实验十三　SOPC 嵌入式系统外设接口控制

一、实验目的

熟悉 NiosII PIO 设备的访问方法

二、实验设备

硬件：　PC 机

GX-SOC/SOPC-DEV-LAB PLATFORM 创新开发实验平台

GX-SOPC-EP2C35-M672 核心板

软件：　Quartus II 9.0

Nios II 9.0

三、实验内容

1. 使用 PIO 口控制 8 个 LED 灯和按键
2. 字符液晶显示实验
3. 16*16 点阵模块显示实验

(一) 使用 PIO 口控制 LED 灯和按键

1. 实验内容：

熟悉 PIO 外设的访问和控制。

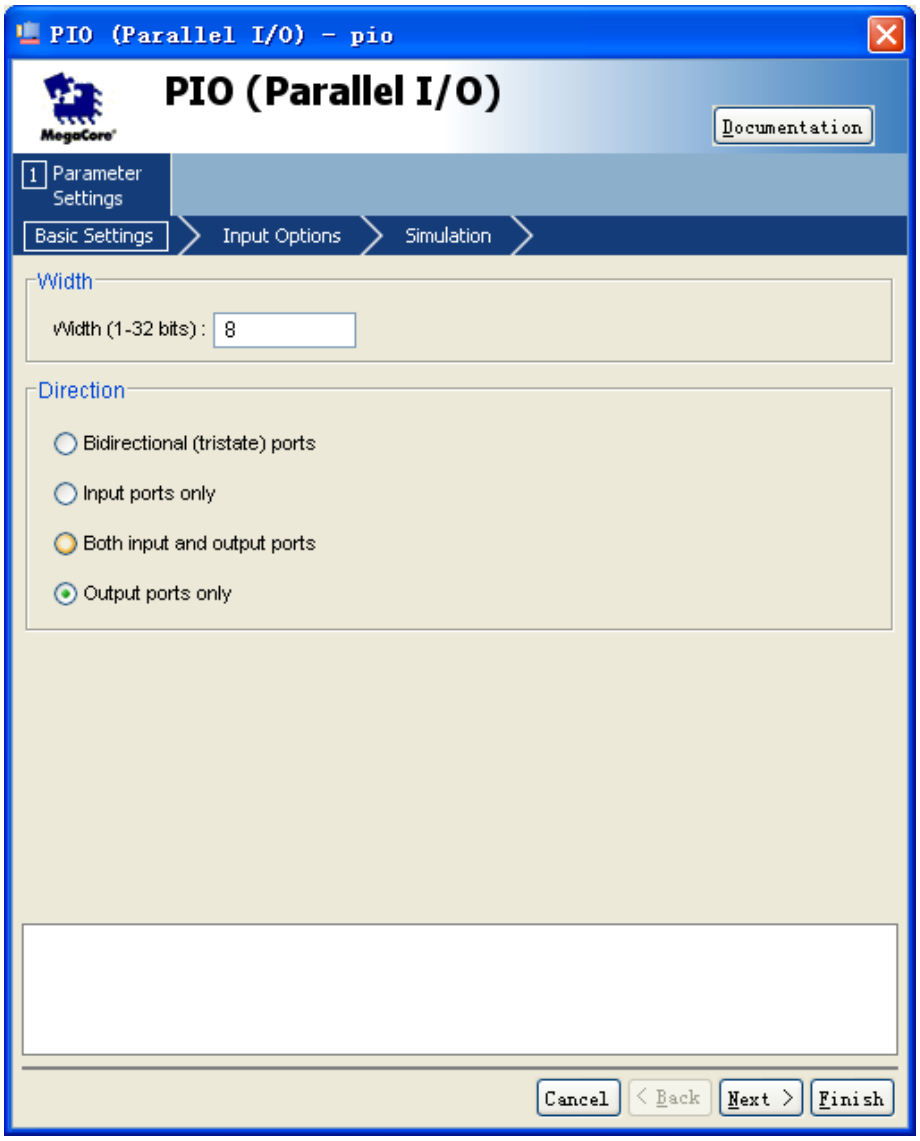
软件编写程序控制八个 LED 循环被点亮。用 dir 控制循环方向，当点亮的 LED 达到边缘时，改变方向。

2．硬件系统设计：

硬件配置图和系统原理图

添加 8 位 PIO:

添加 IO 口控制器。双击 Peripherals -> Microcontroller Peripherals -> PIO (Parallel I/O)，保持默认设置即可，表示有 8 个输出用 IO 口，分别控制开发板上的 8 个绿色 LED 灯，将名字修改为 led_pio.



实验十二已完成！

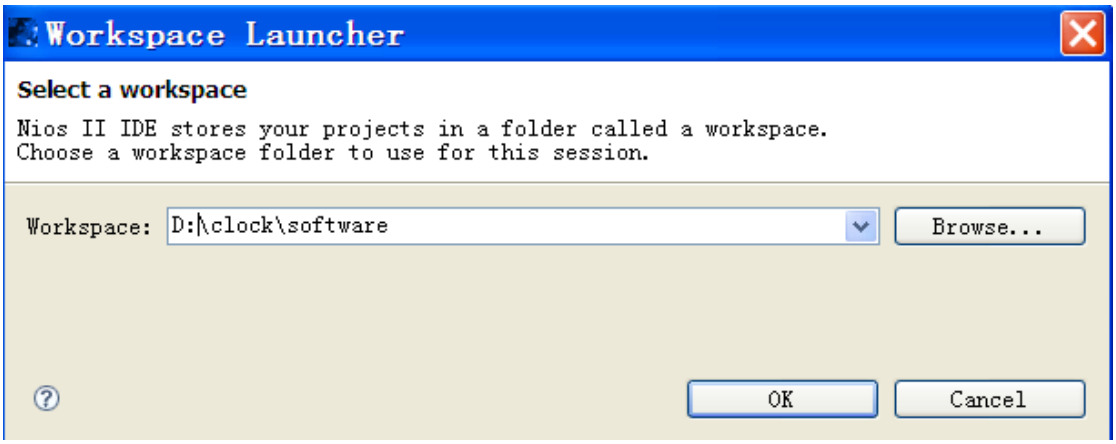
3. 软件系统设计：

建立工程的步骤如实验十二。

双击 NIOS II IDE 图标  ，第一次打开的时候会提示选择工作空间。

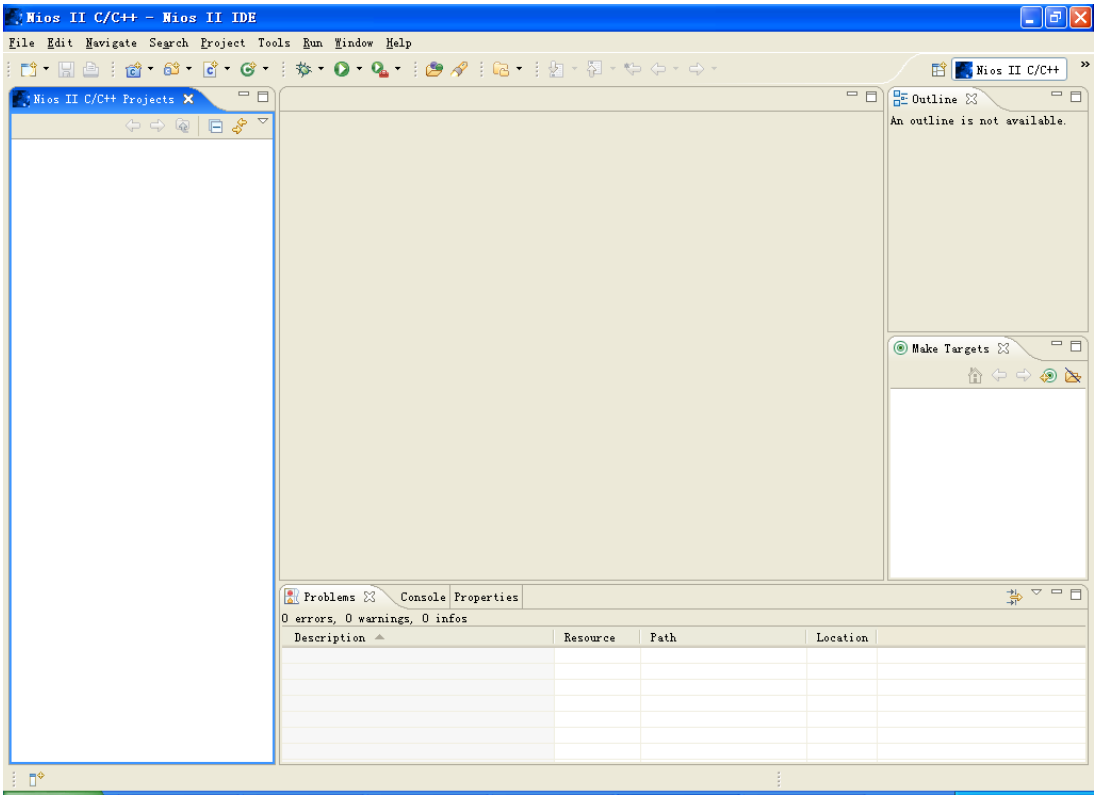
也可在程序打开后选择菜单栏 File -> Switch Workspace…

选择 *<工程所在目录>\software* 作为 NIOS II 的工作空间。确认以后软件会重新启动。

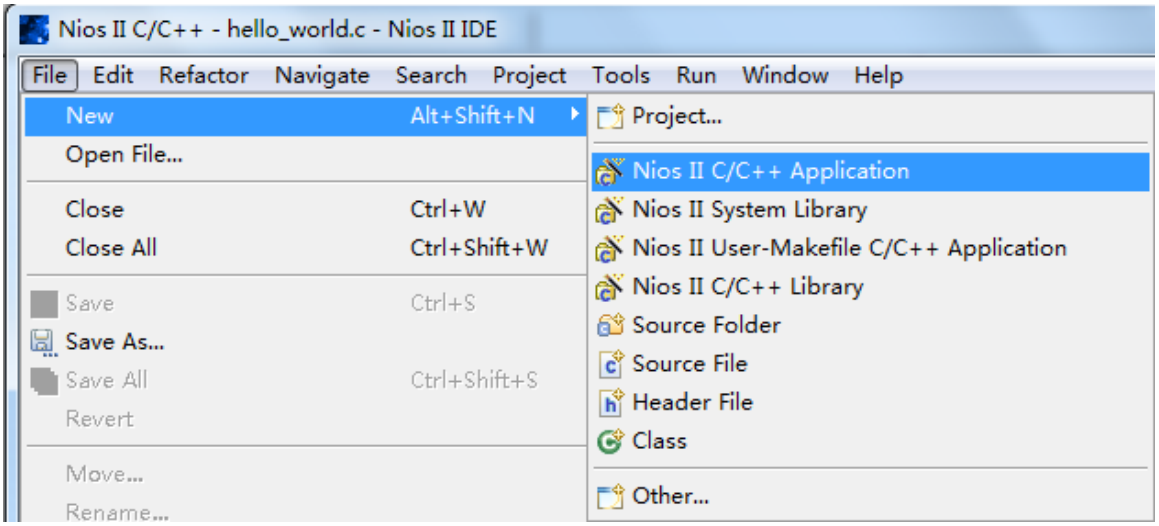


选择 *<工程所在目录>\software* 作为 NIOS II 的工作空间。确认以后软件会重新启动。在欢迎界面中选择 Workbench，

进入主界面



在 NIOS II IDE 软件环境中点击 New->Nios II C/C++ Application,建立一个工程



在标准模板中选择 Hello World，

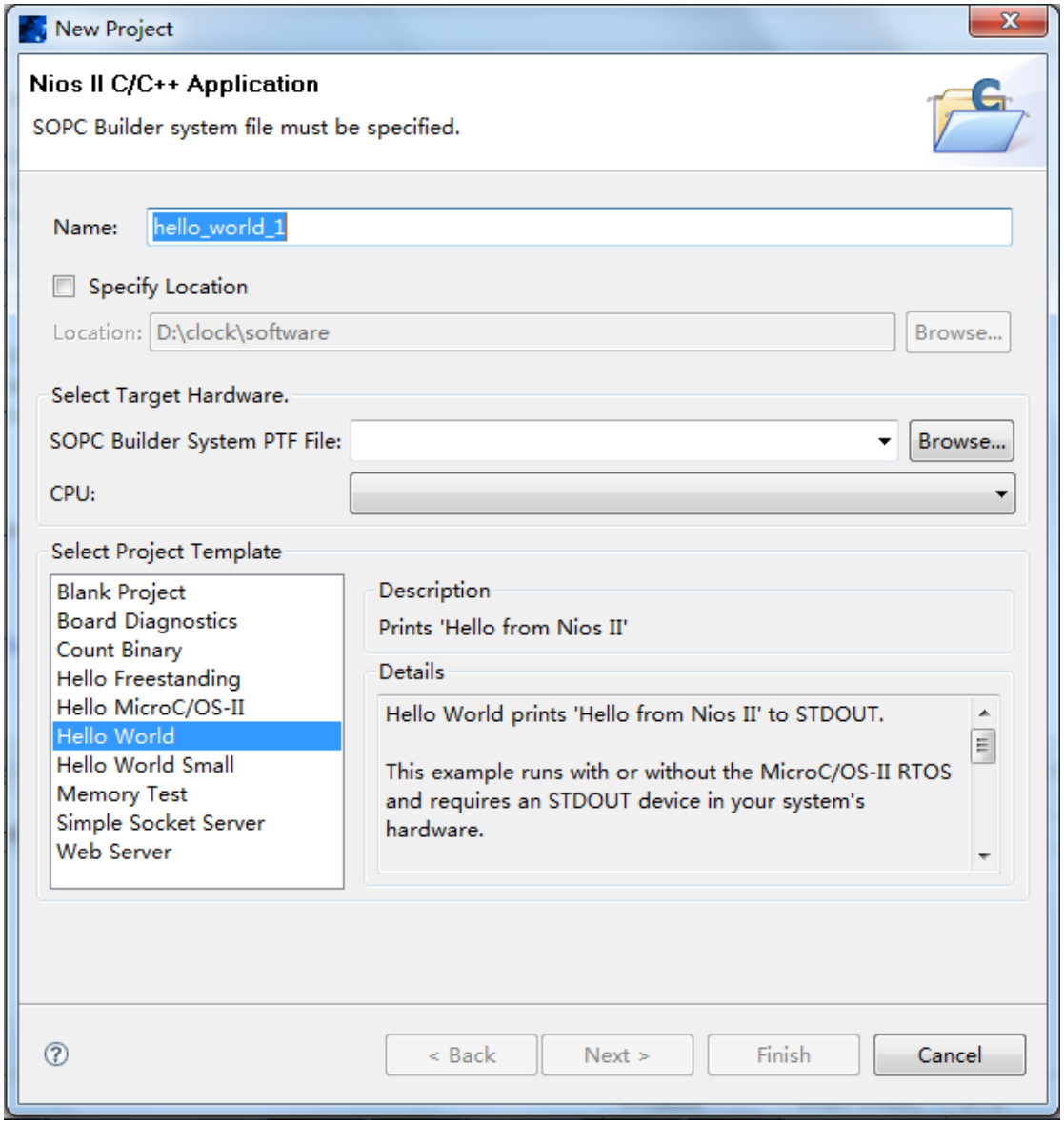


图 46

在 SOPC Builder System PTF File 点击 Browse，选择 nioscpu.ptf

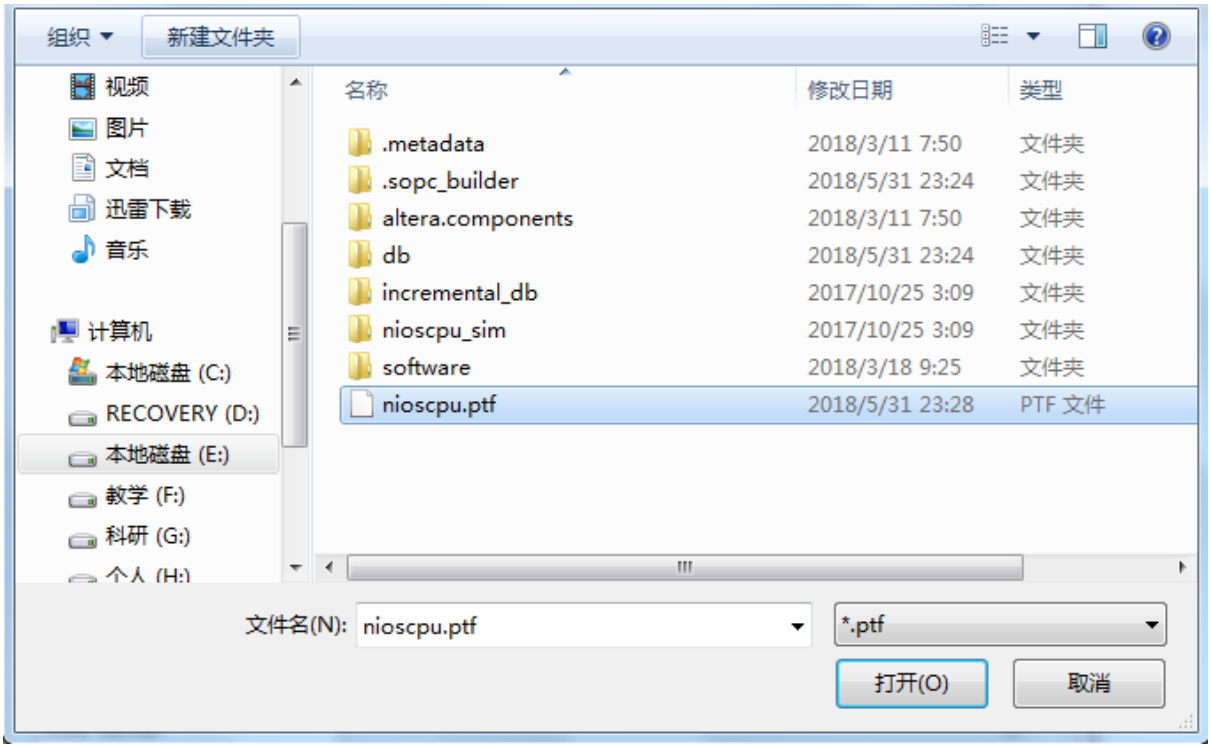
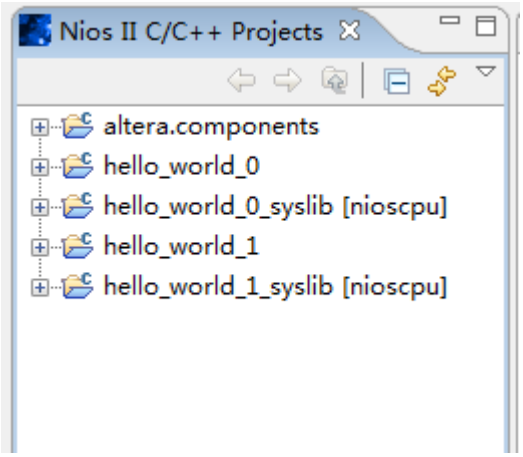


图 47

C/C++的应用工程名 Name 为 hello_word_1，点击 Next

点 Next >，在下一个界面中选择第一项，Create a new system library，点击 Finish 完成。

此时，NIOS II IDE 左侧工程列表将出现新建工程 hello_word_1



软件设计流程与方法：

注意：需要包含 3 个头文件：

1）system.h 系统描述文件

system.h 文件是 HAL 系统库的基础，system.h 文件提供了 Nios II 系统硬件的软件描述，包括：

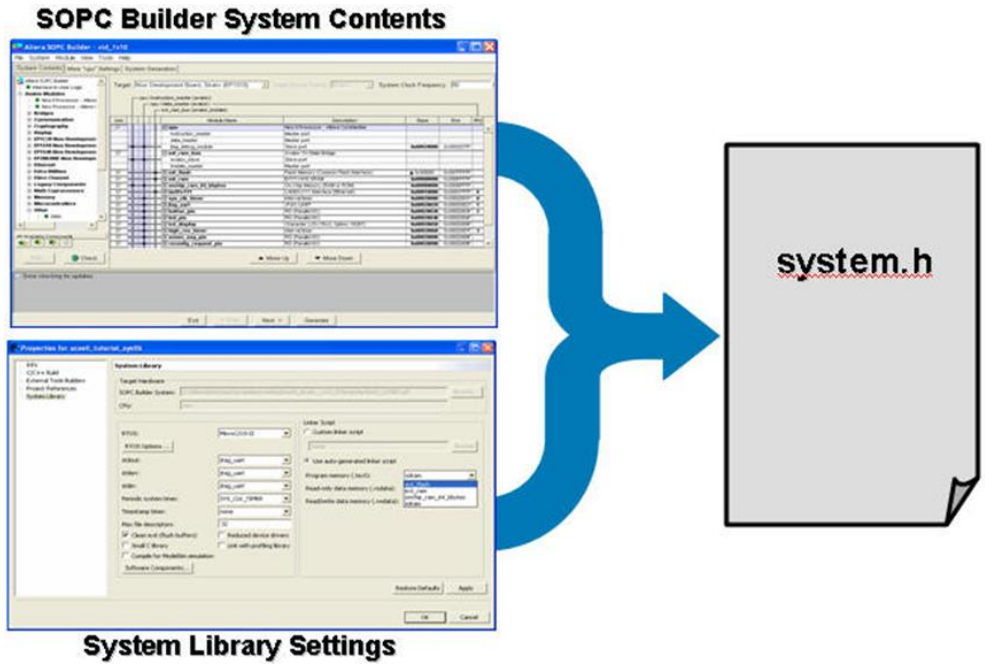
（1）外围设备的硬件配置

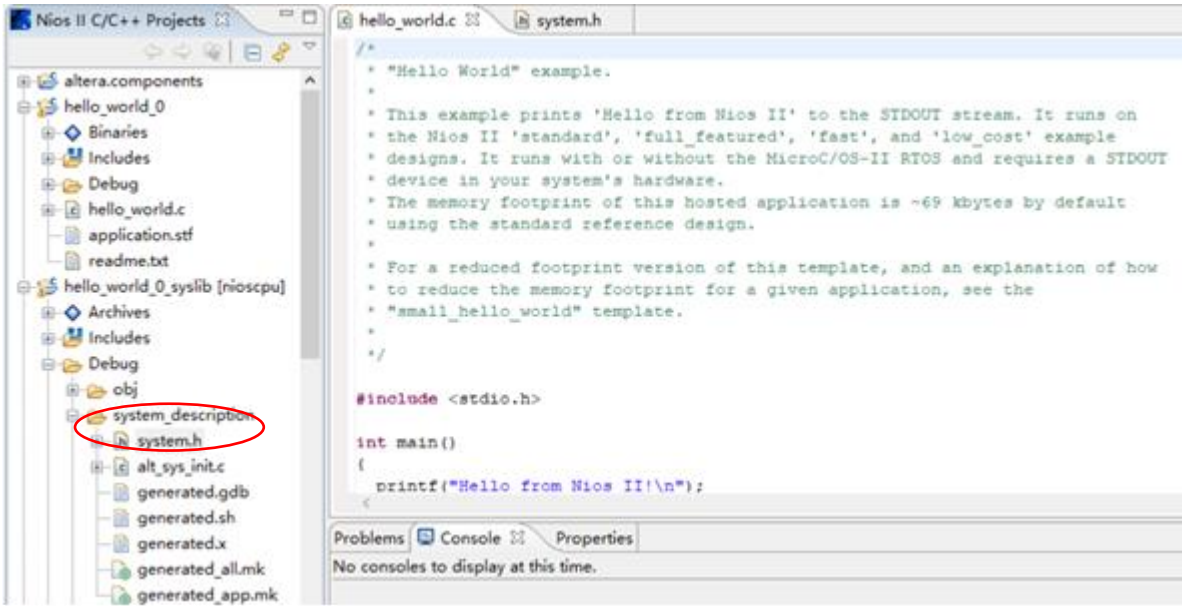
（2）基地址

（3）中断优先级

（4）外围器件的符号名称

编写任何与 Nios II 系统硬件外设有关的程序都需要包含 system.h 文件，它也是 Nios II IDE 根据系统 PTF 文件生成的。

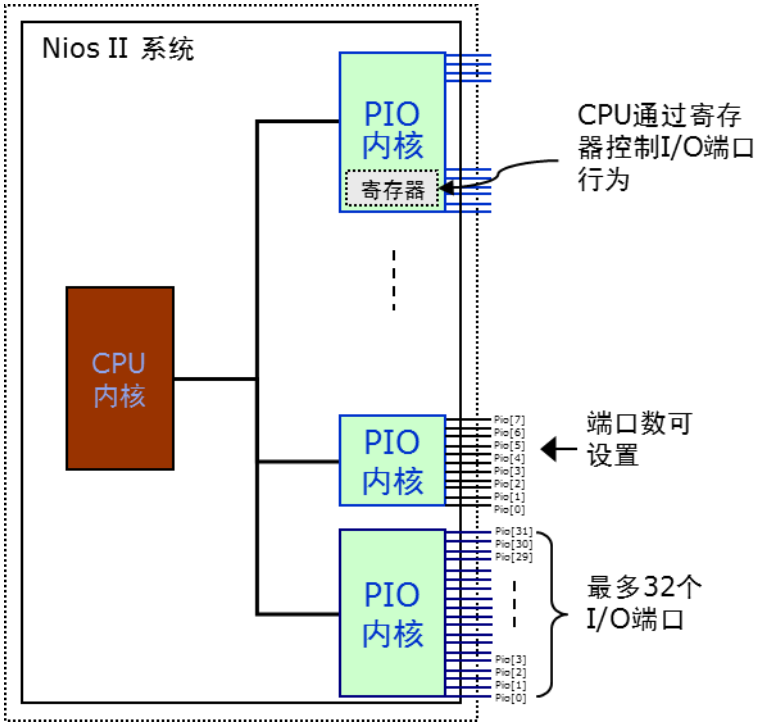




2）"altera_avalon_pio_regs.h"

```
#ifndef __ALTERA_AVALON_PIO_REGS_H__
#define __ALTERA_AVALON_PIO_REGS_H__

#include <io.h>

#define IOADDR_ALTERA_AVALON_PIO_DATA(base)             __IO_CALC_ADDRESS_NATIVE(base, 0)
#define IORD_ALTERA_AVALON_PIO_DATA(base)               IORD(base, 0)
#define IOWR_ALTERA_AVALON_PIO_DATA(base, data)         IOWR(base, 0, data)

#define IOADDR_ALTERA_AVALON_PIO_DIRECTION(base)        __IO_CALC_ADDRESS_NATIVE(base, 1)
#define IORD_ALTERA_AVALON_PIO_DIRECTION(base)          IORD(base, 1)
#define IOWR_ALTERA_AVALON_PIO_DIRECTION(base, data)    IOWR(base, 1, data)

#define IOADDR_ALTERA_AVALON_PIO_IRQ_MASK(base)         __IO_CALC_ADDRESS_NATIVE(base, 2)
#define IORD_ALTERA_AVALON_PIO_IRQ_MASK(base)           IORD(base, 2)
#define IOWR_ALTERA_AVALON_PIO_IRQ_MASK(base, data)     IOWR(base, 2, data)

#define IOADDR_ALTERA_AVALON_PIO_EDGE_CAP(base)         __IO_CALC_ADDRESS_NATIVE(base, 3)
#define IORD_ALTERA_AVALON_PIO_EDGE_CAP(base)           IORD(base, 3)
#define IOWR_ALTERA_AVALON_PIO_EDGE_CAP(base, data)     IOWR(base, 3, data)
```

```
#define IOADDR_ALTERA_AVALON_PIO_SET_BIT(base)          __IO_CALC_ADDRESS_NATIVE(base, 4)
#define IORD_ALTERA_AVALON_PIO_SET_BITS(base)           IORD(base, 4)
#define IOWR_ALTERA_AVALON_PIO_SET_BITS(base, data)     IOWR(base, 4, data)

#define IOADDR_ALTERA_AVALON_PIO_CLEAR_BITS(base)       __IO_CALC_ADDRESS_NATIVE(base, 5)
#define IORD_ALTERA_AVALON_PIO_CLEAR_BITS(base)         IORD(base, 5)
#define IOWR_ALTERA_AVALON_PIO_CLEAR_BITS(base, data)   IOWR(base, 5, data)



/* Defintions for direction-register operation with bi-directional PIOs */
#define ALTERA_AVALON_PIO_DIRECTION_INPUT  0
#define ALTERA_AVALON_PIO_DIRECTION_OUTPUT 1

#endif /* __ALTERA_AVALON_PIO_REGS_H__ */
```



偏移量	寄存器名称		R/W	(n-1)	…	2	1	0
0	数据寄存器	读访问	R	读入输入引脚上的逻辑电平值				
		写访问	W	向PIO输出口写入新值				
1	方向寄存器		R/W	控制每个I/O口的输入输出方向。 0：输入；1：输出。				
2	中断屏蔽寄存器		R/W	使能或禁止每个输入端口的IRQ。 1：中断使能；0：禁止中断。				
3	边沿捕获寄存器		R/W	当边沿事件发生时对应位置1。				

例如：IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE, 0xff)

3） 数据宽度和 HAL 类型定义 alt_types.h

ANSI C 数据类型没有明确地定义宽度，HAL 使用了一套标准的类型定义，支持 ANSI C 类型，但数据宽度取决于编译器的约定。

alt_types.h 头文件定义了 HAL 的数据类型。在以下路径可以查该文件：[Nios II 安装路径]\components\altera_nios2\HAL\inc

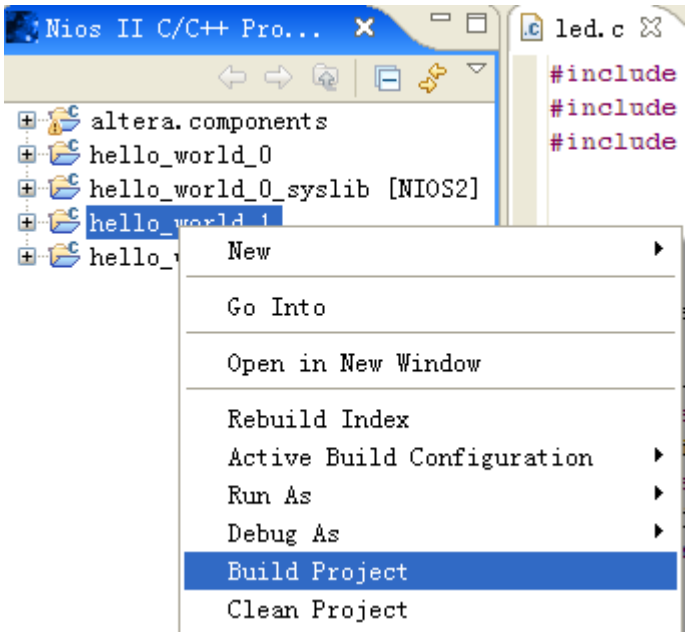
类型	说明
alt_8	有符号8位整数
alt_u8	无符号8位整数
alt_16	有符号16位整数
alt_u16	无符号16位整数
alt_32	有符号32位整数
alt_u32	无符号32位整数

4） 将下面的程序复制到 hello_led.c 中，存盘，编译工程。

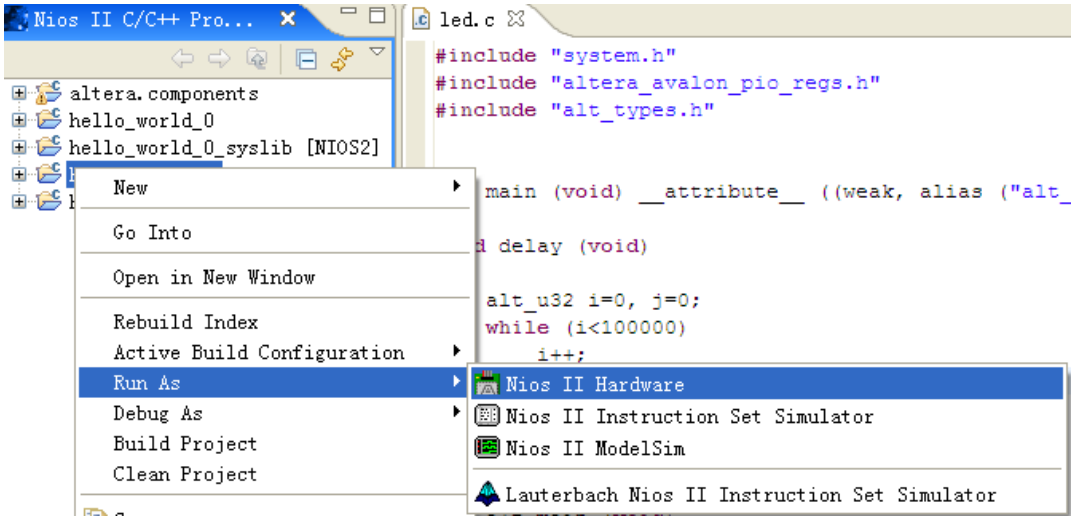
```
#include "system.h"
#include "altera_avalon_pio_regs.h"
#include "alt_types.h"


int main (void) __attribute__ ((weak, alias ("alt_main")));

void delay (void)
{
      alt_u32 i=0, j=0;
      while (i<200000)
            i++;
      while (j<200000)
            j++;
      return;
}

int alt_main (void)
{
   alt_u16 led = 0x2;
   alt_u8 dir = 0;
   /*
    * Infinitely shift a variable with one bit set back and forth, and write
    * it to the LED PIO.    Software loop provides delay element.
    */
   while (1)
```

```
    {
        if (led & 0x81)
        {
            dir = (dir ^ 0x1);
        }

        if (dir)
        {
            led = led >> 1;
        }
        else
        {
            led = led << 1;
        }
       /*利用函数将led的值写到led_pio中，请同学们填写*/

        delay();
}

        return 0;
}
```

存盘，然后编译工程。



编译通过后，下载软件。



可以看到 BICE 创新开发实验平台的 LED0-LED7 发光二极管上 LED 灯循环点亮。

扩展要求：读取按键值，然后将读到的值输出到 LED。

实验内容

通过 8 个按键来控制 LED 的亮灭。

我们按 BICE 创新开发实验平台的 F1-F8 按钮键，可以看到对应控制的 LED1-LED8 发光二极管亮。

实验参考程序

```
#include "system.h"
#include "altera_avalon_pio_regs.h"
#include "alt_types.h"


int main (void) __attribute__ ((weak, alias ("alt_main")));

int alt_main (void)
{
  alt_u8 button;

  while (1)
  {

    //读按键值；
    //将读入的按键值输出到 led
```

```
    }

  return 0;
}
```

在实验报告中对每条源程序加上注释。

（二） 字符液晶显示控制：

1. 实验内容：

1) 利用 IO 完成 2X16 字符液晶屏的访问。

2) 通过写命令来控制将数据写到哪一行；通过写数据，将数据输出在液晶屏上显示。在 2X16 字符液晶屏上，可以看到

1234567890abcdef

ghijklmnopqrstuv 字样。

2. 实验原理：

芯片简介：

开发平台采用HD44780U点阵液晶控制器来驱动2×16液晶模块。HD44780U是目前最常用的字符型液晶显示驱动控制器，能驱动液晶显示文字、数字、符号、日语假名字符。与HD44780S管脚兼容，通过配置，可以连接8位或4位的MCU。单片HD44780U可驱动1行8字符或2行16字符显示。HD44780U的字形发生ROM可产生208个5×8点阵的字体样式和32个5×10的字体样式，共240种不同的字体样式。低电源提供（2.7V 到5.5V)可适应于多种应用。

芯片特点：

- 支持5×8和5×10点阵
- 低电源操作支持：2.7V到5.5V
- 宽范围液晶驱动电源支持：3.0到11V
- 液晶驱动波形————A（一路交流频率波形）

- 支持4位或8位MPU接口
- 80×8位显示RAM（最多80个字符）
- 9,920位字形发生ROM，共提供240个字体样式

  ———208个5×8点阵字体样式

  ———32个5×10点阵字体样式
- 64×8位字形发生RAM

  ———8个5×8点阵字体样式

  ———4个5×10点阵字体样式
- 16-common×40-segment 液晶驱动
- 编程占空因数：

  ——— 行　　点阵带光标：

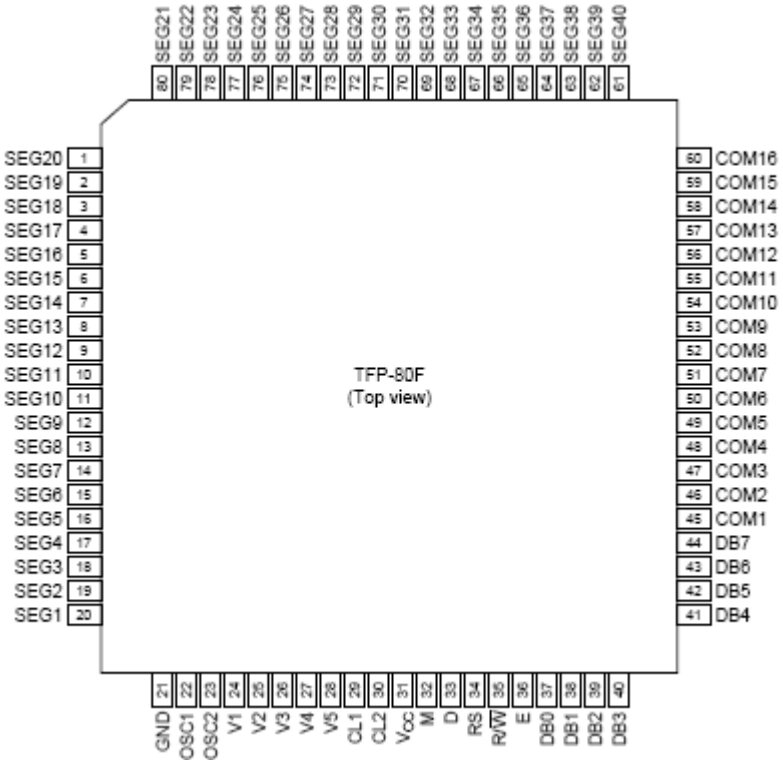
  ——— 行　　点阵带光标：

  ——— 行　　点阵带光标：
- 多种指令功能：

  ———清屏、归home位、显示设置、光标设置、闪烁、光标移动、显示移动
- 管脚与HD44780S兼容
- 上电后，自动复位电路初始化驱动器
- 低功耗

管脚图及管脚说明：

管脚图：屏上采用与之相兼容的芯片。

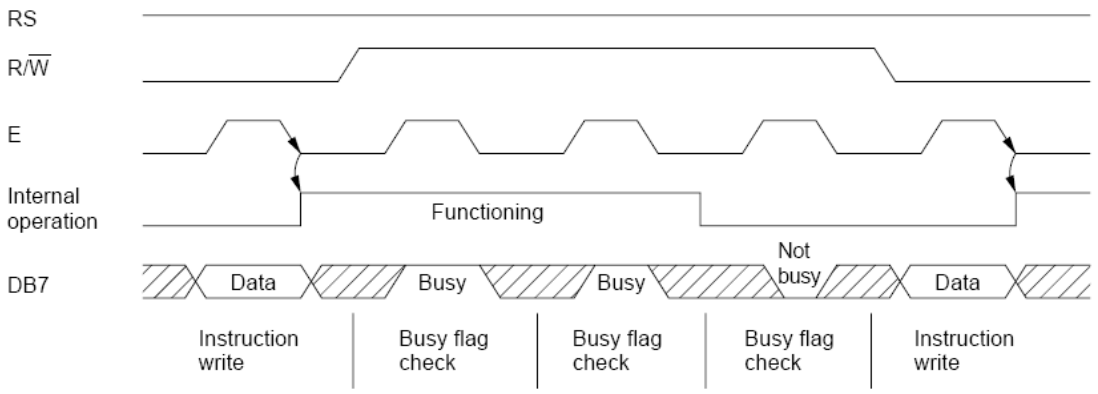


管脚说明：

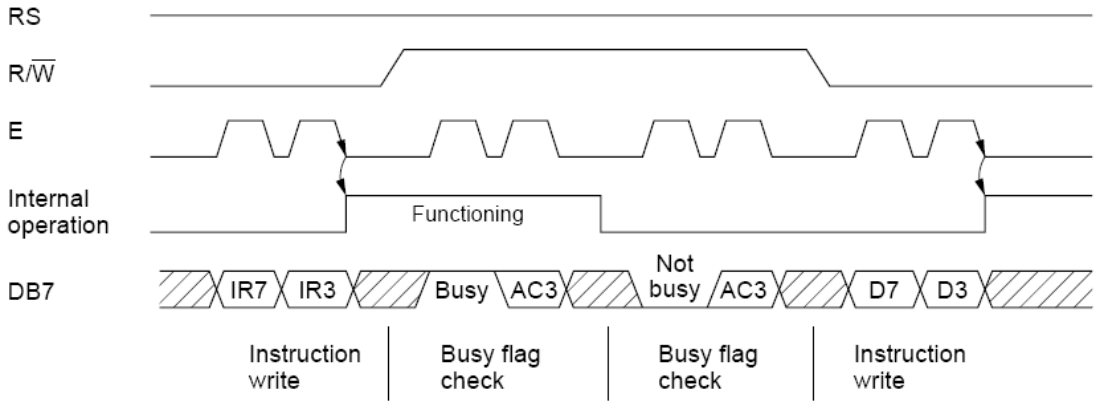
管脚	位数	输入/输出	交互对象	功能描述
RS	1	输入	MPU	寄存器选择信号 0：指令寄存器（写）忙标志位：地址计数器（读） 1：数据寄存器（读写）
R/Wn	1	输入	MPU	读写选择　0：写 1：读
E	1	输入	MPU	读写使能
DB4-DB7	4	输入/输出	MPU	高四位双向数据管脚。DB7可作为忙标志位。
DB0-DB3	4	输入/输出	MPU	低四位双向数据管脚。4位操作时这些管脚不用
CL1	1	输出	扩展驱动	锁存送往扩展驱动串行数据D的时钟
CL2	1	输出	扩展驱动	移位串行数据的时钟
M	1	输出	扩展驱动	转换液晶驱动波形到AC的切换信号
D	1	输出	扩展	符合每个段信号的字符模式数据
COM1-COM16	16	输出	LCD	16位信号，接公共端。
SEG1-SEG40	40	输出	LCD	段信号
V1-V5	5	—	电源	Vcc－V5＝11V（最大）
Vcc，GND	2	—	电源	Vcc：2.7V到5.5V，GND：0V
OSC1，OSC2	2	—	振荡电阻时钟	当晶震工作时，必须外接一个电阻。当输入引脚为外部时钟，必须连接到OSC1。

时序图：

8位模式时序图：



4位模式时序图：



注意：IR7，IR3是操作指令的第7位和第3位；

AC3是地址计数器（AC）的第三位。

原理介绍：

HD44780U有两个8位的寄存器，分别是指令寄存器（IR）和数据寄存器（DR）。指令寄存器存放指令代码，如清屏、光标移位等命令、内部存储器的地址信息等。DR暂存写入或读出HD44780U内部存储器的数据。当MPU从HD44780U写入或读出数据时，对DR的由内部逻辑自动完成。

HD44780U内部还有一个地址计数器（AC），当地址信息被写入IR时，同时也被写入AC。MPU从HD44780U写入或读出数据后，AC的值自动加1。

HD44780U还有一个忙标志位（BF）来标志内部操作是否完成。当RS=0，R/Wn＝1时，忙标志位被写到DB7。下次操作必须保证BF为0后才写入。

HD44780U中有三个内部存储器：显示数据RAM（DDRAM），字形发生ROM（CGROM）和字形发生RAM（CGRAM）。

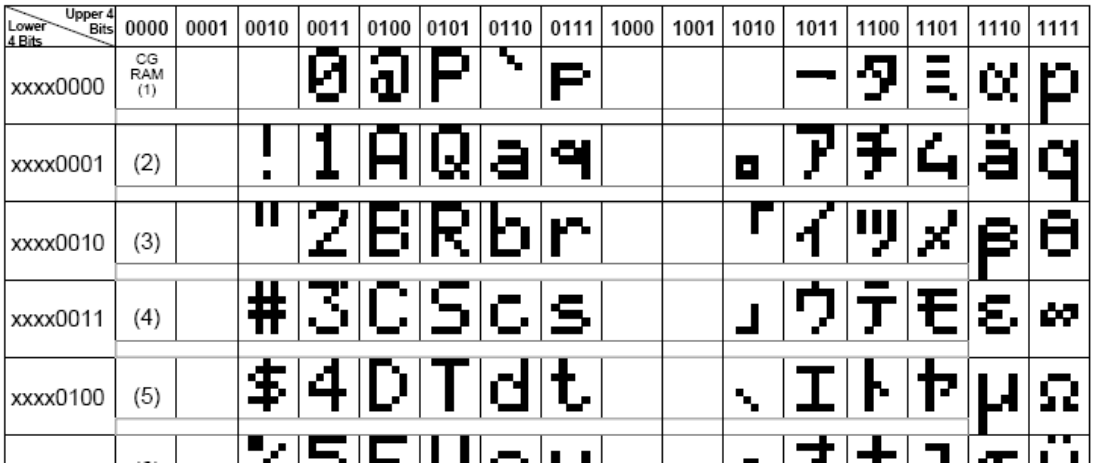
DDRAM保存8位字形代码，容量为80×8bit。DDRAM保存的内容与显示位置对应。拿2×16显示屏为例，DDRAM中内容与显示位置的对应图如下：

显示位置	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
DDRAM地址	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

CGROM根据8位字符码产生5×8或者5×10的点阵字体模型。可以产生208个5×8的点阵字体模型和32个5×10的点阵字体模型。用户也可以通过mask-programmed ROM自定义字体模式。

用户可以利用CGRAM编程重写自形模式。对于5×8点阵字形，可以重写8种字形；对于5×10点阵字形，可以重写4种。

用户写入显示数据时，是写8位字形代码到DDRAM中，这种代码与保存在CGROM或CGRAM中的字体模型有特定的对应关系，芯片内部操作将自动根据对应关系取出相应字体显示在液晶屏上。对应关系如下：



上图只列出了部分对应关系表，全表请查看器件手册。

用户通过向HD44780U写入指令来初始化、设置液晶或向其写入读出数据，指令说明如下：

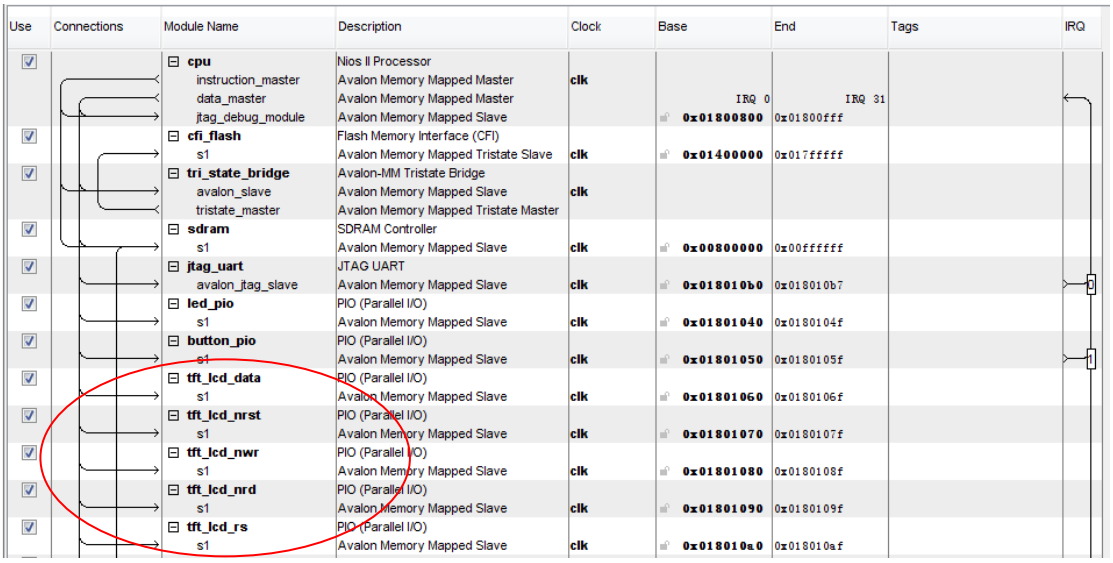
指令名称	控制信号	控制代码	功能描述
	RS R/W	DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0	
清屏	0 0	0 0 0 0 0 0 0 1	清屏，将AC中的DDRAM地址设为0
归home位	0 0	0 0 0 0 0 0 1 —	将AC中的DDRAM地址设为0；将显示还原到初始位置；DDRAM内容不变。
输入方式设置	0 0	0 0 0 0 0 1 I/D S	设置光标移动方向，设定是否移位显示。这些操作在读写数据时执行。
显示方式设置	0 0	0 0 0 0 1 D C B	打开/关闭显示(D)，设置光标打开/关闭/闪烁(C)以及光标显示位置（B）。
光标或显示移位	0 0	0 0 0 1 S/C R/L — —	移动光标或显示，DDRAM内容不变
功能设	0 0	0 0 1 DL N F — —	设置接口数据长度

置			（DL），显示行（N），字体（F）
CGRAM地址设置	0 0	0 1 ACG ACG ACG ACG ACG ACG	设置CGRAM地址，设置后CGRAM的数据送出或接收
DDRAM地址设置	0 0	1 ADD ADD ADD ADD ADD ADD ADD	设置DDRAM地址，设置后DDRAM的数据送出或接收
读BF和AC	0 1	BF AC AC AC AC AC AC AC	读忙标志位和地址计数器AC的值
写数据到CGRAM或DDRAM	1 0	写数据	写数据到DDRAM或者CGRAM中
读数据到CGRAM或DDRAM	1 1	读数据	从DDRAM或者CGRAM中读数据
I/D = 1: 增 I/D = 0: 减 S = 1: 伴随显示移位 S/C = 1: 显示移位 S/C = 0: 光标移动 R/L = 1: 向右移位 R/L = 0: 向左移位 DL = 1: 数据宽度8 位 DL = 0: 数据宽度4 位 N = 1: 2 行 N = 0: 1 行 F = 1: 5 10 点阵 F = 0: 5 8 点阵 BF = 1: 内部操作执行，忙 BF = 0: 非忙状态，可接收指令			缩写解释： DDRAM: Display data RAM CGRAM: Character generator RAM ACG: CGRAM address ADD: DDRAM address (对应光标位置) AC: Address counter 用于指示DDRAM或CGRAM的地址

限于篇幅，以上只是介绍了HD44780U的主要特性和功能，更加具体的介绍和参数说明请参见器件手册。

3.硬件设计：

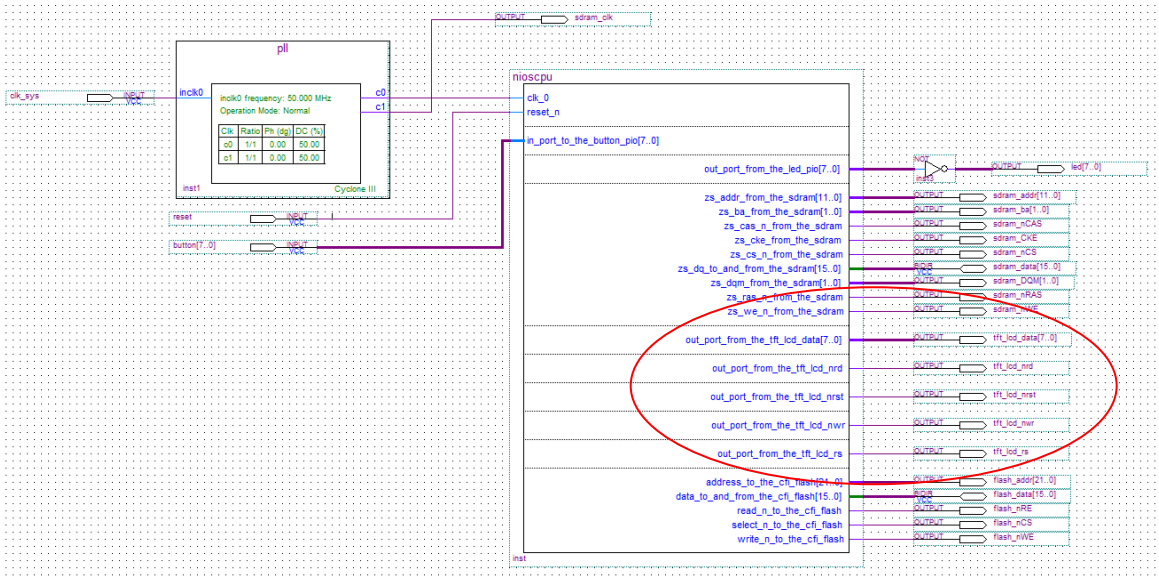
在 SOPC builder 中利用 PIO IO 核实现 LCD 的输入输出引脚，如图所示：



Use	Connections	Module Name	Description	Clock	Base	End	Tags	IRQ
☑		**cpu**	Nios II Processor					
		instruction_master	Avalon Memory Mapped Master	**clk**				
		data_master	Avalon Memory Mapped Master			IRQ 31		
		jtag_debug_module	Avalon Memory Mapped Slave		**0x01800800**	0x01800fff		
☑		**cfi_flash**	Flash Memory Interface (CFI)					
		s1	Avalon Memory Mapped Tristate Slave	**clk**	**0x01400000**	0x017fffff		
☑		**tri_state_bridge**	Avalon-MM Tristate Bridge					
		avalon_slave	Avalon Memory Mapped Slave	**clk**				
		tristate_master	Avalon Memory Mapped Tristate Master					
☑		**sdram**	SDRAM Controller					
		s1	Avalon Memory Mapped Slave	**clk**	**0x00800000**	0x00ffffff		
☑		**jtag_uart**	JTAG UART					
		avalon_jtag_slave	Avalon Memory Mapped Slave	**clk**	**0x018010b0**	0x018010b7		0
☑		**led_pio**	PIO (Parallel I/O)					
		s1	Avalon Memory Mapped Slave	**clk**	**0x01801040**	0x0180104f		
☑		**button_pio**	PIO (Parallel I/O)					
		s1	Avalon Memory Mapped Slave	**clk**	**0x01801050**	0x0180105f		1
☑		**tft_lcd_data**	PIO (Parallel I/O)					
		s1	Avalon Memory Mapped Slave	**clk**	**0x01801060**	0x0180106f		
☑		**tft_lcd_nrst**	PIO (Parallel I/O)					
		s1	Avalon Memory Mapped Slave	**clk**	**0x01801070**	0x0180107f		
☑		**tft_lcd_nwr**	PIO (Parallel I/O)					
		s1	Avalon Memory Mapped Slave	**clk**	**0x01801080**	0x0180108f		
☑		**tft_lcd_nrd**	PIO (Parallel I/O)					
		s1	Avalon Memory Mapped Slave	**clk**	**0x01801090**	0x0180109f		
☑		**tft_lcd_rs**	PIO (Parallel I/O)					
		s1	Avalon Memory Mapped Slave	**clk**	**0x018010a0**	0x018010af		

添加后，重新 generate，生成新的 niosopu 系统

更新顶层原理图如下：



引脚分配情况

下表为 B-ICE-EDA/SOPC-IEELS Platform 开发实验平台引脚分配表：

设计端口	EP3C55F484I7芯片引脚	开发板模块	备注
tft_lcd_data[7]	PIN_A3	字符液晶屏	字符液晶屏数据线
tft_lcd_data[6]	PIN_F7		
tft_lcd_data[5]	PIN_E6		
tft_lcd_data[4]	PIN_C7		
tft_lcd_data[3]	PIN_E5		
tft_lcd_data[2]	PIN_C3		
tft_lcd_data[1]	PIN_AB18		
tft_lcd_data[0]	PIN_AB17		
tft_lcd_nrd	A4	字符液晶屏	字符液晶屏控制线
tft_lcd_nwr	A5		
tft_lcd_rs	A6		
tft_lcd_nrst	R20		

4. 软件设计：

实验源程序如下：

```
#include <stdio.h>
#include "system.h"
#include "altera_avalon_pio_regs.h"
#include "alt_types.h"
#include <unistd.h>
#include <string.h>

unsigned char   seg[]={'1','2','3','4','5','6','7','8','9','0','a','b','c','d','e','f',
            'g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v'};



void Write_data(alt_u8 data)
{
    IOWR_ALTERA_AVALON_PIO_DATA(TFT_LCD_RS_BASE,1);
    IOWR_ALTERA_AVALON_PIO_DATA(TFT_LCD_NWR_BASE,0);
    IOWR_ALTERA_AVALON_PIO_DATA(TFT_LCD_DATA_BASE,data);     //地址
```

```
计数器自动加1，显示屏不移动
    IOWR_ALTERA_AVALON_PIO_DATA(TFT_LCD_NRD_BASE,1);
    IOWR_ALTERA_AVALON_PIO_DATA(TFT_LCD_NRD_BASE,0);
    usleep(1000);
}

void Write_cmd(alt_u8 cmd)
{
    IOWR_ALTERA_AVALON_PIO_DATA(TFT_LCD_RS_BASE,0);
    IOWR_ALTERA_AVALON_PIO_DATA(TFT_LCD_NWR_BASE,0);
    IOWR_ALTERA_AVALON_PIO_DATA(TFT_LCD_DATA_BASE,cmd);     //地
址计数器自动加1，显示屏不移动
    IOWR_ALTERA_AVALON_PIO_DATA(TFT_LCD_NRD_BASE,1);
    IOWR_ALTERA_AVALON_PIO_DATA(TFT_LCD_NRD_BASE,0);
    usleep(1000);
}

void initial(void)
{
    Write_cmd(0x3c);
    Write_cmd(0x06);
    Write_cmd(0x0e);
    Write_cmd(0x01);
}


void main()
{
    unsigned char data_index;
    unsigned char add_first,add_second;
    initial();              //初始化LCD
    data_index=0;add_first=0x80;add_second=0xC0;
    while(data_index<=15) //第一行显示
    {
        Write_cmd(add_first+data_index);
        Write_data(seg[data_index]);
        data_index++;
    }
```

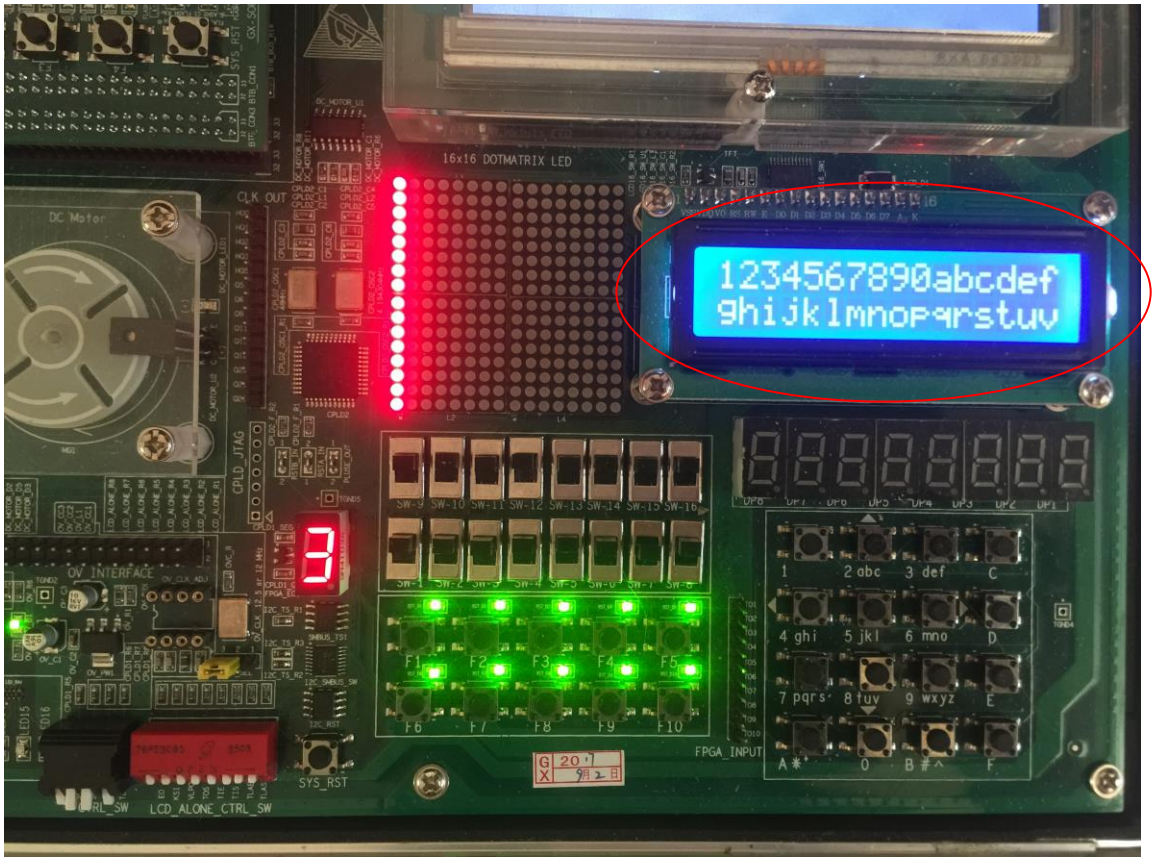
请同学们完成 //第二行显示

```
    return 0;
}
```

4. 实验操作和结果：

打开实验箱开关，连接好 JTAG 下载线，将开发平台上 CTRL_SW 组合开关 SEL1 拨置于下，SEL2 拨置于上，逻辑电平为 01，TLS 拨置于上，TLEN 拨置于下,使 DP9 数码管显示 3。LCD_ALONE_CTRL_SW 组合开关中 TLAE,TLAS 拨置于上,其它 6 个拨置于下。

实验结果如下：



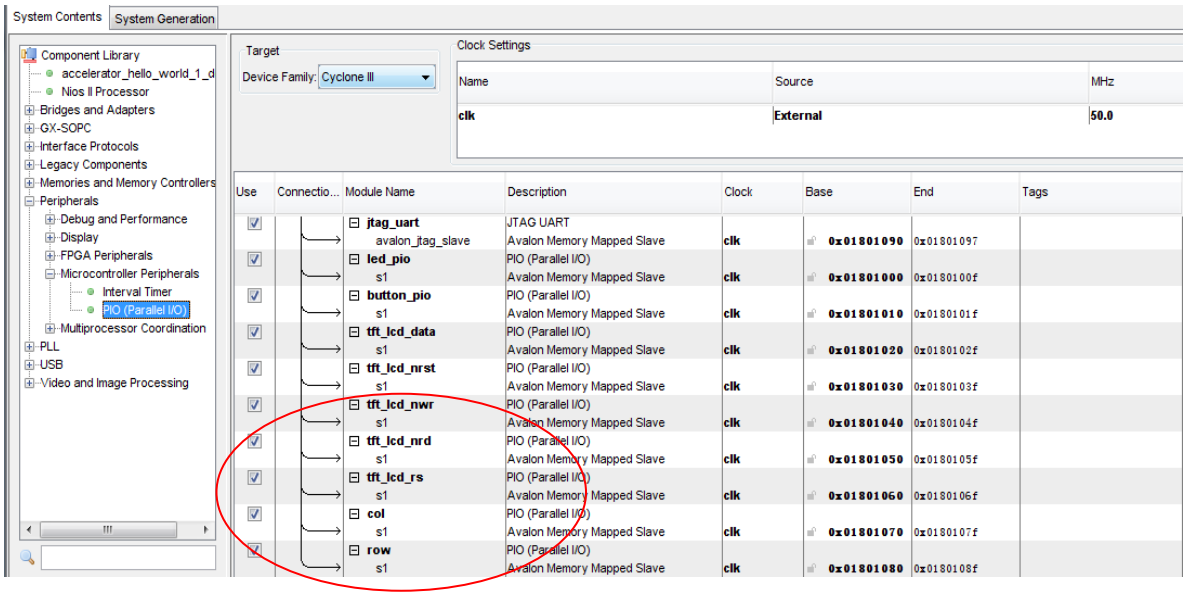
（三）16*16 点阵控制：

1. 实验内容：

通过按键 F1 选择显示两个不同的汉字。

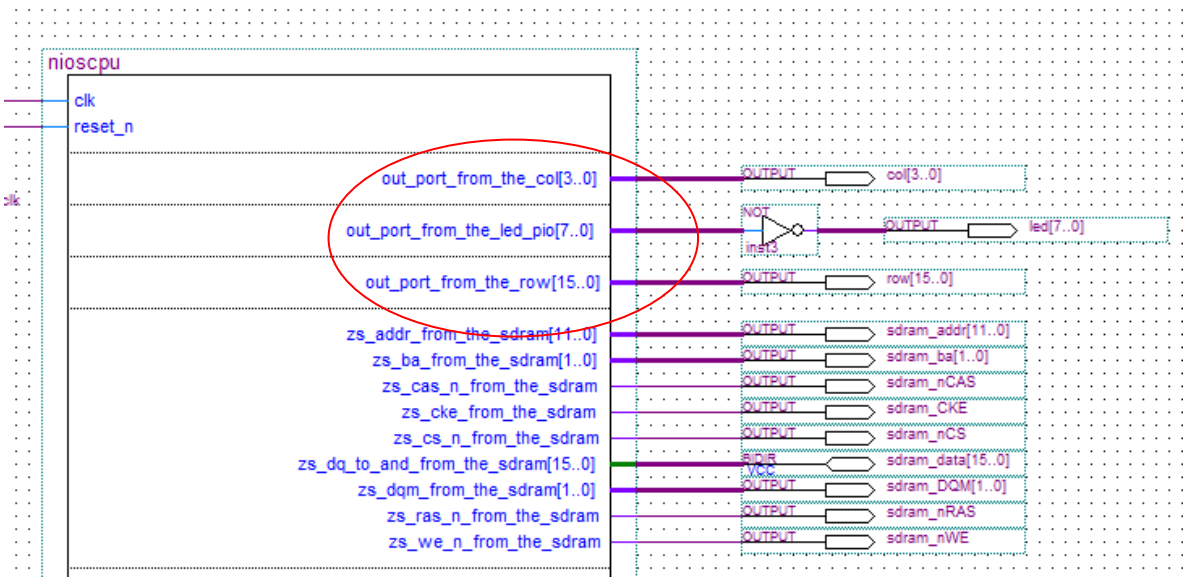
2. 硬件设计：

利用 PIO 核产生 16*16 点阵的列信号和行信号，如图所示：



添加后，重新 generate，生成新的 nioscpu 系统

更新顶层原理图如下：



引脚分配情况

下表为 B-ICE-EDA/SOPC-IEELS Platform 开发实验平台引脚分配表：

设计端口	EP3C55F484I7 芯片引脚	开发板模块	备注
clk	T1	sys_clk	系统时钟 50MHz
reset	N18	SW1	拨码开关： 上：“1”下：“0”
col[3]	C4	COL4	16*16 点阵的列信号
col[2]	A16	COL3	
col[1]	A15	COL2	
col[0]	A14	COL1	
row[15]	A4	ROW1	16*16 点阵的行信号
row[14]	A5	ROW2	
row[13]	A6	ROW3	
row[12]	B6	ROW4	
row[11]	E11	ROW5	
row[10]	C13	ROW6	
row[9]	F11	ROW7	
row[8]	C15	ROW8	
row[7]	E14	ROW9	
row[6]	B7	ROW10	
row[5]	B8	ROW11	
row[4]	B9	ROW12	
row[3]	B10	ROW13	
row[2]	D10	ROW14	
row[1]	F9	ROW15	
row[0]	A13	ROW16	

3.软件设计：

参考实验源程序如下：

```
#include "system.h"
#include "altera_avalon_pio_regs.h"
#include "alt_types.h"

alt_u16 row_num[2][16]={
                        {0x1806,0x1806,
                                                                                }
,//革

                        {0x0000,0x3366,
                                                                                }
//新
};


int main()
{
   alt_u8 col_num=0,read_data;
   while (1)
   {
      //将按键值读入到 read_data  中；
      for(col_num=0;col_num<16;col_num++)
      {
          IOWR_ALTERA_AVALON_PIO_DATA(ROW_BASE,0);
          //将 col_num 的值写入到列地址；
           //将 row_num 的值写入到行地址；
          usleep(1);
      }
      }
    return 0;
}
```