

前 言	4
第 1 章 凌阳公司简介	1
第 2 章 SPCE061A 精简开发板 - 61 板	2
2.1 61 板功能描述	2
2.2 61 板硬件说明	3
2.2.1 输入/输出 (I/O) 接口	5
2.2.2 音频输入/输出接口	5
2.2.3 在线调试器 (PROBE) 和 EZ-PROBE 接口	5
2.2.4 电源接口	6
2.2.5 外部复位	6
2.3 配套 LED 键盘模组说明	6
第 3 章 凌阳 16 位单片机集成开发环境使用	10
3.1 凌阳 16 位单片机集成开发环境简介	10
3.2 举例应用	12
第 4 章 61 板结合 LED 键盘模块实验	17
实验一 发光二极管单向循环点亮	17
【实验目的】	17
【实验设备】	17
【实验说明】	17
【实验步骤】	18
【硬件连接图】	18
【程序流程图】	19
【程序范例】	19
【程序练习】	23
实验二 发光二极管双向循环点亮	23
【实验目的】	23
【实验设备】	23
【实验说明】	23
【实验步骤】	23
【硬件连接图】	24
【程序流程图】	25
【程序范例】	25
【程序练习】	29
实验三 按键点亮发光二极管	30
【实验目的】	30
【实验设备】	30
【实验说明】	30
【硬件连接图】	31
【程序流程图】	31
【程序范例】	31
【程序练习】	36

实验四 键控发光二极管循环点亮.....	36
【实验目的】	36
【实验设备】	37
【实验说明】	37
【实验步骤】	37
【硬件连接图】	37
【程序流程图】	38
【程序范例】	38
【程序练习】	44
实验五 数码管显示 0 - 9	44
【实验目的】	44
【实验设备】	44
【实验说明】	44
【实验步骤】	44
【硬件连接图】	44
【程序流程图】	46
【程序范例】	46
【程序练习】	51
实验六 数码管移位循环显示 0 - 9.....	51
【实验目的】	51
【实验设备】	51
【实验步骤】	51
【硬件连接图】	52
【程序流程图】	52
【程序范例】	53
【程序练习】	56
实验七 发光二极管和数码管交替显示.....	56
【实验目的】	56
【实验设备】	57
【实验说明】	57
【实验步骤】	57
【硬件连接图】	57
【程序流程图】	57
【程序范例】	58
【程序练习】	63
实验八 按键显示数字	64
【实验目的】	64
【实验设备】	64
【实验步骤】	64
【硬件连接图】	64
【程序流程图】	64
【程序范例】	65
【程序练习】	71
实验九 发光二极管巡回点亮并数码管计数.....	71

【实验目的】	71
【实验设备】	71
【实验说明】	71
【实验步骤】	72
【硬件连接图】	72
【程序流程图】	73
【程序范例】	73
【程序练习】	80
实验十 A/D 采样数据在发光二极管上点亮	80
【实验目的】	80
【实验设备】	80
【实验说明】	80
【实验步骤】	80
【硬件连接图】	81
【程序流程图】	81
【程序范例】	81
【程序练习】	85
第 5 章 61 板深入学习向导	86
5.1 学习向导	86
5.2 61 板其他配套模组说明	87
5.2.1 USB 接口模组	88
5.2.2 USB/UART 转换口模组	88
5.2.3 以太网通讯模组	89
5.2.4 SPLC501 液晶模组	89
5.2.5 图像识别模组	90
5.2.6 交通灯模组	90
第 6 章 附录	91

前 言

本教材是结合北京航空航天大学出版社出版的《凌阳 16 位单片机应用基础》一书而设计，与课堂教学内容结合紧密。本书所有实验均是 61 板结合 LED 键盘模组而设计，且附有范例程序。实验内容浅显易懂，属于基础应用实验，读者可以从学习凌阳单片机 SPCE061A 端口的使用方法入手，逐步掌握凌阳 μ^n SP 系列单片机的编程方法以及 61 板的基本功能和常用外围器件如键盘、LCD、LED 的使用方法等。

本书共分五节，基本内容如下：

第一节：凌阳公司简介；

第二节：61 板及配套 LED 模组说明；

第三节：凌阳 16 位单片机集成开发环境使用简介；

第四节：61 板结合 LED 模组基础实验，共十个；

第五节：61 板其它配套模组简介。

本书中的所有实验范例代码均经过调试，范例中既包括汇编语言版程序又包括 C 语言版程序。实验时按照硬件连接说明进行连接后，程序可直接下载运行，使读者达到节省时间、快速入门的目的。

由于编者水平有限，编写时间仓促，书中难免有所错漏，敬请读者和专家指正。

凌阳科技大学计划部

二零零四年八月

第 1 章 凌阳公司简介

全球第一大消费性芯片设计公司——凌阳科技，1990 年 8 月成立于台湾。凌阳科技的主要业务为研发、制造、销售高品质及高附加价值的消费性集成电路（IC）产品。它拥有较先进的工艺设计技术，提供几千种标准产品，广泛应用于工业领域和消费类电子产品领域。部分型号单片机可以完成在线编程、仿真和调试。此设计不仅降低开发者的成本，而且在很大程度上可以加快开发者的设计进程。其主要产品包括：液晶 IC、微控器 IC、多媒体 IC、语音、音乐 IC 及各式 ASICs，同时还提供高性能的外围电路，包括 LCD、AGC、DTMF、A/D、D/A、UART、SPI、PCI、计数器和存储控制器等等。

为了回馈高校，实现“科技落实生活”的企业宗旨，2001 年凌阳科技在大陆隆重推出——凌阳大学计划（<http://www.unsp.com.cn>）。凌阳大学计划内容包括：共建凌阳单片机实验室；支持大学采用 61 板完成电子实习；支持大学采用 61 板完成课程设计；支持大学采用 61 板完成毕业设计；支持大学教师编写凌阳单片机教材或专著；鼓励大学师生发表凌阳单片机论文；鼓励采用凌阳单片机参加各类电子竞赛；鼓励大学教师采用凌阳单片机进行项目开发。凌阳大学计划希望能达到改善大专院校单片机教学条件，增进高校师生接触新技术、新器件的机会，提高在校生的动手能力，推动教学和新技术同步发展的目的。

第 2 章 SPCE061A 精简开发板 - 61 板

2.1 61 板功能描述

61 板是 SPCE061A 精简开发板的简称，是“凌阳大学计划”专为电子爱好者和高校学生进行课程设计、毕业设计 & 电子竞赛所设计的，以凌阳 16 位单片机 SPCE061A 为核心的精简开发 - 仿真 - 实验板。硬件电路包括电源电路、音频电路（含 MIC 输入部分和 DAC 音频输出部分）、复位电路等，让学生在掌握软件设计的同时，熟悉单片机硬件的设计制作，锻炼动手能力。而且它的体积小，采用电池供电，方便随身携带。

具体能完成以下实验内容：

1. 20 多个基础实验：内含单片机常用的功能包括 I/O 口、中断、定时器/计数器、A/D 转换和 D/A 转换等；
2. 综合实验：配合学校培养学生动手能力的需求，实验需要搭配必要的电路完成，包括键盘、数码管、液晶（LCD）、USB 及外扩 FLASH 等模组；
3. 语音处理实验：提供三种应用于不同场合、不同压缩比的放音、录音（DVR）及语音辨识实验。同学们可以通过简单的操作实现放音或录音，了解一般语音处理的功能，极大丰富同学们的单片机知识，增强学生学习单片机的兴趣。

另外，SPCE061A 具有 16*16 位的乘法运算和内积运算的 DSP 功能，这不仅为它进行复杂的语音数字信号的压缩编码与解码提供了便利，还可以做数字滤波器（Digital Filter）。

那么该板主要应用在哪方面呢？

适合学生动手实践——代替原来旧有的安装收音机、万用表及直流稳压源等动手实习课程：

焊接：可以锻炼学生的焊接动手能力。凌阳大学计划提供 PCB 板、元器件、元器件清单，工艺要求及焊接步骤和要求等，让同学们充分了解硬件设计和制作的全过程。

测试：可以了解测试流程，增加测试经验，进一步掌握一般单片机的各种性能指标。针对 61 板我们为您提供标准的测试程序、详细的测试步骤和测试要求。测试程序具有智能的语音提示作用，只需根据提示内容即可完成每一步操作。通过对现象的观察即可完成各部分功能的测试，如电源电路是否正常、I/O 焊接是否有短路、A/D 转换和 D/A 转换电路是否正常等。

下载运行：给出操作步骤，熟悉一般单片机开发环境的使用，而且在下载完毕后即可实现报时器、音乐盒、复读机等功能。

其他：该开发板含有 A/D 和 D/A 的功能，通过动手实践活动可以实现简易万用表功能；板上的电源电路有 5V 和 3.3V 两种电压输出，使学生了解电源电路的基本结构。

适合学生课余自学

学习单片机开发环境（IDE），通过这一环节的学习掌握单片机编程过程中程序的调试、仿真和下载运行的方法。

整体学习单片机的各单元结构功能，如：I/O 口，定时/计数器、中断、A/D 和 D/A 等，而且该 16 位单片机既是对原来 8 位 51 单片机的补充，又具有其独特的语音处理功能，同时又是片上系统（SOC）设计，即内置 A/D 和 D/A 等很多功能。极大的拓宽了学生的知识面，培养了学生浓厚的学习兴趣。

初步编程，学习单片机的汇编语言编程。

适合学生课程设计

设计中包括一般单片机在 I/O 口、A/D、D/A 及 I/O 扩展等方面的应用举例，老师和学生可以结合实际情况选择学习。

学生不外接电路就可以实现 MIC 录音和语音播放的功能，也可以动手外搭面包板或自己用万能板焊接一些简单的外围电路，如发光二极管、LED 数码管或按键，可以实现更多的功能。

适合学生毕业设计

对于学生毕业设计，如果觉得一般单片机的扩展功能还不能满足学习要求，我们还配有：

1. 128 × 64LCD 模组
2. USB 模组
3. 4M 的存储器（memory）扩展
4. 以太网通讯模组
5. 步进电机、直流电机模组
6. 交通灯模组
7. USB/UART 模组
8. 图像识别模组

总之，61 板是一个 16 位单片机的平台，您可以充分发挥自己的想象，只需花很短的时间就可以具备掌握 061 的硬件结构及“精简指令”的基础，可以很快地自己设计制作产品。如果您有自己用 16 位单片机设计制作的语音遥控器、语音报时器等产品，一定不会为毕业时找工作而发愁。

2.2 61 板硬件说明

61 板硬件框图如图 1 所示：

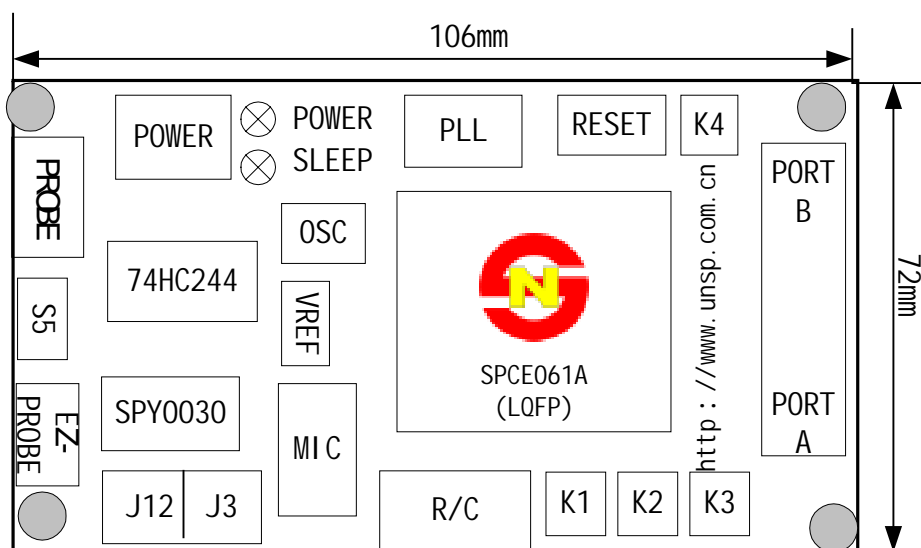


图 1 61 板硬件框图

表 1 框图说明

POWER	5V&3V 供电电路	PLL	锁向环外部电路
⊗	Power - 电源指示灯 Sleep - 睡眠指示灯	RESET	复位电路
K4	复位按键	PROBE	在线调试器串行 5 针接口
S5	EZ-PROBE 和 PROBE 切换的拨断开关	J12、J3	耳机插孔和两针喇叭插针
DAC	一路音频输出电路，采用 SPY0030 集成音频放大器	MIC	麦克风输入电路
OSC	32768 晶振电路	VREF	A/D 转换外部参考电压输入接口
R/C	芯片其他外围电阻、电容电路	K1~K3	扩展的按键：接 IOA0~IOA2
SPCE061A	61 板核心：16 位微处理器	PORTA/B	32 个 I/O 口

61 板接口说明图：

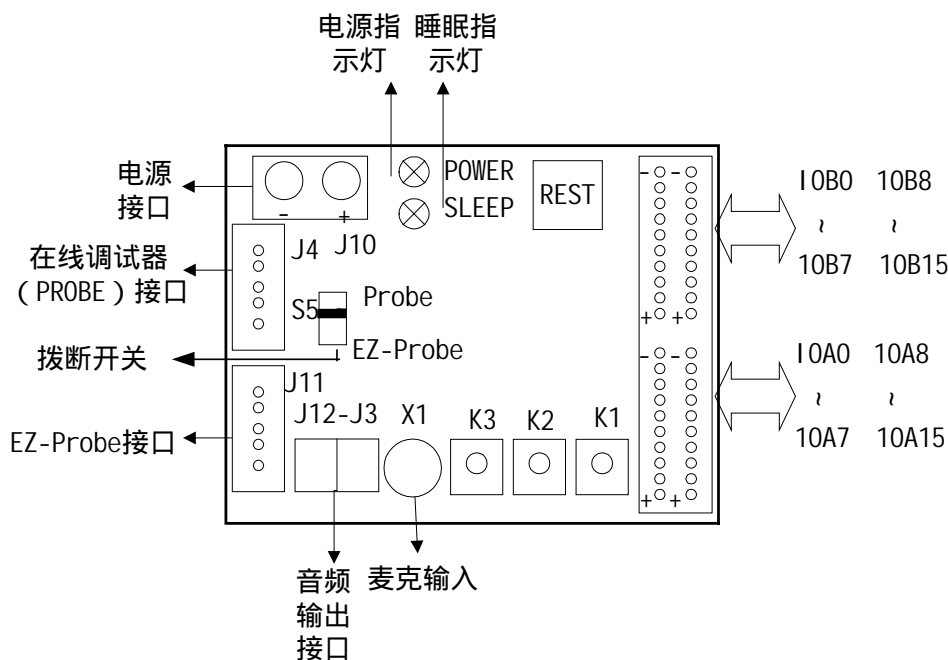


图 2 61 板接口说明图

2.2.1 输入/输出 (I/O) 接口

61 板将 SPCE061A 的 32 个 I/O 口 IOA0~IOA15, IOB0~IOB15 全部引出, 对应的引脚为: A 口, 41~48、53、54~60; B 口, 5~1、81~76、68~64。而且该 I/O 口是可编程的, 即可以设置为输入或输出。设置为输入时, 分为悬浮输入或非悬浮输入。非悬浮输入又可以设置为上拉输入或是下拉输入。在 5V 情况下, 上拉电阻为 150K, 下拉电阻为 110K; 设置为输出时, 可以选择同向输出或者反相输出。

2.2.2 音频输入/输出接口

正如我们在前面介绍的 61 板具有强大的语音处理功能, 如图 2 所示, X1 是语音的 MIC 输入端, 带自动增益 (AGC) 控制。J12 和 J3 都是语音输出接口, 一个是耳机插孔; 另一个是两针的插针外接喇叭, 由 DAC 输出引脚 21 或 22 经语音集成放大器 SPY0030 放大, 然后输出。SPY0030 是凌阳的芯片, 相当于 LM386, 但是比 386 音质好。它可以工作在 2.4~6.0V 范围内, 最大输出功率可达 700mW (LM386 必须工作在 4V 以上, 而且功率只有 100mW)。

2.2.3 在线调试器 (PROBE) 和 EZ-PROBE 接口

图 2 中 J4 为 PROBE 的接口, 该接口有 5 针, 其中两个分别是地 (VSS) 和 3.3V 电源 (VCC)。此接口与 PROBE 的 5 针接口相连, PROBE 的另一端接 PC 机 25 针并口。这样, 就不需要再用仿真器和编程器了, 只要按图 3 所示将其连接好, 就可以通过它在 PC 机上调试程序、在线仿真、最后将程序下载到芯片中, 完成程序的烧写。

图 2 中的 J11 是 EZ-PROBE 的接口, 我们提供一根转接线用作 EZ-PROBE 的下载, 一端连接 PC 机的 25 针并口, 另外一端连接 61 板的 5 针 EZ-PROBE 接口, 参见图 3。

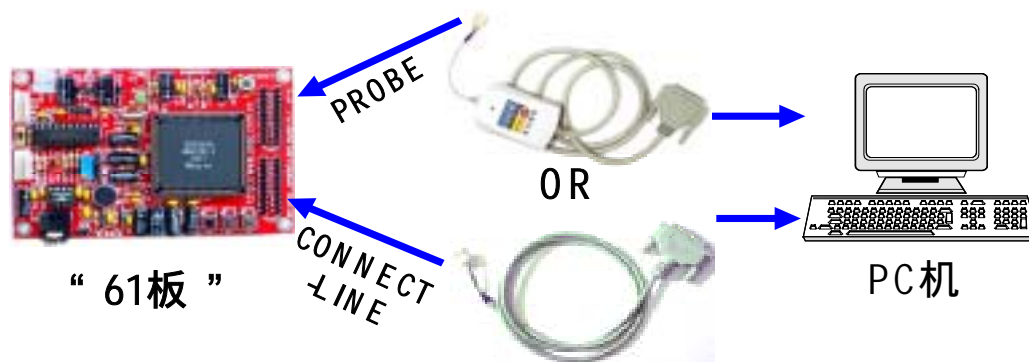


图 3 61 板、PROBE/连接线、计算机三者之间的连接图

2.2.4 电源接口

图 2 中 J10 是电源接口，61 板的内核 SPCE061A 电压要求为 3.3V，而 I/O 端口的电压可以选择 3.3V 也可以选择 5V。所以，在板子上具有两种工作电压：5V 和 3.3V。对应的引脚中 15、36 和 7 必须为 3.3V，对于 I/O 端口的电压 51、52、75 可以是 3.3V 也可以是 5V，这两种电平的选择通过跳线 J5 来控制。61 板的供电电源系统采用用户多种选择方式：

1. DC5V 电池供电

用户可以用 3 节电池来供电。5V 直流电压直接通过 SPY0029(相当于一般 3.3V 稳压器)稳压到 3.3V，为整个 61 板提供了 4.5V 和 3.3V 两种电平的电压。

2. DC5V 稳压源供电

用户可以直接外接 5V 的直流稳压源供电。5V 电压再通过 SPY0029 稳压到 3.3V。

3. DC3V 供电

用户可以提供直流 3.3V 电压为实验板进行供电。此时整个板子只有 3.3V 电压，I/O 端口电压此时只有一种选择。

需要注意的是由于 SPY0029 最大输出电流为 50mA，所以如果需要外接一些模组时先考虑负载是否合适。

2.2.5 外部复位

复位是对 61 板内部的硬件初始化。61 板本身具有上电复位功能，即只要一通电就自动复位。另外，还具有外部复位电路，即在引脚 6 上外加一个低电平就可令其复位。如图 2 中的 REST 按键。

2.3 配套 LED 键盘模组说明

模组资源：

1. 8 个按键，可以组成 1×8 按键，也可以组成 2×4 按键；
2. 8 个发光二极管；
3. 6 位 8 段 LED 数码管；
4. 一个电位器，提供 0 - 5V 的模拟电压信号。

LED 键盘模组接口说明图如图 4 所示：

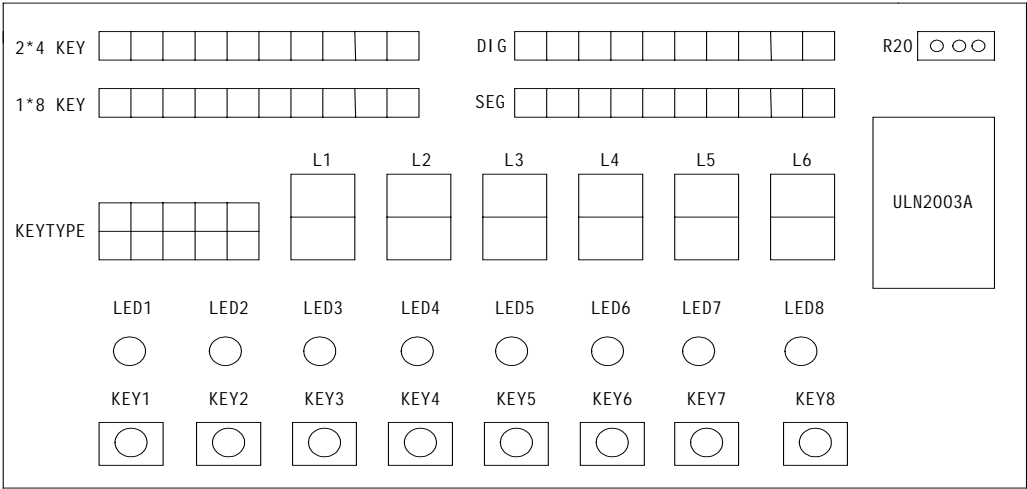


图 4 LED 模组接口说明图

接口说明：

LED 模组使用十分简单，直接用排线与 61 板 I/O 接口相连即可。

1. KEYTEPE 选择接口

此接口是选择 2*4KEY 还是 1*8KEY，具体选择方式如下：

从左至右看，将第 1 至 4 短接块连接，选择了 2*4KEY；只将第 5 个短接块短接，选择 1*8KEY。

选择 2*4KEY 接口连接示意图如下：

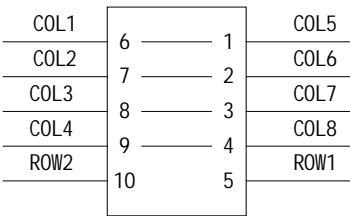


图 5 选择 2*4KEY 时 KEYTEPE 接口连接示意图

选择 1*8KEY 接口连接示意图如下：

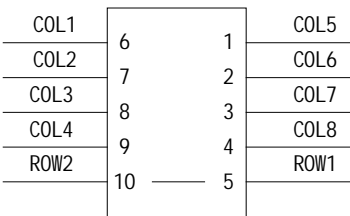


图 6 选择 1*8KEY 时 KEYTEPE 接口连接示意图

2. 2*4KEY 接口

2*4KEY 的第 1 组行定义为 ROW1，第 2 组行定义为 ROW2，4 列定义为 COL1~COL4。

使用时 COL1 是 K1 和 K5 的输入 ,COL2 是 K2 和 K6 的输入 ,COL3 是 K3 和 K7 的输入 ,COL4 是 K4 和 K8 的输入。用户可自行选择 ROW1 与 ROW2 接至 VDD 还是 GND，但同一时间只能使用一组。AV 是模拟电压输出端，通过调整 R20 可以改变 AV 的值，AV 的最大输出值与 VDD 相同。D_DP 是第 4 位数码管后时钟冒号的位信号控制端。

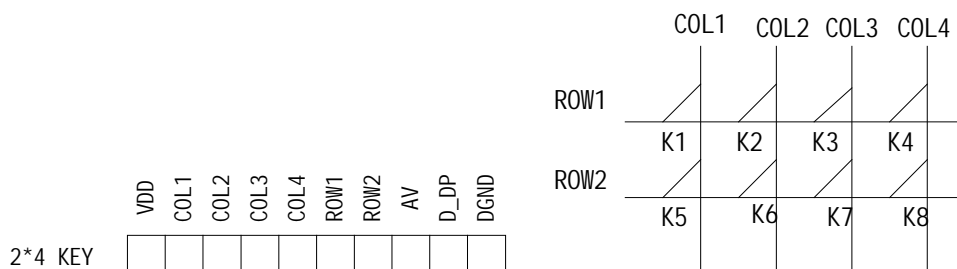


图 7 2*4KEY 接口图及使用示意图

使用 2*4KEY 时相应 KEYTEPE 连接如图 5。

3. 1*8KEY 接口

1*8KEY 的 8 列分别定义为 COL1~COL8，1 行定义为 ROW1。

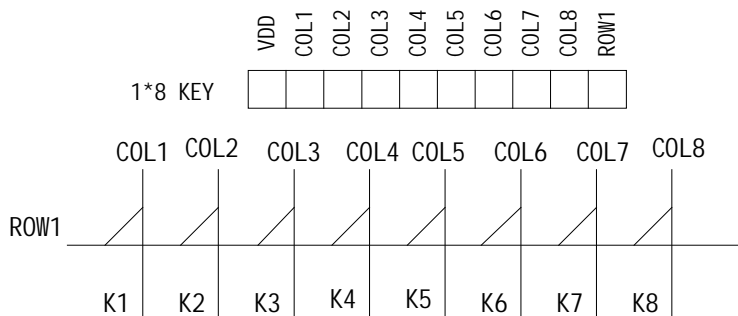


图 8 1*8KEY 接口图及使用示意图

使用时 KEYTEPE 接口的第五个短接块短接。用户可自行选择 ROW1 接至 VDD 还是 GND。

使用 1*8KEY 时相应 KEYTEPE 连接如图 6。

4. DIG 接口

6 位数码管的段发光管阳极和 8 个 LED 指示灯的阳极并联 ,并且 8 个 LED 指示灯采用共阴极方式。6 位数码管的阴极和 8 个 LED 的共阴极分别用 DIG 接口的 DIG1~DIG7 控制，第 4 位数码管后时钟冒号的位信号用 DIG8 控制，位信号均为高有效。



图 9 DIG 接口图

5. SEG 接口

A~G、DP 是 6 位数码管的段信号和 8 个 LED 指示灯的阳极控制信号输入端。控制信号为高有效。

与 61 板配套使用时，2、3、4 接口可用排线直接连至 61 板的 IOA 或 IOB 接口。

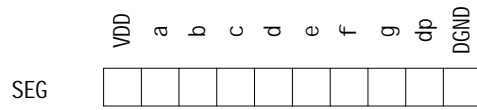


图 10 SEG 接口图

实验前请仔细阅读附录，要了解数码管和 LED 的结构，在用排线连接系统开发板和模组进行实验时一定要注意方向问题：板子的 V_{CC} 与模组的 VDD 是同一点。

第 3 章 凌阳 16 位单片机集成开发环境使用

3.1 凌阳 16 位单片机集成开发环境简介

$\mu'nSP^TM$ 集成开发环境，它集程序的编辑、编译、链接、调试以及仿真等功能为一体。具有友好的交互界面、下拉菜单、快捷键和快速访问命令列表等，使人们在进行编程、调试工作时更加方便高效。此外，它还具有软件仿真功能，可以在不连接仿真板的情况下模拟硬件的各项功能来调试程序。

IDE 的开发界面如图 11 所示。本节将介绍 $\mu'nSP^TM$ 开发环境的菜单、窗口界面以及项目的操作等，使有兴趣者对开发环境有一个总体了解，并能够动手实践。

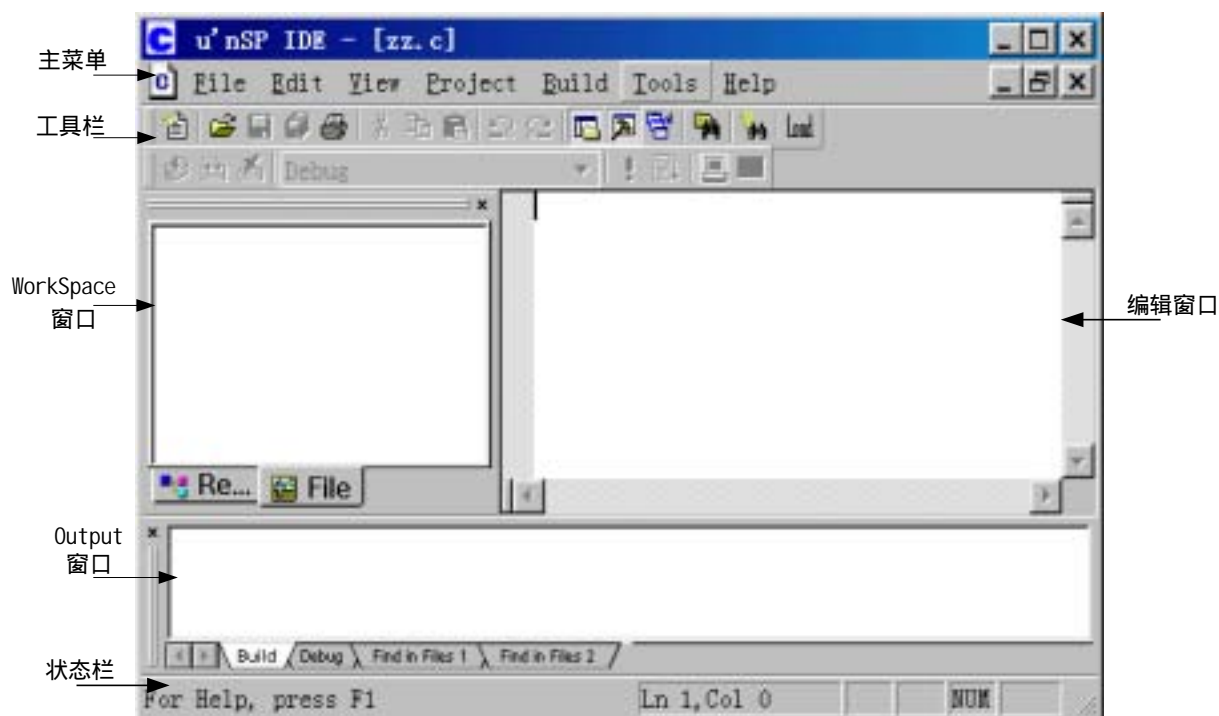


图 11 开发环境界面

凌阳十六位单片机集成开发环境采用项目方式进行文件管理。项目（Project）是指为用户调试程序建立起来的一个开发环境，提供用户程序及资源文档的编辑和管理，并提供各项环境要素的设置途径，最后将通过对用户程序及库的编制（包括编译、汇编以及链接等）**为用户**提供出一个良好的调试环境。

凌阳十六位单片机集成开发环境的工具栏中含有 $\mu'nSP^TM$ IDE 提供的 3 种工具栏：标准、编辑和调试工具栏。每种工具栏都有固定和浮动两种形式。把鼠标移到固定形式工具栏中没有图标的地方，按住左按钮，向下拖动鼠标，即可将工具栏变为浮动工具栏；双击浮动工具栏的标题条，则可将其变为固定工具栏。

固定形式的标准工具栏位于菜单栏的下面，它以图标的形式提供了部分常用菜单命令的功能。只要用鼠标单击代表某个命令的图标按钮，就能直接执行相应的菜单命令。工具条中有 38 个图标，

代表 38 种操作，如图 12 所示。

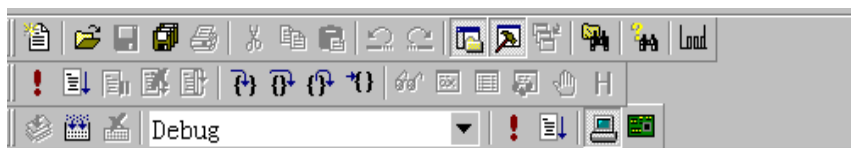


图 12 工具栏

$\mu'nSPT^M$ IDE 支持多文档窗口操作，用户可以在主界面里同时打开多个窗口，如图 13 所示。

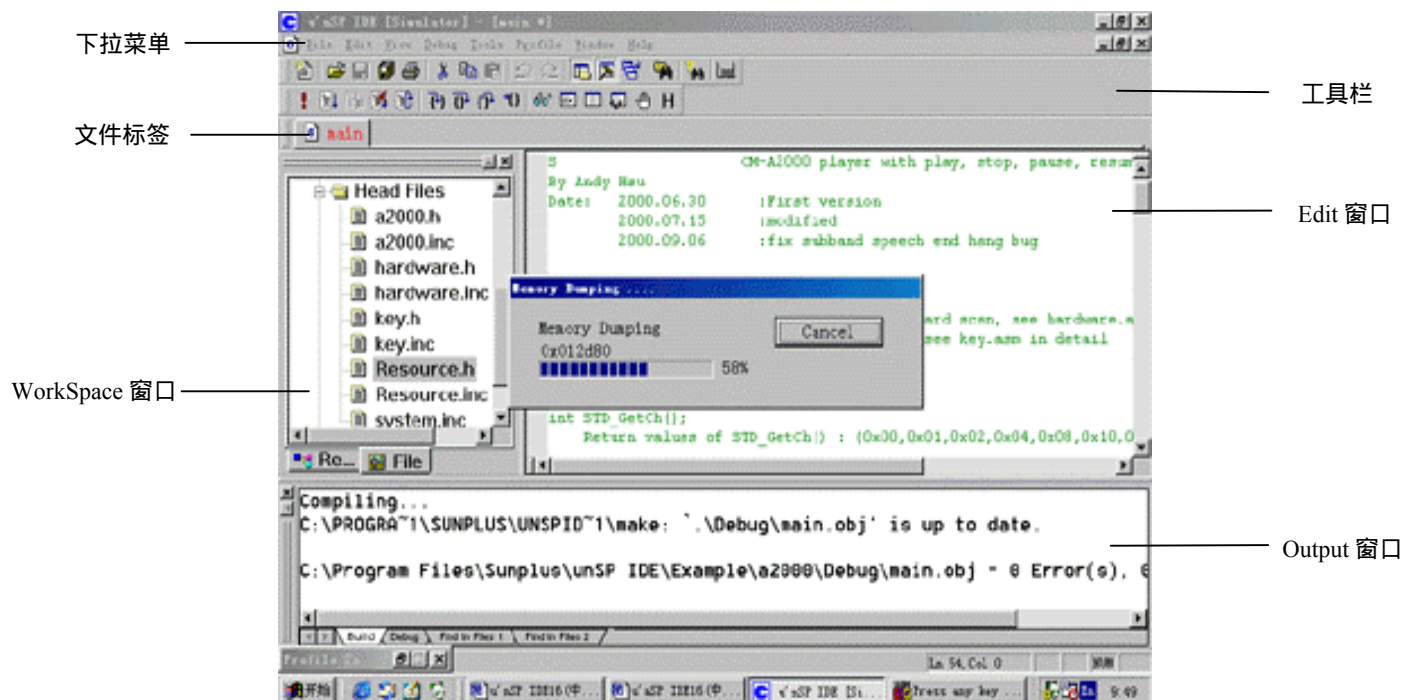


图 13 文档窗口

主界面里通常有三个主要窗口：Workspace（工作区）窗口、Edit（编辑）窗口和 Output（输出）窗口。进行窗口切换只需在各窗口处单击鼠标左键即可。此外，主界面里还提供下拉菜单、工具栏等。

$\mu'nSPT^M$ IDE 支持软件仿真和在线调试。在两种调试过程中，均有单步运行、全速运行、断点调试以及变量窗口、寄存器窗口、内存窗口、反汇编窗口等以方便用户进行软件调试和硬件调试。软件调试时，集成开发环境可以仿真各种中断和端口状态。这些均为软件开发者提供了方便。如图 14 为调试状态界面。

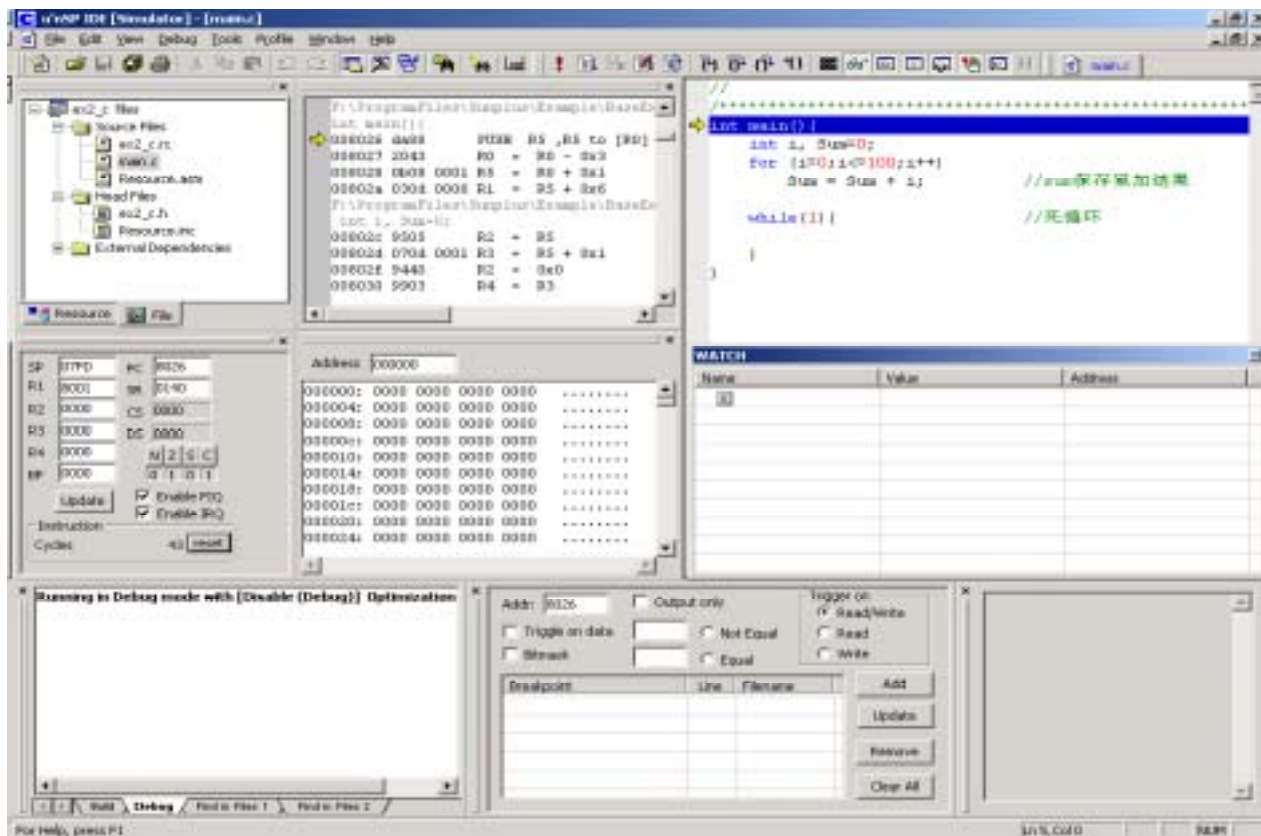


图 14 文件程序 Download 后的调试界面

在μ'nSPT™ IDE 中，配备硬件声明的头文件和常用函数模块，例如键盘扫描模块，语音音量调节模块等等。另外还配备各种库文件，包括标准 C 常用的库函数、凌阳音频库函数等，尤其是凌阳音频库函数，只要了解这些函数的使用方法，用户就可以轻而易举的完成平时认为较神秘的语音播放、语音录入和语音识别功能。这些均可以大大加快开发者的开发进程。

3.2 举例应用

帶有资源文件(.rc) C 与 ASM 文件

范例：自动放音程序

EX1

主程序为 C 文件，程序如下：

```
#include "a2000.h" // 包括 A2000.h
#define SPEECH_1 0 // SPEECH_1 = 0 表示最大语音索引号
#define DAC1 1 // 第一通道
```



```

#define DAC2  2                                // 第二通道

#define Ramp_UpDn_Off 0                        // 禁止音量增/减调节

#define Ramp_Up_On  1                          // 仅允许音量增调节

#define Ramp_Dn_On  2                          // 仅允许音量减调节

#define Ramp_UpDn_On  3                        // 允许音量增减调节


main()
{
    SACM_A2000_Initial(1);                      // 设置为自动放音方式

    SACM_A2000_Play(SPEECH_1, DAC1+DAC2, Ramp_UpDn_On);

                                                // 播放资源的 SACM_A2000 格式语音和乐曲

    while(1)
    {
        SACM_A2000_ServiceLoop();              // 从 SRAM 中获取语音数据，对其进行解码

                                                // 等待中断服务程序将其送出 DAC 通道

    }
}

```

中断服务程序的 ASM 文件,程序如下:

```

.TEXT

.INCLUDE hardware.inc                        // 包含头文件

.INCLUDE a2000.inc

.INCLUDE Resource.inc

.PUBLIC _FIQ;                                // 声明_FIQ 中断

_FIQ:

    PUSH R1,R4 TO [SP];

    R1 = C_FIQ_TMA;

    TEST R1, [P_INT_Ctrl];

    JNZ L_FIQ_TimerA;                        // 定时器 A 的中断入口

L_FIQ_PWM:                                  // 中断 PWM FIQ 入口

    R1 = C_FIQ_PWM;

    [P_INT_Clear] = R1;

```



```
POP R1, R4 FROM [SP];
```

```
L_FIQ_TimerA:                                // 中断子程序 FIQ_TimerA

[P_INT_Clear] = R1;                          // 清中断

CALL F_FIQ_Service_SACM_A2000;              // SACM-A2000 定时器 A FIQ 解码

POP R1,R4 FROM [SP];                        // 出栈

RETI;
```

```
L_FIQ_TimerB:                                // 中断子程序 FIQ_TimerB

[P_INT_Clear] = R1;                          // 清中断

POP R1,R4 FROM [SP];                        // 出栈

RETI
```

方法步骤：

1. 新建项目，项目名称 EX1。
2. 该项目下新建汇编文件，文件名称 SYSTEM.ASM。
3. 在汇编文件中键入范例汇编源代码（图 15）。
4. 该项目下新建 C 文件，文件名称 MAIN.C。
5. 在 C 文件中键入范例 C 源代码（图 16）。
6. 在源文件组中添加 HARDWARE.ASM 文件。
7. 在头文件组中添加 A2000.INC、HARDWARE.INC 和 RESOURCE.INC 头文件。
8. 在资源文件视窗中，添加 RES_A27、RES_A32 和 RES_A38 资源文件。
9. 保存项目。
10. 编译调试该程序。
11. 下载到仿真板中。

注意：当编译时，会出现如下错误：

Error L0080: The external symbol "T_SACM_A2000_SpeechTable" has not a public definition.

这时在系统自动生成的 Resource.asm 文件中添加如下内容即可：

```
.PUBLIC T_SACM_A2000_SpeechTable
T_SACM_A2000_SpeechTable
.DW _RES_A27_SA;                          // 资源起始地址
```

.DW _RES_A32_SA

.DW _RES_A38_SA

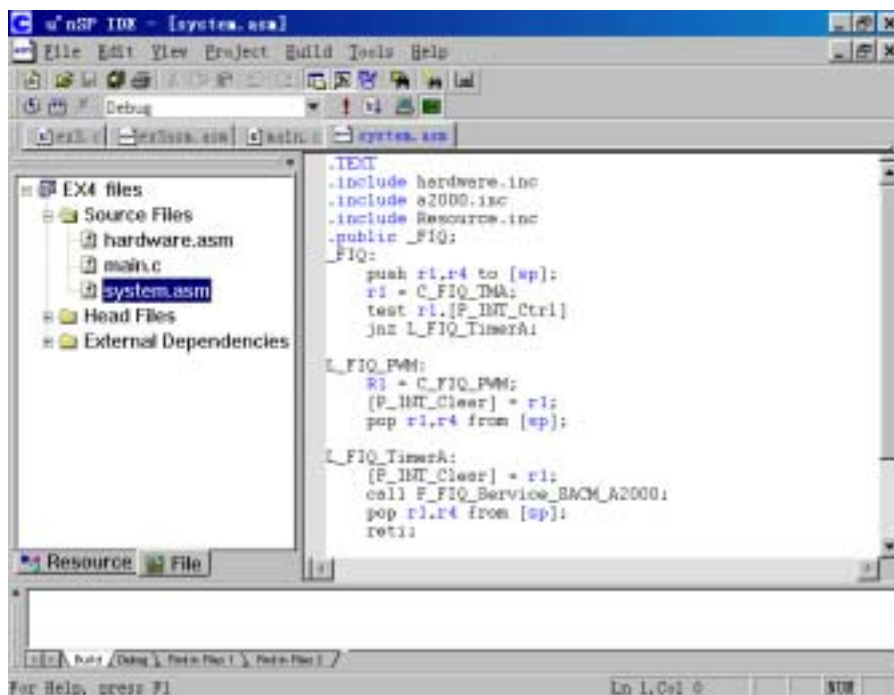


图 15 汇编文件源代码界面

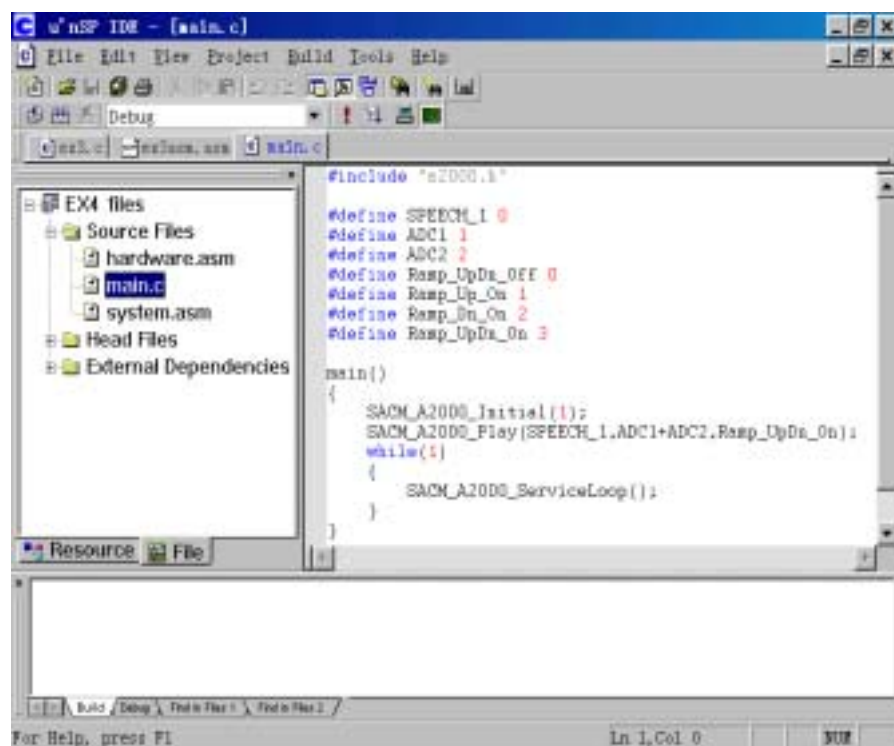
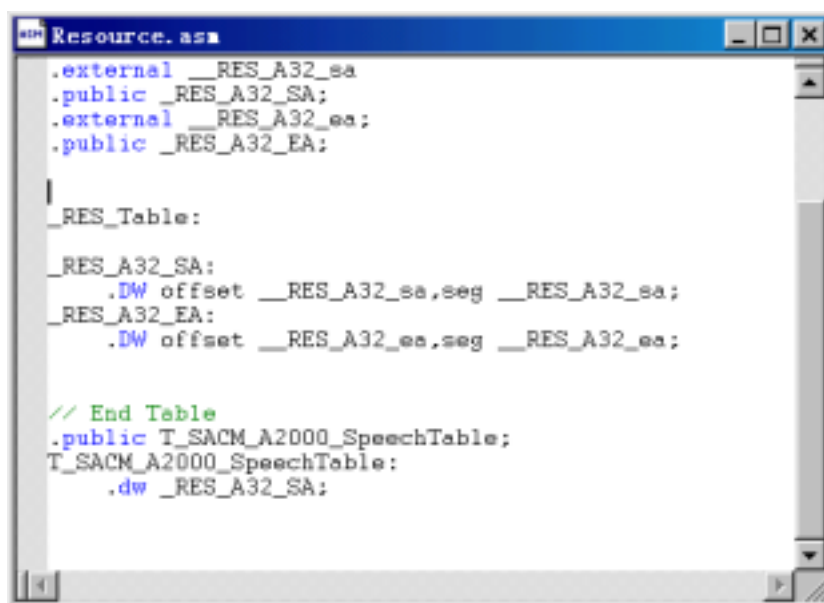


图 16 C 文件源代码界面



```
Resource.asm
.external __RES_A32_ea
.public __RES_A32_SA;
.external __RES_A32_ea;
.public __RES_A32_EA;

|
_RES_Table:

_RES_A32_SA:
.DW offset __RES_A32_ea,seg __RES_A32_ea;
_RES_A32_EA:
.DW offset __RES_A32_ea,seg __RES_A32_ea;

// End Table
.public T_SACM_A2000_SpeechTable;
T_SACM_A2000_SpeechTable:
.dw __RES_A32_SA;
```

图 17 EX4 RESOURCE .ASM 编译成功后的文件界面

第4章 61 板结合 LED 键盘模块实验

实验一 发光二极管单向循环点亮

【实验目的】

1. 熟悉 $\mu'nSPTM$ IDE 环境及在该环境下用汇编和 C 语言编写的应用程序。
2. 熟悉简单的 $\mu'nSPTM$ 汇编语言指令。
3. 以 A 口和 B 口低八位为例，学会使用 SPCE061A 单片机 I/O 口的基本输出功能。

【实验设备】

1. 装有 Windows 系统和 $\mu'nSPTM$ IDE 仿真环境的 PC 机一台。
2. 61 板一套；LED 键盘模组一套；10 针排线两根。

【实验说明】

1. 61 板的 I/O 输出实验主要以 IOA0~7 接 LED 键盘模组上的 8 个发光二极管。因 61 板核心芯片 SPCE061A 已内置上下拉电阻，所以端口直接连接发光二极管的驱动端。
2. 代码编写上，主要涉及 SPCE061A 的端口寄存器。

寄存器	功能	使用说明
P_IOA_Dir	方向寄存器	0：输入 1：输出
P_IOA_Attrib	属性寄存器	0：输入时为电阻输入 输出时为反相输出 1：输入时为悬浮输入 输出时为同相输出
P_IOA_Data	数据寄存器	0：电阻输入时为下拉电阻输入 悬浮输入时为带唤醒输入 同相输出时为低电平输出 反相输出时为高电平输出 1：电阻输入时为上拉电阻输入 悬浮输入时为不带唤醒输入 同相输出时为高电平输出 反相输出时为低电平输出

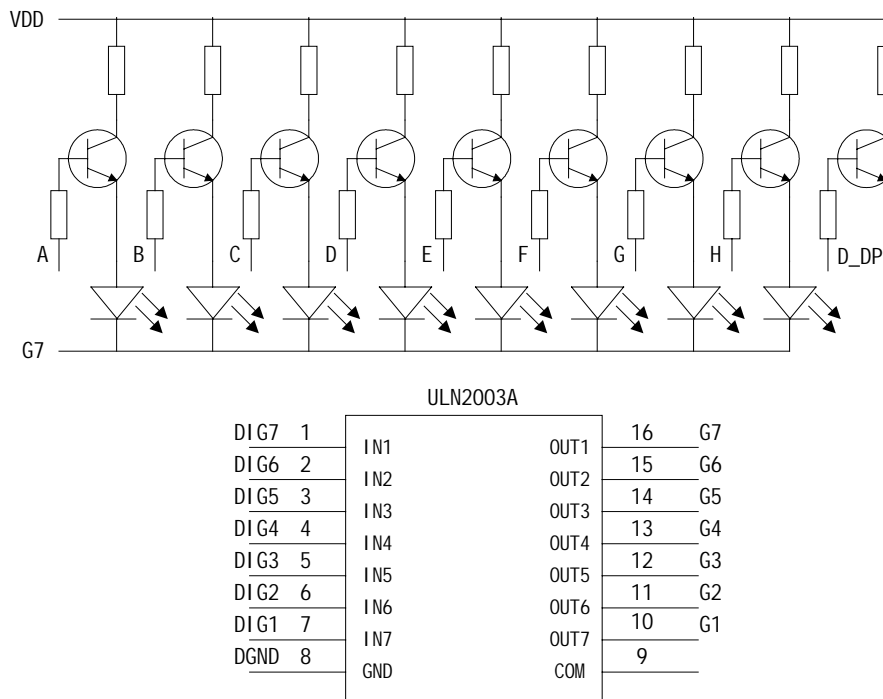
IOB 口寄存器定义同上。

【实验步骤】

1. 用 10 针排线将 61 板的 J8 接口 IOA 低 8 位连接到 LED 键盘模组的 SEG 接口管脚上，再将 J6 接口 IOB 低 8 位连接到 LED 键盘模组的 DIG 接口管脚上。

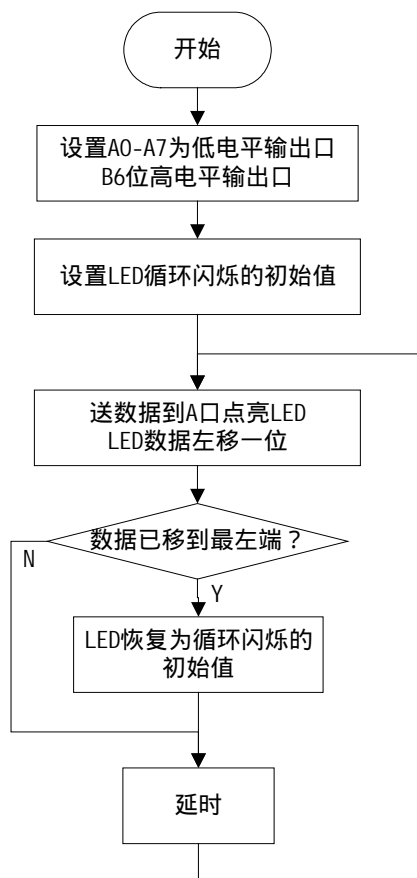
2. 运行参考程序。

【硬件连接图】



IOA 低 8 位接至 SEG 接口控制 LED 的导通，IOB6 连至 DIG7 通过 ULN2003A 控制 8 个 LED 的共阴极电平状态，也可将 DIG7 直接连至 VDD，直接将 LED 共阴极接地，不用程序控制。

【程序流程图】



【程序范例】

汇编语言版：

```

.include hardware.inc                // 包含 hardware.inc

.ram

.var R_LedControl                    // 定义变量
.var R_DelayCounter

.code

.public _main

//=====//

// 函数名称：    _main
// 功能描述：    发光二极管单向循环点亮
// 语法格式：    void main(void);

```



```
// 入口参数:      无

// 出口参数:      无

// 注意事项:      仅为用户模型

//=====//
//=====//

_main:

    r1 = 0x00FF                // 设置 A0~A7 口为同相低电平输出

    [P_IOA_Dir] = r1

    [P_IOA_Attrib] = r1

    r1 = 0x0000

    [P_IOA_Data] = r1

    r1=0x0040

    [P_IOB_Dir]=r1

    [P_IOB_Attrib]=r1

    [P_IOB_Data]=r1           // B6 输出高电平，保证 LED 阴极接地

    r1=0x0001

    [R_LedControl] = r1       // LED 点亮控制初值

L_MainLoop:

    r1 = [R_LedControl]

    [P_IOA_Data] = r1         // 送数据到 A 口

    r1=r1 LSL 1               // 修改 LED 点亮控制参数

    cmp r1, 0x0100           // 只有 8 个 LED，保证输出数据在 8 位以内

    jne NoOver

    r1 = 0x0001

NoOver:

    [R_LedControl] = r1

    r1 = 0x0000               // 延时计数器清零

    [R_DelayCounter] = r1

    call F_Delay              // 延时

    jmp L_MainLoop           // 跳转到 L_MainLoop 循环，点亮 LED

//=====//
```



```
// 函数名称 :      F_Delay
// 功能描述 :      实现延时

//=====//

F_Delay:
L_DelayLoop:
    r1=0x0001                // 清看门狗

    [P_Watchdog_Clear] = r1

    r1 = [R_DelayCounter]

    r1 += 1                  // 延时计数加 1

    [R_DelayCounter] = r1

    jnz  L_DelayLoop        // 加到 65536 吗？

    retf
```

C 语言版：

```
#define P_IOA_Data      (volatile unsigned int *)0x7000
#define P_IOA_Buffer    (volatile unsigned int *)0x7001
#define P_IOA_Dir       (volatile unsigned int *)0x7002
#define P_IOA_Attrib    (volatile unsigned int *)0x7003
#define P_IOB_Data      (volatile unsigned int *)0x7005
#define P_IOB_Buffer    (volatile unsigned int *)0x7006
#define P_IOB_Dir       (volatile unsigned int *)0x7007
#define P_IOB_Attrib    (volatile unsigned int *)0x7008
#define P_Watchdog_Clear (volatile unsigned int *)0x7012

//=====//

// 函数名称 :      Delay()
// 功能描述 :      实现延时

//=====//

void Delay()
{
    // 延时子程序

    unsigned int i;

    for (i=0; i<32768; i++)
    {
```



```
*P_Watchdog_Clear=0x0001;          // 清看门狗
}

}

//=====//

// 函数名称：      int main()
// 日    期：      20040816
// 功能描述：      单向循环点亮
// 语法格式：      void main(void);
// 入口参数：      无
// 出口参数：      无
// 注意事项：      仅为用户模型

//=====//

int main()
{
    int LedControl = 0x0001;

    *P_IOA_Dir = 0x00ff;              // 设置 A 口低 8 位为同向低输出

    *P_IOA_Attrib = 0x00ff;

    *P_IOA_Data = 0x0000;

    *P_IOB_Dir=0x0040;                // 设置 B6 口为高电平输出，保证 LED 共阴极接地

    *P_IOB_Attrib=0x0040;

    *P_IOB_Data=0x0040;

    while(1)
    {
        *P_IOA_Data = LedControl;    // 送数据到 A 口

        LedControl = LedControl << 1;

        If (LedControl > 0x00FF)

            LedControl = 0x0001;

        Delay();
    }
}
```

【程序练习】

在 $\mu'nSP^TM$ IDE 下用汇编语言和 C 语言编写隔一发光二极管单向循环点亮的程序。

```
//=====//  
EX1    END  
//=====//
```

实验二 发光二极管双向循环点亮

【实验目的】

1. 熟悉 $\mu'nSP^TM$ IDE 环境及在该环境下用汇编和 C 语言编写的应用程序。
2. 熟悉简单的 $\mu'nSP^TM$ 汇编语言指令。
3. 以 A 口和 B 口低八位为例，学会使用 SPCE061A 单片机 I/O 口的基本输出功能。

【实验设备】

1. 装有 Windows 系统和 $\mu'nSP^TM$ IDE 仿真环境的 PC 机一台。
2. 61 板一套；LED 键盘模组一套；10 针排线两根。

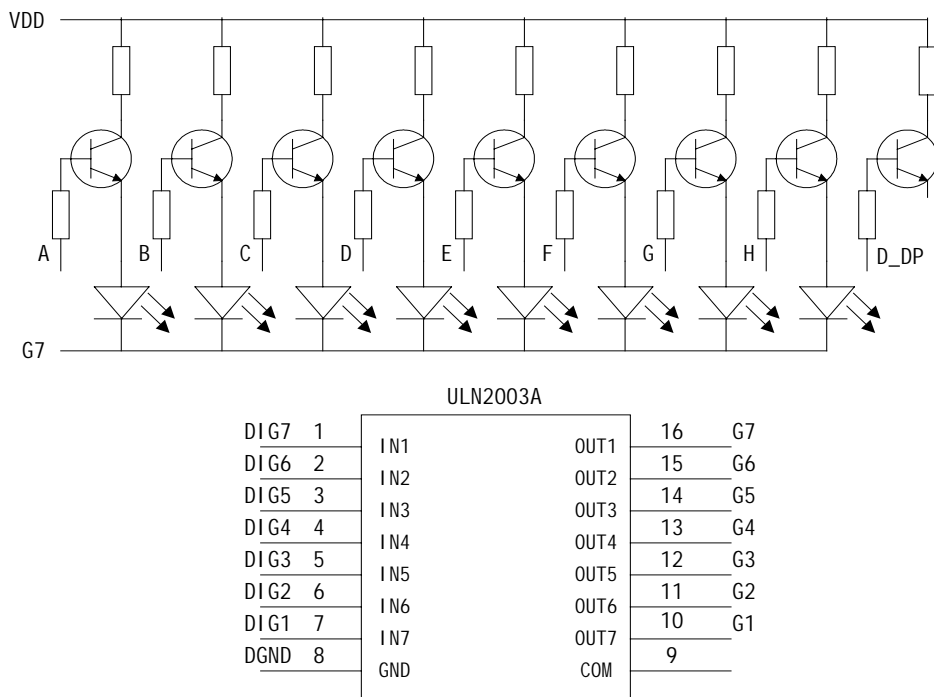
【实验说明】

1. 61 板的 I/O 输出实验主要以 IOA0~7 接 LED 键盘模组上的 8 个发光二极管。因 61 板核心芯片 SPCE061A 已内置上下拉电阻，所以端口直接连接发光二极管的驱动端。
2. 代码编写上，主要涉及 SPCE061A 的端口寄存器 IOA 和 IOB。

【实验步骤】

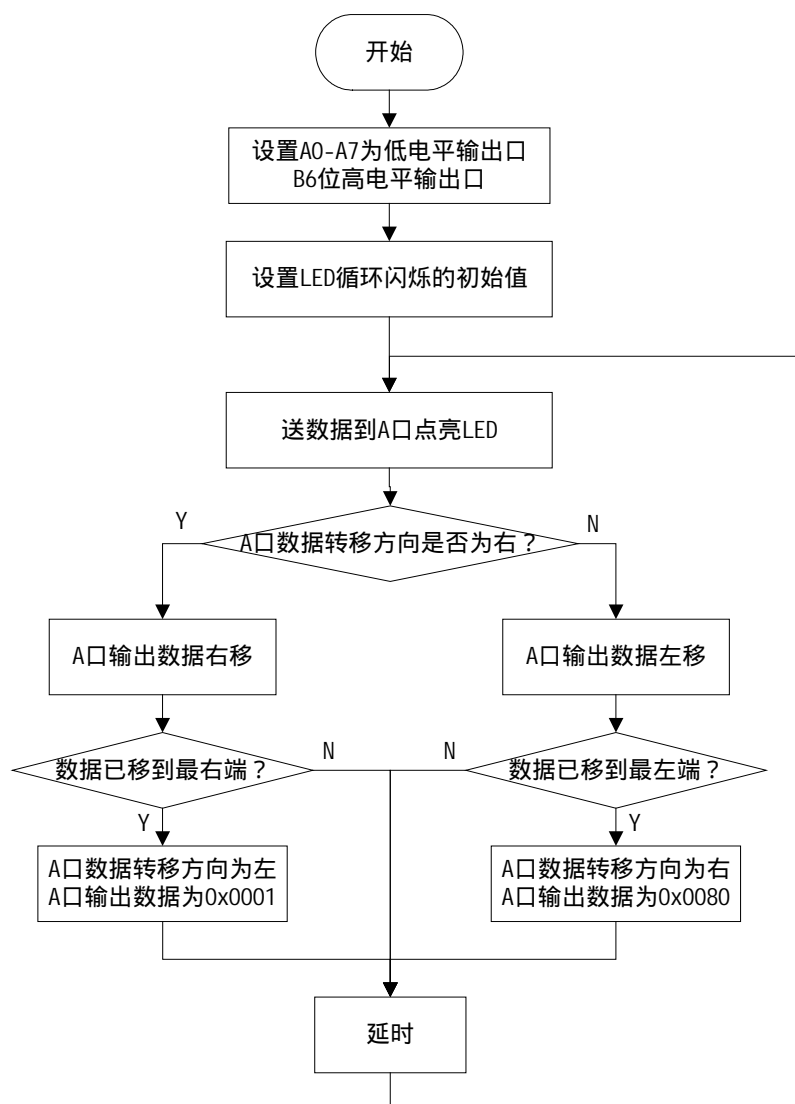
1. 用 10 针排线将 61 板的 J8 接口 IOA 低 8 位连接到 LED 键盘模组的 SEG 接口管脚上，在将 J6 接口 IOB 低 8 位连接到 LED 键盘模组的 DIG 接口管脚上。
2. 运行参考程序。

【硬件连接图】



IOA 低 8 位接至 SEG 接口控制 LED 的导通，IOB6 连至 DIG7 通过 ULN2003A 控制 8 个 LED 的共阴极电平状态，也可将 DIG7 直接连至 VDD，直接将 LED 共阴极接地，不用程序控制。

【程序流程图】



【程序范例】

汇编语言版：

```

//=====//
// 工程名称：    led2.spj
// 功能描述：    发光二极管双向循环点亮
// IDE 环境：    SUNPLUS u'nSPTM  IDE 1.8.4
//=====//

.include hardware.inc                // 包含 hardware.inc

.ram

.var R_LedControl                    // 定义变量

.var R_DelayCounter

```



```
.var R_LedDir

.code

.public _main

//=====//

// 函数名称：    _main()
// 功能描述：    发光二极管双向循环点亮
//=====//

_main:

    r1 = 0x00FF                // 设置 A0~A7 口为同相低电平输出

    [P_IOA_Dir] = r1

    [P_IOA_Attrib] = r1

    r1 = 0x0000

    [P_IOA_Data] = r1

    r1=0x0040

    [P_IOB_Dir]=r1

    [P_IOB_Attrib]=r1

    [P_IOB_Data]=r1           // B6 输出高电平，保证 LED 阴极接地

    r1=0x0001

    [R_LedControl] = r1       // LED 点亮控制初值

    r1 = 0x0000

    [R_LedDir] = r1;

L_MainLoop:

    r1 = [R_LedControl]

    [P_IOA_Data] = r1         // 送数据到 A 口

    r1 = [R_LedDir]

    cmp r1,0x0001

    je MoveR                 // 反向右移

    r1 = [R_LedControl]

    r1 = r1 LSL 1             // 修改 LED 点亮控制参数

    cmp r1,0x0100            // 只有 8 个 LED，保证输出数据在 8 位以内

    jne NoOver
```

```

    r1 = 0x0001                                // 正向左移完毕，移位方向改变

    [R_LedDir] = r1;

    r1 = [R_LedControl]                        // [R_LedControl]数据不变

NoOver:

    [R_LedControl] = r1

    r1 = 0x0000                                // 延时计数器清零

    [R_DelayCounter] = r1

    call F_Delay                                // 延时

    jmp L_MainLoop                            // 跳转到 L_MainLoop 循环，点亮 LED

MoveR:

    r1 = [R_LedControl]

    r1 = r1 LSR 1                                // 修改 LED 点亮控制参数

    cmp r1, 0x0000

    jne NoOver

    r1 = 0x0000                                // 反向右移完毕，移位方向改变

    [R_LedDir] = r1;

    r1 = [R_LedControl]                        // [R_LedControl]数据不变

    jmp NoOver                                // 跳转到延时函数

//=====//

// 函数名称：    F_Delay()

// 功能描述：    实现延时

//=====//

F_Delay:

L_DelayLoop:

    r1=0x0001                                // 清看门狗

    [P_Watchdog_Clear] = r1

    r1 = [R_DelayCounter]

    r1 += 1                                // 延时计数加 1

    [R_DelayCounter] = r1

    jnz L_DelayLoop                            // 加到 65536 吗？

retf

```



C 语言版：

```
#define P_IOA_Data      (volatile unsigned int *)0x7000

#define P_IOA_Buffer    (volatile unsigned int *)0x7001

#define P_IOA_Dir       (volatile unsigned int *)0x7002

#define P_IOA_Attrib    (volatile unsigned int *)0x7003

#define P_IOB_Data      (volatile unsigned int *)0x7005

#define P_IOB_Buffer    (volatile unsigned int *)0x7006

#define P_IOB_Dir       (volatile unsigned int *)0x7007

#define P_IOB_Attrib    (volatile unsigned int *)0x7008

#define P_Watchdog_Clear (volatile unsigned int *)0x7012

//=====//

// 函数名称：    Delay()
// 功能描述：    实现延时

//=====//

void Delay()

{
    // 延时子程序

    unsigned int i;

    for (i=0; i<32768; i++)

    {

        *P_Watchdog_Clear=0x0001;    // 清看门狗

    }

}

//=====//

// 函数名称：    int main()
// 功能描述：    双向循环点亮

//=====//

int main()

{

    int LedControl = 0x0001;

    int LedDir = 0x0000;

    *P_IOA_Dir = 0x00ff;    // 设置 A 口低 8 位为同向低输出

    *P_IOA_Attrib = 0x00ff;
```



```

*P_IOA_Data = 0x0000;

*P_IOB_Dir=0x0040;           // 设置 B6 口为高电平输出，保证 LED 共阴极接地
*P_IOB_Attrib=0x0040;
*P_IOB_Data=0x0040;

while(1)
{
    *P_IOA_Data = LedControl;           // 送数据到 A 口
    if (LedDir == 1)                   // 反向右移循环
    {
        LedControl = LedControl >> 1;

        if (LedControl == 0x0000)       // 反向循环完毕
        {
            LedDir = 0x0000;
            LedControl = 0x0001;
        }
    }
    else
    {
        // 正向左移循环
        LedControl = LedControl << 1;

        if (LedControl == 0x0100)       // 正向循环完毕
        {
            LedDir = 0x0001;
            LedControl = LedControl >> 1;
        }
    }
    Delay();
}
}

```

【程序练习】

在 $\mu'nSP^TM$ IDE 下用汇编语言和 C 语言编写隔一或者隔二发光二极管双向循环点亮的程序。



```
//=====//  
EX2    END  
//=====//
```

实验三 按键点亮发光二极管

【实验目的】

1. 熟悉 $\mu'nSPTM$ IDE 环境及在该环境下用汇编和 C 语言编写的应用程序。
2. 熟悉简单的 $\mu'nSPTM$ 汇编语言指令。
3. 以 A 口和 B 口为例，学会使用 SPCE061A 单片机 I/O 口的基本输出和输入功能。

【实验设备】

1. 装有 Windows 系统和 $\mu'nSPTM$ IDE 仿真环境的 PC 机一台。
2. 61 板一套；LED 键盘模组一套；10 针排线两根。

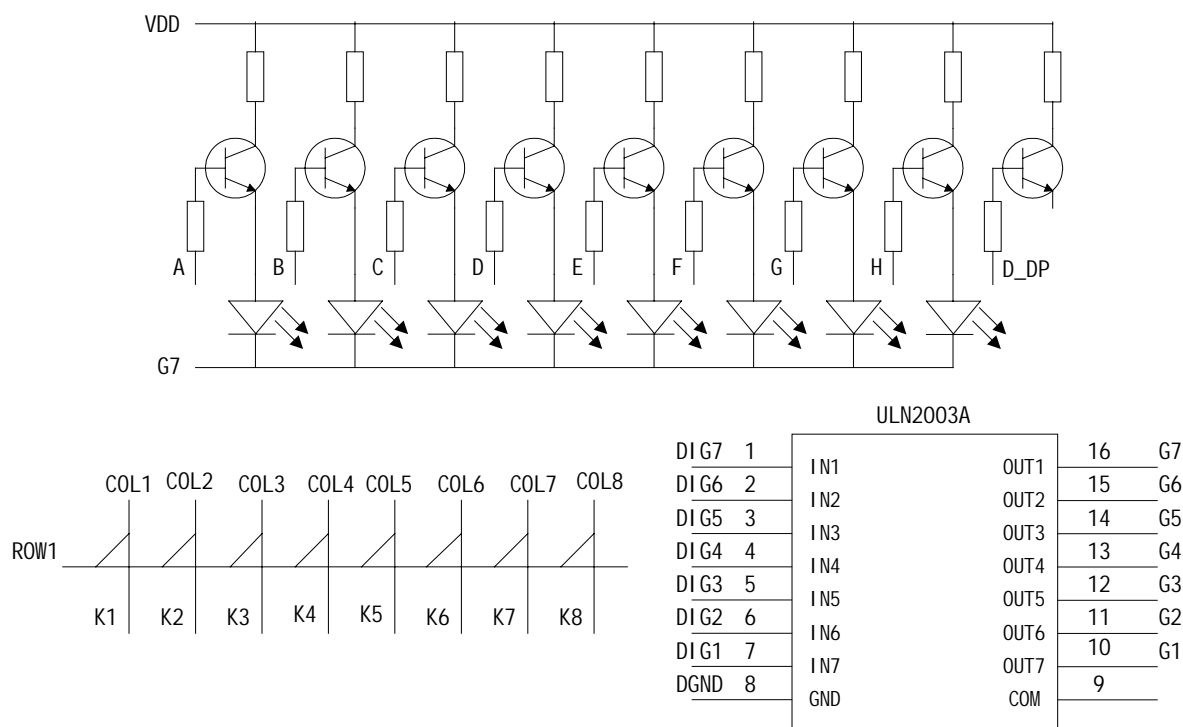
【实验说明】

1. 61 板的 I/O 输出实验主要以 IOA0~7 接 LED 键盘模组上的 8 个发光二极管，IOA8~IOA15 接键盘 K1~K8 的输入。因 61 板核心芯片 SPCE061A 已内置上下拉电阻，所以端口直接连接发光二极管的驱动端。
2. 实验结果是按键后相应发光二极管点亮。
3. 代码编写上，主要涉及 SPCE061A 的端口寄存器 IOA 和 IOB。

【实验步骤】

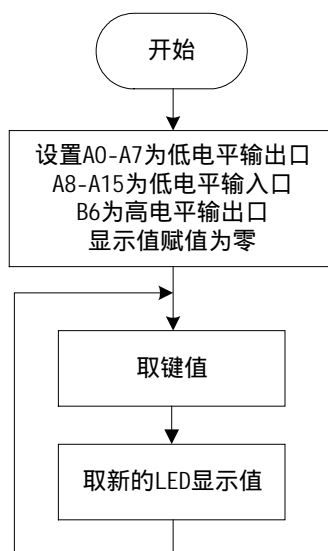
1. 用 10 针排线将 61 板的 J8 接口 IOA 低 8 位连接到 LED 键盘模组的 SEG 接口管脚上，将 J6 接口 IOB 低 8 位连接到 LED 键盘模组的 DIG 接口管脚上，将 J9 接口 IOA 高 8 位连接到 LED 键盘模组的 1*8KEY 接口管脚上，将 1*8KEY 接口中 ROW1 接至 DGND 上。
2. 运行参考程序。

【硬件连接图】



IOA 低 8 位接至 SEG 接口控制 LED 的导通，IOA 高 8 位连接到 LED 键盘模块的 1*8KEY 接口管脚上，读取相应按键值，1*8key 中 ROW1 与 DGND 相连，IOB6 连至 DIG7 通过 ULN2003A 控制 8 个 LED 的共阴极电平状态，也可将 DIG7 直接连至 VDD，直接将 LED 共阴极接地，不用程序控制。

【程序流程图】



【程序范例】

汇编语言版：



```
.include hardware.inc // 包含 hardware.inc

.external F_GetKey

.ram

.var R_key // 定义变量

.code

.public _main

//=====//

// 函数名称： _main
// 日 期： 20040816
// 功能描述： 按键点亮发光二极管
// 语法格式： void main(void)
// 入口参数： 无
// 出口参数： 无
// 注意事项： 仅为用户模型

//=====//

_main:

    r1 = 0x00FF // 设置 A0~A7 口为同相低电平输出
    [P_IOA_Dir] = r1 // A8 ~A15 为高电平输入
    [P_IOA_Attrib] = r1
    r1 = 0xff00
    [P_IOA_Data] = r1
    r1=0x0040
    [P_IOB_Dir]=r1
    [P_IOB_Attrib]=r1
    [P_IOB_Data]=r1 // B6 输出高电平，保证 LED 阴极接地

L_MainLoop:

    call F_GetKey // 取键值
    [R_key] = r1
    r1 = r1 lsr 4 // 取新的显示数据
    r1 = r1 lsr 4
    [P_IOA_Data] = r1
```

```

    jmp L_MainLoop

//=====//

功能：取键值

//=====//

#include hardware.inc

#define KEY_ALL 0xff00                // 使用 IOA8~IOA15 作为键盘输入口

//=====//

// 函数名称：    F_GetKey
// 功能描述：    等待直到有键按下并抬起，返回键值，没有去抖处理
// 输    出：    由 r1 返回 16 位键值

//=====//

.public F_GetKey;

.code

F_GetKey:
    push r2 to [sp]

    r1=[P_IOA_Dir]                //初始化 IOA 的相应端口为上拉输入
    r1&=~KEY_ALL
    [P_IOA_Dir]=r1
    r1=[P_IOA_Attrib]
    r1&=~KEY_ALL
    [P_IOA_Attrib]=r1
    r1=[P_IOA_Buffer]
    r1|=KEY_ALL
    [P_IOA_Buffer]=r1

L_WaitKeyDown:                    //等待有键按下，即有端口变为 0
    r1=1

    [P_Watchdog_Clear]=r1        //清看门狗
    r1=[P_IOA_Data]
    r1&=KEY_ALL
    r1^=KEY_ALL                  //取反
    jz    L_WaitKeyDown          //如果 r1 为 0 说明没有键按下，继续等待

```



```
L_WaitKeyUp:           //如果有键按下则等待该键抬起

    r2=1

    [P_Watchdog_Clear]=r2

    r2=[P_IOA_Data]

    r2&=KEY_ALL

    r2^=KEY_ALL

    jnz L_WaitKeyUp

    pop r2 from [sp]

    retf
```

C 语言版：

```
#define P_IOA_Data      (volatile unsigned int *)0x7000

#define P_IOA_Buffer    (volatile unsigned int *)0x7001

#define P_IOA_Dir       (volatile unsigned int *)0x7002

#define P_IOA_Attrib    (volatile unsigned int *)0x7003

#define P_IOB_Data      (volatile unsigned int *)0x7005

#define P_IOB_Buffer    (volatile unsigned int *)0x7006

#define P_IOB_Dir       (volatile unsigned int *)0x7007

#define P_IOB_Attrib    (volatile unsigned int *)0x7008

#define P_Watchdog_Clear (volatile unsigned int *)0x7012

extern unsigned GetKey(void);

//=====//

// 函数名称：    int main()

// 日    期：    20040816

// 功能描述：    按键点亮发光二极管

// 语法格式      int main()

// 入口参数：    无

// 出口参数：    无

// 注意事项：    仅为用户模型

//=====//
```

```

int main()
{
    unsigned Key = 0x0000;

    *P_IOA_Dir = 0x00ff;           //设置 A 口低 8 位为同向低输出，高 8 位为同向上拉输入
    *P_IOA_Attrib = 0x00ff;
    *P_IOA_Data = 0xff00;
    *P_IOB_Dir=0x0040;           //设置 B6 口为高电平输出，保证 LED 共阴极接地
    *P_IOB_Attrib=0x0040;
    *P_IOB_Data=0x0040;

    while(1)
    {
        Key = GetKey();           // 取键值
        Key = Key >> 8;           // 取 LED 显示初值
        *P_IOA_Data = Key;
    }
}

//=====//
// 功能：取键值
//=====//

#define P_IOA_Data      (volatile unsigned int *)0x7000
#define P_IOA_Buffer    (volatile unsigned int *)0x7001
#define P_IOA_Dir      (volatile unsigned int *)0x7002
#define P_IOA_Attrib    (volatile unsigned int *)0x7003
#define P_Watchdog_Clear (volatile unsigned int *)0x7012
#define KEY_ALL 0xff00      // 使用 IOA8~IOA15 作为键盘输入口

unsigned GetKey(void)
{
    unsigned KeyValue = 0x0000;

```



```
//初始化 IOA 的相应端口为上拉输入

*P_IOA_Dir&=~KEY_ALL;

*P_IOA_Attrib&=~KEY_ALL;

*P_IOA_Buffer|=KEY_ALL;


//等待有键按下，即有端口变为 0

while(KeyValue==0)

{

    KeyValue>(*P_IOA_Data & KEY_ALL)^KEY_ALL;

    *P_Watchdog_Clear=1;        //清看门狗

}

KeyValue>(*P_IOA_Data&KEY_ALL)^KEY_ALL;

//等待按键抬起

while((*P_IOA_Data&KEY_ALL)^KEY_ALL)

{

    *P_Watchdog_Clear=1;

}

return KeyValue;

}
```

【程序练习】

在 $\mu'nSP^TM$ IDE 下用汇编语言和 C 语言编写按键熄灭相应发光二极管程序。

```
//=====//

EX3    END

//=====//
```

实验四 键控发光二极管循环点亮

【实验目的】

1. 熟悉 $\mu'nSP^TM$ IDE 环境及在该环境下用汇编和 C 语言编写的应用程序。
2. 熟悉简单的 $\mu'nSP^TM$ 汇编语言指令。
3. 以 A 口和 B 口为例，学会使用 SPCE061A 单片机 I/O 口的基本输出和输入功能。

【实验设备】

1. 装有 Windows 系统和 $\mu'nSPTM$ IDE 仿真环境的 PC 机一台。
2. 61 板一套；LED 键盘模组一套；10 针排线两根。

【实验说明】

1. 61 板的 I/O 输出实验主要以 IOA0 - 7 接 LED 键盘模组上的 8 个发光二极管, IOA8~IOA15 接键盘 K1~K8 的输入, 将 J6 接口 IOB 低 8 位连接到 LED 键盘模组的 DIG 接口管脚上, 将 1*8KEY 接口中 ROW1 接至 VDD 上。因 61 板核心芯片 SPCE061A 已内置上下拉电阻, 所以端口直接连接发光二极管的驱动端。

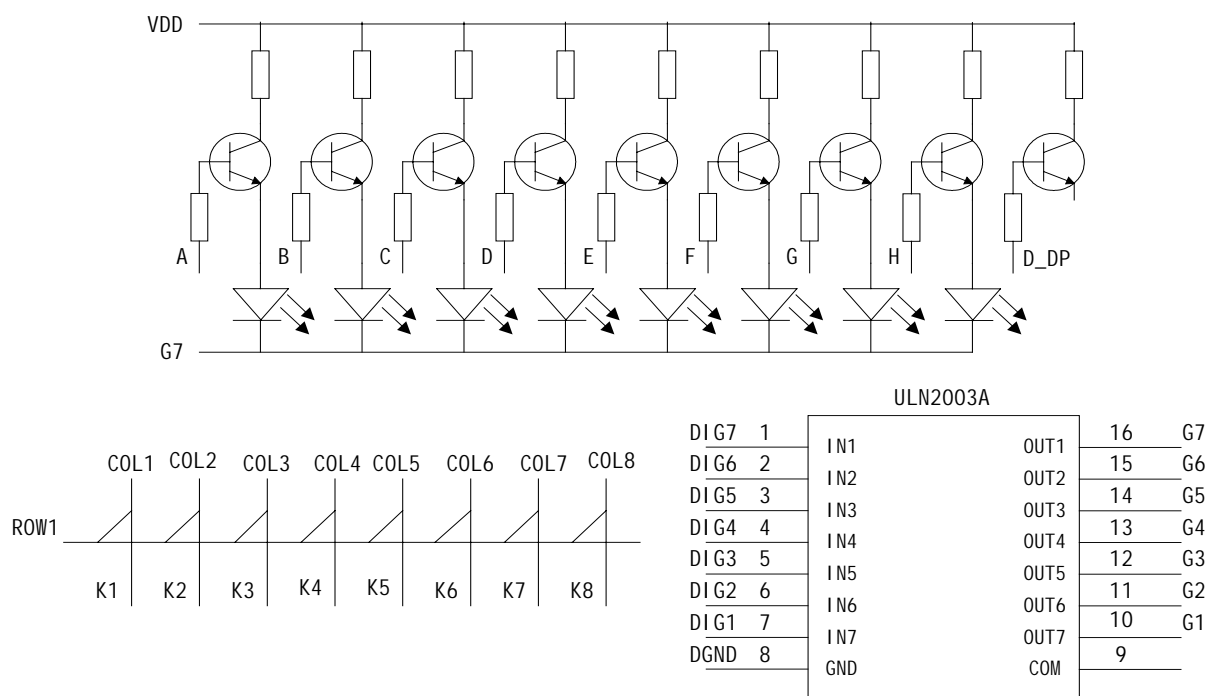
2. 实验的结果是实现按键后从相应按键对应的发光二极管开始循环点亮。
3. 代码编写上, 主要涉及 SPCE061A 的端口寄存器 IOA 和 IOB。

【实验步骤】

1. 用 10 针排线将 61 板的 J8 接口 IOA 低 8 位连接到 LED 键盘模组的 SEG 接口管脚上, 将 J6 接口 IOB 低 8 位连接到 LED 键盘模组的 DIG 接口管脚上, 将 J9 接口 IOA 高 8 位连接到 LED 键盘模组的 1*8KEY 接口管脚上, 将 1*8KEY 接口中 ROW1 接至 VDD 上。

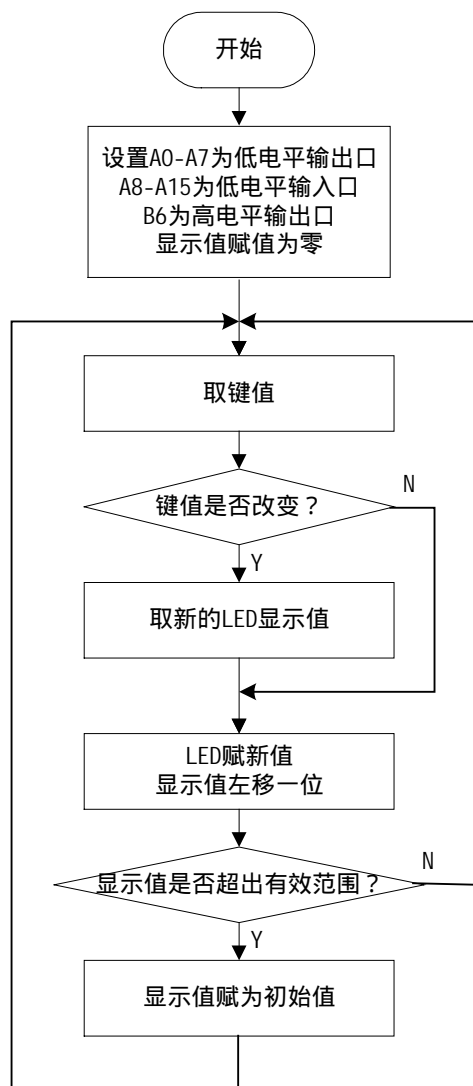
2. 运行参考程序。

【硬件连接图】



IOA 低 8 位接至 SEG 接口控制 LED 的导通, IOA 高 8 位连接到 LED 键盘模组的 1*8KEY 接口管脚上, 读取相应按键值, 1*8key 中 ROW1 与 DGND 相连, IOB6 连至 DIG7 通过 ULN2003A 控制 8 个 LED 的共阴极电平状态, 也可将 DIG7 直接连至 VDD, 直接将 LED 共阴极接地, 不用程序控制。

【程序流程图】



【程序范例】

汇编语言版：

```

        .include hardware.inc                // 包含 hardware.inc

        .external F_GetKey

        .ram

        .var R_key                          // 定义变量

        .var R_LedControl

        .code

        .public _main

//=====//

// 函数名称：      _main
    
```

```

// 日期：      20040816

// 功能描述：    键控发光二极管循环点亮

// 语法格式：    _main

// 入口参数：    无

// 出口参数：    无

// 注意事项：    仅为用户模型

//=====================================================//

_main:

    r1 = 0x00FF                                // 设置 A0~A7 口为同相低电平输出

    [P_IOA_Dir] = r1

    [P_IOA_Attrib] = r1

    r1 = 0xff00

    [P_IOA_Data] = r1

    r1=0x0040

    [P_IOB_Dir]=r1

    [P_IOB_Attrib]=r1

    [P_IOB_Data]=r1                            // B6 输出高电平，保证 LED 阴极接地

    r1 = 0

    [R_LedControl] = r1

L_MainLoop:

    call F_GetKey                              // 取键值

    [R_key] = r1

    cmp r1, 0x0000

    je L_LedLoop

    r1 = r1 lsr 4                              // 取新的显示数据

    r1 = r1 lsr 4

    [R_LedControl] = r1

L_LedLoop:

    r1 = [R_LedControl]

    [P_IOA_Data] = r1                          // 送数据到 A 口

    r1 = r1 LSL 1                             // 修改 LED 点亮控制参数

```



```

    cmp r1,0x0100                                // 只有 8 个 LED，保证输出数据在 8 位以内

    jne L_LedValue

    r1 = 0x0001                                // Led 显示初始值

L_LedValue                                     // Led 下一次显示值

    [R_LedControl] = r1

    jmp L_MainLoop                             // 跳转到 L_MainLoop 循环

//=====//

// 取键值程序如下：

//=====//

.include hardware.inc

#define KEY_ALL 0xff00                        // 使用 IOA8~IOA15 作为键盘输入口

.ram

.var Delay = 0

//=====//

// 函数名称：      F_GetKey
// 语    法：      unsigned F_GetKey(void)
// 功能描述：      等待直到有键按下并抬起，返回键值，没有去抖处理
// 输    出：      由 r1 返回 16 位键值

//=====//

.public F_GetKey;

.code

F_GetKey:

    push r2 to [sp]

    r1=[P_IOA_Dir]                            //初始化 IOA 的相应端口为上拉输入

    r1&=~KEY_ALL

    [P_IOA_Dir]=r1

    r1=[P_IOA_Attrib]

    r1&=~KEY_ALL

    [P_IOA_Attrib]=r1

    r1=[P_IOA_Buffer]

```

```

r1|=KEY_ALL

[P_IOA_Buffer]=r1

r3=0x3000

L_WaitKeyDown:                                //等待有键按下，即有端口变为 0

    r3-=1

    jz L_WaitKeyUp                            //延时没有检测到按键，返回

    r1=1

    [P_Watchdog_Clear]=r1                    //清看门狗

    r1=[P_IOA_Data]

    r1&=KEY_ALL

    r1^=KEY_ALL                            //取反

    jz    L_WaitKeyDown                      //如果 r1 为 0 说明没有键按下，继续等待

L_WaitKeyUp:                                //如果有键按下则等待该键抬起

    r2=1

    [P_Watchdog_Clear]=r2

    r2=[P_IOA_Data]

    r2&=KEY_ALL

    r2^=KEY_ALL

    jnz L_WaitKeyUp

    pop r2 from [sp]

    retf

```

C 语言版：

```

//=====//

#define P_IOA_Data        (volatile unsigned int *)0x7000

#define P_IOA_Buffer      (volatile unsigned int *)0x7001

#define P_IOA_Dir         (volatile unsigned int *)0x7002

#define P_IOA_Attrib      (volatile unsigned int *)0x7003

#define P_IOB_Data        (volatile unsigned int *)0x7005

#define P_IOB_Buffer      (volatile unsigned int *)0x7006

#define P_IOB_Dir         (volatile unsigned int *)0x7007

#define P_IOB_Attrib      (volatile unsigned int *)0x7008

```



```
#define P_Watchdog_Clear    (volatile unsigned int *)0x7012

extern unsigned GetKey(void);

#define KEY_ALL 0xff00      // 使用 IOA8~IOA15 作为键盘输入口

//=====//

// 函数名称：      int main()

// 功能描述：      键控发光二极管循环点亮

// 语法格式：      int main()

//=====//

int main()

{

    unsigned Key = 0x0000;

    int LedControl = 0x0000;

    *P_IOA_Dir = 0x00ff;          // 设置 A 口低 8 位为同向低输出

    *P_IOA_Attrib = 0x00ff;      // 设置 A 口高 8 位为同向高输入

    *P_IOA_Data = 0xff00;

    *P_IOB_Dir=0x0040;          // 设置 B6 口为高电平输出，保证 LED 共阴极接地

    *P_IOB_Attrib=0x0040;

    *P_IOB_Data=0x0040;

    while (1)

    {

        Key = GetKey();          // 取键值

        if (Key != 0)            // 重新按键

        {

            Key = Key >> 8;      // 取 LED 显示初值

            LedControl = Key;

        }

        *P_IOA_Data = LedControl; // LED 循环显示

        LedControl = LedControl << 1;

        if(LedControl == 0x100)  // 保证输出数据在有效范围以内

            LedControl = 0x0001;

    }

}
```

```

}

//=====//

// 功能：取键值

//=====//

#define P_IOA_Data      (volatile unsigned int *)0x7000

#define P_IOA_Buffer     (volatile unsigned int *)0x7001

#define P_IOA_Dir        (volatile unsigned int *)0x7002

#define P_IOA_Attrib     (volatile unsigned int *)0x7003

#define P_Watchdog_Clear (volatile unsigned int *)0x7012

#define KEY_ALL 0xff00    // 使用 IOA8~IOA15 作为键盘输入口

unsigned GetKey(void)
{
    unsigned KeyValue = 0x0000;

    unsigned KeyNumber = 0x0000;

    *P_IOA_Dir&=~KEY_ALL;                //初始化 IOA 的相应端口为上拉输入

    *P_IOA_Attrib&=~KEY_ALL;

    *P_IOA_Buffer|=KEY_ALL;

    while(KeyValue==0)                    //等待有键按下，即有端口变为 0
    {
        KeyNumber++;

        *P_Watchdog_Clear=1;            //清看门狗

        KeyValue=(*P_IOA_Data&KEY_ALL)^KEY_ALL;

        if(KeyNumber==0x1000)
            break;
    }

    while((*P_IOA_Data&KEY_ALL)^KEY_ALL)    //等待按键抬起
    {
        *P_Watchdog_Clear=1;
    }

    return KeyValue;
}

```

}

【程序练习】

在 $\mu'nSPTM$ IDE 下用汇编语言和 C 语言编写键控发光二极管点亮方向程序，按键 K1~K4 向左循环点亮，按键 K5~K8 向右循环点亮。

```

//=====//
EX4  END
//=====//
    
```

实验五 数码管显示 0 - 9

【实验目的】

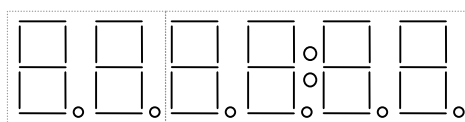
1. 了解 SPCE061A 控制 LED 数码管显示的方法。
2. 进一步熟悉 $\mu'nSPTM$ 汇编语言和 C 语言程序设计。

【实验设备】

1. 装有 Windows 系统和 $\mu'nSPTM$ IDE 仿真环境的 PC 机一台。
2. 61 板一套；LED 键盘模组一套；10 针排线两根。

【实验说明】

LED 键盘模组包含两组共 6 个 LED 数码管，如图所示：



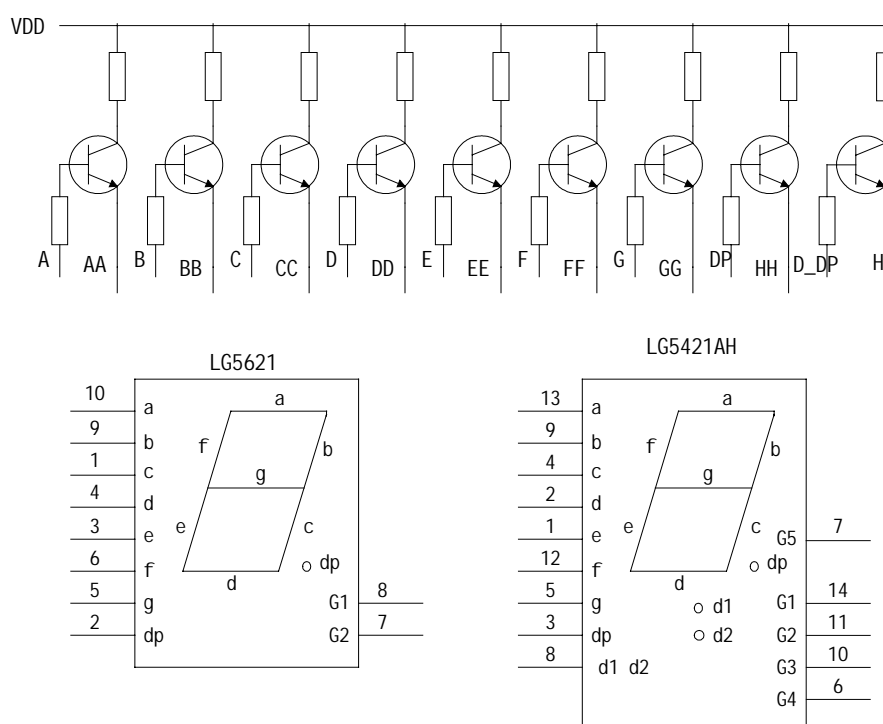
要实现每个数码管分别显示不同数字的效果，例如，要显示“123456”，则需要不断地快速刷新数码管的显示状态，即先使第一个数码管显示数字“1”，其他数码管熄灭，然后熄灭第一个数码管，第二个数码管显示数字“2”，以此类推。这样，利用人眼的视觉残留效应，就产生了“123456”同时显示出来的效果。

【实验步骤】

1. 将 61 板的 IOB0 - 7 用排线连接到 LED 键盘模组的 SEG 排针上；IOB8~IOB15 连接到 LED 键盘模组的 DIG 排针上。
2. 运行参考程序。

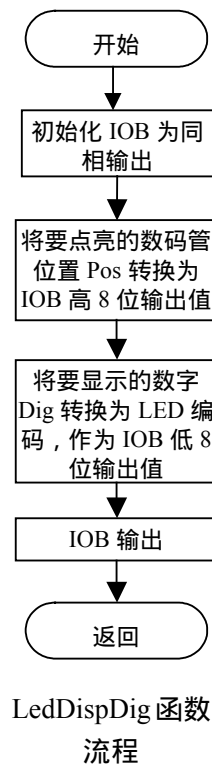
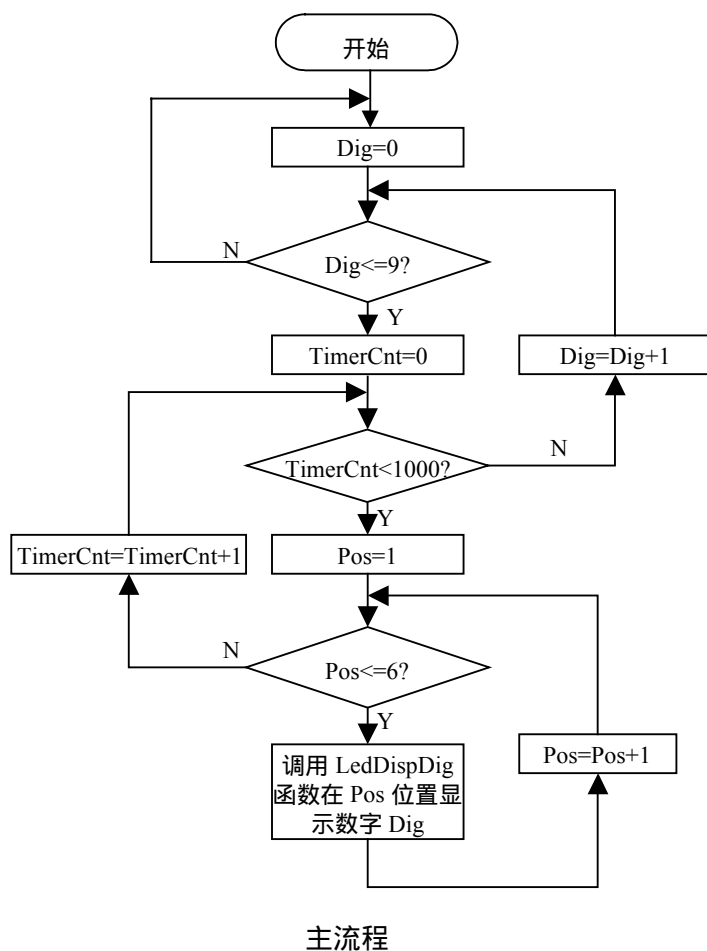
【硬件连接图】

其原理图如下，其中 A~G 与 DP 在 LED 键盘模组中由“SEG”排针引出，G1~G6 则对应 LED 键盘模组“DIG”排针的“DIG1”~“DIG6”端口，时钟冒号 D_DP 在“2*4KEY”排针中可以找到。



单片机的 I/O 控制 G1~G6 可以选 6 个 LED 数码管中的一个或几个点亮，而控制 A~G 以及 DP 则可以设定数码管显示的图形。

【程序流程图】



【程序范例】

汇编语言版：

```

.include hardware.inc

.define LED_SEG 0x00ff           // IOB0~IOB7
.define LED_DIG 0x3f00           // IOB8~IOB13

.ram
.var Pos,Dig,TimerCnt

.data
DigCode:
    
```

```

.dw 0x3F,0x06,0x5B,0x4F,0x66

.dw 0x6D,0x7D,0x07,0x7F,0x6F           // 0~9 十个数字的 LED 编码

.code

//=====//

// 函数名称：      LedDispDig
// 功能描述：      在指定的数码管上显示数字
// 语   法：      void LedDispDig (int Pos,int Dig)
// 输   入：      Pos: 要显示数字的数码管位置，取值范围 1~6
//
//                  Dig: 要显示的数字，取值范围 0~9
// 输   出：      无

//=====//

.public LedDispDig
LedDispDig:
    push r1,r2 to [sp]

    r1=1
    [P_Watchdog_Clear]=r1
    // 初始化 IOB 为同相输出
    r1=[P_IOB_Dir]
    r1|=LED_DIG+LED_SEG
    [P_IOB_Dir]=r1
    r1=[P_IOB_Attrib]
    r1|=LED_DIG+LED_SEG
    [P_IOB_Attrib]=r1

    // 将数字的位置转换为 IOB 高 8 位值，选中相应数码管
    r1=0x0080
    r2=[Pos]           // 将 0x0080 左移(Pos)位
L_SetIOBHigh:
    r1=r1 LSL 1
  
```



```
r2--1

jnz L_SetIOBHigh

[P_IOB_Data]=r1


// 将数字转换为编码，作为 IOB 低 8 位输出

r1=[Dig]

r2=DigCode

r2+=r1

r1=[r2]

r2=[P_IOB_Buffer]

r2|=r1

[P_IOB_Data]=r2


pop r1,r2 from [sp]

retf


//=====//
// 主函数
//=====//

.public _main

_main:


L_MainLoop:                                // 循环 1：主循环

    r1=0

    [Dig]=r1

L_DigLoop:                                // 循环 2：循环显示 0~9 十个数字

    r1=0

    [TimerCnt]=r1

L_TimerLoop:                              // 循环 3：每组数字保持一段时间

    r1=1

    [Pos]=r1
```

```

L_DispLoop:                                // 循环 4：让 6 个数码管显示相同的数字（即 Dig 的值）

    call LedDispDig

    r1=[Pos]                                // 判断循环 4 是否结束

    r1+=1

    [Pos]=r1

    cmp r1,6

    jna L_DispLoop

    r1=[TimerCnt]                            // 判断循环 3 是否结束

    r1+=1

    [TimerCnt]=r1

    cmp r1,1000

    jb L_TimerLoop

    r1=[Dig]                                // 判断循环 2 是否结束

    r1+=1

    [Dig]=r1

    cmp r1,9

    jna L_DigLoop

    jmp L_MainLoop                            // 返回主循环

retf

```

C 语言版：

```

#define P_IOB_Data      (volatile unsigned int *)0x7005

#define P_IOB_Buffer    (volatile unsigned int *)0x7006

#define P_IOB_Dir       (volatile unsigned int *)0x7007

#define P_IOB_Attrib     (volatile unsigned int *)0x7008

#define P_Watchdog_Clear (volatile unsigned int *)0x7012

#define LED_SEG 0x00ff    // IOB0~IOB7

#define LED_DIG 0x3f00    // IOB8~IOB13

const unsigned char DigCode[10]={0x3F,0x06,0x5B,

    0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F};    // 0~9 十个数字的 LED 编码

```



```
//=====//  
// 函数名称：    LedDispDig  
// 功能描述：    在指定的数码管上显示数字  
// 语  法：    void LedDispDig(int Pos,int Dig)  
// 输  入：    Pos: 要显示数字的数码管位置，取值范围 1~6  
//              Dig: 要显示的数字，取值范围 0~9  
// 输  出：    无  
//=====//  
  
void LedDispDig(int Pos,int Dig)  
{  
    *P_Watchdog_Clear=1;  
    // 初始化 IOB 为同相输出  
    *P_IOB_Dir|=LED_SEG+LED_DIG;  
    *P_IOB_Attrib|=LED_SEG+LED_DIG;  
    *P_IOB_Data=(unsigned)0x0100<<(Pos-1);    // 将数字的位置转换为 IOB 高 8 位值，选中相应的  
                                                // 数码管  
    *P_IOB_Data|=DigCode[Dig];                // 将数字转换为编码，作为 IOB 低 8 位输出  
}  
  
//=====//  
// 主函数  
//=====//  
  
int main()  
{  
    int Pos,Dig,TimerCnt;  
    while(1)  
    {  
        for(Dig=0;Dig<=9;Dig++)                // 循环显示 0~9 十个数字  
        {  
            for(TimerCnt=0;TimerCnt<1000;TimerCnt++)    // 每组数字保持一段时间  
            {  
                LedDispDig(Pos,Dig);  
            }  
        }  
    }  
}
```

```

for(Pos=1;Pos<=6;Pos++)    // 让 6 个数码管显示相同的数字（即 Dig 的值）
{
    LedDispDig(Pos,Dig);
}
}
}
}
}
}
}
}

```

【程序练习】

编写程序控制所有数码管循环显示 0~9~0。

```

//=====//
EX5    END
//=====//

```

实验六 数码管移位循环显示 0 - 9

【实验目的】

1. 熟悉 SPCE061A 控制 LED 数码管显示的方法。
2. 进一步熟悉 $\mu'nSPTM$ 汇编语言和 C 语言程序设计。
3. 实验效果：

由六个 LED 数码管移位循环显示 0~9，即：“012345”，“123456”，“234567”，……，“890123”，“901234”……。

【实验设备】

1. 装有 Windows 系统和 $\mu'nSPTM$ IDE 仿真环境的 PC 机一台。
2. 61 板一套；LED 键盘模组一套；10 针排线两根。

【实验步骤】

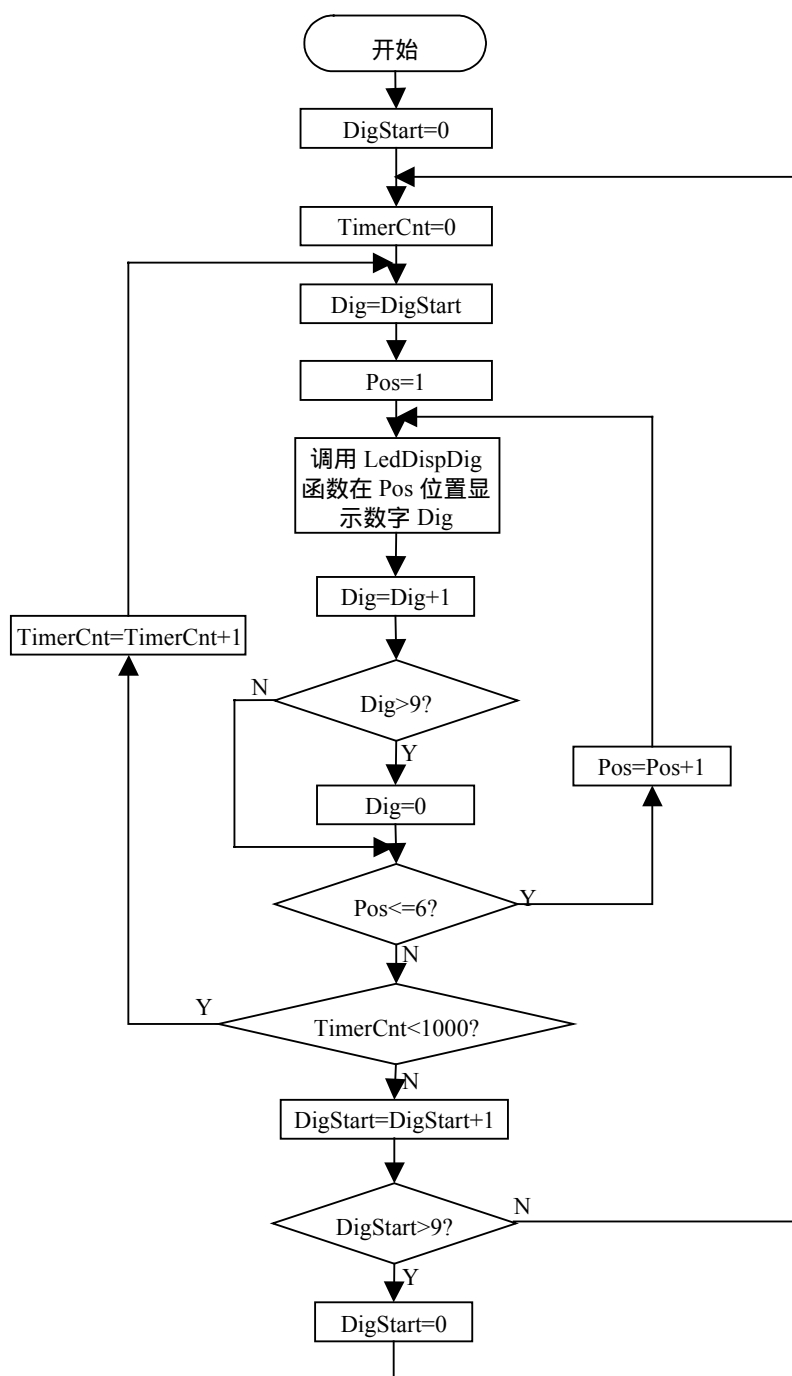
1. 将 61 板的 IOB0 - 7 用排线连接到 LED 键盘模组的 SEG 排针上；IOB8~IOB15 连接到 LED 键盘模组的 DIG 排针上。
2. 运行参考程序。

【硬件连接图】

参见实验五。

【程序流程图】

LedDispDig 函数流程参见实验五。



主流程

【程序范例】

汇编语言版：

```

.include hardware.inc

.define LED_SEG 0x00ff                // IOB0~IOB7
.define LED_DIG 0x3f00                // IOB8~IOB13

.ram

.var Pos,Dig,TimerCnt

.data

DigCode:

.dw 0x3F,0x06,0x5B,0x4F,0x66
.dw 0x6D,0x7D,0x07,0x7F,0x6F          // 0~9 十个数字的 LED 编码

.code

//=====//

// 函数名称：    LedDispDig
// 功能描述：    在指定的数码管上显示数字
// 语   法：    void LedDispDig(int Pos,int Dig)
// 输   入：    Pos: 要显示数字的数码管位置，取值范围 1~6
//              Dig: 要显示的数字，取值范围 0~9
// 输   出：    无

//=====//

(略，见实验五)

//=====//

// 主函数

//=====//

.public _main

_main:

```



```
    r1=0                                // 设置循环的起始数字

L_MainLoop:                            // 循环 1：主循环

    r2=1000                            // 循环 2：每组数字保持一段时间

L_TimerLoop:

    [Dig]=r1

    r3=1

    [Pos]=r3

L_DigLoop:                            // 循环 3：使六个数码管分别显示数字

    call LedDispDig                    // 在第[Pos]个数码管上显示数字[Dig]

    r3=[Dig]

    r3+=1

    cmp r3,9                          // 使数字在 0~9 之间循环

    jna L_DigNext

    r3=0

L_DigNext:

    [Dig]=r3

    r3=[Pos]                          // 判断循环 3 是否结束

    r3+=1

    [Pos]=r3

    cmp r3,6

    jna L_DigLoop

    r2-=1

    jnz L_TimerLoop                  // 判断循环 2 是否结束

    r1+=1                            // 得到下一组数字的起始数字

    cmp r1,9                          // 如果大于 9 则清 0

    jna L_MainLoop

    r1=0

    jmp L_MainLoop
```

```
retf
```

C 语言版：

```
#define P_IOB_Data      (volatile unsigned int *)0x7005
#define P_IOB_Buffer    (volatile unsigned int *)0x7006
#define P_IOB_Dir       (volatile unsigned int *)0x7007
#define P_IOB_Attrib     (volatile unsigned int *)0x7008
#define P_Watchdog_Clear (volatile unsigned int *)0x7012

#define LED_SEG 0x00ff    //IOB0~IOB7
#define LED_DIG 0x3f00    //IOB8~IOB13

const unsigned char DigCode[10]={0x3F,0x06,0x5B,
                                0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F};    // 0~9 十个数字的 LED 编码

//=====//
// 函数名称：    LedDispDig
// 功能描述      在指定的数码管上显示数字
// 语法：        void LedDispDig(int Pos,int Dig)
// 输入：        Pos: 要显示数字的数码管位置，取值范围 1~6
//              Dig: 要显示的数字，取值范围 0~9
// 输出：        无
//=====//
(略，见实验五)
//=====//
// 主函数
//=====//

int main()
{
    int Pos,DigStart,Dig,TimerCnt;
```



```

DigStart=0;                                // 设置循环的起始数字

while(1)
{
    for(TimerCnt=0;TimerCnt<1000;TimerCnt++)    // 每组数字保持一段时间
    {
        Dig=DigStart;

        for(Pos=1;Pos<=6;Pos++)                // 让 6 个数码管分别显示数字（即 Dig 的值）
        {
            LedDispDig(Pos,Dig);

            Dig++;

            if(Dig>9)Dig=0;                      // 使数字在 0~9 之间循环
        }
    }

    DigStart++;                                // 得到下一组数字的起始数字

    if(DigStart>9)DigStart=0;

}
}

```

【程序练习】

编写程序使十个数字在数码管上向右移位循环显示，即“012345”，“901234”，“890123”，……。

```

//=====//
EX6 END
//=====//

```

实验七 发光二极管和数码管交替显示

【实验目的】

1. 熟悉 SPCE061A 控制 LED 数码管显示的方法。
2. 进一步熟悉 $\mu'nSP^{TM}$ 汇编语言和 C 语言程序设计。
3. 实验效果：

发光二极管全部点亮；

发光二极管全部熄灭，数码管全部显示数字 8；

数码管全部熄灭，发光二极管全部点亮，如此反复。

【实验设备】

1. 装有 Windows 系统和 $\mu'nSPT^M$ IDE 仿真环境的 PC 机一台。
2. 61 板一套；LED 键盘模组一套；10 针排线两根。

【实验说明】

发光二极管和数码管的显示数据都由 LED 键盘模组的 SEG 排针控制，而 DIG 排针上的“DIG7”对应发光二极管，“DIG1~DIG6”则对应 6 个数码管。将单片机的 I/O 端口与上述接口连接，即可分别控制发光二极管和各个数码管。

【实验步骤】

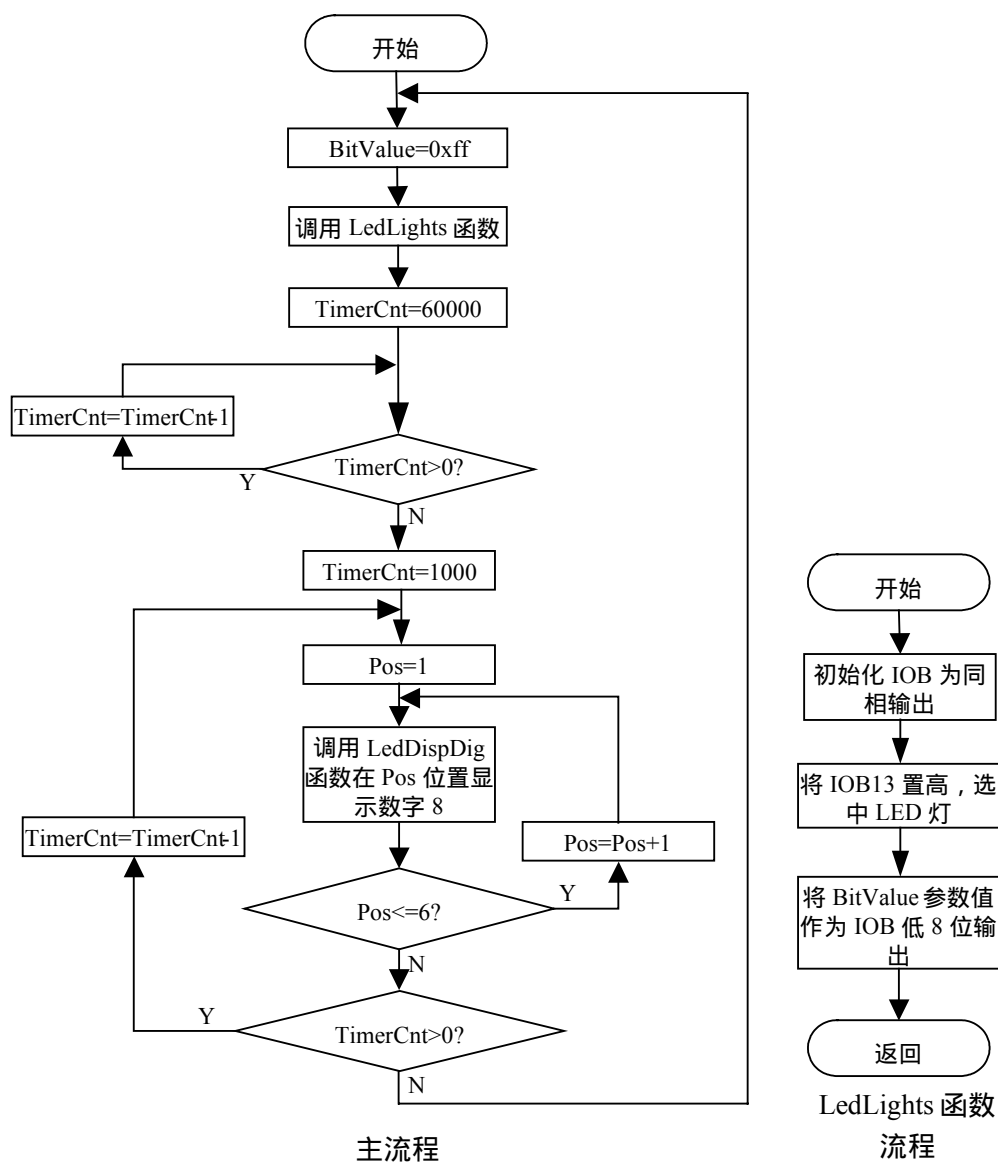
1. 将 61 板的 IOB0 - 7 用排线连接到 LED 键盘模组的 SEG 排针上；IOB8~IOB15 连接到 LED 键盘模组的 DIG 排针上。
2. 运行参考程序

【硬件连接图】

参见实验一和实验五。

【程序流程图】

LedDispDig 函数流程图参见实验五。



【程序范例】

汇编语言版：

```

.include hardware.inc

#define LED_SEG 0x00ff                // IOB0~IOB7 控制数码管或灯的状态
#define LED_DIG 0x3f00                // IOB8~IOB13 分别控制 6 个数码管
#define LED_LIGHTS 0x4000            // IOB14 控制 LED 灯

.ram

.var Pos,Dig,BitValue,TimerCnt
    
```

```
.data

DigCode:

.dw 0x3F,0x06,0x5B,0x4F,0x66
.dw 0x6D,0x7D,0x07,0x7F,0x6F           // 0~9 十个数字的 LED 编码

.code

//=====//

// 函数名称：    LedDispDig
// 功能描述：    在指定的数码管上显示数字
// 语   法：    void LedDispDig(int Pos,int Dig)
// 输   入：    Pos: 要显示数字的数码管位置，取值范围 1~6
//              Dig: 要显示的数字，取值范围 0~9
// 输   出：    无

//=====//

(略，见实验五)

//=====//

// 函数名称：    LedLights
// 语   法：    void LedLights(int BitValue)
// 功能描述：    控制 LED 灯
// 输   入：    BitValue: 8 个 LED 灯的亮灭状态
// 输   出：    无

//=====//

.public LedLights
LedLights:

    push r1 to [sp]

    r1=1

    [P_Watchdog_Clear]=r1

    // 初始化 IOB 为同相输出

    r1=[P_IOB_Dir]
```



```
r1|=LED_LIGHTS+LED_SEG

[P_IOB_Dir]=r1

r1=[P_IOB_Attrib]

r1|=LED_LIGHTS+LED_SEG

[P_IOB_Attrib]=r1


r1=[BitValue]                                // 显示 BitValue 指定的值
r1|=LED_LIGHTS                                // 选中 LED 灯
[P_IOB_Data]=r1


pop r1 from [sp]
retf


//=====//
// 主函数
//=====//

.public _main
_main:

L_MainLoop:                                // 循环 1：主循环
    // 点亮全部 LED 灯
    r1=0xff;
    [BitValue]=r1
    call LedLights

    r1=60000                                // 循环 2：保持 LED 灯点亮一段时间
    [TimerCnt]=r1

L_LightsLoop:
    r1=1
    [P_Watchdog_Clear]=r1

    r1=[TimerCnt]                            // 判断循环 2 是否结束
```



```

    r1--1

    [TimerCnt]=r1

    jnz L_LightsLoop

    r1=1000                                // 循环 3：在一段时间中使所有数码管显示数字“8”

    [TimerCnt]=r1

L_DigTimerLoop:

    r1=1

    [Pos]=r1

L_DigLoop:                                // 循环 4：使六个数码管依次显示数字“8”

    r1=8

    [Dig]=r1

    call LedDispDig

    r1=[Pos]                               q      // 判断循环 4 是否结束

    r1+=1

    [Pos]=r1

    cmp r1,6

    jna L_DigLoop

    r1=[TimerCnt]

    r1--1

    [TimerCnt]=r1

    jnz L_DigTimerLoop                    // 判断循环 3 是否结束

    jmp L_MainLoop                        // 返回主循环

retf

```

C 语言版：

```

#define P_IOB_Data      (volatile unsigned int *)0x7005

#define P_IOB_Buffer    (volatile unsigned int *)0x7006

#define P_IOB_Dir       (volatile unsigned int *)0x7007

```



```
#define P_IOB_Attrib      (volatile unsigned int *)0x7008

#define P_Watchdog_Clear  (volatile unsigned int *)0x7012


#define LED_SEG 0x00ff      // IOB0~IOB7 控制数码管或灯的状态
#define LED_DIG 0x3f00      // IOB8~IOB13 分别控制 6 个数码管
#define LED_LIGHTS 0x4000  // IOB14 控制 LED 灯

const unsigned char DigCode[10]={0x3F,0x06,0x5B,
                                0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F};    // 0~9 十个数字的 LED 编码

//=====//
// 函数名称：    LedDispDig
// 语    法：    void LedDispDig(int Pos,int Dig)
// 功能描述：    在指定的数码管上显示数字
// 输    入：    Pos: 要显示数字的数码管位置，取值范围 1~6
//                Dig: 要显示的数字，取值范围 0~9
// 输    出：    无
//=====//
( 略，见实验五 )
//=====//
// 函数名称：    LedLights
// 语    法：    void LedLights(int BitValue)
// 功能描述：    控制 LED 灯
// 输    入：    BitValue: 8 个 LED 灯的亮灭状态
// 输    出：    无
//=====//
void LedLights(int BitValue)
{
    *P_Watchdog_Clear=1;

    // 初始化 IOB13 为同相输出

    *P_IOB_Dir|=LED_LIGHTS+LED_SEG;
```

```

    *P_IOB_Attrib|=LED_LIGHTS+LED_SEG;

    *P_IOB_Data=LED_LIGHTS;           // 选中 LED 灯

    *P_IOB_Data|=BitValue;           // 显示 BitValue 指定的值
}

// 主函数
//=====//

int main()
{
    unsigned Pos,TimerCnt;

    while(1)                          // 循环 1：主循环
    {
        LedLights(0xff);              //点亮全部 LED 灯

        for(TimerCnt=0;TimerCnt<60000;TimerCnt++) // 保持一段时间
        {
            *P_Watchdog_Clear=1;
        }

        for(TimerCnt=0;TimerCnt<1000;TimerCnt++) // 在一段时间中使所有数码管显示数字“8”
        {
            for(Pos=1;Pos<=6;Pos++)
                LedDispDig(Pos,8);     // 第 j 个数码管显示数字“8”
        }
    }
}

```

【程序练习】

编写程序使发光二极管和数码管同时点亮，同时熄灭。

```

//=====//

EX7    END

//=====//

```



实验八 按键显示数字

【实验目的】

1. 熟悉 SPCE061A 控制键盘和 LED 数码管显示的方法。
2. 进一步熟悉 $\mu'nSPTM$ 汇编语言和 C 语言程序设计。
3. 实验效果：

在数码管上显示按键的值，

按第一个键则显示"1"，按第二个键显示"2"，.....，按第 8 个键显示"8"。

【实验设备】

1. 装有 Windows 系统和 $\mu'nSPTM$ IDE 仿真环境的 PC 机一台。
2. 61 板一套； LED 键盘模组一套； 10 针排线三根。

【实验步骤】

1. 将 LED 键盘模组的"KEYTYPE"跳线跳到"1*8KEY"状态；将 61 板的 IOA8~IOA15 用排线连接到 LED 键盘模组的 1*8KEY 排针上；IOB0~7 连接到 LED 键盘模组的 SEG 排针上；IOB8~IOB15 连接到 LED 键盘模组的 DIG 排针上。

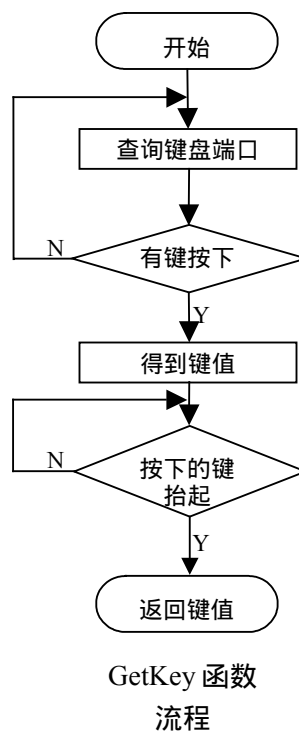
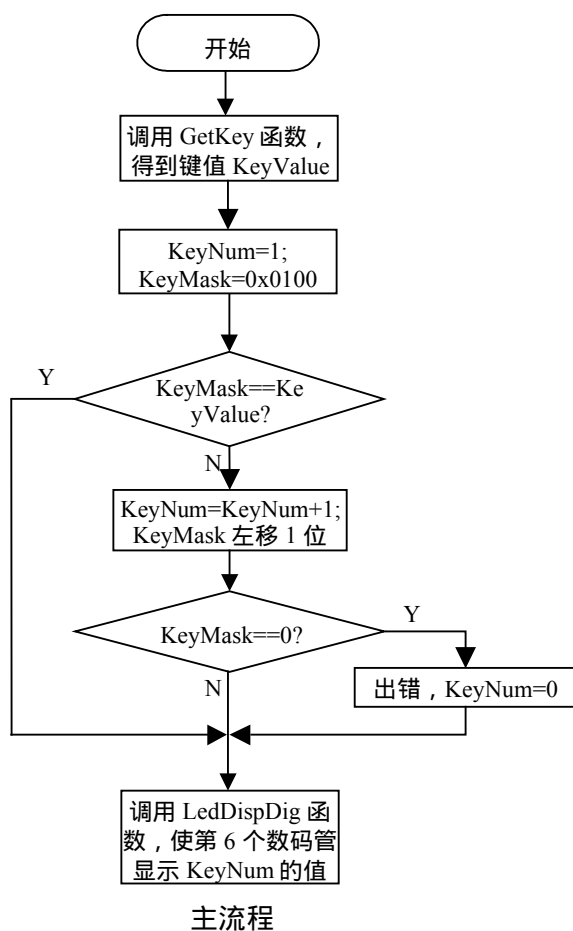
2. 运行参考程序。

【硬件连接图】

参见实验三和实验五。

【程序流程图】

LedDispDig 函数流程图参见实验五。



【程序范例】

汇编语言版：

```

.include hardware.inc

.define LED_SEG 0x00ff           // IOB0~IOB7 控制数码管或灯的状态
.define LED_DIG 0x3f00           // IOB8~IOB13 分别控制 6 个数码管
.define LED_LIGHTS 0x4000        // IOB14 控制 LED 灯
.define KEY_ALL 0xff00           // 使用 IOA8~IOA15 作为键盘输入入口

.ram

.var Pos,Dig

.data

DigCode:

.dw 0x3F,0x06,0x5B,0x4F,0x66
  
```



```
.dw 0x6D,0x7D,0x07,0x7F,0x6F      //0~9 十个数字的 LED 编码

.code

//=====//

// 函数名称：    LedDispDig
// 功能描述：    在指定的数码管上显示数字
// 语    法：    void LedDispDig(int Pos,int Dig)
// 输    入：    Pos: 要显示数字的数码管位置，取值范围 1~6
//                Dig: 要显示的数字，取值范围 0~9
// 输    出：    无
//=====//

( 略，见实验五 )

//=====//

// 函数名称：    GetKey
// 功能描述：    等待直到有键按下并抬起，返回键值，没有去抖处理
// 语    法：    unsigned GetKey(void)
// 输    入：    无
// 输    出：    由 r1 返回 16 位键值
//=====//

.public GetKey;

.code

GetKey:

    push r2 to [sp]

    r1=[P_IOA_Dir]                // 初始化 IOA 的相应端口为上拉输入
    r1&=~KEY_ALL

    [P_IOA_Dir]=r1

    r1=[P_IOA_Attrib]
    r1&=~KEY_ALL

    [P_IOA_Attrib]=r1

    r1=[P_IOA_Buffer]

    r1|=KEY_ALL
```

```

[P_IOA_Buffer]=r1

L_WaitKeyDown:                                     // 等待有键按下，即有端口变为 0

    r1=1

    [P_Watchdog_Clear]=r1

    r1=[P_IOA_Data]

    r1&=KEY_ALL

    r1^=KEY_ALL                                     // 取反

    jz    L_WaitKeyDown                             // 如果 r1 为 0 说明没有键按下，继续等待

L_WaitKeyUp:                                       // 如果有键按下则等待该键抬起

    r2=1

    [P_Watchdog_Clear]=r2

    r2=[P_IOA_Data]

    r2&=KEY_ALL

    r2^=KEY_ALL

    jnz L_WaitKeyUp

    pop r2 from [sp]

    retf

//=====//

// 主函数

//=====//

.public _main

_main:

L_MainLoop:                                       // 循环 1：主循环

    call GetKey                                     // 得到 16 位键值，保存在 r1

    // 将 16 位键值转换为 1~8 数值

    r2=1

    [Dig]=r2

    r3=0x0100

L_KeyLoop:                                       // 循环 2：从低到高依次检查 IOB8~IOB15 的每一位

```



```

    cmp r3,r1
    je L_KeyNum
    r2+=1
    [Dig]=r2
    r3=r3 lsl 1
    jz L_KeyError          // 8 位全部判断完，没有得到键值，则显示"0"表示出错
    jmp L_KeyLoop

L_KeyError:
    r2=0
    [Dig]=r2

L_KeyNum:
    r2=6                  // 在第六个数码管上显示键代表的数值（1~8）
    [Pos]=r2
    call LedDispDig       // 送显示
    jmp L_MainLoop        // 返回主循环
retf

```

C 语言版：

```

#define P_IOA_Data      (volatile unsigned int *)0x7000
#define P_IOA_Buffer    (volatile unsigned int *)0x7001
#define P_IOA_Dir       (volatile unsigned int *)0x7002
#define P_IOA_Attrib    (volatile unsigned int *)0x7003
#define P_IOB_Data      (volatile unsigned int *)0x7005
#define P_IOB_Buffer    (volatile unsigned int *)0x7006
#define P_IOB_Dir       (volatile unsigned int *)0x7007
#define P_IOB_Attrib    (volatile unsigned int *)0x7008
#define P_Watchdog_Clear (volatile unsigned int *)0x7012
#define LED_SEG 0x00ff  // IOB0~IOB7 控制数码管或灯的状态
#define LED_DIG 0x3f00  // IOB8~IOB13 分别控制 6 个数码管
#define LED_LIGHTS 0x4000 // IOB14 控制 LED 灯
#define KEY_ALL 0xff00   // 使用 IOA8~IOA15 作为键盘输入口

```



```
const unsigned char DigCode[10]={0x3F,0x06,0x5B,
    0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F};    // 0~9 十个数字的 LED 编码

//=====//
// 函数名称：    LedDispDig
// 功能描述：    在指定的数码管上显示数字
// 语    法：    void LedDispDig(int Pos,int Dig)
// 输    入：    Pos: 要显示数字的数码管位置，取值范围 1~6
//                Dig: 要显示的数字，取值范围 0~9
// 输    出：    无
//=====//
(略，见实验五)
//=====//
// 函数名称：    GetKey
// 功能描述：    等待直到有键按下并抬起，返回键值，没有去抖处理
// 语    法：    unsigned GetKey(void)
// 输    入：    无
// 输    出：    16 位键值
//=====//
unsigned GetKey(void)
{
    unsigned KeyValue;

    // 初始化 IOA 的相应端口为上拉输入
    *P_IOA_Dir&=~KEY_ALL;
    *P_IOA_Attrib&=~KEY_ALL;
    *P_IOA_Buffer|=KEY_ALL;

    //等待有键按下，即有端口变为 0
    while(!((*P_IOA_Data&KEY_ALL)^KEY_ALL))
    {
```



```
*P_Watchdog_Clear=1;

}

KeyValue=(*P_IOA_Data&KEY_ALL)^KEY_ALL;

//等待按键抬起

while((*P_IOA_Data&KEY_ALL)^KEY_ALL)

{

    *P_Watchdog_Clear=1;

}

return KeyValue;

}

//=====//

// 主函数

//=====//

int main()

{

    unsigned KeyValue,KeyNum,KeyMask;

    while(1)

    {

        KeyValue=GetKey();                // 得到 16 位键值

        // 将 16 位键值转换为 1~8 数值

        KeyNum=1;

        KeyMask=0x0100;

        while (KeyMask)

        {

            // 从低到高依次检查 IOB8~IOB15 的每一位

            if(KeyValue==KeyMask)

            {

                break;

            }

            else
```

```

    {
        KeyMask<<=1;
        KeyNum++;
    }

}

if(!KeyMask)KeyNum=0; //如果没得到正确的键值则显示"0"
LedDispDig(6,KeyNum); //在第六个数码管上显示键的代表数值 (1~8)
}
}

```

【程序练习】

编写程序使六个数码管同时显示按键代表的数字。

```

//=====//
EX8    END
//=====//

```

实验九 发光二极管巡回点亮并数码管计数

【实验目的】

1. 熟悉 $\mu'nSPTM$ IDE 环境及在该环境下用汇编和 C 语言编写的应用程序。
2. 熟悉简单的 $\mu'nSPTM$ 汇编语言指令。
3. 以 A 口和 B 口为例,学会使用 SPCE061A 单片机 I/O 口的基本输出和输入功能以及数码管的使用方法。

【实验设备】

1. 装有 Windows 系统和 $\mu'nSPTM$ IDE 仿真环境的 PC 机一台。
2. 61 板一套; LED 键盘模组一套,连接线两根。

【实验说明】

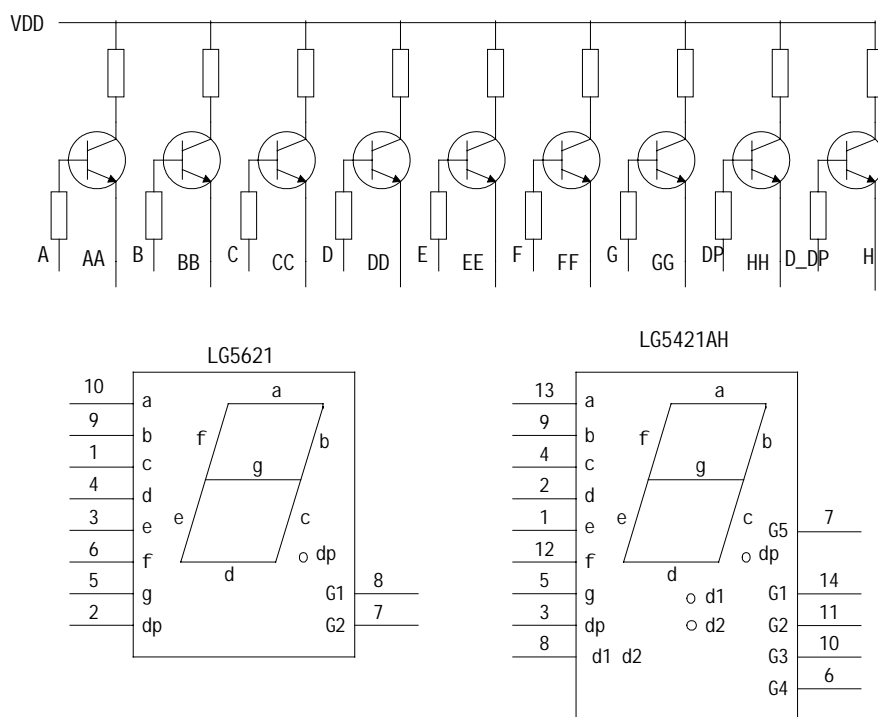
1. 61 板的 I/O 输出实验主要以 A 口低 8 位接 LED 键盘模组上的 8 个发光二极管将 J6 接口 IOB 低 8 位连接到 LED 键盘模组的 DIG 接口管脚上。
2. 实验的结果是 LED 巡回闪烁,同时数码管从左至右显示 1~8,全部显示完成后,LED 停止闪烁,数码管全部显示为 8。
3. 代码编写上,主要涉及显示缓冲区的应用。

【实验步骤】

1. 用 10 针排线将 61 板的 J8 接口 IOA 低 8 位连接到 LED 键盘模组的 SEG 接口管脚上，将 J6 接口 IOB 低 8 位连接到 LED 键盘模组的 DIG 接口管脚上。

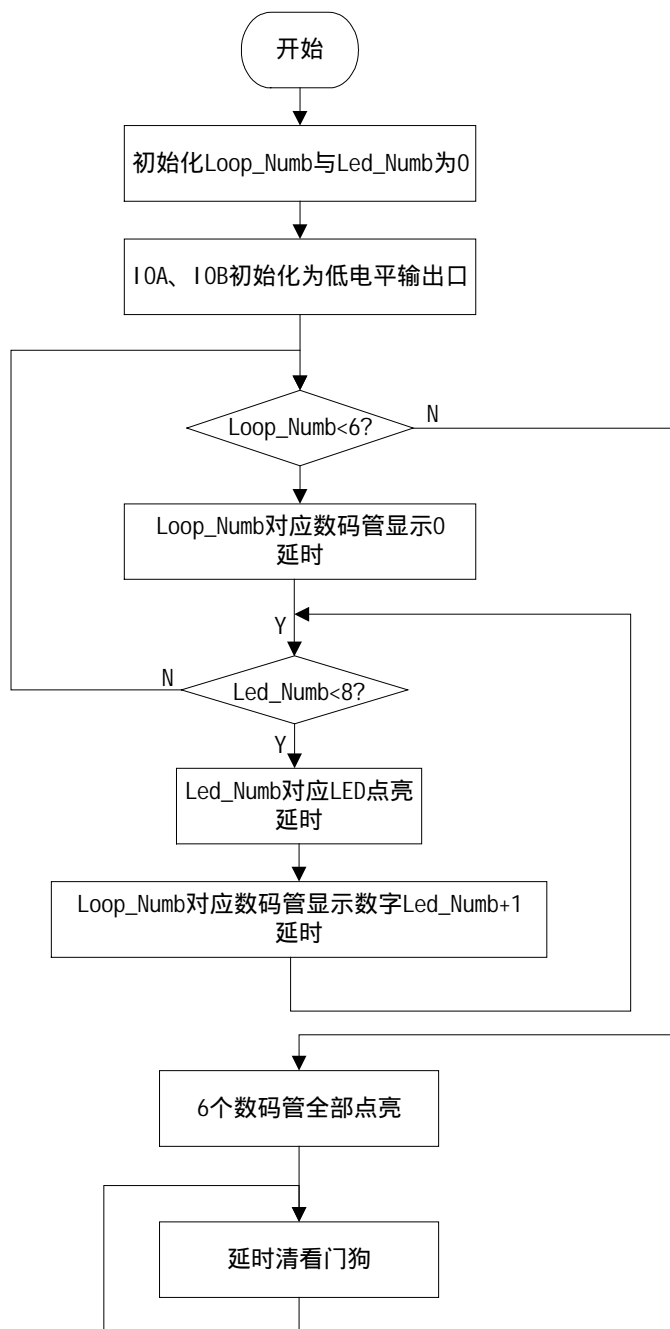
2. 运行参考程序，观察数码管显示数字。

【硬件连接图】



用 10 针排线将 61 板的 J8 接口 IOA 低 8 位连接到 LED 键盘模组的 SEG 接口管脚上，将 J6 接口 IOB 低 8 位连接到 LED 键盘模组的 DIG 接口管脚上。

【程序流程图】



【程序范例】

汇编程序版:

```

.define      P_IOA_Data      0x7000
.define      P_IOA_Dir      0x7002
.define      P_IOA_Attrib    0x7003
.define      P_IOB_Data      0x7005
.define      P_IOB_Dir      0x7007
  
```



```
.define      P_IOB_Attrib      0x7008

.define      P_Watchdog_Clear  0x7012

.data

// 显示段码存放区

DispTbl: .DW 0x003F,0x0006,0x005B,0x004F,0x0066      // 0 , 1 , 2 , 3 , 4
          .DW 0x006D,0x007D,0x0007,0x007F,0x00FF      // 5 , 6 , 7 , 8 , 全亮

.ram

.var R_Led = 0x0000                                // 存储数码管位驱动信息

.code

.public _main

//=====//

// 函数名称：      _main
// 日    期：      20040819
// 功能描述：      发光二极管循环点亮并数码管计数
// LED 巡回闪烁，同时数码管从左至右显示 0~8，全部显示完成后，LED 停止闪烁，数码管全亮
// 语法格式：      _main

//=====//

_main:

    r1 = 0x00FF                                     // 设置 A0~A7 口为同相低电平输出

    [P_IOA_Dir] = r1

    [P_IOA_Attrib] = r1

    [P_IOB_Dir]=r1                                  // 设置 B0~B7 口为同相低电平输出

    [P_IOB_Attrib] = r1

    r1 = 0x0000

    [P_IOA_Data] = r1

    [P_IOB_Data] = r1

    r3 = 0x0001                                     // 从第一个数码管开始

    [R_Led] = r3                                    // 数码管位驱动

L_Loop:

    r2 = 0x0000                                     // 防止误显示

    [P_IOA_Data] = r2;
```

```

r1 = [R_Led]                                // 得到数码管位驱动

[P_IOB_Data] = r1;

BP = DispTbl

r2 = [BP]

[P_IOA_Data] = r2;                          // 数码管显示 0

call F_Delay

call F_LedDisp                              // 发光二极管循回点亮并数码管计数

r1 = [R_Led]                                // 下一次的数码管位驱动

r1 = r1 LSL 0x0001

[R_Led] = r1

r3 += 0x0001

cmp r3,0x0007

jne L_Loop                                  // 数码管未全部显示完成，跳转到 L_Loop 循环

r2 = 0x0000                                  // 防止误显示

[P_IOA_Data] = r2;

r2 = 0x003f;                                // 数码管全亮

[P_IOB_Data] = r2;

BP = DispTbl

BP += 9

r2 = [BP]

[P_IOA_Data] = r2;

L_MainLoop:

    call F_Delay

    jmp L_MainLoop

//=====//

.public F_Delay;

.code

//=====//

// 函数名称：    void F_Delay()

// 功能描述：    延时并清看门狗

//=====//

```



F_Delay:

push r2,r3 to [SP]

r3 = 0x8000

L_Delay:

r2 = 0x0001

[P_Watchdog_Clear] = r2

r3 -= 0x0001

jnz L_Delay

pop r2,r3 from [SP]

retf

//=====//

.public F_LedDisp;

.code

//=====//

// 函数名称 : void F_LedDisp()

// 功能描述 : 发光二极管点亮并数码管计数

//=====//

F_LedDisp:

push r3,r3 to [sp]

r1 = 0x0001

r3 = r1 // 发光二极管状态

L_Led: // 发光二极管一次循环点亮

r2 = 0x0000 // 防止显示错误

[P_IOA_Data] = r2

r2 = 0x0040 // LED 点亮位驱动

[P_IOB_Data] = r2

[P_IOA_Data] = r3 // LED 新状态

r3 = r3 LSL 1 // 下一次 LED 状态

call F_Delay

r2 = 0x0000 // 防止显示错误

[P_IOA_Data] = r2


```

r2= [R_Led]                // LED 点亮位驱动

[P_IOB_Data] = r2

BP = DispTbl                // 对应数字显示在数码管上

B+= r1

r2 = [BP]

[P_IOA_Data] = r2;

call F_Delay

r1 += 0x0001

cmp r1,0x0009

jne L_Led                  // LED 未全部显示完成，跳转到 L_Led 循环

pop r3, r3 from [sp]

retf

```

C 语言版：

```

#define P_IOA_Data          (volatile unsigned int *)0x7000

#define P_IOA_Buffer        (volatile unsigned int *)0x7001

#define P_IOA_Dir           (volatile unsigned int *)0x7002

#define P_IOA_Attrib        (volatile unsigned int *)0x7003

#define P_IOB_Data          (volatile unsigned int *)0x7005

#define P_IOB_Buffer        (volatile unsigned int *)0x7006

#define P_IOB_Dir           (volatile unsigned int *)0x7007

#define P_IOB_Attrib        (volatile unsigned int *)0x7008


#define P_Watchdog_Clear    (volatile unsigned int *)0x7012

int Led_Dispatch(int Led, int Loop);

// 显示段码存放区

int DispTbl[10] = { 0x003F,0x0006,0x005B,0x004F,0x0066,    // 0 , 1 , 2 , 3 , 4
0x006D,0x007D,0x0007,0x007F,0x00FF};                    // 5 , 6 , 7 , 8 , 全亮

unsigned LedControl = 0x0001;

//=====//

// 函数名称：      int main()

```



```
// 日期：    20040819

// 功能描述：    发光二极管循环点亮并数码管计数

// 语法格式：    int main()

//=====================================================//

int main()

{

    int Loop_Numb = 0x0000;           // 循环次数

    int Led_Numb = 0x0000;           // 显示数字个数


    *P_IOA_Dir = 0x00ff;             // 设置 A 口低 8 位为同向低输出，控制 LED 和数码管的
                                     //显示状态

    *P_IOA_Attrib = 0x00ff;

    *P_IOA_Data = 0x0000;

    *P_IOB_Dir=0x00ff;              // 设置 B0~B7 口为同相低电平输出，LED 和数码管的片
                                     //选

    *P_IOB_Attrib=0x00ff;

    *P_IOB_Data=0x0000;


    for (Loop_Numb = 0; Loop_Numb<6; Loop_Numb++)

    {

        *P_IOB_Data = LedControl << Loop_Numb;    // 对应数码管显示 0

        *P_IOA_Data = DispTbl[0];

        Delay();

        for (Led_Numb = 0; Led_Numb<8; Led_Numb++)

        {

            Led_Displ(Led_Numb, Loop_Numb);        // 点亮发光二极管并显示数字 1~8

        }

    }


    *P_IOB_Data = 0x003f;           // 数码管全亮

    *P_IOA_Data = DispTbl[9];
```

```

while (1)
{
    Delay();
}

//=====//

// 函数名称：    void Delay()
// 功能描述：    延时并清看门狗
//=====//

int Delay()
{
    int DelayValue = 0;
    for (DelayValue = 0; DelayValue < 0x8000; DelayValue++)
        *P_Watchdog_Clear = 1;
}

//=====//

// 函数名称：    void Led_Dis()
// 功能描述：    发光二极管循环点亮并数码管计数
// 入口参数：    Led 点亮的发光二极管号码，Loop 显示对应数字的数码管号码
//=====//

int Led_Dis(int Led, int Loop)
{
    *P_IOB_Data = 0x0040;                // LED 点亮
    *P_IOA_Data = LedControl << Led;
    Delay();

    *P_IOB_Data = LedControl << Loop;    // 对应数字显示在数码管上
    *P_IOA_Data = DispTbl[Led + 1];
    Delay();
}

```

【程序练习】

在 $\mu'nSP^TM$ IDE 下用汇编语言和 C 语言编写发光二极管循环点亮并累计计数到 200 ,同时数码管显示数字，累计结束后，发光二极管熄灭数码管全亮。

```
//=====//  
EX9  END  
//=====//
```

实验十 A/D 采样数据在发光二极管上点亮

【实验目的】

1. 熟悉 $\mu'nSP^TM$ IDE 环境及在该环境下用汇编和 C 语言编写的应用程序。
2. 熟悉简单的 $\mu'nSP^TM$ 汇编语言指令。
3. 了解 ADC 的输入接口和转换的原理。
4. 掌握 ADC 寄存器的设置方法。

【实验设备】

1. 装有 Windows 系统和 $\mu'nSP^TM$ IDE 仿真环境的 PC 机一台。
2. 61 板一套； LED 键盘模组一套，连接线一根。

【实验说明】

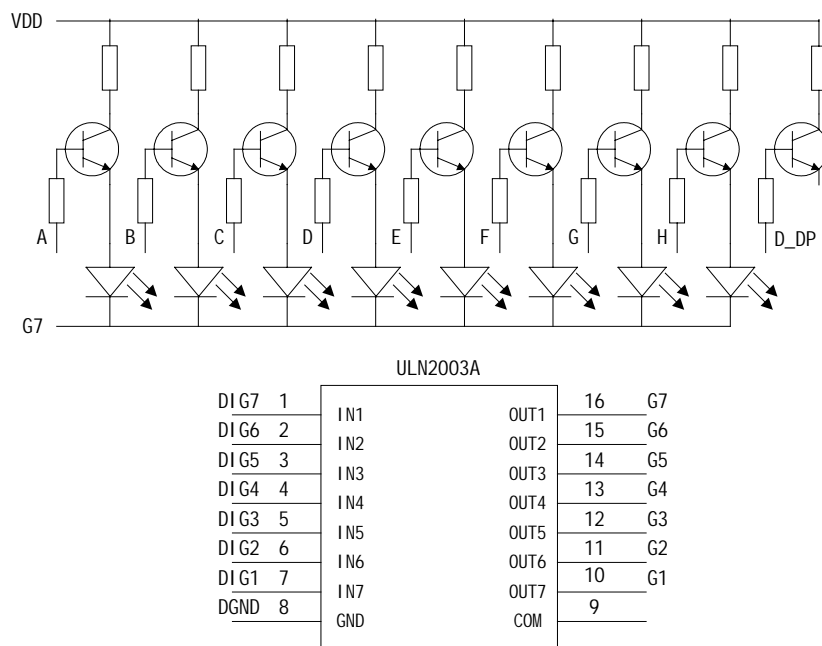
1. 61 板的 I/O 输出实验主要以 IOB8 ~ 15 接 LED 键盘模组上的 8 个发光二极管将 J6 接口 IOB 低 8 位连接到 LED 键盘模组的 DIG 接口管脚上，61 板 SPY0029 接口选择 3.3V，保证 AD 输入在有效范围之内。因 61 板核心芯片 SPCE061A 已内置上下拉电阻，所以端口直接连接发光二极管的驱动端。

2. 实验的结果是调整电位器 R20 后，LED 显示值变化。
3. 代码编写上，主要涉及 SPCE061A 的端口寄存器 IOB 和 ADC 寄存器。

【实验步骤】

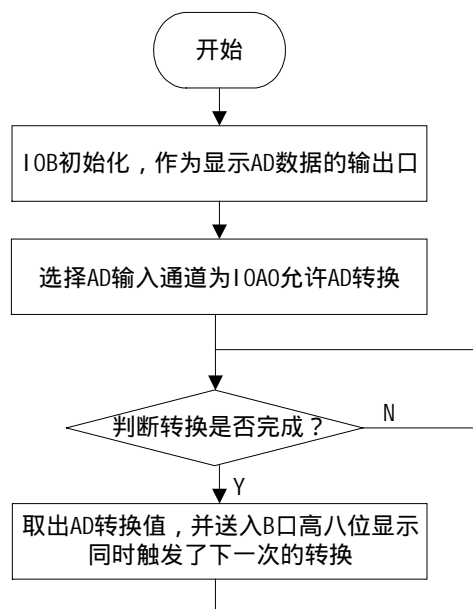
1. 用 10 针排线将 61 板的 J9 接口 IOB 高 8 位连接到 LED 键盘模组的 SEG 接口管脚上，将 J6 接口 IOB 低 8 位连接到 LED 键盘模组的 DIG 接口管脚上，61 板 SPY0029 接口选择 3.3V。
2. 运行参考程序，调整电位器的值。

【硬件连接图】



61 板 SPY0029 接口选择 3.3V，IOB 高 8 位接至 SEG 接口控制 LED 的导通，IOB6 连至 DIG7 通过 ULN2003A 控制 8 个 LED 的共阴极电平状态，也可将 DIG7 直接连至 VDD，直接将 LED 共阴极接地，不用程序控制。

【程序流程图】



【程序范例】

汇编语言版：

```

.define      P_IOB_DATA      0x7005

.define      P_IOB_DIR      0x7007
  
```



```
.define      P_IOB_ATTRI      0x7008

.define      P_INT_Ctrl      0x7010

.define      P_INT_CLEAR      0x7011

.define      P_ADC_Ctrl      0x7015

.define      P_ADC_MUX_Ctrl    0x702b

.define      P_ADC_MUX_DATA    0x702C

.define      P_DAC_Ctrl      0x702A

.define      P_Watchdog_Clear  0x7012

.ram

.var         R_DelayCounter = 0

.code

.public _main

//=====//

// 函数名称：      _main

// 功能描述：      通过改变 LINE_IN 端口的模拟电压来改变 IOB 口输出的数据，

// 采用自动方式即定时器 A 溢出执行 ADC 转换，可以通过 IOB 高 8 位控制发光

// 二极管的点亮了解转换的数据值

// 语法格式：      _main

// 入口参数：      无

// 出口参数：      无

// 注意事项：      仅为用户模型

//=====//

_main:

    r1 = 0xff40

    [P_IOB_ATTRI] = r1                // IOB8-IOB15,IOB6 口设置为同向输出口

    [P_IOB_DIR] = r1

    r1 = 0x0040

    [P_IOB_DATA] = r1

    r1 = 0x0001                      // 选择通道 LINE_IN 为 IOA0

    [P_ADC_MUX_Ctrl] = r1

    r1 = 0x0001                      // 允许 A/D 转换
```

```

[P_ADC_Ctrl] = r1

Loop_AD:

    r1 = [P_ADC_MUX_Ctrl]                // 读寄存器[P_ADC_MUX_Ctrl]的 B15

    test r1,0x8000                       // 判断是否转换完毕

    jz Loop_AD                           // 否，继续转换

    r1 = [P_ADC_MUX_DATA]                // 是，则读出[P_ADC_MUX_DATA]转换结果

                                        // 同时触发 A/D 重新转换

    r1 = r1 lsl 2                         // 保留 A/D 值的最低 8 位

    r1 |= 0x0040                         // 保证 IOB6 为高,LED 阴极共地

    [P_IOB_DATA] = r1

    r1 = 0x0000                          // 延时初值设定

    [R_DelayCounter] = r1

L_DelayLoop:                            // 延时并清看门狗

    r1 = 0x0001                          // 清看门狗

    [P_Watchdog_Clear] = r1

    r1 = [R_DelayCounter]

    r1 += 1                              // 延时间数加 1

    [R_DelayCounter] = r1

    cmp r1,0x100                         // 延时时间到了吗？

    jne L_DelayLoop

    jmp Loop_AD;

```

C 语言版：

```

#define P_IOA_Attrib      (volatile unsigned int *)0x7003

#define P_IOB_Data        (volatile unsigned int *)0x7005

#define P_IOB_Buffer      (volatile unsigned int *)0x7006

#define P_IOB_Dir          (volatile unsigned int *)0x7007

#define P_IOB_Attrib      (volatile unsigned int *)0x7008

#define P_ADC_Ctrl        (volatile unsigned int *)0x7015

#define P_ADC_MUX_Ctrl    (volatile unsigned int *)0x702b

#define P_ADC_MUX_DATA    (volatile unsigned int *)0x702C

#define P_DAC_Ctrl        (volatile unsigned int *)0x702A

```



```
#define P_Watchdog_Clear          (volatile unsigned int *)0x7012

void Delay();

//=====//

// 函数名称：      int main()
// 日    期：      20040817
// 功能描述      AD 采样数据在发光二极管上显示
// 硬件连接：      B 口高八位控制 8 个发光二极管的亮灭，B6 控制 8 个发光二极管的共阴极
// A 口低七位作为 AD 的输入，61 板 SPY0029 接口选择 3.3V，保证 AD 输入在有效范围之内。
// 语法格式      int main()
// 注意事项：      仅为用户模型

//=====//

int main()
{
    unsigned ADValue = 0x0000;

    *P_IOB_Dir=0xff40;                // 设置 B 口高 8 位为同向低输出，设置 B6 口为高电平输
                                     // 出，保证 LED 共阴极接地

    *P_IOB_Attrib=0xff40;

    *P_IOB_Data=0x0040;

    *P_ADC_MUX_Ctrl = 0x0001;          // 模拟电压信号通过 LINE_IN1 输入
    *P_ADC_Ctrl |= 0x0001;             // 允许 AD 转换

    while (1)
    {
        while (!(*P_ADC_MUX_Ctrl & 0x8000)); // 等待 AD 转换完成
        ADValue = *P_ADC_MUX_DATA;         // 读取转换值
        ADValue <= 2;                      // 通过 led 显示 AD 转换结果
        ADValue |= 0x0040;                 // 保证 LED 阴极共地
        *P_IOB_Data = ADValue;
        Delay();                          // 延时显示
    }
}
```



```
}  
  
//=====//  
// 函数名称：    Delay()  
// 功能描述：    实现延时  
//=====//  
  
void Delay()  
{  
    // 延时子程序  
  
    unsigned int i;  
  
    for(i = 0; i < 0x50; i++){  
  
        *P_Watchdog_Clear=0x0001;    // 清 WatchDog  
  
    }  
}
```

【程序练习】

在 $\mu'nSP^TM$ IDE 下用汇编语言和 C 语言编写 A/D 采样数据在发光二极管上点亮，使用 IOB 低 8 位显示 LED，KED 共阴极控制端 DIG7 可直接接高，不用程序控制。

```
//=====//  
  
EX10  END  
  
//=====//
```

第 5 章 61 板深入学习向导

5.1 学习向导

61 板可以供初学者学习单片机的基本使用方法，也可以供单片机高手深入的了解 SPCE061A 单片机。学习流程如下：

第一阶段：初步学习

参考资料：

《61 板使用说明书》《61 板实验教程》《凌阳十六位单片机应用基础》

其中《61 板使用说明书》和《61 板实验教程》两份资料可以到大学计划网站下载。

《凌阳十六位单片机应用基础》一书由北航出版社出版，大学计划网站有售。

实验设备：61 板 LED 键盘模组

1. 《61 板使用说明书》

用户可以采用此说明书了解 61 板的结构及应用。61 板出厂前，均已含有自检程序，用户拿到 61 板后，根据说明书上对自检步骤的说明，完成 61 板的自检。

2. 《61 板实验教程》

用户采用此教程学习 61 板的基本功能及单片机常用外围器件的使用方法，如按键、LED 和 LED 数码管，此实验教程是 61 板结合 LED 键盘模组实现，用户可以自己焊接此模组，也可以到大学计划网站订购。通过 61 板与 LED 键盘模组结合实验，用户会清楚了解 SPCE061A 端口及片上 AD 的使用方法。同时用户也可以采用此套系统深入学习 61 板的使用。

3. 《凌阳十六位单片机应用基础》

此书由罗亚非编著，北航出版社出版，用户可以采用此书了解 SPCE061A 的硬体结构、指令系统、IDE 的使用方法等。因此用户在完成基础实验过程中，也需参考此书。

此书含有丰富的范例，其中每个范例已经与 IDE 打包在一起，用户安装完 IDE 后，在 SPCE061A 文件夹下的 TextBookExample 中。其中范例按照章节列出。

通过第一阶段的学习，用户熟悉了 61 板的结构和板上的元器件，了解了 SPCE061A 的端口设置方式以及 IDE 的使用方法。

第二阶段：深入了解

参考资料：《实验指导书》上册 此指导书用户可以在大学计划网站下载。

实验设备：61 板 LED 键盘模组

《实验指导书》上册含有近 30 个实验，含有 SPCE061A 所有功能模块，如 ADC、DAC、MIC、中断、定时器、语音的录制播放等。这些实验，用户均可以采用 61 板结合 LED 键盘模组完成。

用户学习过程中，结合《凌阳十六位单片机应用基础》，一边学习，同时做一些练习。

《实验指导书》相关范例都已经与 IDE 打包在一起，用户安装完 IDE 后，在 SPCE061A 文件夹下的 Example 中可以找到相关范例。范例包括基础实验（baseExa）和语音实验（VoiceExa）两部分。

第二阶段，是艰苦学习的一个阶段，在此阶段中，用户了解 SPCE061A 所有的功能及软件实现方法，并结合实验进行体会和实践。

第三阶段：成为高手

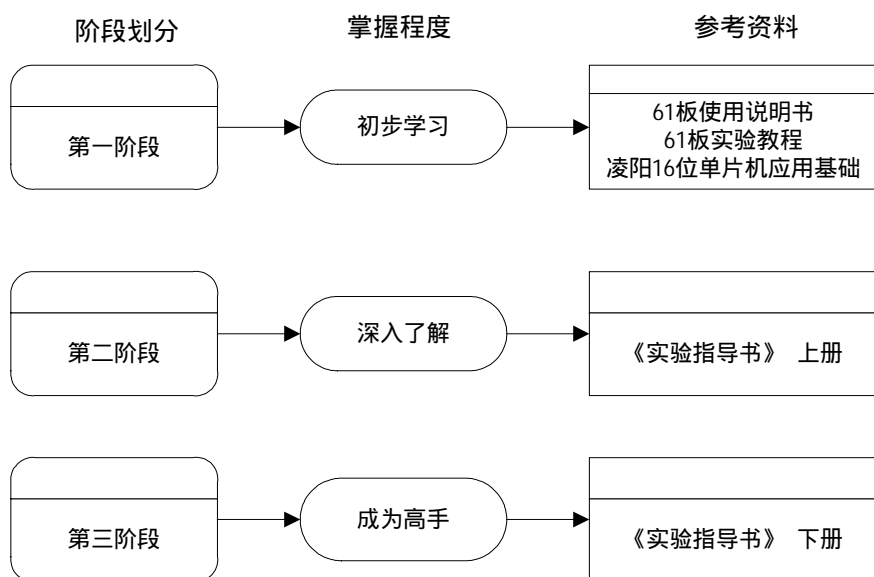
参考资料：《实验指导书》下册 此指导书用户可以在大学计划网站下载。

实验设备：61 板 USB 模组 LCD 模组 LED 键盘模组等

《实验指导书》下册含有 25 个综合实验，其中涉及到常用的液晶、数码管、4×4 键盘、LED 点阵、USB 通讯、FLASH 存储等。一旦用户掌握这些应用，就基本可以使用这些器件来设计自己需要的方案。当然，经历过这三个阶段后，用户不可能马上就可以完成某些大的系统的设计，但随着开发经验的不断积累，用户很快即可进入单片机高手之列。

同时大学计划网站提供 61 板精彩教程，基本达到了手把手的教学方式，用户可以下载到本机进行学习。

最后，希望 61 板带你步入单片机天地。



5.2 61 板其他配套模组说明

下面介绍 6 个模组，它们是：USB 接口模组、USB/UART 转换口模组、以太网模组、SPLC501 液晶模组、图像识别模组和交通灯模组。

5.2.1 USB 接口模组



详细说明：

工作电压：3.3V or 5.0V

外形尺寸：35mm×52mm

系统环境：Windows 98/Me/2000

USB 模组功能：

该模组既支持 USB 总线供电又支持外设供电，同时 USB 总线电源可以通过该模组给外设供电。可以和多种微处理器组合，进行 USB 通信。

USB 模组套件包括：USB 模组，USB 驱动程序，两个例子的 Firmware，两个例子的 PC 端应用程序，USB 模组用户说明书。

USB 模组套件提供的两个例子只是一个 USB 通讯的基础应用，设计者完全可以在此基础上开发出自己的 USB 产品，比如录音笔系统、解说器系统、数据采集系统等等。

5.2.2 USB/UART 转换口模组



详细说明：

SPCP825A 是全新的 USB 转 UART 的单芯片解决方案。该芯片集成了一个符合 USB1.1 标准的全速功能控制器、缓冲器、和带有调制解调器接口信号的异步串行数据总线（适用于 RS-232 协议），同时具有一个集成的内部时钟和 USB 收发器，无需其他外部 USB 电路元件。高性能的 SPCP825A 与其他型号的同类芯片相比功耗更低、体积更小、集成度更高。SPCP825A 是 USB 转串口应用的理想选择。

● USB

支持 USB1.1 协议；

提供全速和低速传输；

使用 6MHZ 工作频率，内建 PLL 电路为两种工作模式提供必要的时钟频率；

提供三个终端，每一个都可软件使能为输入或者输出终端。

- UART

提供全双工的异步串行通讯，波特率可高达 230400bps；

10b 或者 11b 模式带奇偶校验。

5.2.3 以太网通讯模组



详细说明：

1. 可通过此设备和 61 板互连到局域网或者广域网进行通讯。
2. 支持多种连接模式；电端口支持 10MB HALF / 10MB FULL / 100MB HALF / 100MB FULL / AUTO (N-WAY)。
3. 和 MCU 的连接模式有 ISA 8 bit / ISA 16 bit 模式，并且支持 3.3V 和 5V 的 I/O 控制。可方便与不同电压的 MCU 连接。
4. 拥有 4Mb 串行数据存储器 (SPR4096) 及其接口。

应用范围：

1. 通过 61 板与此设备搭配可完成一个简单的 WEB 服务器。
2. 通过 61 板与此设备搭配可完一些远程监控。

5.2.4 SPLC501 液晶模组



详细说明：

该模组是凌阳科技公司的一款 128*64 点阵的液晶模组，驱动芯片采用的是凌阳 SPLC501。模组接口简单，应用方便，功耗低，且可以完成较多液晶特效功能。

该液晶模组可以显示字符、汉字、图形等，且灰度编程可调。

显示模式：黄色模式 STN 液晶

显示格式：128*64 点阵图形液晶显示

输入数据：兼容 68/80 系列 MPU 数据输入

背 光：黄绿色 LED

模块尺寸：72.8 (长) × 73.6 (宽) × 9.5 (高) mm

视屏尺寸：58.84 (宽) × 35.79 (长) mm

像素尺寸：0.46 (宽) × 0.56 (长) mm

点大小：0.42（宽）×0.51（长）mm

大学计划提供此模组的字符、汉字、图片、几何图形、动态图片等范例。用户可以根据这些范例很快了解此液晶模组的应用。液晶模组范例已经包含在 IDE184 路径下的 Example 中。

5.2.5 图像识别模组



详细说明：

图像识别模组【Eagle 模组】由光学镜头、CMOS 传感器(SPCA561A)、图象处理芯片(SPCA563A)组成。

该模组可以实现如下功能：

1. 识别颜色、形状
2. 识别位置

5.2.6 交通灯模组



详细说明：

模拟交通灯控制就是使用单片机、PLC 或其它器件来控制一些 LED 和数码管，模拟真实交通灯的功能。红、黄、绿交替闪亮，倒计时显示时间，方向灯指示方向等。

很多高校都有模拟交通灯实验的实验，但没有专门的模拟交通灯控制板。这就给实验室增加了不少麻烦。为了解决这个问题，我们推出的这个模拟交通灯控制板。它可以让学生轻松完成模拟交通灯的基本实验后，有条件模拟更为复杂的情况，完成与实际路口相差无几的交通控制。

基本参数：

产品型号：TRAFFIC BOARD V1.0

工作电压：5.0V

外形尺寸：140mm×160mm

附录一：LED 键盘模组的原理图



附录二：LED 键盘模组的实物图

