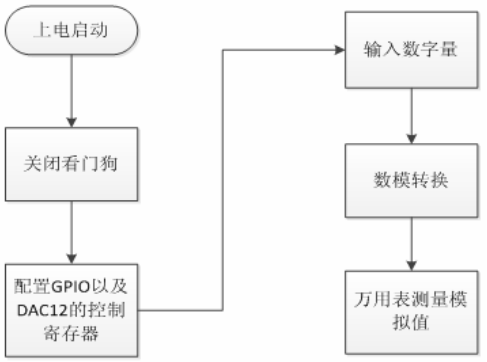

图 3.10.8　程序流程图

- 关键代码分析

实验 DAC 代码
主函数

```
void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;              //关闭看门狗
    P7DIR |= BIT6;                         //设置 P7.6 口为输出口
    P7SEL |= BIT6;                         //使能 P7.6 口第二功能位
    DAC12_0CTL0 |= DAC12IR;
        //设置参考电压满刻度值，使 Vout = Vref×(DAC12_xDAT/4096)
    DAC12_0CTL0 |= DAC12SREF_1;            //设置参考电压为 AVCC
    DAC12_0CTL0  |=  DAC12AMP_5; //设置运算放大器输入输出缓冲器为
                                   中速中电流
    DAC12_0CTL0 |= DAC12CALON;             //启动校验功能
    DAC12_0CTL0 |= DAC12OPS;               //选择第二通道 P7.6
    DAC12_0CTL0 |= DAC12ENC;               //转化使能
    DAC12_0DAT = 0x7FF;                    //输入数据

   __bis_SR_register(LPM4_bits);           //进入低功耗状态
}
```

3.10.2.5 实验步骤与现象

- 实验步骤

1. 构思好编程思路后，画出流程图；
2. 根据流程图在工程的主函数中完成代码编写；
3. 调试编译程序，完善代码，解决问题；
4. 将程序烧入开发板中测试效果，用万用表测量图 3.10.7 中的 T2 测试点（P7.6 端口）。

- 实验现象

图 3.10.9　实验结果（左：DAT=0x7FF，Vout=1.65V；右：DAT=0xFFF，Vout=3.29V）

3.10.2.6 实验思考

1. 按解码网络结构区分，DAC 分为几种？
2. 能否用 DAC 实现多种波形的模拟量输出？

3.11 Flash模块

MSP430F6638 芯片中集成的 256KB FLASH 存储器可以按字节（byte）、字（2-bytes）、双字（4-bytes）寻址或编程，控制器集成了编程与擦除功能。FLASH 模块包含 3 个寄存器、一个定时器以及一个产生编程与擦写电压的电压生成器。

3.11.1 Lab11-1 Flash实验

3.11.1.1 实验介绍

本次实验应用了 MSP430F6638 的 Flash 读写功能，实现了在片内闪存地址中写入数据，通过这些数据对 GPIO 管脚寄存器进行了配置，并点亮了对应的 LED 灯。

311.1.2 实验目的

- 学习 Flash 的基本知识
- 熟练掌握 MSP430F6638 中 Flash 的读写操作

3.11.1.3 实验原理

Flash 闪存是非易失存储器，全名叫 Flash EEPROM Memory，可以对称为块的存储器单元块进行擦写和再编程，它结合了 ROM 和 RAM 的长处，不仅具备电子可擦除可编程（EEPROM）的性能，还可以快速读取数据（NVRAM 的优势），使数据不会因为断电而丢失。目前 Flash 主要有两种 NORFlash 和 NANDFlash。NORFlash 的读取与 SDRAM 的读取一样，用户可以直接运行装载在 NORFLASH 里面的代码。NANDFlash 没有采取内存的随机读取技术，它的读取是以一次读取一块的形式来进行的，通常是一次读取 512 个字节。

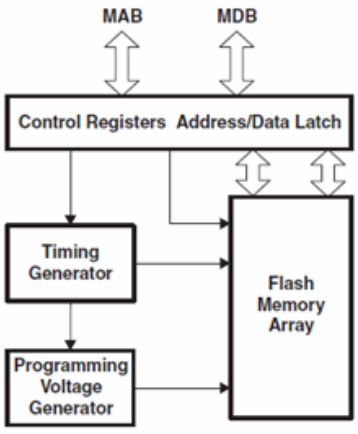

图 3.11.1 Flash 模块结构框图

控制器包括三个寄存器，一个时序发生器及一个提供编程、擦除电压的电压发生器。MSP430F6638 中的 Flash 存储器的特点有：产生内部编程电压；可位、字节、字编程，可以单个操作，也可以连续多个操作；超低功耗操作；支持段擦除和多段模块擦除。

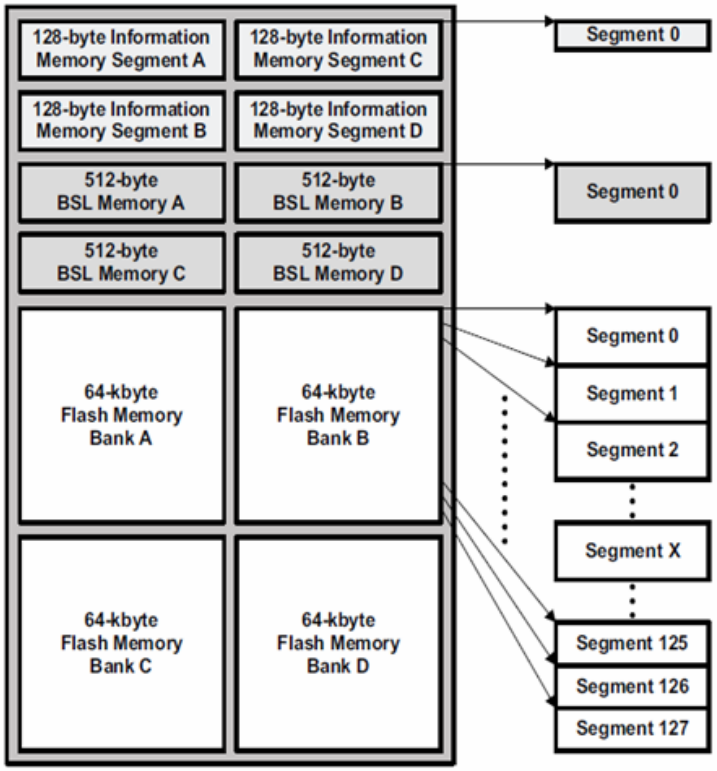

图 3.11.2　Flash 模块区段划分

Flash 存储器被分割成两部分：主存储器和信息存储器，两者在操作上区别不大；但主存储器允许整块(Bank)擦除，且整块擦除时可以继续执行其它块上的程序。MSP430F6638 中的 Flash 可以进行单个字节、字的写入，也可以进行连续多个字、字节的写入操作，但是最小的擦除单位是段。如上图所示，MSP430F6638 Flash 的信息存储器由 4 个 128 字节的段组成；主存储器由 4 个 64K 的块组成，每块又分为 128 段。

3.11.1.4 程序分析

- **编程思路**

对 Flash 的操作需要按照固定的顺序方式来执行，并且执行的结果存储在 Flash 中，为了能体现实验的可视性效果，本实验采取点亮 LED 灯的方法。把控制 LED 灯的 GPIO 寄存器地址写入 Flash，然后再读取该数据，把数据强制转换为地址，并对其赋值。

- **程序流程图**

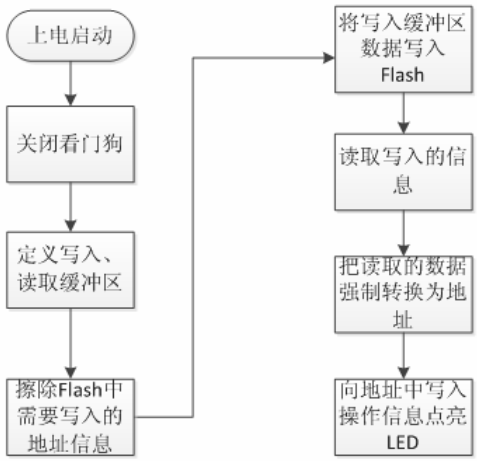

图 3.11.3　程序流程图

- 关键代码分析

实验 Flash 代码

预编译定义

```
typedef unsigned intuint;                    //方便起见用 uint 表示 16 位无符号整型
typedef unsigned char uchar;                 //方便起见用 uchar 表示 8 位无符号字符型

#define LEN 6                                //宏定义缓冲区大小
#define ADR 0x1880                           //宏定义写入 Flash 的起始地址，位于 Segment C
#define BYTE 0                               //单字节写入标志
#define WORD 1                               //双字节写入标志
#define DELETE 0                             //清除值
#define P4_OUT 0x0223                        //宏定义相应寄存器的地址
#define P4_DIR 0x0225
#define P5_OUT 0X0242
#define P5_DIR 0x0244
#define P8_OUT 0x0263
#define P8_DIR 0x0265
```

主函数

```
void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;                                    //关闭看门狗
    //ucharBSetBuf[LEN] = {0};                  //单个字节写入缓冲区，例子中未用到，此处注释
    uintWSetBuf[LEN] = {P4_DIR,P5_DIR,P8_DIR,P4_OUT,P5_OUT,P8_OUT};
                                                                 //双字节写入缓冲区
    uintRxBuf[LEN] = {0};                                        //数据读取缓冲区
  FlashErase(ADR,WORD,LEN);                                      //擦除写入 Flash 中的数据
    FlashWriteWord(ADR,WSetBuf,LEN);                             //双字节写入函数
    FlashRead(ADR,RxBuf,WORD,LEN);                               //数据读取函数
/*取得缓冲区的数据作为地址并对其赋值，此处通过 Flash 中的数据间接地设置连接五个 LED 灯
的 GPIO 管脚方向为输出方向，且输出高电平*/
    *((uchar *)RxBuf[0]) |= (uchar)(BIT7 + BIT6 + BIT5);
    *((uchar *)RxBuf[1]) |= (uchar)BIT7;
    *((uchar *)RxBuf[2]) |= (uchar)BIT0;
    *((uchar *)RxBuf[3]) |= (uchar)(BIT7 + BIT6 + BIT5);
    *((uchar *)RxBuf[4]) |= (uchar)BIT7;
    *((uchar *)RxBuf[5]) |= (uchar)BIT0;
    FlashErase(ADR,WORD,LEN);                                    //擦除写入 Flash 中的数据
    __bis_SR_register(LPM3_bits);                                //进入低功耗状态
```

```
}
```

双字节写入函数

```
void FlashWriteWord(uintaddress,uint *word,uint count)
{
    FCTL3 = FWKEY;                          //Flash 解锁
    while(FCTL3 & BUSY);                    //等待忙
    FCTL1 = FWKEY + WRT;                    //选择写入模式
    while(count--)
    {
        *((uint *)address) = *(word++);     //把双字节缓冲区的数据写入对应的地址
        address = address + 2;              //写入的数据地址按顺序累加
    }
    FCTL1 = FWKEY;                          //清除写入模式
    FCTL3 = FWKEY + LOCK;                   //Flash 上锁
    while(FCTL3 & BUSY);                    //等待忙
}
```

数据读取函数

```
void FlashRead(uintaddress,uint *buf,uintmode,uint length)
{
    if(mode)                                //判断读取模式：0 表示单字节，1 表示双字节
    {
        while(length--)                     //判断读取数据的个数
        {
            *(buf++) = *((uint *)address);
                                            //把读取的双字节数据送入数据读取缓冲区
            address = address + 2;
        }
    }
    else
    {
        while(length--)
        {
            *((uchar *)buf++) = *((uchar *)address);
                                            //把读取的单字节数据送入数据读取缓冲区
            address = address + 2;
        }
    }

}
```

Flash 擦除函数

```
voidFlashErase(uintaddress,uintmode,uint length)
{
FCTL3 = FWKEY;
while(FCTL3 & BUSY);
FCTL1 = FWKEY + ERASE;
if(mode)                                    //判断数据模式
{
 while(length--)                            //判断擦除数据个数
 {
    *((uint *)address) = DELETE;            //将对应地址上的双字节数据擦除
    address = address + 2;
 }
}
else
{
 while(length--)
```

```
    {
        *((uchar *)address) = DELETE;                    //将对应地址上的单字节数据擦除
        address = address + 2;
    }
  }
  FCTL1 = FWKEY;
  FCTL3 = FWKEY + LOCK;
  while(FCTL3 & BUSY);
  }
```

3.11.1.5 实验步骤与现象

- **实验步骤**

1. 实现控制代码的设计，画出流程图；
2. 完成编码，并将代码烧录到发板中；
3. 观察 LED 的状态变化，验证结果。

- **实验现象**

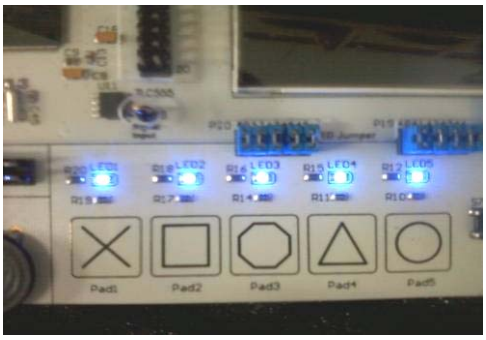

图 3.11.4　实验结果（5 个 LED 灯点亮）

3.11.1.6 实验思考

1. Flash 分为哪几种？它们之间有什么区别？
2. 请尝试实现将 Flash Segment C 中写入数据，并把数据复制到 Segment D 指定地址中。
3. 请设计一个结果更为直观的 Flash 操作实验。

第 4 章 综合实验

上一章我们介绍了系统的各个主要模块并且安排了各个模块的基础实验项目，通过那些实验项目我们能对这套系统有一个全面的了解与掌握。这一章我们设计一些了综合实验项目，这些综合实验项目中往往涉及多个模块的共同使用。MSP430F6638 教学开发系统有着多种多样的外设模块，因此我们也可以开动脑筋，设计出各种实用而又有趣的系统。

4.1 Pro_01 音频播放（外置声卡）实验

4.1.1 实验目的

- 了解 MicroSD 卡的操作方法；
- 了解 wav 文件的结构和播放原理；
- 了解 USB 存储器的访问接口。

4.1.2 实验内容

编写程序使得可以通过 USB 接口从 PC 机上拷贝 wav 音频文件到 SD 卡中，再对这些音频文件进行播放。

4.1.3 实验原理

4.1.3.1 MicroSD卡操作

MSP430F6638 对 MicroSD 卡的操作通过 MicroSD 卡的 SPI/MMC 模式进行。这个模式支持四线串行接口（时钟、串行输入，串行输出，芯片选择），指令与数据在同一通道上传输。这一接口虽然传输速率较慢，但兼容于单片机的 SPI 接口，比较适用于单片机读写 MicroSD 卡。

4.1.3.2 USB存储器接口

USB 大容量存储设备类（The USB mass storage device class）是一种计算机和移动设备之间的传输协议，通过实现这一协议，可以使得 PC 机将 MSP430F6638 当做 U 盘进行访问。在本例中，可以使用 MSP430F6638 将 PC 机一侧的 USB MSC 协议与 MicroSD 卡一侧的 MMC 协议连接起来，从而实现 PC 机直接对 SD 卡进行文件的存取。

TI 已提供有适用于 MSP430F6638 的 USB 模块的 USB MSC 驱动程序，使用时对这一驱

动程序进行正确的初始化并将其读写请求对应到 MMC 程序的读写操作即可。

4.1.3.3 WAV文件播放

WAV 文件通常使用 PCM 无压缩编码（直接来源于 ADC 的信号，也可以直接送到 DAC 输出），其编解码计算量和占用内存空间均较小，在 MSP430F6638 的承受范围内，无需专用解码芯片。播放 WAV 文件时需要使用 FAT 文件系统程序从 SD 卡中读出待播放的 wav 文件，解析其格式并将解码得到的数据按照其采样间隔送至 DAC 进行输出。

WAV 文件是 RIFF 格式的一种，其开头为 RIFF 文件头，格式为

```
struct RIFF_HEADER
{
charszRiffID[4]; // 'R','I','F','F'
uint32_tdwRiffSize;
charszRiffFormat[4]; // 'W','A','V','E'
} audio_riff_header;
```

其中 dwRiffSize 是去掉 szRiffID 和 dwRiffSize 之后的整个文件长度，也就是文件长度减 8。

RIFF 文件头之后则是若干 chunk，每个 chunk 的开头都是 4 个字节的 chunk id，然后是 4 个字节的 chunk size，size 同样是该 chunk 的总长减去 8. 一般来说，RIFF 文件头后应该是 format chunk，chunk 体可能是 16 或者 18 字节，定义如下

```
struct WAV_FMT_CHUNK
{
//RIFF_CHUNK_HEADER chHeader; // 'f','m','t',' '
uint16_twFormatTag;
uint16_twChannels;
uint32_tdwSamplesPerSec;
uint32_tdwAvgBytesPerSec;
uint16_twBlockAlign;
uint16_twBitsPerSample;
uint16_twExtInfo;
} audio_wav_fmt;
```

其中 wExtInfo 就是可选区段，是否存在可由 size 判断。

Format chunk 后可能有 Fact chunk，这一段对于本例的播放没有用处，可以直接跳过。

文件的最后一个 chunk 应当是 data chunk，在其 chunk 头后保存有音频数据，数据格式如下：

- 对于 8 位单声道，每个样本数据由 8 位(bit)表示；
- 对于 8 位立体声，每个声道的数据由一个 8 位(bit)数据表示，且第一个 8 位(bit)数据表示 0 声道(左)数据，紧随其后的 8 位(bit)数据表示 1 声道(右)数据；
- 对于 16 位单声道，每个样本数据由 16 位(bit)表示；其中低字节存放高位，高字节存放低位；
- 对于 16 位立体声，每个声道的数据由一个 16 位(bit)数据表示，且第一个 16 位(bit)数据表示 0 声道(左)数据，紧随其后的 16 位(bit)数据表示 1 声道(右)数据。

4.1.4 实验步骤

- 将程序下载到 MSP430F6638 中，并将 SD 卡插入实验板上 SD 卡插槽；
- 运行程序，使用主板右侧 USB 接口（J1）从 PC 机传输 wav 文件（支持 8 位量化音频文件）到 SD 卡上，并使用 MSP430F6638 进行播放（播放时注意 P25 跳线块位置，使用板载扬声器时跳线块应当在左侧，使用耳机时跳线块应当在右侧）。

4.1.5 实验现象

MSP430F6638 读取 SD 卡中的 WAV 文件并播放（对于例程，播放中 S5 键可跳过当前文件，S7 键设定最小音量，S3 键设定最大音量）；在 USB 口接至电脑时，MSP430F6638 将 SD 卡模拟为 U 盘，PC 机可以向 SD 卡传输文件。

4.1.6 关键代码分析

4.1.6.1 音频播放模块(dr_audio)

音频播放模块提供对 8 或 16bit、单声道或双声道、无压缩 WAV 格式进行播放的功能，采样率不作软件限制但实际可以正常工作的最高采样率在 16KHz 左右。播放模块以流的形式接受待播放的数据，可以很容易的与串口等数据源连接。

播放模块初始化后处于等待设定数据源状态(程序中写作 WAIT_STREAM)，之后由主程序设定了数据源后进入 READ_RIFF_HEADER 状态(准备读取 WAV 文件的 RIFF 头)。在整个文件读取与播放过程中，若出现文件格式错误或者不支持的情况，都会进入 THROW_DATA 状态，抛弃之后输入的所有数据，直到调用 audio_DecoderReset 函数或者超过 0.5s 没有新数据使得解码程序复位为止。

进入 READ_RIFF_HEADER 状态后，程序会等待输入缓冲区中的数据达到 RIFF_HEADER 的长度(使用时注意处理文件头的过程不受 0.5s 超时约束)，然后一次读入整个文件头。之后对 RIFF 和 WAVE 标志进行检查，检查通过则进入 READ_CHUNK_HEADER 状态，否则进入抛弃数据状态。

READ_CHUNK_HEADER 状态下程序会读入 CHUNK 的 ID 和 size，若 ID 为 data，则会对文件格式进行检查，通过后进入 READ_DATA 状态开始音频播放；若为其它 ID，则进入 READ_CHUNK 状态读入 CHUNK 的其余部分。

READ_CHUNK 状态下会判断当前正在读取的 CHUNK 是不是 fmt，如果是的话则按照读入的数据设定当前的播放格式；否则直接抛弃这一 CHUNK(其它的 CHUNK 对这一程序没有解析的必要)。

READ_DATA 状态下则会按照文件中的采样率要求在定时中断中读取数据并播放，直到文件结束或超过 0.5s 没有新数据或外部复位播放模块为止。

4.1.6.2 SD卡模块(dr_sdcard)

SD 卡模块包括 SD 卡驱动、盘级接口和 FAT 文件系统，为 USB MSC 和程序中的文件操作提供支持。

HAL_SDCard 中包含有 SD 卡驱动，提供对 SD 卡通讯的配置和数据收发功能，用于支持盘级接口的运行。

mmc.c 中利用 SD 卡驱动、按照 MMC in SPI 的标准实现了盘级接口。盘级接口用于支持 USB MSC 和 FAT 文件系统的运行。

ff.c 中实现了 FAT 文件系统，所有应用程序模块对 SD 卡的访问都是通过 FAT 文件系统进行的。

4.1.6.3 USB MSC模块(dr_usbmsc)

USB MSC(USB mass storage device class)模块为 PC 机提供了一个 U 盘的操作接口，使得 PC 机可以像操作 U 盘一样对 SD 卡进行操作。这一接口对 SD 卡的操作是通过 SD 卡模块中的盘级接口部分实现的。

4.1.7 问题思考

如何协调 TFT 屏幕刷新和音频文件播放程序，使得音频文件播放能获得尽量多的计算时间？

4.2 Pro_02 图片显示实验

4.2.1 实验目的

- 了解 MicroSD 卡的操作方法
- 了解 bmp 文件的结构和显示方法
- 了解 USB 存储器的访问接口

4.2.2 实验内容

编写程序使得可以通过 USB 接口从 PC 机上拷贝无压缩 bmp 文件到 SD 卡中，再将这些文件在 TFT 屏幕上显示出来。

4.2.3 实验原理

TFT 屏幕控制参见 3.7.2 章节内容，SD 卡操作、USB MSC 接口参见 3.7.1 及 4.1 的相关章节。

BMP 是 Window 操作系统中的标准图像文件格式，它采用位映射存储格式，除了图像色彩深度可选以外，一般不采用其他压缩方式，因此占用空间较大，但显示较为容易。典型的 BMP 图像文件由位图文件头数据结构、位图信息数据结构、调色板和位图数据四部分组成，其中调色板不是必须的结构，24 位图就不存在调色板，为简便起见，以下主要说明 24 位图的结构和显示方法。

位图文件头包括 14 个字节，定义如下：

```
typedefstruct tagBITMAPFILEHEADER
{
WORDbfType; // 位图文件的类型，必须为BM(1-2字节)
DWORDbfSize; // 位图文件的大小，以字节为单位（3-6字节）
WORDbfReserved1; // 位图文件保留字，必须为0(7-8字节)
WORDbfReserved2; // 位图文件保留字，必须为0(9-10字节)
DWORDbfOffBits; // 位图数据的起始位置，以相对于位图（11-14字节）
// 文件头的偏移量表示，以字节为单位
    } BITMAPFILEHEADER;
```

位图信息包括 40 个字节，定义如下：

```
typedefstruct tagBITMAPINFOHEADER{
DWORDbiSize; // 本结构所占用字节数（15-18字节）
LONGbiWidth; // 位图的宽度，以像素为单位（19-22字节）
LONGbiHeight; // 位图的高度，以像素为单位（23-26字节）
```

```
WORDbiPlanes; // 目标设备的级别，必须为1(27-28字节)
WORDbiBitCount;// 每个像素所需的位数，必须是1（双色），(29-30字节)
// 4(16色)，8(256色)或24（真彩色）之一
DWORDbiCompression; // 位图压缩类型，必须是 0（不压缩），(31-34字节)
// 1(BI_RLE8压缩类型)或2(BI_RLE4压缩类型)之一
DWORDbiSizeImage; // 位图的大小，以字节为单位（35-38字节）
LONGbiXPelsPerMeter; // 位图水平分辨率，每米像素数（39-42字节）
LONGbiYPelsPerMeter; // 位图垂直分辨率，每米像素数（43-46字节）
DWORDbiClrUsed;// 位图实际使用的颜色表中的颜色数（47-50字节）
DWORDbiClrImportant;// 位图显示过程中重要的颜色数（51-54字节）
    } BITMAPINFOHEADER;
```

这两个数据结构是固定的，直接以二进制方式从文件中读出即可。读出后可检查各项数据，若发现格式错误或不支持则跳过后续读取与显示过程。

考虑 24 位图没有颜色表，则位图文件头和位图信息后直接就是具体的位图数据。位图数据默认是从左到右、从下到上的顺序（若位图高度为负数，则从上到下，但一般不使用这种格式）。对于 24 位图，每个像素占 3 个字节，每个字节依次表示蓝色、绿色和红色分量。位图的每一行按 4 字节对齐，若一行的像素数据不满 4 字节的倍数，则用 0 填充至达到 4 字节的倍数。读取到的像素数据按照 TFT 屏所需格式编码后发送给 TFT 屏即可显示出来。

4.2.4 实验步骤

- 移除 TFT 屏幕上方 P10 处左侧三个跳线（避免给 TFT 屏的控制信号驱动段式液晶屏出现闪动），保留最右侧的一个跳线；
- 将程序下载到 MSP430F6638 中，并将 SD 卡插入实验板的 SD 卡插槽；
- 运行程序，使用 USB 从 PC 机传输 bmp 文件（支持 24 位色 BMP 格式图片）到 SD 卡上，并使用 MSP430F6638 进行显示。

4.2.5 实验现象

MSP430F6638 读取 SD 卡中的 BMP 文件并显示在 TFT 屏上；另外在 USB 口接至电脑时，MSP430F6638 将 SD 卡模拟为 U 盘，PC 机可以向 SD 卡传输文件。对于例程，可以显示小于 320*240 的无压缩 24 位位图；对于不能显示的，则给出文件名并提示格式不支持；按下 S5 键切换至下一图片。

4.2.5.1 BMP解析模块(app_bmp)

对 SD 卡上指定文件名的文件按 BMP 进行解析并显示，依赖于 SD 卡模块和 TFT 模块。具体解析过程见注释与 BMP 文件格式。

4.2.6 问题思考

为何即使位图信息结构是固定长度的，仍然设有 size 部分？这一设计有什么可能的好处？

4.3 Pro_03 录音与回放实验

4.3.1 实验目的

- 了解 MicroSD 卡的操作方法
- 了解 wav 文件的结构、写入方法和播放方法
- 了解 USB 存储器的访问接口

4.3.2 实验内容

编写程序提供 wav 文件的录制和 wav 文件的播放的两个功能，另外在 USB 口连接至电脑时可以将 SD 卡模拟为 U 盘。

4.3.2.1 音频录制模式

在音频文件录制模式下，首先会询问录制到的文件名，文件名格式固定为“REC000.WAV”。其中编号一项可以改变，范围在 0~255，选择编号时程序会提示是否已经存在这一名称的文件；也可以按 S6 自动找到一个未使用的文件。

选定文件名后按下 S5 进入录制模式，并且可以用 S4 键切换是否启动监听模式（可以使用耳机监听到实时录音效果），并显示已录制时间和文件大小等信息。录制结束时按下 S5 即对文件进行保存并退出。退出录制模式后进入播放模式。

4.3.2.2 音频录制模式

在音频播放模式下，需要可以读取 SD 卡内的音频文件并进行播放，同时可以使用按键控制音量、跳至下一个文件和退出音频播放；另外播放时需要可以在 TFT 屏幕上显示出当前播放文件的文件名、当前播放时间、文件总时间、采样率等信息。退出音频播放模式后进入录制模式。

4.3.3 实验原理

TFT 屏幕控制参见 3.7.2 章节内容，SD 卡操作、USB MSC 接口及 WAV 文件格式参见 3.7.1 及 4.1 的相关章节。

4.3.3.1 WAV文件录制

由于录制时 WAV 文件格式固定，所以 WAV 文件录制实际较 WAV 文件播放更为简单。

录制时可以使用固定的文件头，其中文件长度和数据区长度段可以在一开始用 0 填充，然后将 AD 得到的数据顺序写入文件，录制结束后再按照写入的数据长度改写文件头的文件长度和数据区长度部分。

4.3.4 实验步骤

- 编译并下载程序（注意编译器选项设置，参见 4.3.6 关键代码分析一节）；
- 将 SD 卡插入实验板上的 SD 卡插槽后复位主板（移除 P2 处 TEST、RES 两个跳线后按 RESET 键）；
- 在音频录制模式下录制一段音频（对于例程，开启监听模式时不可使用实验板的扬声器，避免出现自激振荡）；
- 在音频播放模式下将录制的音频播放出来。

4.3.5 实验现象

MSP430F6638 可以录制并播放音频文件。

4.3.6 关键代码分析

对于例程，其使用的 SD 卡模块、USB MSC 模块和 TFT 模块与前述工程中使用的相同。本程序代码较长，需要将编译器 code_model 和 data_model 设置为 large 模式（见图 4.3.1），printf 库函数模式要选为“no float” （见图 4.3.2）。

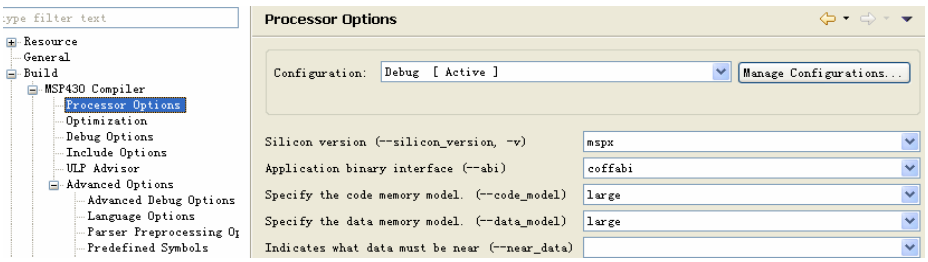

图 4.3.1 编译器模式选择

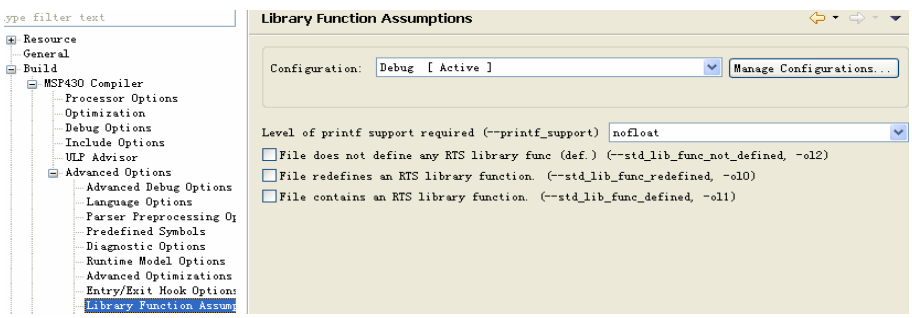

图 4.3.2　库函数模式选择

4.3.6.1　音频输入输出驱动(dr_audio_io)

dr_audio_io 模块封装了音频输入输出过程中与硬件有关的操作，将对音频的输入和输出封装为对两个队列的读取和插入操作，同时提供设定采样率的接口。目前的实现中输出和输出的采样率统一设定、总是相同。

4.3.6.2　轻触开关与LED灯驱动(dr_buttons_and_leds)

dr_buttons_and_leds 提供了比较方便的访问轻触开关按键和 LED 灯的接口。可以按编号直接控制某个 LED 的开关；按编号判断某个按键是否被按下；阻塞或非阻塞的获取被按下的按键编号。按键模块不支持组合键。

其中按键模块按照函数被调用的时刻按键 IO 口的状态来对按键是否被按下进行判断。可以在头文件中通过对 REF_CLOCK 的定义指定一个 int32 型的变量为参考时钟，并定义 BUTTON_DEADZONE_PERIOD 以指定一个死区时间，使得模块在检测到一次按键按下后自动忽略接下来一段时间内的按键按下。

4.3.6.3　音频录制子程序(app_audio_recorder)

音频录制子程序可以以 8 位、单声道、8K 采样率的格式进行音频的录制。进行音频录制时会首先询问录制的文件名，之后建立音频文件并立刻写入文件头，之后不断读取音频输入队列中的数据并写入文件中，当用户要求停止录音后按照写入的数据大小计算文件头中的数据长度部分并填入文件头中。具体操作流程参考源代码。

4.3.6.4　音频播放子程序(app_audio_player)

音频播放子程序是对 8_sdcard_and_audio 中音频播放程序的重新组织。这一子程序将硬件音频输出剥离出去，同时包含了从文件系统读取文件的部分，使得对 WAV 文件的解释过程在 app_PlayWavFile 函数中体现为一个整体，方便作为参考。

4.3.7 问题思考

录制的音频的质量可能受限于哪些因素？

4.4 Pro_04 直流电机调速实验

4.4.1 实验目的

- 熟悉使用 MSP430F6638 的模数转换功能
- 熟悉使用 MSP430F6638 定时器的 PWM 输出功能
- 熟悉使用 MSP430F6638 的段式液晶驱动功能

4.4.2 实验内容

- 通过 AD 采样获得电位器中心抽头电压值；
- 将电位器电压值转换为控制直流电机的 PWM 信号；
- 对直流电机上安装的光电门产生的脉冲进行测量，得到电机转速信息；
- 将电机转速显示在液晶屏上。

4.4.3 实验原理

- 电位器的 ADC 采样部分参考 3.10.1；
- 直流电机控制部分参考 3.6.1；
- 段式液晶的显示部分参考 3.2.1；

4.4.3.1 PWM输出

将电机小板与 P15 或 P17 端口连接时(连接时注意方向!)可以将 MSP430F6638 的 P1.6、P1.7 和 P1.0 端口与电机驱动芯片 DRV8833 的 IN1、IN2 和 nSLEEP 端口连接，从而使用 MSP430F6638 控制 DRV8833 的输出。

xIN1	xIN2	xOUT1	xOUT2	功能
0	0	Z	Z	惯性旋转
0	1	L	H	反转
1	0	H	L	正转
1	1	L	L	制动

表 4.4.1 DRV8833H 桥逻辑表

DRV8833 的真值表如上表所示（nSLEEP 为高电平时；nSLEEP 为低电平时所有输出端为高阻态）。当 IN1 和 IN2 都处于低电平时，两个输出端都处于高阻态，此时电机绕组上的电流流经 DRV8833 中 H 桥的反并联二极管组成回路，因此加在电机绕组上的电压方向必然与电流方向相反，会使得电机绕组电流快速下降；当 IN1 与 IN2 一个处于高电平一个处于

低电平时，两个输出端的电平也与对应输入端的一致，从而给电机绕组加正向或者反向的5V 电压，驱动其正向或反向转动；当 IN1 和 IN2 都处于高电平时，两个输出端都输出低电平，从而将电机绕组短路，使得电机绕组电流在其内阻的作用下慢速下降。因此，可以通过控制两个输入端的电平，来控制加在电机绕组上的电压，从而控制电机的启停与旋转方向。此外，电机是一个具有惯性的环节，因此其输入可以用一系列面积相等的短脉冲来近似等效，通过调节短脉冲的占空比，就可以获得与施加一个幅值变化的模拟信号相同的效果，这就是脉宽调制（PWM）的基本原理。

利用 MSP430F6638 定时器的比较模式就可以输出特定占空比的脉冲，从而可以控制加在电机绕组两端的等效电压幅值，达到调节电机转速的效果。

4.4.3.2 光耦测速

电机轴上安装有一个等间隔开有 4 个槽的圆盘，并安装有一个光耦，电机旋转的过程中圆盘上的槽经过光耦时会使红外光透过，从而在光耦的的输出上产生一段低电平。通过使用 MSP430F6638 的 IO 口的中断功能，可以捕捉到光耦输出的跳变，再通过测量跳变发生的时间，就可以计算得到电机的当前转速。

4.4.4 实验步骤

- 安装 TFT 屏幕上方 P10 处左侧的三个跳线；
- 用 20-pin 连接线将电机板连接到主板 P15 或 P17 端口（注意连接方向，连接线红线对准端口“1”管脚）；
- 运行程序，转动电位器，观察电机转速的变化和液晶屏上的转速显示。

4.4.5 实验现象

转动 R13 拨盘电位器，可以看到电机转速和液晶屏上的转速显示随之发生变化。

4.4.6 关键代码分析

4.4.6.1 整体结构

- 与液晶屏显示有关的函数均在 dr_lcdseg.h/.c 中进行声明与实现；
- main 函数 while 循环中完成转速显示的更新与 PWM 输出电压的更新；
- 使用 PWM 所用的定时器产生 0.1ms 定时中断来作为转速测量的时间参考；
- 测量转速时采用测量光电门上升下降沿间的间隔的方式进行测量，详见之后的说明。

4.4.6.2 转速测量的补充说明

实验板上的光耦在输出跳变的过程中输出会有抖动，导致单片机在下降沿附近时也可以捕捉到上升沿，故实际实现中对上升沿和下降沿都进行测量，并通过时间间隔确定捕捉到的是不是干扰信号。

另外由于电机上的测速盘不很精密，为了使测速精度能达到 1rpm，使用了对多次测量数据进行平均的方式。可以通过计算算出各转速下获得 1rpm 精度需要对多少次脉冲进行平均。但如果都按照最高转速下的测量次数进行平均，会导致低速旋转时转速测量反应过慢，因此程序中采用了先用最近的一次脉冲的时间估算转速，再决定具体使用多少次平均来计算精确转速的方式。

4.4.7 问题思考

- PWM 输出为何要使用保持一个端子为高电平，对另一端子进行 PWM 的方式？保持一个端子为低电平并对另一个端子进行 PWM 是否可以？需要什么改动？(这一问题与 DRV8833 的设计有关)
- 在当前的硬件下，还有可能使用什么方法提高转速测量的精度？
- 如果希望使用电位器来直接控制电机转速(即电位器位置对应到某个电机转速而不是输出电压，电机转速可以在负载改变时保持不变，负载可以用纸片摩擦测速盘的方式模拟)，需要如何改进程序？

4.5 Pro_05 步进电机细分驱动实验

4.5.1 实验目的

- 了解使用 PWM 方法对步进电机进行简单细分驱动的方法。

4.5.2 实验内容

- 编写程序使用 PWM 方法实现对步进电机的细分调速驱动，使得步进电机一步的角度下降到 9°以下（电角度 45°以下），同时可以使用电位器调节步进电机连续旋转的旋转速度。
- 将步进电机的转速显示在段式液晶上。

4.5.3 实验原理

实验所用的步进电机的结构示意图如图 4.5.1 所示。电机中垂直布置了 AB 两个绕组，当给绕组通电时，就可以产生对应方向的磁场矢量。通过分别给 A、B 绕组加正向电压、不加电压或者加反向电压，就可以在对应绕组上产生对应方向的电流，从而产生 8 个不同方向的磁场矢量，从而控制电机转子旋转到对应磁场矢量的位置。

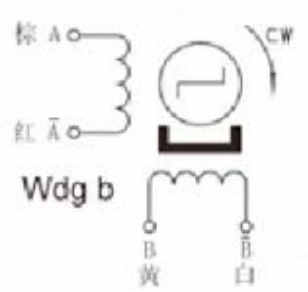

图 4.5.1 步进电机结构图

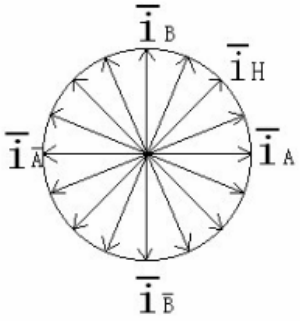

图 4.5.2 磁场矢量示意图

如果可以对 A、B 相的电流幅值大小进行控制，就可以进一步细化调整磁场矢量的方向，从而控制转子转到需要的角度。如图 4.5.2 所示。但本实验板中没有安装电流传感器，所以无法对电流进行直接控制。但对于步进电机，可以认为在稳态下电流和电压成正比，故可以用 PWM 方法对绕组电压进行控制，从而近似得到控制电流的效果。

需要注意的是，由于实验板上步进电机所使用的驱动信号只有一端可以 PWM，而驱动芯片 DRV8833 在两个输入端都为低电平时输出是高阻态而非低电平，故 PWM 的过程中需要对这一情况进行补偿（当一个线圈的驱动信号的固定端输出 0 时，PWM 端输出 0%-50%占空比均对应等效的 0V 电压）。

4.5.4 实验步骤

- 安装 TFT 屏幕上方 P10 处左侧的三个跳线；
- 用 20-pin 连接线将电机板连接到主板 P15 或 P17 端口（注意连接方向，连接线红线对准端口“1”管脚）；
- 运行程序，使用电位器调节旋转速度，使用轻触开关控制电机旋转（对于例程，按下左键和右键分别是逆时针与顺时针旋转 1°，按下上键和下键分别是顺时针和逆时针旋转 1000 圈，按下中键停止）。

4.5.5 实验现象

对于例程，步进电机连续旋转时的转速由可变电阻设置，并显示在液晶屏上，单位为 RPM；按下左键和右键分别是逆时针与顺时针旋转 1°，按下主板右下角蓝色按键 S4 和 S6 分别是顺时针和逆时针旋转 1000 圈，按下中间的蓝色键 S5 停止。但由于没有电流反馈或非线性校正，细分会有一定的不均匀。

4.5.6 关键代码分析

ADC 与按键部分在 main.c 中；驱动电机操作部分在 dr_step_motor.c 中。

4.5.7 问题思考

有哪些可能的方法可以提高步进电机的精度？

4.6 Pro_06　USB与串口实验

4.6.1 实验目的

- 了解 MSP430F6638 的 USB 模块的使用方法；
- 了解 MSP430F6638 通用串行通讯模块异步串行通讯功能的使用方法。

4.6.2 实验内容

编写程序实现以下功能：利用 MSP430F6638 的 USB 库实现 USB 虚拟串口功能、键盘功能、鼠标功能。

4.6.3 实验原理

MSP430F6638 中安装有 USB 模块，通过操作这一模块的相关寄存器，可以通过 USB 接口与 PC 机实现交互。另外 TI 已提供了实现常用功能（包括虚拟串口、U 盘和人机接口设备）的驱动程序，这些常用功能可以直接通过配置和调用相关的库来实现。

4.6.4 实验步骤

- 将程序（可使用例程 6_usb_and_uart）下载到 MSP430F6638 中；
- 使用 USB 线将实验板右侧 USB 端口（J1）与 PC 机相连（对于例程，需要 PC 机安装 Windows Vista 或更高版本的操作系统）；
- 将实验板左侧串口接口与 TTL 电平串口设备相连（若无串口设备也可直接将收发线相连）；
- 尝试使用 PC 机串口调试工具打开虚拟串口，并进行通讯；
- 使用实验板上的轻触开关测试 USB 键盘和鼠标功能（对于例程，按键定义见实验现象部分）。

4.6.5 实验现象

虚拟串口部分完成 USB 到 RS232 端口的转换，可以将从 USB 主机侧传来的数据发送到 RS232 口上、将 RS232 口上传来的数据发到主机侧。支持主机侧设定波特率，但只可以是 8 位数据、1 位停止位、无校验模式。当 USB 虚拟串口从主机收到数据时 LED1 亮起，

向 RS232 发出数据时 LED2 亮起，从 RS232 收到数据时 LED3 亮起，RS232 收到的数据送入 USB 发送缓冲时 LED4 亮起，USB 开始发送时 LED5 亮起。

键盘与鼠标部分可以用开发板上的按键模拟键盘和鼠标操作。触摸 Pad1 同时按下 S3~S7 时模拟键盘操作；不触摸 Pad1 时按下 S3~S7 时模拟鼠标操作。

4.6.6 关键代码分析

在 CCS 下编译该项目时需要 msp430USB.cmd 文件，以提供 USB 模块的链接信息。

程序的主要逻辑写在 main.c 中，其中虚拟串口部分在 processUsbToUart 函数，键盘和鼠标部分在 processButtons 函数。

对波特率的调整在 void setBaudRate(uint32_t lBaudrate) 函数，这一函数是由 dr_usb/usbEventHandling.c 调用的。

需要注意的是，USB 库函数不可以高频率连续调用(包括检查状态的函数)，高频率连续调用会导致一部分操作无法正常进行，故在主循环中有延时，以提供给 USB 模块处理时间。

4.6.7 问题思考

虚拟串口中数据在 PC 机与 MSP430F6638 之间的传输速率受不受波特率设定的限制？

4.7 Pro_07 I²C总线综合实验

4.7.1 实验目的

- 学习使用 MSP430F6638 的通用串行通讯模块访问 I²C 总线的外设；
- 学习矩阵键盘和矩阵 LED 的控制方法。

4.7.2 实验内容

编程实现对键盘与数码管小板的控制，包括读取键盘输入和控制数码管输出。键盘与数码管小板通过 I²C 总线与 MSP430F6638 通讯。

4.7.3 实验原理

- 数码管扫描显示部分可参考 3.8.1；
- 矩阵键盘扫描部分可参考 3.8.2；

4.7.4 实验步骤

- 通过 10-pin 连接线将键盘子板连接到主实验板的 P11 或 P14 端口（注意连接方向，连接线红线对准端口“1”管脚）；
- 运行程序，按下键盘上各按键，观察数码管显示的变化。

4.7.5 实验现象

数码管上循环显示数值 0~F，按下矩阵键盘上的按键时，键值将显示在数码管上。

4.7.6 关键代码分析

4.7.6.1 I²C库

这一 I²C 库由 dr_i2c.c 与 dr_i2c.h 两个文件组成，使用时需要在 dr_i2c.h 中指定 SMCLK

的频率与希望的 I²C 总线时钟的频率，这一库使用 P2.1 作为 SDA、P2.2 作为 SCL，并使用 UCB0 模块作为通讯用的硬件模块。

这一 I²C 库的写入功能提供 20 个缓冲区位置，每个缓冲区位置可以向特定地址的特定寄存器写入一个单字节值，即发送序列为

[地址(写)][寄存器(下行)][写入值(下行)]

最新提交的写入请求会进入缓冲区的末尾并等待进行发送。若提交写入请求时没有在进行任何写入或读取，则提交时会立刻启动发送；若提交写入请求时正在进行读取，则在当前的读取操作完成后启动最早的未完成的写入操作的发送过程。

I²C 库的读取功能也提供 20 个缓冲区位置，每个位置可以从特定地址的特定寄存器读出两字节的数据，即发送序列为

[地址(写)][寄存器(下行)][地址(读)][读出值低位(上行)][读出值高位(上行)]

向 I²C 库插入一个读取请求的函数会返回一个整数，代表这一查询请求的索引值，之后可以用这一索引号获取读到的值。对同一寄存器的读取请求只需要插入一次，且目前的版本不提供删除读取请求的接口。插入读取请求后程序会尽快在空闲时(无写入任务时)进行读取，从从机读取完成后需要使用 I2C_CheckQuery 函数读出读取值后程序才会进行下一次读取。

由于这一个库主要考虑方便互相独立的读写过程，所以不适合用于需要较复杂的时序配合的过程。

4.7.6.2 LED数码管控制

每一个数码管上每一位的亮灭状态保存在 seg_buffer 变量中；

程序采用轮流点亮每一个数码管的方式来实现显示；

程序中有一个周期为 1ms 的定时中断，在中断中对当前点亮的数码管进行切换，即每秒扫描所有数码管 12.5 次；

在切换点亮的数码管时先关闭所有数码管的位选信号，再设定下一个需要显示的数码管的段选信号，再设定下一个需要显示的数码管的位选信号，这一操作序列可以防止下一个数码管上短时间出现上一个数码管的显示内容；

对 seg_buffer 的更新在 main 函数的 while 循环中。

4.7.6.3 LED数码管控制

对矩阵键盘的扫描采取每 10ms 更换一条线、在更换扫描线前读取最近的扫描结果的方式进行。

更换扫描线和读取结果都在 1ms 定时中断中进行。其中每个 ms 都会有一次对扫描结果的空读取操作(读取但不保存结果)，以保证 I²C 库不断更新扫描结果，从而可以读到最新的结果。

4.7.7 问题思考

- I²C 总线使用什么机制使得总线上的多个设备被允许同时向总线上输出？这一机制对设备的输出电路有何要求？